Osborne / McGraw-Hill

THE PROGRAMMER'S

LANDBOOI

Andy Johnson-Laird

.

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 .

THE PROGRAMMER'S CP/M[®] HANDBOOK

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

ix

45

Outline of Contents Notation Example Programs on Diskette

Introduction

This book is a sequel to the Osborne CP/M^{\otimes} 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.

- 4 The CP/M Programmer's Handbook
 - ^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.

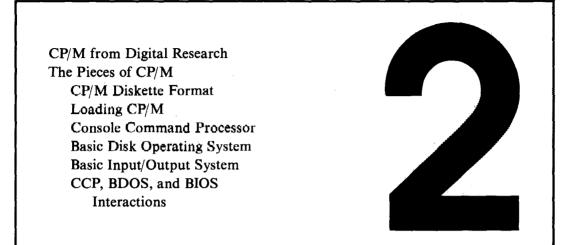
Example Programs on Diskette

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.



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

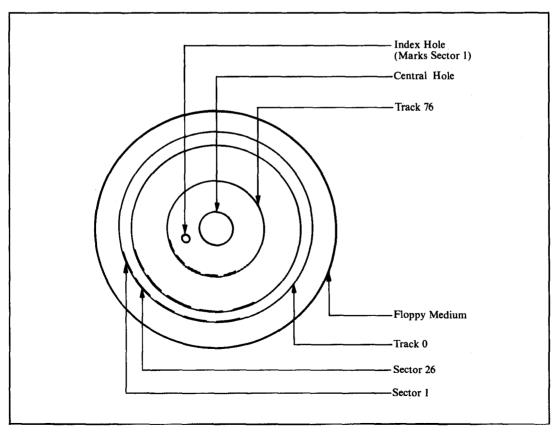


Figure 2-1. Floppy disk layout

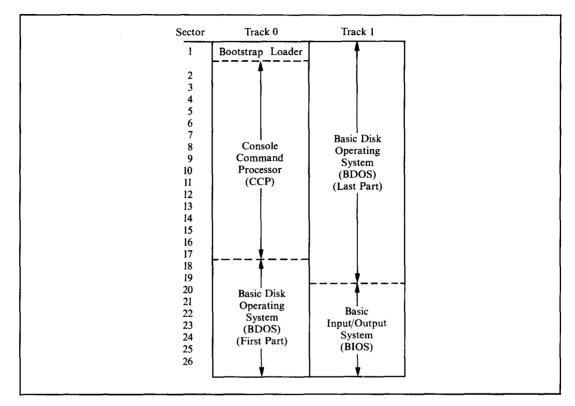


Figure 2-2. Layout of CP/M on tracks 0 and 1 of floppy disk

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.

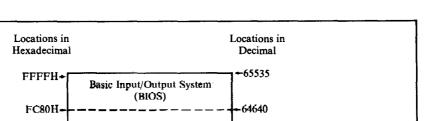
At this point, it is worthwhile to review which CP/M parts are in memory, where in memory they are, and what functions they perform.

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.



FFFFH+ FC80H= FC80H= Basic Disk Operating System (BDOS) E680H= Console Command Processor (CCP) DE80H= Memory Available for Programs 0100H= CP/M Reserved Area +0

1. X. S.

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.

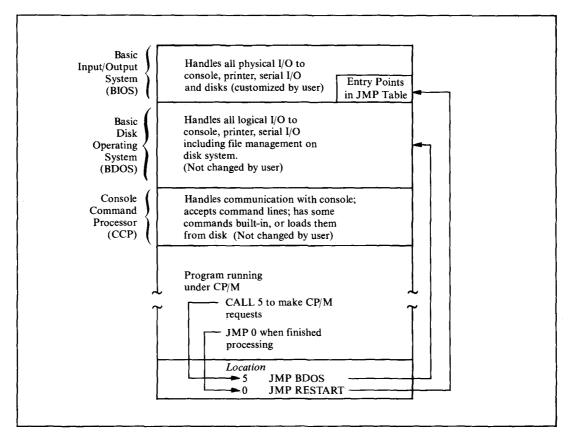
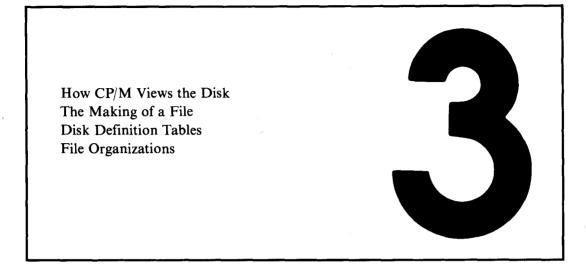


Figure 2-4. CP/M's functional breakdown



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.

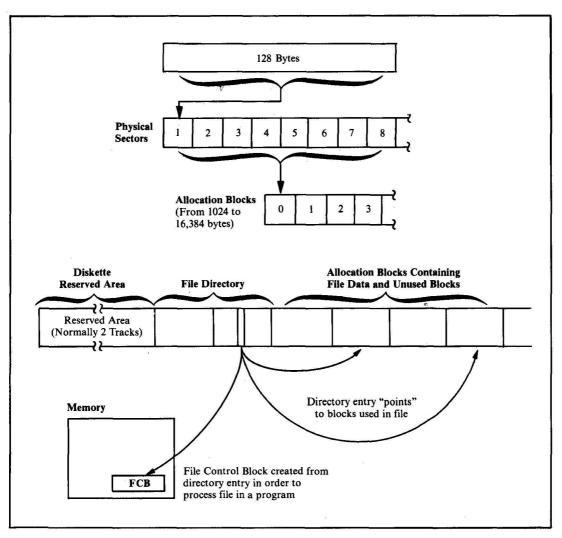


Figure 3-1. The hierarchical relationship between sectors, allocation blocks, directory entires, and FCBs

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 =	FDESUSER	EQU	0	File user number (LS 4 bits)
0001 =	FDE\$NAME	EQU	1	#file name (8 bytes)
0009 =	FDE\$TYP	EQU	9	File type
				Offsets for bits used in type
0009 =	FDE\$R0	EQU	9	Bit 7 = 1 - Read only
000A =	FDE\$SYS	EQU	10	Bit 7 = 1 - System status
000B =	FDESCHANGE	EQU	īī	Bit 7 = 0 = File Written To
				,
000C =	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
	:			Juniocation Dioteks and
0000	FD\$USER:	DS		:File user number
0001	FD\$NAME:	DS	8	File name
0009	FD\$TYP:	DS	š	File type
0000	FD\$EXTENT:	DS	ĭ	Extent
000D	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
0010	r DendQ3ED1	50	10	ANTIOCATION DIOCKS UNED

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

24 The CP/M Programmer's Handbook

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 this 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 2 Alteration 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
					<pre>; sector conversion table</pre>
0002	0000		DW	0	Scratch pad areas used by CP/M
0004	0000		DW	0	
0006	0000		DW	ò	
0008	2A00		DW	DIRBUF	Pointer to Directory Buffer
					I work area
000A	AA00		DW	DPBO	Pointer to disk parameter bloc
	B900		DW	WACD	Pointer to work area (used to
					: check for changed diskettes)
000E	C900		DW	ALVECO	Pointer to allocation vector
		;	<i>—</i> ···		
		;	The fo	ollowing equates	would normally be derived from
		i	value	s found in the d	isk parameter Block.
			They a	are shown here of	nly for the sake of completeness.
003F	±	NODE	EQU	63	Number of directory entries 1
00F2	=	NOAB	EQU	242	Number of allocation blocks
		3			
			Exampl	le data definitio	ons for those objects pointed
				the disk parame	
				• • • • •	
		SKEWTAB	LE:		Sector skew table.
					: Indexed by logical sector
0010	01070D13		DB	01,07,13,19	:Logical sectors 0,1,2,3
	19050B11		DB	25.05.11.17	14,5,6,7
0018	1703090F		DB	23,03,09,15	18, 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
	1016		DB	16.22	: 24. 25
0020	1010		00	10,22	,24,25
002A		DIRBUF:	DS	128	Directory buffer
0044		DPB0:	DS	15	;Disk parameter block
					;This is normally a table of
					; constants.
					A dummy definition is shown
					i here
0089		WACD:	DS	(NODE+1)/4	Work area to check directory
					:Only used for removable media
0009		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						22
5		5				
6					18	
7	1	1				1
8				14		
9		[10			
10						23
11		6			10	
12		{	}		19	ļ
13	2					
14				15		
15 16		1	11			<u>م</u>
16		7				24
18	ł	1 '	l	l l	20	ł
19	3	1			20	1
20	,			16		
21		}	12			ſ
22		ſ				25
23		8				
24	1	1		1	21	
25	4					
26		l		17		l
]	Ì			

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

SKEWTABLE: ;Logical sector 0000 01070013 DB 01,07,13,19 ;0,1,2,3 0004 19050B11 DB 25,05,11,17 14,5,6,7 0008 1703090F 000C 1502080E ;8,9,10,11 ;12,13,14,15 ;16,17,18,19 DB 23,03,09,15 21,02,08,14 **DB** 0010 141A060C DB 20,26,06,12 0014 1218040A DB 18,24,04,10 120,21,22,23 0018 1016 DB 16,22 :24.25 The code to translate logical sectors to physical . sectors is as follows: 7 5 On entry, the logical sector will be transferred from CP/M as a 16-bit value in registers BC. CP/M also transfers the address of the skew table in registers DE (it finds the skew table by looking in the disk parameter header entry). . On return, the physical sector will be placed ; in registers HL. SECTRAN: 001A EB XCHG ;HL ~> skew table base address ;HL -> physical sector
; entry in skew table в 001B 09 DAD ;L = physical sector :HL = Physical Sector 001C 6E MOU L.M 001D 60 MOV H. 0 001E C9 RET Return to BDOS

Figure 3-5. Data declarations for the standard skewtable for standard diskettes

32 The CP/M Programmer's Handbook

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 Irack 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.

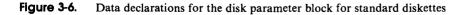
Hard disks normally have sector sizes larger than 128 bytes. This is discussed in the section on considerations for hard disks.

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.

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



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.

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

Table 3-1. Block Shift and Mask Value

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

1 to 255	256 and Above
0	(Impossible)
1	0
3	1
7	3
15	7
	0 1 3 7

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	Tracks/Diskette Sectors/Track	77 - 2	Tracks/Diskette Tracks Reserved for CP/M
128 2 1024	Bytes/Sector Tracks Reserved for CP/M Bytes/Allocation Block	75 ×26 1950 ×128	Tracks for File Storage Number of Sectors Sectors for File Storage Bytes per Sector
		249,600 ÷1024	Bytes for File Storage Bytes/Allocation Block
		243.75	Total Number of Allocation Blocks
		242	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.

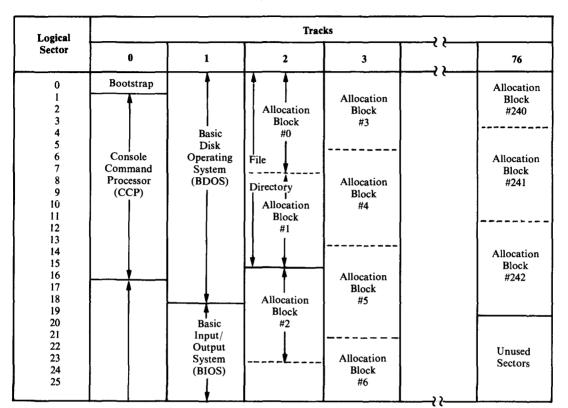
Special Considerations for Hard Disks

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.



Sec. Constants

Chapter 3: The CP/M File System 37

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.

38 The CP/M Programmer's Handbook

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

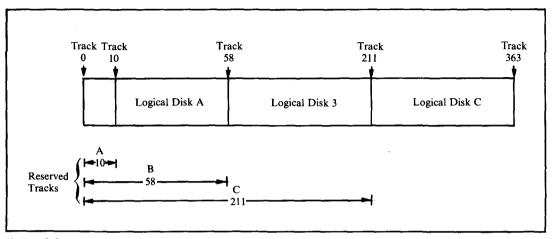


Figure 3-9. Dividing hard disks into logical disks

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	istics:		Calculate:		
364 20	Tracks/Di Sectors/T	rack	A :	B: and C:	
512 10,240	Bytes/Sector Bytes/Track		48 ×10,240	153 ×10,240	0
			491,520 ÷ 2048	1,566,720 ÷ 4096	Bytes/Disk Bytes/Allocation Block
Chosen Logical Characteristics:			240	382.5	Number of Allocation Blocks
	Tracks	Allocation Block Size	239	381	Maximum Block Number
Reserved Area	10	n/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:	DPBI	3: DPI	BC:	
	4	4	4	<pre>#128-byte sectors/"track"</pre>
	4	5	5	Block shift
	15	31	31	Block mask
	1	1	1	Extent mask
	239	381	381	Max. all. block #
	255	1023	1023	No. of directory entries
	11110000B	11111111B	11111111B	Bit Map for allocation blocks
	0000000B	0000000B	0000000B	; used for directory
	0	0	0	;No. of bytes in dir.check buffer
	(10)	(58)	(211)	Actual tracks before directory
	200	1160	4220	<pre>#"Tracks" before directory</pre>

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

40

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 (0 = default. 1=A)
0001 =	FCBE\$NAME	EQU	i	;File name (8 bytes)
0009 =	FCBE\$TYP	EQU	9	File type
	1 OBE + 1 //	2		Offsets for bits used in type
0009 =	FCBE\$RO	EQU	9	Bit 7 = 1 - read only
000A =	FCBE\$SYS	EQU	10	Bit 7 = 1 - system status
000B =	FCBE\$CHANGE	EQU	11	:Bit 7 = 0 - file written to
= 3000	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 =	FCBE\$RANRECO	EQU	35	Random rec. overflow byte (MS)
	1			,
	;			
0000 00	FCB#DISK:	DB	0	Search on default disk drive
0001 46494	C454EFCB\$NAME:	DB	'FILÉ	NAME' ;File name
0009 5459	50 FCB\$TYP:	DB	1TYP	;File type
00 3000	FCB\$EXTENT:	DB	0	;Extent
0000 0000	FCB\$RESV:	DB	0,0	<pre>#Reserved for CP/M</pre>
000F 00	FCB\$RECUSED:	DB	0	Records used in this extent
0010 00000	00000FCB#ABUSED:	DB	0,0,¢	0,0,0,0,0,0 ;Allocation blocks used
0018 00000	000000	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)

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.

48 The CP/M Programmer's Handbook

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: NONSYS
 SYSFILE does not show

 B>DIR
 JUNK

 NO
 FILE

 B>DIR
 SYSFILE

 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>ERA *.C??<cr> 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><u>ren</u> <u>chap*.doc=chapter*.doc<cr></u>
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</u> n FILENAME.TYP<cr> 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.

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 =	RAM	EQU	0	;Start of RAM (and the base page) ;You may need to change this to ; some other value (e.g. 4300H)
0000 0000) WARMBOOT:	org Ds	RAM 3	;Set location counter to RAM base ;Contains a JMP to warm boot entry ; in BIOS Jump vector table
0002 =	; BIOSPAGE	EQU	RAM+2	;BIOS Jump vector page
0003	ÍOBYTE:	DS	1	;Input/output redirection byte
0004 0004 =	CURUSER: CURDISK	ds Equ	1 CURUSER	;Current user (bits 7-4) ;Default logical disk (bits 3-0)
0005 0007 =	BDOSE: TOPRAM	DS EQU	3 BDOSE+2	;Contains a JMP to BDOS entry ;Top page of usable RAM
0005C	,	ORG	RAM+5CH	Bypass unused locations
005C	FCB1:	DS	16	<pre>;File control block #1 ;Note: if you use this FCB here ; you will overwrite FCB2 below.</pre>
0060	FCB21	DS	16	<pre>\$File control block #2 \$You must move this to another \$ place before using it</pre>
0080	3	ORG	RAM+80H	;Bypass unused locations
0080	COMTAIL: COMTAIL+COUNT:	DS	1	;Complete command tail ;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+80H	Redefine command tail area
0080	DMABUFFER:	DS	128	;Default "DMA" address used ; as a 128-byte record buffer
0100	; TPA:	ORG	RAM+100	I ;Bypass unused locations ;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 56 The CP/M Programmer's Handbook

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 so 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

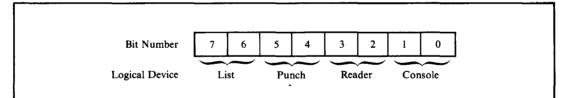


Figure 4-2. Arrangement of the IOBYTE

Tested Darie		Physical	Device	
Logical 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.
- 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

60 The CP/M Programmer's Handbook

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.

TPA

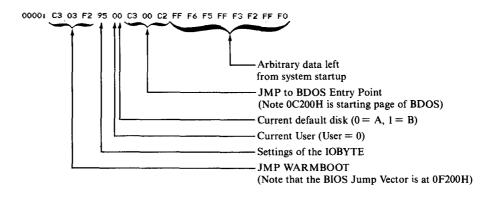
This is the *transient program area* into which the CCP loads programs. The TPA extends up to the base of the BDOS.

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:



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:

Logical Disk								Logical Disk								
005C:	- <u></u>	_54	4F	46 F									ļ			
0060:	49 I	4C L	~		20	54 T	59 Y	50 P	•	•••	00	00'	00	46 F	52 R	4F 0
0070:	410 M	46 F	49 I	4C L	45 E	54 T	59 Y	50 P	00 •	•	•••	00 •	•	F2	34 4	F3 •

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 01 FH (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).

0100 0100 0103	CD3601	START:	ORG CALL NOP	100H CTP	;Test b	ed for CTP
0103	00			inder of	your pre	ogram
		;	This s	ubroutine	e breaks	the command tail apart, placing
		:	each v	alue in a	a separa	le string area.
		;	Return	paramete		
		;				(Z flag set)
		1				ber of parameters
		!		HL -> 18		uddresses Iress points to a null-byte-
						ed parameter string.
		-		If too r		meters are specified, then $A = TMP$
						meter is too long, then A = PTL
		;				ints to the first character of the
		;			offendi	ng parameter in the COMTAIL area.
0080	-	COMTAIL		EQU	80H	Command tail in base page
0080	-	COMTAIL	\$COUNT	EQU	CONTAIL	Count of chars. in command tail
0001		CTP\$TMP		EQU	1	;Too many parameters error code
0002	=	CTP\$PTL		EQU	2	Parameter too long error code
		TABLE:			;Table d	f pointers to parameters
0104	0001		DW	P1	; Param	
	1A01		DW	P2	; Parame	iter 2
0108	2801		D₩	P3	; Parame	ter 3
						parameter addresses here
010A	0000	,	DW	0	; Termin	ator
		;	Parame	ter striv	ngs.	
		;	The fi	rst byte	is 0 so	that unused parameters appear
		;		null stri		
		3	The la	st byte o	of each :	is a 0 and is used to detect
		;		meter tha		
	0001010101					1,1,1,1,1,0 ;Param. 1 & terminator
	0001010101					,1,1,1,1,0 ;Param. 2 & terminator
0128	0001010101	P3:				,1,1,1,1,0 ;Param. 3 & terminator
		,		; <-~- AC	id more p	arameter strings here
		CTP:			1	Main entry point <<<<<
	210401		LXI	H, PTABLE	E j	HL -> table of addresses
	0E00		MVI	с,о		Set parameter count
	348000		LDA	CONTAIL		Character count
013E			ORA	A		Check if any params.
013F			RZ	`		Exit (return params, already set)
0140			PUSH MOV	H B.A		Save on top of stack for later B = COMTAIL char. count
	218100		LXI	H, COMTA		B = COMIAIL char, count HL -> Command tail chars.

Figure 4-3. Command Tail Processor (CTP)

	CTP\$NEXTP:		Next parameter loop
0145 E3	XTHL		;HL -> Table of addresses
			;Top of stack = COMTAIL ptr.
0146 5E	MOV	E,M	;Get LS byte of param. addr.
0147 23	INX	н	;Update address pointer
0148 56	MOV	D, M	;Get MS byte of param. addr.
			;DE -> Parameter string (or is 0)
0149 7A	MOV	A,D	;Get copy of MS byte of addr.
014A B3	ORA	E	;Combine MS and LS byte
014B CA8001	JZ	CTP\$TMPX	;Too many parametersexit
014E 23	INX	н	;Update pointer to next address
014F E3	XTHL		;HL -> comtail
1			;Top of stackupdate addr. ptr.
			int, we have
1			t byte in command tail
4		; DE -> fir	st byte of next parameter string
	CTP\$SKIPB:		.
0150 7E	MOV	A, M	;Get next parameter byte
0151 23	INX	H	;Update command tail ptr.
0152 05	DCR	B	Check if characters still remain
0153 FA7301	MB	CTPX	;No, so exit
0156 FE20	CPI		;Check if blank
0158 CA5001	JZ	CTP\$SKIPB	;Yes, so skip blanks
015B 0C	INR	C	;Increment parameter counter
}	CTP\$NEXTC:	-	.
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	\$Check if terminator
0160 CA7A01	JZ	CTP\$PTLX	Parameter too long exit
0163 AF	XRA	<u>A</u>	Float a 00-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
016B FE20	CPI	OTOOLEVITE	Check if parameter terminator
016D CA4501	JZ	CTP\$NEXTP	;Yes, so move to next parameter
0170 C35C01	JMP	CTP\$NEXTC	;No, so store it in param. string
1	CTPX:		a Navan 3 avri 4
0170 15		•	;Normal exit
0173 AF	XRA	Α	;A = 0 & Z-flag set
{	; Стрсх		Advance with such
0174 51	POP	ы	;Common exit code
0174 E1 0175 210401	LXI	H H, PTABLE	Balance stack
0175 210401	ORA		Return ptr. to param. addr. table
01/8 8/	RET	A	;Ensure Z-flag set appropriately
01/7 (9			
	; CTP\$PTLX:		Parameter too long exit
017A 3E02	MVI	A, CTP\$PTL	;Set error code
017A 3E02 017C EB	XCHG	H, CIEWEIL	;Set error code ;DE -> offending parameter
0170 C37401	JMP	CTPCX	;DE -> offending parameter ;Common exit
0170 (37401	JMP	UIFUA	, Common EXIL
ļ	; CTP\$TMPX:		;Too many parameters exit
0180 3E01	MVI	A.CTP\$TMP	; loo many parameters exit :Set error code
0180 3501	UMP	CTPCX	;Set error code ;Common exit
0102 03/401		CIFUX	;common exit
0185	; END	START	
0100	CNU	JINNI	

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	7	<pre>Most significant byte of</pre>
		7			BDOS entry point
0000	3A0700		LDA	TOPRAM	Get MS byte of BDOS entry point
0003	3D		DCR	A	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 4-4. Setting stack pointer to top of available RAM

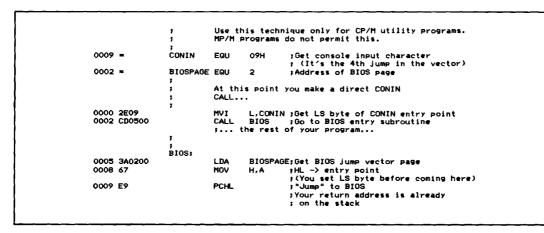


Figure 4-5. Making a direct BIOS call

ş Note: This example assumes you have not overwritten the CCP. ; ; 0100 OBG 100H ;Start at TPA START: 0100 210000 LXT н, о ;Save CCP's stack pointer ;By adding it to 0 in HL 0103 39 0104 220F01 DAD SP CCP\$STACK SHLD 0107 314101 LXI SP, LOCAL\$STACK ; The main body of your program is here 7 ; and when you are ready to return to the CCP... ; ;Get CCP's stack pointer 010A 2A0F01 LHLD CCP\$STACK SPHL 010D F9 :Restore SP 010E C9 RET Return to the CCP CCP\$STACK: 010F DS 2 ;Save area for CCP SP 0111 DS 48 ;Local stack LOCAL\$STACK: 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.

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	
13	Reset disk system	
14	Select specified logical disk drive	
15	Open specified file for reading/writing	
16	Close specified file after reading/writing	
17	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	
21	Write the next "record" sequentially	
22	Create a new file with the specified name	
23	Rename a file to a new name	
24	Indicate which logical disks are active	
25	Return the current default disk drive number	
26	Set the DMA address (read/write address)	
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	
31	Return address of disk parameter block (DPB)	
32*	Set/Get the current user number	
33	Read a "record" randomly	
34	Write a "record" randomly	
35	Return logical file size (even for random files)	
36	Set record number for the next random read/write	
37	Reset specified drive	
40	Write a "record" randomly with zero fill	*These do not work under MP/M.

0000 =	B\$SYSRESET	COL	~	
0001 =	B\$CONIN	EQU EQU	0	;System Reset ;Read Console Byte
0002 =	B\$CONDUT	EQU	2	;Read Console Byte ;Write Console Byte
0002 =	BREADIN	EQU	3	;write Console Byte ;Read "Reader" Byte
0004 =	B\$PUNOUT	EQU	4	;Kead "Reader" Byte ;Write "Punch" Byte
0005 =	B\$LISTOUT	EQU	5	Write Printer Byte
0006 =	B\$DIRCONIO	EQU	6	;Write Frinter Byte ;Direct Console I/O
0007 =	B\$GETIO	EQU	7	:Get IOBYTE
0008 =	B\$SETIO	EQU	8	Set IOBYTE
0009 =	B\$PRINTS	EQU	9	
0009 =	B#READCONS	EQU		;Print Console String ;Read Console String
000B =	B\$CONST	EQU	10 11	
= 2000	BISETVER	EQU	12	;Read Console Status
0000 =	BOSKRESET	EQU	13	;Get CP/M Version Number
000E =	B\$SELDSK	EQU		;Disk System Reset
000F =	B\$OPEN	EQU	14	;Select Disk
0010 =	BOPEN	EQU	15 16	;Open File
0010 =	BISEARCHE		17	;Close File
0012 =	B\$SEARCHN B\$SEARCHN	EQU		Search for First Name Match
0012 =	B\$SEARCHN B\$ERASE	EQU	18 19	Search for Next Name Match
0013 =				;Erase (delete) File
0014 = 0015 =	B\$READSEQ	EQU	20	Read Sequential
0015 = 0016 =	B\$WRITESEQ	EQU	21	;Write Sequential
	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 Bisk
001A =	B\$SETDMA	EQU	26	;Set DMA (Read/Write) Address
001B =	B\$GETALVEC	EQU	27	;Get Allocation Vector Address
001C =	B\$SETDSKRO	EQU	28	;Set Bisk to Read Only
001D =	BIGETRODSKS	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	;C = function code
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	C, FUNCTION\$CODE	;C = function code
LXI	D, ADDRESS	;DE = address
CALL	BDOS	;Location 5
		;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 = 00H Entry Parameters: None Exit Parameters: Does not return

Example

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

- 72 The CP/M Programmer's Handbook
 - **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\$CONÍN	;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
0000 0E02 0002 1E2A 0004 CD0300	MVI MVI CALL	C,B\$CONOUT E,'*' BDOS	;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 ;Output		yte-term	inated	message.
		;Callin	a seque	nce		
		;	MESSAG		DB	'Message',0
		;		:		
		7	LXI CALL	H, MESS MSGOUT	AGE	
		;Exit P				
		;	HL ->	Null byt	e termi	nator
0002	=	B\$CONOU	г	EQU	2	;Write Console Byte
0005	-	BDOS		EQU	5	;BDOS entry point
		MSGOUT:				
0000	7E		MOV	A, M		;Get mext 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	0E02		MVI	C, B\$CO	NOUT	
0008	CD0500		CALL	BDOS		
000B			POP	н		;Recover message pointer
0000	C30000		JMP	MSGOUT		: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
                                       MSGOUTI
                   ;
                             CALL
                             DB
                                       'Message',0
                   ;
                             ... next instruction
                   ;
                   ;Exit Parameters
                             HL -> instruction following message
                   ;
0002 =
0005 =
                   B$CONOUT
                                       EQU
                                                 2
                                                           ;Write Console Byte
                   BDOS
                                       EQU
                                                 5
                                                           ;BDOS entry point
                   MSGOUTI:
                                                           ;HL -> message
;Get next data byte
0000 E1
                             POP
                                       н
0001 7E
0002 23
                                       А, М
Н
                             MOV
                             INX
                                                           ;Update message pointer
;Check if null byte
0003 B7
                             ORA
                                       A
0004 C20800
0007 E9
                             JNZ
                                       MSGOUTIC
                                                           ;No, continue
                             PCHL
                                                           ;Yes, return to next instruction
                                                           ; after in-line message
                 MSGOUTIC:
                             PUSH
                                                           ;Save message pointer
;Ready for BDOS<sup>-</sup>
;Function code
0008 E5
                                       н
                                       E,A
C,B$CONOUT
0009 5F
                             MOV
000A 0E02
                             MVI
000C CD0500
                             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 = 03HEntry 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.

		turns are ied number	ignored of char	, and input terminates acters have been read
	;timeout in	this subro	utine. 3	that there is no It will wait forever at the reader device.
	;Calling seq ; LXI ; LXI ; LXI ; CALL	H, BUFF B, MAXC	OUNT	
	; BC =	> OOH byte	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 SF 0003 CA2000 0006 C5 0007 E5	RL\$RDR: MOV ORA MOV JZ PUSH PUSH		x	;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 000E FE0D 0010 CA0300 0013 E1 0014 C1 0015 FE0A 0017 CA2000 001A 77 001B 23 001C 0B 001D C30000	RL \$RDR I: MVI CALL MOV CPI JZ POP POP CPI JZ MOV INX DCX JMP	C,B\$RE BDOS E,A CR RL\$RDR H B LF RL\$RDR M,A H B RL\$RDR	ı x	<pre>;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.</pre>
	RL\$RDRX: MVI	M , 0		;Null-byte-terminate buffe

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	Ċ,B\$PUNOUT E,'X' 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	MVI	C,B\$LSTOUT	;Function code
0002 1E2A	MVI	E,'*'	;Data byte to output
0004 CD0500	CALL	BDOS	

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.

	;when a OOH ;A carriage ;encountere	e return is		ed. when a line feed is
	;Calling se			
	; LXI			
	; CAL	L WL\$PU	N	
	;Exit param ; HL	neters -> 00H byt	e termina	ator
0004 =	BSPUNOUT	EQU	4	
0005 =	BDOS	EQU	5	
000D =	CR	EQU	ODH	;Carriage return
000A =	LF	EQU	OAH	;Line feed
	WL\$PUN:			
0000 E5	PUS	ан н		;Save buffer pointer
0001 7E	MO\	/ A,M		;Get next character
0002 B7	ORA	A A		;Check if OOH
0003 CA2000	JZ	WL\$PU	NX	;Yes, exit
0006 FE0A	CP1	L LF		Check if line feed
0008 CC1600	CZ		NLF	;Yes, O/P CR
000B 5F	MON			Character to be output
DOOC DE04	MVI		UNOLIT	Function code
000E CD0500	CAL			;Output character
0011 E1	POF			Recover buffer pointer
0012 23	IN			;Increment to next char.
0013 C30000	JMF		N	;Output next char
	WLSPUNLF:			Line feed encountered
0016 0E04	MVI	C.B\$P	UNDUT	Function code
0018 1EOD	MV1			;Output a CR
001A CD0500	CAL			reachas a on
001D 3E0A	MVI			Recreate line feed
0015 SECH	RET			;Output LF
	WL\$PUNX:			:Exit
0020 E1	POF	• н		Balance the stack
0021 C9	REI			Paranee the stark

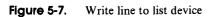
Figure 5-6. Write line to punch device

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.

	when a OOH b	utput terminates ed, when a line feed is		
	;Calling sequ ; LXI ; CALL	ence H,BUFI WL\$LS		
	;Exit paramet ; HL ->	ers OOH byta	e termin	ator
0005 =	B\$LSTOUT	EQU	5	
0005 =	BDOS	EQU	5	
000D =	CR	EQU	орн	;Carriage return
000A =	LF	EQU	OAH	Line feed
0000 E5	WL\$LST: PUSH	н		Save buffer pointer
0001 7E	MOV	A, M		:Get next character
0002 87	ORA	A		Check if OOH
0003 CA2000	JZ	WL\$LS	тх	Yes. exit
0006 FE0A	ČPI	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\$L	STOUT	Function code
000E CD0500	CALL	BDOS		;Output character
0011 E1	POP	н		Recover buffer pointer
0012 23	INX	н		Update to next char.
0013 C30000	JMP	WL\$LS	г	;Output next char.
	WL\$LSTLF:			:Line feed encountered
0016 0E05	MVI	C,B\$LS	STOUT	Function code
0018 1EOD	MVI	E, CR	-	;Output a CR
001A CD0500	CALL	BDOS		
OO1D SEOA	MVI	A, LF		Recreate line feed
001F C9	RET			;Output LF
	WL\$LSTX:			FExit
0020 E1	POP	н		Balance the stack
0021 C9	RET			,



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\$DIRCONIC E, OFFH BDOS) ;Function code ;OFFH means input ;A = 00 if no char. waiting ;A = NZ if character input

;Example of console output

0009 1E2A MVI E, ** * Not OFFH means output of 0008 CD0500 CALL BD0S	0009	0E06 1E2A CD0500	MVI MVI CALL	C,B\$BIRCONIO E,'*' BDOS	;Function ;Not OFFH		output	char
----------------------------------------------------------------------	------	------------------------	--------------------	--------------------------------	------------------------	--	--------	------

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:

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

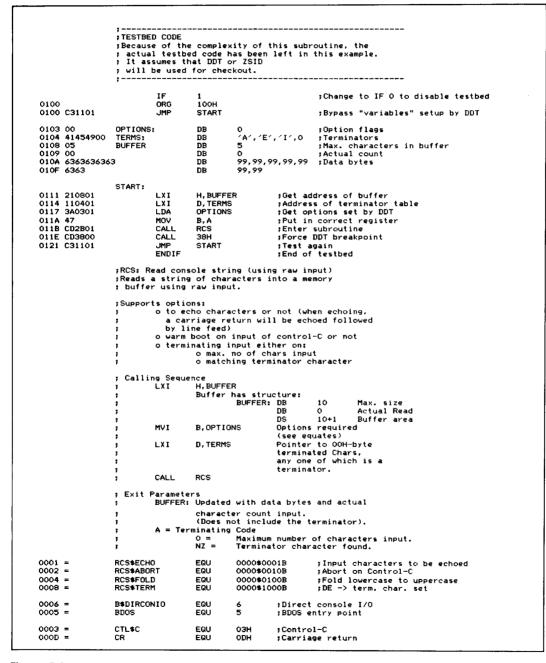


Figure 5-8. Read/write string from/to console using raw I/O

000A = 0008 = 0124 0D 0125 0A 0126 00 0127 08200 0128 23 0126 28 012F E5 0130 CD920 0134 3E08 0134 3E08 0135 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0143 47 0144 0E00 0144 0E00 0145 CD7F0 0145 CD7F0 0148 E9 0146 CD7F0 0148 C9 0148 C9 0148 C9 0148 C9 0146 CD7F0 0155 CD8B0 0152 CD7F0 0155 CD8B0 0158 C22F0 0158 C3440 0150 E5 0161 23 0162 S5 0163 FA7A0	RCS: INX MVI DCX RCS\$L: PUSH CALL POP MVI ANA JNZ LXI RCS\$UST: 1 CALL	EQU OAH EQU OBH ODH OAH O BS, ' ', BS, O H M, O H RCS#GC H A, RCS#TERM B RCS#UST D, RCS#ST RCS#CT	<pre>;Line feed ;Backspace ;Internal standard terminator table ;Carriage return ;Line feed ;End of table ;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators ;Standard terminators</pre>
0008 = 0124 0D 0125 0A 0126 00 0127 08200 0128 23 0122 3600 0128 28 0127 E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0138 11240 0130 CD40 0130 CD40 0130 CA4C0 0130 CA4C0 0143 47 0144 0E00 0144 0E00 0143 47 0144 0E00 0144 CA770 0145 CD7F0 0148 C9 0146 CD7 0148 C9 0147 CA600 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0158 6600 0158 6600 0150 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	BS RCS\$ST: DB DB DB DB DB DB DB DB DB DB	EQU 08H ODH OAH BS,1 1,BS,0 H M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;Backspace ;Internal standard terminator table ;Carriage return ;Line feed ;End of table ;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Get character and execute; ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
0124 0D 0125 0A 0126 00 0127 08200 0128 23 0122 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0130 CDD40 0140 CD400 0140 CD400 0140 CD400 0143 47 0144 0E00 0144 0E00 0144 B7 0148 C9 0146 E9 014F CA600 0158 C22F0 0158 C22F0 0158 C22F0 0158 C400 0158 C22F0 0158 C400 0158 C400 0158 C400 0158 C22F0 0158 C400 0158 C400	RCS\$ST: DB DB DB DB BOO RCS\$BSS: DB RCS: INX MVI DCX PUSH CALL POP MVI ANA JNZ I RCS\$UST: 1 CALL JZ	ODH OAH O BS, ' ', BS, O H M, O H RCS\$GC H A, RCS\$TERM B RCS\$UST D, RCS\$ST	<pre>;Internal standard terminator table ;Carriage return ;Line feed ;End of table ;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
0125 0A 0126 00 0127 08200 0127 08200 012E 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0134 3E08 0134 3E08 0135 CD920 0137 C23D0 0137 C23D0 0137 C23D0 0137 C23D0 0137 C23D0 0137 C23D0 0137 C23D0 0137 C23D0 0140 CD400 0140 CD7F0 0143 47 0144 0E00 0144 CD7F0 0145 CD7F0 0145 CD7F0 0145 CD7F0 0155 CD8B0 0155 CD8B0 0158 C32F0 0158 C32F0 00000000000000000000000000	DB DB DB DB RCS*BSS: DB RCS: INX MVI DCX RCS*L: PUSH CALL POP MVI CALL I I CALL JZ	OAH O BS, ' ', BS, O H M, O H RCS\$GC H A, RCS\$TERM B RCS\$UST D, RCS\$ST	<pre>;Carriage return ;Line feed ;End of table ;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Get character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
0125 0A 0126 00 0127 08200 0128 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 0146 CD7F0 0155 CD8B0 0146 C27F0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0158 6600 0158 6780 0158 6780 00000000000000000000000000000000000	DB DB DB DB RCS*BSS: DB RCS: INX MVI DCX RCS*L: PUSH CALL POP MVI CALL I I CALL JZ	OAH O BS, ' ', BS, O H M, O H RCS\$GC H A, RCS\$TERM B RCS\$UST D, RCS\$ST	<pre>;Carriage return ;Line feed ;End of table ;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Get character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
0125 0A 0126 00 0127 08200 0128 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 0146 CD7F0 0155 CD8B0 0146 C27F0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0158 6600 0158 6780 0158 6780 00000000000000000000000000000000000	DB DB DB RCS\$DB RCS: INX MVI DCX INX MVI DCX INX MVI ANA JNZ INX RCS\$LS POP MVI ANA JNZ INX INX RCS\$LS INX INX POS INX INX RCS INX MVI DCX INX INX INX INX INX INX INX INX INX IN	OAH O BS, ' ', BS, O H M, O H RCS\$GC H A, RCS\$TERM B RCS\$UST D, RCS\$ST	<pre>;Line feed ;End of table ;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
0126 00 0127 08200 0128 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0134 3208 0135 CD920 0134 11240 0130 CD40 0130 CD40 0130 CA4C0 0140 CA4C0 0140 CA4C0 0143 47 0144 0E00 0146 CD7F0 0146 CD7F0 0146 SP 014F CA600 0152 CD7F0 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD840 0155 CD840 0155 CD840 0155 CD840 0155 CD840 0155 CD840 0155 CD840 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD850 0155 CD840 0155 CD850 0155 CD850 0158 C340	DB RCS\$BSS: DB RCS: NX MVI DCX RCS\$L: PUSH CALL POP MVI CALL NZ LXI RCS\$UST: 1 JZ	0 BS,1 1,BS,0 H M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;End of table ;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
0127 08200 012E 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0137 C23D0 0130 CDD40 0140 CD400 0140 CD400 0140 CD400 0140 CD7F0 0143 47 0144 0E00 0144 0E00 0144 0E00 0145 CD7F0 0149 78 0144 D78 0144 C9 0145 C3E08 014E B9 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 CD8D0 0155 C22F0 0155 CC240 0158 0600 0158 0600 00000 000000 0000000 0000000000	RCS\$BSS: BOO RCS: INX MVI DCX RCS\$L: PUSH CALL POP MVI ANA JNZ I RCS\$UST: 1 JZ	BS, ' ', BS, O H M, O H RCS\$GC H A, RCS\$TERM B RCS\$UST D, RCS\$ST	<pre>;Destructive backspace sequence ;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Save buffer pointer ;Bet character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012E 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0136 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 D78 0144 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 0146 C27F0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD840 0158 C600 0158 C7270 0158 C600 0158 C7270 0158 C7270 0	800 DB RCS: INX MVI DCX RCS\$L: PUSH CALL POP MVI ANA JNZ LXI RCS\$UST: 1 JZ	H M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Bet character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012E 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0136 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 D78 0144 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 0146 C27F0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD840 0158 C600 0158 C7270 0158 C600 0158 C7270 0158 C7270 0	800 DB RCS: INX MVI DCX RCS\$L: PUSH CALL POP MVI ANA JNZ LXI RCS\$UST: 1 JZ	H M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;<<<<< Main entry ;HL -> actual count ;Reset to initial state ;HL -> max. count ;Bet character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012E 23 012C 3600 012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0136 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 D78 0144 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 0146 C27F0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD8D0 0155 CD840 0158 C600 0158 C7270 0158 C600 0158 C7270 0158 C7270 0	RCS: INX MVI DCX RCS\$L: PUSH CALL POP MVI ANA JNZ I RCS\$UST: CALL JZ	H M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;HL -> actual count ;Reset to initial state ;HL -> max. count ;Get character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012C 3600 012E 28 012F E5 0130 CD920 0133 E1 0134 3E08 0134 3E08 0136 A0 0137 C23D0 0137 C23D0 0130 CD940 0140 CA4C0 0140 CA4C0 0146 CD7F0 0148 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0158 0600 0158 0600 0158 C3440	INX MVI DCX RCS\$L: PUSH 1 CALL POP MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 JZ	M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;HL -> actual count ;Reset to initial state ;HL -> max. count ;Get character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012C 3600 012E 28 012F E5 0130 CD920 0133 E1 0134 3E08 0134 3E08 0136 A0 0137 C23D0 0137 C23D0 0130 CD940 0140 CA4C0 0140 CA4C0 0146 CD7F0 0148 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0158 0600 0158 0600 0158 C3440	INX MVI DCX RCS\$L: PUSH 1 CALL POP MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 JZ	M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;HL -> actual count ;Reset to initial state ;HL -> max. count ;Get character and execute: ; ECH0, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012C 3600 012E 28 012F E5 0130 CD920 0133 E1 0134 3E08 0134 3E08 0136 A0 0137 C23D0 0137 C23D0 0130 CD940 0140 CA4C0 0140 CA4C0 0146 CD7F0 0148 C9 0146 CD7F0 0149 78 0144 B7 0148 C9 014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0158 0600 0158 0600 0158 C3440	MVI DCX RCS\$L: PUSH CALL POP MVI ANA 1 JNZ LXI RCS\$UST: 1 Z	M,0 H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;Reset to initial state ;HL -> max. count ;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012E 2B 012F E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 014F CA500 0152 CD7F0 0155 CD850 0155 CD850 0155 CD850 0155 CC3440 0150 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	DCX RCS\$L: PUSH CALL POP MVI ANA 1 JNZ LXI RCS\$UST: 1 JZ	H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	<pre>;HL →> max. count ;Save buffer pointer ;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators</pre>
012F E5 0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 0146 C9 0147 CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 CD840 0155 C22F0 0158 0600 0158 0600 0150 C3440	RCS\$L: POP POP MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 JZ	H RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	;Save buffer pointer ;Get character and execute; ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators
0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0137 C23D0 0130 CDD40 0140 CA4C0 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0158 C22F0 0158 C600 0158 C3240 0158 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1 PUSH CALL POP MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators
0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0137 C23D0 0130 CDD40 0140 CA4C0 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0158 C22F0 0158 C600 0158 C3240 0158 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1 PUSH CALL POP MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators
0130 CD920 0133 E1 0134 3E08 0136 A0 0137 C23D0 0137 C23D0 0130 CDD40 0140 CA4C0 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0158 C22F0 0158 C600 0158 C3240 0158 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1 CALL POP MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	RCS\$GC H A,RCS\$TERM B RCS\$UST D,RCS\$ST	;Get character and execute: ; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators
0133 E1 0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0146 CD7F0 0149 78 0148 B7 0148 B7 014E B7 014E B7 014E B7 0152 CD7F0 0155 CDBB0 0155 CDBB0 0158 C22F0 0155 CDB00 0158 C440	POP MVI 1 JNZ 1 LXI RCS\$UST: 1 JZ	H A.RCS\$TERM B RCS\$UST D.RCS\$ST	; ECHO, ABORT, and FOLD options ;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators
0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 0148 C9 0148 C9 0148 C9 0148 C9 0148 C9 0148 C9 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0155 C22F0 0155 C3440	MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	A, RCS\$TERM B RCS\$UST D, RCS\$ST	;C = character input ;Recover buffer pointer ;Check if user-specified terminator ;B = options ;User specified terminators
0134 3E08 0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0144 B7 0148 C9 0148 C9 0148 C9 0148 C9 0148 C9 0148 C9 0148 C9 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0155 C22F0 0155 C3440	MVI ANA 1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	A, RCS\$TERM B RCS\$UST D, RCS\$ST	;Check if user-specified terminator ;B = options ;User specified terminators
0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0146 CD7F0 0146 CD7F0 0149 78 0144 B7 014B C9 014C 3E08 014E B9 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 CC22F0 0158 0600 015D C3440	ANA 1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	B RCS\$UST D,RCS\$ST	;B = options ;User specified terminators
0136 A0 0137 C23D0 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0146 CD7F0 0146 CD7F0 0149 78 0144 B7 014B C9 014C 3E08 014E B9 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0158 C22F0 0158 0600 015D C3440	1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	B RCS\$UST D,RCS\$ST	User specified terminators
0137 C2300 013A 11240 013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 0146 CD7F0 0149 78 0148 E9 0146 E9 014F CA600 0152 CD7F0 0155 CDEB0 0155 CDEB0 0155 CDEB0 0155 CC240 0155 CA400 0150 C3440	1 JNZ 1 LXI RCS\$UST: 1 CALL 1 JZ	D,RCS\$ST	
013D CDD40 0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0146 CD7F0 0148 C9 0146 E9 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 CD8D0 0155 C3440 0150 C3440	RCS\$UST: 1 CALL 1 JZ		;Standard terminators
0140 CA4C0 0143 47 0144 0E00 0146 CD7F0 0149 78 014A B7 014B C9 014C 3E08 014E B9 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0155 C3440 0150 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1 CALL 1 JZ	RCS\$CT	
0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 014A B7 014B C9 014C 3E08 014E B9 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0155 C2470 0155 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1 CALL 1 JZ	RCS\$CT	
0140 CA4C0 0143 47 0144 0E00 0145 CD7F0 0149 78 014A B7 014B C9 014C 3E08 014E B9 014F CA600 0152 CD7F0 0155 CD8B0 0155 CD8B0 0155 C22F0 0155 C2470 0155 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1 JZ	RCS\$CT	
0143 47 0144 0E00 0146 0D7F0 0149 78 0144 B7 0148 C9 0146 E9 0146 E9 0152 CD7F0 0155 CDEB0 0155 CDEB0 0158 0600 015B 0600 015B 03440 0160 E5 0161 23 0162 35 0163 FA7A0	1 JZ 		;Check for terminator
0144 0E00 0146 CD7F0 0149 78 014A 87 014B C9 014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0158 C22F0 0158 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	MEIC	RCS\$NOTT	Not terminator
0146 CD7F0 0149 78 014A B7 014B C9 014C 3E08 014E B9 014F CA600 0155 CD8B0 0155 CD8B0 0155 CD8B0 0158 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1101	B,A	Preserve terminating char.
0146 CD7F0 0149 78 014A B7 014B C9 014C 3E08 014E B9 014F CA600 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 C22F0 0158 0600 0150 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	DOODNOT -		· (Mass - share - Samuela - Share - Abia - Anda)
0146 CD7F0 0149 78 014A B7 014B C9 014C 3E08 014E B9 014F CA600 0155 CD8B0 0155 CD8B0 0155 CD8B0 0155 C22F0 0158 0600 0150 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	RCS\$MCI: MVI	с,о	;(Max. char. input shares this code) ;Terminate buffer
0149 78 014A B7 014B C9 014C 3E08 014E B9 014F CA400 0152 CD7F0 0155 CD8B0 0158 C22F0 0158 0400 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0		RCS\$SC	;Save character
014A B7 014B C9 014C 3E08 014E B9 014F CA600 0152 CD7F0 0155 CD8B0 0158 0600 015B 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	MOV	A, B	;Save character ;Recover terminating char.
014B C9 014C 3E08 014E 89 014F CA600 0155 CD2B0 0155 CD2B0 0158 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	'ORA	A	;Set flags
014C 3E08 014E 89 014F CA600 0152 CD7F0 0155 CD8B0 0158 C22F0 0158 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	RET		,
014E B9 014F CA600 0152 CD7F0 0155 CDEB0 0158 C22F0 015B 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1121		
014E B9 014F CA600 0152 CD7F0 0155 CDEB0 0158 C22F0 015B 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	RCS\$NOTT:		;Not a terminator
014F CA600 0152 CD7F0 0155 CD8B0 0158 C22F0 015B 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	MVI	A, BS	;Check for backspace
0152 CD7F0 0155 CD8B0 0158 C22F0 0158 0600 0150 C3440 0160 E5 0161 23 0162 35 0163 FA7A0	CMP	C	
0152 CD7F0 0155 CD8B0 0158 C22F0 0158 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0	1 JZ	RCS\$BS	;Backspace entered
0155 CD6B0 0158 C22F0 0158 C600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0		RCS\$SC	;Save character in buffer
0158 C22F0 0158 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0		RCS\$UC	Update count
015B 0600 015D C3440 0160 E5 0161 23 0162 35 0163 FA7A0		RCS\$L	Not max. so get another char.
0160 E5 0161 23 0162 35 0163 FA7A0	MVI	в, о	;Fake terminating char.
0161 23 0162 35 0163 FA7A0	1 JMP	RCS\$MCI	;A = 0 for max. chars. input
0161 23 0162 35 0163 FA7A0			
0161 23 0162 35 0163 FA7A0	RCS\$BS:		;Backspace entered
0162 35 0163 FA7A0	PUSH	н	;Save buffer pointer
0163 FA7A0	INX	н	;HL -> actual count
	DCR	M	;Back up one
		RCS\$NBS	;Check if count negative
0166 21270		H, RCS\$BSS	;HL -> backspacing sequence .Nc - check if cabeing
0169 3E01	MVI ANA	A,RCS\$ECHO B	;No, check if echoing ;BS will have been echoed if so
016B A0		B RCS\$BSNE	;BS will have been echoed in so ;No, input BS not echoed
016C CA700 016F 23		H	;Bypass initial backspace
VIOF 20	1144		ywygiadd analaid dae daenag dae
	RCS\$BSNE:		
0170 C5	PUSH	В	;Save options and character
0171 D5	PUSH	D	;Save terminator table pointer
0172 CDF60	1 CALL	WCS	Write console string
0175 D1	POP	в	Recover terminator table pointer;
0176 C1	POP	В	Recover options and character
0177 C37B0	1 JMP	RCS\$BSX	;Exit from backspace logic
			B 1 1 - 0
017A 34	RCS\$NBS:	M	;Reset count to O
0178 51	RCS\$NBS: INR	н	Recover buffer pointer
017B E1	RCS\$NBS: INR RCS\$BSX:	n	;Get next character
017C C32F0	RCS\$NBS: INR RCS\$BSX: POP	RCS\$L	FOLL HEAL CHERWICH

Figure 5-8. (Continued)

	RCS\$SC:		;Save character in C in buffer
			;Save character in c in buiter ;HL -> buffer pointer
017F D5	PUS	SH D	Save terminator table pointer
0180 E5	PUS		Save buffer pointer
0181 23	IN		;HL -> actual count in buffer
0182 5E	MOV		;Get actual count
0183 10	INF		;Get actual count ;Count of O points to first data byte
0184 1600	MVI		Make word value of actual count
0186 19	DAL		;HL -> next free data byte
0187 71	MOV		;HL -> next free data byte ;Save data byte away
0188 E1	POP		;save data byte away ;Recover buffer pointer
0189 D1	POF		Recover buffer pointer Recover terminator table
	20F		;Recover terminator table ; pointer
018A C9	RET		, pullier
	RCS\$UC:		;Update buffer count and check for max.
			Return Z set if = to max., NZ
			; if not HL -> buffer on entry
018B E5	PUS	н н	Save buffer pointer
018C 7E	MOV		;Get max. count
018D 23	INX		HL -> actual count
018E 34	INR		Increase actual count
018F BE	CMP		;Compare max. to actual
0190 E1	POP		Recover buffer pointer
0191 C9	RET		;Z-flag set
	RCS\$GC:		;Get character and execute
			; ECHO, ABORT and FOLD options
0192 D5	PUS		Save terminator table pointer
0193 E5	PUS	н н	Save buffer pointer
0194 C5	PUS	н в	Save option flags
	RCS\$WT:		
0195 0E06	MVI		
0197 1EFF	MVI		Specify input
0199 CD0500	CAL		
019C B7	ORA		Check if data waiting
019D CA9501	JZ	RCS\$WT	:Go back and wait
01A0 C1	POP	B	Recover option flags
01A1 4F	MOV	C, A	Save data byte
01A2 3E02	MVI	A, RCS\$ABORT	;Check if abort option enabled
01A4 A0	ANA	В	
01A5 CAAE01	JZ	RCS\$NA	;No abort
01A8 3E03	MVI	A, CTL\$C	;Check for control-C
01AA B9	CMP	ç	
01AB CA0000	JZ	0	;Warm boot
	RCS\$NA;		
01AE 3E04	MVI	A, RCS\$FOLD	;Check if folding enabled
01B0 A0	ANA	B	,
01B1 C4E501	CNZ	TOUPPER	;Convert to uppercase
01B4 3E01	MVI	A, RCS\$ECHO	Check if echo required
01B6 A0	ANA	B	,
01B7 CAD101	JZ	RCS\$NE	;No echo required
OIBA C5	PUSI		Save options and character
01BB 59	MOV	Ĕ,C	Move character for output
01BC OEO6	MVI	C, B\$DIRCONIO	Function code
01BE CD0500	CALI		Echo character
01C1 C1	POP	B	Recover options and character
01C2 3EOD	MVI	A, CR	Check if carriage return
01C4 B9	CMP	¢	
01C5 C2D101	JNZ	RCS\$NE	: Nc
01C8 C5	PUSI		Save options and character
01C9 0E06	MVI		Function code
ŎĨĊŔ ĬĔŎĂ	MVİ	C,B#DIRCONIO E,LF	;Function code ;Output line feed
	CALL		
01CD CD0500	POP	B	Recover options and character
		-	
01D0 C1	RCS\$NE:		-
01D0 C1 01D1 E1	POP	H	Recover buffer pointer
01CD CD0500 01D0 C1 01D1 E1 01D2 D1 01D2 C9		H D	;Recover buffer pointer ;Recover terminator table ;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
1	01D4 D5		PUSH	п	; character)
	0104 DO		FUSH	ر	;Save table pointer
I					
		RCS&CT	L:		
	01D5 1A		LDAX	D	;Get next terminator character
	01D6 B7		ORA.	A	;Check for end of table
	01D7 CAE201 01DA B9		JZ	RCS#CTX	No terminator matched
			CMP	C	Compare to input character
	01DB CAE201		JZ	RCS\$CTX	Terminator matched
	01DE 13 01DF C3D501		INX	D RCS\$CTL	;Move to next terminator ; loop to try next character in table
	010/ 000001		011	ROOFCIE	, toop to the next character in table
		RCS\$CT)			;Check terminator exit
	01E2 B7		ORA	A	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.
	01E3 D1		POP	D	;Recover table pointer
	01E4 C9		RET		
		; TOUPPE	ER - Fol	d lowercase lett	ers to upper
		;		aracter on entry	
		•			
		TOUPPER			
	01E5 3E60		MVI	A, a -1	;Check if folding needed
	01E7 B9		CMP	С	;Compare to input char.
	01E8 D2F501		JNC	TOUPX	No, char. is $\langle or = "a"-1$
	01EB 3E7A		MVI	A, 'z'	;Maybe, char. is = or > "a"
	OIED B9		CMP	C	- No No - H
	01EE DAF501		JC	TOUPX	;No, char. is > "z"
	01F1 3EDF		MVI	A, ODFH	;Fold character
	01F3 A1 01F4 4F		ANA MOV	C C,A	Return folded character
	0164 46		nuv	U, M	Averurn folded character
		TOUPX:			
	01F5 C9		RET		
		;WCS -	Write c	onsole string (u	(Sing raw 1/U)
					byte is encountered. Men a line feed is
		JA Cari Jencoui		tarn is output w	NICH C TTHE FEED TO
		, encour			
		;Calli	ng seque	nce	
		;	LXI	H, BUFFER	
		1	CALL	WCS	
		·Euit -	aramete	**	
		JEX1()		rs OOH byte termina	tor
		,	/ Man. /	oon byte termine	
		WCS:			
	01F6 E5		PUSH	н	;Save buffer pointer
	01F7 7E		MOV	A, M	;Get next character
	01F8 B7		ORA	A	Check if OOH
	01F9 CA1602		JZ	WCSX	;Yes, exit
	01FC FEOA		CPI	LF	Check if line feed
	01FE CC0C02		CZ	WCSLF	;Yes, output a carriage return
	0201 5F		MOV	E,A	;Character to be output
	0202 0E06		MVI	C, B\$DIRCONIO	;Function code
	0204 CD0500		CALL	BDOS	;Output character
	0207 E1		POP	H	Recover buffer pointer
	0208 23		INX JMP	H WCS	;Update to next char.
	0209 C3F601		Unit	WC3	;Output next char.
		WCSLF:			<pre>\$Line feed encountered</pre>
	020C 0E06		MVI	C, B\$DIRCONIO	Function code

Figure 5-8. (Continued)

020E 0210	1EOD CD0500		MVI CALL	E,CR BDOS	;Output a CR
0213 0215	3E0A		MVI RET	A, LF	;Recreate line feed ;Output LF
		WCSX:			Exit
0216			POP	н	Balance the stack
0217	C9		RET		

Figure 5-8. (Continued)

Example

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.

Notes 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 equat ;These are fo	es r accessing the	IOBYTE.
			fic devices. to preserve all BUT the
0003 =	IO\$CONM EQU	0000\$0011B	:Console mask
000C =	IO\$RDRM EQU	0000\$1100B	Reader mask
0030 =	IO\$PUNM EQU	0011\$0000B	;Punch mask
= 00C0	IO\$LSTM EQU	1100\$0000B	;List mask
			;Console values
0000 =	IO\$CTTY EQU	0	;Console -> TTY:
0001 =	IO\$CCRT EQU	1	;Console ~> CRT:
0002 =	IO\$CBAT EQU	2	;Console input <- RDR:
		_	;Console output -> LST:
0003 =	IO\$CUC1 EQU	з	;Console -> UC1: (user console 1)
			;Reader values
= 0000	IO\$RTTY EQU	0 SHL 2	;Reader <- TTY:
0004 =	IO\$RRDR EQU	1 SHL 2	;Reader <- RDR:
= 8000	IO\$RUR1 EQU	2 SHL 2	Reader <- UR1: (user reader 1)
= 3000	IO\$RUR2 EQU	3 SHL 2	;Reader <- UR2: (user reader 2)
			;Punch values
= 0000	IO\$PTTY EQU	O SHL 4	;Punch -> TTY:
0010 =	IO\$PPUN EQU	1 SHL 4	;Punch -> PUN:
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
= 0000	10\$LTTY EQU	O SHL 6	;List -> TTY:
0040 =	IO\$LCRT EQU	1 SHL 6	;List -> CRT:
0080 =	IQ\$LLPT EQU	2 SHL 6	;List -> LPT: (physical line printer)
>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 =	B\$GETIO	EQU 7	; Set	IOBYTE
0008 =	B\$SETIO	EQU 8		IOBYTE
0005 =	BDOS	EQU 5		entry point
= 2000	IO\$RDRM		000\$1100B	;Reader bit mask
= 8000	IO\$RUR1		SHL 2	;User reader select
	;This example ;reader to the			
0100 0100 0E07	ORG MVI	100H C,B\$GETIO	;Firs	t, get current IOBYTE

0102 CD0500 0105 E6F3	CALL ANI	BDOS (NOT IO\$RDRM)	AND OFFH ;Preserve all but ; reader bits
0107 F608	ORI	IO\$RUR1	:OR in new setting
0109 5F	MOV	E.A	Ready for set IOBYTE
010A 0E08	MVI	C, B\$SETIO	;Set new value
010C CB0500	CALL	BDOS	

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\$GETIO	EQU	7	;Get IOBYTE
0008 =	B\$SETIO	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\$0	011B	;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\$N	ΟΚΊ	No console input
0009 CD4B00	CALL	CONIN		:Get keyboard character
000C CD3000	CALL	SETCOM	м	Connect console -> comm. line
000F CD4500	CALL	CONOUT		Butput to comm. line
0001 024000	ONEL	0011001		youtput to commit Time
	TERM\$CCS:			;Check comm. status
0012 CD5200	CALL	CONST		;Get "console" status
0015 CA0000	JZ	TERM		;No incoming comm. character
0018 CD4B00	CALL	CONIN		:Get incoming comm. character

Figure 5-11. Simple terminal emulator

001B	CD2A00		CALL	SETCRT	;Connect console -> CRT:
001E	CD4500		CALL	CONOUT	;Output to CRT
0021	C30300		JMP	TERM\$CKS	LOOP back to check keyboard status
		TERM\$NO			
	CD3000		CALL	SETCOMM	;Connect console -> comm. line
0027	C31200		JMP	TERMSCCS	;Loop back to check comm. status
		SETCRT:			:Connect console -> CRT:
002A	F5		PUSH	PSW	Save possible data character
	0601		MVI	B. IO\$CCRT	Connect console -> CRT:
	C33300		JMP	SETCON	Common code
~~~~	-	SETCOMM			;Connect console -> comm. line
0030			PUSH	PSW	Save possible data character
0031	0603		MVI	B,IO\$CUC1	;Connect console -> comm. line
					;Drop into SETCON
		SETCON:			;Set console device
					New code in B (in bits 1.0)
0033	C5		PUSH	в	Save code
0034	0E07		MVI	C.B\$GETIO	Get current IOBYTE
0036	CD0500		CALL	BDOS	• • • • • • • • • • • • • • • • • • • •
0039	E6FC		ANI	(NOT IO\$CONM)	AND OFFH :Preserve all but console
003B	C1		POP	в	Recover required code
003C	BO		ORA	в	;OR in new bits
003D	SF		MOV	E.A	Ready for setting
	0E08		MVI	C, B\$SETIO	Function code
0040	CD0500		CALL	BDOS	
0043	F1		POP	PSW	Recover possible data character
0044	C9		RET		
		CONOUT:			
0045	5E	CONDUTE	MOV	E,A	:Get data byte for output
	0E06		MVI	C, B\$DIRCONIO	
	C30500		JMP	BDOS	Function code
0048	030500		JULE	BD08	;BDOS returns to CONOUT's caller
		CONIN:			
	0E06		MVI	C,B\$DIRCONIO	;Function code
004D	1EFF		MVI	E, OFFH	;Indicate console input
004F	C30500		JMP	BDOS	;BDOS returns to CONIN's caller
		CONST:			
0052	OEOB		MVI	C, B\$CONST	Function code
	CD0500		CALL	BDOS	, and then the
0057			ORA	A	:Set Z-flag to result
0058			RET		yest a ring to result

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	ODOA095468MESSAGE		DB	CR,LF,TA	B, This i	s a message',CR,LF,'\$'
0019	0E09 110000 CD0500	MVI LXI CALL	C, B\$PRIN D, MESSAG BDOS		;Function ;Pointer	code to message

**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.

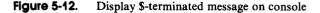
While the BDOS is outputting the string, it expands tabs as previously described, checks to see if there is an incoming character, and checks for CONTROL-S (XOFF, which stops the output until another character is entered) or CONTROL-P (which turns on or off echoing of console characters to the printer).

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.

```
10M (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.
Fach message is declared with a "$" as its last character.
If the A register contains a value larger than the number
; of messages declared, OM will output "Unknown Message".
;As an option, OM can output carriage return / line feed
; prior to outputting the message text.
Entry parameters
        HL -> message table
                   This has the form :
                                           Number of messages in table
                     DB
                                 3
                      DW
                                 MSGO
                                            ;Address of text (A = 0)
                      DW
                                 MSG1
                                            1(A = 1)
                                            i(A = 2)
                     DM
                                 MSG2
           MSGO:
                     DB
                                 'Message text$'
                                 ...etc.
                     A = Message code (from 0 on up)
B = Output CR/LF if non-zero
```



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

Figure 5-12. (Continued)

# Function 10: Read Console String

The CP/M Programmer's Handbook

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

90

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	O 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

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

#### 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
                   .
                                       DW
                                                 TEXT1, SUBPROC1
                                       DW
                                                 0
                                                           ;Terminator
                                                 'add',0 ;00H-byte terminated
                             TEXTO:
                                       ĎΒ
                             TEXT1:
                                       DB
                                                 'subtract',0
                             SUBPROCO:
                                       Code for processing ADD function.
                   2
                             SUBPROC1:
                   :
                                       Code for processing SUBTRACT function.
                   :
                   :Exit parameters
                             DE -> operator input string (OOH-terminated
                   .
                             input string).
Carry Clear, HL -> subprocessor.
Carry Set, HL = 0000H.
                   :
                   2
                   ;
                   ;Calling sequence
                             LXI
                                       H, SUBPROCTAB
                                                            ;Subprocessor table
                             CALL
                                       RSA
                                                           ;Carry set only on error
;Fake CALL instruction
;Push return address on stack
                             JC
                                       ERROR
                             LXI
                                       D, RETURN
                   :
                             PUSH
                                       D
                   3
                                                           ;"CALL" to subprocessor
                             PCHL
                   :
                             RETURN:
                   :
000A =
                   B$READCONS
                                       EQU
                                                 10
                                                            ;Read console string into buffer
0005 =
                   BDOS
                                       EQU
                                                 5
                                                            ;BDOS entry point
0050 =
                   RSASBL
                                       FOU
                                                 80
                                                            ;Buffer length
0000 50
                   RSA$BHE:
                                       DR
                                                 RSA$BL
                                                            ;Max. no. of characters
                   RSA$ACTC:
                                       DB
                                                            ;Actual no. of characters
                                                 0
0002
                                       DS
                                                 RSA$BL
                                                          ;Buffer characters
;Safety terminator
                   RSA$BUEC:
0052 00
                                       DB
                   RSA:
0053 2B
                             DCX
                                       н
                                                            ;Adjust Subprocessor pointer
0054 2B
                             DCX
                                       н
                                                            ; for code below
                                                            ;Top of stack (TOS) -> subproc. table - 2
0055 E5
                             PUSH
                                       н
                                       C. BSREADCONS
0056 OEOA
                             MVT
                                                            ;Function code
;DE -> buffer
                                       D. RSASBUE
0058 110000
                             1 1 1
005B CD0500
                                       BDOS
                             CALL
                                                            ;Read operator input and
                                                            ; Convert to OOH-terminated
005E 210100
0061 5E
                                       H.RSA$ACTC
                                                            HL -> actual no. of chars. input
Get actual no. of chars. input
                             LXI
                             MOV
                                       E,M
0062 1600
0064 23
0065 19
                             MVI
                                       D, 0
                                                            ;Make into word value
                                                            HL -> first data character
HL -> first UNUSED character in buffer
                             INX
                                       н
                             DAD
                                       D
0066 3600
                             MVI
                                       Μ,Ο
                                                            ;Make input buffer OOH terminated
                   RSA$ML:
                                                            ;Compare input to specified values
                                                            ; Main loop
                                                            ;Recover subprocessor table pointer
;Move to top of next entry
;HL -> text address
0068 E1
0069 23
                             POP
                                       н
                             INX
                                       н
006A 23
                                       H
                             INX
006B 5E
                                       E,M
                                                            ;Get text address
                             MOV
```

Figure 5-13. Read console string for keyboard options

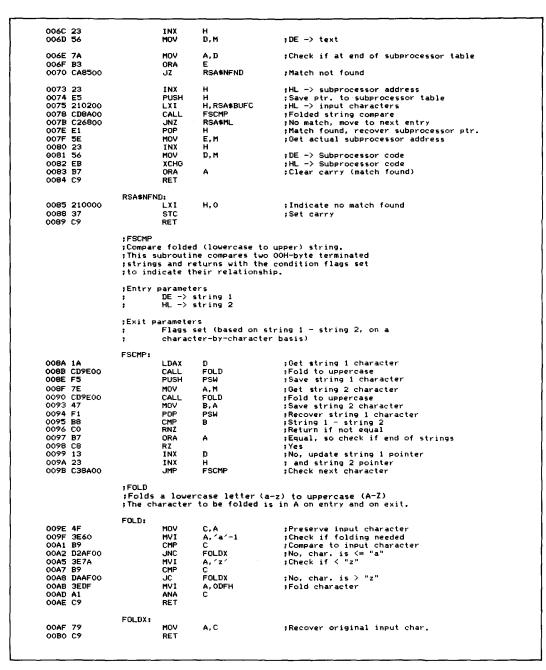


Figure 5-13. (Continued)

# **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\$CON: BDOS	ST	;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 = 0CH Entry 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

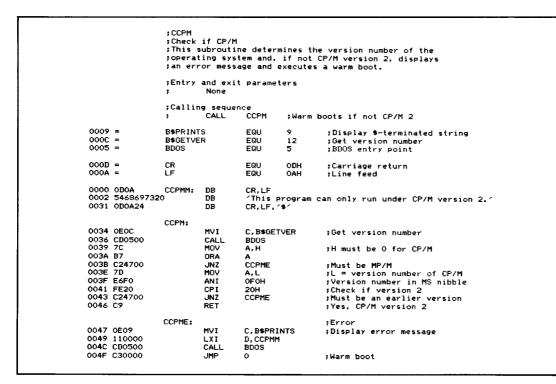
- L = 00H for all releases before CP/M 2.0
- L = 20H for CP/M 2.0, 21H for 2.1, 22H for 2.2, and so on for any subsequent releases.

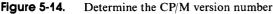
This information is of interest only if your program has some version-specific logic built into it. For example, CP/M version 1.4 does not support the same Random File Input/Output operations that CP/M 2.2 does. Therefore, if your program uses Random I/O, put this check at the beginning to ensure that 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 = 0DHEntry Parameters: None Exit Parameters: None





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\$DSK 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.

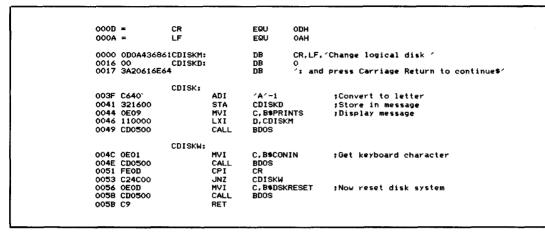
The BDOS will then have to log in each logical disk as each disk is accessed. This involves reading the entire file directory for the disk and rebuilding the allocation vectors (which keep track of which allocation blocks are free and which are used for file storage).

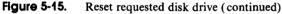
**Notes** This function lets you change the diskettes under program control. If the operator were to simply change diskettes, without CP/M knowing about it, the next access to the (now different) diskette would force CP/M to declare the disk Read-Only, thwarting any further attempts to write on the diskette. 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
               guser 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
                               A,0
                       MVI
                                                :Change drive A:
                                CDISK
                       CALL
               :
000D =
               B$DSKRESET
                                EQU
                                        13
                                                ;Disk Reset function code
0009 =
               B$PRINTS
                                        9
                                EQU
                                                #Print $-terminated string
                                                ;Get console input
0001 =
               B$CONIN
                                EQU
                                        1
5
0005 =
               BDOS
                                FOL
                                                ;BDOS entry point
```

Figure 5-15. Reset requested disk drive





## 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 =	BDOŚ	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

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	'FILEN/	AMÉ' ;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	000000000	OFCB\$ABUSED:	DB	0.0.0.0	0,0,0,0,0 ;Allocation blocks used
0018	000000000	0	DB		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

**Purpose** This 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.
                                         : File not opened.
                            Carry set
                   :
                   ;Calling Sequence
                                      D,FCB
                            LXI
                            LXI
                                      H, FNAME
                            CALL
                                      OPENF
                   :
                            JC
                  .
                                      ERROR
                  ;where
;FCB:
                            DS
                                                         ;Space for file control block
                                      36
                  FNAME: DB
                                      A: TESTFILE. DAT ', 0
000F =
                  B$OPEN
                                      EQU
                                                         File Open function code
                                                15
0005 =
                  BDOS
                                      EQU
                                                5
                                                         BDOS entry point
                  OPENE:
0000 D5
                            PUSH
                                      D
                                                         Preserve pointer to FCB
                                      BF
0001 CB0C00
0004 0E0F
                            CALL
MVI
                                                         ;Build file control block
                                      C, BSOPEN
0006 D1
0007 CD0500
                            POP
                                      D
                                                         Recover pointer to FCB
                                      BDOS
                            CALL
000A 17
                            RAL
                                                         ; If A=OFFH, carry set
                                                         ;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 0'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
                  .
                            LXI
                                      H, FILENAME
                            CALL
                                      RF
                  BF:
```

Figure 5-16. Open file request

0000			INX	н	;Check if 2nd char. is ":"
000D '	7E		MOV	A, M	;Get character from file name
000E	2B		DCX	н	;HL -> now back at 1st char.
000F			CPI	111	;If ";", then disk specified
0011	C21C00		JNZ	BF\$ND	;No disk
0014	7E		MOV	A, M	;Get disk letter
0015	E61F		ANI	0001\$1111B	;A (41H) -> 1, B (42H) -> 2
0017	23		INX	н	;Bypass disk letter
0018	23		INX	н	Bypass ":"
	C31D00		JMP	BF\$SD	Store disk in FCB
****					
		BF\$ND:			;No disk present
001C	AF	21 41121	XRA	A	Indicate default disk
0010			A110	R .	, indicate deficient dish
		BF\$SD:			
001D	10	DI 400.	STAX	D	Store disk in FCB
001E			INX	Ď	;BE -> 1st char. of name in FCB
001F				C,8	
			MVI		;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
					;OOH byte
0024			CPI	·.·	;Check terminating character
0026	C22A00		JNZ	BF\$NT	;No file type specified
0029	23		INX	н	;Bypass "." in file name
		BF\$NT:			
002A	0E03		MVI	С,З	;File type length
	CD3700		CALL	BF\$GT	;Get token
••					Note if no file type is
					present BF\$GT will merely
					spacefill the FCB
002F	0600		MVI	R Ó	;O-fill the remainder of the FCB
0031	0E18		MVI	B,0 C,24	;36 - 12 (disk, name, type = 12 chars.)
	CD6400		CALL	BF\$FT	Re-use fill token S/R
0036			RET	DI TI	The use Titt token STA
0030	67		KE I		
		DEACT			
		;BF\$GT			
		Build I	FCB 9	et token	
		<b>_</b>		• • •	
				e scans a file n	
		placing	a charac	ters into a file	control block.
					haracter ("." or OOH),
				of the token is ·	
					remainder of the token
		pis fill	led with	"?".	
		;Entry p	paramete:	rs	
		;	DE -> 1	nto file control	block
		;	HL -> I	nto file name st	ring
		1		imum no. of char	
		•			
		;Exit p	arameter		
		;	File co	ntrol block cont	ains next token
			A = Ter	minating charact	er
		BF\$GT:			
0037	7E		MOV	A, M	;Get next string character
0038			ORA	A	Check if end of string
	CA5700		JZ	BF\$SFT	;Yes, space fill token
0030			CPI	·**	Check if ?-fill required
	CASCOO		JZ	BF\$QFT	;Yes, fill with ?
0032			CPI	Drewr i	;Assume current token is file
0041			OF I	•	jname
					;name ;Check if file type coming up
					;(If current token is file
					stype this check is
					(benighly redundant)
	CA5700		JZ	BF\$SFT	;Yes, space fill token
0046	12		STAX	D	None of the above, so store
				_	; in FCB
0047			INX	D	;Update FCB pointer
0048	23		INX	н	;Update string pointer

Figure 5-16. (Continued)

0049 004A	0D C23700		DCR JNZ	C BF\$GT	;Countdown on token length 🧼	
		BF\$SKIF	·:		Skip chars. until "." or OOH	
0040	7E		MOV	Α, Μ	Get next string character	
004E	B7		ORA	A .	Check if OOH	
004F	CS		RZ		iYes	
0050	FE2E		CPI	1.1	Check if "."	
0052	C8		RZ	•	:Yes	
0053			INX	н	;Update string pointer (only)	
0054	C34D00		JMP	BF\$SKIP	Try next character	
				DI VORTI	, IT / HEAT CHALACTER	
		BF\$SFT:			Space fill token	
0057	0620		MVI	в, ′ ′	Jopace (III toke)	
	C36400		JMP	BF\$FT	:Common fill token code	
• • • • •				2. 4. 1	BF\$FT returns to caller	
					, at the following to carren	
		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			PUSH	PSW	Save terminating character	
0065	78		MOV	А,В	;Get fill characer	
		BF\$FTL:			:Inner loop	
0066	12	00°#F1C1	STAX	D	Store in FCB	
0067			INX	D	;Update FCB Pointer	
0067			DCR	C D		
	00 C26600		JNZ	C BF\$FTL	;Downdate residual count ;Keep going	
			POP			
0060				PSW	Recover terminating character;	
006D	CY		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 CD0500	MVI LXI CALL	C,B\$CL D,FCB BDOS	.OSE	;Function code ;DE -> File control block ;A = 0,1,2,3 if successful ;A = OFFH 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 block Exit Parameters: A = Directory code

#### Example

0011	=	B\$SEARCHF	EQU	17	;Search First
0005	=	BDOS	EQU	5	BDOS entry point
		FCB:			;File control block
0000	00	FCB\$DISK:	DB	0	Search on default disk drive
0001	4649404536	FCB\$NAME:	DB	'FILE???	??' :Ambiguous file name
0009	543F50	FCB\$TYP:	DB	1T?P1	Ambiguous file type
0000	00	FCB\$EXTENT:	DB	0	Extent
	0000	FCB\$RESV:	DB	0.0	Reserved for CP/M
	00	FCB\$RECUSED:	DB		Records used in this extent
		OFCB\$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	FCB\$RANREC:	DW	-	Random rec. to read/write
0023	00	FCB\$RANRECO:	DB	0	;Random rec. overflow byte (MS)
0024	-0E11	MVI	C, B\$SEAF	RC:HF	Function code
0026	110000	LXI	D,FCB		:DE -> File control block
0029	CD0500	CALL	BDOS		A = 0, 1, 2, 3
					:(A * 32) + DMA -> directory
					; entry
					:A = OFFH if file name not
					; found
					,

**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 GNF	Jor	MÝI	A,1 for NEXT
		,	CALL				
	0011 = 0012 =	B\$SEARCI B\$SEARCI		EQU EQU	17 18		<pre>\$Search for first file name \$Search for next file name</pre>
	0012 = 001A =	BISETOM		EQU	26		;Set up DMA address
1	0005 =	BDOS		EQU	5		BDOS entry point
1	0080 =	GNFDMA	EQU	80H			:Default DMA address
1	000D =	GNFSVL	EQU	13			Save length (no. of chars to move)
	0024 =	GNFFCL	EQU	36			File control block length
	0000	GNFSV:	DS	GNFSVL			;Save area for file name/type
		GNF :					
	000D E5 000E D5		PUSH PUSH	H D			;Save FCB pointer ;Save file name pointer
	000F F5		PUSH	PSW			;Save first/next flag
							On the transmission of the second second
1	0010 118000 0013 0E1A		LXI MVI	D, GNFDM C, B\$SET			<pre>\$Set DMA to known address \$Function code</pre>
	0015 CD0500		CALL	BDOS	_,		
	0018 F1		POP	PSW			Recover first/next flag
	0019 E1 001A D1		POP POP	н D			Recover file name pointer
	001B D5		PUSH	Ď			Resave FCB pointer
							Check if FIRST or NEXT
	001C B7 001D C23E00		ORA JNZ	A GNFN			INEXT
	0020 CD9300		CALL	BF			Build file control block
	0023 E1		POP	н			Recover FCB pointer (to balance stack)
	0024 D8 0025 E5		RC PUSH	н			<pre>Return if error in file name Resave FCB pointer</pre>
	UUZJ EJ		10011				stere top pointer
							;Move ambiguous file name to
1							;save area ;HL -> FCB
	0026 110000		LXI	D, GNFSV	,		;DE -> save area
	0029 0E0D 0028 CD8A00		MVI CALL	C, GNFSV	'L		;Get save length
	002E D1		POP	D			Recover FCB pointer
	002F D5		PUSH	D			;and resave
	0030 0E11		MVI	C, B\$SEA	RCHF		Search FIRST
	0032 CD0500			BDOS			Recover FCB pointer
1	0035 E1 0036 FEFF		CPI	H OFFH			Check for error
1	0038 CA7D00		JZ	GNFEX			;Error exit
	003B C35D00		JMP	GNFC			;Common code
		GNFN:					;Execute search FIRST to re-
•							sestablish contact with
							;previous file ;User's FCB still has
							;name/type in it
1	003E CD7F00		CALL	GNFZF			;Zero-fill all but file name/type
1	0041 D1 0042 D5		POP PUSH	D			Recover FCB address
1	0043 0E11		MVI	C, B\$SEA	ARCHE		Re-find the file
1	0045 CD0500		CALL POP	BDOS D			Recover FCB pointer
1	0048 D1 0049 D5		POP PUSH	D			;Recover FLB pointer ;and resave
1	004A 210000		LXI	H, GNFSV	,		Move file name from save area
1	004D OEOD		MVI	C, GNFS	л		;into FCB ;Save area length
1	004F CD8A00		CALL	MOVE	-		
1			MVI				Search NEXT
1	0052 0E12 0054 CD0500		CALL	C,B\$SE4 BDOS	INCHIN		
	0057 E1		POP	н			Recover FCB address
1	0058 FEFF		CPI	OFFH GNFEX			Check for error
	005A CA7D00		JZ	ONCEA			FEITON EAST
1		GNFC:	-				Saus 500 address
1	005D E5 005E 87		PUSH ADD	H			;Save FCB address ;Multiply BDOS return code * 32
1	500E 5/			••			· · · · · · · · · · · · ·
1							

Figure 5-17. (Continued)

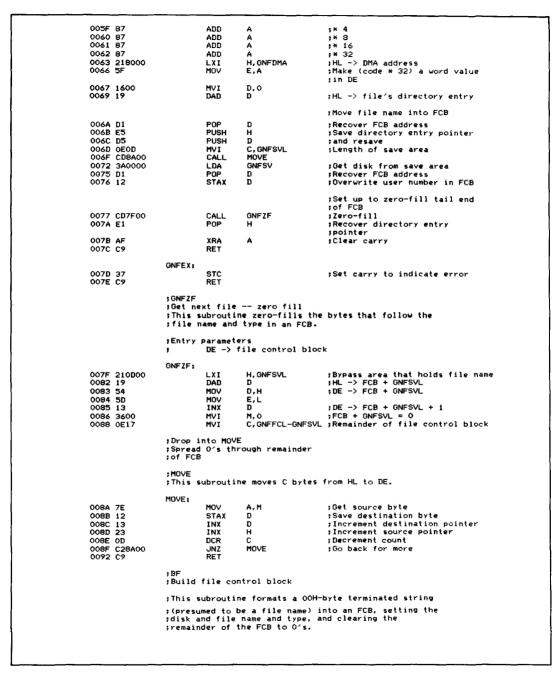


Figure 5-17. (Continued)

```
;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
```



## 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\$SEAR	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 = 13H
Entry Parameters:	DE = Address of file control block
Exit Parameters:	A = Directory code

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	177LENA	
0009	3F5950	FCB\$TYP:	DB	1?YP1	Ambiguous file type
000C	00	FCB\$EXTENT:	DB	0	;Extent
					•

000D 0000 000F 00 0010 00000000 0018 00000000 0020 00 0021 0000 0023 00		DB DB DB DB DB DB DW DB	;Reserved for CP/M ;Records used in this extent 0,0,0,0,0 ;Allocation blocks used 0,0,0,0,0 ;Sequential rec. to read/write ;Random rec. to read/write ;Random rec. overflow byte (MS)
0024 0E13	MVI	C,B\$E	;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\$READSE BDOS	Q	EQU EQU		;Read Sequential ;BDOS entry point
	00 4649404548 545950	FCB: FCB\$DISK EFCB\$NAME FCB\$TYP:		DB DB DB DS	0 1FILENAM 1TYP1	<pre>File control block ;Search on default disk drive. ME' ;file name ;File type ;Set by file open</pre>
0026	0E14 110000 CD0500		MVI LXI CALL	C, B\$REAU D, FCB BDOS	DSEQ	<pre>;Record will be read into ; address set by prior SETDMA ; call ;Function code ;DE -&gt; 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

```
: GETC
                 This subroutine gets the next character from a sequential disk file. It assumes that the file has
                  :already been opened.
                 ;>>>
                          Note : this subroutine changes CP/M's DMA address.
                  ;Entry parameters
                          DE -> file control block
                 ;
                 Exit parameters
                           A = next character from file
                               (= OFFH on physical end of file)
Note : 1AH is normal EOF character for
                  :
                 ;
                                       ASCII Files.
                 :
                 ;Calling sequence
                          LXI
                                    DÉ,FCB
                 :
                           CALL
                                    GETC
                 ;
                          CPI
                                    1AH
                 ;
                           JZ
                                    EDECHAR
                 1
                          CPT
                 ;
                                    OFFH
                                    ACTUALEOF
                           .17
                 •
0014 =
                 B$READSEQ
                                    EQU
                                             20
                                                      ;Read sequential
001A =
                 B$SETDMA
                                    EQU
                                                      ;Set DMA address
                                             26
0005 =
                 BDOS
                                    EQU
                                             5
                                                      BDOS entry point
0080 =
                 GETCBS EQU
                                    128
                                                      ;Buffer size
0000
                 GETCBF: DS
                                    GETCBS
                                                      ;Declare buffer
                                                      ;Char. count (initially
;"empty")
0080 00
                 GETCCC: DB
                                    0
                 GETC:
0081 3A8000
                          LDA
                                    GETCCC
                                                      :Check if buffer is empty
0084 B7
                          ORA
0085 CA9900
                          JZ
                                    GETCFB
                                                      ;Yes, fill buffer
                 GETCRE:
                                                      ;Re-entry point after buffer filled
0088 3D
                          DCR
                                                      ;No, downdate count
                                    GETCCC
0089 328000
                          STA
                                                      ;Save downdated count
008C 47
                          MOV
                                    B.A
                                                      ;Compute offset of next
                                                      ;character
                                    A, GETCBS-1
008D 3E7F
                          MVI
                                                      ;By subtracting
008F 90
0090 5F
                          SHB
                                                      ; (buffer size -- downdated count)
                                    в
                          MOV
                                   E,A
                                                      Make result into word value
0091 1600
                          MVI
                                   D, 0
0093 210000
                          LXI
                                   H, GETCBF
                                                      ;HL -> base of buffer
0096 19
                          DAD
                                   D
                                                      ;HL -> next character in buffer
0097 7E
                          MOV
                                    A,M
                                                      ;Get next character
0098 C9
                          RET
                 GETCFB:
                                                      ;Fill buffer
                                                      Save FCB pointer
0099 D5
                          PUSH
                                   n
009A 110000
                          LXI
                                   D, GETCBF
                                                      ;Set DMA address to buffer
009D 0E1A
                          MVI
                                   C, B$SETDMA
                                                      ;function code
009F CD0500
                          CALL
                                   BDOS
00A2 D1
00A3 0E14
                          POP
                                   n
                                                      Recover FCB pointer
Read sequential "record" (sector)
                          MVI
                                    C, B$READSEQ
00A5 CD0500
                          CALL
                                   BDOS
00A8 B7
00A9 C2B400
                          ORA
                                                      ;Check if read unsuccessful (A = NZ)
                                    GETCX
                           JNZ
00AC 3E80
00AE 328000
                                    A, GETCBS
                          MUT
                                                      ;Reset count
                          STA
                                   GETCCC
00B1 C38800
                          JMF
                                   GETCRE
                                                      :Rementer subroutine
                 GETCX:
                                                      ;Physical end of file
;Indicate such
OOB4 3EFF
                                   A, OFFH
                          MVI
00B6 C9
                          RET
```

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 block Exit Parameters: A = Directory code

0015 0005		B\$WRITESEQ BDOS	EQU EQU	21 5	;Write Sequential ;BDOS entry point
	00 4649404548 545950	FCB: FCB\$DISK: FCB\$NAME: FCB\$TYP:	DB DB DB	0 1FILENAI 1TYP1	;File control block ;Search on default disk drive ME' ;file name ;File type
000C			DS	24	Set by Open or Create File
0026	0E15 110000 CD0500	MVI LXI CALL	C,B\$WRI D,FCB BDOS	TESEQ	<pre>;Record must be in address ; set by prior SETDMA call ;Function code ;DE -&gt; File control block ;A = 00H if operation ; successful ;A = nonzero if disk full</pre>

## **112** The CP/M Programmer's Handbook

**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.

**Notes** 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

0016		B\$CREATE	EQU	22	;File Create
0005		BDOS	EQU	5 /	;BDOS entry point
	46494C454	FCB: FCB\$DISK: EFCB\$NAME: FCB\$TYP: FCB\$EXTENT:	DB DB DB DB	0 1FILENAI 1TYP1 0	;File control block ;Search on default disk drive ME′;file name ;File type ;Extent

000D 0000 000F 00	FCB\$RESV: FCB\$RECUSED:	DB DB	0,0	Reserved for CP/M Records used in this extent
0010 00000000	OOFCB\$ABUSED:	DB	0,0,0	,0,0,0,0,0 ;Allocation blocks used
0018 00000000	00	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)
				<pre>;Note : file to be created ;must not already exist</pre>
0024 OE16	MVI	C, B\$C	REATE	;Function code
0026 110000	LXI	D,FCB	: ·	;DE -> file control block
0029 CB0500	CALL	BDOS		;A = 0,1,2,3 if operation ; successful ;A = OFFH if directory full

		;This s ;to a s ;(128-b) ;remain	;PUTC ;This subroutine either puts the next charafacter out ;to a sequential file, writing out completed "records" ;(128-byte sectors) or, if requested to, will fill the ;remainder of the current "record" with 1AH's to ;indicate end of file to CP/M.						
		;Entry   ; ; ;	<pre>;Entry parameters ; DE -&gt; File control block ; B = 0, A = next data character to be output ; B /= 0, fill the current "record" with IAH's</pre>						
		;Exit p ;	aramete none.	rs					
		;Callin ; ; ; ; ; ; ; ;	LXI MVI LDA CALL LXI MVI	D,FCB B,O CHAR PUTC D,FCB B,1		and of file			
0015	_		CALL	PUTC EQU	21				
0015 001A 0005	=	B\$WRITE B\$SETDM BDOS		EQU EQU	26 5	;Write sequential ;Set DMA address ;BDOS entry point			
0090 0000 0080		PUTCBS PUTCBF: PUTCCC:	DS	128 PUTCBS O		;Buffer size ;Declare buffer ;Char. count (initially "empty")			
	F5 78	PUTC:	PUSH PUSH MOV ORA JNZ CALL	D PSW A, B A PUTCEF PUTCGA		;Save FCB address ;Save data character ;Check if end of file requested ;Yes ;No, get address of next free byte ;HL -> next free byte ;E = Current char, count (as			
0091	77 7B 3C FE80 CAA900 328000 D1		POP MOV MOV INR CPI JZ STA POP RET	PSW M,A A,E A PUTCBS PUTCCB PUTCCC D		<pre>;well as A) ;Recover data character ;Save in buffer ;Get current character count ;Update character count ;Check if buffer full ;Yes, write buffer ;No, save updated count ;Dump FCB address for return</pre>			



		PUTCEF:			;End of file
0099			POP	PSW	;Dump data character
0094	CDC300		CALL	PUTCGA	;HL -> next free byte
					;A = current character count
		PUTCCE:			COPY EOF character
	FE80		CPI	PUTCBS	Check for end of buffer
	CAA900		JZ	PUTCWB	;Yes, write out the buffer
	361A		MVI	M,1AH	No, store EOF in buffer
00A4			INR	Α	;Update count
0045			INX	н	;Update buffer pointer
00A6	C39D00		JMP	PUTCCE	<pre>;Continue until end of buffer</pre>
		PUTCWB:			;Write buffer
0049			XRA	A	Reset character count to 0
	328000		STA	PUTCCC	
	110000		LXI	D, PUTCBF	;DE -> buffer
	OE1A		MVI	C,B\$SETDMA	;Set DMA address -> buffer
	CD0500		CALL	BDOS	
00B5			POP	D	;Recover FCB address
	0E15		MVI	C,B\$WRITESEQ	;Write sequential record
	CD0500		CALL	BDOS	
OOBB			ORA	A	;Check if error
	C2C000		JNZ	PUTCX	;Yes if A = NZ
00BF	C9		RET		;No, return to caller
		PUTCX:			;Error exit
	3EFF		MVI	A, OFFH	;Indicate such
0002	C9		RET		
		PUTCGA:			;Return with HL −> next free char.
					;and A = current char, count
	348000		LDA	PUTCCC	;Get current character count
0006	5F		MOV	E,A	Make word value in DE
0007	1600		MVI	D, O	
0009	210000		LXI	H, PUTCBF	;HL -> Base of buffer
0000	19		DAD	D	;HL -> next free character
OOCD	C9		RET		

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 = 17HEntry Parameters:DE = Address of file control blockExit Parameters:A = Directory code

0017 =	<b>B\$RENAME</b>	EQU	23	;Rename file
0005 =	BDOS	EQU	5	;BDOS entry point
	FCB:			;File control block
0000 00		DB	0	Search on default disk drive
0001 4F4C4	44E41	DB	10LDN	AME ' ;File name
0009 54595	50	DB	TYP'	:File type
0000 0000	0000	DB	0,0,0	,0

```
;CF
                 ;Create file
                 ;This subroutine creates a file. It erases any
                 previous file before creating the new one.
                 ;Entry parameters
                           DE -> File control block for new file
                 :
                 ;Exit parameters
                          Carry clear if operation successful (A = 0, 1, 2, 3)
                 ;
                           Carry set if error (A = OFFH)
                 ;
                 ;Calling sequence
                                    D,FCB
                           LXI
                 2
                           CALL
                                    CF
                 7
                           JIC:
                 B$ERASE
B$CREATE
0013 =
                                    EQU
                                             19
                                                      :Erase file
                                    EQU
0016 = 0005 =
                                             22
5
                                                       ;Create file
                                    EQU
                                                       ;BDOS entry point
                 BDOS
                 CF:
0000 D5
                           PUSH
                                                       Preserve FCB pointer
                                    D
0001 0E13
0003 CD0500
                                                       ;Erase any existing file
                           MVI
                                    C, B$ERASE
                           CALL
                                    BDOS
0006 D1
0007 0E16
0009 CD0500
                           POP
                                                       ;Recover FCB pointer
                                    D
                           MVI
                                    C, B$CREATE
                                                       ;Create (and open new file)
                           CALL
                                    BDOS
000C FEFF
                           CPI
                                                       #Carry set if OK, clear if error
#Complete to use Carry set if Error
                                    OFFH
                           CMC
000E 3F
000F C9
```



### **116** The CP/M Programmer's Handbook

0010 00 0011 4E45574E41 0019 545950 001C 00000000		DB 0 DB 'NEWNAM DB 'TYP' DB 0,0,0,0	;File type
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‡GETACTDSK BDOS	;Example of getting active ; disk function code ;HL = active disk bit map ;Bits are = 1 if disk active ;Bits 15 14 13 2 1 0 ;Disk P 0 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

```
: RE
                    Rename file
                    This subroutine renames a file.
;It uses the BF (build FCB) subroutine shown in Figure 5.16
                    Entry parameters
                              H** 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
                                                                              -1 h
                    Calling sequence
                             N, OL DNAME
                                                             ;HL -> old name
;DE -> new name
                    .
                    2
                              CALL
                                        RF
                    1
                              JC
                                        ERROR
                    :
000F =
                    B$OPEN
                                        EQU
                                                  15
                                                             :Open file
0017 =
0005 =
                   B$RENAME
BDOS
                                        EQU
                                                   23
                                                             Rename file
                                        EQU
                                                   5
                                                             BDOS entry point
0000 000000000RFFCB:
                              nu
                                        0,0,0,0,0,0,0,0 ;1 1/2 FCB's long
0010 0000000000
                              DW
                                        0,0,0,0,0,0,0,0
0020 0000000000
                              Dω
                                        0,0,0,0,0,0,0,0
0030 000000
                              пы
                                        0,0,0
                   RF:
0036 D5
                              PUSH
                                                            Save new name pointer
0037 110000
                              LXI
                                        D.RFFCB
                                                             ;Build old name FCB
                                                             ;HL already -> old name
003A CD5D00
                             CALL
                                        RF
003D E1
                             POP
                                                            Recover new name pointer
003E 111000
                             LXI
                                        D, RFFCB+16
                                                            ;Build new name in second part of file
0041 CD5D00
                             CALL
                                        BE
                                                            control block
0044 111000
                             LXI
                                        D, RFFCB+16
                                                            Experimentally try
0047 0E0F
                                                            stą open the new file
sto ensure it does
                             MVI
                                        C, B$OPEN
0049 CD0500.
004C FEFF
                             CALL
                                        BDOS
                                        OFFH
                                                            inot already exist
Assume erfor (flags unchanged)
004E 3EFE
                             MVI
                                        A, OFEH
0050 DB
                             RC
                                                            Carry set if A was 0,1,2,3
0051 110000
                                       D.RFFCB
                             LXI
                                                            Rename the file
0054 OE17
0056 CD0500
                             MVI
                                        C, B$RENAME
                             CALL
                                        BDOS
0059 FEFF
                             CPI
                                        OFFH
                                                            Carry set if OK, clear if error
005B 3E
                             CMC
                                                            #Invert to use carry, set if error
0050 09
                             RET
                   • RE
                   ;Build file control block
                   This subroutine formats a OOH-byte terminated string f(presumed to be a file name) into an FCB, setting the disk and the file name and type, and clearing the gremainder 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
LXI
                                       D, FCB
                   1
                                       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

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

Chapter 5: The Basic Disk Operating System 119

0000	SECBUFF:	DS 128	;Sector buffer
0080 0E1A 0082 110000 0085 CD0500	MVI LXI CALL	C,B\$SETDMA D,SECBUFF 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.

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: None Exit Parameters: HL = Address of allocation vector

#### Example

Notes

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 = 1CH Entry 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 = 1DH Entry 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 point
0000 0E19 0002 CD0500	MVI Gall	C, B\$GET BDOS	RODSKS	;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

001E		BISETFAT			30	Set File Attribute
0005	2	BDOS	E	QU	5	;BDOS entry point
		FCB:				File control block
0000	00	FCB#DISK:	D	B	0	Search on default disk drive
0001	4649404546	FCB\$NAME:	D	B	'FILENAM	E′ ;File name
0009	D4	FCB\$TYP:	D	B	1T1+80H	Type with R/O
						; attribute
000A	5950		D	в	YP'	
000C	0000000000	<b>)</b>	ם	W	0,0,0,0,	0,0,0,0,0,0,0
0022	OE1E	MV	и с	, B\$SETF	AT	;Function code
0024	110000	LX	I D	,FCB		<pre>#DE -&gt; file control block #MS bits set in file name/type</pre>
0027	CD0500	CA	LL B	dos		A = OFFH if file not found

#### 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:

٠	f1' to f4'	Avai	lable	for	you	to	use
---	------------	------	-------	-----	-----	----	-----

- f5' to f8' Reserved for future CP/M use
- tl' 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.

```
:SFA
                ;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
                        LXI
                                 D, FCB
                :
                        LXI
                                 H,0000$0000$1100$0000B ;Bit Map
                         CALL
                                 SFA
                                 FRROR
                         .IC
                                                  ;File Attribute Equates
                                                           ;F11 - F41
8000 =
                FA$F1
                        EQU
                                 1000$0000$0000$00008
4000 =
                FA$F2
                        EQU
                                 0100$0000$0000$0000B
                                                           ;Available for use by
2000 =
                FA$F3
                        EQU
                                 0010$0000$0000$0000B
                                                           ; application programs
                FA$F4
                                 0001$0000$0000$0000B
1000 =
                        EQU
                                                           ;F51 - F81
0800 =
                EA&E5
                        FOU
                                 0000$1000$0000$0000B
                                 0000$0100$0000$0000B
                                                           Reserved for CP/M
0400 =
                FA$F6
                        FQU
                                 0000$0010$0000$0000B
0200 =
                FA$F7
                        EQU
                FA$F8
                        EQU
                                 0000$0001$0000$0000B
0100 =
                                                           :Ti' -- read/only file
0080 =
                FA$T1
                        EQU
                                 0000$0000$1000$0000B
0080 =
                FA$RO
                        EQU
                                 FA$T1
                                 0000$0000$0100$0000B
                                                           :T2' -- system files
                        EQU
0040 =
                FA$T2
0040 =
                FA$SYS
                        FQU
                                 FA$T2
                                 0000$0000$0010$0000B
                                                           :T3' -- reserved for CP/M
0020 =
                FA$T3
                        EQU
001E =
                B$SETFAT
                                 FOIL
                                         30
                                                  ;Set file attributes
                                         5
                                                  ;BDOS entry point
0005 =
                BDOS
                                 EQU
```

Figure 5-22. Set file attributes

		SFA:			
0000			PUSH	D	;Save FCB pointer
0001			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	A	Clear carry and A
0005	29		DAD	н	\$Shift next MS bit into carry
0006	CEOO		ACI	0	;A = 0 or 1 depending on carry
0008	0F		RRC		Rotate LS bit of A into MS bit;
0009	47		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	77		MOV	M, A	and store back into FCB
0010	EB		XCHG		;DE -> FCB, HL = remaining bit map
0011	13		INX	D	;DE -> next character in FCB
0012	OD		DCR	с	;Downdate character count
0013	C20400		JNZ	SFAL	;Loop back for next character
0016	OE1E		MVI	C, B\$SETFAT	Set file attribute function code
0018	D1		POP	D	Recover FCB pointer
0019	CD0500		CALL	BDOS	
0010	FEFF		CPI	OFFH	Carry set if OK, clear if error
001E	3F		CMC		;Invert to use carry set if error
001F	C9		RET		

Figure 5-22. Set file attributes (continued)

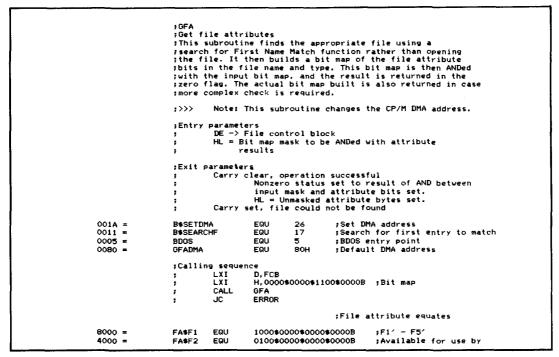


Figure 5-23. Get file attributes

2000 = 1000 =	FA\$F3 FA\$F4	EQU EQU	0010\$0000\$000		;Application programs
0000					
0800 = 0400 =	FA\$F5	EQU	0000\$1000\$000		1F6' - F8'
	FA#F6	EQU	0000\$0100\$000		;Reserved for CP/M
0200 = 0100 =	FA\$F7 FA\$F8	EQU EQU	0000\$0010\$000		
0080 = 0080 =	FA\$T1 FA\$RO	EQU EQU	0000\$0000\$100	0\$0000B	;T1′ read/only file
0040 =	FA\$RU	EQU	FA\$T1 0000\$0000\$010	0600008	;T2′ system files
0040 =	FASSYS	EQU	FA\$T2	0#0000B	jiz system illes
0020 =	FA\$T3	EQU	0000\$0000\$001	0\$0000B	;T3′ reserved for CP/M
	GFA:				
0000 E5		PUSH	н		ND-mask
0001 D5		PUSH	D		FCB pointer
0002 0E1A		MVI	C, B\$SETDMA		1A to default address
0004 118000		LXI	D, GFADMA	;DE ->	DMA address
0007 CD0500		CALL	BDOS		
000A D1		POP	р	:Recove	r FCB pointer
000B 0E11		MVI	C, B\$SEARCHF		for match with name
000D CD0500		CALL	BDOS	,	
0010 FEFF		CPI	OFFH	;Carry	set if OK, clear if error
0012 3F		CMC		;Invert	to use set carry if error
0013 DA4100		JC	GFAX	;Return	) if error
					ly by 32 to get offset into DMA buffer
0016 87 0017 87		ADD	A	;* 2	
0017 87		ADD	A	;* 4	
0018 87		ADD	A	;* 8	
0019 87 001A 87		ADD ADD	A	;* 16 ;* 32	
001B 5F		MOV	Ê, A		nto a word value
0010 1600		MVI	B,O	Friake 1	into a word value
001E 218000		LXI	H, GFADMA	:HI>	DMA address
0021 19		DAD	D	:HL ->	Directory entry in DMA buffer
0022 23		INX	Ĥ	THE ->	1st character of file name
0023 EB		XCHG		DE ->	ist character of file name
0024 0E0B 0026 210000		MVI LXI	C,8+3 H,0	;Count	of characters in file name and type bit map
	GFAL:		_	#Main 1	oop
0029 1A		LDAX	D		ext character of file name
002A E680		ANI	BOH		e attribute bit
002C 07 002D B5		RLC			IS bit into LS bit
002E 6F		MOV			any previously set bits
002F 29		DAD	L,A H	Save r	esult HL left one bit for next time
0030 13		INX	D		next character in file name, type
0031 00		DCR	C		ite count
0032 C22900		JNZ	GFAL		k for next character
0035 29		DAD	н	و المراجع ال	1.68818.0
		DAD	H		ustify attribute bits in HL ribute bit will already be in
0036 29 0037 29		DAD	н		of HL, so only 4 shifts are
0038 29		DAD	H	JDIC II JNECESS	
0039 D1		DOD	в	- De a a ····	
0039 D1 003A 7A		POP MOV	D A.D		r AND-mask
0038 /A		ANA	A,U H		byte of mask th MS byte of result
003B A4 003C 47		MOV	н В.А		nterim result
003D 7B		MOV	A,E		byte of mask
		ANA	L		th LS byte of result
003E A5			B		e two results to set Z flag
		ORA			
003E A5		ORA RET			
003E A5 003F B0 0040 C9	GFAX:	RET		;Error	exit
003E A5 003F B0 0040 C9 0041 E1	GFAX:	RET	н	;Error ;Balanc	
003E A5 003F B0 0040 C9	GFAX:	RET	н		
003E A5 003F B0 0040 C9 0041 E1	GFAX:	RET	н		
003E A5 003F B0 0040 C9 0041 E1	GFAX:	RET	н		

## Function 31: Get Disk Parameter Block Address

Notes

	-	Code: $C = 1$ meters: None meters: $HL = A$		DPR	
Example	001F =	BIGETOPB	EQU	31	;Get Disk Parameter Block : Address
	0005 =	BDOS	EQU	5	BDOS entry point
	0000 0E1F	MVI	C, 8\$GE	TDPB	Returns.DPB address of logical disk previously selected with a Select Disk function. Function code
	0002 CD0500	CALL	BDOS		;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.

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 = 001F = B\$SELDSK EQU 14 ;Select Disk function code B\$GETDPB EQU 31 ;Get DPB address 0005 = BDOS EQU 5 **;BDOS 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 DPBSPT: DW 0000 0000 ;128-byte sectors per track 0 0002 00 DPBBS: DB 0 ;Block shift 0003 00 DPBBM: DB DPBEM: DB ò Block mask 0 ;Extent mask 0005 0000 DPBMAB: DW 0 #Maximum allocation block number 0007 0000 0009 0000 DPBNOD: DW 0 ;Number of directory entries - 1 DPBDAB: DW 0 ;Directory allocation blocks ;Check buffer size 000B 0000 DPBCBS: DW 0 000D 0000 DPBTBD: DW Ô Tracks before directory (reserved tracks) 000F = DPBSZ EQU \$-DPB ;Disk parameter block size ; GETDPB **;Gets disk parameter block** ;This subroutine copies the DPB for the specified ;logical disk into the local DPB variables above. ;Entry parameters A = Logical disk number (A: = 0, B: = 1...) ; ;Exit parameters Local variables contain DPB GETDPB: MOV ;Get disk code for select disk 000E 5E E,A C, B\$SELDSK 0010 0E0E MUT ;Select the disk 0012 CD0500 0015 0E1F CALL BDOS MVI C, B\$GETDPB ;Get the disk parameter base address ;HL -> DPB 0017 CD0500 CALL BDOS 001A 0E0F 001C 110000 C, DPBSZ MVI ;Set count LXI D, DPB ;Get base address of local variables GDPBL: COPY DPB into local variables; Get byte from DPB 001E 7E MOV A.M 0020 12 STAX п ;Store into local variable 0021 13 0022 23 ;Update local variable pointer ;Update DPB pointer TNX n INX н 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 ; ;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
                                      NPRRS
                                                          ;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
002B C22B00
                             JNZ
                                       GTASS
                                                           More shifts required
0030 EB
                             XCHG
                                                          ;DE = all. block * sec. per block
;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.
                             XCHG
                                                          ;BC = HL/DE, HL = remainder
;BC = track, HL = sector
0035 CD8F00
                            CALL
                                       DIVHL
0038 23
                             INX
                                       н
                                                          Sector numbering starts from 1
0039 EB
003A 2A0D00
003D 09
                             XCHG
                                                          ;DE = sector, HL = track
                             LHLD
                                       DPBTBD
                                                          ;Tracks before director;
                             DAD
                                       R
                                                          ;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
                   2
                   GMTAS:
003F 2A0500
0042 C32800
                                      DPBMAB
                            LHLD
                                                          ;Get maximum allocation block
                            , IMP
                                      GTAS
                                                          ;Return from GTAS with parameters in HL and DE
                   ; 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
0048 23
                            INX
                                      н
                                                          ;Make true number of entries
;Each entry is 32 bytes long, so to
;convert to 128 byte sectors, divide by 4
0049 CDD000
004C CDD000
004F E5
                            CALL
                                      SHLR
                                                          ;/ 2 (by shifting HL right one bit)
                            CALL
                                      SHLR
                                                          ;/ 4
                            PUSH
                                      н
                                                          ;Save number of sectors
0050 210000
                                      H. 0
                                                         Directory starts in allocation block 0
HL = track, DE = sector
Recover number of sectors
                            1 XT
0053 CD2800
                                      GTAS
                            CALL
0056 C1
                            POP
                                      Ē
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
                 5
                :
                         equality occurs, the end of disk has been reached.
                .
                ;Entry parameters
                         HL = current track number
                 3
                         DE = current sector number
                 ;Exit parameters
                         HL = updated track number
DE = updated sector number
                 :
                GNTAS:
0058 E5
                         PUSH
                                  н
                                                    Save track
0059 13
005A 2A0000
005D CDC900
                                  D
                          INX
                                                    ;Update sector
                         LHLD
                                  DPBSPT
                                                    ;Get sectors per track
;HL = HL - DE
;Recover current track
                          CALL
                                   SUBHL
0060 E1
                          POP
                                   н
0061 D0
                          RNC
                                                    #Return if updated sector <= sec. per trk.</pre>
0062 23
                          INX
                                                    ;Update track if upd. sec > sec. per trk.
                                  н
0063 110100
                          LXI
                                   D, 1
                                                     Reset sector to 1
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
                         PUSH
                                  n
                                                     ;Save sector
;DE = track
0068 EB
                          XCHG
0069 2A0D00
                          LHLD
                                   DPBTBD
                                                     ;Get no. of tracks before directory
                                                    ; DE = no. of tracks before dir. HL = track
; HL = HL - DE
006C EB
                          XCHG
006D CDC900
                          CALL
                                   SUBHL
                                                     ;HL = relative track within logical disk
                                                     ;DE = relative track
                          XCHG
0070 EB
0071 2A0000
0074 CBA400
                                   DPBSPT
                                                    ;Get sectors per track
;HL = HL * DE
                          LHLD
                         CALL
                                   MULHL
                                                    ;HL = number of sectors
;DE = number of sectors
0077 EB
                          XCHG
```

Figure 5-24. (Continued)

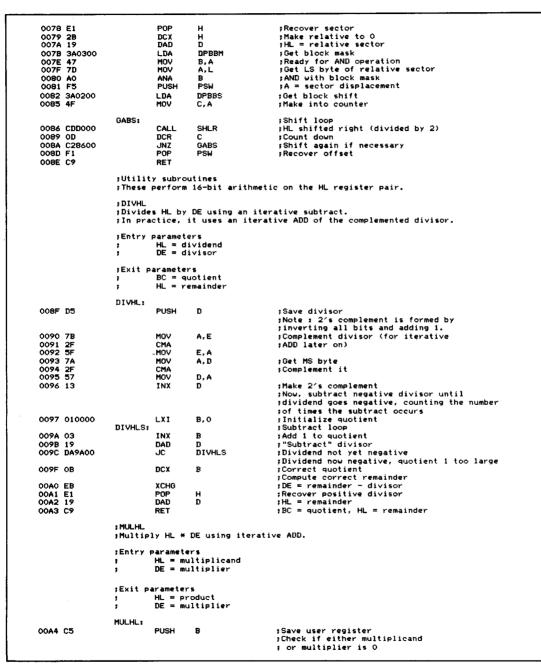


Figure 5-24. (Continued)

00A5 00A6			MOV ORA	A,H L	
	CAC400		JZ	MULHLZ	;Yes, fake product
00AA	7A		MOV	A,D	
OOAB			ORA	E	
OOAC	CAC400		JZ	MULHLZ	;Yes, fake product
					;This routine will be faster if
					; the smaller value is in DE
00AF 00B0	7A BC		MOV CMP	A, D H	Get MS byte of current DE value
	DAB500		JC	MULHLN	;Check which is smaller ;C set if D < H, so no exchange
00B4			XCHG		
00B5	42	MULHLN:	MOV	B,D	;BC = multiplier
00B6			MOV	C,E	
00B7			MOV	D,H	;DE = HL = multiplicand
00B8 00B9	50		MOV DCX	E,L B	Adjust count as
0089	0B		DUX	Б	;1 * multiplicand = multiplicand
					yr a marcipileana - marcipileana
		MULHLA:			;ADD loop
OOBA			MOV	A, B C	Check if all iterations completed
00BB 00BC	B1 CAC700		ORA JZ	L: MULHLX	;Yes, exit
OOBE			DAD	B	;HL = multiplicand + multiplicand
0000	OB		DCX	В	;Countdown on multiplier - 1
0001	C3BA00		JMP	MULHLA	;Loop back until all ADDs done
		MULHLZ:			
0004	210000		LXI	н, о	Fake product as either multiplicand
					; or multiplier is O
		MULHLX:			
0007	C1	HULHEAS	POP	в	Recover user register
0008			RET	-	
		; SUBHL		ne:	
		;Subtra	otHL − I	DE	
		;Entry g	parameter	rs	
		,	HL = sul	btrahend	
		;	DE = sul	btractor	
		:Exit p	arameter	5	
		;		fference	
		SUBHL:			
0009	7D		MOV SUB	A,L E	;Get LS byte ;Subtract without regard to carry
00CA 00CB			MOV	E L,A	;Subtract without regard to carry ;Put back into difference
00000			MOV	A, H	;Get MS byte
OOCD	9A		SBB	D	;Subtract including carry
00CE 00CF	67		MOV	Н,А	;Move back into difference
OUCF	67		RET		
		; SHLR			
		;Shift	HL right	one place (divi	ding HL by 2)
		. Catwo		~ =	
		;Entry	paramete HL = va	rs lue to be shifte	d
			arameter		
		;	HL = va	lue/2	
		SHLR:			
0000			ORA	Α	;Clear carry
0001			MOV	А,Н	;Get MS byte Bit 7 set from provinus corry
00B2	1F		RAR		Bit 7 set from previous carry, ; bit 0 goes into carry
0003	67		MOV	H, A	Put shift MS byte back
0004	7D		MOV	A,L	;Get LS byte
0005			RAR		Bit 7 = bit 0 of MS byte
00B6 00D7			MOV RET	L,A	Put back into result
 0007			- 16- 1		

Figure 5-24. (Continued)

130

## Function 32: Set/Get User Number

Function Code: C = 20HEntry 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>B\$SETGETUN</b>	EQU	32	;Set/Get User Number
0005 =	BDOS	EQU	5	;BDOS entry point
				;To set user number
0000 0E20	MVI	C,B\$SE	TGETUN	;Function code
0002 1EOF	MVI	E,15		Required user number;
0004 CD0500	CALL	BDOS		;To get user number
0007 0E20	MVI	C,B\$SE	TGETUN	;Function code
0009 1EFF	MVI	E,OFFH		;Indicate request to GET
000B CD0500	CALL	BDOS		;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 H
Entry Parameters:	DE = Address of FCB
Exit Parameters:	A = Return code

0021 =	B\$READRAN	EQU	33	;Read Random
0005 =	BDOS	EQU	5	;BDOS entry point
	FCB:			;File control block
0000 00	FCB\$DISK:	DB	0	Search on default disk drive
0001 4649404	54EFCB\$NAME:	DB	'FILE	NAME′ ;File name
0009 545950	FCB\$TYP:	DB	TYP1	File type

## **132** The CP/M Programmer's Handbook

000F 0010 0018 0020	0000 00 0000000000 0000000000 00 00 0000	FCB\$EXTENT: FCB\$RESV: FCB\$RECUSED: PFCB\$ABUSED: FCB\$RANREC: FCB\$RANREC: FCB\$RANREC0:	DB DB DB DB DB DB DW DB	0 0,0 0,0,0,0,0, 0,0,0,0,0,0 0 0 0	;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 CB0500	LHLD SHLD MVI LXI CALL	RANRECNO FCB\$RAN C, B\$REAL 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 -&gt;.file control block ;A = 00 if operation successful ;A = nonzero if no data in ; file specifically: ;A = 01 attempt to read ; unwritten record ; 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 = BD	OS			;BDOS entry point
FC	8:			File control block
0000 00 FC	B\$DISK:	DB	0	;Search on default disk drive
0001 46494C454EFCB\$NAME;		DB	'FILENAM	E' ;File name
0009 545950 FC	B\$TYP:	DB	TYP'	;File type
000C 00 FC	B\$EXTENT:	DB	0	Extent
000D 0000 FC	B\$RESV:	DB	0.0	Reserved for CP/M
000F 00 FC	B\$RECUSED:	DB	0	Records used in this extent
0010 000000000FC	B\$ABUSED:	DB	0,0,0,0,	0,0,0,0 ;Allocation blocks used
0018 000000000		DB	0.0.0.0.	0,0,0,0
0020 00 FC	B\$SEQREC:	DB	0	Sequential rec. to read/write
0021 0000 FC	B\$RANREC:	DW		Random rec. to read/write
0023 00 FC	B\$RANRECO:	DB	0	Random rec. overflow byte (MS)
0024 D204 RA	NRECNO	DW	1234	Example random record number;
				Record will be written from
				; address set by prior

; SETDMA call

0026 2A2400 LHL 0029 222100 SHL 002C 0E22 MVI 002E 110000 LXI 0031 CD0500 CAL	D FCB\$RANREC C,B\$WRITERAN D,FCB	;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.
                1
                ;Exit parameters
                        Carry clear if operation successful (A = 0, 1, 2, 3)
Carry set if error (A = OFFH)
                .
                ;Calling sequence
; LXI D
                                 D,FCB
                        CALL
                                 CRF
                ;
                         JC
                                 ERROR
0013 =
                B$ERASE
                                 FOIL
                                          19
                                                  ;Erase file
0016 =
                BSCREATE
                                 EQU
                                         22
26
                                                  ;Create file
;Set DMA address
001A =
                B$SETDMA
                                 EQU
0015 =
                B$WRITESEQ
                                 EQU
                                         21
                                                  Write sequential record
0005 =
                BDOS
                                 EQU
                                         5
                                                  : BDOS entry point
                CRFBUF:
                                                  Zero-filled buffer
0000 000000000
                        DW
                                 0,0,0
0032 0000000000
                        DW
                                 0.0.0
0064 0000000000
                        nы
                                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0080 0000
                CRFRC:
                        DW
                                 0
                                                  Record count
                CRF:
0082 228000
0085 D5
                        SHL D
                                 CRFRC
                                                  Save record count
                                                  Preserve FCB pointer
                        PUSH
                                 D
0086 0E13
                        MVI
                                 C, B$ERASE
                                                  ;Erase any existing file
0088 CD0500
                        CALL
                                 BDOS
008B D1
                        POP
                                 D
                                                  Recover FCB pointer
008C D5
                        PUSH
                                 D
                                                  ; and resave
008D 0E16
008F CD0500
                        MVI
                                 C, B$CREATE
                                                  (Create (and open new file)
                        CALL
                                 BDOS
0092 FEFF
                        CPI
                                 OFFH
                                                  ;Carry set if OK, clear if error
0094 3E
                        CMC
                                                  Complete to use carry set if error;
Recover FCB address
0095 D1
                        POP
                                 D
0096 D8
                        RC
                                                  Return if error
0097 D5
                        PUSH '
                                                  ;Resave FCB pointer
                                 D
0098 OE1A
                        MVI
                                 C, B$SETDMA
                                                  ;Set DMA address to 0-buffer
009A 110000
009D CD0500
                        LXI
                                 D, CRFBUF
                        CALL
                                 BDOS
00A0 D1
                        POP
                                 n
                                                  Recover FCB pointer
                CRFL:
00A1 2A8000
                        LHLD
                                 CRFRC
                                                  :Get record count
00A4 7D
                        MOV
                                 A.L
00A5 B4
                        ORA
                                 н
                                                  ;Check if count now zero
00A6 C8
                        RΖ
                                                  ;Yes, exit
00A7 2B
                        DCX
                                 н
                                                  ;Downdate count
                                 CRFRC
00A8 228000
                        SHLD
                                                  :Save count
OOAB D5
                        PUSH
                                                  Resave FCB address
                                 D
00AC 0E15
                        MVI
                                 C, B$WRITESEQ
                                                  ;Write sequentially
OOAE CD0500
                        CALL
                                 BDOS
00B1 D1
                        POP
                                                  Recover FCB
                                 CRFL
00B2 C3A100
                        JMP
                                                  ;Write next record
```

Figure 5-25. Create random file

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).

	\$R0			
	;Random operat	ion (rea	d or writ	(e)
	;This subrouti	ne reads	or write	es a random record from a file.
1				than 128-bytes. This
				CP/M record (which
				a, performs a random read record into a user buffer.
				will be read until the complete
	juser-specifie			
				e user-specified record is not an exact
				appropriate sectors will be preread.
				when the user-specified record
				when subroutine is processing by a user-specified record.
	jurin records	entirely	spanneu	by a user-specified record.
	;Entry paramet	ers		
-	; HL->	paramete	r block d	of the form:
	;	DB	0	;OFFH when reading, OOH for write
	;	DW	FCB	Pointer to FCB
	1	DW DW	RECNO	;User record number :User record size
	, :	DW	BUFFER	
ļ				: RECSZ bytes in length
1				
	;Exit paramete			
1	; A = 0			eted (and user record
				er buffer) ad unwritten CP/M record
				d unwritten cr/m record t close an extent
				d unwritten extent
-	; 5	if CP/M	could not	create a new extent
1	; 6	if attem	pt to rea	ad beyond end of disk
	;Calling seque			
1	f LXI	H. PARA	MS	;HL -> parameter block
	; CALL	RO		JUE / Parameter Drock
	; ORA	A		;Check if error
	≱ JNZ	ERROR		
	CODEADANDES	500		Offerst of wester worout as in ECP
0021 =	FCBE\$RANREC B\$SETDMA	EQU	33 26	;Offset of random record no. in FCB :Set the DMA address
0021 =	BISEIUMA	EQU	20 33	;Set the UMA address :Read random record
0028 =	B\$WRITERANZ	EQU	40	Write random record with zero-fill
				; previously unallocated allocation
				; blocks
0005 =	BDOS	EQU	5	;BDOS entry point
1	ROPB:			:Parameter block image
0000 00	ROREAD: DB	o		NZ when reading, Z when writing
0001 0000	ROFCB: DW	ŏ		Pointer to FCB
0003 0000	ROURN: DW	Ō		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	ROFRL:	DR	0	Freenand landth
	0000		RORNE		ŏ	;Fragment length ;Record number pointer (in user FCB)
	000E		ROWECR:		ŏ	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
	OORE	110000	RO:	LXI	D, ROPB	DF -> leastter black
	0092	110000 0E09		MVI	C, ROPBL	;DE -> local parameter block ;Parameter block length
	0094	CDFE01		CALL	MOVE	Move C bytes from HL to DE
						ser record in CP/M record,
				t compu	te the relative	BYTE offset of the start
				J DT CO	e user record wi	thin the file (i.e. record size). The least
				, use, i	ficant 7 bits of	this product give the
				; byte	offset of the st	art of the user record.
				The pro	oduct / 128 (shi	fted left 7 bits) gives the
						the start of the user record.
	0007	240500			DOUDI	
	0097 009A			LHLD MOV	ROURL A,L	#Get user record length
	009B			ANI	7FH	Get LS bytes of user rec. length
	009D	87		ORA		<pre>#Check if exact multiple of 128 #(i.e. exact CP/M records)</pre>
	009E			MVI	A, 0	PA = 0, flags unchanged
	00A00	C2A400		JNZ	RONE	#Not exact CP/M records
	00A3	3D		DCR	A	IA ≠FF
	0044	320E00	RONE:	STA	ROWECR	Set write-exact-CP/M-records flag
	00A7			XCHG	NUMEON	;DE = user record length
	8A00	240300		LHLD	ROURN	Get user record number
	OOAB	CDB801		CALL	MLDL	DE,HL = HL * DE
						<pre>pDE,HL = user-record byte offset in file</pre>
	OOAE	D5		PUSH PUSH	D H	Save user-record byte offset
	0080	EJ 70		MOV	A,L	Get LS byte of product
	00B1			ANI	7FH	; Isolate byte of frontet
	00B3 -			MOV	C,A	CP/M record
	0084	0600 210F00		MVI	B, 0	;Make into word value
	0089			LXI DAD	H, ROBUF B	Get base address of local buffer
		220900		SHLD	ROFRP	;HL -> Start of fragment in buffer ;Save fragment pointer
				U.ILD	10.11	
						it length that could reside in
				fremaind	ler of CP/M recor	d, based on the offset in the
				CP/M re	cord where the f	ragment starts.
(	OOBD 4	17		MOV	B,A	Take copy of offset in CP/M record
	OOBE (	3E80		MVI	A,128	CP/M record size
	0000 9					;Compute 128 - offset
(	0001 3	320B00		STA	ROFRL	;Assume this is the fragment length
				If the	user record leng	th is less than the assumed
						t in place of the result above
	00C4 4					Get copy of assume frag. length
		BA0600			ROURL+1	:Get MS byte of user record length
	0008 1	37 20600		ORA JNZ	A DOELOK	;If NZ, rec. len. must be > 128 ;So fragment length is OK
		20000 3A0300				
	OOCF E					Still a chance that rec. len.
		20600			-	;NC if user rec. len. => frag. len.
		320B00				;User rec. len. < frag. len. so
						; reset fragment length to smaller
			ROFLOK:			
(	0006 3	BAOEOO		LDA	ROWECR	;Get exact CP/M record flag
	0009 4	17				for ANDing with READ flag
		3A0000		LDA		;Get read operation flag
(	oodd a	2F		CMA		;Invert so NZ when writing
						· · · · · · · · · · · · · · · · · · ·

Figure 5-26. (Continued)

00DE A0	ANA	B	;Form logical AND
00DF 320E00	STA	ROWECR	;Save back in flag
			length byte offset within the file
	;to	divide by 128 and	user record. Shift 7 places right I get the CP/M record number for
	;th	e start of the use	er record.
00E2 E1	POP	н	;Recover user rec. byte offset
00E3 D1 00E4 0E07	POP	D C,7	;Count for shift right
0024 0207		0,,	yount for shirt right
00E6 CDF101	ROS: CALI		;DE,HL = DE,HL / 2
00E9 0D	DCR	C	
00EA C2E600	JNZ	ROS	
OOED 7A	MOV	A, D	;Error if DE still NZ after
OOEE B3 OOEF C2ACO1	ORA JNZ	E ROERO	; division by 128.
		1102/10	
OOF2 EB	XCH	3	;Set CP/M record number in FCB ;DE = CP/M record number
00F3 2A0100	LHL	D ROFCB	;Get pointer to FCB
00F6 012100 00F9 09		B, FCBE\$RANREC B	) ;Offset of random record no. in FCB ;HL -> ran. rec. no. in FCB
00FA 220C00	SHLI	D RORNP	;Save record number pointer
00FD 73 00FE 23	MOV INX	M,E H	;Store LS byte
00FF 72	MOV	M, D	;Store MS byte
0100 0E1A	MVI	C, B\$SETDMA	;Set DMA address to local buffer
0102 110F00	LXI	D, ROBUF	
0105 CD0500	CALI	BDOS	
0108 3A0E00	LDA	ROWECR	Bypass preread if exact sector write
010B B7 010C C21F01	ORA JNZ	A ROMNF	
010F 2A0100 0112 EB	LHLI		;Get pointer to FCB ;DE -> FCB
0113 0E21	MVI	C, B\$READRAN	Read random function
0115 CD0500	CALI	BDOS	
0118 FE05	CPI	5	;Check if error code < 5
011A DCAF01	CC	ROCIE	;Yes, check if ignorable error ; (i.e. error reading unwritten part
			; of file for write operation preread)
011D B7 011E C0	ORA RNZ	Α	;Check if error ;Yes
5112 00			
011F 2A0700	ROMNF:	D ROUB	;Move next fragment ;Get pointer to user buffer
0122 EB	XCH	3	;DE -> user buffer
0123 2A0900 0126 3A0B00	LHL	D ROFRP ROFRL	;HL -> start of user rec. in local buffer ;Get fragment length
0129 4F	MOV	C, A	Ready for MOVE
012A 3A0000	LDA	ROREAD	;Check if reading
012D B7	ORA	Α	
012E C23201 0131 EB	JNZ XCH	RORD1	;Yes, so leave DE, HL unchanged ;Writing, so swap source and destination
			;DE -> start of user rec. in local buffer
			;HL -> user buffer
	RORD1:	NOUT	Deadline foremark land. A man hoff of
0132 CDFE01	CALI	MOVE	;Reading - fragment local -> user buffer ;Writing - fragment user -> local buffer
0135 3A0000	LDA	ROREAD	;Check if writing
0138 B7 0139 CA3D01	ORA JZ	A ROWR1	;Writing, so leave HL -> user buffer
013C EB	XCH		;HL -> next byte in user buffer
	ROWR1:		
013D 220700	SHLI	D ROUB ROREAD	;Save updated user buffer pointer
0140 3A0000	LDA	RUREAD	;Check if reading

Figure 5-26. (Continued)

0144 C25001 0147 0E28 0149 2A0100 014C EB 014D CD0500		JNZ	RORD3	;Yes, bypass write code
0149 2A0100 014C EB				
DIAC EB		MVI	C, B\$WRITERANZ	;Write random
			ROFCB	Get address of FCB
		XCHG CALL	BDOS	;DE -> FCB
	RORD3:	<pre>; If ned ; more 0 ; the st ; length ; length ; the ne ; user n</pre>	essary (because CP/M records wil art of the frag 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
		;128.		
150 2A0500		LHLD	ROURL	;Get residual user rec. length
153 3A0B00		LDA	ROFRL.	;Get fragment length just moved
156 5F		MOV	E,A	;Make into a word value
157 1600		MVI	D,O	
159 CDEA01		CALL	SUBHL	;Compute ROURL - ROFRL
15C 7C		MOV	A, H	;Check if result O
15D B5		ORA	L	
015E C8		RZ		;Return when complete USER
				; record has been transferred
015F 220500		SHLD	ROURL	;Save downdated residual rec. length
)162 4D		MOV	C,L	Assume residual length < 128
163 118000		LXI	D,128	;Check if residual length is < 128
166 CDEA01		CALL	SUBHL	;HL = HL - DE
169 FA6E01		JM	ROLT128	negative if < 128
016C 0E80		MVI	C,128	;=> 128, so set frag.length to 128
	ROLT128			
016E 79	NOC 1128	MOV	A,C	
016F 320B00		STA	ROFRL	Fragment length now is either 128 ; if more than 128 bytes left to input ; in user record, or just the right ; number of bytes (< 128) to complete ; the user record
172 210F00		1 11	H. ROBUF	; the user record. ;All subsequent CP/M records will start
0172 210F00		LXI SHLD	ROFRP	;All subsequent CP/M records will start ; at beginning of buffer
11/3 220900		SHLU	NUMAN	t er neðruggið og priser
				;Update random record number in FCB
178 2A0C00		LHLD	RORNP	;HL -> random record number in user FCB
17B 5E		MOV	E,M	;Increment the random record number
17C 23		INX	H	;HL -> MS byte of record number
17D 56		MOV	D, M	;Get MS byte
017E 13 017F 7A		INX MOV	D A, D	;Update record number itself
				;Check if record now 0
)180 B3 )181 C28701		ORA JNZ	E ROSRN	No, so save record number
184 3E06		MVI	A.6	;NO, so save record number ;Indicate "seek past end of disk"
186 C9		RET		Return to user
	ROSRN:			
187 72		MOV	M, D	;Save record number
188 2B		DCX	н	;HL -> LS byte
189 73		MOV	M,E	,
18A 3A0E00			ROWECR	;If writing, check if preread required ;Check if exact CP/M record write
180 B7		ORA	A	JUNECK IN WRACT UP/IN NECOLO WILLS
19E C21F01		JNZ	ROMNF	;Yes, go move next fragment
CALL OF LEVE				tient An House Have Laganeire
191 3A0000		LDA	ROREAD	<pre>;If reading; perform read unconditionally</pre>
0194 B7		ORA	Α	
0195 C2A001		JNZ	RORD2	
198 3A0800		LDA	ROFRL	For writes, bypass preread if
198 FE80		CPI	128	; whole CP/M-record is to be overwritten
19D CAIFOI		JZ	ROMNF	; (fragment length = 128)
	RORD2:	MI 17	0.040510011	
01A0 0E21		MVI LHLD	C,B\$READRAN ROFCB	;Read the next CP/M record ; in sequence

Figure 5-26. (Continued)

01A5 EB		XCHG	•	;DE -> FCB
01A6 CD0500 01A9 C31F01		CALL JMP	18DOS ROMNF	;Go back to move next fragment
	ROERO:			Error because user record number
	ROENUT			; # User record length / 128 gives
				; a CP/M record number > 65535.
01AC 3E04 01AE C9		MVI RET	A,4	;Indicate "attempt to read unwritten ; 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		ORA	A	Scheck If Feed Operation
0184 78		MOV	Α, Β	Restore original error code but Restore flags unchanged
0185 CO		RNZ		Return if reading
0186 AF		XRA	A	Fake "no error" indicator
01B7 C9		RET		
	return	ed in D	E,HL.	ative ADD with product
	;Entry	paramet	ers ultiplicand	
	;		ultiplier	
	;Exit p	aramete	rs	
	;	DE,HL	= product	
	,	DE = m	ultiplier	
	MLDL:			
0188 010000		LXI	B,0 B	Put 0 on top of stack
01BB C5		PUSH	в	; to act as MS byte of product ;Check if either multiplicand
				; or multiplier is O
OIBC 7C		MOV	A.H	
01BD 85 01BE CAE501		ORA JZ	MLDLZ	;Yes, fake product
01C1 7A		MOV	A,D	,
01C2 B3		ORA JZ	E MLDLZ	:Yes, fake product
01C3 CAE501		52	THE DE Z	
				;This routine will be faster if ; the smaller value is in DE
01C6 7A		MOV	A, D	; the smaller value is in DC ;Get MS byte of current DE value
01C7 BC		CMP	н	;Check which is smaller
01C8 DACCO1		JC XCHG	MLDLNX	;C set if D < H, so no exchange
O1CB EB	MLDLNX:			
01CC 42		MOV	B,D	;BC = multiplier
O1CD 4B		MOV	C,E	
01CE 54		MOV	D,H	;DE = HL = multiplicand
01CF 50		MOV	E,L	
01D0 0B		DCX	В	;Adjust count as
				; 1 * multiplicand = multiplicand
01D1 78	MLDLA	MOV	A, B	;ADD loop ;Check if all iterations completed
01D2 B1		ORA	С	
01D3 CAE801		JZ	MLDLX	;Yes, exit
01D6 19		DAD XTHL	D	;HL = multiplicand + multiplicand ;HL = MS bytes of result, TOS = part prod.
01D7 E3 01D8 7D		MOV	A,L	;Get LS byte of top half of product
01D9 CE00		ACI	0	;Add one if carry set
01DB 6F 01DC 7C		MOV MOV	L,A A,H	;Replace ;Repeat for MS byte
01DD CEOO		ACI	0	
01DF 67		MOV	H, A	
01E0 E3		XTHL DCX	в	;Countdown on multiplier - 1
01E1 0B 01E2 C3D101		JMP	MLDLA	;Loop back until all ADDs done

Figure 5-26. (Continued)

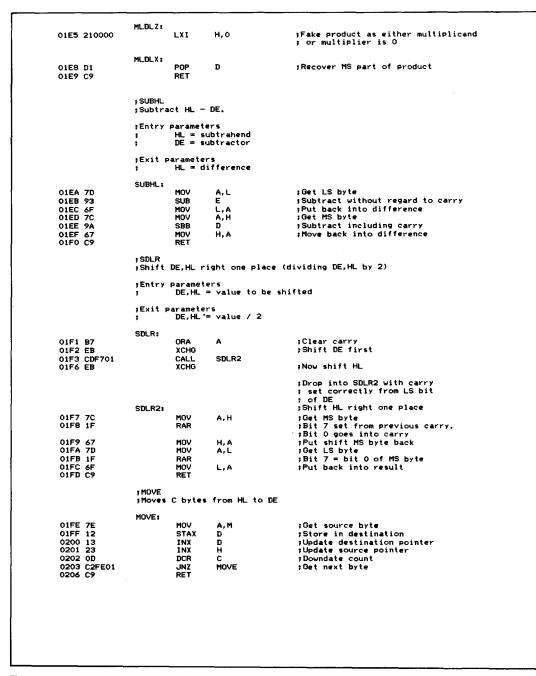


Figure 5-26. (Continued)

#### 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	00	FCB\$DISK:	DB	0	:Search on default disk drive
	4649404546		DB		
					1E' ;File name
	545950	FCB\$TYP:	DB		;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	0	Records used in this extent
0010	0000000000	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
		FCB\$RANREC:	DW	ŏ	Random rec. to read/write
0023	00	FCB\$RANRECO:	DB	0	;Random rec. overflow byte (MS)
0024	0523	MVI	C.B\$GETF	517	:Function code
	110000	LXI	D.FCB		:DE -> file control block
					JUE -> TILE CONTROL DIOCK
0029	CD0500	CALL	BDOS		
002C	2A2100	LHLD	FCB\$RANF	REC	;Get random record number
					: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 0005		B\$SETRAN	REC	EQU EQU	36 5	;Set Rand ;BDOS ent	lom Record	Number	
0005	-	8003		EGO	5	, , , , , , , , , , , , , , , , , , , ,	e porte		
		FCB:				;File con	trol block	k	
0000	00	FCB\$DISk	(:	DB	0	;Search o	n default	disk drive	
0001	4649404548	FCB\$NAME	Ξ.	DB	'FILENAM	É′ τ	File name		
0009	545950	FCB#TYP:	1	DB	TYP'	;File typ	6		
0000	00	FCB\$EXTE	ENT:	DB	0	;Extent			
	0000	FCB\$RES\	/:	DB	0,0	Reserved	i for CP/M		
000F	00	FCB\$RECU	JSED:	DB	0	Records	used in th	his extent	
	00000000000	FCB\$ABUS	SED:	DB	0.0.0.0,	0,0,0,0 ;	Allocatio	n blocks use	1
	00000000000			DB	0,0,0,0,	0,0,0,0			
0020	00	FCB\$SEQ	REC:	DB	0	:Sequenti	al rec. to	o read/write	
	0000	FCB\$RANE	REC:	DW	0	Random r	ec. to re-	ad/write	
0023	00	FCB\$RANF	RECO:	DB	0	;Random r	rec. overf	low byte (MS	)
						: file	e opened a	nd read	
								ntially	
0024	0F24		MVI	C. B\$SETF		:Function	o code		
	110000		LXI	D.FCB			le contro	l block	
	CD0500		CALL	BDOS					
	2A2100		LHLD	FCB\$RAN	REC	:Get rand	iom record	number	
							dom recor		
							prresponds		
							ial progr		
						; the fil			

**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 ... 2 1 0
                                                        0 N ... C B A
                                               :Disk P
0000 110200
                       IXT
                               B,0000$0000$0000$0010B ;Reset drive B:
0003 0E25
                       MVI
                               C. B$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	=	<b>B\$WRITERANZ</b>	EQU	40	;Write Random with Zero-Fill
0005	=	BDOS	EQU	5	;BDOS entry point
		FCB:			File control block
0000	00	FCB\$DISK:	DB	0	Search on default disk drive
0001	4649404548	FCB\$NAME:	DB	FILEN	AME' ;File name
0009	545950	FCB\$TYP:	DB	TYP'	File type
0000	00	FCB\$EXTENT:	DB	0	Extent
	0000	FCB\$RESV:	DB	0.0	Reserved for CP/M
000F		FCB\$RECUSED:	DB		Records used in this extent
		FCB\$ABUSED:	DB		0,0,0,0,0 ;Allocation blocks used
	0000000000		DB		0,0,0,0,0
0020		FCB\$SEQREC:	DB	0	Sequential rec. to read/write
	0000	FCB\$RANREC:	DW	ò	Random rec. to read/write
0023		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	LHLD	RANREC	NO	Get random record number
	222100	SHLD	FCB\$RA	NREC	Set up file control block
0020	0E28	MVI	C, B\$WF	RITERANZ	Function code
002E	110000	LXI	D.FCB		:DE -> file control block
	CB0500	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

# 6

# 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.

<ul> <li>Xx00H JMP BOOT ;"Cold" (first time) bootstrap</li> <li>Xx03H JMP WBOOT ;"Warm" bootstrap</li> <li>Xx06H JMP CONST ;Console input status</li> <li>Xx09H JMP CONUT ;Console output</li> <li>Xx0FH JMP CONUT ;Console output</li> <li>Xx0FH JMP LIST ;List output</li> <li>Xx12H JMP PUNCH ;"Punch" output</li> <li>Xx18H JMP READER ;"Reader" input</li> <li>Xx18H JMP SELDSK ;Select logical disk</li> <li>Xx18H JMP SETTRK ;Set track number</li> <li>Xx21H JMP SETDMA ;Set DNA alvess</li> <li>Xx27H JMP READ ;Read (128-byte) sector</li> <li>Xx22H JMP LIST ;List device output status</li> </ul>				
XX03HJMPWB00T; "Warm" bootstrapXX06HJMPCONST; Console input statusXX09HJMPCONIN; Console inputXX09HJMPCONUT; Console outputXX0FHJMPCONOUT; Console outputXX0FHJMPLIST; List outputXX12HJMPPUNCH; "Peunch" outputXX18HJMPREADER; "Reader" inputXX18HJMPSELDSK; Select logical diskXX18HJMPSETTRK; Set track numberXX21HJMPSETTRK; Set track numberXX24HJMPSETDMA; Set DMA addressXX27HJMPREAD; Read (128-byte) sectorXX20HJMPLISTST; List device output statusXX30HJMPSECTRAN; Sector translate	X X OOH	, IMP	BOOT	:"Cold" (first time) bootstrag
xx06HJMPCONST; Console inputstatusxx09HJMPCONIN; Console inputxx0FHJMPCONUT; Console outputxx0FHJMPLIST; List outputxx12HJMPPUNCH; "Punch" outputxx13HJMPREADER; "Reader" inputxx18HJMPSELDSK; Select logical diskxx12HJMPSETTRK; Set track numberxx2HJMPSETDMA; Set DNAxx2HJMPREAD; Read (128-byte) sectorxx2HJMPREAD; List device output statusxx2DHJMPLISTST; List device output statusxx3OHJMPSECTRAN; Sector translate				
xx09HJMPCONIN; Console inputxx0CHJMPCONOUT; Console outputxx0FHJMPLIST; List outputxx12HJMPPUNCH; "Punch" outputxx15HJMPREADER; "Reader" inputxx18HJMPHOME; Home disk heads (to track 0)xx18HJMPSELDSK; Select logical diskxx12HJMPSETTRK; Set track numberxx21HJMPSETDMA; Set of track numberxx24HJMPSETDMA; Read (128-Byte) sectorxx2AHJMPWRITE; Write (128-byte) sectorxx2DHJMPLISTST; List device output statusxx30HJMPSECTRAN; Sector translate				
XXOCHJMPCONOUT; Console outputXXOFHJMPLIST; List outputXX12HJMPPUNCH; "Punch" outputXX13HJMPREADER; "Reader" inputXX18HJMPHOME; Home disk heads (to track 0)XX18HJMPSELDSK; Select logical diskXX18HJMPSETTRK; Set track numberXX21HJMPSETTRK; Set track numberXX21HJMPSETTRA; Set DNA dressXX27HJMPREAD; Read (128-byte) sectorXX2AHJMPLISTST; List device output statusXX30HJMPSECTRAN; Sector translate				
xx0FHJMPLIST;List outputxx12HJMPPUNCH;"Punch" outputxx15HJMPREADER; "Reader" inputxx18HJMPHOME; Home disk heads (to track 0)xx18HJMPSELDSK; Select logical diskxx1EHJMPSETTRK; Set track numberxx21HJMPSETTRK; Set track numberxx22HJMPSETDMA; Set DMAxx27HJMPREAD; Read (128-byte) sectorxx2AHJMPLISTST; List device output statusxx20HJMPSECTRAN; Sector translate				
xx12HJMPPUNCH; "Punch" outputxx15HJMPREADER; "Reader" inputxx18HJMPHOME; Home disk heads (to track 0)xx18HJMPSELDSK; Select logical diskxx1EHJMPSETTRK; Set track numberxx21HJMPSETSEC; Set sector numberxx24HJMPSETDMA; Set DMA addressxx27HJMPREAD; Read (128-byte) sectorxx2AHJMPWRITE; Write (128-byte) sectorxx2DHJMPLISTST; List device output statusxx30HJMPSECTRAN; Sector translate				
xx15HJMPREADER; "Reader" inputxx18HJMPHOME; Home disk heads (to track 0)xx18HJMPSELDSK; Select logical diskxx1EHJMPSETTRK; Set track numberxx21HJMPSETSEC; Set sector numberxx24HJMPSETDMA; Set DMA addressxx27HJMPREAD; Read (128-byte) sectorxx2AHJMPLISTST; List device output statusxx30HJMPSECTRAN; Sector translate				
xx18HJMPHOME;Home disk heads (to track 0)xx18HJMPSELDSK;Select logical diskxx18HJMPSETTRK;Set track numberxx21HJMPSETSEC;Set sector numberxx24HJMPSETDMA;Set DMA addressxx27HJMPREAD;Read (128-byte) sectorxx2AHJMPRITEiArite (128-byte) sectorxx2AHJMPLISTST;List device output statusxx3OHJMPSECTRAN;Sector translate	xx12H	JMP	PUNCH	
xx1BHJMPSELDSK;Select logical diskxx1EHJMPSETTRK;Set track numberxx21HJMPSETSEC;Set sector numberxx24HJMPSETDMA;Set DMA addressxx27HJMPREAD;Read (128-Byte) sectorxx2AHJMPWRITEimrite (128-byte) sectorxx2DHJMPLISTST;List device output statusxx30HJMPSECTRAN;Sector translate	xx15H	JMP	READER	;"Reader" input
xx1EHJMPSETTRK; Settrack numberxx21HJMPSETSEC; Setsector numberxx24HJMPSETDMA; SetJMA addressxx27HJMPREAD; Read(128-Byte) sectorxx2AHJMPWRITEi Write(128-byte) sectorxx2DHJMPLISTST; Listdevice outputxx30HJMPSECTRAN; Sectortranslate	xx18H	JMP	HOME	;Home disk heads (to track O)
xx21H JMP SETSEC ;Set sector number xx24H JMP SETDMA ;Set DMA address xx27H JMP READ ;Read (128-byte) sector xx2AH JMP WRITE iArite (128-byte) sector xx2DH JMP LISTST ;List device output status xx3OH JMP SECTRAN ;Sector translate	xx1BH	JMP	SELDSK	;Select logical disk
xx24H JMP SETDMA ;Set DMA address xx27H JMP READ ;Read (128-byte) sector xx2AH JMP WRITE ;Arite (128-byte) sector xx2DH JMP LISTST ;List device output status xx3OH JMP SECTRAN ;Sector translate	xx1EH	JMP	SETTRK	;Set track number
xx27H JMP READ ;Read (128-Byte) sector xx2AH JMP WRITE ;Arite (128-byte) sector xx2DH JMP LISTST ;List device output status xx3OH JMP SECTRAN ;Sector translate	xx21H	JMP	SETSEC	;Set sector number
xx2AH JMP WRITE Prite (128-byte) sector xx2DH JMP LISTST ;List device output status xx3OH JMP SECTRAN ;Sector translate	xx24H	JMP	SETDMA	;Set DNA address
xx2DH JMP LISTST ;List device output status xx30H JMP SECTRAN ;Sector translate	xx27H	JMP	READ	Read (128-byte) sector
xx30H JMP SECTRAN ;Sector translate	xx2AH	JMP	WRITE	Write (128-byte) sector
	xx2DH	JMP	LISTST	;List device output status
	xx30H	JMP	SECTRAN	Sector translate
	××30H	JMP	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

#### **152** The CP/M Programmer's Handbook

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:

BDOS Err on X: Bad Sector

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	with BIOS subroutine		
0003 =	WBOOT	EQU	03Н	:Warm boot		
0006 =	CONST	EQU	06H	Console status		
0009 =	CONIN	EQU	09H	Console input		
= 3000	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		
				And further "private" blos codes nere		
	:	BIOS				
			brouting	transfers control to the appropriate		
	,			OS Jump Vector, based on a code number		
	1			the L register.		
	;	Entry parameters				
	7 7 7		address the jump	(which is in fact the page-relative of the correct JMP instruction within p vector)		
	;;	All othe		ters are preserved and handed over to S routine intact.		
	;	Exit par	rameters			

	;	when th	e BIOS i	pes not CALL the BIOS routine, therefore routine RETurns, it will do so directly e's caller.
	;	Calling	sequenc	ce de la companya de
	7 7 7		MVI CALL	L,Code‡Number BIOS
	, BIOS:			
0000 F5		PUSH	PSW	;Save user's A register
0001 3A0200		LDA	0002H	;Get BIOS JMP vector page from ; warm boot JMP
0004 67		MOV	H,A	;HL -> BIOS JMP vector entry
0005 F1		POP	PSW	Recover user's A register
0006 E9		PCHL		;Transfer control into the BIOS routine

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 Number	; Figure 6-4.	
0002 0003	: ************************************	
0003	° ************************************	
0005	/* Simple BIOS Listing *	
0006	1* SIMPLE BIOS LIS(ING *	
0007	* <b>***</b> ********************************	
0008	]	
0009	3	
0010 3030 =	, VERSION EQU '00' :Equates used in the sign on me	
0011 3730 =	MONTH EQU '07'	
0012 3531 =	DAY EQU (15)	
0013 3238 =	YEAR EQU '82'	
0013 3238 -		
0015	。我说我就我就想跟我说说这家就说我就是我就能能能能能能能能能能能能能能能能能能能能能能能能能能能能能能能能能	
0016	;*	*
0017	* This BIOS is for a computer system with the following	ê
0018	;* hardware configuration :	÷
0019	ix	÷.
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	is no baud rate generator or USART programming)	¥
0025	i* ~ A serial port. used for both list and "reader"/	×
0026	* "punch" devices. The serial port chip is an	*
0027	1* 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	These are used as logical disks C: and D:.	¥
0033	3 🛪	×
0034	3* Two intelligent disk controllers are used, one for	×
0035	<pre>## each diskette type. These controllers access memory</pre>	*
0036	<pre># directly, both to read the details of the</pre>	×
0037	sx operations they are to perform and also to read	×
0038	3* and write data from and to the diskettes.	×
0039	3 <del>4</del>	×
0040	3 <del>X</del>	×
0041	· · · · · · · · · · · · · · · · · · ·	****
0042		
0043	3	
0044	Fquates for defining memory size and the base address and	
0045	; length of the system components.	

# **160** The CP/M Programmer's Handbook

0046 0047	0040	-	; Memory‡Siz	e	EQU	64 ;Number of Kbytes of RAM
0048			;			
0049			; The BIC	S Length	MUST De	determined by inspection.
0050			t Comment		URG BIU	S\$Entry line below by changing the first
0052			the BIO	er lo a Sat ìoc	ation 0.	n. (This will make the Assembler start ) Then assemble the BIOS and round up to
0053						dress displayed on the console at the end
0054				assembly		
0055			;			
0056 0057	0900	=	BIOS\$Lengt	h	EQU	0900H
0058	0800	-	CCP\$Length	EQU	0800H	;Constant
0059	0E00	=	BD0S\$Lengt	h	EQU	OEOOH ;Constant
0060			;			
0061	0008	-	Overall\$Le	ngth	EQU	((CCP\$Length + BDOS\$Length + BIOS\$Length) / 1024) + :
0063	E000	-	CCP\$Entry	EQU	(Memory	\$Size - Overall\$Length) * 1024
0064	E806		BDOS\$Entry	EQU	CCP\$Ent	ry + CCP\$Length + 6
0065	F600	=	BIOS\$Entry	EQU	CCP\$Ent	ry + CCP\$Length + BDOS\$Length
0066			*			
0067 0068			7			
0069			,			
0070	F600		ORG	BIOS\$En	try	Assemble code at BIOS address
0071			,			
0072				mp vector		und to the second to estimate the
0073 0074						rred to the appropriate entry point S, both of which compute the relative
0075			address	of the i	BIOS Jum	p vector in order to locate it.
0076						also make direct BIOS calls transferring
0077						OH, where xx is the value in location
0078			; 0002H.			
0079			;		<b>.</b>	
0080	F600	C3F9F6	JMP Uswatte	BOOT		oot entered from CP/M bootstrap loader lled so that the initialization code can
0081			Warm\$Boot\$	Entrys		the warm boot entry address down in location
0083						H and 0002H of the base page
0084	F603	C329FE	JMP	WBOOT	;Warm b	oot entered by jumping to location 0000H.
0085						ads the CCP which could have been
0086						written by previous program in transient
0087	E 404	C362F8	JMP	CONST		ram area e status returns A = OFFH if there is a
0089	FOUG	CJOZFO	UNF	CONST		ole keyboard character waiting
0090	F609	C378F8	JMP	CONIN		e input returns the next console keyboard
0091						acter in A
0092	F60C	C386F8	JMP	CONOUT		e output outputs the character in C to
0093						console device
0094	F60F	C3ACF8	JMP	LIST		utput outputs the character in C to the device
0096	F612	C3BCF8	JMP	PUNCH		output outputs the character in C to the
0097						cal punch device
0098	F615	C3CDF8	JMP	READER		input returns the next input character from
0099	E410	C30255	JMP	HOME		logical reader device in A the currently selected disk to track O
0100 0101		C3D3FB C32BFB	JMP	SELDSK		s the disk drive specified in register C and
0102	. 515	COLDED	011		; retu	rns the address of the disk parameter header
0103	F61E	C358FB	JMP	SETTRK	;Sets t	he track for the next read or write operation
0104						the BC register pair
0105	F621	C35EFB	JMP	SETSEC		he sector for the next read or write operation
0106	5404	C365FB	JMP	SETDMA		the A register he direct memory address (disk read/write)
0108	roz4	LJOJFB	Unit	SETURA	addru	ess for the next read or write operation
0109						the DE register pair
0110	F627	C3FBFB	JMP	READ	;Reads	the previously specified track and sector from
0111					; the	selected disk into the DMA address
0112	F62A	C315FC	JMP	WRITE		the previously specified track and sector onto
0113	E/05	000450	1945	LISTET		selected disk from the DMA address s A = OFFH if the list device can accept
	F 620	C394F8	JMP	LISTST		s A = OFFH it the list device can accept her output character
0114		COOPER	JMP	SECTRAN		ates a logical sector into a physical one
0114 0115	F630	LJUDEB				
0114 0115 0116	F630	L3CDFB	;			
0114 0115 0116 0117 0118	F630	C3CDFB	;			
	F630	C3CDFB	; ;			ation code is only needed once.

Figure 6-4. (Continued)

,											
0121									executed.		
0122											
0123							only being overwritten by data from				
0124				; the disk once the initialization procedure is complete.							
0126			;	7							
0127											
0128				normall	y. Then	the valu	e of the	locatio	on counter following		
0129			3	the buf	fer is n	ot <b>ed.</b> Th	en, usin	g an ORG	(ORiGin) statement, the		
0130									start of the buffer		
0131									en normally. atement is used to		
0133									atement is used to after the buffer had		
0134				been de			4				
0135											
0136			3.				<b></b>		<b>_</b>		
0137	0200	=	Phy	sical#S	ector\$Si	ze	EQU	512	This is the actual sector size		
0138									e 5 1/4" mini-floppy diskettes.		
0139									diskettes use 128-byte sectors. The physical disk buffer for the		
0141									diskettes		
0142	F633		Dis	k\$buffe		DS	Physica	1\$Sector	\$Size		
0143			;								
0144									he location counter		
	F833	-		er\$Disk	\$Buffer	EQU	5	;\$ = Cu	rrent value of location counter		
0146	F633		,			ORG	Disk\$Bu	11az	:Wind the location counter back		
0148	F033					ORO	015K78U	) 1 <b>9 F</b>	;wind the location counter dack		
0149				tializ#	Stream:	;This s	tream of	data is	used by the		
0150									It has the following		
0151						format	:		· · · · · · · · · · · · · · · · · · ·		
0152						;		<b>_</b> .			
0153						3	DB		mber to be initialized		
0154						:	DB DB		of bytes to be output		
0155							:	xx, xx, X	x,xx data to be output		
0157						;					
0158						1	DB	Port nu	mber of OOH terminator		
0159						;					
0160							On this	machine	, the console port does		
0161						!			initialized. This has		
0162						;	ericeda	Deen do	ne by the PROM bootstrap code.		
0164						,	;Initia	lize the	8251A USART used for		
0165									communications devices.		
0166	F633			DB	Communi	cation\$S	tatus\$Po		;Port number		
	F634			DB	6				Number of bytes		
	F635			DB	0				to be programmed by		
0169	F636	00		DB DB	0		; send	ing dumm	ny data out to it		
0170	F637 F638			DB	0100\$00	108	. Peret	and rai-	e data terminal ready		
	F639			DB	01\$10\$1				e data terminal ready parity, 8 bits per character		
0173							t baud	rate di	vide factor of 16.		
0174	F63A	25	1	DB	0010\$01	01B			to send, and enable		
0175									receive.		
0176			;								
0177							;Initia	lize the	8253 programmable interval		
0178							; time ; the	r used t 8251A US	o generate the baud rate for		
	F63B	DF	1	DB	Communi	cation\$B	aud\$Mode		Port number		
0181	F63C	01		DB	1				Number of bytes		
	F63D	B6	1	DB	10\$11\$0	11\$0B	;Select	counter	2, load LS byte first,		
0183							; Mode	3 (for	baud rates), binary count.		
0184	F/0-	<b>DC</b>	;						<b>-</b> · · ·		
0185	F63E F63F			DB DB	Communie 2	cation\$B	aud\$Rate		Port number		
0187		3800		DB	2 0038H		1200 b	aud (hee	;Number of bytes ed on 16X divide-down selected		
0188	, 540				2000m			auo (bas he 8251A			
0189			;				,	02018			
0190	F642	00	· 1	DB	0		;Port n	umber of	O terminates		
0191			,								
0192			;								
0193			3	Equates	for the	sign-on	message				
	000D	=	; CR I	FOIL	ODH		· Carris	ge retur	n		
""	0000	-	UR I		2011		y Carrie	ye retur			

Figure 6-4. (Continued)

0196	000A	=	LF	EQU	OAH	;Line feed
0197 0198			;	anon\$Mes		·M-ft
0199	E443	43502F4D20	~ 319	DR		;Main sign-on message
0200	5440	3030		DW	1CP/M 2.2.1	<b>A A A A</b>
0201	F64E	3030		DB	VERSION	;Current version number
	F64F	20				
0202	F04F	3037		DW	MONTH	;Current date
0203	F651	21		DB	11	
0204	F652			DW	DAY	
	F654			DB	111	
	F655			DW	YEAR	
0207		ODOAOA		DB	CR,LF,LF	
		53696D7060		DB	'Simple BIOS',C	
		4469736B20		DB	'Disk configurat	tion :',CR,LF,LF
		2020202020		DB	A: 0.35 MI	byte 5" Floppy',CR,LF
0211		2020202020		DB	B: 0.35 Mt	byte 5" Floppy',CR,LF,LF
0212		2020202020		DB	C: 0.24 Mit	byte 8" Floppy',CR,LF
0213	F6DA	2020202020	)	DB	D: 0.24 Mt	byte 8" Floppy/,CR,LF
0214			;			
0215	F6F8	00		DB	0	
0216			;			
0217	0004	=	Def	ault\$Di	sk EQU	0004H ;Default disk in base page
0218			;			
0219			BOO	IT:	;Entered direct]	ly from the BIOS JMP vector.
0220						transferred here by the CP/M
0221					; bootstrap loa	
0222						tion state of the computer system
0223					; will be deter	mined by the
0224						ap and the CP/M loader setup.
0225					1	
0226					•	;Initialize system.
0227						;This routine uses the Initialize\$Stream
0228						; declared above.
	F6F9	F3		DI		;Disable interrupts to prevent any
0230		••				; side effects during initialization.
0231	FAFA	2133F6		LXI	H, Initialize\$Str	
0232		2100.0	;			eam , ne / Data stream
0233				tializes		
0234	F6FD	7E		MOV	A,M	;Get port number
0235	F6FE			ORA	A	;If OOH, then initialization complete
0236		CA13F7		JZ	Initialize\$Compl	
0237		320AF7		STA	Initialize#Port	;Set up OUT instruction
0238	F705			INX	H	;HL -> Count of number of bytes to output
0239	F706			MOV	C, M	;Get byte count
0240	.,				et ii	yoer byte count
0241			Ini	tializes	Next\$Byte:	
0242	F707	23			н	;HL -> Next data byte
0243	F708	7F		MOV	A, M	;Get next data byte
0244	F709			DB	OUT	;Output to correct port
0245				tializes		,,
0246	F70A	00		DB	0	;<- Set above
0247	F70B			DCR	č	;Count down
0248		C207F7		JNZ	Initialize\$Next\$	
0249	F70F			INX	H	<pre>;HL -&gt; Next port number</pre>
0250		C3FDF6		JMP	••	Go back for next port initialization
0251			:			you wear two many pers anasamasamasawa
0252			Íni	tialized	Complete:	
0253			;			
0254			•			
0255	F713	3E01		MVI	A,00\$00\$00\$01B	;Set IOBYTE to indicate terminal
0256		320300		STA	IOBYTE	; is to act as console
0257						, is to act as compose
0258	F719	2143F6		LXI	H, Signon\$Message	Display sign-on message on console
0259		CD33F8		CALL	Display\$Message	, , , , , , , , , , , , , , , , , , ,
0260			;			
0261			,			
0262	F71E	AF		XRA	A	Set default disk drive to A:
0263	F71F	320400		STA	n Default\$Disk	ywas waranas waan wrate sw Ma
0264	F722	FR		FI	2	:Interrupts can now be enabled
0265						yinterrapts can now be enabled
	F727	C340F8	,	JMP	Enter\$CPM	Complete initialization and enter
0266	. / 23	0.04010			Circler #Criti	; CP/M by going to the Console Command
0268						; Processor.
0269			1	End of	old boot initial	insting odd
0270					old boot initial	Tracton code
02/1			\$			
·						

Figure 6-4. (Continued)

0272	F833	ORG	After\$Disk\$Buff	er ;Reset location counter
0273		:		
0275		; Display\$Me	ssade: :Disola	lys the specified message on the console.
0276			jOn ent	ry, HL points to a stream of bytes to be
0277			; out;	out. A OOH-byte terminates the message.
0278	F833 7E F834 B7	MOV	A, M	;Get next message byte ;Check if terminator
0279	F834 B/ F835 C8	ORA RZ	A	JUNECK 17 Terminator JYes, return to caller
0281	F836 4F	MOV	C,A	Prepare for output
0282		PUSH	Ĥ	Save message pointer
0283	F838 CD86F8	CALL	CONCUT	;Go to main console output routine
0284	F838 E1	POP	н	Recover message pointer
0285	F83C 23 F83D C333F8	INX JMP	H Display#Massage	<pre>&gt;Move to next byte of message &gt; Loop until complete message output</pre>
0287	F630 C333F6	) )	nishia) suszade	s stoop dittit complete message output
0288		,		
0289		Enter#CPM:	<b>;</b> This routine i	s entered either from the cold or warm
0290				It sets up the JMP instructions in the
0291			<pre>.; base page, a ; input/output</pre>	and also sets the high-level disk driver's t address (also known as the DMA address).
0293		,	i input/outpu	t dudress (diso known as the DhA dudress).
0294	F840 3EC3	MVI	A, JMP	Get machine code for JMP
0295	F842 320000	STA	0000H	Set up JMP at location 0000H
0296	F845 320500	STA	0005H	; and at location 0005H
0297	F848 2103F6	, LXI	H,Warm\$Boot\$Ent	ry ;Get BIOS vector address
0299	F84B 220100	SHLD	0001H	Put address at location 0001H
0300				
0301	F84E 2106E8	LXI	H, BDOS\$Entry	Get BDOS entry point address
0302	F851 220600	SHLD	6	Put address at location 0005H
0303	F854 018000	; LXI	B, 80H	;Set disk I/O address to default
0305	F857 CD65FB	CALL	SETOMA	Use normal BIOS routine
0306		1		, ····
0307	F85A FB	EI		;Ensure interrupts are enabled
0308	F858 3A0400 F85E 4F	LDA MOV	Default\$Disk	Transfer current default disk to
0309	F85F C300E0	JMP	C,A CCP\$Entry	; Console Command Processor ;Transfer to CCP
0311	1001 000000	1	COFWENCTY	iransier to cor
0312		;		
0313			input/output dri	vers
0314		; : These d		at the IOBYTE at location
0316				been set by the cold boot routine.
0317				ied by the STAT utility, by
0318		; BDOS ca	lls, or by a pro	gram that puts a value directly
0319		; into lo	cation 0003H.	
0320		; ; All of	the contines ask	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				lows the call to Select\$Routine.
0325		3 A secon	d entry point, S	elect#Routine#21, uses bits
0325		; 2 and 1 ; by avoi	ding an unnecess	job this saves some space ary instruction.
0328		; 2, 2,01	annecess	
0329	0003 =	IOBYTE	EQU 0003H	;I/O redirection byte
0330		1		
0331		;		
0333		CONST:	:Get co	nsole status
0334			;Entere	d directly from the BIOS JMP vector
0335			; and	returns a parameter that reflects whether
0336			; ther	e is incoming data from the console.
0337			; • A = 00	H (zero flag set) if no data
0339			1A = 0F	FH (zero flag clear) if data
0340			;	
0341			#CONST	will be called by programs that periodic checks to see if the computer
0342			; make	periodic checks to see if the computer ator has pressed any keys for example,
0343			; oper ; to i	ator nas pressed any keys for example, nterrupt an executing program.
0345			,,	
0346	F862 CD6AF8	CALL	Get\$Console\$Sta	
0347				According to status, then convert
L	·····			

Figure 6-4. (Continued)

0348			; to return parameter convention.
0349	F865 B7	ORA	A ;Set flags to reflect status
0350	F866 C8	RZ	; If O, no incoming data
0351	F867 3EFF	MVI	
0352	F869 C9	RET	
	F867 L7	KE I	; indicate incoming data
0353		1	
0354			le\$Status:
	F86A 3A0300	LDA	IOBYTE ;Get I/O redirection byte
0356			Console is selected according to
0357			; bits 1.0 of IOBYTE
0358	F86D CDDCF8	CALL	Select\$Routine ;Select appropriate routine
0359			These routines return to the caller
0360			f of Get\$Console\$Status.
	5070 5/50	-	
0361	F870 F6F8	DW	Teletype\$In\$Status ;00 <- IOBYTE bits 1,0
0362	F872 FCF8	DW	Terminal\$In\$Status ;01
0363	F874 02F9	DW	Communication\$In\$Status ;10
0364	F876 08F9	DW	Dummy\$In\$Status ;11
0365		;	
0366		;	
0367			
0368			
0369		CONIN:	Get console input character
0370			;Entered directly from the BIOS JMP vector;
0371			; returns the next data character from the
0372			Console in the A register. The most significant
0373			; bit of the data character will be 0, except
0374			when "reader" (communication port) input has
0375			; been selected. In this case, the full eight bits
0375			
			; of data are returned to permit binary data to be
0377			; received.
0378			3
0379			Normally, this routine will be called after
0380			; a call to CONST has indicated that a data character
0381			; is ready, but whenever the CCP or the BDDS can
0382			; proceed no further until console input occurs,
0383			; then CONIN will be called without a preceding
0384			
			• • • •
0385			<b>7</b>
	F878 3A0300	LDA	IOBYTE ;Get I/O redirection byte
0387	F878 CDDCF8	CALL	Select\$Routine ;Select correct CONIN routine
0388			;These routines return directly
0389			; to CONIN's caller.
0390	F87E 20F9	DW	Teletype\$Input ;00 <- IOBYTE bits 1,0
0391		DW	Terminal\$Input ;01
	F880 26F9		
0392	F882 2FF9	DW	Communication#Input ;10
0393	F884 35F9	DW	Dummy\$Input ;11
0394		;	
0395		;	
0396			
0397		CONOUT:	;Console output
0398		0010011	;Entered directly from BIOS JMP vector;
0398			, entered directly from blog one vector;
			; outputs the data character in the C register
0400			; to the appropriate device according to bits
0401			; 1,0 of IOBYTE
0402			;
0403	F886 3A0300	LDA	IOBYTE ;Get I/O redirection byte
0404		CALL	Select\$Routine :Select correct CONOUT routine
0405		Unit	;These routines return directly
0405			; to CONOUT's caller.
	F000 0050	-	; to convolts caller.
0407	F88C 38F9	DW	Teletype\$Output ;00 <- IOBYTE bits 1,0
0408	F88E 3EF9	DW	Terminal\$Output ;01
0409	F890 44F9	DW	Communication#Dutput ;10
0410	F892 4AF9	DW	Dummy\$Output ;11
0411		;	
0412		;	
0413		;	
0414		LISTST:	;List device (output) status
0415			;Entered directly from the BIOS JMP vector;
0416			; returns in A list device status that
0417			; indicates whether the list device can accept
0418			; another output character. The IOBYTE's bits
0419			; 7,6 determine the physical device used.
0420			· · · · · · · · · · · · · · · · · · ·
0421			;A = 00H (zero flag set): cannot accept data
0422			;A = OFFH (zero flag clear): can accept data
0423			
0423			;
1			

Figure 6-4. (Continued)

0424				Research's documentation indicates	
0425			; that	you can always return with $A = 00H$	
0426				not accept data") if you do not wish to	
0427				ment the LISTST routine. This is NOT TRUE.	
0428				do not wish to implement the LISTST routine	
0429			t alway	<pre>s return with A = OFFH ("Can accept data").</pre>	
0430			The LI	ST driver will then take care of things rath	er
0431			r than	otentially hanging the system.	
0432					
0433	F894 CD9CF8	CALL	Get\$List\$Status	;Return A = zero or nonzero	
0434				; according to status, then convert	
0435				; to return parameter convention	
0436	F897 B7	ORA	A	;Set flags to reflect status	
0437	F898 C8	87	-	; If O, cannot accept data for output	
0438	F899 3EFF	MVI	A. OFFH	; Otherwise return $A = OFFH$ to	
0439	F898 C9	RET	n, vrrn	julierwise return m - orrn tu	
	F076 L7	REI		; indicate can accept data for output	
0440		· · · · · · ·	<b>_</b>		
0441		Get\$List\$			
0442	F89C 3A0300	LDA	IOBYTE	;Get I/O redirection byte	
0443	F89F 07	RLC		Move bits 7,6 to 1,0	
0444	F8A0 07	RLC			
0445	F8A1 CDDCF8	CALL	Select\$Routine	Select appropriate routine	
0446				;These routines return directly	
0447				; to Get\$List\$Status's caller.	
0448	FBA4 OBF9	DW	Teletype\$Out\$St		
0449	F8A6 11F9	DW	Terminal\$Out\$St		
0450	F8A8 17F9	DW	Communication\$0		
0451	FBAA 1DF9	DW	Dummy\$Out\$Statu	;11	
0452		2	James Four Folder	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	
0453					
		,			
0454		;			
0455					
0456		LIST:	;List o		
0457				directly from BIOS JMP vector;	
0458			; outpu	s the data character in the C register	
0459			; to th	appropriate device according to bits	
0460			; 7,6 0	IOBYTE	
0461			,		
0462	F8AC 3A0300	LDA	IOBYTE	;Get I/O redirection byte	
0463	FBAF 07	RLC		Move bits 7,6 to 1,0	
0464	F8B0 07	RLC			
0465	F8B1 CDDCF8	CALL	Select\$Routine	Select correct LIST routine	
0466		Of the L	de rec concort me	These routines return directly	
0467				; to LIST's caller.	
0468	F8B4 38F9	DW	Teletype\$Output	; to List's caller. ;00 <- IOBYTE bits 1,0	
0469	F8B6 3EF9	DW			
			Terminal\$Output	;01	
0470	F888 44F9	DW	Communication#0		
0471	F8BA 4AF9	DW	Dummy\$Output	;11	
0472					
0473		;			
0474		3			
0475		;			
0476		PUNCH:	Punch (	utput	
0477				directly from BIOS JMP vector;	
0478				s the data character in the C register	
0479				appropriate device according to bits	
0480			; 5,4 0		
0481			; 3,4 0		
0482	F8BC 3A0300	LDA	IOBYTE	·Get I/O redirection buts	
0482		RRC	LOBTIE	;Get I/O redirection byte	
	F8BF OF			;Move bits 5,4 to 2,1	
0484	F8C0 OF	RRC			
0485	F8C1 OF	RRC			
0486	F8C2 CDDDF8	CALL	Select\$Routine\$		
0487				These routines return directly	
0488				; to PUNCH's caller.	
0489	F8C5 38F9	DW	Teletype\$Output	;00 <- IOBYTE bits 1,0	
0490	F8C7 4AF9	DW	Dummy\$Output	;01	
0491	F8C9 44F9	DW	Communication\$D		
0492	F8CB 3EF9	DW	Terminal\$Output	;11	
0493		;		· = =	
0494		;			
0495					
0496		, READER:	Postar	innut	
		NEMPER:	;Reader		
0497				directly from BIOS JMP vector;	
			1 Input	the next data character from the	
0498					
0498				device into the A register	

Figure 6-4. (Continued)

r					·		·			
0500				;The ar	propriat	e device	is sele	cted acc	ordina	
0501				; to bi	ts 3,2 o	f IOBYTE			or gring	
0502				;						
0503	F8D0	340300	LDA RRC	IOBYTE		;Get I/O redirection byte ;Move bits 3,2 to 2,1				
0505		CDDDF8	CALL	Select\$Routine\$	21.					
0506		000010	CHEC	Select #Nout The	-21		correct routines			
0507						to RE	ADER's c	aller.	directly	
0508		38F9	DW	Teletype\$Output			IOBYTE b			
0509		4AF9	DW	Dummy\$Output		;01				
0510		44F9	DW	Communication\$C		;10				
0511	FSUA	3EF9	DW	Terminal\$Output		;11				
0513			;							
0514			;							
0515			;							
0516			Select\$Rou	tine:			rol to a			
0517							calling		accordi	ng to
0518	FBDC	07	RLC				bits 1,0		~ •	
0520	1 020	•/	REC				alues int to word at			
0521			;		, 10 00		o woru a	, i come ci	e	
0522			Select\$Rou	tine\$21:	;Entry ;	oint to	select	routine	selectio	n bits
0523					; are a	iready i	n bits 2,	, 1		
0524	FSDD FSDF		ANI	0000\$0110B	;Isolat	r just b	its 2,1			
0525	FSUF	E3	XTHL				rd of add	iresses	after	
0527	F8E0	5F	MOV	E,A		instruct	on value	to addr	acc + === 1	•
0528	F8E1		MVI	D, 0	; base	3616011	on varue			=
0529	F8E3	19	DAD	D		elected	routine	address		
0530					;Get ro	utine ad	dress int	to HL		
0531	F8E4		MOV	A, M	LS byte					
0532	F8E5		INX	Н	;HL -> 1					
0533	F8E6 F8E7		MOV	H, M L, A	;HS byte ;HL −> 1					
0535	FBEB		XTHL	<b>L</b> , H			> routine	-		
0536	F8E9		RET				lected ro			
0537			;							
0538			;							
0539			7							
0541			; Input/Ou	utput Equates						
0542	OOED	=	, Teletype\$S [.]	tatus\$Port		EQU	OEDH			
0543	OOEC		Teletype\$D		EQU	OECH				
0544	0001	=	Teletype\$O	utput\$Ready		EQU	0000\$000	01B	;Status	mask
0545	0002	-	Teletype\$I	nput\$Ready		EQU	0000\$001	OB	;Status	mask
0546	0001		;							
0547	0001		Terminal\$S Terminal\$Da		EQU	EQU 02H	01H			
0549	0001			ata⇒rort ⊔tput\$Ready	EQU	EQU	0000\$000	18	:Status	mask
0550	0002		Terminal\$I			EQU	0000\$000		;Status	
0551			;					•		
0552	OOED			ion\$Status\$Port	EQU	OEDH				
0553	OOEC			ion\$Data\$Port	500	EQU	OECH	- 64 - 1		
0554	0001 0002			ion\$Output\$Ready	EQU	0000\$00		;Status		
0556	0002	-	i communicati	ion\$Input\$Ready	2010	0000\$00	100	;Status	10 el 5 K	
0557	OODF	=	Communicati	ion\$Baud\$Mode		EQU	ODFH		;Mode Se	elect
0558	OODE			ion\$Baud\$Rate		EQU	ODEH		Rate Se	
0559			,							
0560			,							
0561				device control to	aoles					
0562			; : In order	r to reduce the a	amount of	execute	able code	-		
0564				low-level drive					orts.	
0565			; On entry	/ to the low-lev	el driver				-	
0566			; appropri	iate control tab	le.					
0567			1							
0568	F8EA	cn	Teletype\$Ta DR	able: Teletype\$Status!	Por +					
0569	FBEB		DB	Teletype\$Data\$P						
0571	FREC		DB	Teletype\$Output						
0572	F8ED		DB	Teletype\$Input\$						
0573			1							
0574	F8EE	~	Terminal\$Ta DB	ble: Terminal\$Status!	Davi					
0575	- 022	<u></u>	50	ierminal#Status						

Figure 6-4. (Continued)

0576 0577	F8EF 02 F8F0 01	DB DB	Terminal\$Data\$Port Terminal\$Output\$Ready	
0578	F8F1 02	DB	Terminal\$Input\$Ready	
0580		Communicat	tion\$Table:	
0581	F8F2 ED	DB	Communication\$Status\$Po	prt
0582	F8F3 EC	DB	Communication\$Data\$Port	
0583	F8F4 01	DB DB	Communication#Output#Rea	
0584	F8F5 02	1 DB	Communication>input>Kea	dy
0586		,		
0587		;		
0588				
0589		; The fol	llowing routines are "cal	lled" by Select\$Routine
0590		; to peri	form the low-level input,	/output
0591		Teletynet	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		<u>.</u>		
0597	FBFC 21EEF8	LXI	In\$Status: H,Terminal\$Table	HL -> control table
0599	F8FF C34BF9	JMP	Input\$Status	Note use of JMP. Input\$Status
0600		011	Inputtotutu	; will execute the RETurn.
0601		;		
0602			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 ; will execute the RETurn.
0605				) will execute the Rendrin.
0607		, Dummy\$In\$	Status:	;Dummy status, always returns
0608	F908 3EFF	MVI	A, OFFH	; indicating incoming data is ready
0609	F90A C9	RET		
0610		;		
0611		; 	Out\$Status:	
0612	F908 21EAF8	LXI	H. Teletype\$Table	:HL -> control table
0614	F90E C356F9	JMP	Output\$Status	Note use of JMP. Output\$Status
0615				; will execute the RETurn.
0616		;		
0617			Out#Status:	10 X
0618	F911 21EEF8 F914 C356F9	LXI JMP	H,Terminal\$Table Output\$Status	;HL -> control table ;Note use of JMP. Output\$Status
0620	F914 C330F9	Ulur.	Output#Status	; will execute the RETurn.
0621		;		,
0622		Communica	tion\$Out\$Status:	
0623	F917 21F2F8	LXI	H,Communication\$Table	;HL -> control table
0624	F91A C356F9	JMP	Output\$Status	;Note use of JMP. Output\$Status ; will execute the RETurn.
0625		•		; will execute the REFORM.
0627		, Dummy\$Out	\$Status:	;Dummy status, always returns
0628	F91D 3EFF	MVI	A, OFFH	; indicating ready for output
0629	F91F C9	RET		
0630		;		
0631		1	•	
0632	F920 21EAF8	Teletype\$ ⊾XI	H, Teletype\$Table	;HL -> control table
0633	F923 C360F9	JMP	Input\$Data	Note use of JMP. Input\$Data
0635				; will execute the RETurn.
0636		;		
0637		Terminal\$		14 X 1
	F926 21EEF8	LXI	H,Terminal\$Table	;HL -> control table
0639	F929 CD60F9	CALL	Input\$Data	; will execute the RETurn. ;** Special case **
0641	1727 CBOVF7	UNLL	ingut#Dele	;Input\$Data will return here
0642	F92C E67F	ANI	7FH	; so that parity bit can be set O
0643		RET		
0644		1		
0645			tion\$Input:	-10 ×
0646	F92F 21F2F8	LXI JMP	H,Communication\$Table .	;HL -> control table ;Note use of JMP. Input\$Data
0647	F932 C360F9	JHP	Input\$Data	; will execute the RETurn.
0649		;		
0650		Dummy\$Inp		;Dummy input, always returns
0651	F935 3E1A	MVI	A, 1AH	; indicating CP/M end of file
L				

Figure 6-4. (Continued)

0652	F937 C	:9	RET			
0653			;			
0654			;			
0655			;			
0656			; Teletype\$(			
0658	F938 2	15450	LXI	H,Teletype\$Tab]		-18 - 5 5 - 4 - 5
0659	F938 C			Output\$Data	le	;HL ~> control table ;Note use of JMP. Output\$Data
0660						; will execute the RETurn.
0661			;			
0662			Terminal\$0	Dutput:		
0663	F93E 2	1EEF8	LXI	H,Terminal\$Tabl	le	;HL -> control table
0664						; will execute the RETurn.
0665	F941 C	370F9	JMP	Output#Data		Note use of JMP. Output\$Data
0666			_			; will execute the RETurn.
0668			Formunicat	ion\$Output:		
0669	F944 2	16268	LXI	H,Communication	STable	;HL -> control table
0670	F947 C		JMP	Output\$Data	UT BOIE	Note use of JMP. Output\$Data
0671						; will execute the RETurn.
0672			,			
0673			Dummy\$Outp	uti		;Dummy output, always discards
0674	F94A C	9	RET			; the output character
0675			;			
0676			;			
0678			,			
0679				re the general p	urpose la	w∼level drivers.
0680						priate control table.
0681						ins the data to be output.
0682			;			
0683			Input\$Stat	us:		with A = OOH if no incoming data,
0684		_				ise A ≈ nonzero.
	F94B 78		NOV STA	A, M	;Get sta	
0686	F94C 32		DB	Input\$Status\$Po IN		;*** Self-modifying code *** o A from correct status port
0688	1 247 01	9	1 20	114	, input t	o A from correct status port
0689			, Input\$Stat	us\$Port:		
0690	F950 00	0	DB	00	;<- Set	above
0691	F951 23	3	INX	н	;Move HL	to point to input data mask
0692	F952 23	3	INX	н		
0693	F953 23		INX	н		
0694	F954 Ad		ANA	M	;Mask wi	th input status
0696	F955 C9	·	RET			
0697			,			
0698			Output\$Sta	tus:	Return	with A = 00H if not ready for output
0699						ise A = nonzero.
0700	F956 7E		MOV	A, M	;Get sta	
0701	F957 32		STA	Output\$Status\$P		*** Self-modifying code ***
0702	F95A DE	3	DB	IN		o A from correct status port
0703			;			
0704			Output\$Sta DB	tus\$Port: 00		
0705	F95B 00 F95C 23		INX	H	;<− Set	above to point to output data mask
0707	F95D 23		INX	H	HOVE HL	to point to output data mask
0708	F95E A6		ANA	M	:Mask wi	th output status
0709	F95F C5		RET			
0710	_		;			
0711			,			
0712			Input\$Data	1	Return	with next data character in A.
0713						r status routine to indicate
0714	F960 E5		PUSH	н	; incomi	
0716	F961 CE	ABE9	CALL	n Input\$Status		ntrol table pointer It status in zero flag
0717	F964 E1		POP	H	Recover	control table pointer
0718	F965 CA		JZ	Input#Data		til incoming data
0719	F968 23		INX	н	;HL -> d	ata port
0720	F969 7E		MOV	A,M	;Get dat	a port
0721	F96A 32		STA			f-modifying code ***
0722	F96D DB	5	DB	IN	JINPUt t	o A from correct data port
0723			J ToouleDais	*Dorts		
0724	F96E 00	<b>`</b>	Input\$Data DB	\$Port: 0	;<- Set	ahove
0725	F96F C9		RET	•	, - aet	
0727			1			
1						
L						

Figure 6-4. (Continued)

r							
0728			7				
0729			Output\$Dat	ai	;Output the data character in the C register.		
0730					;Wait for status routine to indicate device		
0731					; ready to accept another character		
	F970		PUSH	H	Save control table pointer		
0733		CD56F9	CALL	Output\$Status	;Get output status in zero flag		
0734	F974		POP	H	Recover control table pointer		
0735	F975 F978	CA70F9	JZ	Output\$Data	Wait until ready for output		
0736	F979		INX MOV	H A.M	;HL -> output port ;Get output port		
0738		327FF9	STA	Output\$Data\$Por			
0739	F97D		MOV	A,C	Get data character to be output		
	F97E		DB	OUT	Butput data to correct port		
0741			•				
0742			Output\$Dat				
0743			DB	0	;<- Set above		
0744	F980	C9	RET				
0745			;				
0746			7 				
0747				vel diskette driv	vers		
0748			; . There d	vivore parform *	ne following functions:		
0750			; inese u ;	A TAKE TO MELION I	IS TOTTOWEND FUNCTIONS!		
0751				Select a specif	ied disk and return the address of		
0752			1		disk parameter header		
0753					umber for the next read or write		
0754					number for the next read or write		
0755			; SETDMA	Set the DMA (rea	ad/write) address for the next read or write.		
0756			; SECTRAN	Translate a log:	ical sector number into a physical		
0757			; HOME	Set the track to	o O so that the next read or write will		
0758			;	be on Track O			
0759				····			
0760					evel drivers are responsible for making		
0761			; the 5 1 ; to CP/M	/+ TIOPPY diske	ttes that use a 512-byte sector appear used a 128-byte sector. They do this		
0763					blocking/deblocking code,		
0764					later in this listing.		
0765				ior to the code i			
0766			1				
0767			;				
0768			;				
0769			; Disk pa	rameter tables			
0770			;				
0771		As discussed in Chapter 3, these describe the physical					
0772		; characteristics of the disk drives? In this example BIOS,					
0773		; there are two types of disk drives; standard single-sided,					
0774		single-density 8", and double-sided, double-density 5 1/4"					
0775		; diskettes.					
0777		; ; The standard 8" diskettes do not need to use the blocking/					
0778			; deblock	ing code, but the	s 5 1/4" drives do. Therefore an additional		
0779					to the disk parameter block to		
0780					ich logical disk's physical		
0781					mer or not it needs deblocking.		
0782			,				
0783			;				
0784			; Disk de	finition tables			
0785			,				
0786					rameter headers, with one entry		
0787					and disk parameter blocks, with		
0788					ock per logical disk or the same		
0790			; paramet( 1	er DIOCK TOP SEVE	ral logical disks.		
0791			, 1				
0792			, Disk\$Parama	eter\$Headers:	Described in Chapter 3		
0793			1				
0794					Disk A: (5 1/4" Diskette)		
0795		6BFB	DW	Floppy\$5\$Skewtab			
0796		000000000		0,0,0	Reserved for CP/M		
0797		C1F9	DW	Directory\$Buffer			
0798	F98B		DW	Floppy\$5\$Paramet	er\$Block		
0799	F98D		DW	Disk\$A\$Workarea			
0800	F98F		DW	Disk\$A\$Allocatio	n\$Vector		
0801			;	-1 1 1	Dieb D. (B. 1/48 Diebette)		
0802	F991	ARER	DW	Floppy\$5\$Skewtab	Disk B: (5 1/4" Diskette) le ;Shares same skew table as A:		
L_0003	F 774			1 AUPP7#J#JREWIED	ie johares same skew laute as hi		

Figure 6-4. (Continued)

F993 0000000000 0804 D₩ 0,0,0 Reserved for CP/M 0805 F999 C1F9 F998 42FA DW Directory\$Buffer ;Share same buffer as A: 0806 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 0807 F99D 81FA DW Private work area Disk\$B\$Workarea 0808 F99F D7FA DW Disk\$B\$Allocation\$Vector Private allocation vector 0809 3 ;Logical Disk C: (8" Floppy) Accountable ;8" skew table 0810 0811 F9A1 B3FB Floppy\$8\$Skewtable DW 0812 F9A3 0000000000 DW 0,0,0 Reserved for CP/M 0813 F9A9 C1F9 DW Directory\$Buffer :Share same buffer as A: F9AB 52FA 0814 Floppy\$8\$Parameter\$Block 0815 F9AD A1FA DW Disk\$C\$Workarea ;Private work area 0816 F9AF EDFA nu Disk\$C\$Allocation\$Vector Private allocation vector 0817 ; 0818 (Logical Disk D: (8" Floppy) 0819 F9B1 6BFB Floppy\$5\$Skewtable DW ;Shares same skew table as A: DW 0820 F9B3 0000000000 0.0.0 Reserved for CP/M 0821 F989 C1F9 DW Directory\$Buffer ;Share same buffer as A: ;Same DPB as C: 0822 F9BB 52FA DW Floppy\$8\$Parameter\$Block 0823 F9BD B1FA Disk\$D\$Workarea DW Private work area Disk\$D\$Allocation\$Vector 0824 F9BF OCEB DW 0825 0826 0827 0828 F9C1 Directory\$Buffer: DS 128 0829 ٠ 0830 0832 0833 . Disk Types 0834 0835 0001 = ;5 1/4" mini floppy Floppv\$5 EQU 1 0836 0002 = Floppy\$8 FOU 2 38" floppy (SS SD) 0837 0838 Blocking/deblocking indicator 0839 0840 0080 = Need\$Deblocking 1000\$0000B EQU ;Sector size > 128 bytes 0841 0842 0843 ; Disk parameter blocks 0844 0845 5 1/4" mini floppy : 0846 0847 ;Extra byte prefixed to indicate ; disk type and blocking required 0848 FA41 81 DB Floppy\$5 + Need\$Deblocking 0849 0850 Floppy\$5\$Parameter\$Block: 0851 FA42 4800 DW 72 ;128-byte sectors per track 0852 FA44 04 DB 4 ;Block shift 0853 FA45 OF DB 15 Block mask 0854 FA46 01 DB ;Extent mask 1 174 FA47 AE00 FA49 7F00 0855 DW ;Maximum allocation block number DW 127 ;Number of directory entries - 1 0856 0857 FA4B CO DB 1100\$0000B Bit map for reserving 1 alloc. block 0858 FA4C 00 DB 0000\$0000B ; for file directory 0859 FA4D 2000 ;Disk changed work area size DW 32 0860 FA4F 0100 DW 1 Number of tracks before directory 0861 ; 0862 0863 ; Standard 8" Floppy 0864 #Extra byte prefixed to DPB for 0865 ; this version of the BIOS 0866 FA51 02 DB Floppy\$8 ;Indicates disk type and the fact 0867 ; that no deblocking is required 0868 Floppy\$8\$Parameter\$Block: 0869 FA52 1A00 DW 26 :Sectors per track ;Block shift ;Block mask 0870 FA54 03 DB 37 0871 FA55 07 DB FA56 00 0872 **DR** 0 ;Extent mask FA57 F200 FA59 3F00 242 0873 DW DW ;Maximum allocation block number 0874 Number of directory entries - 1 63 FA5B CO 1100\$0000B 0875 DB Bit map for reserving 2 alloc. blocks 0876 FA5C 00 DB 0000\$0000B ; for file directory 0877 FA5D 1000 DW 16 Disk changed work area size 0878 FASE 0200 DW 2 Number of tracks before directory 0879 ; 0880 ;

Figure 6-4. (Continued)

· · · · ·									
0881			ork areas						
0882		; . These we would be the DDOC to detect any unsupported							
0884		Figure 1 to the BDOS to detect any unexpected change of diskettes. The BDOS will automatically set							
0885		such a	changed diskette	e to read-only status.					
0886		;							
0887	FA61	Disk\$A\$Wor		32 ; A:					
0888	FA81 FAA1	Disk\$B\$Wor Disk\$C\$Wor		32 ; B: 16 ; C:					
0890	FAB1	Disk\$D\$Wor		16 ; D:					
0891	1 100	1		,					
0892		1							
0893			llocation vectors	6					
0894		; These we used by the BDOC to printerin a bit man of							
0895		First the set of the BDOS to maintain a bit map of which allocation blocks are used and which are free.							
0897		; One byte is used for eight allocation blocks, hence the							
0898				(allocation blocks/8)+1.					
0899		2							
	FAC1 FAD7		location\$Vector location\$Vector	DS (174/8)+1 ; A: DS (174/8)+1 ; B:					
0902	FAU/	DISK#D#A1:	rocation#vector	US (1/4/8/+1 ; b:					
0903	FAED	Disk\$C\$A1	location\$Vector	DS (242/8)+1 ; C:					
0904	FBOC	Disk\$D\$A1	location\$Vector	DS (242/8)+1 ; D:					
0905		;							
0906	0004 =	; hhunhaut-fi	Logical#Disks	EQU 4					
0907		Number soft	ACORICALADIRKS						
0909		;							
0910		SELDSK:		;Select disk in C					
0911				C = 0 for drive A, 1 for B, etc.					
0912				;Return the address of the appropriate ; disk parameter header in HL, or 0000H					
0914				; if the selected disk does not exist.					
0915				;					
	FB2B 210000	LXI	н,о	Assume an error					
0917	FB2E 79	MOV	A,C	;Check if requested disk valid					
0918	FB2F FE04 FB31 D0	CPI RNC	Number\$of\$Logi	cal\$Disks #Return if > maximum number of disks					
0920	1991 90	1		sherein in y maximum namber of disks					
0921	FB32 32EAFB	STA	Selected\$Disk	;Save selected disk number					
0922				;Set up to return DPH address					
0923	FB35 6F FB36 2600	MOV	L, A H, O	Make disk into word value					
0925	FB36 2000	1111	n, v	;Compute offset down disk parameter					
0926				; header table by multiplying by					
0927				; parameter header length (16 bytes)					
0928	FB38 29	DAD	н	; *2					
0929	FB39 29 FB3A 29	DAD DAD	H H	; *4 ; *8					
0931	FB3B 29	DAD	H	; *16					
0932	FB3C 1181F9 FB3F 19	LXI	D,Disk\$Paramet	er\$Headers ;Get base address					
0933		DAD	D	;DE -> Appropriate DPH					
0934	FB40 E5	PUSH	н	;Save DPH address					
0935		;		Access disk parameter block					
0937				; to extract special prefix byte that					
0938				; identifies disk type and whether					
0939				; deblocking is required					
0940	FB41 110A00	LXI	D, 10	; ;Get DPB pointer offset in DPH					
0942	FB44 19	DAD	D, 10	;DE -> DPB address in DPH					
0943	FB45 5E	MOV	Ē,M	;Get DPB address in DE					
0944	FB46 23	INX	H.						
0945	FB47 56 FB48 EB	MOV XCHG	D,M	;DE -> DPB					
0946	FB48 CB	DCX	н	;DE -> prefix byte					
0948	FB4A 7E	MOV	Ä, M	;Get prefix byte					
0949	FB4B E60F	ANI	OFH	;Isolate disk type					
0950	FB4D 32FAFB	STA	Disk\$Type	;Save for use in low-level driver					
0951	FB50 7E FB51 E680	MOV ANI	A,M Need\$Deblockin	;Get another copy of prefix byte g           ;Isolate deblocking flag					
0952	FB53 32F9FB	STA	Deblocking\$Req						
0954	FB56 E1	POP	H	Recover DPH pointer					
0955	FB57 C9	RET							
0956		;							

Figure 6-4. (Continued)

0957	,					
0958						
				gical track for a	next read or wr.	ite
0959			1			
0960			SETTRK:			
0961			MOV	Н, В	;Selected trac	ck in BC on entry
0962			MOV	L,C		
0963		22EBFB	SHLD	Selected\$Track	;Save for low-	-level driver
0964	i FB5D	C9	RET			
0965	5		;			
0966	5		;			
0967	7		; Set lo	gical sector for	next read or w	rite
0968			1		next read of w	
0969			;			
0970			, SETSEC:			
		70			;Logical secto	or in C on entry
0971			MOV	A,C		
0972		32EDFB	STA	Selected#Sector	r ;Save for low-	-level driver
0973		C9	RET			
0974			;			
0975			;			
0976	5		; Set di	sk DMA (input/out	(put) address fo	or next read or write
0977	,		;			
0978		0000	DMA\$Addre	ss: DW	0 : DMA =	address
0979			1		, , , , , , , , , , , , , , , , , , , ,	
0980			SETDMA:		;Address in BC	
0981		40	MOV			
				L,C	;Move to HL to	o save
0982			MOV	н, в		
0983		2263FB	SHLD	DMA\$Address	;Save for low-	-level driver
0984		C9	RET			
0985			;			
0986			;			
0987			; Transla	ate logical secto	r number to phy	sical
0988			;			
0989			; Sector	translation tabl		
0990						ical sector number,
0991			1 and co	ntain the corresp	onding physical	sactor number
0992			1		conding physical	sector Humber.
0993			, Floppy\$5\$5	Skeut ablas	·Each churdes]	
0994			LIObbase:	SKEWLEDIE:		sector contains four
					; 128-byte sec	
0995			1	Physical 128b	Logical 128b	Physical 512-byte
0996		00010203	DB	00,01,02,03	;00,01,02,03	0)
0997		10111213	DB	16,17,18,19	;04,05,06,07	4 )
0998	FB73	20212223	DB	32,33,34,35	;08,09,10,11	8 )
0999	FB77	OCODOEOF	DB	12,13,14,15	112,13,14,15	3 ) Head
1000	FB7B	1C1D1E1F	DB	28,29,30,31	;16,17,18,19	7 ) 0
1001	FB7F	08090A0B	DB	08,09,10,11	;20,21,22,23	2)
1002		18191A1B	DB	24,25,26,27	;24,25,26,27	ā ś
1003		04050607	DB	04,05,06,07	128,29,30,31	i j
1004		14151617	DB			
		14131017		20,21,22,23	; 32, 33, 34, 35	5)
1005			,			
1006		24252627	DB	36,37,38,39	;36,37,38,39	0 1
1007		34353637	DB	52, 53, 54, 55	;40,41,42,43	4 )
1008		44454647	DB	68,69,70,71	;44,45,46,47	8 ]
1009		30313233	DB	48,49,50,51	;48,49,50,51	3 ] Head
1010	FB9F	40414243	DB	64,63,66,67	;52,53,54,55	7]1
1011		2C2D2E2F	DB	44, 43, 46, 47	; 56, 57, 58, 59	2 1
1012		3C3D3E3F	DB	60,61,62,63	;60,61,62,63	6 ]
1013		28292A2B	DB	40, 41, 42, 43	164,65,66,67	1 1
1014		38393A3B	DB	56, 57, 58, 59	\$68,69,70,71	5 1
1015						
1016						
1018			, Floppy\$8\$5	koutsble-	Reandand OF 5	he i una
1017					Standard 8" D	
			1	01,02,03,04,05,		Logical sectors
1019		01070D1319		01,07,13,19,25,	05,11,17,23,03	Physical sectors
1020			;			
1021			;	11, 12, 13, 14, 15,		Logical sectors
1022		090F150208	B DB	09,15,21,02,08,	14,20,26,06,12	Physical sectors
1023			;			
1024			;	21, 22, 23, 24, 25,	26 Logic	al sectors
1025		1218040A10	DB DB	18,24,04,10,16,	22 :Physi	cal sectors
1026					,	
1027						
1028			SECTRAN		Translate los	ical sector into physical
1029						= logical sector number
1030						
1030					, <u>u</u> c	-> appropriate skew table
					7 	
1032					;on exit, HL =	physical sector number

Figure 6-4. (Continued)

r							
1033	FBCD	FB	XCHG		2HI ->	skew ta	able base
1034	FBCE		DAD	в			al sector number
1035	FBCF		MOV	Ľ.M			sector number
1036		2600	MVI	H, 0			ló-bit value
				<b>n</b> , 0	) nake	into a i	C-DIC VALUE
1037	FBD2	L7	RET				
1038			;				
1039			;				
1040			;				
1041			HOME :		: Home	the sele	ected logical disk to track 0.
1042					Befor	e doing	this, a check must be made to see
1043							ical disk buffer has information
1044							written out. This is indicated by
1045							write\$Buffer, set in the
1046							
					•	ocking c	.00e.
1047					, 1		
1048		3AE9FB	LDA	Must\$Write\$Buf	fer		c if physical buffer must
1049	FBD6		ORA	A		; be	written out to disk
1050	FBD7	C2DDFB	JNZ	HOME\$No\$Write			
1051	FBDA	32E8FB	STA	Data\$In\$Disk\$B	uffer	iNo, s	o indicate that buffer
1052							now unoccupied.
1053			HOME\$No\$Wr	ite:			1
1054	FBDD	OFOO	MVI	C,0		: Sat +	o track 0 (logically
1055		CD58FB	CALL	SETTRK			actual disk operation occurs)
1056	FBE2		RET	Set that		, 10	were wish operation occurs/
		.,	nE I				
1057			_				
1058			;				••
1059							floppy drive is transferred
1060							512 bytes long (it was
1061							holds the "one-time"
1062			<pre>; initial:</pre>	ization code us	ed for t	he cold	boot procedure).
1063			;				
1064			; The bloc	cking/deblockin	g code a	ttempts	to minimize the amount
1065			; of actua	al disk I/O by	storing	the disk	, track, and physical sector
1066			: current	ly residing in	the Phys	ical Buf	fer. If a read request is for
1067			; a 128-b;	te CP/M "secto	r" that	already	is in the physical buffer,
1068			; then no	disk access oc	curs.		to in the physical barrery
1069			1	disk access of	cu, s.		
1070							
1071	0800	-	, 	Block\$Size	EQU	2048	
1072	0012		Physical 160	sc\$Per\$Track			
1072	0004				EQU	18	148 1 481 HAR
			CPM\$Sec\$Per		EQU	Physic	al\$Sector\$Size/128
1074	0048		CPM\$Sec\$Per		EQU	CPM\$Se	c\$Per\$Physical*Physical\$Sec\$Per\$Track
1075	0003		Sector\$Masi		EQU		c\$Per\$Physical-1
1076	0002	=	Sector\$Bit	≸Shift	EQU	2	;LOG2(CPM\$Sec\$Per\$Physical)
1077			;				
1078				; These	are the	values	handed over by the BDOS
1079				; whe	n it cal	ls the W	RITE operation.
1080				;The a	llocated.	/unalloc	ated indicates whether the
1081							e to an unallocated allocation
1082							cates this for the first
1083							te) or to an allocation block
1084							en allocated to a file.
1085							es if it is set to write to
1085							es TI TE TE PEL LO MLILE LO
1086					file di	ectory.	
	0000	_	16-24-4477	; ;	500	•	
1088	0000		Write\$Alloc		EQU	<u>o</u>	
1089	0001		Write\$Direc		EQU	1	
1090	0002	-	Write\$Unal]	located	EQU	2	
1091			;				
1092	FBE3	00	Write\$Type:		DB	0	Contains the type of write
1093							; indicated by the BDOS.
1094			;				
1095							
1096			In\$Buffer\$I	k\$Trk\$Sec:			;Variables for physical sector
1097							; currently in Disk\$Buffer in memory
1098	FBE4	00	In\$Buffer\$D	liek.	DB	0	; These are moved and compared
1099	FBE5		In\$Buffer\$1				
1100	FBE7				DW	0	; as a group, so do not alter
1100	r DE /	00	In\$Buffer\$5	ector:	DB	v	; these lines.
		~~	J	1.4D. 11-	~~	•	
1102	FBE8	00	Data\$In\$Dis	k⊅Butter:	DB	0	;When nonzero, the disk buffer has
1103							; data from the disk in it.
	FBE9	00	Must\$Write\$	Buffer:	DB	0	;Nonzero when data has been
1105							; written into Disk\$Buffer but
1106							<pre>p not yet written out to disk</pre>
1107			;				
1108			Selected\$Dk	\$Trk\$Sec:	;Variat	les for	selected disk, track, and sector
1						•-•	
L							

1100						1	
1109	FBEA		Selected	Bieks	; (Se DB	elected by O	SELDSK, SETTRK,and SETSEC) ; These are moved and
1111		0000	Selected		DW	ő	; inese are moved and ; compared as a group so
1112	FBED		Selected		DB	0	; compared as a group so ; do not alter order.
1113	, 050		Gerecied:	Sector.	00	v	, do not after proer.
1113 1114 1115 1116 1117 1118 1119	FBEE	: 00	Selected4	DB	0	;Selected physical sector derived ; from selected (CP/M) sector by ; shifting it right the number of ; of bits specified by ; Sector\$Bit\$Shift	
	FBEF	00	Selected\$	Disk\$Type:	DB	0	;Set by SELDSK to indicate either ; 8" or 5 1/4" floppy
	FBFO	00	Selected\$	Disk\$Deblock:	DB	0	; Set by SELDSK to indicate whether ; deblocking is required.
1124 1125							
1126 1127				ed\$Dk\$Trk\$Sec:			eters for writing to a previously ; unallocated allocation block.
1128	FBF1		Unallocat		DB	0	; These are moved and compared
1129		0000		ed\$Track:	DW	0	; as a group so do not alter
1130	FBF4	00	Unallocat	ed\$Sector:	DB	0	; these lines.
1131	FBF5	00	11			•	<b>N N N N N N N N</b>
1132	r 18 F 3	00	Unallocat	ed\$Record\$Count:	DB	0	Number of unallocated "records"
1133							; in current previously unallocated
1135							; allocation block.
1136	FBF6	00	Disk <b>\$</b> Erro	r\$Flag:	DB	0	Nonzero to indicate an error
1137						v	; that could not be recovered
1138							; by the disk drivers. BDOS will
1139							; output a "bad sector" message.
1140			;				
1141			;Flags us	ed inside the deb	lockin	g code	
1142						-	
1143	FBF7	00	Must\$Prer	ead\$Sector:	DB	0	Nonzero if a physical sector must
1144							: be read into the disk buffer
1145							; either before a write to an
1146							; allocated block can occur, or
1147 1148							; for a normal CP/M 128-byte
1148	FBF8	00	Read\$0per		np.	~	; sector read
1149	r 96 8		Readpuper	ation:	DB	0	;Nonzero when a CP/M 128-byte ; sector is to be read
1151	FBF9	00	Deblockin	g\$Required:	DB	0	; sector is to be read ;Nonzero when the selected disk
1152		~~	SCOLOCKIN			•	; needs deblocking (set in SELDSK)
1153	FBFA	00	DiskSType	1	DB	0	; Indicates 8" or 5 1/4" floppy
1154						-	; selected (set in SELDSK).
1155			;				
1156			;				
1157							ied by previous calls
1158			; to sel	ect disk and to s	et tra	ck and sec	tor. The sector will be read
1159				he address specif	ied in	the previ	ous call to set DMA address.
1160			;				
1161							ors larger than 128 bytes,
1162 1163					used	to "unpack	" a 128-byte sector from
1163			; the ph READ:	ysical sector.			
1165	FBFP	3AF9FB	LDA	Deblocking\$Requ	ired	· Check	if deblocking needed
1166	FBFE		ORA	A			was set in SELDSK call)
1167		CA52FD	JZ	Read\$No\$Deblock			e normal nondeblocked
1168			~			,, 45	maa nongebioekeu
1169				;The de	blocki	ng algoriti	hm used is such
1170							n can be viewed
1171				; up u	ntil t	he actual	data transfer as
1172							rst write to an
1173				; unal	locate	d allocati	
1174	FC02		XRA	Α		;Set th	e record count to 0
1175		32F5FB	STA	Unallocated\$Rec	ord\$Co		
1176	FC06		INR	A Decadeou 17			te that it is really a read
1177		32F8FB	STA	Read\$Operation			is to be performed
1178 1179	FLOA	32F7FB	STA	Must\$Preread\$Se	ctor	; and	force a preread of the sector
1180	FCOD	3502	MVI	A.Write\$Unalloc	-+	; to g	et it into the disk buffer
1180		3E02 32E3FB	STA	A,Write\$Unalloc Write\$Type	aleO	; as i	eblocking code into responding f this is the first write to an
1182	,	ULLUFD	JIM	HITCENIALE			t this is the first write to an located allocation block.
	FC12	C36EFC	JMP	Perform\$Read\$Wr	ite		mmon code to execute read
						,	

~						
	1184					
	1185			: Write	a 128-byte sector from the	e current DMA address to
	1186				eviously selected disk, to	
1	1187			;		
1	1188			; On arr	ival here, the BDOS will H	have set register C to indicate
	1189			; whethe	r this write operation is	to an already allocated allocation
1	1190					the sector may be needed),
ł.	1191					the data will be written to the
	1192					rst 128-byte sector of a previously
	1193 1194			; unallo	cated allocation block (1)	n which case no preread is required).
	1195				wites to the directory tal	ke place immediately. In all other
	1196					rom the DMA address into the disk
	1197					en circumstances force the
	1198					l disk operations can therefore
	1199			; be red	uced considerably.	
E	1200			;		
	1201		3AF9FB	WRITE:		
	1202	FC15		LDA	Deblocking\$Required	Check if deblocking is required
	1203 1204		CA4DFD	JZ	A Write\$No\$Deblock	;(flag set in SELDSK call)
	1205	FCIP	CHADED	52	MLI(Saldoapeniock	
	1206	FC1C	AF	XRA	Α	;Indicate that a write operation
Į.	1207		32F8FB	STA	Read\$Operation	; is required (i.e. NOT a read)
	1208	FC20		MOV	A,C	Save the BDOS write type
	1209	FC21	32E3FB	STA	Write\$Type	
	1210	FC24	FE02	CPI	Write\$Unallocated	;Check if the first write to an
	1211					; unallocated allocation block
	1212	FC26	C237FC	JNZ	Check\$Unallocated\$Block	No, check if in the middle of
	1213					<pre># writing to an unallocated block</pre>
	1214 1215					Yes, first write to unallocated ; allocation block initialize
	1216					; variables associated with
1	1217					; unallocated writes.
	1218	FC29	3E10	MVI	A, Allocation \$Block \$Size/	
	1219					; sectors and
1	1220	FC2B	32F5F8	STA	Unallocated\$Record\$Count	
	1221					\$
1	1222		21EAFB	LXI	H,Selected\$Dk\$Trk\$Sec	;Copy disk, track, and sector
	1223		11F1F8	LXI	D,Unallocated\$Dk\$Trk\$Sec	; into unallocated variables
	1224	FC34	CD35FD	CALL	Move\$Dk\$Trk\$Sec	
	1225			; 		with the second line that
Ļ	1226				if this is not the first w	write to an unallocated he unallocated record count
	1228					of 128-byte sectors in the
	1229				tion block.	
	1230			1		
	1231			Check\$Una	llocated\$Block:	
	1232		3AF5FB	LDA	Unallocated\$Record\$Count	t
	1233	FC3A		ORA	Α	
	1234	FC3B	CA66FC	JZ	Request\$Preread	;No, this is a write to an
	1235					; allocated block
1	1236 1237					;Yes, this is a write to an ; unallocated block
	1238	FC3E	30	DCR	A	; Count down on number of 128-byte sectors
	1239			Den		; left unwritten to in allocation block
		FC3F	32F5FB	STA	Unallocated\$Record\$Count	
	1241					• • • • • •
	1242	FC42	21EAFB	LXI	H,Selected\$Dk\$Trk\$Sec	;Check if the selected disk, track,
	1243		11F1FB	LXI		c; and sector are the same as for
L	1244		CD29FD	CALL	Compare\$Dk\$Trk\$Sec	; those in the unallocated block.
	1245	FC4B	C266FC	JNZ	Request\$Preread	;No, a preread is required
Į.	1246 1247					;Yes, no preread is needed.
	1248					<pre>;Now is a convenient time to ; update the current sector and see</pre>
	1249					; if the track also needs updating.
	1250			;		,
	1251					<b>;By design, Compare\$Dk\$Trk\$Sec</b>
	1252					; returns with
	1253					; DE -> Unallocated\$Sector
	1254	FC4E		XCHG		; HL -> Unallocated\$Sector
	1255	FC4F		INR	M	;Update Unallocated\$Sector
	1256 1257	FC50 FC51		MOV CPI	A,M CPM\$Sec\$Per\$Track	<pre>;Check if sector now &gt; maximum . on a track</pre>
	1257		DASFFC	JC	No\$Track\$Change	; on a track ;No (A < ¹ M)
	1259	. 000			HEALI FERMONICIAL	iYes,
L						····
-						

Figure 6-4. (Continued)

1260	FC56	3600	MVI	M. 0	Peret center to 0
1261		2AF2FB			Reset sector to 0
				Unallocated\$Track	;Increase track by 1
	FC5B		INX	н	
	FC5C	22F2FB	SHLD	Unallocated\$Track	
1264			;		
1265			No\$Track\$	Change:	
1266					;Indicate to later code that
1267					
	FOFF		-	•	; no preread is needed.
1268	FC5F		XRA	A	
1269		32F7FB	STA	Must\$Preread\$Sector	;Must\$Preread\$Sector=0
1270	FC63	C36EFC	JMP	Perform\$Read\$Write	
1271			;		
1272			Request \$P	reread:	N N
1273	FC66	AF	XRA	Δ	;Indicate that this is not a write
		32F5FB	STA	Unallocated\$Record\$Coun	
					t ; into an unallocated block.
	FC6A		INR	A	
	FC6B	32F7FB	STA	Must\$Preread\$Sector	;Indicate that a preread of the
1277					; physical sector is required.
1278			;		
1279			;		
1280			Perform\$R	ead\$Write:	<pre>\$Common code to execute both reads and</pre>
1281					; writes of 128-byte sectors.
	FC6E	AE	XRA	•	
					Assume that no disk errors will
	FC6F	32F6FB	STA	Disk\$Error\$Flag	; occur
1284					
		3AEDFB	LDA	Selected\$Sector	;Convert selected 128-byte sector
1286	FC75		RAR		; into physical sector by dividing by 4
1297	FC76	1F	RAR		
1288	FC77	EAGE	ANI	3FH	Remove any unwanted bits
1289		32EEFB	STA	Selected\$Physical\$Sector	
1290	FC/7	JACCIO	316	Selected#Physical#Sector	
1291	FC7C	21E8FB	LXI	H,Bata\$In\$Disk\$Buffer	;Check if disk buffer already has
1292	FC7F	7E	MOV	Α,Μ	; data in it.
1293	FC80	3601	MVI	M, 1	;(Unconditionally indicate that
1294					; the buffer now has data in it)
1295	FC82	B7	ORA	A	;Did it indeed have data in it?
1296		CAA3FC	JZ		;No, proceed to read a physical
1297	FLOS	CAMORE	02	Keau#Sector #Into#purier	
					; sector into the buffer.
1298					
1299				;The bu	ffer dòes have a physical sector
1300				; in i	t.
1301				: Note	: The disk, track, and PHYSICAL
1302					or in the buffer need to be
1303					ked, hence the use of the
1304					are\$Dk\$Trk subroutine.
1305				, comp.	arevokvink subrodtine.
					Charles is analysis in buddens in the
1306	FC86	11E4FB	LXI		Check if sector in buffer is the
1307		21EAFB	LXI	H,Selected\$Dk\$Trk\$Sec	; same as that selected earlier
		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					
1311	EC92	3AE7FB	LDA	In\$Buffer\$Sector	Get physical sector in buffer
		21EEFB	LXI	H,Selected\$Physical\$Sec	
	FC98		CMP	M	;Check if correct physical sector
1314		CABIFC	JZ	Sector\$In\$Buffer	;Yes, it is already in memory
	FU77	CHDIFC	. 32	Sector #111#Butter	FIEST IT IS GILEGUY IN MEMORY
1315			1		
1316			Sector\$No	t\$In\$Buffer:	
1317					;No, it will have to be read in
1318					; over current contents of buffer
	FC9C	3AE9FB	LDA	Must#Write\$Buffer	Check if buffer has data in that
	FC9F		ORA	Α	; must be written out first
1321		C495FD	CNZ	Write\$Physical	;Yes, write it out
1322	COHO	049010		*** A ********************************	,,
			Passing and		
1323				or\$into\$Buffer:	
1324	FCA3	CD11FD	CALL	Set\$In\$Buffer\$Dk\$Trk\$Se	
1325					; selected disk, track, and sector
1326					; to reflect which sector is in the
1327					; buffer now
1328	FCAA	3AF7FB	LDA	Must\$Preread\$Sector	;In practice, the sector need only
1329	FCA9	87	ORA	A	; be physically read in if a preread
1330	- UH Z		OU CH	••	; is required
	FC++	-	C-117	Peads Physics	
1331		C49AFD	CNZ	Read\$Physical	;Yes, preread the sector
1332	FCAD	AF	XRA	Α	Reset the flag to reflect buffer
1333	FCAE	32E9FB	STA	Must\$Write\$Buffer	; contents.
1334			;		
1335			Sector\$In	Buffer: ;Selecte	ed sector on correct track and

Figure 6-4. (Continued)

1336				
1336				
				7 disk is already in the buffer.
1337				Convert the selected CP/M (128-byte)
1338				; sector into a relative address down
1339				; the buffer.
1340	FCB1 3AEDFB	LDA	Selected#Sector	;Get selected sector number
1341	FCB4 E603	ANI	Sector\$Mask	;Mask off only the least significant bits
1342		MOV		
	FCB6 6F		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 29	DAD	н	;* 4
1346	FCBB 29	DAD	н	;* 8
1347	FCBC 29	DAD	Н	3* 16
1348	FCBD 29	DAD	н	;* 32
1349	FCBE 29	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				<pre>#HL -&gt; 128-byte sector number start</pre>
1355				; address in disk buffer
1356	FCC4 EB	XCHG		;DE -> sector in disk buffer
1357	FCC5 2A63FB	LHLD	DMA\$Address	Get DMA address set in SETDMA call
1358	FCC8 EB	XCHG		Assume a read operation, so
1359				; DE -> DMA address
1360				; HL -> sector in disk buffer
1361	FCC9 OE10	MVI	C,128/8	; Because of the faster method used
1362	1007 0210		0,120,0	; to move data in and out of the
1363				; disk buffer, (eight bytes moved per
1364				; loop iteration) the count need only
1365				
1366				; be 1/8th of normal.
				At this point -
1367				; $C = 1 \text{ op count}$
1368				; DE -> DMA address
1369				; HL -> sector in disk buffer
1370	FCCB 3AF8FB	LDA	Read\$Operation	;Determine whether data is to be moved
1371	FCCE B7	ORA	Α	; out of the buffer (read) or into the
1372	FCCF C2D7FC	JNZ	Buffer\$Move	; buffer (write)
1373				;Writing into buffer
1374				;(A must be 0 get here)
1375	FCD2 3C	INR	A	;Set flag to force a write
1376	FCD3 32E9FB	STA	Must#Write\$Buff	er ; of the disk buffer later on.
1377	FCD6 EB	XCHG		<pre>Make DE -&gt; sector in disk buffer</pre>
		XUN0		
1378	1000 60	XCH0		
		;		
1379				
1379 1380		;	/e1	; HL -> DMA address
1379 1380 1381		;	/e:	; HL -> DMA address ;The folowing move loop moves eight bytes
1379 1380 1381 1382		;	/e:	; HL -> DMA address ; The folowing move loop moves eight bytes ; at a time from (HL) to (DE), C contains
1379 1380 1381 1382 1383		; ; Buffer\$Mov		<ul> <li>; HL -&gt; DMA address</li> <li>; The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> </ul>
1379 1380 1381 1382 1383 1384	FCD7 7E	; ; Buffer\$Mov MOV	A, M	<ul> <li>; HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385	FCD7 7E FCD8 12	; ; Buffer\$Mov MOV STAX	A, M D	<ul> <li>HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>;Get byte from source</li> <li>;Put into destination</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386	FCD7 7E FCD8 12 FCD9 13	; Buffer\$Mov MOV STAX INX	A, M D D	<ul> <li>; HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387	FCD7 7E FCD8 12 FCD9 13 FCD4 23	; J Buffer\$Mov STAX INX INX	A, M D D H	; HL -> DMA address ; HL -> DMA address ; The folowing move loop moves eight bytes ; at a time from (HL) to (DE), C contains ; the loop count. ; Get byte from source ;Put into destination ;Update pointers
1379 1380 1381 1382 1383 1384 1385 1386 1386 1387 1388	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDB 7E	; Buffer\$Mov Stax INX INX MOV	A, M D D H A, M	<ul> <li>; HL -&gt; DMA address</li> <li>; HL -&gt; DMA address</li> <li>; The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> <li>; Update pointers</li> <li>; Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDB 7E FCDC 12	; J Buffer\$Mov Stax INX INX Mov Stax	A, M D D H A, M D	<ul> <li>; HL -&gt; DMA address</li> <li>; HL -&gt; DMA address</li> <li>; The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390	FCD7 7E FCD8 12 FCD9 13 FCD8 23 FCD8 7E FCDC 12 FCDC 13	; Buffer\$Mov STAX INX INX MOV STAX INX	A, M D H A, M D	<ul> <li>; HL -&gt; DMA address</li> <li>; HL -&gt; DMA address</li> <li>; The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> <li>; Update pointers</li> <li>; Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1399 1390	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDB 7E FCDC 12 FCDD 13 FCDE 23	; ; Buffer\$Mov Stax INX INX MOV Stax INX INX	A, M D H A, M D U H	<ul> <li>; HL -&gt; DMA address</li> <li>; HL -&gt; DMA address</li> <li>; The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Fut into destination</li> <li>; Update pointers</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1391	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDB 7E FCDC 12 FCDC 12 FCDD 13 FCDE 23 FCDF 7E	; J Buffer\$Mov Stax INX INX Mov Stax INX INX Mov	A, M D H A, M D D H A, M	<ul> <li>HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1388 1389 1390 1391 1392 1393	FCD7 7E FCD9 12 FCD9 13 FCD8 72 FCDB 72 FCDC 12 FCDD 13 FCDF 72 FCCD 12	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX INX STAX	A, M D H A, M D H H A, M D	<ul> <li>; HL -&gt; DMA address</li> <li>sThe folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; bet box from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1389 1399 1399 1399	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDD 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13	; ; Buffer\$Mov Stax INX Mov Stax INX INX Nov Stax INX Nov Stax INX	A, M D D H A, M D D H A, M D D	<ul> <li>HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394	FCD7 7E FCD9 13 FCD9 13 FCD8 23 FCD8 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23	; ; Buffer\$Mov STAX INX MOV STAX INX INX MOV STAX INX INX INX	A, M D H A, M D H H A, M D	<ul> <li>; HL -&gt; DMA address</li> <li>sThe folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; bet box from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1389 1399 1390 1391 1392 1393 1394 1395	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDC 12 FCDC 12 FCDC 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E	; ; Buffer\$Mov Stax INX INX MOV Stax INX INX INX Stax INX NOV Stax INX	A, M D H A, M D H A, M D H A, M	<ul> <li>; HL -&gt; DMA address</li> <li>; HL -&gt; DMA address</li> <li>; The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; But into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1387 1389 1399 1399 1399 1395 1395	FCD7 7E FCD9 13 FCD9 13 FCDA 23 FCDB 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE3 7E	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX STAX INX INX INX INX STAX	A, M D D H A, M D H A, M D D H A, M D D	<ul> <li>HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1387 1387 1387 1390 1391 1392 1393 1394 1395 1394 1395 1396 1397	FCD7 7E FCD9 13 FCD9 13 FCDB 72 FCDB 72 FCDD 12 FCDD 13 FCDF 72 FCE0 12 FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE3 7E FCE4 12 FCE5 13	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX INX STAX INX INX NOV STAX INX	A, M D H H, M D H A, M D D H A, M D D D H A, M D D	<ul> <li>; HL -&gt; DMA address</li> <li>; HL -&gt; DMA address</li> <li>; The folowing move loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; the loop count.</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; But into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1390 1392 1393 1394 1395 1395 1395 1397	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDD 7E FCDC 12 FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE4 12 FCE4 12 FCE5 13 FCE4 23	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX STAX INX INX INX INX STAX	A, M D D H A, M D H A, M D D H A, M D D	<ul> <li>HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1398 1399 1390 1391 1392 1393 1394 1395 1395 1395 1395 1396 1397	FCD7 7E FCD9 13 FCD9 13 FCD8 23 FCDB 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE3 7E FCE4 12 FCE5 13 FCE5 23 FCE5 7E	; ; Buffer\$Mov STAX INX INX MOV STAX INX MOV STAX INX INX INX INX MOV STAX INX INX MOV STAX INX INX MOV	A, M D H H, M D H A, M D D H A, M D D D H A, M D D	<ul> <li>HL -&gt; DMA address</li> <li>The folowing move loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> <li>(Update pointers</li> <li>(Get byte from source</li> <li>(Put into destination</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1386 1387 1390 1390 1390 1392 1393 1394 1395 1394 1395 1394 1395 1399 1400	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDC 12 FCDC 12 FCDC 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE4 12 FCE4 12 FCE5 13 FCE6 23 FCE7 7E FCE6 12	; ; Buffer\$Mov Stax INX INX MOV Stax INX INX INX Stax INX Stax INX Stax INX	A, M D D H A, M D D H A, M D D H A, M D D H A, M D D D D D D D D D D D D D D D D D D D	; HL → DMA address ; HL → DMA address ; The folowing hove loop moves eight bytes ; at a time from (HL) to (DE), C contains ; the loop count. ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers
1379 1380 1381 1382 1383 1384 1385 1386 1387 1398 1399 1390 1391 1392 1393 1394 1395 1395 1395 1395 1396 1397	FCD7 7E FCD9 13 FCD9 13 FCD8 23 FCDB 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE3 7E FCE4 12 FCE5 13 FCE5 23 FCE5 7E	; ; Buffer\$Mov STAX INX INX MOV STAX INX MOV STAX INX INX INX INX MOV STAX INX INX MOV STAX INX INX MOV	A, M D H H D H H A, M D H H A, M D H H A, M D H A, M	<ul> <li>HL -&gt; DMA address</li> <li>The folowing hove loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> <li>;Put into destination</li> <li>;Update pointers</li> <li>;Get byte from source</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1386 1387 1390 1390 1390 1392 1393 1394 1395 1394 1395 1394 1395 1399 1400	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDB 7E FCDC 12 FCDC 12 FCDC 13 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE3 7E FCE4 12 FCE5 13 FCE5 13 FCE6 12 FCE8 12 FCE8 12	; ; Buffer\$Mov Stax INX INX MOV Stax INX INX INX INX INX Stax INX INX INX Stax INX Stax INX Stax	A, M D D H A, M D H A, M D D H A, M D D H A, M D D D	; HL → DMA address s The folowing move loop moves eight bytes ; at a time from (HL) to (DE), C contains ; the loop count. ; get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Put into destination ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source
1379 1380 1381 1382 1383 1384 1385 1384 1385 1386 1387 1388 1389 1390 1391 1392 1394 1395 1394 1395 1394 1395 1396 1399 1400	FCD7 7E FCD8 12 FCD9 13 FCD8 23 FCDB 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE3 7E FCE4 12 FCE3 13 FCE4 23 FCE7 7E FCE8 12 FCE9 13 FCE9 13 FCE9 23	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX INX INX STAX INX INX INX INX INX INX INX IN	A, M D H H D D H A, M D H A, M D H H A, M D H H H H H H	<ul> <li>; HL -&gt; DMA address</li> <li>sThe folowing hove loop moves eight bytes</li> <li>; at a time from (HL) to (DE), C contains</li> <li>; bet loop count.</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> <li>; Get byte from source</li> <li>; Put into destination</li> <li>; Update pointers</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1390 1392 1393 1394 1395 1395 1396 1397 1397 1398 1397 1398 1399 1400 1401 1402	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDB 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE4 12 FCE4 12 FCE4 23 FCE9 13 FCE4 23 FCE8 7E	; ; Buffer\$Mov Stax INX INX MOV Stax INX INX INX INX INX Stax INX INX INX INX INX INX INX INX INX INX	A, M D D H A, M D D H A, M D D H H A, M D D H H A, M D H H A, M	<pre>; HL → DMA address ; The folowing move loop moves eight bytes ; at a time from (HL) to (DE), C contains ; the loop count. ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source</pre>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1395 1395 1395 1395 1395 1397 1398 1397 1398 1397 1400 1401 1402	FCD7 7E FCD9 13 FCD9 13 FCDB 23 FCDB 7E FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE3 7E FCE3 12 FCE3 12 FCE5 13 FCE6 23 FCE7 7E FCE8 12 FCE8 12 FCE8 12 FCE8 7E FCE8 7E	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX INX INX STAX INX INX INX INX INX INX INX IN	A, M D H H D D H A, M D H A, M D H H A, M D H H H H H H	<pre>; HL -&gt; DMA address ; The folowing hove loop moves eight bytes ; at a time from (HL) to (DE), C contains ; the loop count. ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers</pre>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1396 1397 1398 1391 1392 1393 1394 1395 1394 1395 1399 1397 1398 1399 1400 1403 1404 1405	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDB 7E FCDC 12 FCDC 12 FCDD 13 FCDF 7E FCE0 13 FCE2 23 FCE3 7E FCE4 12 FCE4 12 FCE4 12 FCE5 13 FCE6 23 FCE6 12 FCE8 FCE8 FCE8 FCE8 FCE8 FCE8 FCE8 FCE8	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX INX INX INX STAX INX INX INX INX INX INX INX INX INX IN	A, M D D H A, M D D H A, M D D H A, M D D H A, M D D H A, M D D D H A, M D D D D D D D D D D D D D D D D D D D	<pre>; HL → DMA address ; The folowing move loop moves eight bytes ; at a time from (HL) to (DE), C contains ; the loop count. ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source</pre>
1379 1380 1381 1382 1383 1384 1385 1386 1387 1389 1390 1391 1392 1393 1393 1395 1395 1395 1395 1395 1397 1398 1397 1398 1397 1398 1397 1400 1401 1402	FCD7 7E FCD8 12 FCD9 13 FCDB 23 FCDB 7E FCDC 12 FCDC 12 FCDC 13 FCDF 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE4 12 FCE4 12 FCE6 12 FCE6 12 FCE8 12 FCE8 7E FCE8 23	; ; Buffer\$Mov STAX INX INX MOV STAX INX MOV STAX INX INX INX INX INX INX INX IN	A, M D D H A, M D D H A, M D D H A, M D D H A, M D D H A, M D D H H A, M	<ul> <li>HL -&gt; DMA address</li> <li>The folowing hove loop moves eight bytes</li> <li>at a time from (HL) to (DE), C contains</li> <li>the loop count.</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> <li>Get byte from source</li> <li>Put into destination</li> <li>Update pointers</li> </ul>
1379 1380 1381 1382 1383 1384 1385 1384 1385 1386 1387 1388 1389 1390 1391 1392 1394 1395 1394 1395 1394 1395 1396 1397 1398 1399 1400 1401 1402	FCD7 7E FCD8 12 FCD9 13 FCD8 72 FCDB 72 FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 7E FCE3 7E FCE3 7E FCE3 12 FCE3 13 FCE4 23 FCE7 7E FCE8 12 FCE9 13 FCE8 7E FCE2 12 FCE5 7E	; ; Buffer\$Mov STAX INX INX MOV STAX INX MOV STAX INX INX MOV STAX INX INX INX INX MOV STAX INX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV	A, M D H H D D H H A, M D H H A, M D H H A, M D H H A, M D H H A, M	<pre>; HL → DMA address ; HL → DMA address ; at a time from (HL) to (DE), C contains ; at a time from source ; Put into destination ;Update pointers ;Get byte from source ;Put into destinatinden ;Update pointers ;Get byte from source ;Put into</pre>
1379 1380 1381 1382 1383 1384 1385 1386 1386 1387 1388 1387 1389 1390 1390 1392 1393 1394 1395 1395 1395 1397 1395 1399 1400 1400 1405	FCD7 7E FCD8 12 FCD9 13 FCDA 23 FCDD 7E FCDC 12 FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 23 FCE3 7E FCE4 12 FCE4 12 FCE5 13 FCE6 23 FCE7 7E FCE8 12 FCE8 1	; ; Buffer\$Mov STAX INX INX MOV STAX INX INX INX INX INX STAX INX INX INX INX INX INX INX INX INX IN	A, M D D H A, M D D H A, M D D H H A, M D D H H A, M D D H H A, M D D H H A, M D D H H A, M D D H A, M D D H A, M D D H A, M D D D H A, M D D D H A, M D D D H A, M D D D H A, M D D D H A, M D D D D H A, M D D D D H A, M D D D H A, M D D D D D D D D D D D D D D D D D D D	<pre>; HL → DMA address ; The folowing move loop moves eight bytes ; at a time from (HL) to (DE), C contains ; the loop count. ;Get byte from source ;Put into destination ;Update pointers ;Get byte from source ;Put into destination ;Update pointers</pre>
1379 1380 1381 1382 1383 1384 1385 1384 1385 1386 1387 1388 1389 1390 1391 1392 1394 1395 1394 1395 1394 1395 1396 1397 1398 1399 1400 1401 1402	FCD7 7E FCD8 12 FCD9 13 FCD8 72 FCDB 72 FCDC 12 FCDD 13 FCDE 23 FCDF 7E FCE0 12 FCE1 13 FCE2 7E FCE3 7E FCE3 7E FCE3 12 FCE3 13 FCE4 23 FCE7 7E FCE8 12 FCE9 13 FCE8 7E FCE2 12 FCE5 7E	; ; Buffer\$Mov STAX INX INX MOV STAX INX MOV STAX INX INX MOV STAX INX INX INX INX MOV STAX INX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV STAX INX MOV	A, M D H H D D H H A, M D H H A, M D H H A, M D H H A, M D H H A, M	<pre>; HL → DMA address ; HL → DMA address ; at a time from (HL) to (DE), C contains ; at a time from source ; Put into destination ;Update pointers ;Get byte from source ;Put into destinatinden ;Update pointers ;Get byte from source ;Put into</pre>

1411	FCF2		INX	н	
1412	FCF3	7E	MOV	A, M	;Get byte from source
1413	FCF4		STAX	D	Put into destination
1414	FCF5	13	INX	D	;Update pointers
1415	FCF6	23	INX	н	
1416					
1417	FCF7	ÓD	DCR	C	;Count down on loop counter
1418		C2D7FC	JNZ	Buffer\$Move	Repeat until CP/M sector moved
	FUPO	CZD/FC	ONZ	builler anove	TREPERI UNITE CEVIA SECTOR MOVED
1419					7
1420	FCFB	<b>3AE3FB</b>	LDA	Write\$Type	;If write to directory, write out
1421		FE01	CPI		
					; buffer immediately
1422	FDOO	3AF6FB	LDA	Disk\$Error\$Flag	;Get error flag in case delayed write or read
1423	FD03	C0	RNZ		Return if delayed write or read;
1424	. 200	00			
					¥
1425	FD04	87	ORA	A	Check if any disk errors have occurred;
1426	FD05	CO	RNZ		;Yes, abandon attempt to write to directory
		00	11112		, and a second to write to briettory
1427					;
1428	FD06	AF	XRA	A	;Clear flag that indicates buffer must be
1429		32E9FB	STA	Must#Write#Buff	
1430	FUOA	CD95FD	CALL	Write\$Physical	;Write buffer out to physical sector
1431	FDOD	3AF6FB	LDA	Disk\$Error\$Flag	Return error flag to caller;
1432	FD10		RET		
	- DIO	67			
1433			;		
1434			•		
			0-147-40-4	fer\$Dk\$Trk\$Sec:	;Indicate selected disk, track, and
1435			Setaluagni	terankairkasec:	
1436					; sector now residing in buffer
1437	ED11	<b>3AEAFB</b>	LDA	Selected\$Disk	
				In\$Buffer\$Disk	
1438	FU14	32E4FB	STA	Insbuttersbisk	
1439					
1440	5017	2AEBFB	LHLD	Selected\$Track	
1441	FD1A	22E5FB	SHLD	In\$Buffer\$Track	
1442					
	-	045550	L DA	Selected\$Physic	- 1 #Contact
1443		SAEEFB			
1444	FD20	32E7FB	STA	In\$Buffer\$Sector	
1445					
1446	FD23	09	RET		
1447			:		
1448			Compare\$Dk	*Twb.	Compares just the disk and track
1440			COmparetok	# 5 F K #	foompares just the disk and thack
1449					; pointed to by DE and HL
1450	ED24	0E03	MVI	C,3	;Disk (1), track (2)
			JMP		Sec\$Loop ;Use common code
1451	FD20	C32BFD	Unit	Compare #DK#Irk#	beckebop ; use common code
1452					
1453			Compare\$Dk	STrk\$Sec:	;Compares the disk, track, and sector
			compare to k	+III K+Ceel	
1454					; variables pointed to by DE and HL
1455	FD29	0E04	MVI	C,4	;Disk (1), track (2), and sector (1)
1456				\$Trk\$Sec\$Loop:	
1457	FD2B	1A	LDAX	D	;Get comparitor
1458	FD2C	RF	CMP	M	;Compare with comparand
					Abandon comparison if inequality found
1459	FD2D		RNZ	-	
1460	FD2E	13	INX	D	;Update comparitor pointer
1461	FD2F	23	INX	н	;Update comparand pointer
1462	FD30				
			DCR	C	;Count down on loop count
1463	FD31		DCR RZ	С	;Count down on loop count ;Return (with zero flag set)
		C8	DCR RZ	С	;Count down on loop count ;Return (with zero flag set)
1464			DCR RZ JMP		;Count down on loop count ;Return (with zero flag set)
1464 1465		C8	DCR RZ	С	;Count down on loop count ;Return (with zero flag set)
1464		C8	DCR RZ JMP	С	;Count down on loop count ;Return (with zero flag set) Sec\$Loop
1464 1465 1466		C8	DCR RZ JMP ;	C Compare\$Bk\$Trk\$	;Count down on loop count ;Return (with zero flag set) Sec\$Loop
1464 1465 1466 1467		C8	DCR RZ JMP	C Compare\$Bk\$Trk\$	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track,and sector
1464 1465 1466 1467 1468		C8	DCR RZ JMP ;	C Compare\$Bk\$Trk\$	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track,and sector ; variables pointed at by HL to
1464 1465 1466 1467 1468		C8	DCR RZ JMP ;	C Compare\$Bk\$Trk\$	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track,and sector
1464 1465 1466 1467 1468 1469	FD32	C8 C32BFD	DCR RZ JMP ; Move\$Dk\$Tr	C Compare\$Bk\$Trk\$ k\$Sec:	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE
1464 1465 1466 1467 1468 1469 1470		C8 C32BFD	DCR RZ JMP ; ; Move\$Dk\$Tr MVI	C Compare\$Dk\$Trk\$: k\$Sec: C,4	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track,and sector ; variables pointed at by HL to
1464 1465 1466 1467 1468 1469 1470 1471	FD32	CB C32BFD OE04	DCR RZ JMP ; ; Move\$Dk\$Tr MVI Move\$Dk\$Tr	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop:	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track,and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1)
1464 1465 1466 1467 1468 1469 1470 1471	FD32	CB C32BFD OE04	DCR RZ JMP ; ; Move\$Dk\$Tr MVI	C Compare\$Dk\$Trk\$: k\$Sec: C,4	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE
1464 1465 1466 1467 1468 1469 1470 1471 1472	FD32 FD35 FD37	CB C32BFD OE04 7E	DCR RZ JMP ; Move\$Dk\$Tr MVI Move\$Dk\$Tr MOV	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track,and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473	FD32 FD35 FD37 FD38	C8 C32BFD 0E04 7E 12	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr MOVe\$Dk\$Tr MOV	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination
1464 1465 1466 1467 1468 1469 1470 1471 1472	FD32 FD35 FD37	C8 C32BFD 0E04 7E 12	DCR RZ JMP ; ; Move\$Dk\$Tr Move\$Dk\$Tr MOV STAX INX	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D D	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track,and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte
1464 1465 1466 1467 1468 1469 1470 1470 1471 1472 1473 1474	FD32 FD35 FD37 FD38 FD39	C8 C32BFD 0E04 7E 12 13	DCR RZ JMP ; ; Move\$Dk\$Tr Move\$Dk\$Tr MOV STAX INX	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination
1464 1465 1466 1467 1469 1470 1471 1472 1473 1474 1475	FD32 FD35 FD37 FD38 FD39 FD3A	C8 C32BFD 0E04 7E 13 23	DCR RZ JMP ; Move\$Dk\$Tr MVI Move\$Dk\$Tr MOV STAX INX	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A.M D D H	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1473 1475 1476	FD32 FD35 FD35 FD37 FD38 FD39 FD3A FD3B	C8 C32BFD 0E04 7E 12 13 23 0D	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr MOV STAX INX INX DCR	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D D	:Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count
1464 1465 1466 1467 1469 1470 1471 1472 1473 1474 1475	FD32 FD35 FD37 FD38 FD39 FD3A	C8 C32BFD 0E04 7E 12 13 23 0D	DCR RZ JMP ; Move\$Dk\$Tr MVI Move\$Dk\$Tr MOV STAX INX	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D H C	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved</pre>
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move \$TAX INX INX DCR RZ	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D H C	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved</pre>
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1475 1476	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D	DCR RZ JMP ; Move\$Dk\$Tr MVI Move\$Dk\$Tr MOV STAX INX INX DCR RZ JMP	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A.M D D H	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved</pre>
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move \$TAX INX INX DCR RZ	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D H C	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved</pre>
1464 1465 1466 1467 1468 1469 1470 1470 1477 1472 1473 1475 1476 1477 1478 1479	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move\$Dk\$Tr Move STAX INX INX INX DCR RZ JMP ;	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D H C	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved</pre>
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move\$Dk\$Tr Mov STAX INX INX INX JCR RZ JMP ;	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D H C	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved</pre>
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1482	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move\$Dk\$Tr Move\$Dk\$Tr Move STAX INX INX INX INX INX JMP ; ; ;	C Compare\$Dk\$Trk\$ k\$Sec: C,4 k\$Sec\$Loop: A,M D D H C C Move\$Dk\$Trk\$Sec	:Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1483	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move\$Dk\$Tr Mov STAX INX STAX INX DCR RZ JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	C Compare\$Dk\$Trk\$: k\$Sec: C,4 &Sec\$Loop: A,M D D H C Move\$Dk\$Trk\$Sec re two "smart" d	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop isk controllers on this system, one</pre>
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1482	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move\$Dk\$Tr Mov STAX INX STAX INX DCR RZ JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	C Compare\$Dk\$Trk\$: k\$Sec: C,4 &Sec\$Loop: A,M D D H C Move\$Dk\$Trk\$Sec re two "smart" d	:Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop
1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1475 1475 1476 1476 1479 1479 1482 1482 1484	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr Move\$Dk\$Tr Move\$Dk\$Tr Move STAX INX INX DCR RZ JMP ; ; There a ; for the	C Compare\$Dk\$Trk\$: k\$Sec: C,4 k\$Sec\$Loop: A,M D D H C Move\$Dk\$Trk\$Sec re two "smart" d 8" floppy diske	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop isk controllers on this system, one</pre>
1464 1465 1466 1467 1469 1470 1471 1472 1473 1474 1475 1477 1476 1477 1478 1482 1483 1483 1485	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move\$Dk\$Tr MVI Move\$Dk\$Tr MOV STAX INX INX INX INX INX INX INX INX INX IN	C Compare\$Dk\$Trk\$: k\$Sec: C,4 &Sec\$Loop: A,M D D H C Move\$Dk\$Trk\$Sec re two "smart" d	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop isk controllers on this system, one</pre>
1464 1465 1466 1467 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1477 1480 1482 1483 1484 1485	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move*Dk*Tr Move*Dk*Tr MOV STAX INX INX DCR RZ JMP ; ; There a ; for the ; mini-di ;	C Compare\$Dk\$Trk\$: k\$Sec: C,4 A,M D H C Move\$Dk\$Trk\$Sec re two "smart" d 8" floppy diske skette drives.	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop isk controllers on this system, one tte drives, and one for the 5 1/4"
1464 1465 1466 1467 1469 1470 1471 1472 1473 1474 1475 1477 1476 1477 1478 1482 1483 1483 1485	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move*Dk*Tr Move*Dk*Tr MOV STAX INX INX DCR RZ JMP ; ; There a ; for the ; mini-di ;	C Compare\$Dk\$Trk\$: k\$Sec: C,4 A,M D H C Move\$Dk\$Trk\$Sec re two "smart" d 8" floppy diske skette drives.	<pre>;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop isk controllers on this system, one</pre>
1464 1465 1466 1467 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1477 1480 1482 1483 1484 1485	FD32 FD35 FD37 FD38 FD39 FD38 FD39 FD34 FD38 FD32	C8 C32BFD 0E04 7E 12 13 23 0D C8	DCR RZ JMP ; Move*Dk*Tr Move*Dk*Tr MOV STAX INX INX DCR RZ JMP ; ; There a ; for the ; mini-di ;	C Compare\$Dk\$Trk\$: k\$Sec: C,4 A,M D H C Move\$Dk\$Trk\$Sec re two "smart" d 8" floppy diske skette drives.	;Count down on loop count ;Return (with zero flag set) Sec\$Loop ;Moves the disk, track, and sector ; variables pointed at by HL to ; those pointed at by DE ;Disk (1), track (2), and sector (1) ;Get source byte ;Store in destination ;Update pointers ;Count down on byte count ;Return if all bytes moved \$Loop isk controllers on this system, one tte drives, and one for the 5 1/4"

Figure 6-4. (Continued)

1488								
1489			; in mem ; operat	iory to detect whi	en they	are to p	perform some disk location 0040H, and	
1490			j Operat	1/4# sesturling	oller n	onitors	Tocation UU40H, and	
1490			; the S	1/4" controller i	nonitors	locatio	on 0045H. These are	
			; called	their disk conti	COL DYLE	s. If th	ne most significant	
1492							controller will	
1493						tive control bytes.		
1494						a valid disk control		
1495				that specifies th	ne exact	disk op	peration to be performed.	
1496			,					
1497			; Once t	he operation has	been co	mpleted,	the controller resets	
1498						This ind	ficates completion	
1499			; to the	disk driver code	? <b>.</b>			
1500			;					
1501							r in a disk status block	
1502							n for this; 0043H.	
1503			; If the	first byte of th	nis stat	us block	c is less than 80H, then	
1504			; a disk	error has occuri	ed. For	this si	imple BIOS, no further details	
1505							lote that the disk controller	
1506							writes are attempted ten	
1507			; times	before the contro	ller re	turns ar	error.	
1508			;					
1509			; The di	sk control table	layout	is shown	below. Note that the	
1510			; contro	llers have the ca	pabilit	y for co	ntrol tables to be	
1511			; chaine	d together so the	it a seq	uence of	disk operations can	
1512			; be ini	tlated. In this E	105 thi	s featur	e is not used. However,	
1513				ntroller requires				
1514							the main control bytes	
1515			; in ord	er to indicate th	e end o	f the ch	ain.	
1516			1				<b>.</b>	
1517	0040		Disk#Cont		EQU	40H	<pre>#8" control byte</pre>	
1518	0041	=	Command\$B	lock\$8	EQU	41H	Control table pointer;	
1519			<u>.</u>					
1520	0043	=	Disk\$Stat	us\$Block	EQU	43H	;8" AND 5 1/4" status block	
1521		_	1				<b>.</b>	
1522	0045		Disk#Cont		EQU	45H	\$5 1/4" control byte	
1523	0046	-	Command\$B	1008#3	EQU	46H	Control table pointer	
1524 1525			,					
1525				Diel Control Tab	1			
1526			; Floppy	Disk Control Tab	185			
1528	FD40	00	7 Floppy\$Co	mm a m d a	DB	0	#Command	
1529	0001		Floppy\$Rea		EQU	о́ін	\$ Command	
1530	0002		Floppyswr		EQU	02H		
1531	FD41		Floppy\$Un:		DB	021	;Unit (drive) number = 0 or 1	
1532	FD42		Floppy\$Hea		DB	õ		
1533	FD43		Floppy\$Tra		DB	õ	;Head number = 0 or 1 ;Track number	
1533	FD44				DB	ŏ		
1534		0000	Floppy\$Sec Floppy\$By		DB DW		;Sector number :Number of bytes to read/write	_
						0		
1536 1537		0000		A\$Address:	DW DW	0	;Transfer address	
1537	F-U47	0000	L TOHD X #46)	xt\$Status\$Block:	DW	v	Pointer to next status block	
1538	EDAP	0000	Floppythe	xt\$Control\$Locati	on DU	0	; if commands are chained.	
1540	F 940	0000	L. TOBH A BUG	ALACOULLOIATOTATOCALI	OUT DW	0	<pre>;Pointer to next control byte ; if commands are chained.</pre>	
1540							; il commands are chained.	
1541			;					
1542								
1543			Write\$No\$	Deblock		a kina é é –	contents of disk buffer to	
1544			MLT ( 6 #140 #1	DEDIQURI			rect sector.	
1546	FD4D	3E02	MVI	A,Floppy\$Write\$	Code		rect sector. rite function code	
1546		C354FD	JMP	Common\$No\$Deblo		- 10er W	common code	
1548		5554F B	Read\$No\$D		~ ~		previously selected sector	
1549						; neau	o disk buffer.	
1550	FD52	3E01	MUT	A, Floppy\$Read\$C	ode		ead function code	
1551	1002	~~~	Common \$No!			,	eas function code	
1552	FD54	3240FD	STA		1 Set o	mmand f	unction code	
1553	. 504	02.701 U		· · · · · · · · · · · · · · · · · · ·			locked command table	
1554	ED57	218000	LXI	H, 128		per sec		
1555		2245FD	SHLD	Floppy\$Byte\$Cou				
1556	FD5D		XRA	A			y has head O	
1557		3242FD	STA	Floppy\$Head	,		,	
1558			010	·				
1559	FDA1	<b>3AEAFB</b>	LDA	Selected\$Disk	18" F1		troller only has information	
1560							and 1 so Selected\$Disk must	
1561						converte		
1562	FD64	E601	ANI	01H		into 0 o		
1563		3241FD	STA	Floppy\$Unit		nit numb		
			•		,			

1545       FDGC 3243FD       LDA       SelectedFTrack       Set track number         1557       FDGC 3243FD       STA       FloepPSTrack       Set track number         1557       FDT2 3244FD       STA       FloepPstSector       Set sector number         1577       FDT2 3244FD       STA       FloepPstSector       Set sector number         1577       FDT2 3244FD       STA       FloepPstSector       Set sector number         1577       FDT2 2247FD       SELD       FLOE       DMASAddress       Italis control laties, but in this case, respectively in the sector in the sector in the sector is an in tailous block in the initial is control byte         1578       FDT2 214000       LXI       H, DiskeControlse       iffort next control byte         1589       FDDE 214000       LXI       H, FloepPstFrickScontrolse       iffort next control trait is all control tr							
1565         FLOP         ALEBR         LDA         SelectedFrack         Set track number           1567         FDGF JACDFB         LDA         SelectedSector         SelectedSector           1570         FDGF JACDFB         LDA         SelectedSector         SelectedSector           1570         FDTS JACJFB         LLD         DMASAddress         irresfer directly between DMA address           1571         FDTS JACJFB         LHLD         DMASAddress         irresfer directly between DMA address           1574         SHLD         FloepySDFAAAddress         in an Stevent chained         irresfer directly between DMA address           1574         SHLD         FloepySDFAAAddress         in an Stevent chained         irresfer directly between DMA address           1574         Inte disk controlls         intext be point most status block at asin control byte         intext be point most status block at asin control byte           1575         FDDS 214000         LXI         H.DiskEStatusBlock is forint ext control byte         is disk control byte           1585         FDB2 214000         LXI         H.BiteSControlS         is control trole control trole perform           1585         FDB2 214000         LXI         H.BiteSControlS         is control trole control trole           1586         FDD2 214000							
1566       FDGC 2343FD       STA       FlopsyNTrack       fSet track number         1567       FDG2 3244FD       LHLD       Statefedescor       Statefedescor         1577       FD72 3244FD       LHLD       DMA&Address       Transfer directly between DMA address         1577       FD72 244FD       LHLD       DMA&Address       Transfer directly Detween DMA address         1577       FD72 244FD       SHLD       FlopsyNTrack       Tsate directly Detween DMA address         1577       FD72 244FD       SHLD       FlopsyNTrack       Tsate directly DMA         1577       FD72 2440D       LXI       H.DiskeStatus®Block       renot uset satus back at         1586       FD81 214000       LXI       H.DiskeStontrol®Control%Location       renot uset satus back at         1586       FD82 2440FD       LXI       H.DiskeStontrol%Control%Location       renot uset satus back at         1586       FD80 214000       LXI       H.DiskeStontrol%Control%Location       renot uset satus back at         1586       FD80 24400       LXI       H.DiskeStontrol%Control%Control%Control%Control       renotrol bree         1586       FD80 24400       SHLD       A.FlopsyNTextes       renotrol bree       renotrol bree         1586       FD80 24400       SHL						7	
156         FDF SAEDFB         LDA         Selected#Sector         1           156         FDD2 3244FD         STA         Flopsy#Sector         pset sector number           157         FD72 2344FD         SHLD         Flopsy#Sector         pset sector number           157         FD78 2247FD         SHLD         Flopsy#Sector         pset sector number           157         FD78 2247FD         SHLD         Flopsy#Sector         pset sector number           157         FD78 22437D         SHLD         Flopsy#Sector         pset sector           157         FD77 224300         LXI         H.Disk#Sector:         pset sector         pset sector           158         FD87 244000         LXI         H.Disk#Sector:         pset sector         pset sector           158         FD87 244000         LXI         H.Disk#Control#Control#Control#control table         proint mest control table           158         FD87 244000         LXI         H.Disk#Control#S         proint mest control table           158         FD87 244000         LXI         H.Disk#Control#S         proint mest control table           158         FD87 244000         LXI         H.Disk#Control#S         proint mest control table           158         FD87 24400		F069	3AEBF B			<b>-</b>	
1566       FDGF       ALDR       Selected%sector         1570       FDFS       2244FD       Sth       Floopv%Extern pst sector number         1571       FDFS       2244FD       SthLD       DMAAddress       I and 0° controller.         1572       FDFS       2247FD       SthLD       Floopv%DMAAddress       I and 0° controller.         1573       IThe disk control table. but inthil asset       ifthe disk control table. but inthil asset         1574       Ithe disk control bits in the base page.       ifthe disk control bits in the base page.         1577       FDFS       214300       LXI       H.Disk%Status%Block       iffent net control bits         1587       FDFS       214000       LXI       H.Disk%Status%Block       iffent net control bits         1586       FDBA       214000       LXI       H.Disk%Status%Block       iffent net control bits         1586       FDBA       214000       LXI       H.Disk%Status%Block       iffent net control bits         1587       FDBA       214000       LXI       H.Disk%Stants%Block       iffent net control bits         1586       FDBA       214000       LXI       H.Disk%Stants%Block       iffent net control bits         1587       FDSA       214000       LXI		FD6C	3243FD	STA	Floppy\$Track	Set track n	umber
1550       FD2 2244FD       STA       FlopsytSector       Set sector number         1570       FD75       2443FB       LHLD       DMAAAddress       Transfer directly between DMA address         1571       FD75       2447FD       SHLD       FlopsytEnAAAddress       is controller         1573       IT       disk controller can accest chained       is controller can accest chained         1574       It       H.D iskStatusBlock       is controllers       is controllers       is controllers       is controllers       is controllers       is controllers       is controller       is controller </td <td></td> <td></td> <td></td> <td></td> <td></td> <td>;</td> <td></td>						;	
1571       FD75 2A43FB       LHLD       DPA4Address       rand 0° controller.         1571       FD76 2247FD       SHLD       DPA4Address       rand 0° controller.         1573       FD78 2247FD       SHLD       DPA4Address       rand 0° controller.         1574       The disk controller can accept chained       r disk control tables, but in this case, it has a coept chaines         1577       FD78 214300       LXI       H.Disk%Estaus@Block       r main status block         1580       FD77 2249FD       SHLD       FloppyMEXAddress       r main status block         1581       FD87 214000       LXI       H.Disk%Econtrol%Location       r back at main control byte         1584       FD87 214000       LXI       H.Disk%Econtrol%E       r control byte         1584       FD87 214000       LXI       H.Disk%Econtrol%E       r control table         1584       FD87 214000       LXI       H.Disk%Econtrol%E       r control table         1584       FD87 214000       LXI       H.Disk%Econtrol%E       r controller to perform         1586       FD87 214000       LXI       H.Disk%Econtrol%E       r correct sector.         1586       FD87 23600       MUI       A.Floppy%eralt       r correct sector.         1587       FD	1568	FD6F	<b>3AEDFB</b>	LDA	Selected#Sector		
1512       FD75       24.576       LHLD       DMAAddress       iTransfer directly between DMA address         1572       FD76       24.770       SHLD       FloeperDMAAddress       iam directly between DMA address         1573       FD76       24.370       ist. Control tables, but in this case, i they are not used, so the "Next" pointers         1574       inst be pointed back at the initial       imat be pointed back at the initial         1577       FD75       214300       LXI       H.Diskt@Staus@Block       imat be pointed back at the initial         1577       FD76       214000       LXI       H.Diskt@Staus@Block       imat be pointed back at each control byte         1586       FD81       214000       LXI       H.Diskt@Control80       if point ext control byte         1587       FD82       214000       LXI       H.Diskt@Control80       if point controle at control byte         1586       FD82       214000       LXI       H.Diskt@Control80       if activate controler to perform         1586       FD82       214000       LXI       H.Diskt@Control80       if activate controler         1587       FD82       23400       WII       H.FloepetWrite&Code       if activate controler         1586       FD82       23601       MI       <	1569	FD72	3244FD	STA	Floppy#Sector 1	Set sector	number
1572       FD76 2247FD       SHLD       FloppyBDMA&Address       Iand B* controller.         1573       ITm disk controls       ITm disk controls       Itm disk controls         1574       Itms disk controls       Can accept chained         1575       Itms disk controls       Itms disk controls       Itms disk controls         1576       FD75 214300       LXI       H.DiskStausBlock       IPPint next status back at         1577       FD75 21400       LXI       H.DiskStontrols       IPPint next status back at         1587       FD81 214000       LXI       H.DiskStontrols       IPPint next status back at         1587       FD82 21407D       LXI       H.DiskStontrols       IPPint next status back at         1587       FD82 214000       LXI       H.FloppyPOCommand       IPPint controls byte         1586       FD80 22400       SHLD       IDSHStontrols       IPPint controls to the rest ontrol table         1586       FD80 214000       LXI       H.FloppyPoCommand       IPPint controls to the rest ontrol         1587       FD97 32400       LXI       H.FloppyPoCommand       IPPint controls to the rest ontrol         1587       FD97 3602       WI       A.FloppyPoCommand       IPPint for to controls         1587       FD97 3602<	1570					;	
1572       FD78 2247FD       SHLD       FloppyBDMAAAddress       ind B* controller.         1573       int disk controller.       int disk controller.       int disk controller.         1574       ither are not used. on the "Mest" pointers.       ither are not used. on the "Mest" pointers.         1577       ither are not used. on the "Mest" pointers.       ither are not used. on the "Mest" pointers.         1577       FD7E 2249FD       SHLD       HDiskStatusBBlock       if Point next status back at         1588       FD81 214000       LXI       H.DiskStontrolStocation       is an istus bbook         1584       FD82 2240FD       SHLD       FloppyMextStontrolStocation       is an istus bbook         1584       FD82 214000       LXI       H.DiskStontrolStocation       is an istus bbook         1585       FD82 214000       LXI       H.DiskStontrolStocation       is an istus bbook         1586       FD80 214000       LXI       H.DiskStontrolStocation       is an istus bbook         1587       FD92 21407D       LXI       H.DiskStontrolStocation       is an istus bbook         1587       FD92 214000       LXI       H.DiskStontrolStocation       is an istus bbook         1587       FD92 214000       LXI       H.DiskStontrolStocatis       is an istus bbook </td <td>1571</td> <td>FD75</td> <td>2A63FB</td> <td>LHLD</td> <td>DMA\$Address</td> <td>Transfer di</td> <td>rectly between DMA address</td>	1571	FD75	2A63FB	LHLD	DMA\$Address	Transfer di	rectly between DMA address
1574       The disk controller can accept chained         1574       If a disk control tables, but in this case, but		FD78	2247FD	SHLD	Floppy#DMA#Addre	ss sand	8" controller.
174       iThe disk control tables, but in this case, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit pointers, if they are not used, so the "Mexit point next status back at it control bytes, if they are not used, so the "Mexit point next status back at an in control byte, if point next control byte, if point next control byte, if point controller at control table, if they are not used, so the "Mexit point controller at control table, if point control table, if point controller at control table, if point controller at control table, if point part control table, if point part control table, if point point control table, if point part part part part part part part par							
1575       : disk control tables, but in this case;         1576       : they are not used; so the "Mext pointers;         1577       FDF 214300       LXI         1580       FDF 214300       LXI         1581       FDF 214300       LXI         1582       FDF 214000       LXI         1584       FDF 214000       LXI         1585       FDF 214000       LXI         1586       FDF 214000       LXI         1587       FDF 21407D       LXI         1588       FDF 21407D       LXI         1587       FDF 21407D       LXI         1588       FDF 21407D       LXI         1587       FDF 21407D       LXI         1586       FDF 21407D       LXI         1587       FDF 21407D       LXI         1588       FDF 21407D       LXI         1589       FDF 21407D       LXI         1580       FDF 21407D       LXI         1581       FDF 114000       LXI         1581       FDF 21407D       LXI         1581       FDF 21407D       LXI         1581       FDF 21407D       LXI         1581       FDF 21407D							ntroller can accept chained
1       they are not used, so the "Next pointers in the base page.         1       must be pointed back at the initial         1577       inust be pointed back at the initial         1580       FD7E 214300       LXI         1581       FD7E 214300       LXI         1582       FD81 214000       LXI         1584       FD84 2248FD       SHLD         1584       FD84 2248FD       SHLD         1584       FD84 2248FD       SHLD         1584       FD84 224800       LXI         1584       FD84 224100       LXI         1584       FD85 214000       LXI         1584       FD84 22410       SHLD         1585       FD80 214000       LXI         1586       FD80 214000       LXI         1587       FD80 24000       LXI         1586       FD80 24000       LXI         1587       FD80 24000       LXI         1588       FD80 24000       LXI         1597       JP0       Common#Physical:         1598       FD97 23607       MPI         1599       FD97 238201       MPI         1599       FD97 238201       MPI         1599       FD97							
1577       1 must be pointed back at the initial						they are	not used, so the "Next" pointers
1576       FD72 214300       LXI       H_DiskeStatusBlock       imain status block at instruction byte         1587       FD72 22497D       SHLD       Floppy#Next%StatusBlock       imain status block at instruction byte         1588       FD82 22487D       LXI       H_DiskeStatusBlock       imain status block at instruction byte         1586       FD82 22487D       LXI       H_Floppy#Next%Control%Location       iback at main control byte         1586       FD82 224000       LXI       H_Floppy#Next%Control%S       ifActivate controller at control table         1586       FD82 224000       LXI       H_DiskeControl%S       ifActivate controller to perform         1586       FD82 24400       LXI       H_DiskeControl%S       ifActivate controller to perform         1586       FD82 24500       LXI       H_DiskeControl%S       ifActivate controller to perform         1586       FD82 24500       LXI       H_DiskeControl%S       ifActivate controller to perform       ifActivate controller to perform         1589       FD95 2600       HWI       A.Floppy#Write%Code       ifGet urits function code       ifGet control         1597       FD95 3600       HWI       A.Floppy#Krite%Code       ifGet disk type (set in SELDSK)       ifGet control         1600       FD97 34767       UMP <td></td> <td></td> <td></td> <td></td> <td></td> <td>, must be p</td> <td>ointed back at the initial</td>						, must be p	ointed back at the initial
1576       FD7E 214300       LXI       H.DiskStatus@Block       Point mext status back at         1580       FD81 214000       LXI       H.DiskStatus@Block       ; main status block         1581       FD81 214000       LXI       H.DiskStatus@Block       ; main status block         1582       FD84 2248FD       SkLD       Floepy%Next%Control%       ; back st main control byte         1585       FD85 224100       SkLD       Floepy%Next%Control%Location       ; back st main control table         1586       FD85 224100       LXI       H.Floepy%Command       ; point controller at control table         1586       FD85 24400       LXI       H.Disk%Control%       ; and status back at         1587       FD75 3600       LXI       H.Disk%Complete       ; and status back at         1587       FD75 3602       JUP       Wilk@For%Disk%Complete       ; operation.         1587       FD75 3602       MUI       A.Floepy%Nerite@Code       ; Obe to common code         1586       FD76 3202       MUI       A.Floepy%Red%Code       ; Get disk buffer.         1587       FD76 3200FD       STA       Floepy%Red%Code       ; Get disk type (set in SELDSK)         1601       FD76 3240FD       JL       Disk%Type       ; Set command table <td></td> <td></td> <td></td> <td></td> <td></td> <td>, must be p</td> <td>vies in the base name</td>						, must be p	vies in the base name
<pre>1500 FD7E 2249FD SHLD FloepyPNext#Status#Block ; main status block 1581 1582 FD81 214000 LXI H.Disk#Control#8 ; Point next control byte 1585 FD87 2140FD LXI H.FloepyPNExt#Status#Block 1586 FD87 2140FD LXI H.FloepyPNExt#Status#Block 1586 FD87 2140FD LXI H.FloepyPNExt#Status#Block 1587 1588 FD80 214000 LXI H.Disk#Control#8 ; Point controller at control table 1586 FD87 2140FD LXI H.FloepyPNExt#Status#Block 1587 1588 FD80 214000 LXI H.Disk#Control#8 ; Point controller to perform 1590 FD92 C3F7FD UPF Wait#ForSubstComplete 1590 FD7 C3F7FD UPF Wait#ForSubstComplete 1595 FD97 C39CFD UPF Common#PFFsical ; UPFite contents of disk buffer to 1596 FD97 C39CFD UPF Autimet#Common code 1596 FD97 C39CFD UPF Common#PFFsical ; into disk buffer. 1600 FD87 G39C7 I WI A.FloepyPNExt#Stote ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExt#Stote ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 G39C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 FD87 G30C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 FD87 G30C7 I WI A.FloepyPNExd#Stode ; Det read function code 1600 FD87 FD87 G30C7 I WI A.FloepyPNExd#Stode ; Set command table 1600 FD87 FD87 G30C7 I FloepyPNExd#Stode ; Set controller 1610 FD87 G30C7 I WI A.floepyPNExd#Store ; Set up disk control table 1614 FD80 G30 FD87 FD87 FD87 FD87 FD8 FD8 ; Set up track number i for table 1615 FD80 G400 MI I B.0 In#Buffer#Stack ; Set up track number i for table 1616 FD80 G407 MI B.0 In#Buffer#Stack ; Set up track number i for table 1617 FD80 G400 MI B.0 In#Buffer#Stack ; Set up track number i for table 1618 FD80 G400 M</pre>		CD70	014000	1 7 7			
1581       FDB1 214000       LXI       H_Disk#Control#8       ; back at main control byte         1582       FDB4 2248FD       SHLD       FloppyMPxxt#Control#0.coation       ;       back at main control byte         1584       FDBA 224100       LXI       H_Floppy#Conmand       ;       pPoint controller at control table         1585       FDBA 224100       LXI       H_Disk#Control#8       ;       pPoint controller at control table         1586       FDBA 214000       LXI       H_Disk#Control#8       ;       pActivate controller to perform         1587       FDBA 214000       LXI       H_Disk#Control#8       ;       pActivate controller at control table         1587       FDBA 20400       LXI       H_Disk#Control#8       ;       pActivate control rest control         1587       FDBA 20400       LXI       H_Disk#Control#8       ;       pActivate control rest control         1597       JP       JP       Write#For#Disk#Complete       ;       Deration.         1597       JP       JP       Gommon#Physical:       ;       Det ocommon code         1597       JP       State       pace at antor code       ;       Det ocommon code         1597       JP       State common#Physical:       ;       Det oc		FD/8	214300				
1582FDB1 214000LXIH.Disk&Control&Foint ext control byte1584FDB7 2140CDLXIH.Floppy%CommandiPoint controller at control table1585FDB7 2140CDLXIH.Floppy%CommandiPoint controller at control table1586FDB0 244000LXIH.Disk&Control&SiActivate controller to perform1586FDB0 26800WVIH.Disk&CompleteiActivate controller to perform1590FDS2 CSFFDUMPWait%For%Disk%CompleteiActivate controller to perform1591ificificcorrect sector.iCorrect sector.1597FDS5 2602MVIA.Floppy%Write%CodeiCorrect sector.1597FDS5 2602MVIA.Floppy%Read%CodeiCorrect sector.1600FD9A 3201MVIA.Floppy%Read%CodeiCorrect sector.1601FD9A 3201MVIA.Floppy%Read%CodeiCorrect sector.1602iCommon%PhysicaliControl%iCorrect sector.1603FD9C 3240FDSTAFloppy%CommandiSet commond table1604FD9C 3240FDJZCorrect%Disk*TypeiControl is a 5 1/4" Floppy1605FDA2 32676BSTTDisk*TypeiSet up disk control table1611FDA2 32676BSTTDisk*TypeiSet up disk control table1611FDA2 32676BSTAFloppy%TackiSet up disk control table1611FDA2 32676BSTAFloppy%TackiSet up disk controller1615FDA2 3246FDSTAFloppy%Tack		FU/E	2247FD	SHLU	PIOPPY#Next#Statu	US#DIOCK	F Main Status Diock
1583FD64 2240FDSHLDFloppyWext@control%Location; back at main control byte1584FD87 2140FDLXIH,FloppyWommad;Point controller at control table1584FD87 2140FDLXIH,Jisk@control%S;1584FD80 214000LXIH,Disk@control%S;1585FD80 214000LXIH,Disk@control%S;1586FD80 24600LXIH,Disk@control%S;1587FD80 24600LXIH,SOH;1589FD87 2607WIA,FloppyWitte%Complete;1591JMFKati%For%Disk@Complete;;1592;;;;1593Write%Physical:;;;1594Write%Physical:;;;1595SE02MVIA,Floppy%Write%Code;;1597FD97 28001MVIA,Floppy%Write%Code;;1600FD97 3801MVIA,Floppy%Command;fSet command table1603;Common%Physical:;;;1604FD9C 3240FDSTAFloppy%Command;fSet command table1605;;Correct%Disk%Type;;;1604FD9C 3240FDSTAFlopp%Command;fSet up disk control table1605;;;;;;;1607FD9C 3240FDSTAFlopp%Command;;;1604FD9C 3201M							
1584 1585FDB7 2140FD FDB7 2140FDLXIH,Floppy%Command FDB7 214000Point controller at control table1586 1587 1589FDB0 214000 FDD2 03800LXIH,Disk&Control%B H,Disk&Complete Activate controller to perform ; operation.1589 1590 1591 1592FDB2 24000 HVI M,BOH HLXI H,Disk&Complete operation.1590 1592 1593 1593Hrite%Physical: HVI A,Floppy%Write%Code IS97 FDP5 3602 HVI A,Floppy%Write%Code FRead%Physical: iS98 FD97 C39CFD HVI A,Floppy%Read%Code IS99 FD97 3600 FD97 38001 HVI A,Floppy%Read%Code HVI A,Floppy%Read%Code IS99 FD97 38001 HVI A,Floppy%Read%Code IS99 FD97 38001 HVI A,Floppy%Command STA Floppy%Command IS98 FD97 38001 HVI A,Floppy%S IS98 FD97 38001 HVI A,Floppy%Command IS98 FD97 38001 HVI A,Floppy%Command IS98 FD97 38001 HVI A,Floppy%Command IS98 FD97 38001 HVI A,Floppy%Command IS98 FD97 38001 HVI A,Floppy%Command IS98 FD97 38001 HVI A,Floppy%Command IS98 FD97 38001 HVI A,Floppy%S IS00 IS98 FD97 38001 HVI A,I IS88 FD97 38001 HVI A,I IS88 FD97 38001 HVI A,I IS88 FD97 38001 HVI A,I IS88 FD80 IS88 FD80 IS88 FD80 IS98 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 IS88 FD80 <br< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></br<>							
1585FDB7 2140PDLXIH,Floppy@Command tormand#Sick#8;Point controller at control table1586FDB2 214000LXIH,Disk@Control%8;Activate controller to perform ; operation.1589FDB2 214000LXIH,Disk@Control%8; operation.1599FDB2 2677DJHPWait%fors@Disk@Complete; operation.1591FDS2 2677DJHPWait%fors@Disk@Complete; operation.1593JHPCommon@Fhysical:; uperation.code; operation.1594JHPCommon@Fhysical:; uperation.code; operation.code1595FDS2 5202HVI A,Floppy%Write@Code; Got voice sector.; operation.code1596FDS2 3201HVI A,Floppy%Command; Set command table; foot voice sector.1600FDS2 3240FDSTAFloppy%Command; Set command table1604FDS2 3240FDSTAFloppy%Command; Set up disk control table1605;Correct%Disk%Type; Get disk type (set in SELDSK)1604FDS2 3240FDSTAFloppy%Lik%Type1614FDA2 3261PUPAit1614FDA2 3240FDSTAFloppy%Lik%Type1615FDAD 3AE4FBLDAin%Buffer%Disk1616FDA2 3240FDSTAFloppy%Lik%Type1617FDA2 3240FDSTAFloppy%Tack1616FDA2 3240FDSTAFloppy%Tack1617FDB2 3243FDSTAFloppy%Tack1616FDA2 3243FDSTAFloppy%Tack </td <td></td> <td>FD84</td> <td>224BFD</td> <td>SHLD</td> <td>Floppy\$Next\$Conti</td> <td>rol\$Location</td> <td>; back at main control byte</td>		FD84	224BFD	SHLD	Floppy\$Next\$Conti	rol\$Location	; back at main control byte
1586 1587 1587 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1589 1580 1589 1580 1589 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580 1580<							¥
1586FDBD 214000LXIH, DisktControlt8fActivate controller to perform1586FDSD 3680MVIH,80Hi operation.1590FDS 2C3F7FDJMPWaittFortBoisktCompletei operation.1591jisteration.i operation.1592ji operation.i operation.1593i operation.i operation.i operation.1594ji operation.i operation.1595WritetPortboisktCompletei operation.i operation.1597FDS 3602MVIA,FloppytWritetCodei operation.1598FDS 2602WVIA,FloppytReadtCodei operation.1599FDS 3801MVI A,FloppytReadtCodei operation.i operation.1601FDA 3801MVI A,FloppytReadtCodei operation.i operation.1601FDA 3801MVI A,FloppytReadtCodei operation.i operation.1603i operation.i operation.i operation.i operation.1604FDF 3AFAFBLDADisktTypei operation.i operation.1605FDF 3AFAFBLDADisktStypei operation.i operation.1606FDF 3AFAFBLDADisktStypei operation.i operation.1607FDF 3AFAFBLDADisktStypei operation.i operation.1606FDF 3AFAFBLDADisktStypei operation.i operation.1607FDF 3AFAFBLDADisktStypei operation.i operation. </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>Point controller at control table</td>							Point controller at control table
1588FDBD 214000LXIH,Disk@Control%9;Activate controller to perform1599FD92 CSF7FDJMPWait%For%Disk%Complete; operation.15911592;1592;;1593;;1594; correct sector.1595SEO2MVI1597FD95 3EO2MVI1597FD95 3EO2MVI1597FD95 3EO2MVI1597FD95 3EO2MVI1597FD95 3EO2MVI1604FD95 3EO2MVI1605Read@Physical:; correct sector.1600Read@Physical:; finto disk buffer to1601FD9A 3EO1MVI1604FD9C 3240FDSTA1605;Common%Physical:1606FD9C 3240FDSTA1607FD9F 3AFAFBLDA1608FDA2 FEO1CPI1609FDA2 SEO1MVI1611FDA2 3EFGBSTA1612FDA2 SEC1CPI1611FDA2 3EFGBSTA1612FDA2 SEC4; Set up disk control table1613Correct%Disk%Type:; Set up disk control table1614FDB5 3243FDSTA1615FDB5 3243FDSTA1616FDB5 3243FDSTA1617FDB5 3243FD1618FDB5 3243FD1619FDB5 3243FD1614FDB5 3243FD1615FDB5 3243FD1616FDB5 3243FD1617F		FDBA	224100	SHLD	Command\$Block\$8		
1569FD90 3660MVIM.80H; operation.1590FD92 CSFFDUMPHait%For%Disk%Complete; operation.1591;;1592;1593;1594;1595Write%Physical:; forrect sector.1596FD97 C39CFDUMP1697CSPCTUMP1699FD97 C39CFDUMP1690Read@Physical:; foo to common code1690Read@Physical:; foo to common code1601FD9A 3E01MVI1602;1603Common%Physical:; Get read function code1604; Common%Physical:; Common%Physical:1605TA1606;Correct%Disk%Type1607FD9F 3AFAFBLDA1608FDA2 FEOICPI1609FDA2 FEOICPI1609FDA2 SEOISTA1609FDA2 SEOIBISk%Error%Flag1612FDAD 3AE4FBLDA1614FDB2 3241FDSTA1615FDB2 3241FDSTA1616FDB2 3243FD1617FDB2 3243FDSTA1620FDB2 3243FD1621FDBC 04001622FDBC 04001623FDBC 04001624FDBC 04001625FDBC 04001626FDB2 3243FD1627FDBC 04001628FDBC 04001629FDB2 3243FD1629FDB2 3243FD <t< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td>;</td></t<>							;
1500FD92 CSF7FDJMPWait@For%Disk@Complete15711572;1572;1573;1594;1595Write@Physical:;1597FD95 3E02HVI A,Floppy%Write%Code1597FD95 3E02HVI A,Floppy%Write%Code1597FD95 3E02HVI A,Floppy%Write%Code1597FD95 3E01HVI A,Floppy%Read%Code1600FD97 3201HVI A,Floppy%Read%Code1601FD92 3240FDSTA1602FD92 3240FDSTA1606FD97 3260FBLDA1607FD97 3260FBJZ1608FD42 FE01CPT1608FD42 SE01HVI A,I1608FD42 SE01UZ1609FD4 CAADFDJZ1610FD47 3E01MU A,I1611FD47 3250FBSTA1612FD42 C59RET1613Correct%Disk%Type:; Set up disk control table1614FD80 264FBLDA1615FD80 3245FBLDA1616FD80 3245FBLDA1617FD85 2425FBLDA1618FD85 2425FBLDA1619FD85 2425FB1620FD85 2425FB1621FD80 64001622FD80 C6001623FD201624FD80 547FB1625FD80 C6001626FD80 547FB1627FD80 C6001628FD80 5001629FD80 701629 <td< td=""><td>1588</td><td></td><td></td><td></td><td></td><td></td><td></td></td<>	1588						
1500FD92 CSF7FDJMPWait&For&Disk&Complete15911592;1593;1594;1595HVI A,Floppy&Write&Code correct sector.1597FD95 3E02HVI A,Floppy&Write&Code fourite function code1597FD95 3E02HVI A,Floppy&Fkatle fourite function code1597FD95 3E02HVI A,Floppy&Read&Code fourite function code1600Read&Hrysical:; fourite function code1601FD9A 3E01HVI A,Floppy&Command Set command table1602;Common%Physical:; foet disk type (set in SELDSK)1604FD9C 3240FDSTAFloppy%Command1605FD9F 3AFAFBLDADisk%Type1606FDA2 SE01QZCorrect&Disk%Type1607FD9F 3AFAFBLDADisk%Type1608FDA2 SAFAFBLDADisk%Error%Flag1610FDA7 3E01MVI A,I; No. indicate disk error1611FDA9 32F6FBSTADisk%Error%Flag1612FDAC C9RET; Set up disk control table1613Correct%Disk%Type:; Set up disk controller1614FDB5 24ESFBLHLDInsBuffer%Disk1615FDB5 24ESFBLHLDInsBuffer%Track1626FDB5 70HOVA,L1627FDB5 0400HVI1628FDB5 342F7BLDA1629FDB5 3245FBLDA1620FDB5 24ESFBLDA1621	1589	FD90	3680				; operation.
1591       ;         1592       ;         1593       ;         1594       Write@Physical:         1595       JWP         1596       FD57 3E02         1597       FD57 3E02         1598       FD57 C39CFD         1599       FD57 C39CFD         1599       FD57 C39CFD         1600       JWP Common@Physical         1600       FD57 3E02         1601       FD9A 3E01         1602       ;         1603       Common%Physical:         1604       FD9C 3240FD         1605       Comon%Physical:         1606       FD9C 3240FD         1607       FD9C 3240FD         1608       FD42 FE01         1609       CPI         1600       FD42 FE01         1601       FD42 SE0FD         1602       PD7         1603       CPI         1604       FD42 FE01         1605       CPI         1606       FD42 FE01         1607       FD42 FE01         1618       FD42 SE0FB         1614       FD42 SE0FB         1615       FD42 SE01		FD92	C3F7FD	JMP	Wait\$For\$Disk\$Com	mplete	
1992       ;         1993       ;         1994       ;         1995       Write\$Physical:       ;         1996       ;       correct sector.         1997       FD95 3E02       MVI A.Floppy\$Write\$Code       ; Get write function code         1997       FD95 3E02       MVI A.Floppy\$Write\$Code       ; Get write function code         1999       Read\$Physical:       ; Get read function code         1600       ; into disk buffer.       ; into disk buffer.         1601       FD9A 3E01       MVI A.Floppy\$Command       ; Set command table         1605       ; Correct\$Disk\$Type       ; Get disk type (set in SELDSK)         1606       ; Correct\$Disk\$Type       ; Get disk type (set in SELDSK)         1606       FD9C 3240FD       JSZ Correct\$Disk\$Type       ; Ves         1606       FD4 3E01       MVI A.1       ; No, indicate disk error         1616       FDA7 3E01       MVI A.1       ; No, indicate disk control table         1617       FDA0 3AE4FB       LDA       In\$Buffer\$Disk       ; Convert disk number to 0 or 1         1618       FDA0 3AE4FB       LDA       In\$Buffer\$Track       ; For disk control table         1617       FDB5 2425FB       LHLD       In\$Buffer\$Track <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>							
1993       ;         1994       Write\$Physical;       ; correct sector.         1995				;			
1594 1595 1596Write\$Physical::Write contents of disk buffer to ; correct sector.1597FD97 C39CFDJMPCommon Physical (Go to common code ; Go to common code ; Go to common code1599Read#Physical:; Go to common code ; Go to common code1600Read#Physical: (Common*Physical:; Get read function code ; into disk buffer.1601FD9C 3240FDSTAFloppy%Read%Code ; Get disk type (set in SELDSK) ; Confirm it is a 5 1/4" Floppy1605; (Common*Physical:; Get disk type (set in SELDSK) ; Confirm it is a 5 1/4" Floppy1606; (Common*Physical:; Set command table1607FD9C 3AFAFB (Common*Physical:; No, indicate disk error1608FDA2 FE01CPIFloppy%T (Correct%Disk%Type)1609FDA4 CAAPFD (ACAPFDJZCorrect%Disk%Type ; Set up disk control table ; for disk control table ; for disk controller1611FDA2 S26FB (EA1)LDAIn%Buffer%Disk ; Convert disk number to 0 or 1 ; for disk controller1613FDB5 2AESFB (Correct%Disk%Type; ; FDB5 2AESFBLHLDIn%Buffer%Track ; Set up track number ; hod: This is single byte value ; for the controller.1622: ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; <b< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></b<>							
1995Write@Physical::Write contents of disk buffer to ; correct sector.1976::Correct sector.1977FD95 3E02MVIA,Floppy%brite@Code ; Read previously selected sector ; into disk buffer.1979:::1970:::1970:::1970:::1970:::1971:::1972:::1974:::1975:::1974:::1975:::1974:::1975:::1974:::1974:::1975:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977:::1977<							
<pre>FD95 3E02 HVI A_Floppy\$Write\$Code IS99 FD97 C39CFD UMP Common\$Physical Read\$Physical: IG4 write function code iG5 to common code iG6 to commo</pre>				Write\$Phys	ical:	;Wri	te contents of disk buffer to
1995FD952602MVIA,Floppy&Write\$Code:Get unit f unction code1996FD97 C39CFDJPP Common\$Physical::Read previously selected sector1600into disk buffer.1601FD9A 3E01MVIA,Floppy&Read\$Code:Get read function code1602:into disk buffer.1603:Common\$Physical::Get command table1604:Common\$Physical::Get disk type (set in SELDSK)1605::Get disk type (set in SELDSK)1606::Get disk type (set in SELDSK)1607:DP7 3AFAFB:DA1608FDA2 FE01CPI Floppy*: rest1609:FDA4 CAAPFD:Correct*Disk*Type1610FDA7 38C1MVI1611FDA7 38C1MVI1611FDA7 38C1MVI1612FDA2 SE4FBSTA1613Correct*Disk*Type::Set up disk control table1614:Correct*Disk*Type::Convert disk number to 0 or 11614:Correct*Disk*Type::Convert disk number1614:Correct*Disk*Type::Set up track number1614:DDAIn*Buffer*Track1615:FDB5 2AESFBLHLD1614:Star 0:Star 01615:STAFloppy*Track1616:Star 0:Star 01617:Star 0:Star 01618:Star 01620:STA1619:Star 01619:Star 01611:Star 01611:Star 0<							
1596FD97C39CFDUMPCommon\$Physical:Go to common code1599Read\$Physical::FRead previously selected sector:into disk buffer.1601FD9A 3E01HVIA, Floppy\$Read\$Code:Get common code1603Common\$Physical::Set command table1604FD9C 3240FDSTAFloppy\$Command:Set command table1605;		cnos	2502	MUT	A EloppyskiritesC		
1599Read#Physical:;Read#Physical:;Into disk buffer.1600FD9A 3E01MVIA,Floppy%Read%Code; into disk buffer.1603Common\$Physical:;;Get read function code1604FD9C 3240FDSTAFloppy%Command;Set command table1605jDaFD9C 34FAFB;Da1606r;;Get disk type (set in SELDSK)1607FD97 3AFAFB;DaDisk%Type;Get disk type (set in SELDSK)1608FDA2 FEO1CPT Floppy%S;Foet disk type (set in SELDSK)1609FDA4 CAADFDJZCorrect%Disk%Type;Yes1610FDA7 3E01MVIA,1;No, indicate disk error1611FDA9 32FGFBSTADisk%Error%Flag;Set up disk control table1614fDA5 324FDSTAFloppy%Unit;for disk controller1614FDB5 2AESFBLHLDIn%Buffer%Disk;Set up track number1614FDB5 2AESFBLHLDIn%Buffer%Track;Set up track number1620FDB8 7DHOVA,L;Note: This is single byte value1621FDB5 2AESFBLHLDIn%Buffer%Track;Set up track number1622FDB8 7DHOVA,L;Note: This is single byte value1623FDB9 32AFFBLDAIn%Buffer%Sector;Det must be converted into a1624FDB9 2AESFBLHLDIn%Buffer%Sector;Set up track number1625FDB8 7DHAIn%Buffer%Sector;Set head 01626							
1600; into disk buffer.1601FD9A 3E01WVIA, Floppy%Read%Code;Get read function code1602;;Get read function code;Get read function code1603Common%Physical:;Set command table1604FD9C 3240FDSTAFloppy%Command;Set command table1605;Disk%Type;Get disk type (set in SELDSK)1606FD9F 3AFAFBLDADisk%Type;Set command table1607FD9F 3AFAFBLDADisk%Type;Yes1608FDA4 CAADFDJZCorrect%Disk%Type;Yes1610FDA7 3E01MVIA,1;No, indicate disk error1611FDA9 322F6FBSTADisk%Trpe:;Set up disk control table1612FDA0 3AE4FBLDAIn%Buffer%Disk;Convert disk number to 0 or 11616FDB0 E601ANI;for disk controller1617FDB2 3241FDSTAFloppy%Unit;for the controller1618FDB5 2AE5FBLHLDIn%Buffer%Track;for the controller1620FDB8 7DMOVA,L;for the controller1621FDB9 3243FDSTAFloppy%Track;for the controller1622:::sector must be converted into a1623:::::1624:::::1625:::::1626:::::1627::::		F 977	CSFCFD				
1601FD9A 3E01MVIA,Floppy%Read%Code;Get read function code1603Common%Physical::1604FD9C 3240FDSTAFloppy%Command;Set command table1605STAFloppy%Command;Set command table1606	1399			Readernysi	Call	jnea	a previously selected sector
1602Common #Physical: Common #Physical: Common #Physical: Common #Physical: Common #Physical: Common #Physical: Common #Physical: STA Floppy#Command;Set command table1603FD9C 3240FDSTA Floppy#Command Staffer;Set command table1604FD9F 3AFAFBLDADisk#Type IConfirm it is a 5 1/4" Floppy1608FDA4 FCADFDJZ Correct#Disk#Type: FDA7 3EO1Gorrect#Disk#Type: IConfirm it is a 5 1/4" Floppy1609FDA4 CAADFDJZ Correct#Disk#Type: FDA7 3EO1Gorrect#Disk#Type: IConvert disk number to 0 or 1 ; for disk control table1611FDAD 3AEAFB FDBE 201LDA ANIIn*Buffer*Disk ANI; Convert disk number to 0 or 1 ; for disk controller1617FDBD E601 ANIANI I ; for disk controller; Set up track number ; for disk controller.1618FDB5 2AESFB LHLDLHLD In*Buffer*Track ; Set up track number ; for the controller.; Set on were not sector number.1622FDB8 7D ; BO1 4F HD8HUA HOV AL; For the controller.; ; Sectors 0 - 8 are head 0, 9 - 17 ; are head 1 ; Assume head 01628FDBE 3AEFFB ; FDBC 0600HVI HOV ; C,AHead%0 ; How, modify sector number ; Set to head 1 ; Set to head 1 ; Set to head 11636FDC9 4F ; HOV ; HOV ; ABHEAC ; Set to head 1 ; Set to head 11637FDC9 4F ; HOV ; ABHOV ; AB; Set head number1638FDC9 4F ; HOV ; HOV ; Head%0;; Set head number	1800						
1603Common\$Physical:1604FD9C 3240FDSTAFloppy\$Command;Set command table1605;1606;1607FD9F 3AFAFBLDADisk\$Type;Get disk type (set in SELDSK)1608FDA2 FE01CPIFloppy\$S;Confirm it is a 5 1/4" Floppy1609FDA4 CAADFDJZCorrect\$Disk\$Type;Yes1610FDA9 32F6FBSTADisk\$Error\$Flag;No, indicate disk error1611FDA9 32F6FBLDAIn\$sk\$Error\$Flag;Set up disk control table1612FDA0 3AE4FBLDAIn\$suffer\$Disk;Convert disk number to 0 or 11614FDB0 E601ANI;for disk control table1615FDAD 3AE4FBLDAIn\$suffer\$Disk;Convert disk number to 0 or 11616FDB0 E601ANI;for disk control error1617FDB5 2AE5FBLHLDIn\$suffer\$Track;Set up track number1620FDB6 7DMOVA,L;Note: This is single byte value1621FDB7TMY;1622STAFloppy\$Track; for the controller.1623;Sectors 0 - 8 are head 0, 9 - 17;1624;are head 1;1625;;;1626;;;1627FDBC 0600MVIB,O;1628FDE2 AFCPBLDAIn\$suffer\$Sector;1629FDC4 DAC6PFDJC;;1629FD		FUYA	3E01	₩VI	A, Floppyskeadslo	oe ;uet	read function code
1604FD9C 3240FDSTAFloppy\$Command;Set command table1605;1606;1607FD9F 3AFAFBLDADisk\$Type;Get disk type (set in SELDSK)1608FD42 FE01CPIFloppy\$5;Confirm it is a 5 1/4" Floppy1609FD44 CAADFDJZCorrect\$Disk\$Type;Yes1610FD47 3E01MVIA,1;No, indicate disk error1611FDA7 3E01MVIA,1;No, indicate disk error1611FDA7 3E01MVIA,1;Set up disk control table1612FDA7 4E01ANI;Set up disk control table1613Correct\$Disk\$Type1;Set up track number to 0 or 11614;for disk controller1615FDB2 3241FDSTA1620FDB8 7DMOV1620FDB8 7DMOV1621FDB2 3243FD1622FDB8 7D1623FDB5 3243FD1624intsBuffer\$Track1625;for the controller.1626;1627FDBC 0600MVIB,01628;fDE 04001629FDBE 34E7FB1629LDA1629FDE 04001629FDE 04001629 <td></td> <td></td> <td></td> <td>1</td> <td></td> <td></td> <td></td>				1			
1605;jj1607FD9F 3AFAFBLDADisk\$TypejGet disk type (set in SELDSK)1608FDA2 FE01CPIFloppy\$5;Confirm it is a 5 1/4" Floppy1609FDA4 CAADFDJZCorrect\$Disk\$Type;Ves1610FDA7 3801MVIA.1;No, indicate disk error1611FDA7 3264FBSTADisk\$Frror\$Flag1612FDA2 C2RET;Set up disk control table1613Correct\$Disk\$Type;Set up disk control table1614ii;for disk controller1615FDAD 3AE4FBLDAIn\$Buffer\$Disk1616FDB0 E601ANIi1617FDB2 3241FDSTAFloppy\$Unit1620FDB8 7DMOVA.L1621FDB5 2AESFBLHLDIn\$Buffer\$Track1622FDB8 7DMOVA.L1623FDB8 7DMOV1624isectors 0 - 8 are head 0, 9 - 171625ijAEYFB1627FDBC 0600MVI1628FDE SAE7FBLDA1629ID4In\$Buffer\$Sector1627FDBC 0600MVI1628FDE2 SAEFFDLDA1629SAE7FBLDA1629SAE7FBLDA1629FDC3 4FFMOV1629CA1629SAE7FB1631FDC4 DACBFD1632CO1633FDC4 DACBFD1634ID41635FDC4 ACBFD <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>							
1606;Joet disk typeJoet disk type (set in SELDSK)1609FDA2 FE01CPIFloppy\$5:Confirm it is a 5 1/4" Floppy1609FDA7 3E01MVIA,1;No, indicate disk error1611FDA7 3E01MVIA,1;No, indicate disk error1611FDA7 3E01MVIA,1;No, indicate disk error1611FDA7 3E01MVIA,1;No, indicate disk error1611FDA7 3E01MVIA,1;No, indicate disk error1613Correct\$Disk\$Type:;Set up disk control table;1614isfor disk controlleris1615FDAD 3AE4FBLDAIn\$Buffer\$Disk;Convert disk number to 0 or 11616FDBS 2AE5FBLHLDIn\$Buffer\$Track;Set up track number1620FDB8 7DMOVA,L;Note: This is single byte value1621FDB5 2AE5FBLHLDIn\$Buffer\$Track;for the controller.1622if the sector must be converted into aif the sector number.1623:sectors 0 - 8 are head 0, 9 - 17; are head 11624:sectors 0 - 8 are head 0, 9 - 17; are head 11625:sectors 0 - 8 are head 0, 9 - 17; are head 01626FDC 4AE7FBLDAin\$Buffer\$Sector; Get physical sector number.1627FDE 0400MVIB,0; are head 01628FDE 3AE7FBLDAin\$Buffer\$Sector; Get physical sector number1629FDC4 4FMOVC,A; Sectors 0 - 8 ares		FD9C	3240FD	STA	Floppy\$Command	;Set	command table
1607FD9FSAFAFBLDADisk#TypefGet disk type (set in SELDSK)1608FDA2FE01CPIFloppy\$5fConfirm it is a 5 1/4" Floppy1619FDA4CAADFDJZCorrect%Disk#TypefYes1611FDA732F0FBSTADisk#Error%Flag1611FDA7SZF6FBSTADisk#Error%Flag1612FDACC9RETfSet up disk control table1613fDA7SZ4FFBLDAIn%Buffer%Disk1614fDB0E601ANIifor disk control table1615FDADSAE4FBLDAIn%Buffer%Diskfor disk control table1616fDB0E601ANIifor disk control table1617FDB2SZ45FBLHLDIn%Buffer%TrackfSet up track number1620FDB870MQVA,LfNote: This is single byte value1621FDB9SZ45FDSTAFloppy%TrackfThe sector must be converted into a1622fDB83AE7FBLDAIn%Buffer%SectorfGet physical sector number1623fDE0GOOMVIB,0fAssume head 01624fDC4fSet up track if set is head 0fSet up track if set is head 01625fDE0GOOMVIB,0fAssume head 01626fDE0GOOMVIB,0fAssume head 01627FDBCGOOMVIB,0fSet opy in case it is head 01628FDC4fDC4fDC4 </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>							
1608FDA2 FE01CPIFloppy55: Confirm it is a 5 1/4" Floppy1609FDA4 CAADFDJZCorrect\$Disk\$Type:Yes1610FDA7 3E01MVIA.1:No, indicate disk error1611FDA7 3E01MVIA.1:No, indicate disk error1611FDAC C9RET:Set up disk control table1613Correct\$Disk\$Type::Set up disk control table1614:Convert disk number to 0 or 11615FDAD 3AE4FBLDA1617FDB2 3241FDSTA1618FDB5 2AE5FBLHLD1619FDB5 2AE5FBLHLD1620FDB8 7DMOV1621FDB9 3243FDSTA1622iThe sector must be converted into a1623:Sectors 0 - 8 are head 01624:Sectors 0 - 8 are head 01625:Sectors 0 - 8 are head 01626:Sectors 0 - 8 are head 01627FDBC 0600MVI1628FDC1 4F1629FDC1 4F1629FDC2 FE091631FDC4 4F1632SUC1633FDC4 04173SUC1634FDC9 4F1635FDC4 041636FDC9 4F1637FDC8 781636Head\$0:1637FDC8 781636Head\$0:1637FDC8 781638FDC9 4F1637FDC8 781636FDC8 781637FDC8 781638FDC6 78	1606			;			
1609FDA4CADFDJzCorrect\$Disk\$Type;Yes1610FDA73E01MVIA,1;No, indicate disk error1611FDA932F6FBSTADisk\$Error\$Flag1612FDACCorrect\$Disk\$Type:;Set up disk control table1613Correct\$Disk\$Type:;Set up disk control table1614iffor disk control table1615FDAD3AE4FBLDA1616FDB0EG01ANI1617FDB23241FDSTA1618FDB5ZAE5FBLHLD1619FDB52AE5FBLHLD1621FDB7STA1622FDB7MOV1621FDB7STA1622FDB7STA1623FDB7STA1624iThe sector must be converted into a1625iFDB2jAesume head 01626FDBEAAE7FB1627FDBC 0600MVI1628FDB2AAE7FB1629FDB2AAE7FB1629FDC1JC1630FDC2FE091631FDC4JC1632FDC41633FDC41634FDC91634FDC91635FDCA 04174JA1636FDC41637FDCB 781638FDC41637FDCB 781638FDC41637FDCB 781638FDC31637FDCB							
<pre>1610 FDA7 3EO1 HVI A,1 ;No, indicate disk error 1611 FDA9 32F6FB STA Disk%Error%Flag 1612 FDAC C9 RET 1613 Correct%Disk%Type: ;Set up disk control table 1614 ; 1615 FDAD 3AE4FB LDA In%Buffer%Disk ;Convert disk number to 0 or 1 1616 FDB0 26O1 ANI 1 ; for disk controller 1617 FDB2 3241FD STA Floppy%Unit 1618 1619 FDB5 2AE5FB LHLD In%Buffer%Track ;Set up track number 1620 FDB8 7D MOV A,L ;Note: This is single byte value 1621 FDB9 3243FD STA Floppy%Track ; for the controller. 1623 ; head number and sector number. 1624 ; head number and sector number. 1625 ; sectors 0 - 8 are head 0, 9 - 17 1626 ; are head 1 1627 FDBC 0600 MVI B,0 ;Assume head 0 1628 FDBE 3AE7FB LDA In%Buffer%Sector ; Get physical sector number 1629 FDC1 4F MOV C,A ; Save copy in case it is head 0 1630 FDC2 FE09 CPI 9 ; fOncek if &lt;9 1631 FDC4 DAC8FD JC Head%0 ; Yes it is &lt;9 1632 fDC7 D609 SUI 9 ; No, modify sector number back 1634 FDC9 4F MOV C,A ; Put sector in B 1635 fDC4 04 INR B ;Set to head 1 1636 Head%0; ;Set to head 1 1637 FDC 76 FDC 78 MOV A,B ;Set to head 1 1636 Head%0; ;Set to head 1 1636 FDC3 75 DC4 78 MOV A,B ;Set to head 1 1636 Head%0; ;Set to head 1 1636 FDC5 75 DC4 78 MOV A,B ;Set to head 1 1636 Head%0; ;Set to head 1 1636 Head%0; ;Set head number ;Set head numb</pre>	1608	FDA2	FE01	CPI	Floppy\$5	\$Con	firm it is a 5 1/4" Floppy
1611       FDA9 32F6FB       STA       Disk#Error#Flag         1612       FDAC C9       RET       ;Set up disk control table         1613       Correct#Disk#Type:       ;Set up disk control table         1614       ;       for disk number to 0 or 1         1615       FDAD 3AE4FB       LDA       In#Buffer#Disk       ;Convert disk number to 0 or 1         1616       FDB2 3241FD       STA       Floppy#Unit       ;for disk controller         1618       In#Buffer#Track       ;Set up track number       ;set op track number         1620       FDB5 7D       MOV       AL       ;Note: This is single byte value         1621       FDB9 3243FD       STA       Floppy#Track       ; for the controller.         1622       ;       ;       sector must be converted into a         1623       ;       head number and sector number.         1624       ;       sectors 0 - 8 are head 0, 9 - 17         1625       ;       sector number         1626       ;       sector or number         1627       FDBE 0600       MVI       B,0         1628       FDC1 4F       MOV       C,A         1629       FDC1 4F       MOV       C,A         1630	1609	FDA4	CAADFD	JZ	Correct\$Disk\$Type	e ;Yes	
1612       FDAC C9       RET       Correct\$Disk\$Type:       ;Set up disk control table         1613       Correct\$Disk\$Type:       ;Set up disk control table         1614       i:       ;Convert disk number to 0 or 1         1616       FDB0 E601       ANI       i         1617       FDB2 3241FD       STA       Floppy\$Unit         1618       FDB5 2AE5FB       LHLD       In\$Buffer\$Track       ;Set up track number         1620       FDB8 7D       MOV       A,L       ;Note: This is single byte value         1621       FDB9 3243FD       STA       Floppy\$Track       ; for the controller.         1622       FDBC 6600       MVI       B,0       ;Assume head 0         1624       ;       sector must be converted into a       ;         1625       ;       sectors 0 - 8 are head 0, 9 - 17       ;         1626       ;       sectors 0 - 8 are head 0, 9 - 17       ;         1627       FDBC 0600       MVI       B,0       ;       ;         1628       FDBE 3AE7FB       LDA       In\$Buffer\$Sector       ;       Get physical sector number         1629       FDC1 4F       MOV       C,A       ;       Yes it is < 9	1610	FDA7	3E01	MVI	A, 1	;No,	indicate disk error
1612       FDAC C9       RET         1613       Correct\$Disk\$Type:       ;Set up disk control table         1614       ;Convert disk number to 0 or 1         1615       FDAD 3AE4FB       LDA       In\$Buffer\$Disk       ;Convert disk number to 0 or 1         1616       ;FDBC E601       ANI       i       ;for disk controller         1617       FDB2 3241FD       STA       Floppy\$Unit         1618       ;FDB5 2AE5FB       LHLD       In\$Buffer\$Track       ;Set up track number         1621       FDB9 3243FD       STA       Floppy\$Track       ; for the controller.         1622       ;       ;The sector must be converted into a       ;         1623       ;       insbuffer\$Sector       ; Bet number and sector number.         1624       ;       ; Sectors 0 - 8 are head 0, 9 - 17         1625       ; are head 1       ; Assume head 0         1626       ; BOB 3AE7FB       LDA       In\$Buffer\$Sector         1627       FDBC 0600       MVI       B,0       ; Save copy in case it is head 0         1628       FDC1 4F       MOV       C,A       ; Save copy in case it is head 0         1629       FDC4 AFCBFD       JC       Head\$0       ; Yes it is < 9	1611	FDA9	32F6FB	STA	Disk\$Error\$Flag		
1613Correct\$Disk\$Type:;Set up disk control table1614;1615FDAD 3AE4FBLDA1616FDBC 54011617FDB2 3241FD1618STA1619FDB5 2AE5FBLHLD1618In\$Buffer\$Track1620FDB9 3243FD1621FDB9 3243FD1622STA1623FDB9 3243FD1624iThe sector must be converted into a1625;1626;1627FDBC 06001628MVI1629BB6 7D1626;1627FDBC 06001628MVI1629BB7D1626;1627FDBC 06001628MVV629FDC1 4F1629MOV1630FDC2 FE091631FDC4 PACBFD1632FDC7 D609SUI91633FDC4 041634FDC9 4F1635MOV1636Head\$0;1637FDC8 781633FDC8 781634FDC9 4F1635STA1636Head\$0;1637FDCB 781638FDCB 781639FDCB 781638FDCB 781638FDCB 781639FDCB 781639FDCB 781639FDCB 781639FDCB 781639FDCB 781639FDCB 781639FDCB				RET			
1615       FDAD 3AE4FB       LDA       InsBuffer\$Disk       ;Convert disk number to 0 or 1         1616       FDB0 E601       ANI       1       ; for disk controller         1617       FDB2 3241FD       STA       Floppy\$Unit       ;         1618				Correct\$Di	sk\$Type:	;Set	up disk control table
1615       FDAD 3AE4FB       LDA       InsBuffer\$Disk       ;Convert disk number to 0 or 1         1616       FDB0 E601       ANI       1       ; for disk controller         1617       FDB2 3241FD       STA       Floppy\$Unit       ;         1618						;	
1616       FDB0 E601       ANI       1       ; for disk controller         1617       FDB2 3241FD       STA       Floppy\$Unit       ; Set up track number         1618       1       iN\$Buffer\$Track       ; Set up track number         1620       FDB9 7D       MOV       A,L       ; Note: This is single byte value         1621       FDB9 3243FD       STA       Floppy\$Track       ; for the controller.         1622       ;       sector must be converted into a       ;         1623       ; head number and sector number.       ;         1624       ; are head 1       ; Assume head 0         1625       ; B,O       ; Set physical sector number         1626       ; B,O       ; Save copy in case it is head 0         1627       FDBC 0600       MVI       B,O       ; Save copy in case it is head 0         1628       FDEE 3AE7FB       LDA       In\$Buffer\$Sector       ; Get physical sector number         1629       FDC1 4F       MOV       C,A       ; Yes it is < 9		FDAD	3AE4FB	LDA	In\$Buffer\$Disk	; Con	vert disk number to 0 or 1
1617       FDB2 324iFD       STA       Floppy\$Unit         1618       1619       FDB5 2AE5FB       LHLD       In\$Buffer\$Track       ;Set up track number         1620       FDB8 7D       MOV       A,L       ;Note: This is single byte value         1621       FDB9 3243FD       STA       Floppy\$Track       ; for the controller.         1622       ;       for the controller.       ;         1623       ;       The sector must be converted into a       ;         1624       ;       sectors 0 - 8 are head 0, 9 - 17       ;         1625       ;       sectors 0 - 8 are head 0, 9 - 17       ;         1626       ;       are head 1       ;       ;         1627       FDBC 0600       MVI       B,0       ;       ;       are head 0         1626       ;       are head 0       ;       are head 0       ;       ;         1627       FDBC 0600       MVI       B,0       ;       ;       are head 0         1627       FDBC 14F       MOV       C,A       ;       Set copy in case it is head 0         1630       FDC2 FE09       CPI       9       ;       ;       ;         1631       FDC4 DAC8FD <t< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>							
161816191619162016201621162116221623162316241625162516261627162816291629162116231624162516261627162816291629162016201621162516261627162816291629162016301631162716311632163216331634163516341635163416351636163716371638163816341635163616371638163816341635163616371638163816381638163916391634163516351636163716381638163816391639163916391639163916391634163516351636163716381638 </td <td></td> <td></td> <td></td> <td></td> <td>Floppy\$Unit</td> <td></td> <td></td>					Floppy\$Unit		
1619FDB5 2AE5FBLHLDIn\$Buffer\$Track;Set up track number1620FDB8 7DMOVA,L;Note: This is single byte value1621FDB7 3243FDSTAFloppy\$Track;1623;The sector must be converted into a1624;head number and sector number.1625;sectors 0 - 8 are head 0, 9 - 171626;are head 11627FDBC 0600MVIB,01628;BBE 3AE7FBLDA1629FDC1 4FMOVC,A1630FDC2 FE09CPI91631FDC4 DACBFDJCHead\$01632FDC4 4FMOVC,A1633FDC4 94FMOVC,A1634FDC9 4FMOVC,A1635FDCA 04INRB1636FDCA 78MOV1637FDCB 78MOVA,B1638FDC3 78STA1634FDC8 78MOV1635FDCA 04STA1636FDC3 78STA1637FDCB 78MOV1638FDC3 78STA1639FDC3 78STA1639FDC3 78STA1639FDC3 78STA1639FDC3 78STA1639FDC3 78STA1639FDC3 78STA1639FDC3 78FLOPPy\$Head							
1620       FDB8 7D       MOV       A,L       ;Note: This is single byte value         1621       FDB9 3243FD       STA       Floppy\$Track       ; for the controller.         1623       ;The sector must be converted into a       ;         1624       ;The sector must be converted into a       ;         1625       ;Sectors 0 - 8 are head 0, 9 - 17       ;         1626       ;Sectors 0 - 8 are head 0, 9 - 17       ;         1627       FDBC 0600       MVI       B,0       ;Assume head 0         1628       FDBE 3AE7FB       LDA       In\$Buffer\$Sector       ;Get physical sector number         1629       FDC1 4F       MOV       C,A       ;Save copy in case it is head 0         1630       FDC2 FE09       CPI       9       ;Check if <9		ED85	2AE5EB	LHID	In\$Buffer\$Track	: Set	up track number
1621FDB93243FDSTAFloppy%Track; for the controller.1622;;iThe sector must be converted into a1623;head number and sector number.1624;sectors $0 - 8$ are head $0, 9 - 17$ 1625;sectors $0 - 8$ are head $0, 9 - 17$ 1626;are head $1$ 1627FDBC 0600MVI1628FDBE 3AE7FBLDA1629FDC1 4FMOV1630FDC2 FE09CPI9; Check if $< 9$ 1631FDC4 DACBFDJC1632Head%0; in the $0 - 8$ range.1633FDC9 4F1634FDC9 4F1635FDCA 041188; Set to head 11636Head%0;1637FDCB 781638FDCB 781639FDCB 781634FDCB 781635FDCA 041636FDCB 781637FDCB 781638FDCC 3242FD1639FDCC 3242FD1639FDCC 3242FD1639FLC 3242FD1639							
1622;1623;1623;1624;1625;1625;1626;1627FDBC 06001628FDBE 3AE7FBLDAIn\$Buffer\$Sector1629FDC1 4FMOVC,A1630FDC2 FE09CPI91631FDC4 FEMOVC,A1632FDC7 D609SUI91633FDC7 D609SUI91634FDC9 4FMOVC,A1635FDC4 041636Head\$0:1637FDC8 78MOVA,B1638FDCC 3242FDSTAFloppy\$Head							
1623;The sector must be converted into a1624; head number and sector number.1625; Sectors $O - B$ are head $O, 9 - 17$ 1626; are head 11627; are head 01628FDBC 06001628FDBE 3AE7FBLDAIn\$Buffer\$Sector1629FDC1 4FMOVC,A1630FDC2 FE09CPC2FE09CPC3JC1631FDC4 DACBFDJCHead\$01632FDC7 D609SUI91633FDC9 4FMOVC,A; in the 0 - 8 range.1634FDC9 4FMOVC,A; Set to head 11636Head\$0;1637FDCB 78MOVA,B; Set head number1638FDCC 3242FDSTAFloppy\$Head		r 007	34 <b>3</b> 0° D	ern.	· LOPPY #11 BCK	, 10	
1624       ; head number and sector number.         1625       ; Sectors 0 - 8 are head 0, 9 - 17         1626       ; are head 1         1627       FDBC 0600       MVI         1628       FDBE 3AE7FB       LDA         1629       FDBE 3AE7FB       LDA         1629       FDC1 4F       MOV         1630       FDC2 FE09       CPI         1631       FDC4 DACBFD       JC         1632       FDC4 DACBFD       JC         1633       FDC4 DACBFD       JC         1633       FDC4 DACBFD       JC         1633       FDC4 DACBFD       JC         1633       FDC4 DACBFD       JC         1634       FDC4 PACBFD       JC         1635       FDCA 04       INR         1636       Head\$0:       ;         1637       FDCB 78       MOV         1638       FDCB 78       MOV         1639       FDCB 78       <						, , Tha	sector must be converted into a
1625       ; Sectors 0 - 8 are head 0, 9 - 17         1626       ; are head 1         1627       FDBC 0600       MVI       B,0       ; are head 1         1628       FDBE 3AE7FB       LDA       In\$Buffer\$Sector       ; Get physical sector number         1629       FDC1 4F       MOV       C,A       ; Save copy in case it is head 0         1630       FDC2 FE09       CPI       9       ; Check if < 9							
1626; are head 11627FDBC 0600MVIB,0; Assume head 01628FDBE 3AE7FBLDAIn\$Buffer\$SectorJGet physical sector number1629FDC1 4FMOVC,A; Save copy in case it is head 01630FDC2 FE09CPI9; Check if $< 9$ 1631FDC4 DACBFDJCHead\$0; Yes it is $< 9$ 1632FDC7 D609SUI9; No, modify sector number back1633FDC9#Yes it is $< 9$ 1634FDC9 4FMOVC,A; Fut sector in B1635FDCA 04INRB; Set to head 11636Head\$0:;Set to head 11637FDCB 78MOVA,B; Set head number1638FDCC 3242FDSTAFloppy\$Head							
1627       FDBC 0600       MVI       B,0       Assume head 0         1628       FDBE 3AE7FB       LDA       In\$Buffer\$Sector       ;Get physical sector number         1629       FDC1 4F       MOV       C,A       ;Save copy in case it is head 0         1630       FDC2 FE09       CPI       9       ;Check if < 9							
1628       FDBE 3AE7FB       LDA       In\$Buffer\$Sector       ;Get physical sector number         1629       FDC1 4F       MOV       C,A       ;Save copy in case it is head 0         1630       FDC2 FE09       CPI       9       ;Check if < 9			o./ 00	MUT	<b>B</b> 0		
1629       FDC1       4F       MOV       C,A       ;Save copy in case it is head 0         1630       FDC2       FEC9       CPI       9       ;Check if < 9							
1630       FDC2       FEC9       CPI       9       ;Check if < 9							
1631       FDC4       DACBFD       JC       Head\$0       ;Yes it is < 9					U, A	;sav	e copy in case it is nead o
1632       FDC7       D609       SUI       9       ;No, modify sector number back         1633       ; in the 0 - 8 range.         1634       FDC9       4F       MOV       C,A       ;Put sector in B         1635       FDCA       04       INR       B       ;Set to head 1         1636       Head\$0:       :       :       :Set to head 1         1637       FDCB       78       MOV       A,B       ;Set head number         1638       FDCC       3242FD       STA       Floppy\$Head						; Che	CK 11 N 7
1633       ; in the 0 - 8 range.         1634       FDC9 4F       MOV       C,A       ;Put sector in B         1635       FDCA 04       INR       B       ;Set to head 1         1636       Head\$0;						;Yes	IT 15 N 7
1634 FDC9 4F MOV C,A ;Put sector in B 1635 FDCA 04 INR B ;Set to head 1 1636 Head\$0: 1637 FDCB 78 MOV A,B ;Set head number 1638 FDCC 3242FD STA Floppy\$Head		FDC7	U609	SUI	У		
1635 FDCA 04 INR B ;Set to head 1 1636 Head\$0: 1637 FDCB 78 MOV A,B ;Set head number 1638 FDCC 3242FD STA Floppy\$Head							
1636 Head\$0; 1637 FDCB 78 MOV A,B ;Set head number 1638 FDCC 3242FD STA Floppy\$Head					C, A		
1637 FDCB 78 MOV A, B ;Set head number 1638 FDCC 3242FD STA Floppy\$Head		FDCA	04		в	;Set	to head 1
1638 FDCC 3242FD STA Floppy\$Head						_	
						; Set	head number
	1638						
				MOV	A,C	;Set	sector number

Figure 6-4. (Continued)

1440	FDDO 3C	INR		; (Physical sectors start at 1)
1641	FDD1 3244FD	STA	Floppy#Sector	; (physical sectors start at 1)
1642		014	1 TOPP / Dector	
1643	FDD4 210002	LXI	H,Physical#Sector#Size	Set byte count
1644	FDD4 210002 FDD7 2245FD	SHLD	Floppy\$Byte\$Count	
1645				:
1646	FDDA 2133F6	LXI	H,Disk\$Buffer	Set transfer address to be
1647	FDDD 2247FD	SHLD	Floppy\$DMA\$Address	; disk buffer
1648				7
1649				As only one control table is in
1650				; use, close the status and busy
1651 1652				; chain pointers back to the
1653	FDE0 214300	LXI	H,Disk\$Status\$Block	; main control bytes.
1654	FDE3 2249FD	SHLD	Floppy\$Next\$Status\$Block	
1655	FDE6 214500	LXI	H, Disk\$Control\$5	ι κ
1656	FDE9 224BFD	SHLD	Floppy\$Next\$Control\$Loc	ation
1657				
1658	FDEC 2140FD	LXI	H,Floppy\$Command	<pre>set up command block pointer</pre>
1659	FDEF 224600	SHLD	Command\$Block\$5	
1660				
1661	FDF2 214500	LXI	H, Disk\$Contro1\$5	Activate 5 1/4" disk controller;
1662	FDF5 3680	MVI	M, 80H	
1663		; Linitterat	)isk\$Complete:	Wait until Disk Status Block indicates
1665		MET CALOLAT	A SKYWUNPIELEI	; operation complete, then check
1666				; if any errors occurred.
1667				;On entry HL -> disk control byte
1668	FDF7 7E	MOV	A, M	Get control byte
1669	FDF8 B7	ORA	A	
1670	FDF9 C2F7FD	JNZ	Wait\$For\$Disk\$Complete	;Operation still not yet done
1671				
1672 1673	FDFC 3A4300 FDFF FE80	LDA CPI	Bisk\$Status\$Block BOH	Complete now check status
1674	FEO1 DA09FE	JC	Disk\$Error	<pre>#Check if any errors occurred #Yes</pre>
1675	FEO4 AF	XRA	A	; res ; No
1676	FE05 32F6FB	STA	Disk\$Error\$Flag	Clear error flag
1677	FE08 C9	RET		,
1678		Disk\$Error		
1679	FE09 3E01	MVI	A, 1	şSet disk-error flag nonzero
1680	FEOB 32F6FB	STA	Disk\$Error\$Flag	
1681	FEOE C9	RET		
1682		;		
1684		;		
1685			ontrol table images for w	arm boot
1686		; , , , , , , , , , , , , , , , , , , ,		
1687		Boot\$Contr	ol\$Part\$1:	
1688	FEOF 01	DB	1	Read function
1689	FE10 00	DB	0	;Unit (drive) number
1690	FE11 00	DB	0	;Head number
1691	FE12 00	DB	0	;Track number
1692 1693	FE13 02 FE14 0010	DB DW	2	Starting sector number
1694	FE14 0010	DW	8×512 CCP\$Entry	;Number of bytes to read ;Read into this address
1695	FE18 4300	DW	Disk#Status#Block	Pointer to next status block
1696	FE1A 4500	DW	Disk\$Contro1\$5	Pointer to next control table
1697		Boot\$Contr		· · · · · · · · · · · · · · · · · · ·
1698	FE1C 01	DB	1	Read function
1699	FE1D 00	DB	0	;Unit (drive) number
1700	FEIE 01	DB	1	Head number
1701	FE1F 00	DB	0	;Track_number
1702	FE20 01 FE21 0006	DB DW	1 3*512	;Starting sector number ;Number of bytes to read
1704	FE23 00F0	DW	CCP\$Entry + (8*512)	;Read into this address
	FE25 4300	שמ	Disk#Status#Block	Pointer to next status block
1706	FE27 4500	DW	Disk\$Contro1\$5	Pointer to next control table
1707				
1708		7		
1709		;		
1710		;		
1711		WBOOT:	:Warm boot entr	
1712 1713			;On warm boot,	the CCP and BDOS must be reloaded
1714			; into memory,	In this BIOS, only the 5 1/4" 11 be used. Therefore this code
1			7 DISKETTES W1	II DE USED, INERETORE UNIS CODE
L				

Figure 6-4. (Continued)

1715 ; is hardware specific to the controller. Two 1716 ; prefabricated control tables are used. 1717 FE29 318000 SP.80H LXI 1718 FE2C 110FFE LXI D.Boot\$Control\$Part1 ;Execute first read of warm boot 1719 FE2F CD3BFE CALL Warm\$Boot\$Read ;Load drive 0, track 0, 1720 head 0, sectors 2 to 8 ; 1721 FE32 111CFE LXI D,Boot\$Control\$Part2 Execute second read 1722 FE35 CD3BFE 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 1727 Warm\$Boot\$Read: ;On entry, DE -> control table image This control table is moved into 1728 ; the main disk control table and 1729 then the controller activated. FE3B 2140FD FE3E 224600 1730 LXI ;HL -> actual control table ;Tell the controller its address ;Move the control table image H,Floppy\$Command 1731 SHLD Command\$Block\$5 1782 1733 into the control table itself 1734 FE41 OEOD MVI C, 13 ;Set byte count 1735 Warm\$Boot\$Move: FE43 1A 1736 LDAX n ;Get image byte 1737 FE44 77 MOV M.A Store into actual control table 1738 FE45 23 INX н ;Update pointers FE46 13 FE47 0D 1739 INX n 1740 DCR С ;Count down on byte count 1741 FE48 C243FE Warm\$Boot\$Move ;Continue until all bytes moved JNZ 1742 1743 FE4B 214500 LXI H, Disk\$Control\$5 Activate controller 1744 FE4E 3680 MVI M, 80H 1745 Wait\$For\$Boot\$Complete: 1746 FE50 7E MOV A,M ;Get status byte 1747 1748 FE51 B7 **NRA** ;Check if complete FE52 C250FE JNZ Wait\$For\$Boot\$Complete ;No 1749 ;Yes, check for errors 1750 FE55 3A4300 LDA Disk\$Status\$Block 1751 FE58 FE80 CPI 80H 1752 FESA DASEFE ĴĊ. Warm\$Boot\$Error :Yes, an error occurred 1753 FE5D C9 RET 1754 1755 Warm\$Boot\$Error: 1756 FE5E 2167FE LXI H,Warm\$Boot\$Error\$Message 1757 FE61 CD33F8 CALL Display#Message 1758 1759 FE64 C329FE JMP WBOOT ;Restart warm boot 1760 Warm\$Boot\$Error\$Message: 1761 1762 FE67 0D0A576172 CR,LF, 'Warm Boot Error - retrying...', CR, LF, O DB . 1763 ; 1764 FE89 END ;Of simple BIOS listing

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.

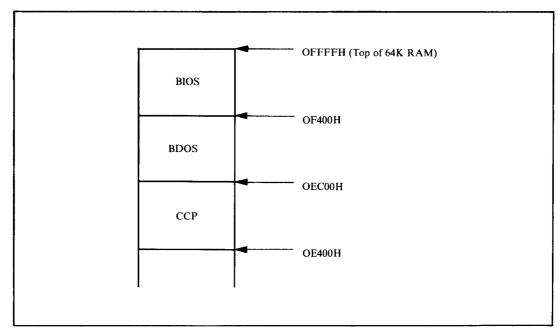


Figure 7-1. Memory layout of CP/M

# 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.

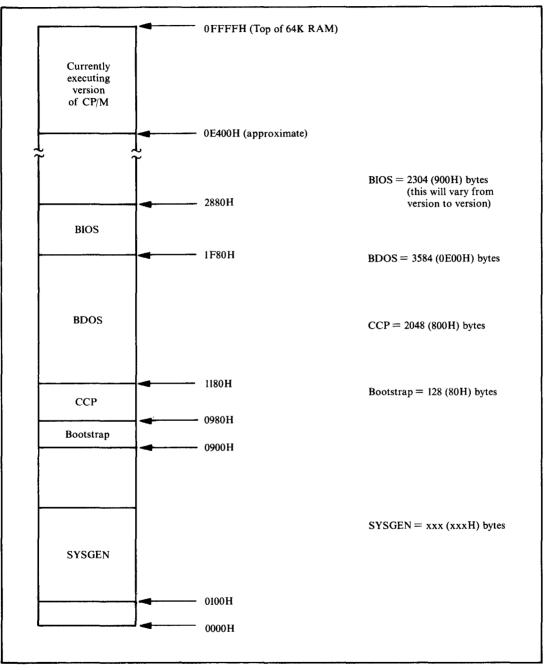


Figure 7-2. SYSGEN's memory layout

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><cr></u>

FUNCTION COMPLETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <u>B</u>

DESTINATION ON B: THEN TYPE RETURN <u><cr></u>

FUNCTION COMPLETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <u><cr></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.

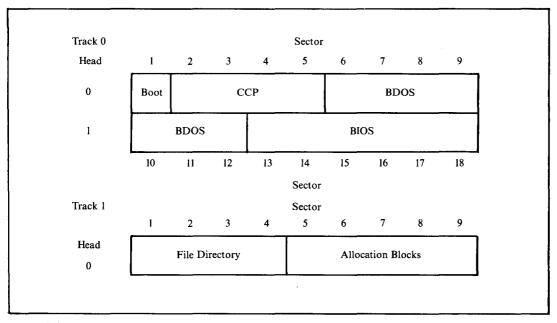


Figure 7-3. Disk layout for example BIOS on 5 1/4-inch diskettes

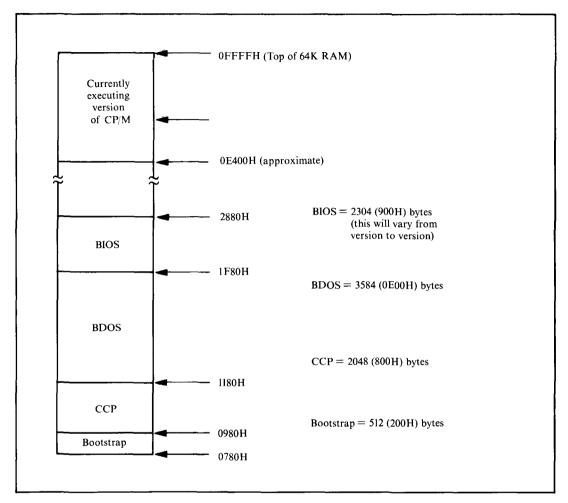


Figure 7-4. Addresses for example BIOS image

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.
                3
                2
                1
3130 =
                Version
                                 EQU
                                          1011
                                                  ;Equates used in the sign-on
                                                   ; message
3730 =
                Month
                                 EQU
                                          1071
3432 =
                                 EQU
                                          124
                Day
                                 EQU
                                          1821
3238 =
                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:
                                               Base Address
1F80H
                               Component
                                 BIOS
                                                   1180H
                                 BDOS
                                 CCP
                                                   0980H
                                 BOOTF5
                                                  0780H
                         The components are produced as follows:
                                 BIOS.HEX
                                                  By assembling source code
                                 BDOS )
CCP )
                                                  From a CPMnn.COM file output
                                                    by MOVCPM and SAVEd on disk
                2
                                 BOOTF5.HEX
                                                  By assembling source code
                1
                .
                        The components are pieced together using DDT with the
                ;
                        following commands:
                2
                                 DDT CPMnn.COM
                                 IPUTCPMF5.HEX
                                 R
                                                           (Reads in this program)
                                 IBOOTF5.HEX
                                 R680
                                                           (Reads in BOOT at 0780H)
                                 IBIOS.HEX
                                 R2980
                                                           (Reads in BIOS at 1F80H)
                2
                                                           (Exit from DDT)
                                 60
                :
                                 SAVE 40 PUTCPMF5.COM
                                                           (Create final .COM file)
                3
                :
                         The actual layout of the diskette is as follows:
                ;
                •
                ; Track O
                                                Sector
                           1
                                 2
                                       3
                                              Δ
                                                    5
                                                           6
                                                                 7
                                                                       8
                                                                              •
                ; Head
                                            ----
                                                  -+----
                                                               ------
                ;
                         0
                                   ----+----+-
                                              ----+----+----+----
                :
                                                                    --+----+----+
                   1
                         :===== BDOS ====>:<==============
                                                             BIOS ========>;
                ;
                ;
                                                  14
                                       12
                                                           15
                                                                       17
                                                                              18
                           10
                                 11
                                              13
                                                                 16
                - 2
                                                Sector
                .
                        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
0900 =
                                 EQU
                                          0900H
0200 =
                Boot$Length
                                 EQU
                                          512
0800 =
                CCP$Length
                                 EQU
                                          0800H
                                                  ;Constant
0E00 =
                BDOS$Length
                                 EQU
                                          OEOOH
                                                  ;Constant
1E00 =
                Length$In$Bytes EQU
                                          CCP$Length + BDOS$Length + BIOS$Length
0780 =
                .
StartsImage
                                 EQU
                                          980H - Boot$Length
                                                                   ;Address of CP/M image
2100 =
                Length#Image
                                          Length$In$Bytes + Boot$Length
                                 EQU
                .
```

Figure 7-5. Example PUTCPM

		,	Disk ch	aracteri	stics	
		7 7 7 7 7	the flop one sect	py disk tor to t	ette so i	the physical characteristics of hat the program can move from updating the track and resetting y.
0001	=	; First <b>\$</b> Se	ctor\$on!	Track	EQU	1
0012		Last\$Sec			EQU	18
0009		Last#Sec		lead\$0	EQU	9
0200	=	Sector\$5 7	ize		EQU	512
		; ;	Control	ler char	acteristi	cs
		, , , , ,	multiple to prode	sector	s in a si re genera	the floppy disk controller can write ngle command. However, in order I example it is shown only reading one
0001	=	; Sectors¶	Per\$Wri	te	EQU	i
		;				
		;	Cold boo	ot chara	cteristic	5
0000	=	; Start\$Tr	ack		EQU	0 ;Initial values for CP/M image
0001	=	Start\$Se	ctor		EQU	1 ;= " =
0011		Sectors ;	To\$Write	2	EQU	(Length\$Image + Sector\$Size - 1) / Sector\$Siz
0000		; B\$PRINTS		EQU	•	Print string terminated by \$
0009 0005		BERINIS	, ,	EQU	5	BDOS entry point
		;			-	··
		;				
0100		D	ORG	100H		
0100	C33F01	Put\$CPM:	JMP	Mai		- Fatan and a sale had
0100	L33F01		JMP	Main\$Co	0e	;Enter main code body ;For reasons of clarity, the main ; data structures are shown before the
0000	-	CR	EQU	ODH		; executable code. ;Carriage return
000A		LF	EQU	OAH		Line feed
		; Signon\$M				
0103	0D0A507574		DB	CR,LF,	Put CP/M	on Diskette'
	ODOA		DB	CR,LF		
	5665727369 3031	,	DB DW	'Version Version		
0125	20		DB	/ /		
0126	3037		DW	Month		
0128	2F		DB	11		
0129 012B	3234 25		DW DB	Day		
0120	3832		DW	Year		
	ODOA24		DB	CR,LF,	\$1	
		;	Disk c	ontrol t	ables	
0045	-	; Disk\$Cor	ntro1\$5	EQU	45H	;5 1/4" control byte
	-	Commands	Block\$5	EQU	46H	;Control table pointer
0046		Disk\$Sta ;	itus	EQU	43H	;Completion status
0046		;				and DMA\$Address can also be used
0046		7				
0046		7 7 7	as work	ing stor	age and u	updated as the load process
0046		7 7 7 8	as work continue	ing stor es. The	age and u sector in	updated as the load process n the command table cannot be
0046		7 7 7	as work continu used di	ing stor es. The rectly a	age and u sector in s the dis	updated as the load process
0046		777777777777777777777777777777777777777	as work continue used die the sec	ing stor es. The rectly a tor numb e sector	age and u sector in s the dis er on the	updated as the load process In the command table cannot be Sk controller requires it to be

Figure 7-5. (Continued)

0131		Sector: ;	DB	Start\$Sector	
0132		Command#Table		02H	;Command Write
0133		Unit: Head:	DB DB	0	;Unit (drive) number = 0 or 1 ;Head number = 0 or 1
0135		Tracks	DB	Start\$Track	;Used as working variable
0136		Sectorsonshe		O	Converted by low-level driver
0137		Byte\$Count:	DW		Sectors#Per#Write
0139		DMA\$Address:	DW	Start\$Image	
013B		Next#Status:	DW	Disk\$Status	Pointer to next status block
					; if commands are chained
013D	4500	Next\$Control	: DW	Disk\$Contro1\$5	
					; if commands are chained
		Main\$Code:			
013F	310001	LXI	SP.Pu	t\$CPM ;Stack	grows down below code
	110301	LXI		non\$Message	Sign on
	0E09	HVI	C,B\$P	RINIS	Print string until \$
0147	CD0500	CALL	BDOS		
014A	213201	LXI	H, Com	mand#Table	Point the disk controller at
	224600	SHLD	-	nd\$Block\$5	: the command block
0140	2270UU	SHLU	comma	104D10CK#3	) the command DIOCK
0150	OE11	MVI	C,Sec	tors#To#Write	;Set sector count
0152	CD7C01	Write\$Loop: CALL	p	PM\$Write	Write data onto diskette
0152		DCR	C Fut ac	CITAMLT CK	;Downdate sector count
	CA0000	JZ	õ		;Warm boot
		~			,
0159	213101	LXI	H, Sec		;Update sector number
0150	3E01	MVI		tors\$Per\$Write	; by adding on number of sectors
015E		ADD	Μ.		; by controller
015F 0160		MOV MVI	M, A		;Save result
0160		CMP	A, Las M	t\$Sector\$On\$Track	+ 1 ;Check if at end of track
	C26F01	JINZ	••	nd\$Track	
0166		MVI			k ;Yes, reset to beginning
	243501	LHLD	Track		;Update track number
016B		INX	H		
0160	223501	SHLD	Track		
		Not\$End\$Tracl			
	2A3901	LHLD	DMA\$A	ddress	;Update DMA address
	110002	LXI		tor\$Size * Sector	s\$Per\$Write
0175		DAD	D		
	223901	SHLD		dress	- the state of the
0179	C35201	JMP	Write	PLOOP	;Write next block
		7 Put\$CPM\$Write		sAt th	is point, the description of the
					ration required is in the variables
				; con	tained in the command table, along
				; wit	h the sector variable.
017C	C5	PUSH	в		;Save sector count in C
			- 	nuting to match A	he disk controller in use
				sating to match th	
017D		MVI	B, 0		Assume head O
017F	3A3101	LDA	Sector	•	Get requested sector
0182	4F	MOV CPI	C, A	Santovineti	;Take a copy of it
	DABCO1	JC	Head\$		1 ;Check if on head 1 ;No
0188		SUI		, Sector\$on\$Head\$0	Bias down for head 1
018A		MOV	C,A		Save copy
018B		INR	B		;Set head 1
		Head\$0:			
018C		MOV	A,B		;Get head
	323401	STA	Head		- <b>D</b> - <b>h</b> - <b>n</b> - <b>h</b> - <b>n</b>
0190	79 323601	MOV STA	A,C	r\$On\$Head	;Get sector

Figure 7-5. (Continued)

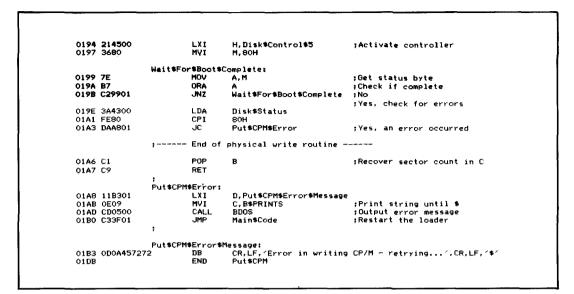


Figure 7-5. (Continued)

# 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:

A>DDT B:XYZ.HEX <er> DDT VERS 2.0</er>	<- 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	file	name	and	type
- <u>R<cr></cr></u>	<-	Read i	n th	ne fil	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>\$5C<cr></cr></u>	<- "S"et location 5C
005C 00 02 <cr></cr>	<- 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)

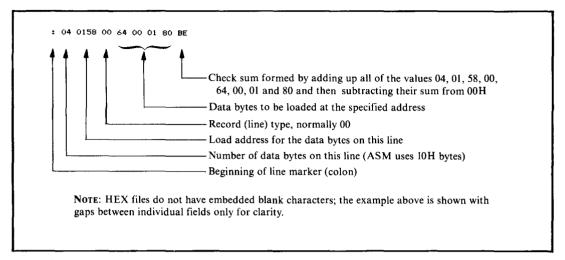


Figure 7-6. Example line from HEX file

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:

-H1F80,F600 <cr></cr>	<- Use the H command
1580,2980	<- Sum, difference

Thus, to read in the BIOS HEX file called FIG6-4.HEX at location 1F80H, you would enter the following commands to DDT:

-IFIG6-4.HEX <cr></cr>	<-	Specify	file	name	and	type	
- <u>R2980<cr></cr></u>	<-	Load at	0F600	H +	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
                 . 2
                          PROM-based bootstrap mechanism that gets control of the
                 2
                          CPU on power up or system reset.
3130 =
                 Version
                                   EQU
                                            1011
                                                     ;Equates used in the sign-on message
3730 =
                 Month
                                   FOU
                                            1071
                                             1241
3432 =
                                   EQU
                 Day
                                            1821
3238 =
                                   EQU
                 Year
0000 =
                 Debug
                                   EQU
                                            0
                                                     ;Set nonzero to debug as normal
                                                     ; transient program
                          The actual layout of the diskette is as follows :
                 ; Track O
                                                   Sector
                            1
                                   2
                                          з
                                                 4
                                                        5
                                                              6
                                                                     7
                                                                            8
                                                                                   9
                 ; Head
                                                   ----
                                                               ____
                    0
                          Boot (<======= CCP =====>)(<====== BDOS =======))</pre>
                 ;
                                   ;
                          | ===== BDOS ====> | <========= BIOS =======> |
                    1
                 ;
                              ---+-
                                   ----+----+----+----+----+----+----
                                                                         -+---+-
                 2
                            10
                                   11
                                        12
                                               13
                                                      14
                                                             15
                                                                   16
                                                                          17
                                                                                 18
                                                   Sector
                          Equates for defining memory size and the base address and
                          length of the system components.
0040 ≃
                 Memory$Size
                                   FOIL
                                            64
                                                     ;Number of Kbytes of RAM
                          The BIOS Length must match that declared in the BIOS.
0900 =
                 BIOS$Length
                                   EQU
                                            0900H
0800 =
                 CCP$Length
                                   EQU
                                            0800H
                                                     ;Constant
                 BDOS$Length
0E00 =
                                   EQU
                                            OEOOH
                                                     ;Constant
                                            ((CCP$Length + BDOS$Length + BIOS$Length) / 1024) + 1 c
CCP$Length + BDOS$Length + BIOS$Length
                                   FOL
0008 =
                 Length$In$K
1F00 =
                 Length$In$Bytes EQU
                 :
                                   NOT Debug
                          IF
E000 =
                 CCP$Entry
                                   EQU
                                            (Memory$Size - Length$In$K) * 1024
                          ENDIF
                          IF
                                   Debug
                 CCP$Entry
                                            3980H
                                                     ;Read into a lower address.
                                   EQU
                                                     This address is chosen to be above
                                                     ; the area into which DDT initially loads
                                                        and the 980H makes the addresses similar
to the SYSGEN values so that the memory
image can be checked with DDT.
                                                     ;
                                                     .
                          ENDIF
                                            CCP$Entry + CCP$Length + 6
CCP$Entry + CCP$Length + BDOS$Length
                                   EQU
E806 =
                 BDOS$Entry
F600 =
                BIOS$Entry
                                   EQU
                 :
                 ;
                          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$Track
                                            EQU
                                                     1
0012 =
                Last$Sector$on$Track
                                            EQU
                                                     18
0009 =
                Last$Sector$on$Head$0
                                            FOU
                                                     9
0200 =
                Sector#Size
                                            FOU
                                                     512
                 .
                 ;
                          Controller characteristics
                 1
                 .
```



```
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
                1
                          sector at a time.
0001 =
                Sectors#Per#Read
                                            EQU
                                                     1
                         Cold boot characteristics
0000 =
                StartsTrack
                                            FOU
                                                     0
                                                              ;Initial values for CP/M image
0002 =
                Start#Sector
                                            EQU
                                                     2
0010 =
                Sectors#To#Read
                                           EQU
                                                     (Length$In$Bytes + Sector$Size - 1) / Sector$Size
                ;
0100
                         ORG
                                   100H
                Cold$Boot$Loader:
0100 034001
                          JIMP
                                   Main$Code
                                                     Enter main code body
                                                     For reasons of clarity, the main
                                                     ; data structures are shown before the
                                                        executable code.
                                                     2
000D =
                CR
                         EQU
                                   ODH
                                                     ;Carriage return
000A =
                LF
                          EQU
                                   OAH
                                                     ;Line feed
                Signon#Message:
                                   CR, LF, 'CP/M Bootstrap Loader'
0103 0D0A43502F
                         DB
                         TE
                                   Debug
                         DB
                                    (Debug) 1
                         ENDIF
011A ODOA
                         DB
                                   CR, LF
011C 5665727369
                          DB
                                   'Version '
0124 3031
                         DW
                                   Version
0126 20
                         DB
0127 3037
                          DW
                                   Month
0129 2F
                         DB
                                   11.
012A 3234
                                   Day
                          DW
012C 2F
                                   11
                          DB
0120 3832
                          DW
                                   Year
012F ODOA00
                          DB
                                   CR, LF, O
                           Disk Control Tables
0045 =
                 Disk$Control$5 EQU
                                            45H
                                                     ;5 1/4" control byte
0046 =
                 Command$Block$5 EQU
                                            46H
                                                     ;Control table pointer
0043 =
                 Disk$Status
                                  EQU
                                            43H
                                                     ;Completion status
                          The command table track and DMA$Address can also be used
                          as working storage and updated as the load process
                          continues. The sector in the command table cannot be used directly as the disk controller requires it to be the sector number on the specified head (1 - -9) rather
                          than the sector number on track. Hence a separate variable
                          must be used.
0132 02
                                   DB
                                            Start#Sector
                 Sector:
0133 01
                 Command$Table:
                                   DB
                                                     ;Command -- read
                                            01H
0134 00
                 Unit:
                                   DB
                                                     ;Unit (drive) number = 0 or 1
                                            0
                                                     Head number = 0 or 1
0135 00
                 Head:
                                   DB
                                            ò
0136 00
                                                              ;Used as working variable
                 Track:
                                   DB
                                            Start$Track
0137 00
                 Sector$on$head: DB
                                                    ;Converted by low-level driver
                                            0
0138 0002
                 Byte$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.
                                            Disk$Control$5 ;Pointer to next control byte
013E 4500
                 Next#Control:
                                   DW
                                                              ; if commands are chained.
                 Main$Code:
0140 310001
                         LXI
                                   SP,Cold$Boot$Loader
                                                              ;Stack grows down below code
```

Figure 7-7. (Continued)

0143 210301 LXI H,Signon\$Message ;Sign on 0146 CDD901 Display#Message CALL 0149 213301 LXI H.Command%Table ;Point the disk controller at 014C 224600 SHLD Command\$Block\$5 ; the command block 014F 0E10 MVI C, Sectors\$To\$Read ;Set sector count Load\$Loop: 0151 CD7B01 CALL Cold\$Boot\$Read ;Read data into memory 0154 OD DCR C. ;Downdate sector count IF NOT Debug 0155 CA00F6 BIOS\$Entry ;Enter BIOS when load done JZ ENDIF IF Debug JZ ۵ ;Warm boot ENDIF 0158 213201 0158 3E01 0150 86 ;Update sector number ; by adding on number of sectors ; by controller LXI H, Sector A, Sectors \$Per \$Read MVT ADD M 015E 77 MOV Save result M.A 015F 3E13 0161 BE MVI A,Last\$Sector\$On\$Track + 1 ;Check if at end of track CMP 0162 C26E01 JNZ Not\$End\$Track 0165 3601 0167 2A3601 016A 23 MVI M,First\$Sector\$On\$Track ;Yes, reset to beginning LHLD Track ;Update track number INX 016B 223601 SHLD Track Not\$End\$Track: 016E 2A3A01 LHLD DMA\$Address Update DMA Address 0171 110002 0174 19 0175 223A01 LXI D,Sector\$Size * Sectors\$Per\$Read DAD D SHED **DMA\$Address** 0178 C35101 JMP 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 ------0170 0600 MVI B, 0 ;Assume head O 017E 3A3201 LDA Sector ;Get requested sector 0181 4F MOV C,A ;Take a copy of it 0182 FEOA CPI Last\$Sector\$on\$Head\$0+1 ;Check if on head 1 0184 DA8801 JC SUİ Head\$0 ;No 0187 D609 0189 4F Last\$Sector\$on\$Head\$0 ;Bias down for head 1 MOV C,A Save copy 018A 04 INR в :Set head 1 Head\$0: 018B 78 MOV A,B ;Get head 018C 323501 018F 79 STA Head MOV A,C ;Get sector 0190 323701 STA Sector\$On\$Head 0193 214500 LXI H,Disk\$Control\$5 Activate controller M,SOH 0196 3680 MUT Wait\$For\$Boot\$Complete: 0198 7E MOV ;Get status byte A.M ORA 0199 B7 ;Check if complete 019A C29801 JNZ Wait\$For\$Boot\$Complete ,No ;Yes, check for errors 019D 3A4300 LDA Disk\$Status 01A0 FE80 CPI 80H Cold\$Boot\$Error 01A2 DAA701 .IC ;Yes, an error occurred ;----- End of physical read routine ------

Figure 7-7. (Continued)

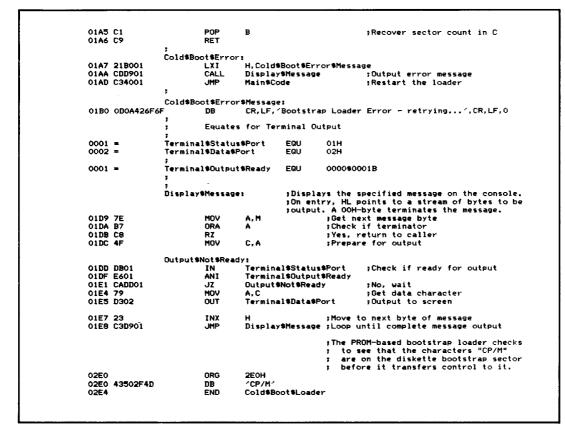


Figure 7-7. (Continued)

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:

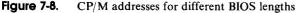
Memory in system - BIOS length from Figure 7-8

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	CCP	BDOS
Length	Base	Offset	'nn'	Base	Base
600	FA00	2580	64	E400	EC00
A00	F600	2980	63	E000	E800
E00	F200	2080	62	DCOO	E400
1200	EEOO	3180	61	D800	E000
1600	EAOO	3580	60	D400	DC00
1A00	E600	3980	59	0000	D800
1E00	£200	3080	58	CCOO	B400
2200	DEOO	4180	57	C800	0000
2600	DAOO	4580	56	C400	CC00
2A00	<b>B60</b> 0	4980	55	C000	C800
2E00	D200	4080	54	BCOO	C400
3200	CEOO	5180	53	B800	C000
3600	CAOO	5580	52	B400	BCOO
3A00	C600	5980	51	B000	B800
3E00	C200	5080	50	ACOO	B400
4200	BEOO	6180	49	A800	B000
4600	BAOO	6580	48	A400	AC00
4A00	<b>B600</b>	6980	47	A000	A800
4E00	B200	6080	46	9000	A400
5200	AEOO	7180	45	9800	A000
5600	AAOO	7580	44	9400	9000
5A00	A600	7980	43	9000	9800
5E00	A200	7080	42	8000	9400
6200	9E00	8180	41	8800	9000
6600	9A00	8580	40	8400	8000
6A00	9600	8980	39	8000	8800
	Apart from the	MOVCPM 'nn' va	lue all other values are	in hexadecimal	



```
Call up MOVCPM requesting a 163K1 system
                    and the image to be left in memory.
A>Movepm 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 UMP (C3H) instructions,
followed by 7FH, 00H, 20H's, and the Digital
Research Copyright notice. These identify 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>d980,9cf(cr)</u>
0980 C3 5C E3 C3 58 E3 7F 00 20 20 20 20 20 20 20 20 .\..X...
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48 COPYRIGH
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>
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.
-d1f80<cr>> 1F80 C3 B3 F6 C3 C3 F6 C3 61 F7 C3 64 F7 C3 6A F7 C3 .....a..d..j..
1FB0 C3 B1 F7 B2 F6 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 6E F8 73 F6 6B .....n.s.k
2030 OD OA OO 31 OO O1 21 9C F6 CD D3 F7 AF 32 04 00 ...1..!.....2..
```

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.
-d780,7af<cr>
0780 C3 40 01 0D 0A 43 50 2F 4D 20 42 6F 6F 74 73 74 .8...CP/M Bootst
0790 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).
-<u>d980,9b1<cr></u>
0980 C3 5C E3 C3 58 E3 7F 00 20 20 20 20 20 20 20 20 .\..X...
0990 20 20 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
 0980 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 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....

1F90 A4 F8 C3 B4 F8 C3 C5 F8 C3 B6 FB C3 0E FB C3 3B ......

1F40 F8 C3 41 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 45 F8 C3 55 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8 C3 56 F8
  IFC0 38 00 00 43 50 2F 4D 20 32 2E 32 2E 30 30 20 30 8..CP/M 2.2.00 0
IFD0 37 2F 31 35 2F 38 32 0D 0A 0A 53 69 6D 70 6C 65 7/15/82...Simple

      IFD0 37 2F 31 35 2F 38 32 0D 0A 0A 33 69 6D 70 6C 65 715782...51mFle

      IFE0 20 42 49 4F 53 0D 0A 0A 44 69 73 6B 20 43 6F 6E BIOS...Disk Con

      IFF0 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 20 5" Floppy...

      2020 20 20 42 3A 20 30 2E 33 35 20 4D 62 79 74

      A: 0.35 Mbyt

      2030 35 22 20 46 6C 6F 70 70 79 0D 0A 20 20 20 5" Floppy...

   -<u>^c</u>
 A>_
```

Figure 7-9. Using DDT to check CP/M images (continued)

# 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.

	Assemble the CP/M Bootstrap Loader,
	with the source code and HEX file
	on drive C:, no listing output.
C> <u>asm_bootf5.ccz<cr></cr></u> CP/M_ASSEMBLER - VER_2.0	
02E4	
004H USE FACTOR	
END OF ASSEMBLY	
1	Assemble the PUTCPMF5 program (that writes CP/M onto the disk), with
	the source code and HEX file on
	drive C:, no listing output.
C>asm putcpmf5.ccz <cr></cr>	
CP/M ASSEMBLER - VER 2.0	
01DB 003H USE FACTOR	
END OF ASSEMBLY	
	Assemble the BIOS with the source
	code and HEX file on drive C:, no
C>asm bios1.ccz <cr></cr>	listing output.
CP/M ASSEMBLER - VER 2.0	
FE6C	
011H USE FACTOR	
END OF ASSEMBLY	
1	Start piecing the CP/M image together. Load DDT and ask it to
	together. Load DD1 and ask it to read in the file previously SAVEd
	after a MOVCPM 63 *.
C>ddt cpm63.com <cr></cr>	
DDT VERS 2.2	
NEXT PC 2300 0100	
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 ORG 100H it
	100H because of the DRG 100H it
-r <cr></cr>	contains) <u>iputcpmf5.hex<cr></cr></u>
NEXT PC	
2300 0100	
4	
	Indicate the file name of
1	BOOTF5.HEX and read in with an offset of 680H to make it load at
1	780H on up (it contains QRG 100H
	too).
-ibootf5.hex <cr></cr>	
- <u>r680<cr></cr></u> NEXT PC	
NEXT PC 2300 0100	
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
-ibiost bay(ar)	0F600H).
- <u>ibios1.hex<cr></cr></u> - <u>r2980<cr></cr></u>	
NEXT PC	
27EC 0000	
J	Exit from DDT by going to location
-g0(cr)	0000H and executing a warm boot.
Acreix	
	Save the complete CP/M image on
l	disk. Saving 40 256-byte pages from
	location 100H to 2900H.
C>save 40 putcpmf5.com <cr></cr>	
}	

Figure 7-10. Console dialog for system build

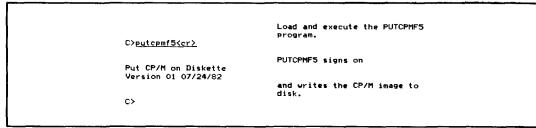
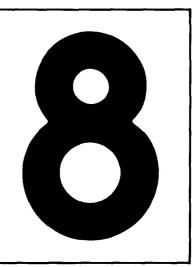


Figure 7-10. Console dialog for system build (continued)

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. 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

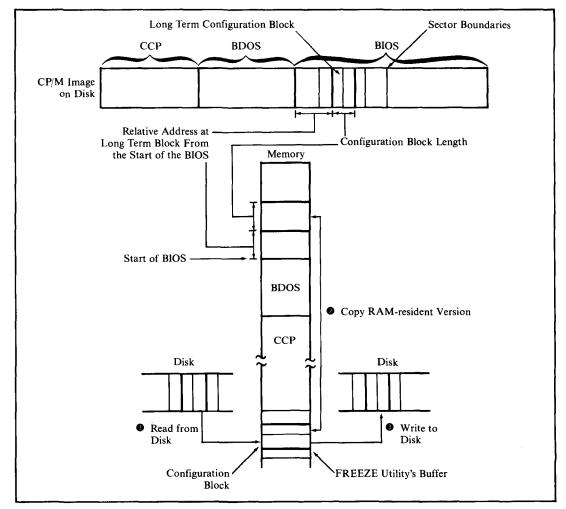


Figure 8-1. Saving the long term configuration block

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

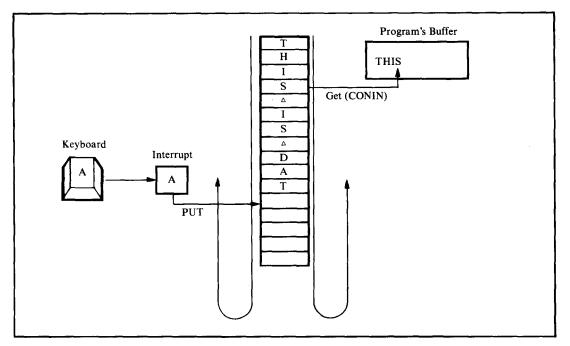


Figure 8-2. Circular buffer type-ahead

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 typeahead.

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.

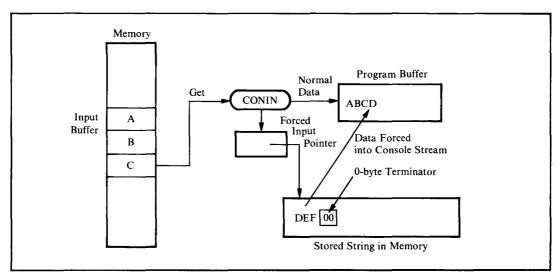


Figure 8-3. CONIN uses forced input data if pointer points to nonzero byte

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

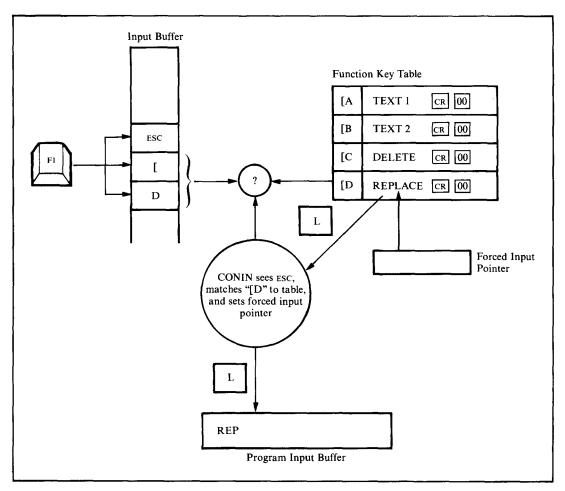


Figure 8-4. CONIN decodes terminal function keys

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.

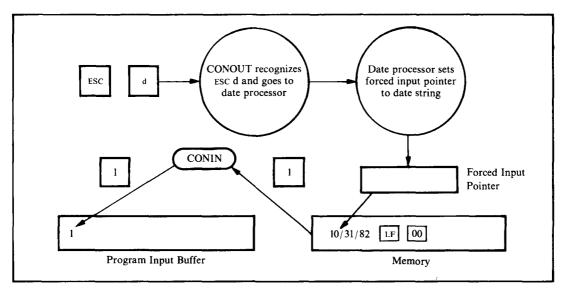


Figure 8-5. Escape sequences sent to CONOUT allow the date to be read by CONIN

## "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

	<ul> <li>physical device</li> <li>are used to acc</li> </ul>	e they sa	ce table for each ervice. The equat various fields w	tes that follow
	; device table.			
	;	Port n	umbers and status	s bits
0000 = 0001 =	DT\$Status\$Port DT\$Data\$Port	EQU EQU	0 ;Device DT\$Status\$Port+	e status port number ⊦1
0002 =	DT\$Output\$Ready	EQU	;Device DT\$DataPort+1	e data port number
0003 =	DT\$Input\$Ready	EQU	DT\$Output\$Ready	
0004 =	DT\$DTR\$Ready	EQU	DT\$Input\$Ready+	
0005 =	DT\$Reset\$Int\$Port	EQU	DT\$DTR\$Ready+1 ;Port n	eady to send mask number used to reset an
0006 =	DT\$Reset\$Int\$Value	EQU	; inte DT\$Reset\$Int\$Po	ort+1
0007 =	DT\$Detect\$Error\$Port	EQU	DT\$Reset\$Int\$Va	output to reset interrupt alue+1 number for error detect
0008 =	DT\$Detect\$Error\$Value	EQU	DT\$Detect\$Error	*Fort+1
0009 =	DT\$Reset\$Error\$Port	EQU	DT\$Detect\$Error	for detecting error (parity etc.) *\$Value+1 t to port to reset error
= A000	DT\$Reset\$Error\$Value	EQU	DT\$Reset\$Error\$	Port+1 to output to reset error
000B =	DT\$RTS\$Contro1\$Port	EQU	DT\$Reset\$Error\$	
	DT\$Drop\$RTS\$Value	EQU	DT\$RTS\$Control\$	
000B =	DT\$Raise\$RTS\$Value	EQU	DT\$Drop\$RTS\$Val	
	; Bayian	logical	status (incl. pr	
000E =	, DEVICE DT\$Status	EQU	DT\$Raise\$RTS\$Va Status	alue+1
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\$Qutput\$Etx	EQU	0001\$0000B	;Output uses Etx/Ack
0020 =	DT\$Output\$Timeout	EQU	0010\$0000B	;Output uses Timeout
0040 =	DT\$Input\$RTS	EQU	0100\$000B	;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	f chars.sent in Etx protocol
0012 =	DT\$Etx\$Message <b>\$Length</b>	EQU	DT\$Etx\$Count+2	fied message length
	;	Inout 4	buffer values	
0014 =	; DT\$Buffer\$Base	EQU	DT\$Etx\$Message\$	\$Length+2 ss of input buffer
0016 =	DT\$Put\$Offset	EQU	DT\$Buffer\$Base+	
0017 =	DT\$Get\$Offset	EQU	DT\$Put\$Offset+1	
0018 =	DT\$Buffer\$Length\$Mask	EQU	DT\$Get\$Offset+1 ;Length ;Note: ; a bi ;This m ; 32 - ; 64 -	

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
0019 =	DT\$Character\$Count	EQU	; the buffer has been reached. DT\$Buffer\$Length\$Mask+1
			;Count of the number of characters
			; currently in the buffer
001A =	DT\$Stop\$Input\$Count	EQU	DT\$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
001C =	DT\$Control\$Count	EQU	; this value
0010 -	DISCONTROISCOUNT	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
		240	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\$Recidy** 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\$Reacty** 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\$RT\$\$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.
- **DI\$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 input.

- **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\$Inificilize\$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.

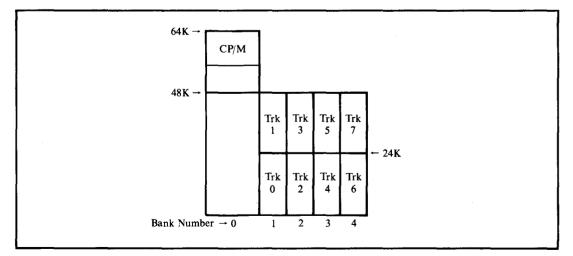


Figure 8-7. Memory organization for M-disk

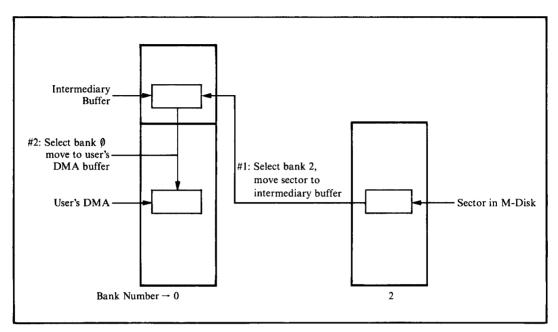


Figure 8-8. Reading a sector from the M-disk image

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.

Sta	rt Line	Functional Component or Routine
0	0001	Introductory Comments and Equates
0	0200	BIOS Jump Table with Additional Private Entries
0	0400	Long Term Configuration Block
0	0800	Interrupt Vector
0	0900	Device Port Numbers and Other Equates
0	1100	Display\$Message Subroutine
0	1200	Enter\$CPM Setup
0	1300	Device Table Equates
0	1500	Device Table Declarations
0.	1700	General Device Initialization
0	1800	Specific Device Initialization
0	2000	Output Byte Stream
02	2100	CONST Routine
02	2200	CONIN Routine with Function Key Processing
0	2500	Console Output
02	2700	CONOUT Routine with Escape Sequence Processing
0.	2900	AUXIST—Auxiliary Input Status Routine
0.	3000	AUXOST—Auxiliary Output Status Routine
0.	3100	AUXIN—Auxiliary Input Routine
0.	3200	AUXOUT—Auxiliary Output Routine
0.	3300	LISTST-List Status Routine
0.	3400	LIST—List Output Routine
0.	3500	Request User Choice—Request Action After Error
0.	3600	Output Error Message
0.	3656	Get Composite Status from Selected Output Devices
0.	3800	Multiple Output of Byte to All Output Devices
04	4000	Check Output Device Logically (Protocol) Ready
04	4200	Process ETX/ACK Protocol
04	4400	Select Device Table from I/O Redirection Bit Map
04	4600	Get Input Character from Input Buffer
04	4800	Introductory Comments for Interrupt-Driven Drivers
04	4900	Character Interrupt Service Routine
0:	5000	Service Device—Puts Character into Input Buffer
0:	5300	Get Address of Character in Input Buffer
0:	5400	Check if Control Character (not CR, LF, TAB)
0:	5500	Output Data Byte
0:	5700	Input Status Routine
0:	5900	Set Watchdog Timer Routine
00	6000	Real Time Clock Interrupt Service Routine
06	6200	Shift HL Right One Bit Routine
06	6300	Introductory Comments for High-Level Disk Drivers
00	6400	Disk Parameter Headers
00	6600	Disk Parameter Blocks
00	6800	SELDSK—Select Disk Routine
01	7000	SETTRK—Set Track Routine
01	7100	SETSEC—Set Sector Routine

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	WBOOT—Warm 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 is	a skele	etal exa	mple of an emhanced BIOS.				
		00010	;	It includes fragments of the standard BIOS							
		00011	;	shown as Figure 6-4 in outline, so as to							
		00012	;	avoid c	avoid cluttering up the enhancements with the						
		00013	;	support	e. Many of the original						
		00014	;	comment	blocks	have be	en abbreviated or deleted				
		00015	,	entirel	¥.						
		00016	7								
		00017	·;< 1	NOTE:	The lit	ne numbe	rs at the left are included				
		00018	;	to allow reference to the code from the text. There are deliberate discontinuities in the							
		00019	;								
		00020	;		numbers	s to all	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	<b>~8</b> 3~					
		00026	;								
		00027	027 ; ***********************************								
		00028	;*					×			
		00029	;*	This BI	DS is fo	or a com	puter system with the following	×			
		00030	;×	hardward	e config	guration	:	×			
		00031	; *					×			
		00032	;*		8080	O CPU		×			
		00033	;×		64K	bytes o	f RAM	×			
		00034	;*		3 50	erial I/	O ports (using signetics 2651) for:	*			
		00035	; *		c0	nsole, c	ommunications and list	×			
		00036	;*		Two	5 1/4"	mini floppy, double-sided, double-	×			
		00037	; *		dei	nsity dr	ives. These drives use 512-byte sectors	*			
		00038	;*		Th	ese are	used as logical disks A: and B:,	×			
		00039	3 *		Fu	ll track	buffering is supported.	×			

Figure 8-10. Enhanced BIOS listing

1		00040	;*	Two	8" standard di	skette drives (128-byte sectors)	
		00041	1 m		se ave used as	logical disks C: and B:.	×
		00042	3 *			<pre>&lt; (M-disk) is supported.</pre>	×
		00043	; <del>*</del>		mony based dis	( (I DISK) IS SUPPORTED.	*
		00044	*	Two	intelligent di	sk controllers are used, one for	×
		00045	;*	ea	h diskette type	. These controllers access memory	×
		00046	;*	di	ectly, both to	read the details of the	×
		00047	;*	OP	rations they an	e to perform and also to read	×
		00048	;*	an	write data fro	om and to the diskettes.	*
		00049	;*				×
		00050	;*				×
		00051	************	********	水水水水水水水水水水水水	***************************************	e¥
		00052					
		00053	<b>—</b> .				
1		00054	; Equate	is for ch	racters in the	ASCII character set	
0011	_	00055	;				
0013		00056 00057	XON EQU	11H		insmission of data	
0003		00058	XOFF EQU ETX EQU	13H 03H		ismission of data	
0006		00059	ACK EQU	06H	;End of transm	115510h	
0000		00060	CR 'EQU	ODH	;Acknowledge		
0000		00061	LF EQU	OAH	;Carriage retu ;Line feed	irn	
0009		00062	TAB EQU	09H	;Horizontal ta	ъ	
0007		00063	BELL EQU	07H	Sound termina		
		00064	;	w/11	, sound termine		
1		00065	;				
l		00066		s for de	ining memory si	ze and the base address and	
		00067	; length	of the	ystem component	5	
		00068	;				
0040	=	00069	Memory\$Size	EQU	64 ;Numbe	er of Kbytes of RAM	
		00070	;				
1		00071	; The BI	OS lengt	must be determ	nined by inspection.	
		00072	; Commen	t out th	ORG BIOS\$Entry	line below by changing the first	
		00073 00074	; charac	ter to a	semicolon (this	will make the assembler start	
		00075	; the Bi	US at 10	ation (). Then	assemble the BIOS and round up to	
		00076	; the ne ; of the	arest 10	H the address d	lisplayed on the console at the end	1
		00077	, or the	assembl	•		
2500	-	00078	, BIOS\$Length	EQU	2500H :< R	evised to an approximate value	
		00079	Dissiendin	Lao		o reflect enhancements	
		00080	:		, ,	o reflect enhancements	
0800	=	00081	CCP\$Length	EQU	0800H ;Const	ant	
0E00	=	00082	BDOS\$Length	EQU	OEOOH ;Const		
		00083	;				
000F	=	00084	Overall\$Length	EQU	(CCP\$Length +	BDOS\$Length + BIOS\$Length + 1023)	/ 1024
		00085	,				
C400		00086	CCP\$Entry	EQU	(Memory\$Size -	Overall\$Length) * 1024	
CC06		00087	BDOS\$Entry	EQU	CCP\$Entry + CC		
DAOO	-	00088	BIOS\$Entry	EQU	CCP\$Entry + CC	P\$Length + BDOS\$Length	
		00089	;				
0005	=	00090	BDOS	EQU	0005H ;BDOS	entry point (used for making	
		00091			; 575	tem reset requests)	
1		00092	;				
		00200 00201	;# : ORG	<b>D</b> 10045			
		00202	; ORG	BIOS\$Er	ry ;Assem	ble code at BIOS address	
		00202	•	ump vecto			
		00203	; B105 J	amp vecto			
0000	C31311	00205	, JMP	BOOT	:Cold boot	entered from CP/M bootstrap loader	
		00206	Warm\$Boot\$Entr			that the initialization code can	
1		00207		• •		m boot entry address in location	
l		00208			; 0001H and 0	002H of the base page	
0003	C3750E	00209	JMP	WBOOT	;Warm boot	entered by jumping to location 000	он
1		00210			; Reloads the	CCP, which could have been	
		00211			; overwritten	by previous program in transient	
0000	0000000	00212			; program are		
0006	C32D03	00213	JMP	CONST		s returns A = OFFH if there is	a
0000	C22402	00214			; console key	board character waiting	
0009	C33A03	00215	JMP	CONIN	;Console input	returns the next console keybo	ard
0000	C3D703	00216 00217	JMP	CONOUT	; character in		
0000	030/03	00218	CAURA CAURA	000001	; the console	t outputs the character in C to device	
000F	C3F504	00219	JMP	LIST		- outputs the character in C to th	-
		00220	On	2101	; list device	outputs the character in 5 to th	e
0012	C3CE04	00221	JMP	AUXOUT		out outputs the character in C	to the
		00222				iliary device	
L							

Figure 8-10. (Continued)

3A104				
30104	00223	JMP	AUXIN	;Auxiliary input returns the next input character from
	00224			; the logical auxiliary device in A
3160A	00225	JMP	HOME	Homes the currently selected disk to track O
36309	00226	JMP	SELDSK	;Selects the disk drive specified in register ${\mathbb C}$ and
	00227			; returns the address of the disk parameter header
39B09		JMP	SETTRK	;Sets the track for the next read or write operation
				; from the BC register pair
3A109		JMP	SETSEC	;Sets the sector for the next read or write operation
				; from the A register
3A809		JMP	SETDMA	;Sets the direct memory address (disk read/write)
				; address for the next read or write operation
				; from the DE register pair
3370A		JMP	READ	Reads the previously specified track and sector from
				; the selected disk into the DMA address
34B0A		JMP	WRITE	;Writes the previously specified track and sector onto
				; the selected disk from the DMA address
30704		JMP	LISTST	Returns A = OFFH if the list device(s) are
			OF OT O AN	; logically ready to accept another output byte ; Translates a logical sector into a physical one
3100A		JMP	SECTRAN	; Franslates a logical sector into a physical one
		7		
			tional "pri	vate Blos entry points
			AUVICT	Returns A = OFFH if there is input data for
-30F 04		Unit	HOVIOL	; the logical auxiliary device
39804		IMD		; Returns A = OFFH if the auxiliary device(s) are
.37504		UFIF	HOXODI	; logically ready to accept another output byte
35403		IMD.	Spacifi	s logically ready to accept another output byte
OF NUZ		OHE	OPECITI	;Initializes character device whose device
				; number is in register A on entry
34009		IMD	Setelist	
		one	og tømelt	Sets up watchdog timer to CALL address specified
				; in HL, after BC clock ticks have elapsed
20005		.IMP		
.3300P		on	0040614	Configuration block get address
				; Returns address in HL of data element whose
				; code number is specified in C
			term confi	guration block
	00402			
	00403	Long\$Term\$CB	:	
	00404	;		
	00405	;		
	00406			iles in user O accessible from all
	00407	; othe	r user numb	ers) enabled when this flag is set
		; nonz	ero.	
		;		
00	00410	CB\$Public\$Fi	les:	DB 0 ;Default is OFF
		;		
		;		
	00413	; The	forced inpu	t pointer is initialized to point to the
		; foll	owing strin	ng of characters. These are injected into
		; the	console inp	out stream on system start-up.
	00416	<u>.</u>		
		CB\$Startup:		DBSUBMIT_STARTUP_,LF,0,0,0,0,0,0
5355424D				
5355424D4	00418	,		s s suite and the state
5355424D4	00418 00419	; ; Logi	cal to phys	ical device redirection
5355424D4	00418 00419 00420	; ; Logi ;		
5355424D4	00418 00419 00420 00421	; ; Logi ;	Each lo	gical device has a 16-bit word associated
5355424D4	00418 00419 00420 00421 00422	; Logi ; ;	Each lo with it	ngical device has a 16-bit word associated . Each bit in the word is assigned to a
5355424D4	00418 00419 00420 00421 00422 00423	; Logi ; ; ;	Each lo with it specifi	pgical device has a 16-bit word associated Each bit in the word is assigned to a .c physical device. For input, only one bit
53 <b>55</b> 424D4	00418 00419 00420 00421 00422 00423 00423	; Logi ; ; ; ;	Each lo with it specifi can be	pgical device has a ló-bit word associated . Each bit in the word is assigned to a .c physical device. For input, only one bit set input will be read from the
53 <b>55</b> 424D4	00418 00419 00420 00421 00422 00423 00424 00425	; Logi ; ; ; ; ;	Each lo with it specifi can be corresp	ogical device has a 16-bit word associated . Each bit in the word is assigned to a .c physical device. For input, only one bit set input will be read from the oonding physical device. Output can be
535542404	00418 00419 00420 00421 00422 00423 00424 00425 00426	; Logi ; ; ; ;	Each lo with it specifi can be corresp directe	pgical device has a 16-bit word associated Each bit in the word is assigned to a physical device. For input, only one bit set input will be read from the bonding physical device. Output can be ed to several devices, so more than one
53 <b>55</b> 424D4	00418 00419 00420 00421 00422 00423 00424 00425 00425 00426 00427	; Logi ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe	ogical device has a 16-bit word associated . Each bit in the word is assigned to a .c physical device. For input, only one bit set input will be read from the oonding physical device. Output can be
53 <b>55424</b> D4	00418 00419 00420 00421 00422 00423 00423 00424 00425 00426 00427 00428	; Logi ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can	pgical device has a 16-bit word associated . Each bit in the word is assigned to a .c physical device. For input, only one bit set input will be read from the sonding physical device. Output can be ed to several devices, so more than one h be set.
53 <b>55</b> 424D4	00418 00419 00420 00421 00422 00423 00424 00425 00425 00425 00426 00425 00426	; Logi ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol	pgical device has a 16-bit word associated Each bit in the word is assigned to a physical device. For input, only one bit set input will be read from the bonding physical device. Output can be ed to several devices, so more than one h be set. Nowing equates are used to indicate
53 <b>55</b> 424D4	00418 00419 00420 00421 00422 00423 00424 00425 00426 00427 00428 00427 00428 00429 00430	; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol	egical device has a 16-bit word associated . Each bit in the word is assigned to a .c physical device. For input, only one bit set input will be read from the sonding physical device. Output can be ed to several devices, so more than one h be set.
53 <b>55</b> 424D4	00418 00419 00420 00422 00423 00423 00424 00425 00425 00425 00425 00426 00427 00428 00429 00430	; Logi ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol	sgical device has a 16-bit word associated . Each bit in the word is assigned to a . Each pit in the word is assigned to a . physical device. For input, only one bit set input will be read from the sonding physical device. Output can be ed to several devices, or more than one h be set. llowing equates are used to indicate to physical devices.
53 <b>55</b> 424D4	00418 00419 00420 00421 00422 00423 00425 00425 00425 00425 00426 00427 00428 00429 00430 00431	; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol	bgical device has a 16-bit word associated . Each bit in the word is assigned to a . c physical device. For input, only one bit set input will be read from the bonding physical device. Output can be d to several devices, so more than one h be set. llowing equates are used to indicate (physical devices. 1111 11 )
	00418 00419 00420 00421 00422 00423 00424 00425 00425 00426 00427 00428 00429 00430 00431 00432 00433	; Logi ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lc with it specifi can be corresp directe bit can The fol specifi	pgical device has a 16-bit word associated . Each bit in the word is assigned to a ic physical device. For input, only one bit set input will be read from the bonding physical device. Output can be ed to several devices, owner than one to be set. Howing equates are used to indicate ic physical devices. 1111 1
-	00418 00419 00420 00421 00422 00423 00423 00423 00424 00425 00426 00427 00428 00429 00430 00431 00432 00433 00434	; Logi ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol specifi EQU	bgical device has a 16-bit word associated . Each bit in the word is assigned to a . physical device. For input, only one bit set input will be read from the bonding physical device. Output can be ed to several devices, so more than one i be set. 10wing equates are used to indicate to physical devices. 1111 11 ) 5432 1098 7654 3210 <- Device number 00005000050000180
-	00418 00419 00420 00421 00422 00423 00424 00425 00425 00425 00425 00426 00427 00432 00433 00433 00433	; Logi ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol specifi EQU EQU	bgical device has a 16-bit word associated Each bit in the word is assigned to a ic physical device. For input, only one bit set input will be read from the bonding physical device. Output can be id to several devices, so more than one in be set. Howing equates are used to indicate ic physical devices. 1111 11 ) 5432 1098 7654 3210 \<- Device number 0000\$0000\$0000\$0001B 0000\$0000\$0000\$00010B
-	00418 00419 00420 00421 00423 00423 00424 00425 00425 00425 00426 00427 00428 00429 00431 00431 00433 00433 00434	; Logi ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol specifi EQU	bgical device has a 16-bit word associated . Each bit in the word is assigned to a . physical device. For input, only one bit set input will be read from the bonding physical device. Output can be ed to several devices, so more than one i be set. 10wing equates are used to indicate to physical devices. 1111 11 ) 5432 1098 7654 3210 <- Device number 00005000050000180
-	00418 00419 00420 00421 00422 00423 00424 00425 00425 00425 00425 00426 00427 00432 00433 00433 00433	; Logi ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Each lo with it specifi can be corresp directe bit can The fol specifi EQU EQU EQU	bgical device has a 16-bit word associated Each bit in the word is assigned to a ic physical device. For input, only one bit set input will be read from the bonding physical device. Output can be id to several devices, so more than one in be set. Howing equates are used to indicate ic physical devices. 1111 11 ) 5432 1098 7654 3210 \<- Device number 0000\$0000\$0000\$0001B 0000\$0000\$0000\$00010B
	39809 3A109 3A809 3370A 3480A 3100A 39804 39804 35A02 36D08 33C0F	39809         00228 00299           3A109         00230           00231         00233           00234         00235           00235         00234           3370A         00235           00236         00234           3370A         00233           00240         00236           34B0A         00237           00240         00240           3100A         00241           00244         00244           39B04         00245           37804         00246           39B04         00245           00250         00251           36D08         00255           00255         00255           00256         00255           00257         00256           00258         00257           00400         00401           00400         00401           00400         00402           00400         00403           00400         00406           00400         00407           00400         00411           00411         00411           00411         00411           00	39809         00228         JMP           0029         JMP           00230         JMP           00231         JMP           00232         JMP           00233         JMP           00234         JMP           00235         JMP           00236         JMP           00237         JMP           00238         JMP           00240         JMP           00241         JMP           00242         J           00243         T           00244         JMP           00245         JMP           00246         JMP           00247         JMP           00248         JMP           00249         JMP           00246         JMP           00250         00251           36D08         00252         JMP           00254         JMP           00255         JMP           00256         00257           00257         00258           00259         7           00400         1           00401         1           00402         00402	39809         00228         JMP         SETTRK           00229         JMP         SETTRK           3A109         00230         JMP         SETSEC           00231         JMP         SETDMA         00233           00234         JMP         SETDMA           00235         JMP         READ           00236         JMP         WRITE           00237         JMP         WRITE           00238         JMP         LISTST           00240         JMP         SECTRAN           00242         ;         Additional "pri           00243         ;         Additional "pri           00244         ;         JMP         AUXIST           00245         JMP         AUXIST         00246           38F04         00247         JMP         AUXIST           00246         JMP         Secifi         00250           00250         00251         JMP         SetsWat           00254         JMP         SetsWat         00255           00255         JMP         CB\$Get\$         00256           00255         JMP         CB\$Get\$         00400           00400

Figure 8-10. (Continued)

		00440 00441	?	the a	ppropria	te physica	l device drivers
0058	0100	00442	, CB\$Console\$Ing		DW	Davidana	
	0100	00443	CB\$Console\$Out			Device\$	
	0100	00444	CD#CONSOle#001	(put)	DW	Device\$	)
0050			,				
	0200	00445	CB\$Auxiliary\$]		DW	Device\$	
005E	0200	00446	CB\$Auxiliary\$(	Output:	DW	Device\$	l
		00447	;				
0060	0400	00448	CB\$List\$Input:		DW	Device\$2	2
0062	0400	00449	CB\$List\$Output	:	DW	Device\$2	2
		00450	;				
		00451	;	The t	able belo	ow relates	specific bits in the
		00452	;	redir	ection w	ords above	to specific device
		00453	7	table	s used b	v the physi	cal drivers
		00454					
		00455	CB\$Device\$Tabl	esAddre	5585		
0064	8E02	00456	DW	DT\$O			
	AE02	00457	DW	DT\$1			
	CE02	00458	DW	DT\$2			
	000000000		- DW				• · · ·
0006	000000000	00460		0,0,0	, 0, 0, 0, 0, 0,	,0,0,0,0,0,0,	0 ;Unassigned
			;				
		00461					
		00462		initia	lization	byte strea	ms
		00463	;				
		00464	; These	initial	ization s	streams are	output during the device
		00465	; initia	lizatio	n phase,	or on requ	est whenever the baud rate fined in the long term
		00466	; needs	to be cl	hanged. 1	They are de	fined in the long term
		00467	; config	uration	block so	o as to "fr	eeze" their contents from one
		00468		ı startu	p until t	the next.	
		00469	;				
		00470	f The ad	dress o	f each st	tream is co	ntained in each device table.
		00471	;				
		00472	; The st	ream for	rmat is:		
		00473	;				
		00474	;	DB	××		;Port number (OOH terminates)
		00475	;	DB	nn		Number of bytes to output to port
		00476	;	DB	vv, vv,	vv	;Values to be output
		00477					,
		00478	DO\$Initialize\$	Stream:		:Example	data for an 8251A chip
0084	ED	00479	DB	OEDH			Port number for 8251A
0085		00480	DB	6			;Number of bytes
	000000	00481	DB	0.0.0			;Dummy bytes to get chip ready
0089		00482	DB	0100\$0	0010B		Reset and raise DTR
008A		00483	DB		\$11\$10B		;1 stop, no parity, 8 bits/char,
00011	01	00484					; divide down of 16
0088	25	00485	DB	0010\$0	21018		RTS high, enable Tx/Rx
0000	20	00486	55	00100	01010		data for an 8253 chip
0080	DE	00487	DB	ODEH			Port number for 8253 mode
0080		00488	DB	1			Number of bytes to output
008E		00489	DB	-	6011\$0B		;Select:
OUGE	/0	00490	66	01#114	011000		
		00490					; Counter 1 ; Load LS byte first
		00492					
							. Mada 2 history anyot
0000			DB	ODEN			; Mode 3, binary count
008F		00493	DB	ODEH			Port number for counter
008F 0090		00493 00494	DB	2			;Port number for counter ;Number of bytes to output
0090	02	00493 00494 00495	DB DO\$Baud\$Rate\$C	2 onstant:			;Port number for counter ;Number of bytes to output ;Label used by utilities
0090 0091	02 0700	00493 00494 00495 00495	DB DO\$Baud\$Rate\$C DW	2 onstant: 0007H			;Port number for counter ;Number of bytes to output ;Label used by utilities ;%600 Baud (based on 16x divider)
0090	02 0700	00493 00494 00495 00496 00497	DB DO\$Baud\$Rate\$C	2 onstant:			;Port number for counter ;Number of bytes to output ;Label used by utilities
0090 0091	02 0700	00493 00494 00495 00496 00497 00498	DB DO\$Baud\$Rate\$C DW DB	2 onstant: 0007H 0			;Port number for counter ;Number of bytes to output ;Label used by utilities ;9600 Baud (based on 16x divider) ;Port number of 00 terminates stream
0090 0091 0093	02 0700 00	00493 00494 00495 00496 00497 00498 00499	DB DO\$Baud\$Rate\$C DW DB DI\$Initialize\$	2 onstant: 0007H 0 Stream:		;Example	Port number for counter Number of bytes to output Label used by utilities 9600 Baud (based on 16x divider) Port number of 00 terminates stream data for an 8251A chip
0090 0091 0093 0094	02 0700 00 DD	00493 00494 00495 00496 00497 00498 00499 00500	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB	2 onstant: 0007H 0 Stream: 0DDH		;Example	;Port number for counter ;Number of bytes to output ;Label used by utilities ;9600 Baud (based on 16x divider) ;Port number of 00 terminates stream data for an 8251A chip ;Port number for 8251A
0090 0091 0093 0094 0095	02 0700 00 DD 06	00493 00494 00495 00496 00497 00498 00499 00500 00501	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB	2 onstant: 0007H 0 Stream: 0DDH 6		;Example	Port number for counter Number of bytes to output Label used by utilities 9600 Baud (based on 16x divider) Port number of 00 terminates stream data for an 8251A chip Port number for 8251A Number of bytes
0090 0091 0093 0094 0095 0096	02 0700 00 DD 06 000000	00493 00494 00495 00496 00497 00498 00499 00500 00500 00501 00502	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0		;Example	Port number for counter Number of bytes to output Slabel used by utilities 9600 Baud (based on 16x divider) Port number of 00 terminates stream data for an 8251A chip Port number for 8251A Number of bytes Number of bytes
0090 0091 0093 0094 0095 0096 0099	02 0700 00 DB 06 000000 42	00493 00494 00495 00496 00497 00498 00499 00500 00501 00502 00503	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB DB DB	2 0007H 0 Stream: 0 0 0 0,0,0 0100\$0	0010B	;Example	;Port number for counter ;Number of bytes to output ;Label used by utilities ;9600 Baud (based on 16x divider) ;Port number of 00 terminates stream data for an 8251A chip ;Port number for 8251A ;Number of bytes ;Dummy bytes to get chip ready ;Reset and raise DTR
0090 0091 0093 0094 0095 0096	02 0700 00 DB 06 000000 42	00493 00494 00495 00495 00496 00497 00498 00499 00500 00501 00502 00503 00504	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB	2 0007H 0 Stream: 0 0 0 0,0,0 0100\$0		;Example	Port number for counter Number of bytes to output Label used by utilities 9600 Baud (based on 16x divider) Port number of 00 terminates stream data for an 8251A chip Port number for 8251A Number of bytes 9Dummy bytes to get chip ready Reset and raise DTR 1 stop, no parity, 8 bits/char,
0090 0091 0093 0095 0095 0096 0099 009A	02 0700 00 DB 06 000000 42 6E	00493 00494 00495 00496 00497 00498 00499 00500 00501 00502 00503 00503 00504 00505	DB DO\$Baud\$Rate\$C DW DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$10\$	0010B \$11\$10B	;Example	Port number for counter Number of bytes to output Slabel used by utilities Sooo Baud (based on 16x divider) Port number of 00 terminates stream data for an 3251A chip Fort number for 8251A Number of bytes Dummy bytes to get chip ready Reset and raise DTR Stop, no parity, 8 bits/char, i divide down of 16
0090 0091 0093 0094 0095 0096 0099	02 0700 00 DB 06 000000 42 6E	00493 00494 00495 00495 00496 00497 00498 00499 00500 00501 00502 00503 00504	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB DB DB	2 0007H 0 Stream: 0 0 0 0,0,0 0100\$0	0010B \$11\$10B	;Example	Port number for counter Number of bytes to output Label used by utilities 9600 Baud (based on 16x divider) Port number of 00 terminates stream data for an 8251A chip Port number for 8251A Number of bytes 9Dummy bytes to get chip ready Reset and raise DTR 1 stop, no parity, 8 bits/char,
0090 0091 0093 0095 0095 0096 0099 009A	02 0700 00 DB 06 000000 42 6E	00493 00494 00495 00496 00497 00498 00499 00500 00501 00502 00503 00503 00504 00505 00506 00507	DB DO\$Baud\$Rate\$C DW DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$10\$	0010B \$11\$10B	;Example	Port number for counter Number of bytes to output Slabel used by utilities Sooo Baud (based on 16x divider) Port number of 00 terminates stream data for an 3251A chip Fort number for 8251A Number of bytes Dummy bytes to get chip ready Reset and raise DTR Stop, no parity, 8 bits/char, i divide down of 16
0090 0091 0093 0095 0095 0096 0099 009A	02 0700 00 DB 06 000000 42 6E	00493 00494 00495 00495 00497 00498 00499 00500 00501 00502 00503 00504 00505 00506	DB DO\$Baud\$Rate\$C DW DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$10\$	0010B \$11\$10B	;Example	Port number for counter Number of bytes to output Slabel used by utilities 9600 Baud (based on 16x divider) Port number of 00 terminates stream data for an 8251A chip Port number for 8251A Number of bytes Dummy bytes to get chip ready PReset and raise DTR 1 stop, no parity, 8 bits/char, ; divide down of 16 PRTS high, enable Tx/Rx
0090 0091 0093 0095 0095 0096 0099 009A	02 0700 00 06 000000 42 6E 25	00493 00494 00495 00496 00497 00498 00499 00500 00501 00502 00503 00503 00504 00505 00506 00507	DB DO\$Baud\$Rate\$C DW DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$10\$	0010B \$11\$10B	;Example ;Example	Port number for counter Number of bytes to output Slabel used by utilities Sooo Baud (based on 16x divider) Port number of 00 terminates stream data for an 3251A chip Fort number for 8251A Number of bytes Dummy bytes to get chip ready Reset and raise DTR Stop, no parity, 8 bits/char, i divide down of 16
0090 0091 0093 0094 0095 0096 0099 009A 009B	02 0700 00 06 000000 42 6E 25 DF	00493 00494 00495 00495 00496 00497 00498 00499 00500 00500 00500 00503 00504 00505 00506 00507 00508 00509	DB D0\$Baud\$Rate\$C DW DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$10\$ 0010\$0	0010B \$11\$10B	;Example ;Example	<pre>;Port number for counter ;Number of bytes to output ;Label used by utilities ;9600 Baud (based on 16x divider) ;Port number of 00 terminates stream data for an 8251A chip ;Port number for 8251A ;Number of bytes ;Dummy bytes to get chip ready ;Reset and raise DTR ;1 stop, no parity, 8 bits/char, ; divide down of 16 ;RTS high, enable Tx/Rx data for an 8253 chip ;Port number for 8253 mode</pre>
0090 0091 0093 0094 0095 0094 0099 009A 009B	02 0700 00 06 000000 42 6E 25 DF 01	00493 00494 00495 00495 00497 00497 00499 00500 00500 00500 00503 00504 00505 00505 00505 00505 00505 00508 00509 00510	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$104 0010\$0 0010\$0	0010B \$11\$10B 0101B	;Example ;Example	Port number for counter Number of bytes to output Slabel used by utilities Port number of 00 terminates stream data for an 3251A chip Port number for 3251A Number of bytes Dummy bytes to get chip ready Reset and raise DTR Stop, no parity, 8 bits/char, Stoph, enable Tx/Rx data for an 8253 chip Port number for 8253 mode Number of bytes to output
0090 0091 0093 0094 0095 0096 0099 009A 009B	02 0700 00 06 000000 42 6E 25 DF 01	00493 00494 00495 00495 00497 00498 00500 00502 00502 00503 00504 00505 00506 00505 00506 00507 00508	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$104 0010\$0 0010\$0	0010B \$11\$10B	;Example ;Example	<pre>;Port number for counter ;Number of bytes to output ;Label used by utilities ;9600 Baud (based on 16x divider) ;Port number of 00 terminates stream data for an 8251A chip ;Port number for 8251A ;Number of bytes ;Dummy bytes to get chip ready ;Reset and raise DTR ;1 stop, no parity, 8 bits/char, ; divide down of 16 ;RTS high, enable Tx/Rx data for an 8253 chip ;Port number for 8253 mode ;Number of bytes to output ;Select:</pre>
0090 0091 0093 0094 0095 0094 0099 009A 009B	02 0700 00 06 000000 42 6E 25 DF 01	00493 00494 00495 00495 00497 00502 00501 00502 00503 00503 00504 00505 00506 00507 00508 00507 00508 00507	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$104 0010\$0 0010\$0	0010B \$11\$10B 0101B	;Example ;Example	Port number for counter Number of bytes to output Slabel used by utilities Port number of 00 terminates stream data for an 8251A chip Port number for 8251A Number of bytes Dummy bytes to get chip ready Reset and raise DTR 1 stop, no parity, 8 bits/char, 3 divide down of 16 RTS high, enable Tx/fx data for an 8253 chip Port number for 8253 mode Number of bytes to output Select: 5 Counter 2
0090 0091 0093 0094 0095 0094 0099 009A 009B	02 0700 00 06 000000 42 6E 25 DF 01	00493 00494 00495 00495 00497 00497 00498 00498 00500 00501 00503 00503 00504 00503 00506 00507 00506 00507 00500 00511 005112 00512	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$104 0010\$0 0010\$0	0010B \$11\$10B 0101B	;Example ;Example	<pre>;Port number for counter ;Number of bytes to output ;Label used by utilities ;9600 Baud (based on 16x divider) ;Port number of 00 terminates stream data for an 8251A chip ;Port number for 8251A ;Number of bytes ;Dummy bytes to get chip ready ;Reset and raise DTR ;1 stop, no parity, 8 bits/char, ;1 stop, no parity, 8 bits/char, ;1 stoph, enable Tx/Rx data for an 8253 chip ;Port number for 8253 mode ;Number of bytes to output ;Select: ; Counter 2 ; Load LS byte first</pre>
0090 0091 0093 0094 0095 0094 0099 0094 0099 0098 0098 0099 0098	02 0700 06 060 000000 42 6E 25 DF 01 B6	00493 00494 00495 00495 00496 00497 00500 00501 00502 00503 00504 00503 00504 00505 00506 00507 00508 00509 00510 00511 00512 00513	DB DO\$Baud\$Rate\$C DW DB DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 01810f 01810f 0010\$0 00FH 1 10\$11f	0010B \$11\$10B 0101B	;Example ;Example	<pre>:Port number for counter :Number of bytes to output :Label used by utilities ;%GOO Baud (based on 16x divider) ;Port number of 00 terminates stream data for an 8251A chip ;Port number for 8251A ;Number of bytes ;Dummy bytes to get chip ready ;Reset and raise DTR ;1 stop, no parity, 8 bits/char, ; divide down of 16 ;RTS high, enable Tx/Rx data for an 8253 chip ;Port number for 8253 mode ;Number of bytes to output ;Select: ; Counter 2 ; Load LS byte first ; Mode 3, binary count</pre>
0090 0091 0093 0094 0095 0094 0099 009A 009B	02 0700 00 06 000000 42 6E 25 DF 01 B6 DE	00493 00494 00495 00495 00497 00497 00498 00498 00500 00501 00503 00503 00504 00503 00506 00507 00506 00507 00500 00511 005112 00512	DB DO\$Baud\$Rate\$C DW DB D1\$Initialize\$ DB DB DB DB DB DB DB DB DB DB DB DB DB	2 onstant: 0007H 0 Stream: 0DDH 6 0,0,0 0100\$0 01\$104 0010\$0 0010\$0	0010B \$11\$10B 0101B	;Example ;Example	<pre>;Port number for counter ;Number of bytes to output ;Label used by utilities ;9600 Baud (based on 16x divider) ;Port number of 00 terminates stream data for an 8251A chip ;Port number for 8251A ;Number of bytes ;Dummy bytes to get chip ready ;Reset and raise DTR ;1 stop, no parity, 8 bits/char, ;1 stop, no parity, 8 bits/char, ;1 stoph, enable Tx/Rx data for an 8253 chip ;Port number for 8253 mode ;Number of bytes to output ;Select: ; Counter 2 ; Load LS byte first</pre>

Figure 8-10. (Continued)

OchD018Bud@RateConstant; D038H1200 badd (based on 160 divider) iPort number of 00 for divider) iPort number of 00 for divider) iPort number of 00 for divider)OchD0500D0511 tiliziesStream; D0501TExample data for an BCS14 chip iPort number of 02514OchD0502D05D01011100Och00525D0D01001010Och00525D0010001010Och00527D0010001010Och00527D0010001010Och00530D000541Och00531D0ODFHItalia1511(10001511(1000Och00531D0Och00532D0Och00533D0OA00533D0OA00534D0OA00535D0OA00537D0OA00537D0OA00539D0OA00539D0OA00539D0OA00534D0OA00535D0OAD0OA00539D0OAD0OA00539OAD0OA00539D000539D000539D000539D000539D000539D000539D000540D000559D000541O0550D0O0550D0D0								······································
Code 0         DB         0         Port number of 00 trainates stream           Code 0         D2011111111*****************************		00						And a state of the second state of the second
CAA BD       COSS20       DefinitializetStream:       :Example data for an B251A chip         CAA BD       COSS2       DefinitializetStream:       :Example data for an B251A chip         CAA BD       COSS2       DE       0.0.0       :Plummy byte to set chip ready         CAA BD       COSS2       DE       0.0.0       :Plummy byte to set chip ready         CAA BD       COSS2       DE       0.000000       :Plummy byte to set chip ready         CAA BD       COSS2       DE       0.0104010E       :Plumby byte to set chip         CAA BD       COSS2       DE       0.0104010E       :Plumby byte to set chip         CAA BD       COSS3       DE       0.0104010E       :Plumby byte to set chip         CAA BD       COSS3       DE       0.014011E       :Plumby byte to set chip         CAA BD       COSS3       DE       0.01411E       :Plumby byte to set chip         COAS DE       COSS3       DE       0.01411E       :Plumby byte to set chip         COAS DE       COSS3       DE       0.01411E       :Plumby byte to set chip         COAS DE       COSS3       DE       0.01411E       :Plumby byte to set chip         COAS DE       COSS3       DE       0.01411E       :Plumbyte of bytes to set chip								1200 baud (based on 16x divider)
OCAL         D2211111111111111111111111111111111111	00A3 00			DB	0			;Port number of 00 terminates stream
CoA4 DD       COS22       DB       ODDH       :Fort number for S251A         CoA5 66       COS23       DB       O.CONCOLOR       DBumm briss to senihip ready         CoA6 66       COS23       DB       O.CONCOLOR       DBumm briss to senihip ready         CoA6 46       COS27       DB       O.CONCOLOR       I storn on serity. B bitcher.         CoA8 26       COS27       DB       COS0010B       iffS high, enable first counter         CoA8 26       COS27       DB       COS0010B       iffS high, enable first counter         CoA8 27       COS33       DB       OCF       iffS high, enable first counter         CoA9 01       COS33       DB       OCF       iffS high, enable first counter         CoA9 02       COS33       DB       IffS high, enable first counter       iffS high, enable first counter         CoA9 02       COS33       DB       ODFH       :Faunter of bites to counter         CoA9 02       COS33       DB       ODFH       :Fort number for S253 mode         CoA9 02       COS33       DB       ODFH       :Fort number for S253 mode         CoA9 02       COS33       DB       ODFH       :Fort number for S251A         CoA9 02       COS33       DB       ODFH       <								
COASS 00000       COSSS 00000       DB 6       iNumber of bytes to up this ready         COAS 00000       COSSS 0000       DB 010000100       react and rais Dis ready         COAS 00000       COSSS 0000       DB 010000100       react and rais Dis ready         COAS 00000       COSSS 0000       DB 010000100       react and rais Dis ready         COAS 00000       COSSS 0000       DB 010001000       react and rais Dis ready         COAS 00000       COSSS 00000       DB 0000100       react and rais Dis ready         COAS 00000       COSSS 00000       DB 00000100       react and rais Dis ready         COAS 00000       COSSS 00000       DB 00000100       react and rais Dis ready         COAS 00000       COSSS 00000       DB 000001000000       react and rais Dis ready         COAS DE 00000000000000000000000000000000000							;Example	
COAD 2       DES 0.0.0       iDLamp bytes to set chip ready         COAD 4       COESS       DE 01000000       if tep; no parity. Bits/char.         COAD 4       COESS       DE 01000000       if tep; no parity. Bits/char.         COAD 4       COESS       DE 00100000       if tep; no parity. Bits/char.         COAD 25       COESS       DE 00100000       if tep; no parity. Bits/char.         COAD 26       COESS       DE 00100000       if tep; no parity. Bits/char.         COAD 26       COESS       DE 00004       if tep; no parity. Bits/char.         COAD 26       COESS       DE 00004       if tep; no parity. Bits/char.         COAD 26       COESS       DE 00004       if tep; no parity. Bits/char.         COAD 26       COESS       DE 0004       if tep; no parity. Bits/char.         COAD 26       COESS       DE 00064       if tep; no parity. Bits/char.         COAS 26       COESS       DE 00064       if tep; no parity. Bits/char.         COAS 26       COESS       DE 00064       if tep; no parity. Bits/char.         COAS 27       DE 00037       DE 00064       if tep; no parity. Bits/char.         COAS 26       COESS       DE 00064       if tep; no parity. Bits/char.         COESS 26       DE 00064				~~				
COAR 42       COS25       DB       Di000010B       ;Reset and raise DTR         COAR 42       COS27       DB       Oli000101B       ;I ston, no parity, B bits/char, i divide down fo         COAR 42       COS27       DB       Oli000101B       ;I ston, no parity, B bits/char, i divide down fo         COAR 50       COS30       JER angle data for an S23 chip       ;PEnample data for an S23 chip         COAR 50       COS31       DB       10000       introduct rest to output         COAR 50       COS33       DB       1111001BOB       ;PEnample data for an S23 chip         COAR 50       COS33       DB       1111001BOB       ;PEnample data for an S23 chip         COAR 50       COS35       DB       00000       ; Load 15 byte first         COAR 50       COS35       DB       00000       ; Load 15 byte first         COAR 50       DE       DE       2       ; Number of Viet to output         COB3 00       COS44       DE       000000       ; Number of Viet to output         COB3 00       COS44       JE       This following table is used to determine the maximum         COS45       JU value for each character position in the ASCII time       int the ASCII time         COS46       JU value for each character position in the ASCII tim								
ODAA dEODS26DBOls106118108: stor, no party, 8 bits/char, i divide down of 16ODAB 25ODS26DBOLD00101BiRTS high, enable is for an 8253 chip i for thumber for an 8253 modeODAD 10ODS32DBIiFExample data for an 8253 chip i Fort number for stass modeODAD 10ODS32DBIiFExample data for an 8253 modeODAD 10ODS32DBIiFert number for an 8253 modeODAD 10ODS32DBIiFert number for S233 modeODAD 10ODS33DBODEHiFert number for contertODAT 10ODS34DBODEHiFort number for contertODAT 10ODS35DBODEHiFort number for contertODB1 2600ODS36DBODEHiFort number of 00 terminates streamODB3 00ODS41DBOiFort number of 00 terminates streamODS3 00ODS41DBOiFort number for conterODS3 00ODS41DBOiFort number for conterODS3 00ODS41DBOiFort number for an 8251 timeODS3 00ODS41DBOiFort number for an 8251 timeODS3 00ODS41DBOiFort number for an 8251 timeODS4 00ODS42The long irre contiguration block so that the clockODS4 11Cost 11IFort number for 227 for a 12-hour clockODS4 00ODS52DB0iFort number for an 12-hour clockODS53 07DBOFFHiFort nu				~~				;Dummy bytes to get chip ready
0082 2       00527       if divide down of 16         0082 5       00527       if Stample data for an 823 chip         0082 6       00531       DB       00FH         0082 7       if Stample data for an 823 chip       if of state is to output         0082 6       00533       DB       1811801180B       if State is to output         0082 7       00533       DB       1811801180B       if State is to output         0084 76       00533       DB       1811801180B       if Mode 3, binyr count         0085 02       00533       DB       DDEH       if Port number of bres to output         0085 02       00533       DB       DDEH       if Port number of bres to output         0085 02       00533       DB       DDEH       if Port number of bres to output         0085 02       00544       DB       0       if Port number of bres to output         0085 02       00544       if This following table is used on if an and the asset is processed backwards to correspond         00544       if the table is processed backwards to correspond       if uith the ASCII time         00545       if uith the ASCII time       if uith the acerry-and-reset-to-zero         00546       DB       0       if Terminator* <td< td=""><td>00A9 42</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Reset and raise DTR</td></td<>	00A9 42							Reset and raise DTR
OAB 25     Object     DB     O01000101B     #RTS high, enable fait TX/RX       OAAD DF     (Construction)     (Construction)     (Construction)       OAAD 01     OOSS3     DB     (DFH)     (FAmple fait for an 0253 chip       OAAD 01     OOSS3     DB     (DFH)     (FAmple fait for an 0253 chip       OAAD 01     OOSS3     DB     (IFI)     (FAmple fait for an 0253 chip       OAAD 01     OOSS3     DB     (IFI)     (FAmple fait fait fait fait fait fait fait fait	00AA 6E	00	526	DB	01\$10\$1	1\$10B		
000000000000000000000000000000000000								
OCCCFExample data for an 223 chip iPort number for 223 modeOCAL DFOCCCOCAL DFD28Baud#Bate#Constant:OCAL DFD28Baud#Bate#Constant:OCAL DFD28Baud#Bate#Constant:OCAL DFD28Baud#Bate#Constant:OCAL DFOCCCOCAL DFOCCCOCAL DFValue for each character position in the ASCII timeOCAL DFOCCCOCAL DFValue for each character position in the ASCII timeOCAL DFOCCCOCAL DFOCCCOCAL DFOCCCOCAL DFOCCCOCAL DFOCCCOCAL DFValue dove (except the ""."). Note - this table isOCAL DFValue dove (except the ""."). Note - this table isOCAL DFOCCCOCAL DFOCCCOCAL DFOCCCOCAL DFOCCCOCAL DFOCCCOCAL DFOCCCOCAL DFDFOCAL DFOCCCOCAL DFDFOCAL DFOCCCCOCAL DFDFOCAL DFOCCCCOCAL DF <t< td=""><td>00AB 25</td><td></td><td></td><td>DB</td><td>0010\$01</td><td>01B</td><td></td><td>;RTS high, enable Tx/Rx</td></t<>	00AB 25			DB	0010\$01	01B		;RTS high, enable Tx/Rx
OOAD DF         OOS31         DB         ODFH         :Port number of bytes to output           OOAD DF         OOS32         DB         11811801180B         :Select         :Select           OOAD DF         OOS32         DB         11811801180B         :Select         :Select           OOAD DE         OOS32         DB         ODEFH         :Select         :Select           OOAD DE         OOS32         DB         ODEFH         :Fort number of output         :Select           OOB3 OO         OOS36         DB         ODEFH         :Fort number of OO terminates stream           OOB3 OO         OOS36         DB         ODEFH         :Fort number of OO terminates stream           OOB3 OO         OOS40         DW         OOS40         :Select         :Select           OOB3 OO         OOS40         DW         OOS40         :Select         :Select         :Select           OOS30 OO         OOS40         DW         OOS40         :Select         :Selec								
OADE 01ODS22DB1Number of bytes to outputOADE F6ODS34DB15811901190B1581eetrOADE DEODS37DBODEH15041eetrOADE DEODS37DBODEH15041eetrOADE DEODS37DBODEH15041eetrOADE DEODS37DBODEH1200 baud (based on 16x divider)ODB0 02ODS37DDBDDEH1200 baud (based on 16x divider)ODB1 02ODS34JThis following table is used to determine the maximumODS3 00ODS41DBO1200 baud (based on 16x divider)ODS3 00ODS41DBO1200 baud (based on 16x divider)ODS3 00ODS41DBO1200 baud (based on 16x divider)ODS4 1This following table is used to determine the maximum00546 1value for ach character position in the ASCII timeODS4 00ODS54can be set "permanelly" to either 12 or 24 hour format.00557 1ODS5 1With the ASCII time.CorrespondingODS5 2DB0:"Terminator"ODS5 3334ODS55DBOFFHODS5 334ODS55DBOFFHODS5 344DBOS:"Terminator"ODS5 344DBOS:"Terminator"ODS5 4050DBOFFH:"Ship" characterODS5 57DBOFFH:"Ship" characterODS5 67DBOFFH:"Ship" characterODS5 70DBOFFH:"Ship" character </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>;Example</td> <td>e data for an 8253 chip</td>							;Example	e data for an 8253 chip
ODAE F6       OD335       DB       1181101180B       iSelect:         OD35       i       Load LS brtefirst         OD35       DB       ODEH       incomparent         OD30       D2       OD35       DB       DE         OD30       D2       D2       incomparent       incomparent         OD30       D2800       D2800       D2800       incomparent       incomparent         OD31       D2000       D2800       D2800       incomparent       incomparent         OD31       D2000       D000       D4       D0000       incomparent       incomparent         OD31       D2000       D2000       D4       D0000       incomparent       incomparent         OD31       D2010       D4       D0000       incomparent       incomparent       incomparent         OD32       D2000       D300       D300       incomparent       incomparent       incomparent         OD33       D2000       Incomparent       incomparent       incomparent       incomparent         OD33       Incomparent       Incomparent       incomparent       incomparent       incomparent         OD34       Incomparent       Incomparent       incomparent<								
00083 000       i       Counter 3         00047 DE       00036       DB       i       Mode 1, S, Dinary counter 3         00047 DE       000350       D28Baud\$FateConstant:       i       Mode 7, S, Dinary counter 3         0015 000       000360       D28Baud\$FateConstant:       i       Number of votes to output         0015 300       00041       DB       0       iFort number of 00 terminates stream         0015 000       000541       DB       0       iFort number of 00 terminates stream         0015 000       000542       if if if if if if if if if if if if if i								
0035       i       Load LS byte first         00047 DE       00037       DB       00EH       iPort number for counter         00080       00337       DB       00EH       iPort number for counter         0080       00337       DB       00BH       iPort number for counter         0080       00337       DB       00BH       iPort number for counter         0080       00344       j       iport number of OO terminates stream         00542       i       This following table is used to determine the maximum         00544       i       This following table is used to determine the maximum         00545       i       value above (accept the "i"). Note this table is         00546       i       value above (accept the "i"). Note this table is         00547       i       out above (accept the "i"). Note this table is         00548       i       i       table is processed backwards to correspond         00551       i       NOTE: The table is processed backwards to correspond         00553       i       DB       i"the ASCII time at which a carry-and-reset-carer         00554       DB       0354       icharacter represents the value for the carestor         00555       DB       0354       icharacter <td>00AE F6</td> <td></td> <td></td> <td>DB</td> <td>11\$11\$0</td> <td>11\$08</td> <td></td> <td></td>	00AE F6			DB	11\$11\$0	11\$08		
OOAF DE       00536       j       Mumber of counter         OOBD 02       00339       DB       2       jRumber of bytes to output         OOBD 02       00339       DB       2       jRumber of bytes to output         OOBD 02       00340       DW       003941       j1200 baud thased on 16x divider)         OOBD 00       00540       DW       003941       j1200 baud thased on 16x divider)         0083 00       00543       ;       This following table is used to determine the maximum         00543       ;       This following table is used to determine the maximum         00545       ;       value above (except the "!"). Note this table is         00556       ;       value above (except the "!"). Note this table is         00550       ;       in the long term configuration block so that the clock         00551       ;       out the the ASCII time.         00552       ;       Each character represents the value for the corresponding         00552       ;       DB       0       ;"Terminator"         00552       ;       DB       0       ;"Terminator"         00553       ;       DB       0       ;"Terminator"         00554       ;       sholid occur.       ;"Terminator								
OORF DEODESTDBODEH:Fort number for counterOORD 02OCTSPD2EBaud9RateEConstant:;1200 baud (based on 16x divider)OOBS 00OCTSPD2EBaud9RateEConstant:;1200 baud (based on 16x divider)OOBS 00OCTSIDB0OOBS 00OCTSIDB0OCTSIOCTSIThis following table is used to determine the maximumOCTSIOCTSIvalue above cancer position in the ASCII timeOCTSI: value above cancert the ":"). Note this table isOCTSI: value above cancert the ":"). Note this table isOCTSI: value above cancert mosition in the ASCII timeOCTSI: value above cancert mostion in the ASCII timeOCTSI: value above cancert mostion in the clockOCTSI: value above cancert mostion in the ASCII timeOCTSI: value for the correspondOCTSI: NOTE: The table is processed backwards to correspondOCTSI: DB: "Terminator"OCTSI: DB: "Terminator"OCTSI: DB: "Terminator"OCTSI: DB'fffOCTSI: DB'fffOORD 3C3A: OCTSIOOBD 3C3A: Variables for the real time clock and watchodgOOTSI: Variables for the real time clock								
0000       0000       0000       10000       100000         0000       000000       000000       1000000       10000000         00000       0000000       00000000       100000000       100000000000         000000       00000000000000000       1000000000000000000000000000000000000								
00537         D289aud9Rate8Constant: DW         ;1200 baud (based on 16x divider) ;Port number of 00 terminates stream 00543           0083         000         00541         DB         0         ;Port number of 00 terminates stream 00543           00543         ;         This following table is used to determinate the maximum 00545         ; value for each charler position in the ASCII time 00546           00543         ; value above (except the ";"). Note this table is 00546         ; value above (except the ";"). Note this table is 00550           00545         ; value for each charler position in the ASCII time 00550         ; Uith the ASCII time 00550         ; Uith the ASCII time 00550           00546         ; uith the ASCII time 00555         ; DB         0         ; "Terminator"           00547         ; DB         0         ; "Terminator"           00550         ; DB         0         ; "Terminator"           00551         ; DB         0         ; "Terminator"           00552         DB         '24'         ; Charge to '22' for a 12-hour clock           00853         00556         DB         '24'         ; Charge to '22' for a 12-hour clock           00857         DB         '24'         ; Charge to '22' for a 12-hour clock           00858         00550         DB         '24'         ; Charge to								Port number for counter
OOB1 3800       OOS40       DW       OO38H       ;1200 hand (based on 16x divider)         OOB3 00       OO541       DB       0       ;Port number of OUterminates stream         OO542       OO544       ;       This following table is used to determine the maximum         OO545       ;       Value for each character position in the ASCII time         OO547       ;       Value for each character position in the ASCII time         OO547       ;       value for each character position in the ASCII time         OO549       ;       can be set "permanently" to either 10 correspond         OO550       ;       NOTE: The table is processed backwards to correspond         OO551       ;       with the ASCII time at which a carry-and-reset-to-zero         OO552       ;       DE       ;"Terminator"         OO553       ;       DE       ;"Terminator"         OO554       DE       OFFH       ;"Skif" character         OO555       ;       DE       'Skif" character         OOB5       OOS56       DE       'Skif" character         OOB5       OOS56       DE       'Skif" character         OOB5       OOS51       DE       'Skif" character         OOB53       OO561       DE       'S	0080 02							Number of bytes to output
0005 00       00542 00543 00544 00544 00545       00542 00545       ipPort number of 00 terminates stream 00545 00545         00544 00545       1       This following table is used to determine the maximum 00545       00547 00545         00545       1       value above (except the "1"). Note this table is 00547       1         00547       1       in the long term configuration block so that the clock 00549       1         00549       1       in the table is processed backwards to correspond 00552       1         00549       1       with the ASCII time 00553       1       character represents the value for the corresponding 00553         00540       00552       is character in the ASCII time 00555       1       0       1         00551       00       1       "Terminator"       12-hour clock       12-hour clock         00552       1       should occur.       1       "Terminator"       12-hour clock         00553       1       0       1       "Terminator"       12-hour clock       12-hour clock         0058       0056       0       1       "Terminator"       12-hour clock       12-hour clock         0058       0056       0       1       "Terminator"       12-hour clock       12-hour clock         0058       <	1							ويتصفينهم بالمحاصر فتأسيط ممصار
00542       00543       ;       This following table is used to determine the maximum         00544       ;       This following table is used to determine the maximum         00545       ;       value above (except the ";"). Note this table is         00546       ;       value above (except the ";"). Note this table is         00547       ;       in the long term configuration block so that the clock         00548       ;       can be set "permanently" to either 12 or 24 hour format.         00550       ;       NDTE: The table is processed backwards to corresponding         00552       ;       Each character represents the value for the corresponding         00553       ;       obtaid occur.         00554       00555       DB       0         0055       DB       0       ;"Terminator"         0055       DB       0541       ; fchange to '23' for a 12-hour clock         0055       DB       0541       ; fchange to '23' for a 12-hour clock         0055       DB       0541       ; fchange to '23' for a 12-hour clock         0055       DB       0541       ; fchange to '23' for a 12-hour clock         0058       3034       00550       DB       0541         0058       00552       DB								
00543       ;         00545       ;       This following table is used to determine the maximum         00545       ;       value for each character position in the ASCII time         00547       ;       in the long term configuration block so that the clock         00547       ;       in the long term configuration block so that the clock         00547       ;       in the long term configuration block so that the clock         00547       ;       in the long term configuration block so that the clock         00550       ;       with the ASCII time.         00552       ;       Each character represents the value for the corresponding         00552       ;       character in the ASCII time.         00551       ;       och character represents the value for the corresponding         00552       ;       bb 0       ;"Terminator"         00553       ;       character in the ASCII time at which a carry-and-reset-to-zero         00554       ;       should occur.       costs         00555       ;       DB       0;"Terminator"         00557       CBB122/24Clock:       rCharge to '23' for a 12-hour clock         00576       DB       0;"Terminator"       costs         0058       00562       DB       0;"T	0083 00			DB	0			Port number of 00 terminates stream;
00544       i       This following table is used to determine the maximum         00545       i       value above (except the "!"). Note — this table is         00546       j       unteracter position in the ASCII time         00547       j       in the long term configuration block so that the clock         00548       j       can be set "permanently" to either 12 or 24 hour format.         00559       j       NOTE: The table is processed backwards — to correspond         00551       with the ASCII time.       store of the corresponding         00553       character represents the value for the corresponding         00554       character represents the value for the corresponding         00555       DB       0         00556       DB       0         00557       CD8128248Clock:       character         0058       00559       DB       off/f         0058       00559       DB       off/f         0058       00561       DB       off/f         0058       00562       DB       off/f         0058       00563       UndateTime#End:       jUsed when updating the time         0058       00562       DB       fo/f       jMaximu minutes are 59         00563       Undat	1							
00545       i       value for each character position in the ASCII time         00547       i       in the long term configuration block so that the clock         00547       i       in the long term configuration block so that the clock         00547       i       in the long term configuration block so that the clock         00550       ;       NOTE: The table is processed backwards — to correspond         00551       ;       with the ASCII time.         00552       ;       Each character represents the value for the corresponding         00553       ;       character in the ASCII time at which a carry-and-reset-to-zero         00553       ;       block       character         00550       ;       DB       (zithig) character         00551       ;       should occur.       character         00552       ;       DB       (zithig) character         00551       DB       (zithig) character       character         00552       ;       DB       (zithig) character         00554       ;       ishuid occur.       character         00564       ;       :       ishuid occur.         00565       iddate\$Time\$End:       ;       ishuid occur.         00566       ;       <							·	
00546       ;       value above (except the "!"). Note this table is         00547       ;       in the long term configuration block so that the clock         00548       ;       can be set "permanently" to either 12 or 24 hour format.         00550       ;       NOTE: The table is processed backwards to correspond         00551       :       with the ASCII time.         00552       :       Each character represents the value for the corresponding         00553       :       character represents the value for the corresponding         00553       :       character represents the value for the corresponding         00554       :       should occur.         00555       :       DB       :"Terminator"         00557       CBB12824%Clock:       :"Terminator"         00557       CBB12824%Clock:       :"Terminator"         0058       3634       00560       DB       '34'         0058       :       DB       '34'       :Tharing the time is ecold         0058       :       DB       '61'       :Thaximum seconds are 59         0058       :       :       DB       '14'       :Thaximum seconds are 59         0058       :       :       :       :       :				This fo	llowing	table is	used to	determine the maximum
00547       ;       in the long term configuration block so that the clock         00548       ;       can be set "permanently" to either 12 or 24 hour format.         00550       ;       NOTE: The table is processed backwards to correspond         00551       ;       with the ASCII time.         00552       :       Each character represents the value for the corresponding         00552       :       Character represents the value for the corresponding         00552       :       Character represents the value for the corresponding         00552       :       Cataracter in the ASCII time at which a carry-and-reset-to-zero         00553       :       Character represents the value for the correspond         00554       :       should occur.         00555       :       DB       :"Terminator"         00564       :       DB       OFH       :"Baying character         00556       :       DB       OFH       :"State         00561       DB       OFH       :"Baying character       :"It was the inter so interest         00564       :       itime       :!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!	1			value f	or each	characte	r positio	on in the ASUII time
00550;NOTE; The table is processed backwards to correspond00551;with the ASCII time.00552;Each character in the ASCII time at which a carry-and-reset-to-zero00533;character in the ASCII time at which a carry-and-reset-to-zero00554;should occur.00555;DB;"Terminator"00556DB0;"Terminator"005700557DB05740058333400558DB0057r00570058ASA005600058GS4100058GS4100058GS4100058GS4100058GS41000564iyinsum seconds are 5900565iVariables for the real time clock and watchdog00566iVariables for the real time clock and watchdog0057100572itimer0057200574isecond0057400575itimer0057500576iyitck per elapsed second0057600577ii00577iii00578iyitchick@cunt00579ii00570ii00571ii00572ii00573ii00574ii00575ii00576ii00577i<	1			value a	bove (ex	cept the	":"). No	DTE THIS TADIE 15
00550;NOTE; The table is processed backwards to correspond00551;with the ASCII time.00552;Each character in the ASCII time at which a carry-and-reset-to-zero00533;character in the ASCII time at which a carry-and-reset-to-zero00554;should occur.00555;DB;"Terminator"00556DB0;"Terminator"005700557DB05740058333400558DB0057r00570058ASA005600058GS4100058GS4100058GS4100058GS4100058GS41000564iyinsum seconds are 5900565iVariables for the real time clock and watchdog00566iVariables for the real time clock and watchdog0057100572itimer0057200574isecond0057400575itimer0057500576iyitck per elapsed second0057600577ii00577iii00578iyitchick@cunt00579ii00570ii00571ii00572ii00573ii00574ii00575ii00576ii00577i<				in the	long ter	m config	uration b	plock so that the clock
00550;NOTE: The table is processed backwards to correspond00551;With the ASCII time.00552;Each character represents the value for the corresponding00553;character in the ASCII time at which a carry-and-reset-to-zero00554;should occur.00557C0812824€Lock:;"Terminator"00557C0812824€Lock:0055900853334005590085G0561DB0085363A005600085641';Maximum minutes are 590088363A0056200854Udate\$Time\$End:00555;00565;00563Udate\$Time\$End:00564;00565;00565;00566;00567;00568;00568;00569RTC\$Tick\$Sper\$Second0056;0056;00570C037200567;00568;00579RTC\$Tick\$Count00570;00571RTC\$Tick\$Count0057;00572;00573;00574;00575;00575;0057;0058;0059;0059;0059;0059;0059;0059;0059;0057				can be	set "per	manentlý	" to eith	her 12 or 24 hour format.
00551iwith the ASCII time.00552iEach character in the ASCII time at which a carry-and-reset-to-zero00553icharacter in the ASCII time at which a carry-and-reset-to-zero00554ishould occur.00555ioctor.00556DBi0057CD\$125248Clock:00580057CD\$125248Clock:00580057CD\$125248Clock:00580057DB005800560DB0058633400561005800561DB0058633400562005836340056200583634005620058iVariables for the real time clock and watchdog00566iVariables for the real time clock and watchdog0057100572i timer0058200567i timer005800570RTC\$Tick\$ScuntDB0057100574i second will elapse0057200574i second will elapse0057300575i second will elapse0057400576i watchdog timer set)0057500577i watchdog timer set)00576i watchdog structure!00577i watchdog scunt hits 000578i watchdog structure!00579i watchdog scunt hits 000570i watchdog scunt hits 000571i watchdog structure!00572i bl bransferred if the00573i bl bransferred if the <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>								
00552       :       Each character in the ASCII time at which a carry-and-reset-to-zero         00553       :       should occur.         00554       :       should occur.         00555       DB       0       ;"Terminator"         00557       CB#128248Clocki       ;Change to '23' for a 12-hour clock         00853334       00559       DB       OFF         0085       TF       00557       CCB#128248Clocki         0085       :       :       ::"Stip" character         0085       :       DB       :"Stip" character         0085       :       :       :"Stip" character         0085       :       :       :"Stip" character         0085       :       :       ::"Stip" character         0085       :       :       ::"Stip" character         0085       :       :       ::"Stip" character         0085       :       ::"Stip" character       ::"Stip" character         0085       :       ::::::::::::::::::::::::::::::::::::							essed bad	ckwards to correspond
00553       ;       character in the ASCII time at which a carry-and-reset-to-zero         00554       ;       should occur.         00555       ;       bb       0       ;"Terminator"         00557       ;       DB       0       ;"Terminator"         00557       ;       DB       0       ;"Terminator"         00557       ;       DB       (arracter)       ;"Terminator"         00557       ;       DB       (arracter)       ;"Skip" character         0057       ;       DB       (arracter)       ;"Maximum minutes are 59         0058       ;       DB       (arracter)       ;Maximum seconds are 59         00561       DB       :       ;Maximum seconds are 59         00562       Update\$Tim#End:       ;Used when updating the time         00563       ;Update\$Tim#End:       ;Used when updating the time         00564       ;       Variables for the real time clock and watchdog         00565       ;       timer         00566       ;       Variables for the real time clock and watchdog         00567       ;       timer         00568       ;       ;         00569       ;       ;         0								
00554       ;       should occur.         0084       00555       DB       0       ;"Terminator"         0085       00557       CB\$12\$2424Clock:       ;Change to '23' for a 12-hour clock         0085       3334       00558       DB       off*         0085       363A       00560       DB       off*         0088       363A       00561       DB       off*         0088       363A       00562       UpdateSTime%End:       ;Used when updating the time         00564       ;       00564       ;       UndateSTime%End:       ;Used when updating the time         00564       ;       variables for the real time clock and watchdog       00567       ;       timer         00565       ;       variables for the real time clock and watchdog       ;       second will elapse         0080       00567       ;       timer       ;       second will elapse         0085       30       00573       RTC%Tick%Count       DB       60       ; Number of real time clock         00857       ;       iticks peresize       iticks pere elapsed second       ; second will elapse         00850       :       ;       second will elapse       ; will bese       ; will bese <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>								
00850         00555         j           0085         00557         CB#1224%Clock:         ;Change to '23' for a 12-hour clock           00857         FF         00559         DB         OFFH         ;"Skip" character           0085         3334         00560         DB         '61'         ;Maximum nutues are 59           0086         00561         DB         OFFH         ;"Skip" character           0086         00562         DB         '61'         ;Maximum seconds are 59           0086         00563         Update%Time%End:         ;Used when updating the time           00564         ;         Variables for the real time clock and watchdog           00565         ;         00566         ; ticks per elapsed second           00566         ;         Variables for the real time clock and watchdog           00567         ;         timer         ;           00568         RTC%Tick%Scount         DB         60         ;Number of real time clock           00570         RTC%Tick%Count         DB         60         ;Residual count before next           00570         00577         RTC%Hatchdog%Address         DW         ; Hatchdog timer tick count           00576         ;         uill be transferred						e ASCII	time at v	which a carry-and-reset-to-zero
0084 0000556DB0;"Terminator"0085 333400558CB122446Clock:;Change to '23' for a 12-hour clock0087 FF00359DB0FFH;"%kip" character0086 363A00360DB '61';Maximum minutes are 590086 363A00360DB '61';Maximum seconds are 590088 363A00362Update9Time%End:;Used when updating the time00563Update9Time%End:;Used when updating the time00564 :00566 :Variables for the real time clock and watchdog00565 :00566 :timer00566 :Variables for the real time clock and watchdog00567 :timer00568 :00567 :008B 3C00570008E 3C00571008F 000000573008F 0000005730077second will elapse008F 00000057400570 :; will be transferred if the0087 :; will be transferred if the0088 :; Function key table0089 :; DB0080 :; Function key table00858 :; O0586 :00859 :; DB00859 :; DB<				should	occur.			
00857       CB\$12\$24\$Clock:         00857       CB\$12\$24\$Clock:         0087       FF       00559       DB       0FFH       ;"Skip" character         0086       363A       00560       DB       '61'       ;Maximum minutes are 59         0088       363A       00561       DB       off'       ;"Skip" character         0088       363A       00562       DB       '61'       ;"Maximum seconds are 59         0088       363A       00563       Update\$Time\$End:       ;Used when updating the time         00565       ;       00566       ;       Variables for the real time clock and watchdog         00567       ; timer       00566       ;       Update\$Tick\$\$per\$Second       DB       60       ;Number of real time clock         008D       00570       00571       RTC\$Tick\$Count       DB       60       ;Residual count before next       ;         008F       0000       00573       RTC\$Hatchdog\$Address       DW       ; will be transferred if the         002F       0000       00575       RTC\$Hatchdog\$Address       DW       ; watchdog count hits 0         00570       ;       in function key table       ; will be transferred if the       ; watchdog count hits 0					~			
00B5 333400B5DB'34':Change to '23' for a 12-hour clock00B7 FF00559DBOFFH:"Skip" character00B6 363A00560DB'61':Maximum minutes are 5900B8 363A00561DBOFFH:"Skip" character00B8 363A00562Update\$Time\$End::JUsed when updating the time00563Update\$Time\$End::JUsed when updating the time00564:0056500565:immer00566:Variables for the real time clock and watchdog00567:timer00568:Costo00569RTC\$Tick\$Sper\$SecondDB008E 3C00571RTC\$Tick\$Count008E 3C00572:008F 000000573RTC\$Hatchdog\$Count008F 000000574:00257::008F 000000577:00550 ::00257 ::00257 ::00257 ::00257 ::00257 ::00258 :::00259 ::00550 ::00550 ::00577 ::00578 ::00580 ::00581 ::00580 ::00581 ::00583 ::00584 ::00586 ::00586 ::00587 ::00588 ::00588 :: <td>0084 00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>1 . LELIATI</td> <td>ator</td>	0084 00						1 . LELIATI	ator
00B7 FF00B39DB0FFH:"Skip" character00B8 4FF00560DB '6i': Maximum minutes are 5900B8 4FF00561DB0FFH: "Skip" character00E8 363A00562DB '6i': Haximum seconds are 5900563Update%Time%End:: JUsed when updating the time00564:: O0565 :00565:: O0566 :00566:Variables for the real time clock and watchdog00577:timer00586:: ticks per elapsed second00BE 3C00571RTC%Tick%CountDB0057:: ticks per elapsed second00BF 000000573RTC%Hatchdog%CountDW00586:: will be transferred if the00577::: will be transferred if the00578::: will be transferred if the00579::: watchdog count hits 000582 ::::: watchdog count hits 000583 ::::: will be transferred if the00584 :::::: will be transferred if the00586 :::::: will be transferred if the00586 :::::: will be transferred if the00586 :::::: will be transferred if the00586 ::::::00586 ::::::00586 ::<	0005 00						Change	to (23) for a 12-bour clock
00B8 363A       0050       DB       '6'       Haximum minutes are 59         00BA FF       00561       DB       OFFH       ''Skip" character         00B8 363A       00562       DB       '6'       Haximum seconds are 59         00564       i       00565       jUsed when updating the time         00565       i       variables for the real time clock and watchdog         00566       i       Variables for the real time clock and watchdog         00567       timer       00566         00580       7       timer         00581       00570       icks per elapsed second         008B 3C       00571       RTC#TicksSecont       DB       60       ;Number of real time clock         008B 40000       00572       RTC#TicksCount       DB       60       ;Rescond will elapse         008F 0000       00573       RTC#Hatchdog\$Count       DW       0       ;Hatchdog timer tick count         j(0 = no watchdog timer set)       00574       j(0 = no watchdog timer set)       j(0 = no watchdog timer set)         00C1 0000       00575       RTC#Hatchdog\$Address       DW       j ;Hatchdog timer set)       j ;Will be transferred if the         00579       j       00580       Function key table <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></td<>								
00BA FF       00S31       DB       0FFH       i"Skip" character         00BB 363A       00S62       DB       '61'       jHaximum seconds are 59         00S6       Update%Time%End:       jUsed when updating the time         00S65       00S66       Variables for the real time clock and watchdog         00S66       itime*       00S66       itime*         00S67       timer       00S67       itime*         00S8       00S67       time*       iticks per elapsed second         00BE 3C       00S71       RTC%Ticks%count       DB       60       ;Number of real time clock         00BF 0000       00S73       RTC%Hatchdog%Count       DH       0       ;Hatchdog timer tick count         00S74       00S76       if 0       no atchdog timer tick count       ;0 = no watchdog timer set)         00S74       00S77       ; will be transferred if the       ;0 = no watchdog count hits 0         00S79       ;       00S80       ; terminal's function key table       ;         00S81       ;       DB       Second character of sequence emitted by       ;         00S82       ;       DB       Second character of sequence = NOTE: this       )         00S86       ;       ifeld will not be present i								
000BB 363A       00562       DB '6' ' 'Maximum seconds are 59         000563       Update\$Time\$End: 'Used when updating the time         000564       ;         000565       Variables for the real time clock and watchdog         000566       imer         000566       imer         000567       timer         000568       RTC\$Ticks\$per\$Second DB 60 ;Number of real time clock         000BD 3C       00556         00566       00570         000BE 3C       00570         00057       RTC\$Ticks\$count DB 60 ;Number of real time clock         00357       RTC\$Watchdog\$Count DW 0 ;Matchdog timer tick count         00576       is second will glapse         00577       00576         00578       RTC\$Watchdog\$Address DW 0 ;Address to which control         00579       ; will be transferred if the         00579       ;         00579       ;         00580 ; Function key table         00581 ;       This table consists of a series of entries, each one having the         00583 ;       following structure:         00584 ;       DB Second character of sequence = NOTE: this )         00586 ;       UB Third character of sequence = NOTE: this )         00586 ;       10 field								
00563       Update\$Time\$End:       ;Used when updating the time         00564       ;         00565       ;         00566       ;         00567       ; timer         00568       ;         00580       ;         00581       ;         00582       ;         00583       ;         00584       ;         00585       ;         00586       ;         00586       ;         00587       RTC\$Ticks\$per\$Second       DB         60       ; Number of real time clock         00570       ;       second         00580       RTC\$Tick\$Count       DB       60       ; Number of real time clock         00570       ;       second       will playse         0085       ;       second will glayse         0085       ;       second will glayse         0085       ;       will be transferred if         00576       ;       will be transferred if the         00577       ;       watchdog count hits 0         00578       ;       following structure:         00580       ;       following structure:								
00554       ;         00565       ;         00566       ;         00567       ;         00568       ;         00589       RTC\$Ticks\$per\$Second       DB       60         00570       ;       ticks per elapsed second         008E       3C       00572         008F       0000       00573       RTC\$Tick\$Count       DB       60       ;Residual count before next         008F       0000       00572       RTC\$Hatchdog\$Count       DW       0       ;Watchdog timer set)         008F       0000       00574       ;O = no watchdog timer set)       ;O = no watchdog timer set)         00570       ;OS74       ;Will be transferred if the       ;Will be transferred if the         00577       ;Will be transferred if the       ;Will be transferred if the         00579       ;       ;Will be transferred if the         00580       ;Function key table       ;       Watchdog count hits 0         00581       ;       00582       ;       This table consists of a series of entries, each one having the         00581       ;       00585       ;       DB       Second character of sequence =mitted by         00588       ;        ; <td>0088 36</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>Flax Linus</td> <td>a seconds are 57</td>	0088 36						Flax Linus	a seconds are 57
00565       ;         00566       ;       Variables for the real time clock and watchdog         00567       ;       timer         00568       ;         00569       RTC\$Ticks\$per\$Second       DB       60       ;Number of real time clock         00569       RTC\$Ticks\$per\$Second       DB       60       ;Number of real time clock         00570       00570       RTC\$Tick\$Count       DB       60       ;Residual count before next         00571       RTC\$Hatchdog\$Count       DW       0       ;Watchdog timer tick count         00570       00573       RTC\$Hatchdog\$Count       DW       0       ;Watchdog timer tick count         0057       RTC\$Hatchdog\$Address       DW       0       ;Address to which control       ;         00570       :       :       :       :Will be transferred if the       ;         00579       :       :       :Will be transferred if the       :       :       :         00580       :       Function key table       :       :       :       :       :         00581       :       This table consists of a series of entries, each one having the       :       :       :       :       :         00582 <t< td=""><td></td><td></td><td></td><td>uates i meschu</td><td>1</td><td></td><td>; osed w</td><td>ien apaderny che crme</td></t<>				uates i meschu	1		; osed w	ien apaderny che crme
00566 ;       Variables for the real time clock and watchdog         00566 ;       timer         00580 3C       00567 RTC\$Tick\$\$per\$Second DB       Stumber of real time clock         008D 3C       00570       ticks per elapsed second         008E 3C       00571 RTC\$Tick\$Count       DB       60       ; Number of real time clock         008F 0000       00573 RTC\$Watchdog\$Count       DW       0       ; Watchdog timer tick count         008F 0000       00573 RTC\$Watchdog\$Address       DW       0       ; Watchdog timer set)         0010 0000       00575 RTC\$Watchdog\$Address       DW       0       ; Watchdog timer set)         00576       ; will be transferred if the       ; will be transferred if the         00577       ;       ; watchdog count hits 0         00577       ;       ; watchdog count hits 0         00577       ;       ; watchdog count hits 0         00577       ;       ;       ; watchdog count hits 0         00577       ;       ;       ; watchdog count hits 0         00577       ;       ;       ; watchdog count hits 0         00580 ;       ;       ; function key       ;         00581 ;       ;       ; function key       ;         00585 ;								
00567       ; timer         00588       ;         00589       RTC\$Ticks\$per\$Second       DB       60       ;Number of real time clock         00570       00570       ; ticks per elapsed second         008E 3C       00571       RTC\$Tick\$Count       DB       60       ;Residual count before next         008F 0000       00573       RTC\$Watchdog\$Count       DW       0       ;Watchdog timer tick count         008F 0000       00575       RTC\$Watchdog\$Count       DW       0       ;Watchdog timer set)         0051       00576       RTC\$Watchdog\$Address       DW       0       ;Address to which control         00570       ;Will be transferred if the       ;Will be transferred if the       ;Watchdog count hits 0         00577       ;Watchdog count hits 0       ;Watchdog count hits 0       ;Watchdog count hits 0         00577       ;Watchdog count hits 0       ;Watchdog count hits 0       ;Watchdog count hits 0         00580       ;Function key table       ;Watchdog count hits 0       ;Watchdog count hits 0         00581       ;       DB       Second character of sequence emitted by       ;Watchdog count hits 0         00582       ;       DB       Second character of sequence == NOTE: this       )         00588 </td <td>1</td> <td></td> <td></td> <td>Variabl</td> <td></td> <td></td> <td>time clos</td> <td>ak and watchdog</td>	1			Variabl			time clos	ak and watchdog
00568       ;         00BB 3C       00569         00BE 3C       00570         00BE 3C       00571         00BE 3C       00572         00BE 3C       00573         RTC\$Hatchdog\$Count       DW         0057       RTC\$Hatchdog\$Count         0057       RTC\$Hatchdog\$Address         0057       RTC\$Hatchdog\$Address         0057       RTC\$Hatchdog\$Address         0057       RTC\$Hatchdog\$Address         00577       ; will be transferred if the         00578       ; will be transferred if the         00580       ; Function key table         00581       ;         00582       ; This table consists of a series of entries, each one having the         00583       ; following structure:         00584       ;         00585       DB       Second character of sequence = nOTE: this )         00586       ;       field will not be present if the source code )         00589       (		00	567 .		es for t	ie ieal	same cloc	in and watchdog
00BD 3C       00559       RTC\$Tick\$\$per\$Second       DB       60       :Number of real time clock         00BE 3C       00570       RTC\$Tick\$Count       DB       60       ; ticks per elapsed second         00BE 3C       00571       RTC\$Tick\$Count       DB       60       ;Residual count before next         00BF 0000       00572       RTC\$Hatchdog\$Count       DW       0       ;Watchdog timer tick count         00570       00574       ;Watchdog timer tick count       ;(0 = no watchdog timer set)       ;(0 = no watchdog timer set)         0051000       00575       RTC\$Watchdog\$Address       DW       0       ;Address to which control         00570       ;Watchdog count hits 0       ;Watchdog count hits 0       ;Watchdog count hits 0       ;Watchdog count hits 0         00577       ;Watchdog count hits 0       ;Watchdog count hits 0       ;Watchdog count hits 0         00580       ;Function key table       ;Watchdog count hits 0       ;Watchdog count hits 0         00581       ;       00582       ;This table consists of a series of entries, each one having the         00581       ;       00585       ;       DB       Second character of sequence emitted by       ;         00582       ;       DB       Third character of sequence NOTE: this	1	00	568 ·	C 7 10 M L				
00570       ; ticks per elapsed second         00BE 3C       00571         00BE 3C       00572         00BF 0000       00573         RTC\$Hatchdog\$Count       DW         00BF 0000       00573         RTC\$Hatchdog\$Count       DW         0010       00574         0011       0         0011       0         0011       0         0011       0         0011       00573         RTC\$Hatchdog\$Address       DW         0011       00576         0011       00577         0011       00578         0011       00579         00580       ;         00581       ;         00582       ;         00583       ;         00584       ;         00585       ;         00586       ;         00587       ;         00588       ;         00588       ;         00589       ;         00588       ;         00589       ;         00589       ;         00589       ;         00589       ;<	0.080.90			C&Ticks&nev#S	econd	DB	60	Number of real time clock
00BE 3C       00571 00572       RTC\$Tick\$Count       DB       60       :Residual count before next ; second will elapse         00BF 0000       00573       RTC\$Watchdog\$Count       DW       0       :Watchdog timer tick count ;(0 = no watchdog timer set)         00C1 0000       00576       ;(0 = no watchdog timer set)       ;(0 = no watchdog timer set)         00576       ;will be transferred if the ; with be transferred if the ; watchdog count hits 0         00577       ;will be transferred if the ; watchdog count hits 0         00579       ;         00580       ; Function key table         00581       ;         00582       ; This table consists of a series of entries, each one having the 00583         00584       ;         00585       ;         00586       ;         00587       ;         00588       ;         00588       ;         00588       ;         00588       ;         00589       ;         00589       ;         00589       ;         00589       ;         00589       ;         00589       ;         00589       ;         00590       ;      <	0000 30			Of ITCK Stable 1 40	econa	20		
00BF 0000       00572       RTC\$Watchdog\$Count       DW       ; second will glapse         00BF 0000       00573       RTC\$Watchdog\$Count       DW       0       ; Watchdog timer tick count         00C1 0000       00575       RTC\$Watchdog\$Address       DW       0       ; Address to which control         00570       ; will be transferred if the       ; will be transferred if the       ;         00577       ; watchdog count hits 0       00579       ;         00580       ; Function key table       ;       00580         00581       ;       This table consists of a series of entries, each one having the         00582       ;       DB       Second character of sequence emitted by         00586       ;       terminal's function key         00587       ;       DB       Third character of sequence NOTE: this         00588       ;       (       field will not be present if the source code         00589       ;       (       in function key sequences.       )         00590       ;       (       in function key sequences.       )         00590       ;       (       NOTE: Adjust the equates for;       )         00592       ;       (       NOTE: Adjust the equates for;	0000 00			C\$Tick\$Court		DB	60	
00BF 0000       00573       RTC\$Watchdog\$Count       DW       0       :Watchdog timer tick count         00C1       00074       ;(0 = no watchdog timer set)       ;(0 = no watchdog timer set)         00C1       00076       ; will be transferred if the         00577       ; will be transferred if the         00578       ; watchdog count hits 0         00579       ;         00580       ; Function key table         00581       ;         00582 ;       This table consists of a series of entries, each one having the         00583 ;       following structure:         00584 ;       00585 ;         00585 ;       DB         00586 ;       terminal's function key         00587 ;       (DB         00588 ;       (G         00588 ;       (G         00589 ;       (G         00589 ;       (G         00589 ;       (G         00589 ;       (G         00590 ;       (G         00590 ;       (G         00590 ;       (G         00591 ;       (G         00592 ;       (G         00592 ;       (G	UDE 30			ee i sen ecount		22		
00574       ;(0 = no watchdog timer set)         0000       00575       RTC\$Watchdog\$Address       DW       0         00570       ;will be transferred if the         00577       ;will be transferred if the         00578       ;watchdog count hits 0         00579       ;         00580       ;Function key table         00581       ;         00582       ;This table consists of a series of entries, each one having the         00583       ;following structure:         00584       ;         00585       ;         00586       ;terminal's function key         00587       ;         00588       ;         00588       ;         00588       ;         00589       ;         00589       ;         00589       ;         00589       ;         00590       ;         00590       ;         00591       ;         00592       ;         00592       ;         00592       ;	OORE OO			CSWatchdogsCo	unt	nw	0	
00C1 0000       00575       RTC\$Hatchdog\$Address       DW       0       iAddress to which control         00576       ; will be transferred if the         00577       ; will be transferred if the         00578       ; will be transferred if the         00579       ;         00580       ; Function key table         00581       ;         00582       ; This table consists of a series of entries, each one having the         00583       ; following structure:         00586       ;         00587       ;         00586       ;         00587       ;         00588       ;         00588       ;         00588       ;         00588       ;         00589       ;         00589       ;         00589       ;         00589       ;         00589       ;         00590       ;         00591       ;         00592       ;         00592       ;				offerer changero		2.	•	
00576       ; will be transferred if the         00577       ; watchdog count hits 0         00578       ;         00579;       ;         00580;       Function key table         00581;       ;         00582;       This table consists of a series of entries, each one having the         00583;       following structure:         00584;       00585;         00585;       DB         00586;       terminal's function key         00587;       (         00588;       (         00589;       (         00589;       (         00590;       (         00590;       (         00590;       (         00591;       (         00592;       (         00592;       (	0001 00				drest	ħы	0	Address to which control
00577       ; watchdog count hits 0         00578       ;         00579       ;         00580       ;         00581       ;         00582       ;         00583       ;         00584       ;         00585       ;         00586       ;         00587       ;         00588       ;         00588       ;         00588       ;         00589       ;         00589       ;         00589       ;         00590       ;         00590       ;         00591       ;         00592       ;         00592       ;	1 000100			~~###(C)UU9#M0		2	-	
00578         00579;         00580;       Function key table         00581;         00582;       This table consists of a series of entries, each one having the         00584;         00585;       following structure:         00586;       DB         00587;       DB         00586;       terminal's function key         00587;       (DB         00588;       (field will not be present if the source code )         00589;       (has been configured to accept only two characters )         00590;       (in function key sequences.)         00591;       Function%Key%Length								
00579 ;         00580 ;       Function key table         00581 ;         00582 ;       This table consists of a series of entries, each one having the         00583 ;       following structure:         00584 ;         00585 ;       DB         00586 ;       terminal's function key         00587 ;       DB         00588 ;       terminal's function key         00588 ;       OB         00589 ;       (         00589 ;       (         00590 ;       (         00590 ;       (         00591 ;       (         00592 ;       (								, accuracy count mits o
00580 ;       Function key table         00581 ;       This table consists of a series of entries, each one having the         00582 ;       This table consists of a series of entries, each one having the         00583 ;       following structure:         00584 ;       DB         00585 ;       DB         00586 ;       terminal's function key         00587 ;       DB         00588 ;       (         00588 ;       (         00589 ;       (         00590 ;       (         00590 ;       (         00591 ;       (         00592 ;       (         00592 ;       (								
00581 ;         00582 ;       This table consists of a series of entries, each one having the         00583 ;       following structure:         00584 ;       00585 ;         00585 ;       DB         00586 ;       terminal's function key         00587 ;       DB         00588 ;       (         00587 ;       (         00588 ;       (         00589 ;       (         00589 ;       (         00590 ;       (         00591 ;       (         00592 ;       (         00592 ;       (         00592 ;       (				Functio	n key ta	ble		
00582 ;       This table consists of a series of entries, each one having the         00583 ;       following structure:         00584 ;       00585 ;         00585 ;       DB         00586 ;       terminal's function key         00587 ;       DB         00588 ;       field will not be present if the source code         00588 ;       field will not be present if the source code         00589 ;       has been configured to accept only two characters )         00590 ;       in function key sequences.         00591 ;       NOTE: Adjust the equates for:         00592 ;       (	1			Functio	n Key te			
00583 ;       following structure:         00584 ;       00585 ;         00586 ;       DB         00587 ;       DB         00588 ;       terminal's function key         00587 ;       DB         00588 ;       (         00587 ;       (         00588 ;       (         00589 ;       (         00589 ;       (         00590 ;       (         00590 ;       (         00591 ;       (         00592 ;       (         NUTE: Adjust the equates for:       )         00592 ;       (				This 4-	hle cons	ists of	a sprige	of entries, each one having the
00584 ;       00585 ;       DB       Second character of sequence emitted by         00586 ;       terminal's function key         00587 ;       (DB       Third character of sequence NOTE: this )         00588 ;       ( field will not be present if the source code )         00589 ;       ( has been configured to accept only two characters )         00590 ;       ( in function key sequences. )         00591 ;       ( NOTE: Adjust the equates for: )         00592 ;       ( Function%Key%Length )	1							
O0585 ;     DB     Second character of sequence emitted by       00586 ;     terminal's function key       00587 ;     DB     Third character of sequence NOTE: this       00588 ;     GB     Third character of sequence NOTE: this       00589 ;     GB     Third character of sequence NOTE: this       00589 ;     GB     Third character of sequence NOTE: this       00589 ;     GB     Third character of sequence code       00590 ;     GB     Infunction key sequences.       00591 ;     GB     NOTE: Adjust the equates for:       00592 ;     GE     Function%Key%Length	1			.0110W1				
00586 ;       terminal's function key         00587 ;       DB       Third character of sequence NOTE: this )         00588 ;       (       Field will not be present if the source code )         00589 ;       (       has been configured to accept only two characters )         00590 ;       (       in function key sequences.         00590 ;       (       NOTE: Adjust the equates for:         00592 ;       (       Function\$Key\$Length		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	585		DB	Second	character	r of sequence emitted by
00587 ;       (       DB       Third character of sequence NOTE: this       )         00588 ;       (       field will not be present if the source code       )         00589 ;       (       has been configured to accept only two characters )       )         00590 ;       (       in function key sequences.       )         00591 ;       (       NOTE: ddjust the equates for:       )         00592 ;       (       Function%Key%Length       )					20			
00588;       (       field will not be present if the source code       )         00589;       (       has been configured to accept only two characters)         00590;       (       in function key sequences.       )         00591;       (       NOTE: Adjust the equates for:       )         00592;       (       Function\$Key\$Length       )				(	DB	Third c	haracter	of sequence NOTE: this )
00589 ;       (       has been configured to accept only two characters )         00590 ;       (       in function key sequences.       )         00591 ;       (       NOTE: Adjust the equates for:       )         00592 ;       (       Function*Key*Length       )	1					field w	ill not b	present if the source code )
00590 ;         (         in function key sequences.         )           00591 ;         (         NOTE: Adjust the equates for;         )           00592 ;         (         NOTE: Adjust the equates for;         )						has bee	n configu	ured to accept only two characters )
00591 ;     (     NOTE: Adjust the equates for:     )       00592 ;     (     Function\$Key\$Length     )				•				
00592 ; ( Function\$Key\$Length )	1			•				
	1							
	1			ì				

Figure 8-10. (Continued)

		00594	;				
		00595		DB .	A abase		tion to be descend into the second
		00596	?	20			ring to be forced into the console
			1		input	stream_wh	nen the corresponding function key
		00597	;		is pres	ssed. The	e last byte of this string must be
		00598	;		OOH to	terminat	te the forced input.
		00599	,				
001B	=	00600	Function\$Key\$L	ead	EQU	1 BH	;Signals function key sequence
0003		00601	Function\$Key\$L		EQU	3	Number of characters in function
		00602	. 4	engen	200		
							; key input sequence (NOTE: this
		00603					; can only be 3 or 2 characters),
		00604					
		00605	;				
		00606					;The logic associated with function
		00607					; key recognition is made easier with
		00608					
0001	-	00609	Three\$Characte		-	EQU	; the following equate
0001	-		ini eescharacte	rarette	311		Function\$Key\$Length - 2
		00610					Character\$Function will be TRUE if the
		00611					tion keys emit a three character
		00612				; sequ	ence, FALSE if they emit a two character
		00613				: sequ	ience.
		00614				,	
			. Each a		h		Abe some terrate in distance but
		00615	; Each e	ntry in t	ne tablé	must De	the same length, as defined by:
		00616	1				
0013	=	00617	CB\$Function\$Ke	y\$Entry\$S	ize	EQU	16 + 1 + Function\$Key\$Length - 1
		00618	;				* * *
		00619	;				
		00620	1	Maximum	length	of subst	itute   Lead character is not
		00621	;	string			in table entry
		00622		311119			
		00622					For the terminating 00H
			;				· · · · · · · ·
		00624		st entry	in the t	table is	marked by a 00-byte.
		00625	;				
		00626	; The ex-	ample val	ues show	vn below	are for a VT-100 terminal.
		00627					
		00628	CB\$Function\$Ke	v\$Table:			
		00629		/	100454	100 1004	↓ 5 67 <- Use to check length
0000	45504/7		, DB	101 101	123430	5/0/.12.34	V G G V V OSE LO CHECK TENGLI
		56E00630		U , P	Functi	ion Key I	,LF,0,0
	4F51467		DB	0.0	, Functi	ion Key 2	2', LF, 0, 0
		56E00632	DB	(0^,^R^	, Functi	ion Key 3	3',LF,0,0
OOFC	4F53467	56E00633	DB	101,1S1	, 'Functi	ion Key 4	¥',LF,0,0
		00634	<del>;</del>				
		00635	;		123456	5789.1	
OTOF	5R41557	02000636	DB	111.141			), 0, 0, 0, 0, 0, 0, 0
		F7700637	DB	11/10/			,0,0,0,0,0,0
		96700638	DB	1.1.1.1.1.	· /Dishi		
					, Right	AFTOW ,L	F,0,0,0,0,0
0148	5844406	56600639	DB	· · · · · · · ·	, Lett f	arrow , LF	5,0,0,0,0,0
		00640					
		00000641	DB				),0,0,0,0,0,0,0 ;Spare entries
016E	0000000	00000642	DB	0,0,0,0	,0,0,0,0	0,0,0,0,0	),0,0,0,0,0,0,0
		00000643	DB				0, 0, 0, 0, 0, 0, 0, 0
0104	0000000	00000644	DB				), 0, 0, 0, 0, 0, 0, 0
		00000645	DB				
							), 0, 0, 0, 0, 0, 0, 0
		00000646	DB				), 0, 0, 0, 0, 0, 0, 0
		00000647	DB				), 0, 0, 0, 0, 0, 0, 0
01E0	0000000	00000648	DB				), 0, 0, 0, 0, 0, 0, 0
01F3	0000000	00000649	DB				0,0,0,0,0,0,0,0
		00000650	DB				,0,0,0,0,0,0,0
****		00651					
0210	FFFF	00652	DB	OFFH, OF	FH	Termin	ator for utility that preprograms
0219	FFFF		DD	OF PHY OF			
		00653				; runc	tion key sequence
		00654	;				
		00655	;				
		00656	; Consol	e output	escape s	sequence	control table
		00657	;				
		00658		able is r	eference	d after	a Function\$Key\$Lead character
		00659		en detert	ed in ++	CONCLUT	routine. The next character
		00660					compared to the first byte
		00661	; in eac	n 3-byte	14016 et	11ry. 11	a match is found, then control
		00662	; istra	nsterred	τo the a	address f	ollowing the byte that matched.
		00663	;				
		00664	CONOUT\$Escape\$				
021B	74	00665	DB	1t1		;Read c	current time
	4804	00666	DW	CONDUT®	Time		
021E	44	00667	DB	'd'		•Read o	current date
DZIE	4404			CONOUTS	Date	inead C	.urient bate
	4104	00668	DW		-Dare	<u>.</u>	
0221		00669	DB	1u1			arrent time
0222	5004	00670	DW	CONCUT\$	Set\$Time	2	
_		-					

Figure 8-10. (Continued)

0224 45 0227 00 00677 DB 00677 DB 00677 LongTersECBEEdi 00677 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00678 LongTersECBEEdi 00797 LongTersECBEEdi 00797 LongTersECBEEdi 00797 LongTersECBEEdi 00797 L						
0225 4604 00072 DW CONCURSEIDATE 0227 00 00075 J 00075 J 00077 J 00077 J 00077 J 00077 J 00080 J Controller	0224 65	00671	DB		;Set d	current date
0227 00     00073 00077 00077 00000 00077 000000 00000 1     DB 0 100077 1 000000 1     iTerminator 00007 1       000000 000000 1     interrupt vector 000000 000000 1     interrupt vector 000000 1     interrupt vector 000000 1       00000000000000000000 0000000000000000				CONOUT	\$Set\$Date	
0227 00 00674 DB 00677 00677 00680 00600 1 1 006804 006805 1 006806 1 006806 1 006806 1 006806 1 006806 1 006806 1 006806 1 006806 1 006806 1 006806 1 006806 1 006806 0 006807 0 006807 0 006807 0 006807 0 006807 0 006807 0 006807 0 006807 0 006807 0 006807 0 006807 0 0 0 0 0 0 0 0 0 0 0 0 0	1					
00075       ;       CongSTermsCDSEEnd;         00075       ;       .         000801       ;       .         000802       ;       Interrupt vector         000803       ;       Control is transferred here by the programmable interrupt         000803       ;       Control is transferred here by the programmable interrupt         000803       ;       Control is transferred here by the programmable interrupt         000803       ;       Control is transferred here by the programmable interrupt         000803       ;       Control is transferred here by the programmable interrupt         000803       ;       Interrupt Vector is transferred here by the programmable interrupt         000812       UPP of transferred here by the programmable interrupt         000812       UPP of transferred here by the programmable interrupt         000812       UPP of transferred here by the programmable interrupt         000812       UPP of transferred here by the programmable interrupt         000812       UPP of transferred here by the programmable interrupt         000812       UPP of transferred here interrupt       13 clock         00240       COB000       00811       UPP of transferred here interrupt       13 not used         0240       COB001       UPP of transferre	0227 00		ne	0	Term	inator
00677         Long%Term%CS%End:           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :           00677         :	022/00			~	, (@fill.	
00000         :#           00000         :#           00000         :#           00000         :#           00000         :#           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           000000         :           000000         :           000000         :           000000         :           000000         :           00000000000000         :           000000000000000000000000000000000000	1		/			
00000         :#           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           00000         :           000000         :           000000         :           0000000         :           000000000000         :           000000000000000000000000000000000000	1			0:		
000001         :         Interrupt vector           000003         :         Control is transferred here by the programmable interrupt           000003         :         Control is transferred here by the programmable interrupt           000003         :         Control is transferred here by the programmable interrupt           000003         :         Interrupt controller chip requires that the collowing ORO line           00001         :         Interrupt humber           00001         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :         :           00001         :         :         :         :         :           00001         :         :         :         :         :         :           00001         :         :         :         :	1					
000001         :         Interrupt vector           000003         :         Control is transferred here by the programmable interrupt           000003         :         Control is transferred here by the programmable interrupt           000003         :         Control is transferred here by the programmable interrupt           000003         :         Interrupt controller chip requires that the collowing ORO line           00001         :         Interrupt humber           00001         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :           00001         :         :         :         :           00001         :         :         :         :         :           00001         :         :         :         :         :         :           00001         :         :         :         :	1	00800	;#			
000002       interrupt vector         000004       Control is transferred here by the programmable interrupt         000007       Control is transferred here by the programmable interrupt         000007       NUTE: The interrupt vector table start on a perserven         000007       ORG         000017       ORG         00017       ORG         00017       ORG         00017       ORG         00017       ORG         000217       ORG         00021       ORG	1	00801				
000003       ;       Control is transferred here by the programmable interrupt         000005       ;       Controlier	1			ipt vect	or	
00004         Control is transferred here by the programmable interrupt           00005         Control is transferred here by the programmable interrupt           00005         Control is transferred here by the programmable interrupt           00005         NOTE: The interrupt cort alige start on a paragraph           00006         Decomposition           00007         OB           00011         Interrupt Wector:           00012         UPP RTOSINETrupt           00013         DB           00240         COBIS           00241         CSB006           00242         CSB006           00243         COBIS           00244         CSB006           00245         DB           00246         COBIS           00247         OB           00248         COBIS           00249         OB           0244         CSB006           00232         DP           0044         CSB006           00232         DP           00451         DP           0245         COB062           00232         DP           00451         DP           00451         DP	1					
000005         ;         controller an Intel 2529A.           000007         ;         NOTE: The interrunt (controller chip requires that the interrunt vector table start on a paragraph 00000           0240         00010         ORG         ;           0240         00011         interrunt vector table start on a paragraph 00011           0240         00012         ;         interrunt vector table start on a paragraph 00011           0241         00011         ;         interrunt vector table start on a paragraph 00011           0242         00011         JHP         CharacterSinterrunt is on clock           0243         0001         0001         DHP           0244         00001         0001         DHP           0244         00001         0001         DHP           0245         000000         00010         DHP           0245         000000000000000000000000000000000000	1			ie tra	neferred here by	v the programmable interrupt
00806         ;         NOTE: The interrupt controller chip requires that the interrupt vector laste start on a perception 00806           0240         00810         interrupt vector laste start on a perception 00812           00811         Interrupt vector laste start on a perception 00812         interrupt vector laste start on a perception 00811           00812         0081         Interrupt Number 10811         interrupt vector laste start on a perception 00811           0240         C37506         00813         UMP         RTCSINETrupt number 10811           0240         C37506         00813         UMP         CharactersInterrupt if - clock 10824           0240         C380605         00818         DB         0         octastinterrupt if - character 1/0           0242         C380605         00818         DB         0         octastinterrupt if - not used           0243         C380605         00823         UMP         OnostSinterrupt if - not used         octastinterrupt if - not used           0245         C380605         00825         UMP         OnostSinterrupt if - not used         octastinterrupt if - not used           0255         00825         UMP         OnostSinterrupt if - not used         octastinterrupt if - not used           0255         00825         UMP         OnostSinterrupt if if - not used <td>1</td> <td></td> <td>, control</td> <td>ler</td> <td>an Intel 20504</td> <td>, ,</td>	1		, control	ler	an Intel 20504	, ,
00807         NOTE: The interrunt controller this requires that the interrunt vector table start on a peraph 0040           0240         00809         ;         DRC (s is achieved by the following DRC line 00812           0240         02812	1				an 111144 0207A.	
Ocided Ocide0         interrupt vector table start on a parameth Source of the pollowing DRO line           0240         Ocide0         ORG         (% AND OFFECH) + 20H           0240         Ocide0         ORG         (% AND OFFECH) + 20H           0240         Ocide0         Interrupt number Interrupt number         Interrupt number           0243         OC         Ocide0         D         Iskin a bote           0244         OC         Ocide0         D         Iskin a bote           0244         OC         Ocide0         Ocide0         D           0244         OCide0         Ocide0         D         D           0245         OCide0         Ocide0         D         D           0246         OCide0         Ocide0         D         D         D           0247         OCide0         Ocide0         D         D         D         D           0247         OCide0         Ocide0         D         D         D         D         D           0248         Ocide0         Ocide0         D         D         D         D         D         D         D         D         D         D         D         D         D         D         D	1					
00009         :         boundary. This is achieved by the following DRO line           0240         00001         InterruptWector:         :Interrupt number           0240         00001         INterruptWector:         :Interrupt number           0243         00000         00014         DB         0         - cleck           0244         00000         00014         DB         0         - cleck           0244         000000         00016         DB         0         - character I/0           0245         000000         00017         JMP         OnostEinterrupt         :3         not used           0245         000000         00022         DB         0         not used         0           0245         000000         00022         DB         0         not used         0           0250         00022         JMP         GhostEinterrupt         :5         not used         0           0255         00000         00225         JMP         GhostEinterrupt         :5         not used           0256         0026         00225         JMP         GhostEinterrupt         :5         not used           0257         000000         :	1		NOTE: 1			
0240         0081         0RG         (4 AND 0FEC0H) + 20H           0240         00812         Interrupt with the number         0 clock           0243         0000         0RG         HP         RTGBInterrupt number         10 clock           0243         000         0RG         UMP         Character#Interrupt 11 character 1/0           0244         00000         00812         UMP         GhostBInterrupt 12 not used           0248         00001         0081         DB         0           0248         000000         00812         UMP         GhostBInterrupt 13 not used           0248         000000         00822         UMP         GhostBInterrupt 13 not used           0250         008000         00822         UMP         GhostBInterrupt 13 not used           0250         00802         0082         UMP         GhostBInterrupt 16 not used           0250         00802         00802         UMP         GhostBInterrupt 17 not used           0250         00802         UMP         GhostBInterrupt 17 not used           0250         00802         UMP         GhostBInterrupt 17 not used           0250         00802         UMP         GhostBInterrupt 17 not used				interr	upt vector table	start on a paragraph
0240         0081         0RG         (4 AND 0FEC0H) + 20H           0240         00812         Interrupt with the number         0 clock           0243         0000         0RG         HP         RTGBInterrupt number         10 clock           0243         000         0RG         UMP         Character#Interrupt 11 character 1/0           0244         00000         00812         UMP         GhostBInterrupt 12 not used           0248         00001         0081         DB         0           0248         000000         00812         UMP         GhostBInterrupt 13 not used           0248         000000         00822         UMP         GhostBInterrupt 13 not used           0250         008000         00822         UMP         GhostBInterrupt 13 not used           0250         00802         0082         UMP         GhostBInterrupt 16 not used           0250         00802         00802         UMP         GhostBInterrupt 17 not used           0250         00802         UMP         GhostBInterrupt 17 not used           0250         00802         UMP         GhostBInterrupt 17 not used           0250         00802         UMP         GhostBInterrupt 17 not used	1			bounda	ry. This is achi	ieved by the following ORG line
00811 00812         InterruptWvector: interrupt number 00814         interrupt number 10 - clock 1084           0240         02813         JMP         RTCSInterrupt 10 - clack 00814         10 - clock 11 - character 1/0           0240         02006         00814         DE 0         0         11 - character 1/0           0248         02006         00815         DE 0         0         0         11 - character 1/0           0248         02006         00815         DE 0         0         0         14 not used           0247         02006         00822         DE 0         0         0         14 not used           0247         020060         00822         DE 0         0         0         15 not used           0240         020062         DE 0         0         0         0         0         0           0255         00062         DE 0         0         0         0         0         0           0255         00062         JMP         DhostSInterrupt         17 not used         0           0258         00062         JMP         DhostSInterrupt         17 not used         0           0256         0082         JMP         DhostSInterrupt         1	0240			(\$ AND	OFFEOH) + 20H	
Orde         Construct         interrupt         interrupt         interrupt           Orde         Orde         Orde         Orde         Imp         Onaracter#Interrupt         I	1		Interrupt\$Vecto	r:		
0243 00         00813         JHP         RTCBInterrupt         10 clock           0243 00         00814         DB         0         ISkip a byte           0244 02E806         00815         JHP         CharacterBInterrupt         11 character I/0           0244 00         00816         DB         0         0           0248 00         00819         DF         0         0           0240 00         00819         DF         0         0           0247 00         00810         DF         0         0         0           0240 02000         00821         JHP         OhostBInterrupt         14 not used         0           0250 020000         00822         JHP         OhostBInterrupt         15 not used         0           0258 020000         00823         JHP         OhostBInterrupt         17 not used         0           0258 020000         00823         JHP         OhostBInterrupt         17 not used         0           0258 020000         00823         JHP         OhostBInterrupt         17 not used         0           0250 00000         1         DEvice Port numbers and other equates         0         0         0           <	1		• • • • • • • • •		;Interrupt nu	aber
0244 3 00         00814         DB         0         FSkip a byte           0244 32806         00815         JMP         CharacterSinterrupt         11 character 1/0           0248 32806         00815         JMP         ChostSinterrupt         12 not used           0248 3200         00815         JMP         ChostSinterrupt         13 not used           0247 00         00805         JMP         ChostSinterrupt         14 not used           0247 00         00821         JMP         ChostSinterrupt         15 not used           0250 00         00823         JMP         ChostSinterrupt         15 not used           0257 00         00823         JMP         ChostSinterrupt         17 not used           0258 02808         00823         JMP         ChostSinterrupt         17 not used           0250 00000         00823         JMP         ChostSinterrupt         17 not used           0250 000000         00823         JMP         ChostSinterrupt         17 not used           0250 000000000         108         009000         10900000000000000000000000000000000000	0240 037808		. IMP	RTCSIn		
0244 C3E806         CO0015         UMP         Character#Interrupt         i1 character 1/0           0248 C3B00E         CO0015         UMP         Chost#Interrupt         i2 not used           0248 C3B00E         CO0015         UMP         Chost#Interrupt         i3 not used           0248 C3B00E         CO0220         UMP         Chost#Interrupt         i4 not used           0247 C0         CO0220         UMP         Chost#Interrupt         i5 not used           0255 C3B00E         CO0220         UMP         Chost#Interrupt         i5 not used           0258 C3B00E         CO0220         UMP         Chost#Interrupt         i7 not used           0258 C3B00E         CO0220         UMP         Chost#Interrupt         i7 not used           0258 C3B00E         CO0220         UMP         Chost#Interrupt         i7 not used           0258 C3B00E         CO0220         UMP         Chost#Interrupt         i7 not used           0258 C3B00E         CO0220         UMP         Chost#Interrupt         i7 not used           0258 C3B00E         CO0220         UMP         Chost#Interrupt         i7 not used           0258 C3B00E         CO0220         UMP         Chost#Interrupt         i7 not used<						
0249 C3D80E       00817       JMP       OhostBInterrupt       12 not used         0248 C3D80E       00818       DB       0       0       0         0248 C3D80E       00819       JMP       OhostBInterrupt       13 not used         0240 C3D80E       00820       DB       0       0       0         0240 C3D80E       00822       DB       0       0       0         0235 C3D80E       00822       DB       0       0       0         0235 C3D80E       00822       DB       0       0       0         0235 C3D80E       00823       JMP       OhostBInterrupt       15 not used         0235 C3D80E       00828       J       D       0       0         0255 C3D80E       00828       J       D       0       0       0         0255 C3D80E       00828       J       D       D       J       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D       D				-	tartInterrunt	
0248 C3D00E         00017         JMP         Ohost@Interrupt         j2 not used           0248 C0         00018         DB         0           0247 C0         00020         DB         0           0245 C0         00022         DB         0           0233 C0         00022         DB         0           0235 C3D00E         00023         JMP         Chost@Interrupt         j5 not used           0235 C3D00E         00023         JMP         Chost@Interrupt         j6 not used           0235 C3D00E         00023         JMP         Chost@Interrupt         j7 not used           0235 C3D00E         00023         j         Device port numbers and other equates         000001           000000         ;         Device port numbers and other equates         000001         000001           000000         ;         Device port numbers and other equates         jDevice 0         00001           00000         ;         Device port numbers and other equates         jDevice 0         00001           00000         ;         Device port numbers and other equates         jDevice 0         00001           00000         :         Device port numbers and other equates         jDevice 0         0001					rei trincei i ubf	ya character 170
0248 00         00618         DB         0           0240 C3080E         00819         JMP         Dhost\$Interrupt         ;3 not used           0240 C3080E         00822         DB         0						and work would
024C C3D00E         00819         UMP         OhostSInterrupt         ;3 not used           0250 C3D00E         00820         DB         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0					interrupt	j∠ not useo
0246 00       00820       DB       0         0250 C3080E       00821       JMP       0host\$Interrupt       ;5 not used         0253 00       00822       DB       0       0       0         0250 C3080E       00824       DB       0       0       0         0250 C3080E       00825       JMP       0host\$Interrupt       ;5 not used         0250 C3080E       00825       JMP       0host\$Interrupt       ;7 not used         0250 C3080E       00825       JMP       0host\$Interrupt       ;7 not used         0250 C3080E       00825       JMP       0host\$Interrupt       ;7 not used         0250 C3080E       00825       JMP       0host\$Interrupt       ;7 not used         00820 -       00902 ;       Device port numbers and other equates       00903         0080 -       00907       DobBase\$Port EQU       C108Base\$Port ;       jDevice 0         0080 -       00906       DobBase\$Port EQU       D08Base\$Port + 1       jDevice 0         0081 -       00906       DobStatusPort EQU       D18Base\$Port + 2       jDevice 1         0081 -       00911 j       JDBata\$Port EQU       D18Base\$Port + 1       jDevice 2         0085 -       0				•		
0247 00       00820       DB       0         0250 C3080E       00822       DB       0         0253 00       00822       DB       0         0256 C3080E       00822       DB       0         0257 00       00824       DB       0         0258 C3080E       00825       DMP       Ohost\$Interrupt       ;5 not used         0258 C3080E       00825       DMP       Ohost\$Interrupt       ;7 not used         0258 C3080E       00825       DMP       Ohost\$Interrupt       ;7 not used         0250 C3080E       00825       DMP       Ohost\$Interrupt       ;7 not used         0250 C3080E       00825       DMP       Ohost\$Interrupt       ;7 not used         00900 r       r#       00900 r       r#       00900 r         00900 r       D0900000000000000000000000000000000000	024C C3D80E	00819	JMP	Ghost\$	Interrupt	;3 not used
0250 C3D90E         00821         JMP         Ghost\$Interrupt         if not used           0253 00         00822         JMP         Ghost\$Interrupt         i5 not used           0257 00         00824         JMP         Ghost\$Interrupt         i6 not used           0258 03060         00825         JMP         Ghost\$Interrupt         i7 not used           0258 00         00826         DB         0         not used           0258 00         00827         JMP         Ghost\$Interrupt         i7 not used           0258 00         00828         ,         Device port numbers and other equates         not used           0080 =         00905         DosBase\$Port         EQU         CloBBase\$Port         :pevice 0           0080 =         00905         DosBase\$Port         EQU         DosBase\$Port + 1         :pevice 0           0081 =         00905         DosBase\$Port EQU         DosBase\$Port + 2         :pevice 1         :pevice 1           0082 =         00905         DosBase\$Port EQU         DosBase\$Port + 3         :pevice 1         :pevice 1           0084 =         00911         DisStatus\$Port EQU         DosBase\$Port + 2         :pevice 1         :pevice 1           00864 =         00	024F 00	00820	DB	0		
0 0253 00 0254 C30B00E         00823 00824         DB D B 0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0 <th0< th=""></th0<>	0250 C3080E	00821	JMP	Ghosts	Interrupt	;4 not used
0225         0257         00         0024         DB         0				0	····	
0250         00600         00624         DB         0           0250         00800         00625         DB         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0 <td></td> <td></td> <td></td> <td>Ghosts</td> <td>Interrupt</td> <td>:5 not used</td>				Ghosts	Interrupt	:5 not used
0258 C3080E       00825       JMP       Ghost#Interrupt       16 not used         0258 C3080E       00826       JMP       Ghost#Interrupt       17 not used         0250 C3080E       00827       JMP       Ghost#Interrupt       17 not used         00900 =       00900 ; #       00900 ;       00900 ;       00900 ;         00900 =       00900 ; D09Base#Port       EQU       80H       ;Base port number         00900 =       00900 D09Base#Port       EQU       C10#Base#Port       ;Device 0         0080 =       00907 D09Base#Port       EQU       D09Base#Port 1       ;Device 0         0081 =       00900 D09Command#Port EQU       D09Base#Port 1       2       Device 1         0082 =       00912 D09Command#Port EQU       D19Base#Port 1       1       Device 1         0084 =       00913 D1#Base#Port EQU       D19Base#Port 1       2       Device 1         0085 =       00919 D2#Base#Port EQU       D19Base#Port 1       3       Device 2         0086 =       00919 D2#Base#Port EQU       D19Base#Port 1       3       Device 2         0088 =       00919 D2#Base#Port EQU       D2#Base#Port 1       3       Device 2         0088 =       00921 D2#Base#Port EQU       D2#Base#Port 1		00924				
0256 00       00827       UHP       Ghost\$Interrupt       ;7 not used         00828       ;       00900       ;#         00900       ;#       00901       ;         00901       ;       Device port numbers and other equates         00902       ;       Device port numbers and other equates         00903       ;       Clo\$Base\$Port       EQU         0080 =       00905       Clo\$Base\$Port       EQU         0080 =       00907       DO\$Base\$Port       EQU         0081 =       00907       DO\$Base\$Port       EQU         0081 =       00907       DO\$Base\$Port       EQU         0082 =       00907       DO\$Base\$Port       EQU       DO\$Base\$Port + 2         0081 =       00912       DO\$Command\$Port       EQU       DO\$Base\$Port + 4       ;Device 1         0084 =       00912       DI\$Base\$Port       EQU       DI\$Base\$Port + 2       DO\$Command\$Port       EQU       DI\$Base\$Port + 1         0085 =       00912       DI\$Base\$Port       EQU       DI\$Base\$Port + 2       Device 1       DEvice 1         0086 =       00912       DI\$Base\$Port       EQU       DI\$Base\$Port + 2       DEvice 2       DEvice 2       DEvice 2       DE		00925		•	Interrupt	to not used
025C C3DB0E         00827         JMP         Ghost\$Interrupt         ;7 not used           00900         ;00900         ;00900         ;00900         ;00900           00900         ;00900         ;00900         ;00900           00900         ;00900         ;00900         ;00900           00800 =         00904         C100Base\$Port         EBU         SOH         ;Base port number           0080 =         00905         D05Base\$Port         EBU         D05Base\$Port         ;Device 0           0081 =         00906         D050tatasPort         EBU         D05Base\$Port + 1         ;Device 1           0082 =         00907         D05Command\$Port         EBU         D05Base\$Port + 2         ;Device 1           0084 =         00912         D15Base\$Port         EBU         D15Base\$Port + 4         ;Device 1           0084 =         00915         D15Mase\$Port         EBU         D15Base\$Port + 1         ;Device 2           0085 =         00917         D15Command\$Port         EBU         D15Base\$Port + 1         ;Device 2           0087 =         00921         D25Base\$Port         EOU         D25Base\$Port + 2         ;Device 2           0088 =         00921         D25Base\$Port <td< td=""><td></td><td></td><td></td><td></td><td>ANCEL I GPU</td><td>yw HWC WAW</td></td<>					ANCEL I GPU	yw HWC WAW
00826         ;           00900         ;           00901         00902           00902         ;           00903         ;           00904         Cl08Base\$Port           00905         ;           00906         D09Base\$Port           00907         009Base\$Port           0080         00905           0080         00906           00905         D08Base\$Port           0081         00906           0082         00907           0082         00906           0081         00906           0082         00907           0083         00907           0084         00912           0084         00913           0084         00914           0085         00915           0086         00914           018BasePort         EOU           0085         00915           0086         00916           0087         018BasePort           0088         00918           0088         00921           00895         02920           028basePort           0088         00				-	T = 4 =	17 mm not used
00900         ;#           00901         00902         ;           0080 =         00904         CIO\$Base\$Port         EQU         80H         ;Base port number           0080 =         00905         D0\$Base\$Port         EQU         CIO\$Base\$Port         ;Device 0           0080 =         00905         D0\$Base\$Port         EQU         D0\$Base\$Port         ;Device 0           0081 =         00906         D0\$Base\$Port         EQU         D0\$Base\$Port         1           0082 =         00907         D0\$Bota\$Port         EQU         D0\$Base\$Port         2           0083 =         00910         D0\$Command\$Port         EQU         D0\$Base\$Port         4           0084 =         00913         D1\$Base\$Port         EQU         D1\$Base\$Port         1           0084 =         00914         D1\$Base\$Port         EQU         D1\$Base\$Port         1           0085 =         00915         D1\$Command\$Port         EQU         D1\$Base\$Port         1           0088 =         00910         D2\$Base\$Port         EQU         D2\$Base\$Port         1           0088 =         00920         D2\$EData\$Port         EQU         D2\$Base\$Port         1           0088 =	0250 C3080E			unost\$	interrupt	;/ not useu
00901 00902 ; Device port numbers and other equates           0080 =         00903 00903 ;           0080 =         00905 00905 D08Base\$Port EQU C10\$Base\$Port ;Device 0           0080 =         00906 D08Base\$Port EQU D08Base\$Port ;Device 0           0081 =         00908 D08Status\$Port EQU D08Base\$Port +1           0082 =         00909 D08Command\$Port EQU D08Base\$Port +2           0083 =         00910 D08Command\$Port EQU D08Base\$Port +2           0084 =         00913 D1\$Base\$Port EQU D1\$Base\$Port +4 ;Device 1           0084 =         00914 D1\$Data\$Port EQU D1\$Base\$Port +3           0085 =         00915 D1\$Status\$Port EQU D1\$Base\$Port +2           0086 =         00916 D1\$Status\$Port EQU D1\$Base\$Port +3           0086 =         00919 D2\$Data\$Port EQU D1\$Base\$Port +3           0088 =         00919 D2\$Data\$Port EQU D2\$Base\$Port +2           0088 =         00910 D2\$Data\$Port EQU D2\$Base\$Port +1           0088 =         00920 D2\$Data\$Port EQU D2\$Base\$Port +2           0088 =         00921 D2\$Status\$Port EQU D2\$Base\$Port +2           0088 =         00922 D2\$Data\$Port EQU D2\$Base\$Port +3           00872 00927 rull p2\$Status\$Port EQ						
00902         j         Device port numbers and other equates           0080 =         00903         i           0080 =         00905         CIOBBase\$Port         EQU         80H         ;Base port number           0080 =         00905         D08Base\$Port         EQU         D00Base\$Port         ;Device 0           0081 =         00907         D08Base\$Port         EQU         D09Base\$Port + 1         000Base\$Port + 2           0082 =         00909         D08Hode\$Port         EQU         D08Base\$Port + 2         00912           0083 =         00911         J08Ecommand\$Port         EQU         D18Base\$Port + 4         ;Device 1           0084 =         00913         D18Base\$Port         EQU         D18Base\$Port + 2         00813           0085 =         00916         D18Hode\$Port         EQU         D18Base\$Port + 2         00818           0088 =         00919         D28Base\$Port EQU         D18Base\$Port + 3         00818         0082           0088 =         00920         D28Base\$Port EQU         D28Base\$Port + 1         0082         0082         00921         028Base\$Port EQU         D28Base\$Port + 3         0088         00921         00824         0082         00921         00824         0082 <t< td=""><td></td><td></td><td>2 带</td><td></td><td></td><td></td></t<>			2 带			
00900 =         00904 CIOBBase\$Port EQU         80H         ;Base port number           0080 =         00905 DO\$Base\$Port EQU         DO\$Base\$Port EQU         DO\$Base\$Port + 1           0081 =         00906 DO\$Bota\$Port EQU         DO\$Base\$Port + 2           0082 =         00907 DO\$Bota\$Port EQU         DO\$Base\$Port + 2           0083 =         00911 j         DO\$Command\$Port EQU         DO\$Base\$Port + 4           0084 =         00913 DI\$Base\$Port EQU         DI\$Base\$Port + 1           0084 =         00914 DI\$Data\$Port EQU         DI\$Base\$Port + 3           0085 =         00915 DI\$Statu\$Port EQU         DI\$Base\$Port + 3           0086 =         00917 DI\$Command\$Port EQU         DI\$Base\$Port + 3           0086 =         00918 D2\$Base\$Port EQU         DI\$Base\$Port + 3           0087 =         00917 DI\$Command\$Port EQU         DI\$Base\$Port + 3           0088 =         00919 D2\$Base\$Port EQU         DI\$Base\$Port + 4         ;Device 2           0088 =         00920 D2\$Bata\$Port EQU         D2\$Base\$Port + 1         D2\$Base\$Port + 2           0088 =         00921 D2\$Bata\$Port EQU         D2\$Base\$Port + 2         D2\$Base\$Port + 3           0088 =         00922 D2\$Bata\$Port EQU         D2\$Base\$Port + 2         D2\$Base\$Port + 2           0088 =         00922 D2\$Bata\$Port EQU		00901				
00900 =         00904 CIOBBase\$Port EQU         80H         ;Base port number           0080 =         00905 DO\$Base\$Port EQU         DO\$Base\$Port EQU         DO\$Base\$Port + 1           0081 =         00906 DO\$Bota\$Port EQU         DO\$Base\$Port + 2           0082 =         00907 DO\$Bota\$Port EQU         DO\$Base\$Port + 2           0083 =         00911 j         DO\$Command\$Port EQU         DO\$Base\$Port + 4           0084 =         00913 DI\$Base\$Port EQU         DI\$Base\$Port + 1           0084 =         00914 DI\$Data\$Port EQU         DI\$Base\$Port + 3           0085 =         00915 DI\$Statu\$Port EQU         DI\$Base\$Port + 3           0086 =         00917 DI\$Command\$Port EQU         DI\$Base\$Port + 3           0086 =         00918 D2\$Base\$Port EQU         DI\$Base\$Port + 3           0087 =         00917 DI\$Command\$Port EQU         DI\$Base\$Port + 3           0088 =         00919 D2\$Base\$Port EQU         DI\$Base\$Port + 4         ;Device 2           0088 =         00920 D2\$Bata\$Port EQU         D2\$Base\$Port + 1         D2\$Base\$Port + 2           0088 =         00921 D2\$Bata\$Port EQU         D2\$Base\$Port + 2         D2\$Base\$Port + 3           0088 =         00922 D2\$Bata\$Port EQU         D2\$Base\$Port + 2         D2\$Base\$Port + 2           0088 =         00922 D2\$Bata\$Port EQU		00902	; Device	port nu	mbers and other	equates
0080 =         00905 00905         C10\$Base\$Port         EQU EQU         80H         ;Base port number           0080 =         00905         D0\$Base\$Port         EQU         C10\$Base\$Port         ;Device 0           0080 =         00907         D0\$Base\$Port         EQU         D0\$Base\$Port         ;Device 0           0081 =         00907         D0\$Base\$Port         EQU         D0\$Base\$Port + 1         ;Device 0           0082 =         00907         D0\$Mode\$Port         EQU         D0\$Base\$Port + 2         ;Device 1           0081 =         00910         D0\$Command\$Port         EQU         D0\$Base\$Port + 3         ;Device 1           0084 =         00913         D1\$Base\$Port         EQU         D1\$Base\$Port + 4         ;Device 1           0085 =         00915         D1\$Mode\$Port         EQU         D1\$Base\$Port + 2         ;Device 2           0087 =         00917         D1\$Command\$Port         EQU         D1\$Base\$Port + 8         ;Device 2           0088 =         00921         D2\$Base\$Port         EQU         D2\$Base\$Port + 1         ;Device 2           0088 =         00922         D2\$Mode\$Port         EQU         D2\$Base\$Port + 2         ;Device 2           0088 =         00922         D2\$Mode\$Port<	1		7			
00000         00000         00000         00000         00000         00000         000000         000000         000000000000000000000000000000000000	0080 =		CIO\$Base\$Port	EQU	80H	;Base port number
OOB0 =         OO906 b0%Bass%Port         EQU b0%Bass%Port         ;Device 0           OO80 =         OO907 b0%Bats%Port EQU b0%Bass%Port + 1         D0%Bass%Port + 2           O082 =         OO907 b0%Dods%Port EQU b0%Bass%Port + 2         D0%Bass%Port + 2           O084 =         O0910 b0%Command%Port EQU b0%Bass%Port + 3         D0%Dats%Port EQU b1%Bass%Port + 4         ;Device 1           O084 =         O0913 b1%Bass%Port EQU b1%Bass%Port + 4         ;Device 1         D0%Bass%Port + 3           O085 =         O0914 b1%Dats%Port EQU b1%Bass%Port + 1         D0%Bass%Port + 2           O086 =         O0917 b1%Command%Port EQU b1%Bass%Port + 3         D0%918           O088 =         O0917 b2%Bass%Port EQU b1%Bass%Port + 3         ;Device 2           O088 =         O0919 b2%Bass%Port EQU b1%Bass%Port + 3         ;Device 2           O088 =         O0920 D2%Bass%Port EQU D2%Bass%Port + 1         D0%Bass%Port + 2           O088 =         O0921 D2%Stats%Port EQU D2%Bass%Port + 2         D0%Bass%Port + 2           O088 =         O0922 D2%Stats%Port EQU D2%Bass%Port + 2         D0%Bass%Port + 2           O088 =         O0922 D2%Stats%Port EQU D2%Bass%Port + 2         D0%D3           O088 =         O0922 D2%Stats%Port EQU D2%Bass%Port + 2         D0%D3           O0924 00%20 D2%Command%Port EQU D2%Bass%Port + 3         D0%D3 <t< td=""><td>1</td><td></td><td></td><td>-</td><td></td><td></td></t<>	1			-		
0080 =         00907         D08DataSPort         EQU         D08DataSPort           0081 =         00908         D08DataSPort         EQU         D08BaseSPort + 1           0082 =         00909         D08DataSPort EQU         D08BaseSPort + 2           0083 =         00910         D08CommandSPort EQU         D08BaseSPort + 3           00911 ;         00912         008BaseSPort EQU         D18BaseSPort + 4 ;Device 1           0084 =         00914         D18DataSPort EQU         D18BaseSPort + 1           0085 =         00915         D18StatusSPort EQU         D18BaseSPort + 2           0087 =         00916         D18ModeSPort EQU         D18BaseSPort + 3           0088 =         00919         D28BaseSPort EQU         D18BaseSPort + 3           0088 =         00920         D28DataSPort EQU         D28BaseSPort + 3           0088 =         00921         D28baseSPort EQU         D28BaseSPort + 1           0088 =         00922         D28ModeSPort EQU         D28BaseSPort + 2           0088 =         00922         D28ModeSPort EQU         D28BaseSPort + 3           0088 =         00922         D28ModeSPort EQU         D28BaseSPort + 3           0088 =         00922         D28ModeSPort EQU         D18DaSBaseSPort + 3<	0080 =		BO\$Base\$Port	EQU	CIO\$Base\$Port	;Device O
0081 =         00909         D0\$Node#Port D0\$Mode#Port 0091         EQU EQU D0\$Base#Port + 1         D0\$Base#Port + 1           0083 =         00910         D0\$Command#Port 00911         EQU D0\$Base#Port + 3         D0\$Base#Port + 3           0084 =         00913         D1\$Base#Port EQU D1\$Base#Port + 4         ;Device 1           0085 =         00915         D1\$Status#Port EQU D1\$Base#Port + 2         D1\$Base#Port + 2           0086 =         00916         D1\$Mode#Port EQU D1\$Base#Port + 2         D1\$Base#Port + 2           0086 =         00917         D1\$Command#Port EQU D1\$Base#Port + 2         D1\$Ease#Port + 3           0087 =         00919         D2\$Base#Port EQU D2\$Base#Port EQU D2\$Base#Port + 1         D2\$Base#Port + 1           0088 =         00920         D2\$Btats#Port EQU D2\$Base#Port + 2         D2\$Base#Port + 2           0088 =         00921         D2\$Btats#Port EQU D2\$Base#Port + 3         D2\$Command#Port EQU D2\$Base#Port + 3           0088 =         00922         D2Mode\$Value\$1 EQU D2\$Base#Port + 3         D0022           0088 =         00922         D2Mode\$Value\$1 EQU D1\$Base#Port + 3         D2\$Command\$Value\$1 EQU D2\$Base#Port + 3           0032 =         00925         D\$Mode\$Value\$1 EQU D3\$Command\$Value\$2 EQU D3\$Command\$Value\$2 EQU D3\$Command\$Value\$2 EQU D3\$Command\$Value\$2 EQU D3\$Command\$Value\$2 EQU D3\$Command\$Value\$2 EQU D3\$Command\$Value\$2 EQU D3\$Command\$Value\$						,
0082 =         00909         D0%Edemport         EQU         D0%BasePort + 2           0083 =         00910         D0%Command%Port EQU         D0%Base%Port + 2           0084 =         00913         D1%Base%Port EQU         D0%Base%Port + 4         ;Device 1           0084 =         00914         D1%Base%Port EQU         D1%Base%Port + 1         ;Device 1           0086 =         00915         D1%Status%Port EQU         D1%Base%Port + 2         ;Device 2           0087 =         00917         D1%Command%Port EQU         D1%Base%Port + 3         ;Device 2           0088 =         00919         D2%Base%Port EQU         D1%Base%Port + 3         ;Device 2           0088 =         00920         D2%Base%Port EQU         D2%Base%Port + 1         2           0088 =         00921         D2%Status%Port EQU         D2%Base%Port + 2         2           0088 =         00922         D2%Mode%Port EQU         D2%Base%Port + 3         2           0088 =         00923         D2%Command%Port EQU         D2%Base%Port + 3         2           0088 =         00923         D2%Command%Port EQU         D2%Base%Port + 3         2           0088 =         00922         D2%Mode%Value%1         EQU         D1%0%11%108         2 <tr< td=""><td></td><td></td><td></td><td></td><td></td><td>+ 1</td></tr<>						+ 1
0083 =         00910 00911 00912         D0%Command%Port EQU 00912         D0%Base%Port + 3 00912           0084 =         00913         D1%Base%Port EQU D1%Base%Port + 4 0085 =         D1%Data%Port EQU D1%Base%Port + 1 0086 =         D1%Data%Port EQU D1%Base%Port + 2 0087 =         D1%Data%Port EQU D1%Base%Port + 2 00917         D1%Data%Port EQU D1%Base%Port + 3 00918           0088 =         00919         D2%Base%Port EQU D2%Base%Port + 6 00920         D2%Base%Port + 6 12%Base%Port + 1 22%Base%Port + 1 02%Base%Port + 1 22%Base%Port + 1 22%Base%Port + 1 0088 =         00920         D2%Base%Port + 6 12%Base%Port + 1 22%Base%Port + 2 0088 =         00922         D2%Mode%Port EQU D2%Base%Port + 2 00928         D2%Base%Port + 3 12%Base%Port + 3 00924           004E =         00925         D5Mode%Value%1         EQU 00%11%1008         D1%Sometrix + 8 18 bits, Async. 16x rate           003C =         00928         D%Mode%Value%2         EQU 00%11%1008         00%10%118108           00927 =         D%Mode%Value%2         EQU 00%11%11008         00%10%11008           00928 =         D%Command%Value         EQU 00%101118         00%10%118           00930 =         Fix/Rx on internal clock 19600 baud         10%074           00932 =         D%Error         EQU 0011%10008         0011%1008           00933 =         D%Error%Exet         EQU 00011%10008         0011%10008           0037 =						
00911 0084 =         00913 00914         D1\$Base\$Port         EQU D1\$Base\$Port + 4         ;Device 1           0084 =         00914         D1\$Data\$Port EQU D1\$Base\$Port + 1         D1\$Base\$Port + 1           0085 =         00915         D1\$Statu\$Port EQU D1\$Base\$Port + 2         D1\$Base\$Port + 2           0087 =         00917         D1\$Command\$Port EQU D2\$Base\$Port + 2         D1\$Base\$Port + 3           0088 =         00919         D2\$Base\$Port EQU D2\$Base\$Port + 1         D2\$Base\$Port + 1           0088 =         00920         D2\$Btatu\$Port EQU D2\$Base\$Port + 2         D2\$Base\$Port + 1           0088 =         00921         D2\$Statu\$Port EQU D2\$Base\$Port + 2         D2\$Base\$Port + 3           0088 =         00922         D2#Mode\$Port EQU D2\$Base\$Port + 3         D2\$Base\$Port + 4           0088 =         00922         D2#Mode\$Port EQU D2\$Base\$Port + 3         D2\$Base\$Port + 3           0088 =         00924         01\$00\$11\$10B         1\$\$ top bit, no parity 1\$\$ top bit, no parity 1\$\$ bits, Async. 16x rate           0032 =         09330         1\$\$ top bit.00B         1\$\$ point           0032 =         0\$\$ top 1\$\$ point         PO         1\$\$ point           0038 =         00935         D\$\$Command\$Value EQU         0\$\$ point         1\$\$ point           00392 (0033)         1\$\$ point <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
00912         00913         D1\$Base\$Port         EQU         CIO\$Base\$Port         4         ;Device 1           0084 =         00914         D1\$Data\$Port         EQU         D1\$Base\$Port         + 1           0085 =         00915         D1\$Status\$Port         EQU         D1\$Base\$Port + 1           0086 =         00916         D1\$Command\$Port         EQU         D1\$Base\$Port + 2           0087 =         00917         D1\$Command\$Port         EQU         D1\$Base\$Port + 8         ;Device 2           0088 =         00919         D2\$Base\$Port         EQU         D2\$Base\$Port + 1         0086           0088 =         00920         D2\$Bata\$Port         EQU         D2\$Base\$Port + 2         0088           0088 =         00921         D2\$Bata\$Port         EQU         D2\$Base\$Port + 2         0088           0088 =         00922         D2\$Hode\$Port         EQU         D2\$Base\$Port + 2         0088           0088 =         00923         D2\$Command\$Port         EQU         D2\$Base\$Port + 3         00924           0042 =         00925         D\$Mode\$Value\$1         EQU         01\$011\$108         0925           0032 =         00928         D\$Mode\$Value\$2         EQU         00\$1101118         1\$600 ba	0083 =			EGO	DU#Base#Port -	F 3
0084 =         00913         DisBasesPort         EOU         CIOSBasesPort         4         ;Device 1           0084 =         00914         DisBasesPort         EOU         DisBasesPort         1           0085 =         00915         DisStatusSPort         EQU         DisBasesPort + 1           0086 =         00916         DisModesPort         EQU         DisBasesPort + 2           0087 =         00917         DisCommandsPort         EQU         DisBasesPort + 3           0088 =         00920         D2sDatasPort         EQU         D2sBasesPort + 1           0088 =         00920         D2sDatasPort         EQU         D2sBasesPort + 2           0088 =         00920         D2sDatasPort         EQU         D2sBasesPort + 4           0088 =         00920         D2sDatasPort         EQU         D2sBasesPort + 1           0088 =         00921         D2sModesPort         EQU         D2sBasesPort + 2           0088 =         00923         D2sCommandsPort         EQU         D2sBasesPort + 3           00924         ::Distore         ::Stop bit, no parity         :Stop bit, no parity           0032 =         :Distore         :Stop bit, no internal clock         :P600 baud           00920 :	1		;			
0084 =         00914         DisData\$Port         EQU         DisBass\$Port           0085 =         00915         DisStatus\$Port         EQU         DisBass\$Port + 1           0086 =         00916         DisModsPort         EQU         DisBass\$Port + 2           0087 =         00917         DisCommand\$Port         EQU         DisBass\$Port + 3           0088 =         00919         D2\$Bass\$Port         EQU         D2\$Bass\$Port + 4           0088 =         00920         D2\$Data\$Port         EQU         D2\$Bass\$Port + 3           0088 =         00921         D2\$Bass\$Port         EQU         D2\$Bass\$Port + 2           0088 =         00922         D2\$Mods\$Port         EQU         D2\$Bass\$Port + 2           0088 =         00922         D2\$Mods\$Value\$Port         EQU         D2\$Bass\$Port + 3           0088 =         00922         D2\$Mods\$Value\$1         EQU         D2\$Bass\$Port + 3           0088 =         00925         D\$Mods\$Value\$1         EQU         D2\$Bass\$Port + 3           00924         0042         00925         D\$Mods\$Value\$1         EQU         D1\$00\$11\$10B           00927         .         .         Stop bit, no parity         .           0032         .         . <td>1</td> <td></td> <td></td> <td></td> <td></td> <td></td>	1					
0085 =       00915       D1\$Status\$Port       EQU       D1\$Base\$Port + 1         0086 =       00916       D1\$Mode\$Port       EQU       D1\$Base\$Port + 2         0087 =       00917       D1\$Command\$Port       EQU       D1\$Base\$Port + 3         0088 =       00919       D2\$Base\$Port       EQU       C10\$Base\$Port + 8       ;Device 2         0088 =       00920       D2\$Bata\$Port       EQU       D2\$Base\$Port + 1       008         0089 =       00921       D2\$Bata\$Port       EQU       D2\$Base\$Port + 2       008         0088 =       00922       D2\$Bata\$Port       EQU       D2\$Base\$Port + 2       008         0088 =       00923       D2\$Command\$Port       EQU       D2\$Base\$Port + 3       0092         0088 =       00923       D2\$Command\$Port       EQU       D2\$Base\$Port + 3       0092         0088 =       00925       D\$Mode\$Value\$1       EQU       D1\$00\$\$11\$108       10       10         0045 =       00925       D\$Mode\$Value\$2       EQU       00\$11\$1108       19600 baud       197.7Kx on internal clock       197.7Kx       17.7Kx       17.7Kx       17.7Kx       17.7Kx						+ 4 ;Device 1
0085 =       00915       D1\$Status\$Port EQU       D1\$Base\$Port + 1         0086 =       00916       D1\$Mode\$Port EQU       D1\$Base\$Port + 2         0087 =       00917       D1\$Command\$Port EQU       D1\$Base\$Port + 3         00918       00919       D2\$Base\$Port EQU       C10\$Base\$Port + 6 ; Device 2         0088 =       00920       D2\$Bata\$Port EQU       D2\$Base\$Port + 1         0088 =       00920       D2\$Bata\$Port EQU       D2\$Base\$Port + 2         0088 =       00922       D2\$Mode\$Port EQU       D2\$Base\$Port + 2         0088 =       00922       D2\$Mode\$Port EQU       D2\$Base\$Port + 3         0088 =       00923       D2\$Command\$Port EQU       D2\$Base\$Port + 3         0088 =       00925       D\$Mode\$Value\$1 EQU       D1\$0\$\$I\$1\$10B         00927       ;8 bits, Async. ióx rate       00927         0032 =       00928       D\$Mode\$Value\$2 EQU       00\$\$1\$1\$10B         00929       ;7/Kx on internal clock       ;9600 baud         0032 =       00930       ;8 bits, Async. ióx rate         00930 =       ;9600 baud       ;9600 baud         0033 =       ;97       ;8 bits of DT clock         0038 =       00935       D\$Error       EQU       001\$10111B <t< td=""><td></td><td>00914</td><td>D1\$Data\$Port</td><td>EQU</td><td>D1\$Base\$Port</td><td></td></t<>		00914	D1\$Data\$Port	EQU	D1\$Base\$Port	
0086 =       00916       D1\$Mode\$Port       EQU       D1\$Base\$Port + 2         0087 =       00917       D1\$Command\$Port       EQU       D1\$Base\$Port + 3         0088 =       00919       D2\$Base\$Port       EQU       C10\$Base\$Port + 8       ;Device 2         0088 =       00920       D2\$Base\$Port       EQU       D2\$Base\$Port + 1       0088         0088 =       00921       D2\$Status\$Port       EQU       D2\$Base\$Port + 1       0088         0088 =       00922       D2\$Mode\$Port       EQU       D2\$Base\$Port + 2       0088         0088 =       00922       D2\$Mode\$Port       EQU       D2\$Base\$Port + 3       00928         0088 =       00922       D2\$Mode\$Value\$1       EQU       D2\$Base\$Port + 3       01\$00\$11\$108         00924       00925       D\$Mode\$Value\$1       EQU       01\$00\$11\$108       01\$20\$27       18 bits, Async. 16x rate         00927       00920       ;Tx/Rx on internal clock       ;9600 baud       00\$1001118         00928       0\$Sterror       EQU       00\$1001118       ;Normal mode         00930       :strs and DTR active       00\$10110118       00\$100110008       0\$28me as command value plus error reset         0038 =       00935       D\$Error \$Reset	0085 =		D1\$Status\$Port	EQU	D1\$Base\$Port ·	+ 1
0087 =         00917 00918         D1\$Command\$Port EQU         D1\$Base\$Port + 3 00918           0088 =         00920         D2\$Base\$Port EQU         C10\$Base\$Port + 8 ;Device 2           0088 =         00920         D2\$Bata\$Port EQU         D2\$Base\$Port + 1           0088 =         00920         D2\$Bata\$Port EQU         D2\$Base\$Port + 1           0088 =         00920         D2\$Bata\$Port EQU         D2\$Base\$Port + 2           0088 =         00922         D2\$Mode\$Port EQU         D2\$Base\$Port + 3           0088 =         00923         D2\$Command\$Port EQU         D2\$Base\$Port + 3           0088 =         00925         D\$Mode\$Value\$1         EQU         01\$0011\$10B           00927         istop bit, no parity         is bits, Async. 16x rate           0030 =         00928         D\$Mode\$Value\$2         EQU         00\$11\$100B           0032 =         00930         istop bit, no internal clock         istop bit           0032 =         00930         istop bit         istop bit         istop bit           0032 =         00930         istrinion         istrinion         istrinion           0033 =         00931         istrinion         istrinion         istrinion           0038 =         00935         D\$Error						
00918       00919       D2\$Base\$Port       EQU       C10\$Base\$Port + 6       ;Device 2         0088 =       00920       D2\$Data\$Port       EQU       D2\$Base\$Port + 1         0088 =       00921       D2\$Status\$Port EQU       D2\$Base\$Port + 1         0088 =       00922       D2\$Mode\$Port EQU       D2\$Base\$Port + 3         0088 =       00923       D2\$Command\$Port EQU       D2\$Base\$Port + 3         0088 =       00926       ;1 stop bit, no parity         00926       ;1 stop bit, no parity       ;8 bits, Async. 16x rate         00927       00930       ;9600 baud         00928       D\$Mode\$Value\$2 EQU       00\$100111B         00929       ;1x/Rx on internal clock         00930       ;9600 baud         00931       D\$Command\$Value EQU       00\$100111B         00932       ;Perror EQU       00\$11011B         0033       ;Perror \$Reset       EQU       00\$11011B         00337 =       00935       D\$Error \$EQU       00\$11011B         0037 =       00938       D\$Eutput\$Ready       EQU       0001100118         0002 =       00939       1\$Same as command value plus error reset       00000\$0000168         0002 =       00939       D\$Dutput\$Ready						
0088 =         00919         D28Base\$Port         EQU         C108Base\$Port         6         ;Device 2           0088 =         00920         D28Data\$Port         EQU         D28Base\$Port         0         D28Base\$Port         0           0088 =         00921         D2%Base\$Port         EQU         D2%Base\$Port + 1         0           0088 =         00922         D2%Mode\$Port         EQU         D2%Base\$Port + 2         0           0088 =         00923         D2%Command\$Port         EQU         D2%Base\$Port + 3         0           00924         00925         D\$Mode\$Value\$1         EQU         01\$00\$11\$10B         0         0           00926         00927         :8 bits, Async. 16x rate         0         1500bit; no parity           0030 =         00928         D\$Mode\$Value\$2         EQU         00\$11\$10B         12%Command\$Value         12%Command\$Val	1				227222 <b>377 97 1</b>	
0088 =       00920       D28Data\$Port       EQU       D28Base\$Port         0089 =       00921       D28Data\$Port       EQU       D28Base\$Port + 1         0088 =       00923       D28Command\$Port       EQU       D28Base\$Port + 2         0088 =       00923       D28Command\$Port       EQU       D28Base\$Port + 3         0088 =       00925       D\$Mode\$Value\$1       EQU       D1\$00\$11\$108         004E =       00926       ;1       stop bit, no parity         003C =       00928       D\$Mode\$Value\$2       EQU       00\$11\$1008         003C =       00930       provide\$Value\$2       EQU       00\$11\$1008         0037 =       00931       D\$Command\$Value       EQU       00\$110118         0038 =       00935       D\$Error       EQU       00\$110118         0037 =       00935       D\$Error\$Reset       EQU       0011\$10008         0037 =       00936       D\$Error\$Reset       EQU       0011\$10008         0037 =       00935       D\$Error\$Reset       EQU       00110118         00937       00937       jSame as command value plus error reset         0001 =       00938       D\$Utput\$Ready       0000000000         00038 =	0088 =		D2\$Base\$Port	FOU	CIOSRasaSPort	+ 8 •Device 2
0089 =       00921       D2%Status\$Port       EQU       D2%Base\$Port + 1         008A =       00922       D2%Mode\$Port       EQU       D2%Base\$Port + 1         008B =       00924       D2%Dase\$Port       EQU       D2%Base\$Port + 2         004E =       00925       D\$Mode\$Value\$1       EQU       D1%00\$11\$10B         00926       00927       18 top bit, no parity         003C =       00928       D\$Mode\$Value\$2       EQU       00\$11\$1100B         00927       18 top bit, no parity       18 top bit, no parity         003C =       00929       17X/Rx on internal clock         00930       00930       17X/Rx on internal clock         00932       18@Command\$Value       EQU       00\$100111B         00932       18@Command\$Value       EQU       00\$100110B         00932       18@Fror       EQU       00\$100111B         0038 =       00935       D\$Fror\$Reset       EQU       00\$110111B         0037 =       00936       D\$Fror\$Reset       EQU       00\$100110B         0037 =       00938       D\$D\$Dutput\$Ready       EQU       0000100010B         0001 =       00938       D\$Dutput\$Ready       EQU       000000000000000000000000000000000000						· · · · · · · · · · · · · · · · · · ·
OOBA =         OO922         D2%Hode%Port         EQU         D2%Base%Port + 2           OOBB =         OO923         D2%Command%Port EQU         D2%Base%Port + 2           OO4E =         OO925         D\$Mode%Value%1 EQU         O1%O0%11%10B           O0927         istop bit, no parity           O03C =         O0928         D\$Mode%Value%2 EQU         O0%11%100B           O0327         i% bits, Async. 16x rate           O0328         D\$Mode%Value%2 EQU         O0%11%1100B           O0927         i% bits, Async. 16x rate           O0329         j%600 baud           O0930         i% bits           O0931         D\$Command%Value EQU         O0%101011B           O0932         i% normal mode           O0334         i% RTS and DTR active           O037 =         O0935         D%Error           O037 =         O0936         D%Error%eset           O037 =         00938         D%Dutput%Ready         EQU           O001 =         O0938         D%Dutput%Ready         EQU           O0000010B         O0000010B         O0000010B						+ 1
008B =         00923 00924         D2%Command%Port EQU 00925         D2%Base%Port + 3 00926           004E =         00925         D\$Mode%Value\$1         EQU         01\$00\$11\$10B           00926         ;1         stop bit, no parity ;8         j8           003C =         00928         D\$Mode%Value\$2         EQU         00\$11\$1100B           00320         ;7X/Rx on internal clock 00930         ;9600 baud         i9600 baud           0027 =         00931         D\$Command%Value         EQU         00\$100111B           00932         ;Normal mode         ;RTS and DTR active           00934         iRTS and DTR active           0037 =         00936         D%Fror%Reset         EQU         00011011B           00937         ;Same as command value plus error reset         0000000001B         iSame as command value plus error reset           0001 =         00938         D\$Dutput%Ready         EQU         00000010B						
00924         01\$00\$11\$10B           004E =         00925         D\$Mode\$Value\$1         EQU         01\$00\$11\$10B           00927         :8 bits, Async. ióx rate         :8 bits, Async. ióx rate           00927         :8 bits, Async. ióx rate           00928         D\$Mode\$Value\$2         COU           00929         ;Tx/Rx on internal clock           00927 =         00930         :9600 baud           00928 =         00931         D\$Command\$Value         EQU           00928 =         00932         :Normal mode           00930 =         :Panble Tx/Rx         :Panble Tx/Rx           00931 =         00935         D\$Error         EQU         00110111B           0038 =         00935 D\$Error \$EQU         001101100         O0\$101111B           0037 =         00935 D\$Error\$Reset         EQU         00\$110111B           00937 00001 =         00938 D\$D\$Dutput\$Ready         EQU         0000\$100110B           0002 =         00939 D\$Dutput\$Ready         EQU         0000\$00010B						
004E =         00925 00926         D\$Mode\$Value\$1         EQU         01\$00\$11\$10B           00927         18 top bit, no parity 18 bits, Async. 16x rate         16x rate           003C =         00928         D\$Mode\$Value\$2         EQU         00\$11\$100B           00929         00930         17.7Kx on internal clock 19600 baud         19600 baud           00930         19600 baud         00933         18 note           00932         18 note         18 note         19600 baud           00933         18 note         18 note           0038 =         00935         D\$Error         EQU         0011\$1000B           0037 =         00935         D\$Error\$Reset         EQU         0011011B           00937         15 Same as command value plus error reset         00001 =         000930           0001 =         00938         D\$Utput\$Ready         EQU         0000010B	0088 =		D2\$Command\$Port	EQU	D2\$Base\$Port	+ 3
00926         ;1 stop bit, no parity           00927         ;8 bits, Async. 16x rate           003C =         00928         D\$Mode\$Value\$2 EQU         00\$11\$1100B           00929         ;7x/Rx on internal clock         ;9600 baud           00930         ;9600 baud         00\$10111B           00932         ;Normal mode         ;9600 baud           00932         ;Normal mode         ;975 and DTR active           00934         ;RTS and DTR active           0037 =         00935         D\$Fror\$Reset         EQU           00937         :Same as command value plus error reset           0001 =         00938         D\$Utput\$Ready         000000000           0000         000000000         0000000000	1					
00927         ;8 bits, Async. ióx rate           003C =         00928         D\$Mode\$Value\$2         EQU         00\$11\$100B           00929         ;7x/Rx on internal clock         ;9600 baud           00930         ;9600 baud         00\$10111B           00932         ;Enable Tx/Rx         00           00933         ;Enable Tx/Rx           0038 =         00935         D\$Error           0037 =         00936         00\$10111B           0037 =         00935         D\$Error\$ EQU         00\$1001101B           00937         ;Enable Tx/Rx         00\$10111B           0038 =         00935         D\$Error \$EQU         00\$10111B           0037 =         00936         D\$Error \$EQU         00\$1001101B           00937         :Same as command value plus error reset         0000110002           0001 =         00938         D\$Dutput\$Ready         EQU         0000\$00010B	004E =	00925	D\$Mode\$Value\$1	EQU		
00927         ;8 bits, Async. ióx rate           003C =         00928         D\$Mode\$Value\$2         EQU         00\$11\$100B           00929         ;7x/Rx on internal clock         ;9600 baud           00930         ;9600 baud         00\$10111B           00932         ;Enable Tx/Rx         00           00933         ;Enable Tx/Rx           0038 =         00935         D\$Error           0037 =         00936         00\$10111B           0037 =         00935         D\$Error\$ EQU         00\$1001101B           00937         ;Enable Tx/Rx         00\$10111B           0038 =         00935         D\$Error \$EQU         00\$10111B           0037 =         00936         D\$Error \$EQU         00\$1001101B           00937         :Same as command value plus error reset         0000110002           0001 =         00938         D\$Dutput\$Ready         EQU         0000\$00010B	1	00926			;1 st	op bit, no parity
003C =         00928         D\$Mode\$Value\$2         EQU         00\$11\$1100B           00929         ;Tx/Rx on internal clock         ;Tx/Rx on internal clock           00930         ;Tx/Rx on internal clock         ;Tx/Rx on internal clock           00931         D\$Command\$Value         EQU         00\$100111B           00932         ;Normal mode         ;Normal mode           00934         ;RTS and DTR active           0037 =         00935         D\$Error \$Reset           0037 =         00936         D\$Error\$Reset           00937         ;Same as command value plus error reset           0001 =         00938         D\$Luput\$Ready         EQU           00000100         000000000         000000000	1					
00929         ;Tx/Rx on internal clock ;9600 baud           0027 =         00931         D\$Command\$Value EQU 00932         00\$100111B ;Normal mode 00933           00934         ;Riormal mode (00935)         ;Enable Tx/Rx ifTS and DTR active           0038 =         00935         D\$Error EQU 00\$10111B           0037 =         00936         \$Error\$Reset EQU 00\$10111B           00937 =         00938         D\$Euror\$Reset EQU 00\$100110B           00937 =         00938         D\$Euror\$Reset EQU 0000\$0001B           0001 =         00938         D\$Dutput\$Ready EQU 0000\$00010B	003C =		D\$Mode\$Value\$2	EQU		
00930         :9600 baud           0027 =         00931 D\$Command\$Value EQU         00\$100111B           00932         :Normal mode           00934         :Enable Tx/Rx           0038 =         00935 D\$Error         EQU           0037 =         00936 D\$Error\$Reset         EQU           0037 =         00938 D\$Error\$Reset         EQU           0001 =         00938 D\$Dutput\$Ready         EQU           0001 =         00938 D\$Dutput\$Ready         EQU           0001 =         00938 D\$Dutput\$Ready         EQU           0000         00000000         D	1					x on internal clock
0027 =         00931         D\$Command\$Value         EQU         00\$100111B           00932         ;Normal         ;Normal         ;Normal         mode           00933         ;Enable         Tx/Rx           00934         ;RTS and DTR active           0035         D\$Error         EQU         001191000B           0037 =         00936         D\$Error\$Reset         EQU         00911011B           00937         ;Same as command value plus error reset         00000000000001B           0001 =         00938         D\$D\$utput\$Ready         EQU         000000000000000000000000000000000000	1					
00932         ;Normal mode           00933         ;Enable Tx/Rx           00934         ;RTS and DTR active           0037 =         00935         D\$Error \$Rest EQU         00110100B           0037 =         00936         D\$Error \$Rest EQU         000110111B           00937         isame as command value plus error reset           0001 =         00938         D\$Dutput\$Ready         EQU         0000100B           0002 =         00939         D\$Input\$Ready         EQU         000000010B	0027 -		D&Command&Uslue	FOU		
00933         ;Enable Tx/Rx           00934         ;RTS and DTR active           0038 =         00935         D\$Error         EQU         0011\$1000B           0037 =         00936         D\$Error\$Reset         EQU         000110111B           00937         ;Same as command value plus error reset           0001 =         00938         D\$Output\$Ready         EQU         000010B           0002 =         00939         D\$Input\$Ready         EQU         0000\$00010B	,		5+COmmand #Value	. 200		a) mode
00934         rRTS and DTR active           0038 =         00935         D\$Error         EQU         0011\$1000B           0037 =         00936         D\$Error\$Reset         EQU         00\$10111B           00937					; NOT M	el mout
0038 =         00935         D\$Error         EQU         0011\$1000B           0037 =         00936         D\$Error\$Reset         EQU         00\$110111B           00937         ;Same as command value plus error reset           0001 =         00938         D\$Output\$Ready         EQU         0000\$00010B           0002 =         00939         D\$Input\$Ready         EQU         0000\$0010B						
0037 =         00936         D#Error#Reset         EQU         000#110111B           00937         ;Same as command value plus error reset           0001 =         00938         D#Output#Ready         EQU         000000001B           0002 =         00939         D#Input#Ready         EQU         0000#0010B						and DTR active
00937         :Same as command value plus error reset           0001 =         00938         D\$Dutput\$Ready         EQU         000050001B           0002 =         00939         D\$Input\$Ready         EQU         0000\$0010B						
0001 = 00938 D\$Output\$Ready EQU 0000\$0001B 0002 = 00939 D\$Input\$Ready EQU 0000\$0010B	0037 =	00936	D\$Error\$Reset	EQU	00\$110111B	
0001 = 00938 D\$Output\$Ready EQU 0000\$0001B 0002 = 00939 D\$Input\$Ready EQU 0000\$0010B		00937				as command value plus error reset
0002 = 00939 D\$Input\$Ready EQU 0000\$0010B	0001 =		D\$Output\$Ready	EQU		
			D\$Input\$Ready			
						Note: this is actually the

Figure 8-10. (Continued)

00941 00942 00943 0027 = 00944 D\$Raise\$RTS EQU 00\$1\$0011 0007 = 00945 D\$Drop\$RTS EQU 00\$0\$0011 00945 :						
00943 0027 = 00944 D\$Raise\$RTS EQU 00\$1\$0011: 0007 = 00945 D\$Drop\$RTS EQU 00\$0\$0011	; to the DTR pin on the cable 1B ;Raise RTS, Tx/Rx enable					
0027 = 00944 D\$Raise\$RTS EQU 00\$1\$0011: 0007 = 00945 D\$Drop\$RTS EQU 00\$0\$0011:	1B ;Raise RTS, Tx/Rx enable					
0007 = 00945 D\$Drop\$RTS EQU 00\$0\$0011						
	15 ; Drop RIS, 12/RX enable					
00947						
00949 ;	00949 ;					
00950 ; Note : these equates are ;	placed here so that they					
	<pre>; follow the definition of the interrupt vector ; and thus avoid 'P' (phase) errors in ASM.</pre>					
00953 ;						
	Operational control word 1					
	Operational control word 2					
	Operational control word 3					
	Initialization control word 1					
00959 :	Initialization control word 2					
	Nonspecific end of interrupt					
00961 ;	Nonspectific end of interrupt					
	t\$Vector AND 1110\$0000B) + 000\$10110B					
	Sets the A7 - A5 bits of the interrupt					
00964	vector address plus:					
00965 ;	Edge triggered					
00966 ;	4-byte interval					
00967 ;	Single 8259 in system					
00968 ;	No ICW4 needed					
0002 = 00969 IC\$ICW2 EQU Interrupt\$	\$Vector SHR 8					
00970 ;4	Address bits A15 - A8 of the interrupt					
00971 ; 00972 ;	vector address. Note the interrupt					
	vector is the first structure in the long term configuration block					
00974	the long term configuration block					
00FC = 00975 IC\$0CW1 EQU 1111\$1100E	3 ;Interrupt mask					
	Interrupt 0 (clock) enabled					
	Interrupt 1 (character input) enabled					
00978 ;						
01100 ;#						
01101 ;						
01102 ;						
	the specified message on the console.					
	, HL points to a stream of bytes to be					
	A OOH-byte terminates the message.					
	Get next message byte Check if terminator					
	fes, return to caller					
	Prepare for output					
	Save message pointer					
	30 to main console output routine					
0267 E1 01112 POP H ;F	Recover message pointer					
	love to next byte of message					
	oop until complete message output					
01115 ;						
01200 ;#						
01202 Enter\$CPM: ;This routine is e	entered either from the cold or warm					
	ets up the JMP instructions in the also sets the high-level disk driver's					
	dress (the DMA address).					
01206 ;	HESS THE DAM QUIESS/.					
026C 3EC3 01207 MVI A, JMP ;G	Get machine code for JMP					
026E 320000 01208 STA 0000H ;S	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					
	Put address at location 0001H					
01213						
027A 2106CC 01214 LXI H, BD0S\$Entry ; G	Get BDOS entry point address					
	Put address at location 0005H					
01216 ;						
	Set disk I/O address to default					
	Jse normal BIOS routine					
	nsure interrupts are enabled					
0286 FB 01220 E1 ;= 0287 3A0400 01221 LDA Default\$Disk ;H	landover current default disk to					
	console command processor					

Figure 8-10. (Continued)

0288 C300C4	01223	JMP CCP\$Ent	try	;Transfer to CC	Ρ.	
	01224	;				
	01300	; #				
	01301	7				
	01302	; Device table equates ; The drivers use a device table for each ; physical device they service. The equates that follow				
	01303					
	01304					
	01305	; are used to access the various fields within the				
	01306	; device table.				
	01307	7				
	01308	;		numbers and status		
0000 =	01309	DT\$Status\$Port	EQU		status port number	
0001 =	01310	DT\$Data\$Port	EQU	DT\$Status\$Port+		
	01311				data port number	
0002 =	01312	DT\$Output\$Ready	EQU	DT\$BataPort+1		
	01313				ready status mask	
0003 =	01314	DT\$Input\$Ready	EQU	DT\$Output\$Ready		
	01315				ready status mask	
0004 =	01316	DT\$DTR\$Ready	EQU	DT\$Input\$Ready+		
	01317				ady to send mask	
0005 =	01318	DT\$Reset\$Int\$Port	EQU	DT\$DTR\$Ready+1		
	01319				number used to reset an	
aaa	01320	DT4D	F		errupt	
0006 =	01321	DT\$Reset\$Int\$Value	EQU	DT\$Reset\$Int\$Port+1		
	01322		5011	;Value output to reset interrupt		
0007 =	01323	DT\$Detect\$Error\$Port	EQU	DT\$Reset\$Int\$Va		
	01324		E0		umber for detecting error	
0008 =	01325	DT\$Detect\$Error\$Value	EQU	DT\$Detect\$Error	SPORT+1	
	01326				or detecting error (parity etc.)	
0009 =	01327	DT\$Reset\$Error\$Port	EQU	DT\$Detect\$Error		
	01328		Fou		to port to reset error	
000A =	01329	DT\$Reset\$Error\$Value	EQU	DT\$Reset\$Error\$		
	01330		Fou	;value	to output to reset error	
000B =	01331	DT\$RTS\$Control\$Port	EQU	DT\$Reset\$Error\$		
	01332	DT4D	For		ol port for lowering RTS	
= 3000	01333	DT\$Drop\$RTS\$Value	EQU	DT\$RTS\$Control\$		
	01334		EQU	;value,	when output, to drop RTS	
000D =	01335	DT\$Raise\$RTS\$Value	E.6(U	DT\$Drop\$RTS\$Val		
	01336	_		;value,	when output, to raise RTS	
	01337	; Davidaa	1 /			
	01338		logical	1 status (incl. pr	0100015/	
000E =	01339	DT\$Status	EQU	DT\$Raişg\$RTS\$Va		
	01340				;Status bits	
0001 =	01341	<b>BT\$Output\$</b> Suspend	EQU	0000\$0001B	Quiput suspended pending	
	01342		Fou	0000000000	; protocol action	
0002 =	01343	DT\$Input\$Suspend	EQU	0000\$0010B	;Input suspended until	
	01344	DTAQUALUAADTD	EQU	0000\$0100B	; buffer empties ;Output uses DTR-high-to-send	
0004 =	01345	DT\$Output\$DTR				
0008 =	01346	DT\$Output\$Xon	EQU	0000\$1000B	;Output uses XON/XOFF	
0010 =	01347	DT\$Output\$Etx	EQU	0001\$0000B	;Output uses ETX/ACK ;Output uses timeout	
0020 =	01348	BT\$Output\$Timeout	EQU	0010\$00008	;Input uses RTS-high-to-receive	
0040 =	01349	DT\$Input\$RTS	EQU EQU	0100\$0000B	; Input uses XIN-XOFF	
0080 =	01350	BT\$Input\$Xon	EWU	1000\$0000B	JUNKUL USES AUN/AUFF	
000F =	01351	7 DTACANANASO	EQU	DT\$Status+1	;Secondary status byte	
	01352	DT\$Status\$2 DT\$Esko\$Typesbaad	EQU	0000\$0001B	;Secondary status byte ;Requests Input\$Status to	
0001 =	01353	DT\$Fake\$Typeahead	<b>E</b> 90	0000#00015	; return "Data Ready" when	
	01354				; control dharacters are in	
	01355					
	01356				; input buffer	
0010 -	01357	; BT\$Etx\$Count	EQU	DT\$Status\$2+1		
0010 =	01358	DIDEIXDUNI	EWO		f chars, sent in Etx protocol	
0010 -	01359 01360	DT\$Etx\$Message\$Length	EQU	DT\$Etx\$Count+2	i chara, sent in dix protocol	
0012 =		nisc(xsuessadescendiu	EQU		fied message length	
	01361			; opeciti	I A CHILLING AND A CHILLEN	
	01362 01363	1	Inne+	buffer values		
0014 =	01363	; DT\$Buffer\$Base	EQU	DT\$Etx\$Message\$	BLength+2	
0014 =		DIADULELADER	EQU			
0014 -	01365		EQU		ss of Input buffer	
0016 =	01366	DT\$Put\$Offset	EQU	DT\$Buffer\$Base4	+∠ t for putting chars. into buffer	
0017 -	01367	DT\$Get\$Offset	EQU	;UTISE1 DT\$Put\$Offset+1		
0017 =	01368	DIDUGLDUTISEL	200			
0019 -	01369	DT¢Duffav¢lanath¢M1	EQU	;urrset DT\$Get\$Offset+1	t for getting chars. from buffer	
0018 =	01370	DT\$Buffer\$Length\$Mask	EWU			
	01371				h of buffer - 1 Ruffer length must always be	
	01372 01373				Buffer length must always be	
	01373			; a o;	inary number; e.g. 32, 64 or 128	

Figure 8-10. (Continued)

		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
		01381					; the buffer has been reached
0019	=	01382	DT\$Chai	racter\$Co	unt	EQU	DT\$Buffer\$Length\$Mask+1
		01383					;Count of the number of characters
		01384					; currently in the buffer
001A	=	01385	DT\$Stop	p\$Input\$(	ount	EQU	DT\$Character\$Count+1
		01386					Stop input when the count reaches
		01387					; this value
001B	=	01388	DT\$Res	ume\$Input	\$Count	EQU	DT\$Stop\$Input\$Count+1
		01389					Resume input when the count reaches
		01390					; this value
001C	z	01391	DT\$Coni	tro1\$Cour	t	EQU	DT\$Resume\$Input\$Count+1
		01392			-		;Count of the number of control
		01393					; characters in the buffer
001D	=	01394	DT\$Eund	tion\$Del	av	EQU	DT\$Control\$Count+1
		01395	27 <i>47</i> and			Lao	;Number of clock ticks to delay to
		01396					
		01397					; allow all characters after function ; key lead-in to arrive
001E	=	01398	DT\$Ini+	tialize\$9	tream	EQU	DT\$Function\$Delay+1
		01399	2, 21,11		- /	200	;Address of byte stream necessary to
		01377					; initialize this device
		01400					, Initialize (M18 Gevice
		01500	;#				
		01501	; **				
		01502		Device	tables		
		01502	:	Device	740162		
		01503	BT\$0:				
028E	01		BI#0:	nn	DOtCLAN		
028F	80	01505 01506		DB DB			;Status port (8251A chip)
0290		01508		DB	DO\$Data		;Data port
0291		01508		DB	D#Uutpu D#Input	t\$Ready	;Output data ready
0291				DB			;Input data ready
		01509			D\$DTR\$H		;DTR ready to send
0293	08	01510		DB	IC\$0CW2 IC\$E01	*Port	;Reset interrupt port (OOH is an junused port)
0294		01511				***	Reset interrupt value (nonspecific EOI)
0295		01512		DB		us\$Port	;Detect error port
0296		01513		DB	D\$Error		;Mask: framing, overrun, parity errors
0297		01514		DB			Reset error port
0298		01515		DB	D\$Error		Reset error: RTS high, reset, Tx/Rx enable
		01516		DB			;Drop/raise RTS port
029A		01517		DB	D\$Drop\$		;Drop RTS Value (keep Tx & Rx enabled)
029B		01518		DB	D\$Raise		Raise RTS value (keep Tx & Rx enabled)
029C		01519		DB		t\$Xon + E	T\$Input\$RTS ;Protocol and status
029D		01520		DB	0		;Status #2
029E		01521		DW	1024		;Etx/Ack message count
0240		01522		DW	1024		;Etx/Ack message length
02A2		01523		DW	DO\$Buff	er	;Input buffer
02A4	00	01524		DB	0		;Put offset into buffer
0245		01525		DB	0		;Get offset into buffer
02A6		01526		DB		er\$Length	n −1 ;Buffer length mask
02A7		01527		DB	0		;Count of characters in buffer
		01528		DB			a = 5 ;Stop input when count hits this value
02A8		01529		DB		er\$Length	) / 2 ;Resume input when count hits this value
02A8 02A9		01530		DB	0		;Count of control characters in buffer
02A8 02A9 02AA				DB	6		Number of 16.66ms ticks to allow function
02A8 02A9 02AA	06	01531					; key sequence to arrive (approx. 90ms)
02A8 02A9 02AA 02AB		01532					
02A8 02A9 02AA		01532 01533		DW	D0\$Init	ialize\$St	ream ;Address of initialization stream
02A8 02A9 02AA 02AB		01532 01533 01534	;	DW	D0\$Init	ialize\$St	ream ;Address of initialization stream
02A8 02A9 02AA 02AB 02AB	8400	01532 01533 01534 01535	; DT\$1:				
02A8 02A9 02AA 02AB 02AC	8400	01532 01533 01534	; DT\$1:	DB	D1\$Stat	us\$Port	;Status port (8251A chip)
02A8 02A9 02AA 02AB 02AC 02AC	8400 85 84	01532 01533 01534 01535	; DT\$1:	DB DB	D1\$Stat D1\$Data	us\$Port \$Port	;Status port (8251A chip) ;Data port
02A8 02A9 02AA 02AB 02AC 02AC 02AE 02AF 02B0	8400 85 84 01	01532 01533 01534 01535 01536	; DT\$1:	DB DB DB	D1\$Stat D1\$Data D\$Outpu	us\$Port \$Port t\$Ready	;Status port (8251A chip) ;Data port ;Output data ready
02A8 02A9 02AA 02AB 02AC 02AC	8400 85 84 01	01532 01533 01534 01535 01536 01537	; DT\$1:	DB DB DB DB	D1\$Stat D1\$Data D\$Outpu D\$Input	us\$Port \$Port t\$Ready \$Ready	;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready
02A8 02A9 02AA 02AB 02AC 02AC 02AE 02AF 02B0	8400 85 84 01 02	01532 01533 01534 01535 01536 01537 01538	; DT\$1:	DB DB DB	D1\$Stat D1\$Data D\$Outpu	us\$Port \$Port t\$Ready \$Ready	;Status port (8251A chip) ;Data port ;Output data ready
02A8 02A9 02AA 02AB 02AC 02AC 02AE 02AF 02B0 02B1 02B2	8400 85 84 01 02 80	01532 01533 01534 01535 01536 01537 01538 01539	; [\T\$1:	DB DB DB DB	D1\$Stat D1\$Data D\$Outpu D\$Input	us\$Port \$Port t\$Ready \$Ready igh	;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready ;DTR ready to send
02A8 02A9 02AA 02AB 02AC 02AC 02AC 02AF 02B0 02B1 02B2 02B3	8400 85 84 01 02 80 D8	01532 01533 01534 01535 01536 01537 01538 01539 01539 01540 01541	; []T\$1:	DB DB DB DB DB	D1\$Stat D1\$Data D\$Outeu D\$Input D\$DTR\$H	us\$Port \$Port t\$Ready \$Ready igh	;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready
02A8 02A9 02AA 02AB 02AC 02AC 02AF 02B0 02B1 02B2 02B3 02B3	8400 85 84 01 02 80 D8 20	01532 01533 01534 01535 01536 01537 01538 01539 01539 01540 01541 01542	; DT\$1:	DB DB DB DB DB DB DB	D1\$Stat D1\$Data D\$Outpu D\$Input D\$DTR\$H IC\$OCW2 IC\$EOI	us\$Port \$Port t\$Ready \$Ready igh \$Port	;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready ;DTR ready to send ;Reset interrupt port (00H is an unused port ;Reset interrupt value (nonspecific EOI)
02A8 02A9 02AA 02AB 02AC 02AC 02AC 02AF 02B0 02B1 02B2 02B3 02B3 02B4 02B5	8400 85 84 01 02 80 D8 20 85	01532 01533 01534 01535 01536 01537 01538 01539 01540 01541 01542 01543	; [iT\$1:	DB DB DB DB DB DB DB DB DB	D1\$Stat D1\$Data D\$Outpu D\$Input D\$DTR\$H IC\$OCW2 IC\$EOI	us\$Port \$Port t\$Ready \$Ready igh \$Port us\$Port	;Status port (8251A chip) ;Data port ;Dutput data ready ;Input data ready ;DTR ready to send ;Reset interrupt port (00H is an unused port ;Reset interrupt value (nonspecific EOI) ;Detect error port
02A8 02A9 02AA 02AB 02AC 02AC 02AF 02B0 02B1 02B2 02B3 02B3 02B4 02B5 02B6	8400 85 84 01 02 80 D8 20 85 38	01532 01533 01534 01535 01536 01537 01538 01539 01540 01541 01542 01543 01544	; DT\$1:	DB DB DB DB DB DB DB DB DB DB	D1\$Stat D1\$Data D\$Outpu D\$Input D\$DTR\$H IC\$OCW2 IC\$EOI D1\$Stat D\$Error	us\$Port \$Port t\$Ready \$Ready igh \$Port us\$Port	;Status port (8251A chip) ;Data port ;Dutput data ready ;Input data ready ;DTR ready to send ;Reset interrupt port (00H is an unused port ;Reset interrupt value (nonspecific EOI) ;Detect error port ;Mask: framing, overrun, parity errors
02A8 02A9 02AA 02AB 02AC 02AC 02AC 02B0 02B1 02B2 02B3 02B3 02B4 02B5 02B6 02B7	8400 85 84 01 02 90 90 98 20 85 38 87	01532 01533 01534 01535 01535 01537 01538 01539 01549 01541 01542 01543 01544 01545	; [īT\$1:	DB DB DB DB DB DB DB DB DB DB DB	Di\$Stat Di\$Data D\$Outpu D\$Input D\$DTR\$H IC\$OCW2 IC\$EOI D1\$Stat D\$Error D1\$Comm	us\$Port \$Port t\$Ready \$Ready igh \$Port us\$Port us\$Port	;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready ;DTR ready to send ;Reset interrupt port (00H is an unused port ;Reset interrupt value (nonspecific EOI) ;Detect error port ;Mask: framing, overrun, parity errors ;Reset error port
02A8 02A9 02AA 02AB 02AC 02AC 02AF 02B0 02B1 02B2 02B3 02B3 02B4 02B5 02B6	8400 85 84 01 02 80 D8 20 85 38 85 38 87 37	01532 01533 01534 01535 01536 01537 01538 01539 01540 01541 01542 01543 01544	; [:T\$1:	DB DB DB DB DB DB DB DB DB DB	D1\$Stat D1\$Data D\$Outpu D\$Input D\$DTR\$H IC\$OCW2 IC\$EOI D1\$Stat D\$Error D1\$Comm D\$Error	us\$Port \$Port t\$Ready \$Ready igh \$Port us\$Port and\$Port \$Reset	;Status port (8251A chip) ;Data port ;Dutput data ready ;Input data ready ;DTR ready to send ;Reset interrupt port (00H is an unused port ;Reset interrupt value (nonspecific EOI) ;Detect error port ;Mask: framing, overrun, parity errors

Figure 8-10. (Continued)

OBSE 27         OLS9         DB         DPFRieseRTS         rRaise RTS value (keep Tx & Kn enabled)           OBSE 00         OLSS0         DB         OPFInut&STS         rPFInutWSTS         rProcess are cont           OBSE 00         OLSS0         DB         OPFInutWSTS         rProcess are cont           OBSE 00         OLSS0         DB         OPFInutWSTS         rProcess are cont           OBSE 00         OLSS0         DB         DISUIFERT (input buffer         Input of reacters in buffer           OBSE 01         OLSS0         DB         DISUIFERT_Ength - J Stop input when control that is this value           OBSE 01         OLSS0         DB         DISUIFERT_Ength - J Stop input when control that is this value           OBSE 01         OLSS0         DB         DISUIFERT_Ength - J Stop input when control that is this value           OBSE 01         DISUIFERT_Ength - J Stop input when control that is this value         Control that is this value           OBSE 01         DISUIFERT_Ength - J Stop input when control that is this value         Control that is this value           OBSE 01         DISUIFERT_Ength - J Stop input when control that is value         Control that is this value           OBSE 01         DISUIFERT_Ength - J Stop input when control that is value         Control that is this value           OBSE 01         DISUIFER						
0285 0.0         01550         DB         DffInutifXT         :Protocol and status           0280 0.0         01551         DB         0.4         :Status 222         :Protocol and status           0280 0.00         01553         DB         104         :Etu/Ack message length           0222 4422         01556         DB         0         :Put offset into buffer           0224 0.0         01555         DB         0         :Put offset into buffer           0226 0.0         01556         DB         0         :Deutoffset into buffer           0226 0.0         01556         DB         0         :Deutoffset into buffer           0226 0.0         01556         DB         0         :Count of cohroid characters in buffer ion           0226 0.0         01556         DB         0         :Count of cohroid characters in buffer ion           0226 0.0         01556         DB         0         :Count of cohroid characters in buffer ion           0226 0.0         01556         DB         D24Data%Port : Data port ion and tot ion indicaters in buffer ion           0227 0.0         01556         DB         D24Data%Port : Data port ion and tot ion ion ion and ion ion ion ion is an unused port i           0228 0.0         01577         DB         D24Data%Port	0288 27	01549	D.	в	D\$Raise\$RTS	Raise RTS value (keep Tx & Rx enabled)
0280 000 01551 DB 0 rstatus #2 0282 004 01552 DH 1024 relay to the sesses could 024 00 01555 DH 1024 relay to the sesses could 024 00 01555 DB 0 rstatus #2 025 00 01555 DB 0 rstatus #2 025 00 01555 DB 0 rstatus #2 025 10 01559 DB 1980/fertLength - 3 rstop input when could hits this value 025 00 01560 DB 0 rstatus #2 025 10 01550 DB DE D24Status#Port rstatus port (8251A chip) 025 10 01550 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01554 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01554 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01577 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01577 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01577 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01577 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01577 DB DE D24Status#Port rstatus port (8251A chip) 025 0 01577 DB D25Status#Port rstatus port (8251A chip) 025 0 01577 DB D25Status#Port rstatus port (8251A chip) 025 0 01577 DB D25Status#Port rstatus red 025 0 01577 DB D25Status#Port rstatus red 025 0 01577 DB D25Status#Port rstatus red 025 0 01577 DB D25Command#Port rstatus red 025 0 01578 DB D25Command#Port rstatus red 025 0 01579 DB D25Command#Port rstatus red 0250 00 01583 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 00 01585 DB 0 0 rstatus #2 0250 0 01595 DB 0 0 rstatus red 0250 0 0						
0226         0004         0152         DW         1024         FEx/Ack message count           0200         00150         DW         1024         FEx/Ack message length           0200         0155         DW         DisBuffer         iPut offer           0201         0155         DW         DisBuffer/Length         iPut offer           02026         01555         DW         DisBuffer/Length         iDuffer           02026         01550         DB         DisBuffer/Length         iDuffer           02027         01560         DB         DisBuffer/Length         iAck is to allow function           02026         01560         DB         DisBuffer/Length         iAck is to allow function           02027         01564         DW         DisInitialize\$Stream         iAddress of initialization stream           01565         i         DisBuffer         iStatus port         (8251a chip)           02026         01564         DW         DisBuffer/Length         iInitialization stream           02026         01570         DB         DisBuffer/Length         iInitialization stream           02026         01575         DB         DisBuffer/Length         iInitialization           02026         0						
0222 00001       01001       1024       jEtv/Ack message length         0222 0422       01554       DW       0       jPut offset into buffer         0200 000       01555       DB       0       jPut offset into buffer         0200 000       01555       DB       0       jPut offset into buffer         0200 000       01555       DB       0       jPut offset into buffer         0200 000       01555       DB       DIBUfferLength - 5 jStop input when count hits this value         0200 000       01556       DB       DIBUfferLength - 7 jStop input when count hits this value         0200 000       01562       DB       0       ministalized         0200 000       01563       DW       DIBUTferLangth / 2 iffsume to fis contained to arrive (approx. 90ms)         01563       DW       DIBUTSHistalizedStream i Address of initialization stream       initializedStream         0200 000       01564       DB       DBSTRAISHOT i JDState mort (2514 chip)         0201 02       01571       DB       DB DSTRAISHOT i JDState mort       JEState mort         0201 02       01571       DB       DB DSTRAISHOT i JDState mort       JEState mort         0202 000       01571       DB       DB DSTRAISHOT i JDState mort       JEState mort						
0222 4322       01554       DW       Disputfer       ifnut buffer         0225 00       01555       DB       0       Put offset into buffer         0225 00       01555       DB       0       Put offset into buffer         0225 00       01555       DB       0       Put offset into buffer         0226 10       01555       DB       DisputfersLength / 2 ifterus in buffer         0226 10       01560       DB       DisputfersLength / 2 ifterus into out hits this value         0226 10       01564       DB       0       ifterus equance to arrive (approx.)0861         0226 60       01563       DB       D255tatus#Port ifterus equance to arrive (approx.)0861         0226 60       01564       DH       D1%Inteready to arrive (approx.)0861         0226 61       01565       DB       D255tatus#Port ifterus to arrive (approx.)0861         0226 62       01572       DB       D255tatus#Port ifterus to arrive (approx.)0861         0226 63       01577       DB       D255tatus#Port ifterus to arrive (approx.)0861         0226 64       01577       DB       D255tatus#Port ifterus to arrive (approx.)0861         0226 60       01577       DB       D255tatus#Port ifterus to arrive (approx.)0861         0226 60       01577						
0225 00     01555     DB     0     iGet offset into buffer       0225 00     01555     DB     D BufferLength -1 iBuffer Length mask       0226 10     01550     DB     D BufferLength -1 iBuffer Length mask       0227 01     01560     DB     D BufferLength -2 iStor ingut when count hits this value       0226 01     01560     DB     D BufferLength -2 iStor ingut when count hits this value       0227 00     01564     DW     D IsfniferLength -2 iStor ingut when count hits this value       0226 01     01564     DW     D IsfniferLength -2 iStor ingut when count hits this value       0226 06     01563     I     D IsfniferLength -2 iStor ingut when count hits this value       0226 06     01564     DW     D IsfniferLength -2 iStor ingut when count hits this value       0226 08     01572     D D DStatasPort iStar port (S21A chip)       0226 00     01571     DB     D DStatasPort iStar port       0201 02     01571     DB     D DStatasPort iStar port       0202 03     01572     DB     D DStatasPort iStar port       0202 03     01571     DB     D DStatasPort iStar port       0202 03     01572     DB     D DStatasPort iStar port       0204 03     01577     DB     D DStatasPort iStar port       0205 89     01579     DB	02C0 0004	01553	D	W		
6225 00       01556       DB       0       iSet offset into buffer         0226 1F       01557       DB       D19BufferElength -1 Biferelength askt         0227 00       01580       DB       0         0226 1F       01580       DB       0         0227 00       01580       DB       0         0226 00       01561       DB       0       1Februard reprint of characters in buffer         0226 00       01562       DB       0       1Februard reprint of characters in buffer         0226 00       01562       DB       0       1Februard reprint of characters in buffer         0262 01       01563       DH       D1%Initialize%Stream       Addressor of initialization stream         0262 01       01571       DB       D2SStatus#Port       1Status port       5211 chip)         0262 01       01571       DB       D2SStatus#Port       1Data ready       0201 chip)         0262 01       01574       DB       D2SStatus#Port       1Data ready       0201 chip)         0263 00       01574       DB       D2SStatus#Port       Data ready       0201 chip)         0264 00       01574       DB       D2SStatus#Port       Datachip)       D2SStatus#Port       Data ready	0202 4422	01554	D	W	D1\$Buffer	;Input buffer
6225 00       01556       DB       0       iSet offset into buffer         0226 1F       01557       DB       D19BufferElength -1 Biferelength askt         0227 00       01580       DB       0         0226 1F       01580       DB       0         0227 10       01580       DB       0         0226 06       01561       DB       0       1Februard reprint of characters in buffer         0226 06       01562       DB       0       1Februard reprint of characters in buffer         0226 06       01562       DB       0       1Februard reprint of characters in buffer         0262 07       01565       DH       D1%Initialize%Stream       Addressor of initialization stream         0262 08       01565       DB       D2Status#Ort       1Status port       1Status port         0262 01       01571       DB       D2Status#Ort       1Status port       1St	0204 00	01555	n	R	0	:Put offset into buffer
0226 IF     01557     DB     D18DufferLength -1 JBdfferlength mask       0227 00     01558     DB     0     1Count of Characters in buffer       0228 00     01561     DB     0     1Filtering       0226 00     01561     DB     0     0       0226 00     01561     DB     0     0       0226 00     01562     DB     6     1Number of 16.66ms ticks to allow function       0262 00     01563     DB     D18InitializeStream     1Address of initialization stream       0262 00     01564     I     01567     DTS2       0262 00     01567     DTS2     01567     DTS2       0260 01     01572     DB     D28StatusFort iStatus port (82SiA chip)       0261 02     01577     DB     D28StatusFort iStatus port (00H is an unused port)       0262 03     01577     DB     D28StatusFort iFstatus for unused port)       0263 08     01577     DB     D28CommandFort Reset error RTS high, reset, TA/Rt enable       0264 08     01577     DB     D28CommandFort Reset error RTS high, reset, TA/Rt enable       0260 00     01580     DB     D28CommandFort Reset error RTS high, reset, TA/Rt enable       0260 00     01580     DB     D28CommandFort Reset error RTS high, reset, TA/Rt enable       0260 00<						
0227 00 0228 10 01550 DB 01551 DB 01551 DB 01551 DB 01551 DB 01551 DB 01551 DB 01551 DB 01551 DB 01551 DB 01551 DB 01552 DB 01552 DB 01552 DB 01552 DB 01553 DB 01553 DB 01554 DD 01554 DD 01554 DD 01555 1 0255 64 0255 65 01555 DB 01557 DB 01557 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01578 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01578 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01577 DB 01578 DB 01577 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01577 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01578 DB 01579 DB 01578 DB 01579 DB 01578 DB 01579 DB 01578 DB 01579 DB 01579 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01580 DB 01500 C1 01500 C1 01500 C1 01500 C1 01500 C1 0				5	DIADUS (	t shuffen laast maak
0228 18       01559       DB       D18BufferLength - 5 ;Stoe input When count hits this value         0226 04       01561       DB       0       fCount of control characters in buffer         0226 04       01561       DB       0       fCount of control characters in buffer         0226 04       01564       DH       D18InitializetStream       iAddress of initialization stream         01565 1       01565       1       01566       1         01566 1       01567       DB       D2%Eatus%Port iStatus port (#251A chip)         0226 04       01559       DB       D2%Eatus%Port iStatus port (#251A chip)         0226 05       01572       DB       D2%Eatus%Port iStatus port (#251A chip)         0226 06       01572       DB       D2%Eatus%Port istatus port (#251A chip)         0226 01       01572       DB       D2%Eatus%Port istatus port (#251A chip)         0226 02       01577       DB       D2%Eatus%Port istatus port (#251A chip)         0226 03       01577       DB       D2%Eatus%Port istatus port (#251A chip)         0226 04       01577       DB       D2%Eatus%Port istatus port (#00016666666666666666666666666666666666				в	DisbuttersLengtn	a -1 ;Butter length mask
0225 10       01560       DB       DisEuffertLength / 2 ;Resume input when court hits this value         0226 00       01562       DB       0       round of control characters in buffer         0205 00       01564       DW       DisInitialize\$Stream       rAddress of initialization stream         0205 0400       01565       I       DisInitialize\$Stream       rAddress of initialization stream         01565       I       DisStream       rAddress of initialization stream         01567       DT\$2;       DisStream       rAddress of initialization stream         0200 01557       DB       D2%Etatus\$Fort       rStatus port (8251A chip)         0201 02       01571       DB       D10 D2%Etatus\$Fort       ready         0202 80       01572       DB       D10 D2%Etatus\$Fort       ready         0202 80       01574       DB       D10 D2%Etatus\$Fort       ready       ready         0201 02       01577       DB       D2%Etatus\$Fort       ready       ready         0208 01577       DB       D2%Etatus\$Fort       ready       ready       ready         0208 01579       DB       D2%Etatus\$Fort       ready       ready       ready       ready         0209 01       01577       DB						
0225 06     01561     DB     0     fCount of control characters in buffer       0225 06     01562     DW     Disinitialize@Stream iAddress tick to allow function       026 04     01563     DW     Disinitialize@Stream iAddress       026 05     01565     DW     Disinitialize@Stream iAddress       026 05     01565     DE     D2&Status#Port iBata model       026 05     01565     DE     D2&Status#Port iBata model       0210 02     01567     DE     D2&Status#Port iBata model       0211 02     01571     DE     DE     D2&Status#Port iBata model       0210 02     01572     DE     D2&Status#Port iBata model       0211 02     01571     DE     DE     D2&Status#Port iBata model       0210 02     01574     DE     DE     DE       0210 02     01574     DE     DE     DE       0210 02     01576     DE     DE     DE       0217 08     01577     DE     DE     DE       0218 02     01577     DE     DE     DE       0219 02     01576     DE     DE     DE       0210 02     01577     DE     DE     DE       0217 08     01577     DE     DE     DE       0218 00<	0208 18	01559				
0225 06     01562     DB     6     :Number of 16.66ms ticks to allow function       0260 9400     01564     DH     DifinitializeStream     iAddress of initialization stream       01566     DT%1     DE     D2%5tatus#Port iStatus port (82514 chip)       0267 98     01566     DF     DE       0270 01     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0271 01     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0272 80     01572     DB     D2%5tatus#Port iStatus port (82514 chip)       0273 08     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0274 20     01574     DB     D2%5tatus#Port iStatus port (82514 chip)       0275 80     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 81     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus #Differ       0276 07     01580     DB     D2%5tatus#Port iStatus #Differ       0276 07     01580     D4 <td>0209 10</td> <td>01560</td> <td>D</td> <td>в</td> <td></td> <td></td>	0209 10	01560	D	в		
0225 06     01562     DB     6     :Number of 16.66ms ticks to allow function       0260 9400     01564     DH     DifinitializeStream     iAddress of initialization stream       01566     DT%1     DE     D2%5tatus#Port iStatus port (82514 chip)       0267 98     01566     DF     DE       0270 01     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0271 01     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0272 80     01572     DB     D2%5tatus#Port iStatus port (82514 chip)       0273 08     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0274 20     01574     DB     D2%5tatus#Port iStatus port (82514 chip)       0275 80     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 81     01577     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus port (82514 chip)       0276 07     01576     DB     D2%5tatus#Port iStatus #Differ       0276 07     01580     DB     D2%5tatus#Port iStatus #Differ       0276 07     01580     D4 <td>02CA 00</td> <td>01561</td> <td>D</td> <td>B</td> <td>0</td> <td>;Count of control characters in buffer</td>	02CA 00	01561	D	B	0	;Count of control characters in buffer
01563       ; key sequence to arrive (approx. 90mi)         020C 9400       01564       DW         01563       ;         01564       DW       DISINITIALIZEStream       Address of initialization stream         020C 9400       01564       DB       D24Status#Port iStatus port (8251A chip)         020C 9400       01570       DB       D24Status#Port iStatus port (8251A chip)         020D 01       01570       DB       D100Utut#Ready iDut data ready         020D 20       01571       DB       D100Utut#Ready iDut data ready         020D 20       01573       DB       D100Utut#Ready iDut data ready         020D 20       01575       DB       D24Status#Port iDetect error port         020D 20       01577       DB       D24Stommand#Port iRest interror ITs hat         020D 20       01575       DB       D24Stommand#Port iRest interror ITs hat         020D 20       01580       DE       D45Input#Streaterror ITs hat         020D 20       01580       DE       D			n	B	6	Number of 16.66ms ticks to allow function
022C 9400     01554     DH     D1%Initialize\$Stream     ;Address of initialization stream       01555     1     01555     ;       02CE 99     01567     DB     D2%Data%Port     ;Data port       02D0 01     01570     DB     D9%Data%Port     ;Data port       02D1 02     01571     DB     D9%Data%Port     ;Pasta       02D2 00     01572     DB     D9%Data%Port     ;Rest interrupt oft       02D3 02     01573     DB     D9%Data%Port     ;Rest interrupt oft       02D4 02     01574     DB     D9%Data%Port     ;Rest interrupt oft       02D5 08     01575     DB     D9%From #Rest interrupt oft     (00H is an unused port)       02D6 08     01576     DB     D9%From #Rest interrupt oft     (00H is an unused port)       02D6 08     01577     DB     D9%From #Rest interrupt oft     (00H is an unused port)       02D7 08     01576     DB     D9%From #Rest interrupt     (00H is an unused port)       02D8     01577     DB     D9%From #Rest interrupt     (00H is an unused port)       02D8     01576     DB     D9%From #Rest interrupt     (00H is an unused port)       02D8     01576     DB     D9%From #Rest interrupt     (00H is an unused port)       02D8     01577 </td <td>0200 00</td> <td></td> <td>-</td> <td>-</td> <td>•</td> <td>: key sequence to arrive (approx, 90ms)</td>	0200 00		-	-	•	: key sequence to arrive (approx, 90ms)
01565       1         02026       01567       DT#2;         02026       01568       01569       DB         02021       021571       DB       DE Datatafort         02021       021571       DB       DE Datatafort         02021       01572       DB       DE Differing         02025       01572       DB       DE Datatafort         02025       01572       DB       DE Datatafort         02026       01574       DB       DE StonestReady         02026       01576       DB       DEStron       FReet interrupt value (nonsecific EOI)         02026       01576       DB       DEStron       FReet interrupt value (nonsecific EOI)         02026       01576       DB       DEStron       FReet errors       TS high, reset, Tx/Rx enabled)         02026       01576       DB       DEStron       From RTS value (keep Tx & Rx enabled)         02026       01580       DE       DEStron       From RTS value (keep Tx & Rx enabled)         02026       01582       DE       DEStron       From RTS value (keep Tx & Rx enabled)         02026       01582       DE       DEStron       From RTS value (keep Tx & Rx enabled)         02026	0000 0400		n	1.1		
01566         1           02CE 89         01556         DF22:           02CF 89         01556         DB         D28Data%Fort         1Data port           02D1 02         01571         DB         DDEInputReady         input data ready           02D1 02         01571         DB         DDEInputReady         input data ready           02D2 80         01572         DB         DECOMPARISHING         inff ready to send           02D3 02H         01573         DB         DICSCOLE         iff reads         iff reads           02D5 80         01575         DB         DESErcor effort         iff reads         iff reads           02D5 81         01575         DB         DESErcor effort         iff reads         iff reads         iff reads           02D5 82         01575         DB         DESErcor effort         iff reads         iff	0200 9400			~	DI#INICIALIZE#30	Health Houless of Intelativation Stream
01567         DT#2:           02CE 89         01569         DB         D28Data#Fort         jData port           02D0 01         01570         DB         D28Data#Fort         jData port           02D1 01         01570         DB         D28Data#Fort         jData port           02D1 02         01571         DB         D8ThPut\$Medy         input data ready           02D2 03         01574         DB         D16CV28Fort         input data ready           02D5 08         01575         DB         D28Tatus#Fort         ifRest interrupt volt (00H is an unused port)           02D5 08         01577         DB         D28Tore         rHask frame, overrun, parity errors           02D6 07         01580         DB         D5Fror* frame         ifRest interrupt value (knonsectific EDI)           02D6 07         01580         DB         D5Fror*Frame         ifRest error port           02D6 07         01580         DB         D5Fror*Frame         ifRest error if the ifrest into buffer           02D6 07         01580         DB         D5Fror*Frame         ifRest error iffer         ifrest into buffer           02D6 07         01580         DB         D5Fror*Frame         ifrest into buffer         iffer           02D6 00 </td <td></td> <td></td> <td>;</td> <td></td> <td></td> <td></td>			;			
O2CE 8901508DBD28StatusPort(Status port(Status chip)O2CF 8801559DBDSUtputSkeadyfoutput data readyO2D1 0201571DBDBDSUtputSkeadyO2D2 8001572DBDBInput data readyO2D2 8001572DBDBDSUTPUSKeadyO2D5 8901573DBDBDSUTPUSKeadyO2D6 8801574DBDBDSETrorO2D7 8801577DBD28StatusPortIPset interrupt party errorsO2D8 8701577DBD28Command#FortRest error portO2D8 8701578DBDSETrorRest error PortO2D6 8801577DBD28Command#FortIProp RTS value (keep Tx & Rx enabled)O2D6 0701580DBDBDFFrorO2D6 0701580DBDSETrorITsatus#FTO2D6 0701582DBDTFRISSIProp RTS value (keep Tx & Rx enabled)O2D6 0701582DBDTFRISSIProp RTS value (keep Tx & Rx enabled)O2D7 0701580DBD28ErrorITsatus#FTO2D8 0801577DBD28ErrorITsatus#FTO2D6 0001581DH1024IEtx/Ack message countO2D7 01580DB01674ITsatus#FTO2EE 000401584DH1024IEtx/Ack message countO2EE 00001585DHD28EufferElength - 5 Stop input when count hist this valueO2EE 00101590DB0<			;			
0226 01         01559         DB         D24Data%ort         Duty to data ready           0210 02         01571         DB         D%Input%Ready         Input data ready           0210 02         01571         DB         D%Input%Ready         Input data ready           0210 02         01571         DB         D%Input%Ready         Input data ready           0210 02         01573         DB         ICMOUNT         Reset interrupt volt (OOH is an unused port)           0214 20         01574         DB         ICMOUNT         Reset interrupt volt (OOH is an unused port)           0214 20         01575         DB         D%Error*Reset         Reset reror port           0215 88         01579         DB         D%Error*Reset         Reset reror RTS high, reset, Tx/Rx enabled           0215 88         01579         DB         D%Error*Reset         Reset reror RTS high, reset, Tx/Rx enabled           0216 021         01581         DB         D%Error*Reset         Reset reror RTS high, reset, Tx/Rx enabled           0216 021         01581         DB         D%Error*Reset RTS volue (keep Tx A Rx enabled)           0216 021         01582         DB         D*Error*Reset RTS volue (keep Tx A Rx enabled)           0216 021         01582         DB         D*Error						
02D0 0101570DBDSOLFPLIKREADY Input data ready input data ready input data ready input data ready input data ready DTR ready to send PERSET interrupt value (nonspecific ED1) iPERSET interrupt value (near its interrupt value (nonspecific ED1) iPERSET interrupt value (near its interrupt value (near its interrupt value (near its interrupt value (near its interrupt value (near its interrupt value (near its its its value) iPERSET interrupt intervalue value intervalue value intervalue value intervalue value intervalue value intervalue value intervalue value intervalue value intervalue intervalue value value value intervalue value val	02CE 89	01568	D	B	D2\$Status\$Port	;Status port (8251A chip)
02D0 0101570DBDSOLFPLIKREADY Input data ready input data ready input data ready input data ready input data ready DTR ready to send PERSET interrupt value (nonspecific ED1) iPERSET interrupt value (near its interrupt value (nonspecific ED1) iPERSET interrupt value (near its interrupt value (near its interrupt value (near its interrupt value (near its interrupt value (near its interrupt value (near its its its value) iPERSET interrupt intervalue value intervalue value intervalue value intervalue value intervalue value intervalue value intervalue value intervalue value intervalue intervalue value value value intervalue value val	02CF 88	01569	D	в	D2\$Data\$Port	;Data port
02D1         02D         01571         DB         D\$Input\$Ready         ;IDut data ready           02D2         00         01572         DB         DDITR*Nigh         ;IDT ready to send           02D4         00         01574         DB         IC\$OCW28Port         ;Rest interrupt port (00H is an unused port)           02D4         00         01574         DB         IC\$OCW28Port         ;Rest interrupt port (00H is an unused port)           02D5         01575         DB         D2\$Status\$Port         ;Rest interrupt port (00H is an unused port)           02D5         01576         DB         D2\$Commad\$Port         ;Rest interrupt value (incompetific EDI)           02D5         01577         DB         D2\$Commad\$Port         ;Portext interrupt value (keep Tx & Rx enabled)           02D5         01580         DB         D\$Fort#Rest         ;Rest interrupt value (keep Tx & Rx enabled)           02D5         01580         DB         D\$Fort#Status\$Port         ;Portocol and status           02D5         01585         DW         D2\$Euffers         ;Portocol and status           02D5         01585         DW         D2\$Euffers         ;Portocol and status           02E5         00         01587         DB         D\$Eufferstenotbuffer <t< td=""><td></td><td></td><td>n</td><td></td><td></td><td></td></t<>			n			
0202 8001572DBDeDTResignDTR ready to send0203 DB01573DBICCMCW28FortReset interrupt port (00H is an unused port)0204 2001574DBD26E01Reset interrupt part (00H is an unused port)0205 8901575DBD26Econmand#FortPlatk: framing, overrun, parity errors0205 8901576DBD26Econmand#FortReset errors Port0205 8001576DBD26Econmand#FortReset errors Port0206 8701578DBD26Econmand#FortProve the second0206 8701578DBD26Econmand#FortProve the second0206 8701580DBDEFror#ResetProve the second0206 8701581DBD26EcondProve the second0207 8801572DBD26EcondReset errors0208 8701582DBDED76InputKSOProve the second0209 8701581DBD26EnputKSOProve the second0200 0001582DBDED76InputKSOProve the second0200 000401584DH1024FEtX/Ack message count0217 0001587DBD26EvitesFetu enter0218 0001587DBD26EvitesFor input When count hits this value0219 0001587DBD26EvitesFor input When count hits this value0219 0001597DBD26EvitesFor input When count hits this value0219 0001597DBD26EvitesFor i						
0205         01573         DB         IC\$0CW28Port         ;Reset interrupt port (00H is an unused port)           0205         0205         01574         DB         IC\$EDI         ;Reset interrupt value (nonspecific EDI)           0205         01576         DB         DETror         ;Reset interrupt value (nonspecific EDI)           0205         01577         DB         D24Commad\$Fort         ;Reset error port           0205         01577         DB         D24Commad\$Fort         ;Reset error port           0206         01577         DB         D24Commad\$Fort         ;Reset error port           0206         01578         DB         D4Foro*FRiss         ;Protoci and status           0206         01583         DB         D4Foro*FRiss         ;Protoci and status           0207         01580         DB         D4Foro*FRiss         ;Protoci and status           0208         01575         DH         1024         ;Etx/Ack message count           0205         0004         01585         DH         1024         ;Etx/Ack message count           0225         01583         DB         0         ;Fut offset into buffer           0255         01         01592         DB         D24Buffer\$Lenpt - 1 ;Buffer lenpth makt <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
02D4 20         01574         DB         ICMEON         :Rest interrup value (nonspecific EDI)           02D5 59         01575         DB         DSEtatusPort         iNext framing, overrun, parity errors           02D6 88         01577         DB         DSEcror ; Hast framing, overrun, parity errors           02D7 88         01577         DB         DSEcror ; Hest error port           02D8 97         01580         DB         DSEcror ; How is error ; Bort is error ; A R enabled)           02D8 07         01580         DB         DSEcror ; How is error ; Frotocol and status           02D8 07         01580         DB         DSEcror ; How is error ; Frotocol and status           02D8 07         01581         DB         DSEcror ; How is error ; Frotocol and status           02D8 0004         01585         DB         DSEcret ; How is the is enabled)           02D2E 0004         01585         DB         D24         : EtwiAck ressage count           02E2 4422         01586         DB         DSEuffersitenth Uffer         : Edwidek ressage count           02E5 00         01587         DB         D24Buffersitenth - 1 ; Buffer is hout iffer           02E6 116         01592         DB         D24Buffersitenth - 2 ; Fissume input when count hits this value           02E6 116						
0205 89         01575         DB         D25Status#Port         iDetect error port           0205 88         01577         DB         D24Command#Port         rRest error: RTS high, reset, TX/Rx enable           0208 87         01577         DB         D24Command#Port         rRest error: RTS high, rest, TX/Rx enable           0208 87         01579         DB         D24Command#Port         rRest error: RTS high, rest, TX/Rx enable           0209 07         01580         DB         D8Frors#Rest ir Rest error: RTS high, rest, TX/Rx enable           0209 07         01580         DB         D8Frors#Rest ir Rest error: RTS high, rest, TX/Rx enable           0209 07         01580         DB         D8Frors#Rest ir Rest error: RTS high, rest, TX/Rx enable           0200 07         01580         DB         D8Frors#Rest error: RTS high, rest, TX/Rx enable           0200 07         01583         DB         D8Finitation         Protocol and staus           0200 07         01584         DH         1024         Etx/Ack message count           0225 0400         01584         DH         1024         Etx/Ack message count           0255 00         01587         DB         D24Buffer         Enatfer         Interter           0255 0         DB         D24Bufferstensth / 2 rResume input						Reset interrupt port (UVH is an unused port)
0202 SB         01576         DB         DMETror         ::::::::::::::::::::::::::::::::::::						
0206 38         01576         DB         DMETror         TMask: framing, overrun, parity errors           0207 88         01577         DB         DMETror FRest         Frest error port           0208 83         01578         DB         DMETror FRest         Frest error TST high, rest, TX/Rx enable           0209 88         01579         DB         DMETror FRest         Frest error size error law (keep Tx & Rx enabled)           0208 27         01580         DB         DMETror FS value (keep Tx & Rx enabled)           0200 001         01582         DB         DMETror FS value (keep Tx & Rx enabled)           0200 001         01582         DB         DMETror FS value (keep Tx & Rx enabled)           0200 001         01582         DB         DMETror FS value (keep Tx & Rx enabled)           0200 001         01582         DB         DMETror FS value (keep Tx & Rx enabled)           0200 001         01582         DB         DFSInputSME         Protocol and status           0201 01583         DB         D 024         Etx/Ack message length           0225 00         01586         DB         D 28Buffer FLength -1 ; Euriffer length mask           0226 00         01597         DB         D 28BufferSLength - 5 ; Stor input when count hits this value           0256         <					D2\$Status\$Port	
02D7         DB         D24Command#Fort ; Reset error port           02D8         01579         DB         D24Fror#Reset error: RTS high, reset, T.X/Rx enable           02D8         01579         DB         D24Command#Fort ; Drop/raise RTS port           02D8         01579         DB         D3From#Reset ; Reset error: RTS high, reset, TX/Rx enabled)           02D6         01581         DB         DBFnop#RTS ; Drop PTS value (keep Tx & Rx enabled)           02D6         01582         DB         D3Fniput%ert ; Drop/raise RTS value (keep Tx & Rx enabled)           02D0         01583         DE         0         ; Faise RTS value (keep Tx & Rx enabled)           02D0         01583         DE         0         ; Faise RTS value (keep Tx & Rx enabled)           02D0         01583         DE         0         ; Faise RTS value (keep Tx & Rx enabled)           02D0         01584         DH         1024         ; Etx/Ack message count           02E5         000         1585         DF         DF         D24Cofferster           02E6         000         1587         DF         DF         D24Cofferster           02E6         01597         DF         D28Differster         1 ; Stoo input when count hits this value           02E6         01597 <td></td> <td></td> <td>D</td> <td></td> <td>D\$Error</td> <td>;Mask: framing, overrun, parity errors</td>			D		D\$Error	;Mask: framing, overrun, parity errors
O2DB 37         O1578         DB         DBErrorsRest         :Rest error: RTS high, rest, TX/Kx enable           O2DP 38         O1579         DB         DECommandPort i Droyraise RTS port         Status RTS value (keep Tx & Kx enabled)           O2DB 27         O1581         DB         DEFromsRest         :Faise RTS value (keep Tx & Kx enabled)           O2DD 00         O1582         DB         DT%Input%ion + DT%Input%is RTS value (keep Tx & Kx enabled)           O2DD 00         O1582         DB         DT%Input%ion + DT%Input%is RTS value (keep Tx & Kx enabled)           O2DD 00         O1582         DB         DT%Input%ion + DT%Input%is RTS value (keep Tx & Kx enabled)           O2DD 004         O1584         DW         IO24         :Etx/Ack message cont           O2E4 00         O1587         DB         D 2%Buffer         :Input buffer           O2E5 00         O1588         DB         O         :Furthor buffer           O2E5 10         O1592         DB         D2%Buffer%Length - 5 stop input when count hits this value           O2E5 10         O1592         DB         D2%Buffer%Length - 1 scatus #2           O2E6 06         O1594         DB         O         :Count of control characters in buffer           O2E6 040         O1595         DW         D2%Buffer%Length - 1 sc			D	в	D2\$Command\$Port	Reset error port
02D9 8B         01579         DB         D2@Command@Fort ;Drop/rsise RTS port           02D8 07         01580         DB         DBDropSRTS ;Drop RTS value (keep Tx & Rx enabled)           02D0 00         01582         DB         DfsInput\$KTS ;Protocol and status           02D0 00         01583         DB         0 ::status #2           02D0 004         01584         DW         1024           02E0 0004         01585         DW         1024           02E0 0004         01586         DW         1024           02E3 00         01587         DB         0         :Put offset into buffer           02E4 00         01587         DB         0 :Put offset into buffer           02E5 10         01591         DB         D2Buffer\$Length - 5 iston input when count hits this value           02E6 04         01584         DB         0 :Count of control characters in buffer           02E6 04         01595         DB         0 :Count of control characters in buffer           02E6 04         01596         DW         D2@Initia			n	B	D\$Error\$Reset	Reset error: RTS high, reset, Tx/Rx enable
02DB 07         01580         DB         DeDropERTS         :Drop RTS value (keep Tx & Rx enabled)           02DB 27         01581         DB         DTFinput\$Xon + DTFinput\$RTS value (keep Tx & Rx enabled)           02DD 00         01582         DB         DTFinput\$Xon + DTFinput\$RTS :/Frotcol and status           02DD 00         01583         DB         0 :Status #2           02E0 0004         01584         DW         1024         :Etx/Ack message count           02E4 00         01587         DB         0         :FurdAck message length           02E5 00         01588         DB         0         :FurdAck message length           02E5 10         01589         DB         D2#Buffer\$Length -1; Furffer length mask           02E5 10         01590         DB         D2#Buffer\$Length -1; Stor input when count hits this value           02E5 00         01592         DB         D2#Buffer\$Length /2; Resume input when count hits this value           02E6 06         01592         DB         D2#Buffer\$Length /2; Resume input when count hits this value           02E6 06         01597         DB         0; Scout of control characters in buffer           02E6 06         01597         is the sequence to arrive (approx, 90ms)           02E7 400         01597         is the sequence to					12\$Command\$Port	:Drop/raise RTS port
O2DE 27         01581         DB         D\$Raise\$RTS         ; Raise RTS value (keep Tx & Rx enabled)           02DE C0         01582         DB         DT\$Input\$KTS         ; Protocol and status           02DE 0004         01583         DB         0         ; Status #2           02DE 0004         01584         DW         1024         ; Etx/Ack message count           02E2 0004         01585         DW         1024         ; Etx/Ack message count           02E2 0004         01586         DW         D2\$SUffer ; Input buffer           02E2 0001         01587         DB         0         ; Get offset into buffer           02E5 00         01588         DB         0         ; Get offset into buffer           02E7 00         01590         DB         D2\$Buffer\$Length - 1; Suffer length mask         02E8           02E5 10         01592         DB         D2\$Buffer\$Length - 2; Resume input when count hits this value           02E5 40         01593         DB         0         ; Count of control characters in buffer           02E6 400         01594         DB         0 2\$Initialize\$Teram ; Address of initialization stream           02E6 400         01595         ; Key sequence to arrive (approx. 90ms)           02E7 10         ; Set device ini						
02DD C001582DBDT#Input#Xon + DT#Input#TS:Protocol and status02DD 0001583DB0:Status #202ED 000401584DW1024:Etx/Ack message count02ED 000401585DW1024:Etx/Ack message length02E4 00001587DB0:Put offset into buffer02E4 10001587DB0:Put offset into buffer02E5 1001589DB02#0000 (characters in buffer02E6 1F01589DB02#0000 (characters in buffer02E7 0001591DBD2#Buffer#Length - 5 :Stop input when count hits this value02E7 1B01592DBD2#Buffer#Length - 2 :Resume input when count hits this value02E8 0601593DB0:Lews equence to arrive (approx. 90ms)02EA 40001594DB6:Number of 16.66ms ticks to allow function02EA 440001595DWD2#Initialize#Stream :Address of initialization stream01597:General character I/O device initialization01700:#:initialization code.01701:General#CloBinitialization01705:It makes repeated calls to the specific character I/O01707:device initialization01708::table to device table addresses in the01709:::02E7 4F0171::02E7 4F0171::02E7 4F0171:02E7 4F017						
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	020B 27					
02E0000401584DW1024:Etx/Ack message count02E0000401585DW1024:Etx/Ack message length02E40001586DWD28buffer:Input buffer02E401587DB0:Get offset into buffer02E50001588DB0:Get offset into buffer02E501589DBD2%buffer%Length -1 :Buffer length mask02E501589DBD2%buffer%Length -5 :Stop input when count hits this value02E70001592DBD2%buffer%Length /2 :Resume input when count hits this value02E40001593DB0:Count of control characters in buffer02E40001593DB6:Number of 16.66ms ticks to allow function02E50101595DHD2%Initialize*Stream:Address of initialization stream02E60401595DHD2%Initialize*Stream:Address of initialization stream01701:General character I/O device initialization::01702:This routine will be called from the main CP/M01703:It makes repeated calls to the specific character I/O01704:initialization:02E701705:It makes repeated calls to the specific character I/O01707:device initialization:01707:device initialization01707:device initialization02E701710XRA:Set device nu					DISINPUtSion + 1	JISINPUTSRIS ;Protocol and status
02E0000401585DW1024;Etx/Ack message length02E202E201586DWD2%Duffer input buffer02E50001587DB002E501588DBDDet offset into buffer02E501589DBD2%Duffer%Length -1 ;Buffer length mask02E701590DB0:Count of characters in buffer02E701592DBD2%Duffer%Length -5 ;Stop input when count hits this value02E401592DBD2%Duffer%Length -5 ;Stop input when count hits this value02E401592DBD2%Duffer%Length -5 ;Stop input when count hits this value02E401592DBD2%Duffer%Length -5 ;Stop input when count hits this value02E401594DB6;Rumber of 16.66ms ticks to allow function02E601595D2D2%Initialize%Stream;Address of initialization stream0159502ECA40001596DWD2%Initialize%Stream01700:#01702;initialization code.01702:01704: initialization code.01703:this routine will be called from the main CP/M01704:initialization01705:the initialization01706:It makes repeated calls to the specific character I/O01706:it makes repeated calls to the specific character I/O01703:device table addresses in the01704:it alization:01705: </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
02E2       01586       DW       D2Buffer       :Thout Duffer         02E4       00       01587       DB       0       :Det offset into buffer         02E5       00       01588       DB       0       :Det offset into buffer         02E5       00       01588       DB       0       :Det offset into buffer         02E5       01590       DB       D2SBuffer\$Length -1 :Buffer length mask       :Det offset into buffer         02E5       01       01590       DB       D2SBuffer\$Length / 2 :Resume input when count hits this value         02E5       01       01592       DB       D2SBuffer\$Length / 2 :Resume input when count hits this value         02E6       01       01593       DB       0       :Count of control characters in buffer         02E6       01       01595       :       Key sequence to arrive (approx. 90ms)         02E7       01700       :       General character I/0 device initialization         01701       :       General character I/0 device initialization         01702       :       It makes repeated calls to the specific character I/0         01703       :       It makes repeated calls to the specific character I/0         01705       :       It makes repeated calls to the specific character I/0	02DE 0004					
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		01585	D	W	1024	;Etx/Ack message length
02E50001587DB0:put offset into buffer02E50001588DB0:foet offset into buffer02E701589DBD2\$Buffer\$Length -1 : Buffer length mask02E701590DB0:fount of characters in buffer02E81B01591DBD2\$Buffer\$Length - 5 : Stop input when count hits this value02E701592DBD2\$Buffer\$Length / 2 :Resume input when count hits this value02E401593DB0:fount of control characters in buffer02E401593DB0:fount of control characters in buffer02E401593DB0:fount of control characters in buffer02E401594DB6:Number of 16.66ms ticks to allow function01595DWD2\$Initialize\$Stream:Address of initialization stream01700:#D2\$Initialize\$Stream:Address of initialization stream01701:General character I/0 device initialization:01702:#initialization code.01704:initialization code.:01705:It makes repeated calls to the specific character I/001705:it makes repeated calls to the specific character I/001706:It makes repeated calls to the specific to access the01706:it makes repeated calls to the specific character I/002E701715General \$CO\$INITIALIZETON02E701716It makes repeific \$CO\$INITIALIZETON bl			D	iW	D2\$Buffer	
02E5 0001588DB0;Get offset into buffer02E6 1F01589DB0;Count of characters in buffer02E7 0001590DB0;Count of characters in buffer02E8 1B01591DBD29Buffer\$Length - 5 ;Stop input when count hits this value02E9 1001592DBD29Buffer\$Length / 2 ;Resume input when count hits this value02E8 0001593DB0;Count of control characters in buffer02E8 0001595DB0;Count of control characters in buffer02E0 0001596DWD29Initialize\$Stream ;Address of initialization stream01700;#01597;01701;General character I/0 device initialization01702 ;initialization code.01703 ;This routine will be called from the main CP/M01704 ;initialization routine.01705 ;It makes repeated calls to the specific character I/O01706 ;device initialization routine.01707 ;device initialization01708 ;;01709 General\$CIO\$Initialization;02EF 4F0171302F6 CDFA020171502F6 CDFA020171502F7 C3F0020171502F7 C3F0020171502F6 C80171802F7 C3F0020171801709 ;01700 ;#01700 ;#01700 ;#01700 ;#01705 ;02F6 C80171802F7 C3F002 ;;01719 ;01719						
02E6 1F01589DBD2%Buffer%Length -1 ;Buffer length mask02E7 0001590DB0;Count of characters in buffer02E8 1B01591DBD2%Buffer%Length - 5 ;Stop input when count hits this value02E9 1001592DBD2%Buffer%Length - 5 ;Stop input when count hits this value02E4 0001593DB0;Count of control characters in buffer02E4 0001594DB0;Count of control characters in buffer02E5 0601595;Key sequence to arrive (approx. 90ms)02EC A40001596DWD2%Initialize*Stream;Address of initialization stream01597;01701;General character I/O device initialization01702;01703;This routine will be called from the main CP/M01704;initialization code.01705;01706;It makes repeated calls to the specific character I/O01707;device initialization01708;02EE AF01710XRA A01711gcoffextsDevice;02EF 4F0171302F0 CDFA020171502F4 CD0171702F5 CS0171602F4 CD0171702F5 CS0171602F4 CS0171702F5 CS0171802F6 CS0171602F4 FE100171702F4 FE1001					-	.Get offset into buffer
QZE7 0001590DB0fcount of characters in bufferQZE8 1B01591DBD2#Bufferstength / 2 ;Resume input when count hits this valueQZE9 1001592DBD2#Bufferstength / 2 ;Resume input when count hits this valueQZE0 0001593DB0;Count of control characters in bufferQZE0 40001594DB6;Number of 16.66ms ticks to allow function01595	0253 00					yoet offer langth mack
02E8 1B01591DBD2*Buffer\$Length - 5 ; Stop input when count hits this value02E9 1001592DB0;Count of control characters in buffer02E8 0601593DB0;Count of control characters in buffer02E8 0601594DB6;Number of 16.66ms ticks to allow function02E7 0001595DWD2*Initialize*Stream;Address of initialization stream01597;01700;#01701;General character I/O device initialization.01702;01703;This routine will be called from the main CP/M01704;initialization code.01705;01706;It makes repeated calls to the specific character I/O01707;device initialization:02EE AF01710XRA A; table of device table addresses in the01702:01703:02EF 4F0171301714CI\$Next\$Device:.02F0 CDFA0201715CALL Specific\$CI0\$Initialization02F101717CPI1602F3 3C01718RZ.02F4 CS100201718RZ02F50270.01703:.01704:.01705.02F4 FE100171702F4 FE1001717 <td>0200 17</td> <td></td> <td></td> <td></td> <td></td> <td>r -i ybarrer iengin mask -Couch of chousehous is buffer</td>	0200 17					r -i ybarrer iengin mask -Couch of chousehous is buffer
02E9 1001592DBD2*Buffer\$Length / 2 ;Resume input when count hits this value02EA 0001593DB0;Count of control characters in buffer02ED 0601594DB6;Number of 16.66ms ticks to allow function01595						
OZEA 0001593DB0;Count of control characters in bufferOZEB 0601594DB6;Kumber of 16.66ms ticks to allow function01595DWD2\$Initialize\$Stream;Address of initialization stream01597;01700;#01700;#01701;01701;General character I/O device initialization01702;0170301703;This routine will be called from the main CP/M01704;initialization code.01705;0170601706;It makes repeated calls to the specific character I/O01707;device initialization01708;it makes repeated calls to the specific character I/O01709General\$CIO\$Initialization;Set device number (used to access the ; configuration block)02EE AF01710XRA;Set device number (used to access the ; configuration block)02EF 4F01713MOVC,A ; Match to externally CALLable interface02F3 6201714GC1\$Next\$Device:02F4 FE1001717CPI02F7 C3F00201718RZ ; have been initialized01700 ;#						
02EB 06       01594       DB 6       Number of 16.66ms ticks to allow function 01595         02EC A400       01596       DW D2\$Initialize\$Stream ;Address of initialization stream 01597 ;         01500       ;#         01701       ;       General character I/O device initialization 01702 ;         01703       ;       This routine will be called from the main CP/M 01704 ;         01705 ;       initialization code.         01706 ;       It makes repeated calls to the specific character I/O 01707 ;         01706 ;       it makes repeated calls to the specific character I/O 01707 ;         01708 ;       it makes repeated calls to the specific character I/O 01707 ;         01708 ;       it makes repeated calls to the specific character I/O 01707 ;         01708 ;       it makes repeated calls to the specific character I/O 01707 ;         01708 ;       it makes repeated calls to the specific character I/O 01707 ;         02EE AF       01710 XRA A ;Set device number (used to access the 01711 ;         02EF 4F       01713 MOV C,A ;Match to externally CALLable interface         02F14 GCI\$Next\$Device:       ;Initialization ;Initialization ;Initialize the device         02F3 C2       01715 CALL Specific\$CI0\$Initialization ;Initialized         02F4 FE10       01717 CF1 16 ;Check if all possible devices (0 - 15)         02F7 C3F002       01719 JMP GCI\$Next\$D	02E9 10	01592				
02EB 0601594DB6Number of 16,66ms ticks to allow function (Key sequence to arrive (approx. 90ms)02EC A40001595DWD2\$Initialize\$Stream;Address of initialization stream01597;	02EA 00	01593	D	B	0	;Count of control characters in buffer
<ul> <li>01595</li> <li>02EC A400</li> <li>01596</li> <li>DW D2\$Initialize\$Stream ;Address of initialization stream</li> <li>01700 ;#</li> <li>01701 ;</li> <li>01701 ;</li> <li>01702 ;</li> <li>01703 ;</li> <li>01703 ;</li> <li>01704 ;</li> <li>01704 ;</li> <li>01704 ;</li> <li>01705 ;</li> <li>01705 ;</li> <li>01706 ;</li> <li>01706 ;</li> <li>01707 ;</li> <li>device initialization code.</li> <li>01707 ;</li> <li>device initialization routine.</li> <li>01708 ;</li> <li>01709 General\$CI0\$Initialization:</li> <li>02EE AF</li> <li>01710 XRA A ;</li> <li>Set device number (used to access the</li> <li>01711 ;</li> <li>01709 General\$CI0\$Initialization:</li> <li>02EE AF</li> <li>01710 XRA A ;</li> <li>Set device number (used to access the</li> <li>01712 ;</li> <li>01714 ;</li> <li>01714 ;</li> <li>01715 GCI\$Next\$Device:</li> <li>02F0 CDFA02 01715 CALL Specific\$CI0\$Initialization ;</li> <li>02F4 FE10 01717 CPI 16 ;</li> <li>02F7 C3F002 01718 RZ ;</li> <li>02F7 C3F002 01718 RZ ;</li> <li>02F7 C3F002 ;</li> <li>01709 GCI\$Next\$Device:</li> <li>01709 ;</li> <li>02F7 C3F002 ;</li> <li>01718 RZ ;</li> <li>01800 ;</li> <li>01800 ;</li> <li>01800 ;</li> <li>01801 ;</li> <li>01801 ;</li> <li>01802 ;</li> <li>01804 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0159 ;</li> <li>0150 ;</li> <li>0150 ;</li> <li>0</li></ul>			D	в		
02EC A400       01596       DW       D2\$Initialize\$Stream       ;Address of initialization stream         01700       ;#         01701       ;       General character I/0 device initialization         01702       ;         01703       ;       This routine will be called from the main CP/M         01704       ;       initialization code.         01705       ;       01706         01706       ;       It makes repeated calls to the specific character I/0         01707       ;       device initialization         01708       ;       it makes repeated calls to the specific character I/0         01707       ;       device initialization         01708       ;       it makes repeated calls to the specific character I/0         01709       General\$CIO\$Initialization       routine.         01709       General\$CIO\$Initialization       ;         01709       General\$CIO\$Initialization       ;         01709       General\$CIO\$Initialization       ;         01701       ;       device         01702       ;       ;         01701       ;       device         01702       ;       ;         02E6 CDFA02       01715       CAL			•		-	
<ul> <li>01597 ;</li> <li>01700 ;#</li> <li>01701 ; General character I/O device initialization</li> <li>01702 ;</li> <li>01703 ; This routine will be called from the main CP/M</li> <li>01704 ; initialization code.</li> <li>01705 ;</li> <li>01706 ; It makes repeated calls to the specific character I/O</li> <li>01707 ; device initialization routine.</li> <li>01708 ;</li> <li>01709 General\$CIO\$Initialization:</li> <li>02EE AF 01710 XRA A ; Set device number (used to access the 01711 ; table of device table addresses in the 11712 ; configuration block)</li> <li>02EF 4F 01713 MOV C,A ; Match to externally CALLable interface</li> <li>02F3 C0FA02 01715 CALL Specific\$CIO\$Initialization ; Initialize the device 02F3 3C 01716 INR A ; Move to next device</li> <li>02F7 C3F002 01718 RZ ; have been initialized</li> <li>02F7 C3F002 01719 JMP GCI\$Next\$Device:</li> <li>01800 ;#</li> <li>01800 ;</li> <li>01800 ;</li> <li>01801 ;</li> <li>01803 ;</li> <li>01804 ; This routine outputs the specified byte values to the specified other streams in the</li> </ul>	0250 4400		n		D2#Initializa#94	
<ul> <li>01700 ;#</li> <li>01701 ; General character I/O device initialization</li> <li>01702 ;</li> <li>01703 ; This routine will be called from the main CP/M</li> <li>01704 ; initialization code.</li> <li>01705 ;</li> <li>01705 ;</li> <li>01706 ; It makes repeated calls to the specific character I/O</li> <li>01707 ; device initialization routine.</li> <li>01708 ;</li> <li>01709 General\$CIO\$Initialization:</li> <li>02EE AF</li> <li>01710 XRA A ; Set device number (used to access the</li> <li>01711 ; table of device table addresses in the</li> <li>01712 ; table of device table addresses in the</li> <li>01713 MOV C,A ; Match to externally CALLable interface</li> <li>02FF 4F</li> <li>01715 CALL Specific\$CIO\$Initialization ; Initialize the device</li> <li>02F7 C3F002 01715 CALL Specific\$CIO\$Initialization ; have been initialized</li> <li>02F7 C3F002 01718 RZ ; have been initialized</li> <li>02F7 C3F002 ;</li> <li>01709 GC1\$Next\$Device</li> <li>01700 ;</li> <li>01714 GCI\$Next\$Device</li> <li>01715 CALL Specific character I/O</li> <li>01716 OFF 16 ; have been initialized</li> <li>01717 CPI 16 ; have been initialized</li> <li>01700 ;</li> <li>01800 ;</li> <li>01800 ;</li> <li>01800 ;</li> <li>01800 ;</li> <li>01801 ;</li> <li>01802 ; Specific character I/O initialization</li> <li>01803 ;</li> <li>01804 ; This routine outputs the specified byte values to the specified olisot streams in the</li> </ul>	VALU MAUV				D2+101(101128#30	ream products of initialization stream
<ul> <li>01701 ; General character I/O device initialization</li> <li>01702 ;</li> <li>01703 ; This routine will be called from the main CP/M</li> <li>01704 ; initialization code.</li> <li>01705 ;</li> <li>01706 ; It makes repeated calls to the specific character I/O</li> <li>01707 ; device initialization routine.</li> <li>01708 ;</li> <li>01709 General\$CIO\$Initialization routine.</li> <li>01709 General\$CIO\$Initialization:</li> <li>02EE AF</li> <li>01710 XRA A ;Set device number (used to access the 01711 ; table of device table addresses in the 01712 ; configuration block)</li> <li>02EF 4F</li> <li>01715 GCI\$Next\$Device:</li> <li>02F0 CDFA02 01715 CALL Specific\$CIO\$Initialization ; Initialize the device 02F3 3C 01716 INR A ;Move to next device</li> <li>02F7 C3F002 01719 JMP GCI\$Next\$Device in the been initialized</li> <li>01709 JMP GCI\$Next\$Device initialization ; Initialized</li> <li>02F7 C3F002 01719 JMP GCI\$Next\$Device initialization 01803 ;</li> <li>01804 ; This routine outputs the specified byte values to the specified olign streams in the</li> </ul>						
<ul> <li>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 01707 ; device initialization routine. 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization: 01709 General\$CIO\$Initialization block)</li> <li>02EF 4F 01713 MOV C,A ;Match to externally CALLable interface 01714 GCI\$Next\$Device: 01714 GCI\$Next\$Device: 01716 INR A ;Move to next device 02F3 C3C 01715 CALL Specific\$CIO\$Initialization ;Initialize the device 02F4 FE10 01717 CPI 16 ;Check if all possible devices (0 - 15) 02F6 C8 01718 RZ 01709 ; 01800 ;# 01800 ;# 01800 ; 01800 ; 01800 ; 01801 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the</li> </ul>						
<ul> <li>01703 ; This routine will be called from the main CP/M</li> <li>01704 ; initialization code.</li> <li>01705 ;</li> <li>01706 ; It makes repeated calls to the specific character I/O</li> <li>01707 ; device initialization routine.</li> <li>01708 ;</li> <li>01709 General\$CIO\$Initialization:</li> <li>01709 General\$CIO\$Initialization:</li> <li>01709 General\$CIO\$Initialization:</li> <li>01710 XRA A ; Set device number (used to access the 01711 ; table of device table addresses in the 01712 ; configuration block)</li> <li>02EF 4F 01713 MOV C,A ; Match to externally CALLable interface 01714 GCI\$Next\$Device:</li> <li>02F0 CDFA02 01715 CALL Specific\$CIO\$Initialization ; Initialize the device 02F3 3C 01716 INR A ; Move to next device</li> <li>02F7 C3F002 01719 JMP GCI\$Next\$Device in the been initialized 01720 ; 01800 ;# 01800 ;</li> <li>01800 ;# 01801 ; 01802 ; Specific character I/O initialization discontrolled by the initialization streams in the</li> </ul>				ieneral	character I/O de	vice initialization
01704 ; initialization code. 01705 ; 01705 ; 01706 ; It makes repeated calls to the specific character I/O 01707 ; device initialization routine. 01708 ; 01709 General\$CIO\$Initialization: 02EE AF 01710 XRA A ;Set device number (used to access the 01711 ; table of device table addresses in the 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 01715 CALL Specific\$CIO\$Initialization ;Initialize the device 02F4 FE10 01717 CPI 16 ; Check if all possible devices (0 - 15) 02F7 C3F002 01719 JMP GCI\$Next\$Device 01720 ; 01800 ;# 01800 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the						
01704 ; initialization code. 01705 ; 01705 ; 01706 ; It makes repeated calls to the specific character I/O 01707 ; device initialization routine. 01708 ; 01709 General\$CIO\$Initialization: 02EE AF 01710 XRA A ;Set device number (used to access the 01711 ; table of device table addresses in the 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 01715 CALL Specific\$CIO\$Initialization ;Initialize the device 02F4 FE10 01717 CPI 16 ; Check if all possible devices (0 - 15) 02F7 C3F002 01719 JMP GCI\$Next\$Device 01720 ; 01800 ;# 01800 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the	1					led from the main CP/M
<ul> <li>01705 ; 01706 ; It makes repeated calls to the specific character I/0 01707 ; device initialization routine. 01708 ; 01709 General\$CIO\$Initialization:</li> <li>02EE AF 01710 KR A ; set device number (used to access the 01711 ; table of device table addresses in the 01712 ; table of device table addresses in the 01711 ; table of device table interface 01712 ; table of device table interface 01714 GCI\$Next\$Device:</li> <li>02F0 CDFA02 01715 GCL\$Next\$Device:</li> <li>02F3 3C 01716 INR A ;Move to next device 01714 GCI * RZ ; there in the initialized 02F7 C3F002 01719 ; UMP GCI\$Next\$Device</li> <li>02F7 C3F002 01719 ; MMP GCI\$Next\$Device</li> <li>01800 ;# 01800 ;</li> <li>01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the</li> </ul>						
01706       ;       It makes repeated calls to the specific character I/0         01707       ;       device initialization routine.         01708       ;       01709         01709       General\$CIO\$Initialization:       01709         02EE AF       01710       XRA       ; Set device number (used to access the 01711         02EF AF       01710       XRA       ; configuration block)         02EF 4F       01713       MOV       C,A       ; Match to externally CALLable interface         02F0       CDFA02       01715       CALL       Specific\$CIO\$Initialization       ; Initialize the device         02F3       01716       INR       A       ; Move to next device       02F3         02F4       FE10       01717       CPI       16       ; have been initialized       02F3         02F7       C3F002       01718       RZ       ; have been initialized       01710       15         02F7       C3F002       01719       JMP       GCI\$Next\$Device       01720       ;       01800       ;         01800       ;#       01800       ;       inve been initialized       01720       ;         01801       ;       01801       ;       01801       ;	1		•			
01707       ;       device initialization routine.         01708       ;         01709       General\$CIO\$Initialization:         02EE AF       01710       XRA       ; Set device number (used to access the off711         01711       ; table of device table addresses in the off712       ; configuration block)         02EF 4F       01713       MOV       C,A       ; Match to externally CALLable interface         02F0 CDFA02       01715       CALL       Specific\$CIO\$Initialization       ; Initialize the device         02F3 C0       01715       CALL       Specific\$CIO\$Initialization       ; Initialize the device         02F3 C0       01717       CPI       16       ; Move to next device         02F4 FE10       01717       CPI       16       ; Check if all possible devices (0 - 15)         02F7 C3F002       01718       RZ       ; have been initialized         02F7 C3F002       01719       JMP       GCI\$Next\$Device         01800       ;       01800       ;         01800       ;       01800       ;         01801       ;       Diso1       ;         01802       ;       Specific character I/0 initialization         01803       ;       O1805       ;				t 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 the 01712 ; table of device table addresses in the 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 ;# 01800 ; 01801 ; 01802 ; Specific character I/0 initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the						
01709       General\$CIO\$Initialization:         02EE AF       01710       XRA       ; table of device number (used to access the 01711         01712       ; table of device table addresses in the 01712       ; configuration block)         02EF 4F       01713       MOV       C,A         02F0 CDFA02       01715       CALL       Specific\$CIO\$Initialization         02F3 CC       01715       CALL       Specific\$CIO\$Initialization         02F4 FE10       01717       CPI       16         02F7 C3F002       01718       RZ       ; have been initialized         02F7 C3F002       01719       JMP       GCI\$Next\$Device         01712       .       .       .         01714       GCI\$Next\$Device:       .       .         02F3 CC       01717       CPI       16       ; have been initialized         02F7 C3F002       01719       JMP       GCI\$Next\$Device       .         01800       ;#       .       .       .         01800       ;#       .       .       .         01800       .       .       .       .         01717       .       .       .       .         01800       .       .<	1			CATCE 1	manage and the second second	
02EE AF       01710       XRA: A       ;Set device number (used to access the 01711         01711       ; table of device table addresses in the j configuration block)         02EF 4F       01713       MOV C,A       ;Match to externally CALLable interface 01714         02EF 4F       01713       MOV C,A       ;Match to externally CALLable interface 01714         02F0 CDFA02       01715       CALL       Specific%CIO\$Initialization       ;Initialize the device 02F4 FE10         02F6 CB       01716       INR       A       ;Move to next device       02F0         02F6 CB       01717       CPI       16       ;Check if all possible devices (0 - 15)       02F6 CB         02F7 C3F002       01719       JMP       GCI\$Next\$Device       .       .       .         02F7 C3F002       01719       JMP       GCI\$Next\$Device       .       .       .         01800       ;#       .       .       .       .       .       .         01801       ;       .       .       .       .       .       .       .         01802       ;       Specific character I/0 initialization       .       .       .       .       .       .       .       .       .       .       .			1		4-14	
01711       ; table of device table addresses in the         01712       ; configuration block)         02EF 4F       01713         01714       GCI\$Next\$Device:         02F0 CDFA02       01715         02F3 C       01716         01717       CPI 16         02F6 CB       01717         01718       RZ         02F7 C3F002       01719         01719       JMP         01710       F         01711       RZ         01712       ; have been initialized         01714       GCI\$Next\$Device:         01715       CALL         01716       INR A         : Check if all possible devices (0 - 15)         02F6 C8       01718         01720       ;         01800       ;#         01800       ;#         01801       ;         01802       ;         01803       ;         01803       ;         01804       ;         01805       ;         01805       ;         01805       ;         01805       ;         01805       ;						
01712       j configuration block)         02EF 4F       01713       MOV C.A       ; Match to externally CALLable interface         02F0 CDFA02       01714       GCI\$Next\$Device:       ; Match to externally CALLable interface         02F3 CD 01715       CALL       Specific\$CI0\$Initialization       ; Initialize the device         02F3 CD       01716       INR       A       ; Move to next device         02F4 FE10       01717       CPI       16       ; Check if all possible devices (0 - 15)         02F7 C3F002       01718       RZ       ; have been initialized       02F7         02F7 C3F002       01719       JMP       GCI\$Next\$Device       01800       ;         01800       ;       01800       ;       01800       ;         01800       ;       01800       ;       01802       ;         01802       ;       Specific character I/0 initialization       01803       ;         01803       ;       01804       ;       This routine outputs the specified byte values to the specified         01805       ;       ports as controlled by the initialization streams in the       01805	02EE AF		X	RA	A	
01712       ; configuration block)         02EF 4F       01713       MOV C,A       ;Match to externally CALLable interface         02F0 CDFA02       01714       GCI\$Next\$Device:       ;Match to externally CALLable interface         02F0 CDFA02       01715       CALL       Specific\$CI0\$Initialization       ;Initialize the device         02F3 C0       01716       INR       A       ;Move to next device         02F4 FE10       01717       CPI       16       ;Check if all possible devices (0 - 15)         02F7 C3F002       01718       RZ       ; have been initialized         02F7 C3F002       01719       JMP       GCI\$Next\$Device         01800       ;#       01800       ;         01800       ;#       01800       ;         01802       ;       Specific character I/0 initialization         01803       ;       01804       ;         01805       ;       ports as controlled by the initialization streams in the		01711				
02EF 4F       01713       MOV C,A       ;Match to externally CALLable interface         01714       GCI\$Next\$Device:       01714       GCI\$Next\$Device:         02F0 CDFA02       01715       GCI \$Next\$Device:       ;Initialization         02F3 3C       01716       INR       A       ;Move to next device         02F4 FE10       01717       CPI       16       ;Check if all possible devices (0 - 15)         02F6 CS       01718       RZ       ; have been initialized         02F7 C3F002       01719       JMP       GCI\$Next\$Device         01800       ;#       01800       ;         01800       ;#       01800       ;         01802       ;       Specific character I/0 initialization         01803       ;       01803       ;         01803       ;       This routine outputs the specified byte values to the specified         01805       ;       ports as controlled by the initialization streams in the						
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 CS 01718 RZ ; have been initialized 01720 ; 01800 ;# 01800 ; 01802 ; Specific character I/0 initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the	02EE 4E		м	aν	C. A	
O2F0 CDFA02       01715       CALL Specific\$CI0\$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         02F7 C3F002       01719       JMP GCI\$Next\$Device						,
O2F3 3C       01716       INR       A       Hove to next device         O2F4 FE10       01717       CPI       16       Check if all possible devices (0 - 15)         O2F6 C8       01718       RZ       ; have been initialized         02F7 C3F002       01719       JMP       GC1\$Next\$Device         01800       ;#       01800       ;         01800       ;       01801       ;         01802       ;       Specific character I/O initialization       01803         01803       ;       O1804       ;       This routine outputs the specified byte values to the specified         01805       ;       ports as controlled by the initialization streams in the       01805	0050 005400					itistics . Initialize the device
02F4 FE10       01717       CPI       16       ; Check if all possible devices (0 - 15)         02F6 CS       01718       RZ       ; have been initialized         02F7 C3F002       01719       JMP       GCI\$Next\$Device         01700       ;       01800       ;#         01800       ;#       01801       ;         01802       ;       Specific character I/O initialization       01803         01803       ;       01804       ;       This routine outputs the specified byte values to the specified         01805       ;       ports as controlled by the initialization streams in the       01805					~ ~	
02F6 C8       01718       RZ       ; have been initialized         02F7 C3F002       01719       JMP       GCI\$Next\$Device         01720       ;       01800       ;#         01800       ;#       01801       ;         01802       ;       Specific character I/O initialization       01802         01803       ;       01804       ;       This routine outputs the specified byte values to the specified         01805       ;       ports as controlled by the initialization streams in the       01805						THORE TO HEXT DEVICE
02F7 C3F002 01719 JMP GCI\$Next\$Device 01720 ; 01800 ;# 01801 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the					16	
01720 ; 01800 ;# 01801 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the						; have been initialized
01720 ; 01800 ;# 01801 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the	02F7 C3F002		J	MP	GCI\$Next\$Device	i de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de la constante de
01800 ;# 01801 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the			;			
01801 ; 01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the						
01802 ; Specific character I/O initialization 01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the						
01803 ; 01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the	1			manifi-	obaractor 1/0 -	nitialization
01804 ; This routine outputs the specified byte values to the specified 01805 ; ports as controlled by the initialization streams in the	1			MAC1110	character 1/U 1	
01805 ; ports as controlled by the initialization streams in the						terran a causa a secondaria a
01806 ; configuration block. Each device table contains a pointer to		01805				
		01806	; c	onfigur	ation block. Eac	ch device table contains a pointer to
	L					

Figure 8-10. (Continued)

r					
	01807	; these	streams.	The devic	e table itself is selected according
	01808	; to the	device M	UMBER	this is an entry parameter for this
	01809	; routin	e.		
	01810				led either from the general device
	01811	; initia	lization	routine a	bove, or directly by a BIOS call from
	01812		em utilit	y executi	ng in the TPA.
	01813 01814	; 			
	01814	; Entry	parameter	5	
	01816	,		vice numbe	
	01817		C - 084	TCe Humbe	T
	01818		arameters		
	01819		arameters		
	01820		A = Dev	ice numbe	r (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 02FD 4F	01827	ADD	<u>A</u> .		Make device number into word pointer;
02FE 0600	01828 01829	MOV MVI	C,A		-Male into a could
0300 216400	01830	LXI	B,0 H CR#D-		;Make into a word e\$Addresses    ;Get table base
0303 09	01831	DAD	R		;HL -> device table address
0304 5E	01832	MOV	Ë.M		;Get LS byte
0305 23	01833	INX	H		, D,
0306 56	01834	MOV	<u>р</u> ,м		;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\$Exi	t	;Yes, device table nonexistent
	01839				
030C 211E00	01840	LXI		itialize\$	
030F 19 0310 5E	01841	DAD	D		;HL -> initialization stream address
0310 55	01842	MOV	E,M		;Get LS byte
0312 56	01843 01844	INX MOV	н D, M		-Cat MC but-
0312 08 0313 EB	01845	XCHG	L, M		;Get MS byte ;HL -> initialization stream itself
0314 CD1903	01846	CALL	Outouts	Byte\$Stre	am ;Output byte stream to various
	01847		cutputt	271110111	; 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 str	eam	
	02002	₽ <u></u> , ,	• •		
1	02003 02004				tialization bytes to port
	02004	t number:	s. ine by	ie stream	has the following format:
	02003	7	DB	PPH I	Port number
	02007	,	DB		Number of bytes to output
	02008	;	DB	VVH, VVH.	
	02009	;			
	02010	;	:	Repeated	
	02011	;	:		
	02012	;	DB	оон і	Port number of 0 terminates
1	02013	;			
	02014		parameter	s	
	02015	2	มางก		_
	02016 02017	;	nu -> B	yte stream	n
	02017	; Output\$Byte\$Str	ream:		
	02019	OBS\$Loop:	e ame		
0319 7E	02020	MOV	Α,Μ		;Get port number
031A B7	02021	ORA	A		Check if OOH (terminator)
031B C8	02022	RZ		1	Exit if at end of stream
031C 322503 031F 23	02023	STA	0BS\$Por	t i	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	7 ODO¢NI-uk¢Duk			
0322 7E	02028 02029	OBS\$Next\$Byte: MOV	A. M		Get paut buta
0322 75	02029	INX	8,11 H		;Get next byte ;HL -> next data byte (or port number)
L	02000	4130			The share data after the Pole Democry

	02031					
0324 D3	02032		DB	OUT		
	02033					
0325 00	02034		DB	0	;<- Set (	up in instruction above
0326 OD	02035		DCR	С		own on byte counter
0327 C222	203 02036		JINZ	0BS\$Next\$Byte		next data byte
032A C319	903 02037		JMP	OBS\$Loop		for next port number
	02038				•	
	02100					
	02101		CONST -	Console status		
	02102		001101			
	02103		This ro	utine checks bot	h the for	ced input pointer and
	02104					opriate input buffer.
	02105					e whether or not there
	02106			waiting.	o indicat	e whether of not chere
	02107		IS Uala	walling.		
	02108		Entry n	arameters: none.		
	02105			arameters: none.		
			<b>F</b>			
	02110		EXIT Pa	irameters		
	02111					
	02112			A = 000H if the		
	02113			A = OFFH if the	ere is dat	a waiting
	02114					
	02115					
	02116					;<=== BIOS entry point (standard)
	02117		*******			
032D 2A5	800 02118	1	LHLD	CB\$Console\$Inpu	at	Get redirection word
0330 116	400 02119		LXI	D,CB\$Device\$Tab		
0333 CD6	F06 02120		CALL	Select\$Device\$		;Get device table address
0336 C34			JMP	Get\$Input\$Statu		Get status from input device
	02122					; and return to caller
	02200					
	02201					
	02202		CONIN -	- console input		
	02203					
	02204		This rd	utine returns ++	e next ch	aracter for the console input
	0220		etrase	Depending on **	le ners su	tances, this can be a character
	02206		atrwall. fram +=	sepending on th	huffer -	r from a previously stored
	02207		studente	of characters to	be Mere	ed" into the input streamsfor
	02208					em initialization routines.
	02209					any previously stored character inject the current time and date
	02211		urast	CLARE ASSOCIATED	with a fu	nction key into the console
	02212		stream.	on system star	up, a str	ing of "SUBMIT STARTUP" is
	02213		Torced	into the consolu	e input st	ream to provide a mechanism.
	02214		New	/ Baum 6 mar =		form chickers physical decise
	02215					from whichever physical device
	02214				nsole inpu	t redirection word (see the
	02217		configu	aration block).		
	02218				•	
0339 00	02219		Delay#Ela	apsed: DB	0	;Flag used during function key
	02220					; processing to indicate that
	02221					; a predetermined delay has
	02222					; elapsed
	02223					
	02224		********	*===============		
	02225	5 CONIN:				;<=== BIOS entry point (standard)
	02220					
033A 2A8			ԼℍԼⅅ	CB\$Forced\$Inpu	t	;Get the forced input pointer
			MOV	A.M		Get the next character of input
033D 7F	02228	3				
033D 7E 033E B7	02228		ORA	Α		:Check if a null
033E B7	02228	,	ORA JZ			;Check if a null :Yes. no forced input
033E B7 033F CA4	02228 02229 703 02230	) )	JZ	CONIN\$No\$FI		;Yes, no forced input
033E B7 033F CA4 0342 23	02228 02229 703 02230 02231	9 ) 	JZ INX	CONIN\$No\$FI H		;Yes, no forced input ;Yes, update the pointer
033E B7 033F CA4 0342 23 0343 228	02228 02229 703 02230 02233 8D0F 02232	9 ) 1 2	JZ INX SHLD	CONIN\$No\$FI		;Yes, no forced input
033E B7 033F CA4 0342 23	02228 02229 1703 02230 02233 BD0F 02233 02233	9 0 1 2 3	JZ INX	CONIN\$No\$FI H		;Yes, no forced input ;Yes, update the pointer
033E B7 033F CA4 0342 23 0343 228	02228 02229 703 02230 02230 8D0F 02230 02233 02234	9  -  -	JZ INX SHLD RET	CONIN\$No\$FI H	t	;Yes, no forced input ;Yes, update the pointer ; and store it back
033E B7 033F CA4 0342 23 0343 228 0346 C9	02228 02229 02230 02230 02230 02230 02230 02230 02230 02230	9 2 3 5 5 5 CONINSN	JZ INX SHLD RET	CONIN\$No\$FI H CB\$Forced\$Inpu	t	;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input
033E B7 033F CA4 0342 23 0343 228 0346 C9 0347 2A5	02225 02225 02230 02233 02233 02233 02233 02234 02235 02235 02235	) 2 3 5 CONIN\$N 5	JZ INX SHLD RET IO\$FI LHLD	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu	t ut	;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word
033E B7 033F CA4 0342 23 0343 228 0344 C9 0347 2A5 034A 116	02228 02223 02233 02233 02233 02233 02233 02233 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235 02235	) 2 3 4 ; 5 CONIN\$N	JZ INX SHLD RET NO\$FI LHLD LXI	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Tal	t ut ple\$Addres	;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses
033E B7 033F CA4 0342 23 0343 228 0344 C9 0347 2A5 034A 116 034D CD6	02225 02225 1703 02233 02233 02233 02233 02235 1800 02235 1800 02235 1800 02235 1800 02235	)   2 3 5 5 CONIN\$N 3	JZ INX SHLD RET NO\$FI LHLD LXI CALL	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Inp Select\$Device\$Tal	t ole\$Addres Table	;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses ;Get device table address
033E B7 033F CA4 0342 23 0343 228 0344 C9 0347 2A5 034A 116	02228 02223 1703 02233 3B0F 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02232 02233 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02232 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 02233 0223 02233 02233 02233 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0223 0200 0223 0200 0223 0200000000	)   	JZ INX SHLD RET NO\$FI LHLD LXI	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Tal	t ole\$Addres Table	;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses
033E B7 033F CA4 0342 23 0343 228 0344 C9 0347 2A5 034A 116 034D CD6	02228 02223 02233 02233 02233 02234 02234 02234 02234 02234 02235 02234 02235 02234 02235 02234 02235 02240	) 2 3 5 CONIN\$ 5 7 3 9	JZ INX SHLD RET NO\$FI LHLD LXI CALL	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Inp Select\$Device\$Tal	t ble\$Addres Table acter	;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses ;Get device table address ;Get next character from input device
033E B7 033F CA4 0342 23 0343 228 0343 228 0346 C9 0347 2A5 034A 116 034D CD6 0350 CD9	0222 0222 0223 0223 0223 0223 0223 0223	2 2 5 CONIN\$1 5 7	JZ INX SHLD RET LHLD LXI CALL CALL	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Inp Select\$Device\$ Get\$Input\$Char:	t ble\$Addres Table acter ;Functio	;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses ;Get device table address ;Get device table address ;Get next character from input device n key processing
033E B7 033F CA4 0342 23 0343 228 0344 C9 0347 2A5 034A 116 034D CD6	0222 0223 0223 0223 0223 0223 0223 0223	) 2 3 4 5 CONINS 5 3 9 9	JZ INX SHLD RET NO\$FI LHLD LXI CALL	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Inp Select\$Device\$Tal	t ble\$Addres Table acter ;Functio zad	<pre>;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses ;Get device table address ;Get device table address ;Get next character from input device n key processing ;Check if first character of function</pre>
033E B7 033F CA4 0342 23 0343 228 0346 C9 0347 2A5 034A 116 0350 CD9 0353 FE1	0222 0222 0223 0223 0223 0223 0223 0223	) 2 3 5 5 6 7 7 9 9 9 9 9 1 2 2 3	JZ INX SHLD RET Ho\$FI LHLD LXI CALL CALL CALL	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Inp Select\$Device\$ Get\$Input\$Char:	t Je\$Addres Table acter ;Functio ead	<pre>;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses ;Get device table address ;Get device table address ;Get next character from input device n key processing ;Check if first character of function ; key sequence (normally escape)</pre>
033E B7 033F CA4 0342 23 0343 228 0343 228 0346 C9 0347 2A5 034A 116 034D CD6 0350 CD9	0222 0223 0223 0223 0223 0223 0223 0223	2 2 3 5 5 6 7 3 7 7 3 7 7 7 7 7 7 7 7 7 7 7 7 7 7	JZ INX SHLD RET LHLD LXI CALL CALL	CONIN\$No\$FI H CB\$Forced\$Inpu CB\$Console\$Inpu D,CB\$Device\$Inp Select\$Device\$ Get\$Input\$Char:	ut ble\$Addres Table acter ;Functio ead	<pre>;Yes, no forced input ;Yes, update the pointer ; and store it back ;No forced input ;Get redirection word ses ;Get device table address ;Get device table address ;Get next character from input device n key processing ;Check if first character of function</pre>

r				
0357 211000	02246	LXI	H,DT\$Function\$Delay	;Get delay time constant for
	02247		() Die die Cronebergy	
	02248			; delay while waiting for subsequent
	02249			; characters of function key sequence
035A 19	02250			; to arrive
		DAD	D	
035B 4E	02251	MOV	C, M	;Get delay value
0350 0600	02252	MVI	B, 0	:Make into word value
035E AF	02253	XRA	Α	;Indicate timer not yet out of time
035F 323903	02254	STA	CONIN\$Delay\$Elapsed	
0362 217803	02255	LXI	H. CONINSSetSDelaySFlag	psed ;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
1	02258			
f	02259			; transferred to specified address
	02260	CONIN\$Wait\$for\$	- 1 - · · ·	; after time interval has elapsed
0368 3A3903				;Wait here until delay has elapsed
	02261	LDA	CONIN\$Delay\$Elapsed	;Check flag set by watchdog routine
036B B7	02262	ORA	A	
036C CA6803	02263	JZ	CONIN\$Wait\$for\$Delay	
	02264			
	02265	CONIN\$Check\$for	*Function:	
036F 211900	02266	LXI	H,DT\$Character\$Count	Now check if the remaining characters;
	02267			; of the sequence have been input
0372 19	02268	DAD	D	
0373 7E	02269	MOV	Ã, M	;Get count of characters in buffer
0374 FE02	02270	CPI	Function\$Key\$Length -	i the court of characters th privet.
0376 D28103	02271	JNC	CONIN\$Check\$Function	
00/0 DE0103	02272	CHNC.	CONTUNECRALING	Enough characters in buffer for
0070 51			DOU	; 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 fol	lowing routine is calle	d by the watchdog routine
	02280	; when th	e specified delay has e	lapsed.
	02281			
	02282	CONIN\$Set\$Delay	SElapsed:	
037B 3EFF	02283	MVI	A, OFFH	;Indicate watchdog timer out of time
037D 323903	02284	STA	CONIN\$Delay\$Elapsed	findicate watchbog times out of time
0380 C9	02285	RET	contracta, actupata	Return to watchdog routine
	02286	:		Therdra to watchdog Fodring
	02287	-		
	02288	, CONIN\$Check\$Fur	ation.	
0381 211700	02289	LXI		- Course - Albert
0384 19	02290	DAD	H,DT\$Get\$Offset D	Save the current "get pointer"
			-	; in the buffer
0385 7E	02291	MOV	A, M	;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)
038A CDF007	02295	CALL	Get\$Address\$in\$Buffer	; character in the sequence
038D 46	02296	MOV	B,M	;Get the second character
	02297			
	02298	IF	Three\$Character\$Functi	on
038E C5	02299	PUSH	В	;Save for later use
038F 211700	02300	LXI	H, DT\$Get\$Offset	Retrieve the third character
0392 CDF007	02301	CALL	Get\$Address\$in\$Buffer	
0395 C1	02302	POP	R	Recover second character
0396 4E	02303	MOV	Č.M	Now BC = Char 2, Char 3
0070 TE	02303	ENDIF	0,11	now be - char 2, char 3
		ENUIF		
0007 55	02305	<b>B</b> 11011	P	· Only along the balance of the
0397 D5	02306	PUSH		Save device table pointer
0398 218000	02307	LXI	H,CB\$Function\$Key\$Tabl	
	02308			Get pointer to function key table
	02309			<pre>; in configuration block</pre>
039B 111300	02310	LXI	B,CB\$Function\$Key\$Entr	y\$Size ;Get entry size ready for loop
	02311	CONIN\$Next\$Func		
039E 19	02312	DAD	D	;Move to next (or first) entry
039F 7E	02313	MOV	A, M	;Get second character of sequence
03A0 B7	02314	ORA	A	Check if end of function key table
03A1 CAC203	02315	JZ	CONIN\$Not\$Function	;Yes it is not a function key
03A4 B8	02316	CMP	B	Compare second characters
03A5 C29E03	02317	JNZ	CONIN\$Next\$Function	;No match, so try next entry in table
VORU 627E03	02318	UNZ	CONTINUES CALCIIC (100	THE MERCHY SE ILY HEAL BULLY IN LEDIE
		**		
	02319	IF	Three\$Character\$Functi H	
0040 00		INX	n	;HL -> third character
03A8 23	02320		A M	
03A9 7E	02321	MOV	A, M	;Get third character of sequence
			А, М Н	;Get third character of sequence ;Simplify logic for 2 & 3 char. seq.

Figure 8-10. (Continued)

03AB BP       02323       CMP       C       ICOMNENTEFFUnction       ICOMPart of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of the methan of					·	
03AC C29E03       02324       JAX       CONINNExtFunction       :No match, so try next entry in table         03AF 23       02327       INX       H       rextra decrement         03BF 23       02329       INX       H       :HL -> first character of substitute         03BE 22000F       0331       02332       :HL       :HL -> first character of substitute         02330       02332       :HL       CBFForced#Input       :Hake the CONN Froutine inject them, intention into the input         02333       02333       :Stream       :Stream       :Stream         02333       :O2337       :Stream       :Stream       :Stream         02334       :O2337       :D234       :Stream       :Stream       :Stream         02345       :D2342       :LI       H.DT6Character#Scout       :Domdate the character is newed         0355       :D2347       :DD0       :Stream       :Stream       :Stream         0356       :D2347       :DD0       :Stream       :Stream <td>03AB 89</td> <td>02323</td> <td></td> <td></td> <td></td> <td></td>	03AB 89	02323				
02327       ENDIF         02327       ENDIF         02328       INX         02329       INX         02320       INX         02321       SHLD         02322       INX         02323       SHLD         02331       SHLD         02332       INX         02333       SHLD         02334       Interview         02335       Interview         02334       Interview         02335       Interview         02335       Interview         02335       Interview         02335       Interview         02335       Interview         02335       Interview         02336       Interview         02337       Interview         02338       Interview         02340       Interview         02341       Interview         02342       LXI         H       Interview         02342       LXI         H       Interview         02344       Interview         02345       Interview         02346       Interview         02347	03AC C29E03		JNZ	CONIN\$Nex1	<pre>\$Function</pre>	
0380 23     03237     ENDIF       0380 23     03238     INX H     H     HL -> first character of substitute       0381 22800F     03331     SHLD     CD#Forced#Input     Fatra for characters (00-byte term.)       0383 23800F     03333     SHLD     CD#Forced#Input     Fatra for characters (00-byte term.)       0383 61     03335     POP     Now that a function sequence has been i balanced prior to return       0385 61     02336     POP     PSU     Fort the depite of the character for other       0385 61     02334     POP     PSU     Fort the depite of the character for other       0385 72     02344     POP     PSU     Fort the depite of the character for other       0385 72     02344     POP     PSU     Fort the depite of the character for other       0386 72     02346     POP     PSU     Fort the depite of the character for other       0387 72     02346     POP     Fort the character for other     Fort the depite of the character for other       0387 72     02346     POP     Fort the character for other     Fort the character for other       0387 72     02346     POP     Fort the character for other     Fort the fort for fort is a stare at the character for other       0387 72     02346     POP     Fort the fort is a stare at the fort fort is a stare at the	03AF 23	02325	INX	н		;When match found, compensate for
0380 23       03229       INK       H       IHL -> first character of substitute         0381 22900F       03230       SHLD       CBSForced*Input       Intering CONSTRUCTS (Orbit Terms)         0381 22900F       03333       SHLD       CBSForced*Input       Intering into the input         0383 2330       SHLD       CBSForced*Input       Intering into the input         0383 2330       SHLD       CBSForced*Input       Intering into the input         0383 2330       CONSTRUCT       Substitute string into the input         0383 71100       CONSTRUCT       Substitute string into the input         0385 F1       CONSTRUCT       Substitute string into the input         0385 F1       CONSTRUCT       Substitute string into the input         0385 F1       CONSTRUCT       Substitute       Substitute string into the input         0385 F1       CONSTRUCT       Substitute       Substitute         0386 F7       CONSTRUCT       Substitute       Substitute         0386 F7       CONSTRUCT       Substitute       Substitute         0387 C10000000       CONTIN       Interacter input       Substitute         0386 F7       CONTIN       Interacter       Substitute         0387 C100000000       CONTIN       Interacter<						; extra decrement
0380 23       03230       INX       H       IHL >> first characters of substitute string or characters (0x)brits term in substitute string into the input string or characters (0x)brits term in substitute string into the input string or characters (0x)brits term in substitute string into the input string or characters (0x)brits term 02335         0381 D1       02336       POP       D       Field term is balanced prior to the input string or character count is balanced prior to the input string or character count is balanced prior to sequence has been is balanced prior to the term plume the "get" offset value         0385 F1       02337       POP       PSW       Dum the "get" offset value         0387 F1       02334       DAD       D       to reflect the character count is to reflect the character semoved is from the buffer         0387 F2       02344       MOV       A,H       The fore display term has already is the count the sequence is the forced input there as incards is the forced input there as incards is the forced input there as incards is the constrainer as incards is the is the forced input to as that is the character (s) presume as not lost.         0387 F27       02347       POP       D       POP         0386 F1       02357       POP       D       POP         0386 F27       02350       DAD       POP       D         0387 F27       02349       JW       POP       D         0386 F27       02360       DAD		02327	ENDIF			
0381 2280F       02331       SkLD       CB#Forced#Input       i string of characters (00-byte term.)         0381 2280F       02331       SkLD       CB#Forced#Input       i string into the input         02332       02333       istrage       istrage         02333       02334       Now that a function sequence has been i lentified, the stack must be         02336       02337       POP       D         02347       POP       PSW       iDown the function sequence lead char.         0385 F1       02340       POP       PSW       iDown the function sequence lead char.         0387 F2       02344       LXI       H_DT#Character#Count       iDowndate the character count         0388 F1       02347       MOV       A.M       iDowndate the character count         0388 F2       02345       MOV       A.M       iDowndate the character count         0388 F2       02347       MOV       A.M       iDowndate the character count         0388 F2       02347       MOV       A.M       iDowndate the character count         0388 F2       02347       MOV       N.A       iDowndate the count         0388 F2       02347       MOV       N.A       iDowndate the count         0388 F1       02350		02328				
OSBI 22800F       OZ332 OZ333 OZ334 OZ335 OZ335 OZ335 OZ335 OZ335 OZ335 OZ335 OZ335 OZ335 OZ336 OZ336 OZ336 OZ337 OZ336 OZ337 OZ336 OZ337 OZ336 OZ337 OZ336 OZ337 OZ336 OZ337 OZ336 OZ337 OZ336 OZ337 OZ341 OZ341 OZ341 OZ341 OZ341 OZ342 OZ342 OZ342 OZ342 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ344 OZ346 OZ346 OZ346 OZ346 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ347 OZ	0380 23	02329	INX	н		
02332 02334 02334 02334 02334 02334 02334 02334 02335 02334 02335 02336 02337 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02384 02385 02385 02385 02384 02385 02384 02385 02384 02385 02384 02384 02387 02384 02387 02384 02385 02384 02385 02384 02385 02384 02385 02384 02385 02384 02385 02384 02385 02385 02384 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 02385 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		02330				
02333       istream         0234       indentified, the tack must be account in the tack must be account in the tack must be account in the tack must be account in the account in the tack must be account in the account of the state must be account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account in the account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account account ac	03B1 228D0F	02331	SHLD	CB\$Forced	Finput	
02235       invertified, the stack must be         02336       identified, the stack must be         02337       POP       DB         0385 F1       02337       POP         0385 F1       02337       POP         0385 F1       02337       POP         0385 F1       02340       POP         02341       02341       iDum the "get" offset value         0385 F1       02344       DAD       jount the stack must be         0385 F1       02344       DAD       jount the function sequence lead char.         0385 F2       02344       DAD       jount the stack must be         0385 F1       02344       DAD       jount the character count         0385 F2       02345       MOV       A.H       joet the count         0385 F1       02344       JMP       CONIN       ite forced insut character stall and the count         0385 F1       02345       JMP       CONIN       ite function key sequence         0385 F1       02357       FOP       D       ite character stall apointer         0362 010       02357       FOP       B       ifecover previous value so that         0362 11       02357       FOP       B       ifecover previous value		02332				
02335       iNow that a function sequence has been         02387       POP       D         0385 F1       02337       POP         02387       POP       PSW         0385 F1       02337       POP         0237       POP       PSW         0385 F1       02341       LXI         0387       POP       PSW         0388       POP       PSW         0388       POP       PSW         0388       POP       PSW         0387       POP       PSW         0388       POP       PSW         0387       POP       PSW         03250       POP       PSW         02351       POP       PSW <td></td> <td>02333</td> <td></td> <td></td> <td></td> <td>; stream</td>		02333				; stream
02336       identified. the stack must be         0284 D1       02337       POP       DA         0285 F1       02330       POP       PSN       Jump the device table pointer         0285 F1       02340       POP       PSN       Jump the device table pointer         0287 21100       02341       LXI       H.DT\$Character\$Count       JDundate the character count         0287 21100       02345       DAD       D       , to reflect the character sensore         0288 F2       02346       MOV       A.H       , to reflect the character sensore         0385 F2       02346       MOV       A.H       , the character count         0386 F7       02344       MOV       A.H       , the forced input character sensore         0386 F7       02346       MUF       Forcetont       ; the character of the sequence         0386 F7       02347       MOV       A.H       ; the character of the sequence         0387 2100       02346       JMF       interest of coopaize a function key sequence         0387 100       02345       j the character(s) presumed to be part of         02351       CONINNIC\$Function:       i the character store       isterest of sease are not lost.         02352       CONIN\$Not \$Function:       <		02334				
02337       is balanced prior to return         0284 D1       02338       POP       Soft Hermitian Soft Hermitian Software table pointer         0285 F1       02339       POP       PSW       JDump the "get" offset value         0287 F1       02341       DAD       Damp the "get" offset value         0287 F2       02342       LXI       H,DTSCharacter\$Count       jDowndate the characters removed         0288 7F       02345       MOV       A,M       is to reflect the characters removed         0388 7E       02344       mov       A,M       is to reflect the characters removed         0388 7E       02345       MOV       A,M       is the count       count         0388 7E       02344       UPP CONIN       is the forced input characters       count         0385 F1       02350       ODIN       is the forced input character is         02351       OZ350       CONINNENCISFUNCTION       is the forced input character is         02354       JDF       CONIN       is the forced input character is         02355       CONINNENCISFUNCTION       is the function sequence are not lost.         02356       OZ357       POP       is Recover previous matter is offset         02357       POP       POP       is Recover l		02335				
0385 F1       0235 F1       0235 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1       0236 F1		02336				
0385 F1       0233       029       PDP       PSW       ;Dump the "uet" offset value         0385 F1       02340       02341       ;Dump the "unction sequence laad char.         0381 F1       02342       LXI       H,DT\$Character\$Count       ;Boundate the character count         0381 71       02344       DD       ;Torm the buffer character's removed         0385 72       02344       DU       ,It is the count         0385 72       02345       MOV       A.H       ;Boundate the character has already         0385 72       02344       SUI       Function%reyklength -1;       It the lead character has already         0385 72       02344       JUN       NA       ;Boundate the character has already         0385 72       02344       JUN       Recover durits to recognize a function key sequence         0385 72       02344       it the character(s) preside to be part of       ;2235         02340       JUNNNOT\$Function       ;1the character(s) preside to be part of       ;2235         02350       CONINNONO*Function       ;1th character(s) preside to be part of         02351       CONINNONO*Function       ;Recover previous value so that         02352       POP       PSW       ;Recover previous value so that         02351       <		02337				
0386 F1       02340       POP       PSW       ; Dump the function sequence laad char.         0387 10       02342       LX1       H, DT%Character%Count       ; Doundate the character count         0388 19       02343       DAD       D       ; Form the builder         0388 77       02344       MOV       A, M       ; Form the builder         0385 0502       02345       SUI       Function%Key%Length -11       (the lead character has already         0385 0502       02346       JHP       CONIN       ; Been deducetd)         0385 0502       02348       JHP       CONIN       ; Been deducetd)         0385 0502       02348       JHP       CONIN       ; Been deducetd)         0385 0502       02348       JHP       CONIN       ; Been deducetd)         0385 10       02351       ; Fattempts to recognize a function key sequence         02352       ; The function sequence are not lost.       ; Control (Control (						;Get the device table pointer
0387 21190       02341       UI       H,DT%Character%Count       ; Bowndate the character count         0388 719       02343       DAD       ; to reflect the characters removed         0380 72       02344       DAU       ; to reflect the characters removed         0380 72       02344       MOV       A.M       ; to reflect the character sequence         0380 72       02344       SUI       Function%Key%Length -1       ithe lead dracter has already         0380 72       02346       JMP       CONIN       r, been deducted)         0387 73       MOV       N.A       ; here deducted)         0387 74       MOV       N.A       ; here failed. The "get" offset pointer must bi (2335         0387 71       02387       NO       ; here failed. The "get" offset sequence         0323 71700       02380       LAI       i here failed. The "get" offset as it usa stfer         0325 71						
0387 211900       02342       LX1       H,DT%Character%Count       ; Downdate the character actor count         0387 19       02343       DA       D       ; from the builder         0387 20       02345       MOV       A,M       ; Get the count         0380 7E       02345       MOV       A,M       ; Get the count         0380 7E       02346       SUI       Function%Key%Length -1;       ; the lead character has already         0380 7E       02346       UHP       CONIN       ; the forced input characters         02357       CONINNNOt%Function:       ; the forced input character must bi       ; restored to its previous value so that         02356       02357       POP       D       ; restored to its previous value so that         02356       02357       POP       D       ; restored to its previous value so that         02357       POP       D       ; restored to its previous value so that         02356       02357       POP       D       ; restored to its previous value so that         02357       POP       D       ; restored to its previous value so that         02356       02357       POP       PS       ; restored to its previous value so that         03262       10       02354       ; restor	03B6 F1		POP	PSW		;Dump the function sequence lead char.
038A 19       02343       DAD       p to reflect the characters removed provide the count         038D 0502       02345       MOV       A,H       ; Get the count         038D 0502       02344       SUI       Function%Key%Length -1; (the lead character has already reducted)         038E 0502       02344       SUI       Function%Key%Length -1; (the lead character has already reducted)         038E 0502       02347       MOV       H.A       ; been deducted)         038E 0502       02347       MOV       H.A       ; been deducted)         038E 0502       02347       MOV       H.A       ; been deducted)         038E 0502       02350       CONINN       ; thempt to recognize a function key sequence is have failed. The "get" offset pointer must be 02353         02355       FOP       D       ; restored to its previous value so that         02356       FOP       D       ; restored to its previous value so that         02357       POP       D       ; Recover device table pointer         02358       DAD       D       H.D.* "get" offset in table         0357       02361       MOV       M.A       ; Recover device table pointer         02357       D235       LIN       H.D.TSGet%Offset       ; Ite lead character						
0384       ; from the buffer         0385 7E       02344       SUI       Function%(s)Length -1; (the lead character has already         0386 77       02344       SUI       Function%(s)Length -1; (the lead character has already         0386 77       02344       UMP       CONIN       :Return to CONIN processing to get         02385       CONIN®Not%Function:       :Attempts to recomize a function key sequence       ; have failed. The "get" offset pointer must bu         02335       CONIN®Not%Function:       :Attempts to recomize a function key sequence       ; the character(s) presumed to be part of         02335       : the character(s) presumed to be part of       ; the function sequence are not lost.         02335       : the function sequence are not lost.       : the function sequence are not lost.         02356       02357       POP       D       :Recover device table pointer         02362 77       02364       DAD       rHL -> "get" offset in table         03263 77       02364       DAD       rRecover device table pointer         03264       : the present of the second device in table       : the character         03265       : Q2364       : the second device in table       : the character         03264       : This routine outputs data characters to the console device(s).       : the character in table					acter\$Count	;Downdate the character count
0388 76       02345       MOV       A,M       ; Det the count         0380 D502       02344       SUI       Function%eysLengh -1;       the lead character has already         0386 C002       02344       MOV       M.A       ; been deducted)         0387 D50300       02344       WDV       return to CONIN processing to get         02331       : Attempts to recognize a function key sequence         02335       : have failed. The "set" offset pointer must bit         02335       : pastored to its previous value so that         02335       : pastored to its previous value so that         02335       : pastored to its previous "get" offset         02336       : pastored to its previous "get" offset         02335       : pastored to its previous "get" offset         02336       : pit function sequence are not lost.         02337       : pit its its it able         03362       : pit its its its as it was after         03361       : pit its cover lead character was detected         03242       : pit its cover lead character to the user         03362       : pit its cover lead character to the user         03362       : pit its cover lead character to the user         03251       : Console output         03262       : its cover lead	03BA 19		DAD	D		
038C D&02 038E 03A6 038E 77 034 038E 77 034 038E 77 034 038E 77 034 038E 77 034 035 035 035 035 035 035 035 035 035 035		02344				
038F 7       02347       MOV       M.A.       is been deducted)         038F C33A03       02348       UMP       CONIN       is the forced input characters         02351       02353       CONINNENGT\$Function:       is the forced input characters       is the forced input characters         02352       is have failed. The "get" offset pointer must bised       is previous value so that       is the forced input character (s) previous value so that         02353       is the forced input character (s) previous value so that       is previous value so that         02354       is the forced input character (s) previous value so that       is previous value so that         02354       is the forced input character (s) previous value so that       is the forced input character (s) previous value so that         02355       CONINSTOR       prescover device table pointer       is the forced input character (s) previous value so that         03364       11700       02358       POP       PSW       if Recover previous "get" offset is the as after         0337       02360       DAD       D       prever lead character (s) previous value so the user         03257       02361       Console cutput       if Recover lead character       if the user         0326       7       02361       RET       if Recover lead character       if the user	03BB 7E			Α,Μ		;Get the count
038F 77       02347       MOV       M.A       ; been deducted)         038F C33A03       02349       JHP CONIN       ; Return to CONIN processing to get         02351       ; have failed. The "get" offset pointer must bi         02353       ; have failed. The "get" offset pointer must bi         02354       ; the forced input characters         02355       ; the function sequence are not lost.         02356       02357       POP         02357       POP       percover device table pointer         03352       02358       POP         03354       102358       POP         03357       POP       percover device table pointer         03354       102358       POP         03357       POP       percover device table pointer         03354       102358       POP         03354       102358       POP         03357       102361       DAD         03357       102363       POP         03357       102364       POP         03357       102358       POP         0335       Console output       rest "rest" and character         03250       1       This routine outputs data characters to the console device(s).				Function\$	(ey\$Length −1	; (the lead character has already
02349       i the forced input characters         02351       cONINENOT\$Function:         02351       ;Attempts to recognize a function key sequence         02352       ; have failed. The "get" offset pointer must bi         02353       ; the character(s) presumed to be part of         02354       ; the character(s) presumed to be part of         02355       ; the function sequence are not lost.         02356       ; the function sequence are not lost.         02357       POP       prestored to its previous "get" offset         03257       02357       POP         03258       POP       prestore device table pointer         03267       02359       LXI       H.DT%Get#Offset         03268       POP       prestore previous "get" offset         03261       MOV       M.A       ;Recover previous "get" offset         03261       prestore feed character       it use distore feed character         03261       MOV       M.A       ;Recover lead character         03262       prestore feed console output       ;Recover lead character         03263       This routine outputs data characters to the console device(s).         02506       ;A       primitive "state-machine" is used to step through escape         02505				M,A		; been deducted)
02350       CONINNENCESFUNCTION:         02351       ; Attempts to recognize a function key sequence (02352         02352       ; have failed. The "get" offset pointer must built (02354         022351       ; the character(s) presumed to be part of (02355         022352       ; the character(s) presumed to be part of (02355         022351       ; the character(s) presumed to be part of (02355         02352       pOP         02353       pOP         02354       ; the character(s) presumed to be part of (02356         0357       pO350         0357       pO350         0357       pO351         0357       pO351         0357       pO3361         pO363       pO7         pO364       RET         pO365       pO7         pO366       pO7         pO367       post post post post post post post post	03BF C33A03		JMP	CONIN		
02351       ;Attempts to recognize a function key sequence         02352       ; heve failed. The "get" offset pointer must bi         02353       ; restored to its previous value so that         02354       ; the character(s) presumed to be part of         02355       ; the function sequence are not lost.         02356       ; the function sequence are not lost.         02357       POP       pervious "get" offset         02358       POP       PSW         02357       POP       pervious "get" offset         02357       POP       PSW         03508       POP       PSW         03507       PO2361       MOV         03508       POP       PSW         03504       POP       PSW         03505       ;       Console output         03506       ;       Console output         02501       ;       Console output         02502       ;       Console output         02503       This routine outputs data characters to the console device(s).         02504       II talso "traps" escape sequences being output to the console.         02505       triggering specific actions according to the sequences.         02506       A primitite "state-machine" is used to step through						; the forced input characters
02352       ; have failed. The "get" offset pointer must bit         02353       ; the character(s) presumed to be part of         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       POP       D         03C2 DI       02357       POP         03C3 F1       02358       POP         03C3 719       02360       DAD       D         03C3 719       02363       POP       PSW         03C3 719       02363       POP       FSW         03C3 719       02363       POP       FSW         03C4 21700       02364       RET       ; the lead character usa detected         03C5 71       02363       POP       FSW       ; fRecover lead character         03C4 72       02364       RET       ; fRecover lead character       is used to sequences.         02350       ;       Console output       02505       ;       is used to sequences.       is used to sequences.         02302 ;       :       This routine outputs data characters to the console device(s).       02506       is used to sequences.       is used to sequences.       :			CONIN\$Not\$Funct	ion:		
02352       ; have failed. The "get" offset pointer must bit         02353       ; herefailed. The "get" offset pointer must bit         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       POP       presume are not lost.         03C2 DI       02357       POP         03C3 F1       02358       POP         03C4 21700       02359       LXI       H.DTSGEteOffset         03C7 19       02360       DAD       D       HL<-> "get" offset in table         03C7 19       02363       POP       PSW       ; frecover lead character         03C8 77       02364       RET       ; frecover lead character         03C4 210700       02364       RET       ; frecover lead character         03C5 7       02363       POP       PSW       ; frecover lead character         03C4 2007; #       Console output       02505;       ;       02505;         02300; #       Console output       02506;       if talse "machine" is used to see wheres.       02506;         02505; #       Trisgering specific actions according to the sequences.       02506;       02506;       02506;					;Attem	pts to recognize a function key sequence
02354       ; the character(s) presumed to be part of         02355       ; the function sequence are not lost.         02356       ; the function sequence are not lost.         02357       POP       p         0363 F1       02358       POP       ; the cover device table pointer         0374       21700       02359       LXI       H,DT%GE40ffset         0375       02360       DAD       D       ; the lead character is as it uss after         0376       77       02361       MOV       M.A       ; Reset" offset is it uss after         0376       77       02362       POP       PSW       ; Recover lead character         0376       77       02363       POP       PSW       ; Recover lead character         0376       7       02364       RET       ; Return the lead character to the user         02360       ;       Console output       ;       02505       ;         02360       ;       This routine outputs data characters to the console device(s).       ;         02505       ;       Triggering secific actions according to the sequences.       ;         02506       ;       A primitive "state-machine" is used to step through escare       ;         02507       ;						
02355; the function sequence are not lost.0362 D102357POPp0363 F102357POPp0363 F102357POPPSW0367 F102360DADD0367 F102361MOVM,A0368 7702361MOVM,A0368 7702361MOVM,A0368 7702361MOVM,A0368 7702363POPPSW0364 7102363POPPSW0364 7202364RET0365 1;Console output0366 20;;0367 1902363;0368 7002364RET0364 71;Console output0365 1;Console output0366 20;;0370 1;Console output0366 3;;This routine outputs data characters to the console device(s).0370 1;Console output02505;0350 2;;tragering specific actions according to the sequences.0250 2;;tragering specific actions according to the sequence.0250 3;th addition to outputing the next character to all of the0250 4;;addition to output to the selected device has not been0250 5;tragering specific actions according to the sequence.0250 7;sequence recognition.0250 1;to the specified length of message has been output; an Etx0251 2;it should be.0251 3;Once the character has bee		02353				
0235602357POPD;Recover device table pointer03C2 DI02358POPPSW;Recover previous "get" offset03C4 2170002359LXIH,DT\$Get\$Offset;H03C7 1902360DADD;Recover jeet" offset in table03C8 7702361MOVM,A;Recover jeet" offset as it was after03C9 F102362POPPSW;Recover jeet" offset as it was after03C9 F102363POPPSW;Recover jeet" offset as it was after03C4 2170002364RET;Return the lead character to the user02501;Return the lead characteris used to the console device(s).02502 ;This routine outputs data characters to the console device(s).02503 ;#it is geting specific actions according to the sequences.02504 ;It also "traps" 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 ;li checks to see that output to the sequence here double the double the double the sequence.02506 ;it checks to see that output to the sequence here double the double the double that the device is a the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the double the doub						
03C2 D102357POPD;Recover device table pointer03C3 F102358POPPSW;Recover previous "get" offset03C7 1902360DADD;HL -> "get" offset in table03C7 1902363MOVM,A;Reset "get" offset as it was after03C9 F102363POPPSW;Recover lead character03C4 C902364RET;Recover lead character02365 ;02364RET;Recover lead character02360 ;;02364reset "get" offset in table02360 ;;02364RET02360 ;;02364reset "get" offset02362 ;02364reset "get" offset02363 ;This routine outputs data characters to the console device(s).02364 ;I also "traps" escape sequences being output to the console,02365 ;tragger gespecific actions according to the sequences.02366 ;A primitive "state-machine" is used to step through escape02506 ;I naddition to outputing the next character to all of the02507 ;suspende by XON/XOFF protocol, and that DTR is high if02510 ;it checks to see that output to the selected device has not been02511 ;suspende by XON/XOFF protocol, and that BTR is high if02512 ;it character is output and the device is flagged as being suspended.02513 ;Once the character has been output, an Etx02514 ;and the specified length of message has been output, an Etx02515 ;convoltshracter;02516 ;CONOUT\$Character; <tr< td=""><td></td><td></td><td></td><td></td><td>; the</td><td>function sequence are not lost.</td></tr<>					; the	function sequence are not lost.
<ul> <li>Carca Fi 02358 POP PSW ;Recover previous "get" offset</li> <li>Carca 21700 02359 LXI H,DT\$Get\$Offset</li> <li>Carca 21700 02359 LXI H,DT\$Get\$Offset</li> <li>Carca 2170 02360 DAD D ;HL-&gt; "get" offset in table</li> <li>Carca 2170 02361 MOV M,A ;Reset" get" offset as it was after ; the lead character was detected</li> <li>Recover lead character was detected</li> <li>Carca 2265 ; return the lead character to the user</li> <li>Carca 2265 ; return the lead character to the user</li> <li>Carca 2362 return the lead character to the user</li> <li>Carca 2363 POP PSW ;Recover lead character to the user</li> <li>Carca 2364 RET ; Return the lead character to the user</li> <li>Carca 2365 ; return 1 also "traps" escape sequences being output to the console,</li> <li>Carca 2367 ; A primitive "state-machine" is used to step through escape</li> <li>Carca 2366 ; devices currently selected in the console output redirection word,</li> <li>Carca 2360 ; devices currently selected in the console output redirection word,</li> <li>Carca 1 ; it should be.</li> <li>Carca 1 ; once the character to be output, an Etx</li> <li>Carca 1 ; once the character to be output, an Etx</li> <li>Carca 1 ; once the character to be output, an Etx</li> <li>Carca 1 ; convourstor as variables</li> <li>Carca 2 ; convourstor secon: DW Convourstormal</li> <li>Carca 1 ; convourstor as variables</li> <li>Carca 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ; convourstor 2 ;</li></ul>						
03C5 211700       02359       LXI       H.DT\$Get\$Offset         03C7 19       02360       DAD       D       H.L> "get" offset in table         03C7 19       02361       MOV       M.A       ; The lead character was detected         03C7 19       02362       POP       PSW       ; The lead character was detected         03C4 C9       02363       POP       PSW       ; Recover lead character to the user         02264       RET       ; Return the lead character to the user       ; Return the lead character to the user         02260       ; #       02503 ;       This routine outputs data characters to the console device(s).         02250       ; #       02506 ;       I talso "traps" escape sequences being output to the console.         02250       ;       Tring\$ring specific actions according to the sequences.       02506 ;         02506 ;       I naddition to outputting the next character to all of the       02507 ;       sequence "recognition.         02506 ;       In addition to outputting the next character to all of the       02510 ;       it checks to see that output to the selected device has not been         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         02511 ;       Once the character has been						
03C7 19 03C8 77 02361 03C8 77 02362 03C9 F1 02363 03C9 F1 02363 03C4 C9 03C4 C9 02364 03C4 C9 02364 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 02500 1 0 02500 1 0 02500 1 0 0 0 0 0 0 0 0 0 0 0 0 0						;Recover previous "get" offset
03CB 7702361MOVM.A: FRest "get" offset as it was after (3 character was detected ; Fte lead character was detected ; Recover lead character was detected (3 character 10 23 character) (3 character)03CA C902364RET; Return the lead character was detected (3 character) (3 character)03CA C902364RET; Return the lead character to the user (3 character) (3 character)03CA C902364RET; Return the lead character to the user (3 character) (3 character)03CA C902364RET; Return the lead character to the user (3 character)02300; #02501; Console output (3 character)02502;Console output (3 character)02503; This routine outputs data characters to the console device(s). (3 character)02504; It also "traps" escape sequences being output to the console, (3 character)02505; triggering specific actions according to the sequences. (3 character)02506; A primitive "state-machine" is used to step through escape (3 character)02507; sequence recognition. (3 character)02508; In addition to outputing the next character to all of the (3 character)02509; devices currently selected in the console device has not been (3 character)02511; suspended by XOM/XOFF protocol, and that DIR is high if (3 character)02512; inth sould be. (3 character)02513; Once the character has been output, if ETX/ACK protocol is in use, (3 character)02514; and the specific length o					ffset	
02362       i the lead character was detected         03C9 F1       02364       POP       FRECover lead character         03C4 C9       02364       RET       ;Return the lead character to the user         02360       ;#       02300       ;#         02501       ; Console output       02502       ;         02503       ; This routine outputs data characters to the console device(s).       02503         02505       ; Triggering specific actions according to the sequences.       02507         02507       ; sequence recognition.       02509         02508       ; In addition to outputing the next character to all of the         02509       ; devices currently selected in the console output redirection word,         02511       ; suspended by XON/XOFF protocol, and that DTR is high if         02512       ; it should be.       02513         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       ; CONOUT storage variables         02520       ;       02519         02521       ; CONOUT storage variables         02522       ;       02520         02520       ;       ; code that will process of the piece of <td></td> <td></td> <td></td> <td></td> <td></td> <td>;HL -&gt; "get" offset in table</td>						;HL -> "get" offset in table
03C9 F1 02363 POP PSW ;Recover lead character ;Return the lead character 03CA C9 02363 ; 02500 ;# 02500 ; 02500 ; 02502 ; 02503 ; This routine outputs data characters to the console device(s). 02502 ; 02503 ; This routine outputs data characters to the console device(s). 02504 ; 11 also "traps" escape sequences being output to the console, 02505 ; 02506 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 ; In addition to outputing the next character to all of the 02509 ; devices currently selected in the console device(s) and that DTR is high if 02510 ; it suspended by XON/XOFF protocol, and that DTR is high if 02513 ; 02513 ; 02513 ; 02514 ; 02515 ; 02515 ; 02515 ; 02515 ; 02517 ; 02517 ; 02517 ; 02517 ; 0251 ; 02519 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 025 ; 0252 ; 0252 ; 0252 ; 0252 ; 0252 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 025 ; 02 ; 02	0308 77		MOV	M, A		
03CA C9       02364       RET       jReturn the lead character to the user         02300       ;#         02301       :       Console output         02302       ;#         02303       :       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.         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       :       supended by XON/XOFF protocol, and that DTR is high if         02512       :       it should be.         02513       :       character is output and the device is flagged as being supended.         02517       :       character is output and the device is flagged as being supended.         02518       :       C = character to be output         02520       :       C2520       :         03CB 00       02523       CONOUT\$Character:       DB       ;S						
02365 ; 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 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 ; 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 02526 ; 02527 ; character. The default case is 02528 ; 02529 ; 03CE 0000 02530 CONOUT\$String\$Pointer: DW 0 ;This points to a string (normally 02530 ; is being preset by characters from				PSW		
02500 ;# 02501 ; Console output 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 escape 02507 ; sequence recognition. 02508 ; In 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 02515 ; character is output and the device is flagged as being suspended. 02518 ; 02518 ; 02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be output 02524 ; 03CC DB03 02523 CONOUT\$Processor: DW CONOUT\$Normal 02526 ; 03CE 000 02530 CONOUT\$Processor: DW CONOUT\$Normal 02527 ; code that will process the next 02528 ; 03CE 0000 02530 CONOUT\$String\$Pointer: DW 0 ;This points to a string (normally ; in the configuration block) that 02531 ; is being preset by character from	03CA C9		RET			Return the lead character to the user
02501Console output02502This routine outputs data characters to the console device(s).02503It also "traps" escape sequences being output to the console,02504It also "traps" escape sequences being output to the console,02505:02506:02507:02508:02508:02509:02509:02509:02509:02509:02509:02509:02509:02510:02510:02511:suspended by XON/XOFF protocol, and that DTR is high if02512:02513:02514:02515:02515:02517:02518:02519:02520:02521:02522:0252202523:02531:0254:02513:02514:02515:02515:02520:02521:02522025220252302600:025240302:0302:0302:0302:0303:02525:0304:02526:0305:02527<						
02502;02503;02504;11also "traps" escape sequences being output to the console,02505;02506;02507;02508;02509;02509;02509;02509;02509;02509;02509;02509;02509;02509;02509;02509;02510;it checks to see that output to the selected device has not been02512;it should be.02513;02514;and the specified length of message has been output, and the selected device is flagged as being suspended.02517;02518;02519;02519;02521;03CB 000252202522;03CB 000252503CB 000252602527;03CB 000252602527;03CB 000252603CE 00000252703CE 00000253002520;03CE 00000253002520;03CE 00000253002520;03CE 00000253002530CONOUT\$String\$Pointer: DW03CE 00000253002530CONOUT\$String\$Pointer: DW03CE 000002530						
02503This routine outputs data characters to the console device(s).02504It also "traps" escape sequences being output to the console,02505triggering specific actions according to the sequences.02506A primitive "state-machine" is used to step through escape02507sequence recognition.02508In addition to outputting the next character to all of the02509devices currently selected in the console output redirection word,02510it checks to see that output to the selected device has not been02511suspended by XON/XOFF protocol, and that DTR is high if02512it should be.02513Once the character has been output, if ETX/ACK protocol is in use,02514and the specified length of message has been output, an Etx02517Entry parameters02518C = character to be output02522j03CB 0002523CONOUT storage variables02524: convolutshormal03CC DB030252503CE 00000253002520: convolutshormal03CE 00000253002520: convolutshormal03CE 00000253002520: convolutshormal03CE 00000253003CE 000002530 <td></td> <td></td> <td>•</td> <td>output</td> <td></td> <td></td>			•	output		
02504       i       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       ;       supended 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       :       C         02517       :       Entry parameters         02521       :       CONOUT storage variables         02522       :       0         03CB 00       02525       CONOUT\$Character:       DB         03CE DB03       02525       CONOU						
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         02519 ;       C = character to be output         02520 ;       02520 ;         02521 ;       CONOUT storage variables         02522 ;       03CB 00       02525 CONOUT\$character:       DB       ; Save area for character to be output         02527 ;       connout\$shormal       ; code that will process the next       ; code that will process the next         02526 ;       :       :       condut\$shormal       ; conaracter. The default case is						
02506 ;       A primitive "state-machine" is used to step through escape         02507 ;       sequence recognition.         02508 ;       In 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         02515 ;       character is output and the device is flagged as being suspended.         02516 ;       character to be output         02517 ;       Entry parameters         02521 ;       CONOUT storage variables         02522 ;       03CB 00       02523 CONOUT\$Character:       DB       ; Save area for character to be output         02524       03CE DB03       02525 CONOUT\$Character:       DW       CONOUT\$Normal         02529 ;       code that will process the next       ; code that will process the next         02529 ;       code that will process the next       ; code that will process the next         02520 ;       ; coNOUT\$Normal       ; conout\$Normal <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
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       ; Entry parameters         02521       ; CONOUT storage variables         02521       ; CONOUT storage variables         02524       ;         03CB 00       02525         02000       02526         03CB 00       02527         03CB 00       02526         03CB 00       02527         03CE 0000       02526         03CE 0000       02529         03CE 0000       02530         03CE 0000       02530         03CE 0000       02530         02530       CONOUT\$String\$Poi			; trigger	THA PACTI	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 thecks 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 = character to be output         02520       CONOUT storage variables         02521       CONOUTSCharacter:       DB         02522       it convolutshormal         02523       CONOUT\$Processor:       DW         02524       it code that will process the next         02527       it code that will process the next         02528       CONOUT\$Normal         02529       CONOUT\$Normal         02529       CONOUT\$Normal         02520       it character. The default case is         02527       it convolt\$Normal         02528       CONOUT\$Normal         02529						asea to steb turonàn escape
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       ;       susended 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       02525       CONOUT\$Character:       DB         03CE DB03       02525       CONOUT\$Processor:       DW       CONOUT\$Normal         02529       ;       .       .       .         03CE 0000       02530       CONOUT\$Processor:       DW       CONOUT\$Normal         02529       .       .       .       .						maut observator to sll 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 ;       Entry parameters         02519 ;       C = character to be output         02520 ;       02521 ;         02520 ;       02521 ;         02521 ;       CONOUT storage variables         02522 ;       000         02524 ;       ONOUT\$CONOUT\$Normal         03CB 00       02525 CONOUT\$Processor:       DW         03CC DB03       02525 CONOUT\$Processor:       DW         03CE 0000       02527 ;       code that will process the next         02529 ;       : convOUT\$Normal       ; character. The default case is         02529 ;       : CONOUT\$String\$Pointer: DW       ; This points to a string (normally         02531 ;       : is being preset by characters from						
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 ;         02518 ;       C = character to be output         02519 ;       C = character to be output         02520 ;       02521 ;         02521 ;       CONOUT storage variables         02522 ;       03CB 00       02525 ;         03CE 000       02526 ;         03CE 000       02527 ;       DB       ; Save area for character to be output         02524       03CC DB03       02525 ;       CONOUT\$Character:       DB       ; Save area for character to be output         02527 ;       contout\$Normal       ;       ;       ;       ;       ;         03CE 0000       02526 ;       ;       ;       ;       ;       ;       ;       ;         03CE 0000       02527 ;       ;       DW       CONOUT\$Normal       ;       ;       ;       ;       ;       ;         03CE 0000       0						
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         02524       .         03CB 00       02525         03CB 01       02525         02000       02526         03CC DB03       02525         03CC DB03       02526         03CE 0000       02529         03CE 0000       02529         03CE 0000       02530         CONOUT\$String\$Pointer: DW       0         03CE 0000       02530						
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       ;       O2522         02521       ;       CONOUT storage variables         02522       ;       O2524         03CE 00       02525       CONOUT\$Character:       DB       0       ;Save area for character to be output         02524       ;       is is the address of the piece of         02527       ;       code that will process the next         02528       ;       code that will process the next         02529       ;       CONUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer:       DW       ; This points to a string (normally         03CE 0000       02530       CONOUT\$String\$Pointer:       DW       ; This points to a string (normally         02531       ;       is being preset by character from					ore protocol	, and that DIV 12 0190 11
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       ; C = character to be output         02520       ;         02521       ; CONOUT storage variables         02522       ;         03CB 00       02523         02524       03CC DB03         02525       CONOUT\$Character:         03CC DB03       02525         02520       ; This is the address of the piece of         02521       ; code that will process the next         02526       ; code that will process the next         02527       ; code that will process the next         02528       ; CONOUT\$String\$Pointer:         03CE 0000       02530       CONOUT\$String\$Pointer:         03CE 0000       02531       ; in the configuration block) that						tout if ETY/ACK exctonel is in use
02515       ;       character is output and the device is flagged as being suspended.         02516       ;         02517       ;         02518       ;         02519       ;         02520       ;         02521       ;         02522       ;         03CB 00       02523         02524       ;         03CC DB03       02525         000UT\$Processor:       DW         02520       ;         03CC DB03       02525         000UT\$Processor:       DW         02527       ;         02528       ;         02529       ;         03CE 0000       02530         CONOUT\$String\$Pointer:       DW         03CE 0000       02530						
02516 ;         02517 ;       Entry parameters         02518 ;         02519 ;       C = character to be output         02520 ;         02521 ;       CONOUT storage variables         02522 ;         03CB 00       02523 ;         03CC DB03       02525 ;         03CC DB03       02525 ;         03CC DB03       02525 ;         03CE 000       02526 ;         02529 ;       ; This is the address of the piece of ; code that will process the next ; claracter. The default case is ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%normal ; concoltf%no						
02517       ; Entry parameters         02518       ;         02519       ; C = character to be output         02520       ;         02521       ; CONOUT storage variables         02522       ;         03CB 00       02523         02524       DB         03CC DB03       02525         02526       ; This is the address of the piece of         02527       ; code that will process the next         02528       ; character. The default case is         02529       ; CONOUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer: DW       0 ;This points to a string (normally         03CE 0000       02532       ; in the configuration block) that				er is outpu	and the de	AICE IS HI <b>gaden a</b> s neind ensheuden.
02518       :       C = character to be output         02519       ;       C = character to be output         02520       ;       02521         02521       ;       CONOUT storage variables         02522       ;       000         02523       CONOUT\$Character:       DB       0       ;Save area for character to be output         03CB       00       02523       CONOUT\$Character:       DB       0       ;Save area for character to be output         03CE       02525       CONOUT\$Processor:       DW       CONOUT\$Normal         02526       ; code that will process the next       ; coharacter.       The default case is         02527       ; coharacter.       CONOUT\$Normal       ; CONOUT\$Normal         02529       ; CONOUT\$Normal       ; coharacter.       The default case is         02529       ; in the configuration block) that       ; in the configuration block) that         02531       ; is being preset by character from				aramatar-		
02519       ;       C = character to be output         02520       ;         02521       ;         02522       ;         03CB 00       02523         02524       0         03CC DB03       02525         02526       ;         03CC DB03       02525         02527       ;         02528       ;         02529       ;         02520       ;         02521       ;         02524       ;         03CC DB03       02525         02526       ;         02527       ;         02528       ;         02529       ;         03CE 0000       02530         CONDUT\$String\$Pointer:       DW         03CE 0000       02531         02532       ;         is being preset by characters from				arameters		
02520 ;       CONOUT storage variables         02521 ;       CONOUT storage variables         02522 ;       03CB 00       02523 CONOUT\$Character:       DB 0       ; Save area for character to be output         03CE 00       02524       03CE 0000       02525       CONOUT\$Processor:       DW CONOUT\$Normal         03CE 0803       02525       CONOUT\$Processor:       DW CONOUT\$Normal       ; code that will process the next         02527       ; code that will process the next       ; code that will process the next       02529         02529       ; concutsNormal       02529       ; concutsNormal         03CE 0000       02530       CONOUT\$String\$Pointer:       DW 0       ; This points to a string (normally 02531         03CE 0000       02532       ; in the configuration block) that 02532       ; is being preset by characters from			ř.	C = oberor	ter to be	*
02521       ; CONOUT storage variables         02522       ;         03CB 00       02523         03CB 00       02524         03CB 00       02525         03CD DB03       02525         03CE 000       02526         02527       ;         02528       ;         02529       ;         02529       ;         02529       ;         02529       ;         02529       ;         02529       ;         02529       ;         02521       ;         03CE 0000       02530         CONOUT\$String\$Pointer:       DW         03CE 0000       02530         03CE 0000       02530         03CE 0000       02530         02531       ;         in the configuration block) that         02532       ;         in the configuration block) that         02532       ;         in the configuration block) that         02532       ;         in the configuration block) that			7	u – charac	ter to be ou	
02522       ;         03CB 00       02523       CONOUT\$Character:       DB       0       ;Save area for character to be output         02524       03CC DB03       02525       CONOUT\$Processor:       DW       CONOUT\$Normal         02525       02526       ;This is the address of the piece of       ;This is the address of the piece of         02527       ; code that will process the next       ; code that will process the next         02528       ; character. The default case is         02529       ; CONOUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer:         03CE 0000       02532       ; in the configuration block) that         02532       ; is being preset by characters from				storana var	inhlas	
03CB 00       02523       CONDUT\$Character:       DB       0       ;Save area for character to be output         02524       02525       CONDUT\$Processor:       DW       CONDUT\$Normal         02526       ;This is the address of the piece of         02527       ; code that will process the next         02529       ; character. The default case is         02529       ; CONDUT\$Normal         03CE 0000       02530       CONDUT\$String\$Pointer:         03CE 0000       02531       ; in the configuration block) that				storage var	100163	
02524         03CC DB03       02525       CONOUT\$Processor:       DW       CONOUT\$Normal         02526       ; This is the address of the piece of         02527       ; code that will process the next         02528       ; character. The default case is         02529       ; CONOUT\$Normal         02529       ; character. The default case is         02529       ; CONOUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer:       DW       0 ; This points to a string (normally         02531       ; in the configuration block) that       ; is being preset by characters from	0308-00		CONCUT&Characte		. 0	•Save area for character to be output
03CC DB03       02525       CONOUT\$Processor:       DW       CONOUT\$Normal         02526       ; This is the address of the piece of         02527       ; code that will process the next         02528       ; character. The default case is         02529       ; CONOUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer:       DW         03CE 0000       02531       ; in the configuration block) that         02532       ; is being preset by characters from		02524	CONCOLACIIAL ACTE	06	, v	yours alea for character to be output
02526       ;This is the address of the piece of         02527       ; code that will process the next         02528       ; character. The default case is         02529       ; CONOUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer: DW       ; This points to a string (normally         02532       ; in the configuration block) that         02532       ; is being preset by characters from	OBCC DBOB		CONDUTSProcesso	ur: Dia	CONDUT	\$Normal
02527       ; code that will process the next         02528       ; character. The default case is         02529       ; CDNOUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer: DW       0 ;This points to a string (normally         02531       ; in the configuration block) that         02532       ; is being preset by characters from					00,001	
02528       ; character. The default case is         02529       ; CONOUT\$Normal         03CE 0000       02530       CONOUT\$String\$Pointer: DW       0 ; This points to a string (normally         02531       ; in the configuration block) that         02532       ; is being preset by characters from						
02529 ; CONOUT\$Normal 03CE 0000 02530 CONOUT\$String\$Pointer: DW 0 ;This points to a string (normally 02531 ; in the configuration block) that 02532 ; is being preset by characters from						
03CE 0000 02530 CONDUT\$String\$Pointer: DW 0 ;This points to a string (normally 02531 ; in the configuration block) that 02532 ; is being preset by characters from						
02531 ; in the configuration block) that 02532 ; is being preset by characters from	03CE 0000		CONDUTEStringE	Cointert DL	0	
02532 ; is being preset by characters from	0000		000001#3tr 109#F	STUCELY DA	, v	
						,

Figure 8-10. (Continued)

03D0 00	02534 02535 02536	CONOUT\$String\$	Length: DB ()	;This contains the maximum number of ; characters to be preset into a
	02537			; from the console output stream
	02538	;		
	02539		RNING ***	
	02540	7 The ou	tput error message rout	ine shares the code in this
	02541	; subrou	tine. On entry here, th	e data byte to be output
	02542	; will D	e on the stack, and the	DE registers set up correctly.
	02543	;		
	02544	7		
03D1 32CB03	02545	CONOUT\$0EM\$Ent		
	02546	STA	CONOUT\$Character	;Save data byte
03D4 C3E803	02547 02548	JMP	CONQUT\$Entry2	;HL already has special bit map
		;		
	02549 02550	;=====================================		
			; <===	BIOS entry point (standard)
03D7 2ACC03	02551 02552	, ====================================		• • • • • • • • • •
03B7 2ACC03	02553	LHLD	CONCUT\$Processor	;Get address of processor to handle
	02554			; the next character to be output
03DA E9	02555	PCHL		;(Default is CONOUT\$Normal)
USDA EX	02556			;Transfer control to the processor
	02557			
	02558	CONCUT\$Normal:		
03DB 79	02558	MOV	A.C	Normal processor for console output
03DC FE1B	02560	CPI	Function\$Key\$Lead	;Check if possible start of escape
03DE CA1204	02561	JZ	CONOUT\$Escape\$Found	; sequence
CODE ONIZON	02562	CONOUT\$Forced:	CONDOT #ESC4P8#FOUND	;Perhaps
03E1 79	02563	MOV	A,C	Fornad output antwo
03E2 32CB03	02564	STA	CONCUT\$Character	Forced output entry point
COLE GEODOO	02565	SIA	concorrection acter	;Not escape sequence Save data byte
03E5 2A5A00	02566	LHLD	CB\$Console\$Output	Get console redirection word
VOLO ZROHVO	02567		CBaconsoleaoutput	; uet console redirection word
	02568	, CONOUT\$Entry2:	• /	output error message entry point
	02569	1	, 、	output error message entry point
03E8 116400	02570	L X T	D,CB\$Device\$Table\$Add	resses ;Addresses of dev. tables
OBEB D5	02571	PUSH	n	Put onto stack ready for loop
OBEC ES	02572	PUSH	Ĥ	stat onto stack ready for 100p
	02573	1 0011		
	02574	CONOUT\$Next\$Dev	vice:	
03ED E1	02575	POP	Н	Recover redirection bit map
OBEE D1	02576	POP	D	Recover device table addresses pointer
O3EF CD6F06	02577	CALL	Select\$Device\$Table	;Get device table in DE
03F2 B7	02578	ORA	A	Check if a device has been
	02579			; selected (i.e. bit map not all zero)
03F3 CAODO4	02580	JZ	CONOUT\$Exit	No, exit
03F6 C5	02581	PUSH	B :Yes - B.,	Save redirection bit map
03F7 E5	02582	PUSH	н	Save device table addresses pointer
	02583	CONDUT\$Wait:		
03F8 CD0F06	02584	CALL	Check\$Output\$Ready	;Check if device not syspended 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
03FF 3ACB03	02590	LDA	CONOUT\$Character	Recover the data byte
0402 4F	02591	MOV	C,A	;Ready for output
0403 CD2608	02592	CALL	Output\$Bata\$Byte	;Output the data byte
0406 FB	02593	EI		
	02594			
0407 CD3A06	02595	CALL	Process\$Etx\$Protocol	;Deal with Etx/Ack protocol
040A C3ED03	02596	JMP	CONOUT\$Next\$Device	\$Loop back for next device
	02597			
	02598	CONOUT\$Exit:		
040D 3ACB03	02599	LDA	CONDUT\$Character	Recover data character
0410 79	02600	MOV	A,C	;CP/M "convention"
0411 C9	02601	RET		
	02602			
	02603	CONOUT\$Escape\$F		Possible escape sequence
0412 211904	02604	LXI		pe ;Vector processing of next character
A.L.E. 000000	02605	CONOUT\$Set\$Proc		
0415 22CC03 0418 C9	02606	SHLD	CONOUT\$Processor	;Set vector address
	02607 02700	RET		Return to BIOS caller;
0410 07	02700	; <del>*</del>		
0418 07	03704			
0410 07	02701 02702	; Console	output: escape sequen	a Properting

Figure 8-10. (Continued)

	02703	•		
	02704	CONOUT\$Process	Escape:	Control arrives here with character
	02705			; after escape in C
0419 211B02	02706	LXI	H,CONOUT\$Escape\$Table	
	02707	CONCUT\$Next\$Ent		
041C 7E	02708	MOV	A,M	;Check if at end of table
041D B7	02709	ORA	A	,
041E CA2B04	02710	JZ	CONOUT\$No\$Match	;Yes, no match found
0421 B9	02711	CMP	C	;Compare to data character
0422 CA3B04	02712	JZ	CONOUT\$Match	They match
0425 23	02713	INX	H	Move to next entry in table
0426 23	02714	INX	н	phote to heat each a the test
0427 23	02715	INX	н	
0428 C31C04	02716	JMP	CONOUT\$Next\$Entry	;Go back and check again
0426 031004	02717		CONCOLANEX CAPITLE	, oo back and check again
	02718	, CONCUT\$No\$Match	•	:No match found, so original
	02719	Concertater	•	; escape and following character
	02720			; must be output
0428 C5	02721	PUSH	B	;Save character after escape
042C 0E1B	02722	MUT	C,Function\$Key\$Lead	;Get escape character
042E CDE103	02723	CALL	CONCUT\$Forced	;Output to console devices
0431 C1	02724	POP	B	;Get character after escape
0432 CDE103	02725	CALL	© CONOUT\$Forced	;Output it, too
STOR CUEIVO	02725		Concert of Cen	,02.PU( 11) 100
	02726	; CONOUT\$Set\$Norm		
0435 21DB03	02728	LXI	H.CONOUT\$Normal	;Set vector back to normal
0438 C31504	02728	JMP	CONCUT\$Set\$Processor	; for subsequent characters
0430 031304	02729		00100140214710025501	4 (of Subseduent Clidi Screis
	02730	;		
	02732	CONCUT\$Match:		
043B 23	02733	INX	н	;HL -> LS byte of subprocessor
043C 5E	02734	MOV	É,M	;Get LS byte
043D 23		INX	E,11 H	JOEL LO DYLE
0430 23 043E 56	02735 02736	MOV	D, M	;Get MS byte
043F EB		XCHG	<b>0</b> , n	;HL -> subprocessor
	02737	PCHL		
0440 E9	02738 02739	PUHL		;Goto subprocessor
	02740	, CONOUT\$Date:	, Subo	rocessor to inject current date
	02741	concorvbate.		to console input stream (using
	02742			wrced input)
0441 218F0F	02743	LXI	H.Date	
	02744	CONOUT\$Set\$Ford	ed\$Input:	
0444 228D0F	02745	SHLD	CB\$Forced\$Input	
0447 C9	02746	RET		rn to BIOS' caller
	02747	;		
	02748	CONOUT\$Time:	; Subp	rocessor to inject time into
	02749		; co	nsole input stream
0448 21990F	02750	LXI	H,Time\$In\$ASCII	
044B C34404	02751	JMP	CONCUT\$Set\$Forced\$Inp	ut
	02752	;		· · · · · · · · · · · · ·
	02753	CONOUT\$Set\$Date		processor to set the date by taking
	02754			e next 8 characters of console output
	02755		; an	d storing them in the date string
044E 21A30F	02756	LXI	H, Time\$Date\$Flags	;Set flag to indicate that the
0451 3E02	02757	MVI	A, Date#Set	; date has been set by program
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
045A C36C04	02762	JMP	CONCUT\$Set\$String\$Poi	Inter
	02763	;		
	02764	;		
	02765	CONOUT\$Set\$Tim		processor to set the time by taking
	02766			e next 8 characters of console output
	02767			d storing them in the time string
045D 21A30F	02768	LXI	H, Time\$Date\$Flags	Set flag to indicate that the
	02769	MVI	A, Time\$Set	; time has been set by program
0460 3E01	02770	ORA	M	
0462 B6		MOV	M, A	
0462 B6 0463 77	02771		A,8	;Set character count
0462 B6 0463 77 0464 3E08	02771 02772	MVI		joet character count
0462 B6 0463 77 0464 3E08 0466 21990F	02771 02772 02773	LXI	H, Time\$in\$ASCII	;Set address
0462 B6 0463 77 0464 3E08	02771 02772 02773 02774			;Set address
0462 B6 0463 77 0464 3E08 0466 21990F	02771 02772 02773 02774 02775	LXI JMP \$	H,Time\$in\$ASCII CONOUT\$Set\$String\$Poi	;Set address Inter
0462 B6 0463 77 0464 3E08 0466 21990F 0469 C36C04	02771 02772 02773 02774 02775 02776	LXI JMP ; CONQUT\$Set\$Str:	H, Time\$in\$ASCII CONOUT\$Set\$String\$Poi ing\$Pointer:	;Set address inter ;HL -> string, A = count
0462 B6 0463 77 0464 3E08 0466 21990F 0469 C36C04 046C 32D003	02771 02772 02773 02774 02775 02776 02777	LXI JMP ; CONQUT\$Set\$Str: STA	H,Time\$in\$ASCII CONOUT\$Set\$String\$Poi ing\$Pointer: CONOUT\$String\$Length	;Set address inter ;HL -> string, A = count ;Save count
0462 B6 0463 77 0464 3E08 0466 21990F 0469 C36C04	02771 02772 02773 02774 02775 02776	LXI JMP ; CONQUT\$Set\$Str:	H,Time\$in\$ASCII CONOUT\$Set\$String\$Poi ing\$Pointer: CONOUT\$String\$Length CONOUT\$String\$Pointer	;Set address inter ;HL -> string, A = count ;Save count

0475 C31504	02780	JMP CONOUT\$Set\$Processor
	02781	7
	02782 02783	CONOUT\$Process\$String: ;Control arrives here for each character
	02784	; in the string in register C. The ; characters are stacked into the
1	02785	; receiving string until either a 00-byte
	02786	; is encountered or the specified number
	02787	; of characters is stacked.
0478 2ACE03	02788	LHLD CONCUT\$String\$Pointer ;Get current address for stacking chars
047B 79 047C B7	02789 02790	MOV A,C ;Check if current character is 00H ORA A
047D CA3504	02791	JZ CONOUT\$Set\$Normal ;Revert to normal processing
0480 77	02792	MQV M,A ;Otherwise, stack character
0481 23	02793	INX H ;Update pointer
0482 3600	02794	MVI M,OOH ;Stack fail-safe terminator
0484 22CE03 0487 21D003	02795 02796	SHLD CONOUT\$String\$Pointer ;Save updated pointer LXI H.CONOUT\$String\$Length :Downdate count
0487 210003	02795	LXI H,CONOUT\$String\$Length ;Downdate count DCR M
048B CA3504	02798	JZ CONOUT\$Set\$Normal ;Revert to normal processing
	02799	; if count hits O
048E C9	02800	RET ;Return with output vectored back
	02801	; to CONOUT\$Process\$String
	02802 02900	
	02900	;# ;
	02902	; Auxiliary input status
	02903	<b>7</b>
	02904	; This routine checks the character count in the
	02905	; appropriate input buffer.
	02906 02907	; The A register is set to indicate whether or not ; data is waiting.
	02908	, Data is waiting.
	02909	; Entry parameters: none.
	02910	;
1	02911	; Exit parameters
	02912 02913	
	02913	; A = 000H if there is no data waiting ; A = 0FFH if there is data waiting
	02915	7
	02916	, 2008.000.000.000.000.000.000.000.000.000
	02917	AUXIST: ;<=== BIOS entry point (Private)
048F 2A5C00	02918 02919	;=====================================
0492 116400	02920	LXI D.CBSDevice\$Table\$Addresses ; and table pointer
0495 CD6F06	02921	CALL Select\$Device\$Table ;Get device table address
0498 C34708	02922	JMP Get\$Input\$Status ;Get status from input device
	02923	; and return to caller
	02924 03000	3 
	03001	
J	03002	; Auxiliary output status
	03003	;
	03004	; This routine sets the A register to indicate whether the
1	03005 03006	; Auxiliary device(s) is/are ready to accept output data. : As more than one device can be used for auxiliary output, this
	03008	; notice returns a Boolean AND of all of their statuses.
	03008	
	03009	; Entry parameters: none
1	03010 03011	
	03011	; Exit parameters ;
	03012	; A = 000H if one or more list devices are not ready
	03014	; A = OFFH if all list devices are ready
l .	03015	;
1	03016 03017	;
	03017	;<==== BIOS entry point (Private)
	03019	; =====================================
049B 2A5E00	03020	LHLD CB\$Auxiliary\$Output ;Get list redirection word
049E C37905	03021	JMP Get\$Composite\$Status
	03022 03100	7 
	03100	; # :
	03102	Auxiliary input (replacement for READER)
1	03103	÷
	03104	; This routine returns the next input character from the
1		

## The CP/M Programmer's Handbook

Figure 8-10. (Continued)

254

r

03100       i appropriate logical auxiliary device.         03100       i Entry parameters: nome.         03100       i Entry parameters: nome.         03100       i A = data Character         03111       i A = data Character         03112       i A = data Character         03113       i A = data Character         03114       i A = data Character         03115       i A = data Character         03111       i A = data Character         03111       i A = data Character         03111       i Auxiliary output (replaces PURCH)         03200       i This routime output (replaces PURCH)         03201       i Auxiliary device tabus for more and auxichados         03201       i i in the parameters         03202       i Entry parameters         03203       i AuxOUTBUSSHERSong B B CR.[F.7, 'Auxiliary device not Ready?', CR,LF.0         03204       i is the parameters         03205       i Ltr.D CBHARMILIARYBULPUT (Feel acce tabus of the onstrond)         03205		
03107       Entry parameters         03107       Exit parameters         03107       A = data character         03111       A = data character         03111       (amount of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second o	03105	; appropriate logical auxiliary device.
03100 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03101 03401 03401 03401 03401 03500 03501 03500 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035001 035000 035001 035000 0350000 0350000 03500000 03500000000		·
<pre>03109 ; Exit parameters 03111 ; A = data character 03113 ; 03113 ; 03114 AUXIN: (=== BIDS entry point (standard) 03114 AUXIN: (=== BIDS entry point (standard) 03114 AUXIN: (=== BIDS entry point (standard) 03114 AUXIN: (=== BIDS entry point (standard) 03115 List status 04A1 C35106 03115 List entry point (replaces PUNCH) 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 03200 ; 04DP 200003219 ; 04DP 20000; 03200 ; 04DP 20000; 03200 ; 04DP 20000; 03200 ; 04DP 20000; 04DP 2000; 04DP 2000; 04DP 2000; 04DP 2000; 04DP 2000; 04DP 200; 04DP 200; 04DP 200; 04DP 200; 04DP 200; 04DP 200; 04DP 200</pre>		
<pre>03110 ; A = data character 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111 ; 03111</pre>		
00112 00113 00114 00114 00111 00110 00114 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 00111 001111 00111 001111 00111 00111 00111 00111 00111 00111 001		7
03113 03114 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 03115 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		
0411 AXCON       03114       AUXIN:       ;(=== BIOS entry point (standard)         0441 145/00       03115       LHLD CBRAuxiliaryEnput       :Get redirection word         0447 145/00       03116       LHLD CBRAuxiliaryEnput       :Get redirection word         0447 145/00       03117		
0441 14400 03117 0447 114400 03117 0447 114400 03117 0447 114400 03117 0447 104400 03117 0447 104400 03117 0447 104400 03117 0447 104400 03117 0447 104400 03117 0447 10440 03118 0447 10440 03118 0447 10440 03118 0447 10440 03118 0447 10440 03118 0447 10440 03118 0447 10440 03118 0448 10440 144 0448 104047 0458 1044 0458 1044		
04A4 CB500 03117 LXI D,CB9DeviceTableAddresses ; and table pointer 04A7 CB500 03118 CAL SelectBoviceTable job evice table address 04A7 CB500 03119 JMP DetBoviceTable job evice table address 03200 ; 03200 ; 03210 ; 03211 ; 03210 ; 03212 ; 04AD 0D0A0741750213 AUXOUT&BusytMessage BB CR,LF,7,'Auxiliary device not Ready?',CR,LF,0 03214 ; 03217 ; 03214 ; 03217 ; 03214 ; 03217 ; 03214 ; 03217 ; 03214 ; 03217 ; 03214 ; 03217 ; 03218 ; 03219 ; 04D 02040741750213 AUXOUT&BusytMessage ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D 02040 ; 04D		
04A CD9106 03119 JHP GetsInputsCharacter joet device table address 04A CD9106 03119 JHP GetsInputsCharacter joet mext input character j and return to caller 03200 if 03201 if Auxiliary output (replaces PUNCH) 03202 i This routine outputs a data byte to the auxiliary device(s). 03203 i This routine outputs a data byte to the auxiliary device(s). 03204 i It is similar to CONQUT except that it uses the watchdog 0306 i Entry parameters 03205 i Entry parameters 03210 j C = data byte 03215 j = C = data byte 03216 j = C = data byte 03216 j = C = data byte 03217 j = C = data byte 03218 j = C = data byte 03219 j = C = DES entry point (standard) 03219 j = C = DES entry point (standard) 03220 j JHD DESeauxiliarySoutput i foet aux. redirection word 03210 j = C = DESeauxiliarySoutput i foet aux. redirection word 03220 j = List status 03206 j = List status 03207 j = List status 03300 j = List status 03303 j = List status 03304 j = This routine sets the A register to indicate whether the 03305 j = List status 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = Entry parameters 03306 j = Entry parameters 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s) is/are ready to accent output data. 03306 j = CALE Device(s)	04A1 2A5C00 03116	LHLD CB\$Auxiliary\$Input ;Get redirection word
03120       ; and return to caller         03121       ;         03121       ;         03121       ;         03120       ;         03201       ;         03202       ;         03203       ;         03203       ;         03203       ;         03203       ;         03204       ;         03205       ;         03206       ;         03207       ;         03208       ;         03209       ;         03200       ;         03201       ;         03202       ;         03203       ;         03204       ;         03205       ;         03206       ;         03217       ;         03218       ;         1       ;         03217       ;         03218       ;         03217       ;         03218       ;         03217       ;         03218       ;         03217       ;         032111ABOD       ; <t< th=""><th>04A7 CD6F06 03118</th><td>LAI D,CBURYCEFIADIEFAGOPESES ; and table pointer CALL SelectSDeviceStable :Get device table address</td></t<>	04A7 CD6F06 03118	LAI D,CBURYCEFIADIEFAGOPESES ; and table pointer CALL SelectSDeviceStable :Get device table address
03120       ; and return to caller         03121       ;         03121       ;         03121       ;         03120       ;         03201       ;         03202       ;         03203       ;         03203       ;         03203       ;         03203       ;         03204       ;         03205       ;         03206       ;         03207       ;         03208       ;         03209       ;         03200       ;         03201       ;         03202       ;         03203       ;         03204       ;         03205       ;         03206       ;         03217       ;         03218       ;         1       ;         03217       ;         03218       ;         03217       ;         03218       ;         03217       ;         03218       ;         03217       ;         032111ABOD       ; <t< th=""><th>04AA C39106 03119</th><td>JMP Get\$Input\$Character ;Get next input character</td></t<>	04AA C39106 03119	JMP Get\$Input\$Character ;Get next input character
03200       if         03201       is nutliary output (replaces PUNCH)         03202       This routime outputs a data byte to the auxiliary device(s).         03202       It is similar to COMOUT except that it uses the watchdog         03203       If is similar to COMOUT except that it uses the watchdog         03204       If is similar to COMOUT except that it uses the watchdog         03207       if this happens.         03208       If this happens.         03209       Entry parameters         03210       C = data byte         03211       C = data byte         03212       Image: the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of the top of top of the top of the top of top of the top of the top of the top of top of the top of top of the top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of top of		; and return to caller
03201 ;       Auxiliary output (replaces PUNCH)         03203 ;       This routime outputs a data byte to the auxiliary device(s).         03203 ;       This similar to COMOUT except that it uses the watchdog         03204 ;       It is similar to COMOUT except that it uses the watchdog         03205 ;       If this happens.         03206 ;       Entry parameters         03210 ;       C = data byte         03211 ;       C = data byte         03212 ;       C = data byte         03213 ;       C = data byte         03214 ;       C = data byte         03215 ;       C = data byte         03216 ;       C = data byte         03217 ;       C = data byte         03218 ;       LHLD CBMAUXITERUSPHESSAGE :         04AD 0D0A0741750514 ;       CHLD CBMAUXITERUSPHESSAGE :         03217 ;       LHLD CBMAUXITERUSPHESSAGE :         03217 ;       LHLD CBMAUXITERUSPHESSAGE :         03217 ;       LHLD CBMAUXITERUSPHESSAGE :         03218 ;       LHLD CBMAUXITERUSPHESSAGE :         03219 ;       List status         03210 ;       JMP Multiple@output@syte         03301 ;       List status         03302 ;       List status         03303 ;       S	03121	7 • #
<pre>03203 ; This routine outputs a data byte to the auxiliary device(s). 03205 ; This routine outputs a massage to the console 03205 ; This routine sets the appens. 03206 ; Thry parameters 03210 ; C = data byte 03211 ; C = data byte 03212 ; C = data byte 03213 ; C = data byte 03214 ; C = data byte 03215 ; C = data byte 03215 ; C = data byte 03216 ; C = data byte 03217 ; C = data byte 03218 ; LHLD CBMAUXIIarySOUtput ; Get aux. redirection word 03209 ; LHLD CBMAUXIIarySOUtput ; Get aux. redirection word 03200 ; J 04CE 2ASE00 03219 ; LHLD CBMAUXIIarySOUtput ; Get aux. redirection word 0320 ; List status 03303 ; C = data byte 03304 ; This routine sets the A register to indicate whether the 03305 ; List bwice(s) is/are ready to accept output data. 03306 ; A smore than one device can be used for list output, this 03306 ; A smore than one device and but of their statuses. 03306 ; A smore than one device and but of their statuses. 03307 ; List bwice(s) is/are ready to accept output, this 03308 ; A smore than one device and but add for list output, this 03309 ; Entry parameters 03310 ; A smore than one device are not seady 03310 ; A smore than one device are ready 03310 ; A smore than one device are ready 03310 ; A smore than one device are ready 03310 ; A smore than one device are ready 03311 ; Exit parameters 03312 ; A = 000H if one or more list devices are not ready 03312 ; 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 0FFH if all list devices are not ready 03315 ; A = 000H if one or more list device. 03316 ; 03317 ; 03317 ; 03319 ; 03319 ; 03319 ; 03319 ; 03310 ; 03319 ; 03310 ; 03319 ; 03310 ; 03319 ; 03310 ; 03319 ; 03310 ; 03319 ; 03310 ; 03319 ; 03319 ; 03310 ; 03310 ; 03310 ; 03310 ; 03310 ; 03311 ; 03311 ; 03312 ; 03312 ; 03312 ; 03312 ; 03312 ; 03314 ; 03315 ; 03315 ; 03316 ; 03316 ; 03317 ; 03317 ; 03318 ; 03318 ; 03319 ; 03319 ; 03319 ; 03310 ; 03319 ; 03310 ; 03310 ; 03310 ; 03310 ; 03310 ; 03310 ; 03310 ; 03310 ; 03310 ; 03310 ; 03311 ; 03311 ; 03311 ; 03311 ; 03312 ; 033</pre>	03201	; Auxiliary output (replaces PUNCH)
03204       j       It is similar to CONOUT except that it uses the watchdog         03205       j       30 seconds at a time. It outputs a message to the console         03206       j       30 seconds at a time. It outputs a message to the console         03207       if this happens.         03208       j         03209       Entry parameters         03211       C = data byte         03212       J         04AD 0D0A07417503213       AUXOUTSEusyHessage: DB       CR.LF.7.'Auxiliary device not Ready?'.CR.LF.0         03215       j====================================	03202	•
<pre>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 03207 ; if this happens. 03209 ; Entry parameters 03210 ; C = data byte 03211 ; C = data byte 03212 ; C = data byte 03214 ; 03214 ; 03214 ; 03214 ; 03214 ; 03216 AUXOUT\$Busy#Message: DB CR.LF.7. 'Auxiliary device not Ready?',CR.LF.0 03214 ; 03216 AUXOUT\$Busy#Message: JDS entry point (standard) 03216 AUXOUT\$Busy#Message: Message to be output if time 03216 JHP Multiple\$Output\$Byte 03200 ; 03300 ; 03310 ; 03311 ; 03311 ; 03312 ; 03312 ; 03312 ; 03312 ; 03312 ; 03312 ; 03312 ; 03313 ; 04 ; 05 ; 05 ; 05 ; 05 ; 05 ; 05 ; 05 ; 05</pre>		
03206 ; 30 seconds at a time. It outputs a message to the console 03207 ; if this happens. 03208 ; 03209 ; Entry parameters 03210 ; C = data byte 03211 ; C = data byte 03211 ; C = data byte 03212 ; 03213 AUXOUTSBusySMessage: DB CR.LF.7, 'Auxiliary device not Ready?',CR.LF.0 03215 ; 03215 AUXOUT: 03216 LHLD CBSAuxiliarySOutput ;Get aux. redirection word 04CE 2ASE00 03219 LXI D.AUXOUTSBusySMessage ;Message to be output if time 03210 ; 04CE 2ASE00 03219 LXI D.AUXOUTSBusySMessage ;Message to be output if time 03200 ; 04CE 2ASE00 03219 LXI D.AUXOUTSBusySMessage ;Message to be output if time 03200 ; 04CE 2ASE00 03221 JMP MultipleSOutputsByte 03200 ; 03300 ; 03311 ; Exit parameters: none 03311 ; 03312 ; 03312 ; 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; 03315 LISTST: ; 03316 ; 03316 ; 03317 ; 03318 LISTST: ; 03318 ; 03322 ; 03320 LHLD CBSListSOutput ;Get list redirection word 04D7 2A6200 03320 LHLD CBSListSOutput ;Get list redirection word 04D7 2A6200 ;# 03320 ; 04D7 ; 04D7 2A6200 ;# 03320 ; 04D7 ; 04D7 2A6200 ;# 03320 ; 04D7 ; 04D7 2A6200 ;# 04D7 ; 04D7 2A6200 ;# 04D7 ; 04D7 ; 0	03204	; It is similar to convol except that it uses the watchoog ; timer to detect if a device stave busy for more than
03207 ; if this happens. 03200 ; Entry parameters 03210 ; C = data byte 03212 ; C = data byte 03213 AUXOUTBEDUSYBHessage: DB CR.LF.7. 'Auxiliary device not Ready?', CR.LF.0 03214 ; 03214 ; 03214 ; 03214 ; 03216 AUXOUTBEDUSYBHessage: DB CR.LF.7. 'Auxiliary device not Ready?', CR.LF.0 03216 AUXOUTBEDUSYBHESsage ; BIOS entry point (standard) 03216 AUXOUTS LIKE CONSUMERSAGE ; runs out 04D1 11AD0 03219 LIXI D. AUXOUTBEDUSYBHESsage ; be output if time 03202 ; 04D2 C3A205 03220 JMP Multiple\$DutputsByte 03300 ; 03300 ; 03400 ;	03206	; 30 seconds at a time. It outputs a message to the console
03209       Entry parameters         03211       C = data byte         03212       AUXOUTSBusyBMessage:       DB CR.LF.7.'Auxiliary device not Ready?'.CR.LF.0         03215       JHE       CBAUXIT:       ; C=== BIOS entry point (standard)         03215       LLID       CBAUXIT:       ; C=== BIOS entry point (standard)         04D1       11AD04       03215       LXI       D.AUXOUTBUSyBMessage:       ; Hessage to be output if time         04D4       C3A205       03221       JMP       Multiple@OutputsBusyBMessage:       ; Ums out         04D4       C3A205       03222       JMP       Multiple@OutputsBusyBMessage:       ; Ums out         04D4       C3A205       03222       JMP       Multiple@OutputsBusyBMessage:       ; Ums out         04D4       C3A205       is/are ready to accept output data.       033005       ;       033005         03300       ;       This routine sets the A register to indicate whether the       033005       ;         03300       ;       Entry parameters:       nore       is/are       ;         0331	03207	; if this happens.
<pre>03210 ; 03212 ; 03212 ; 03213 AUXOUT@Busy\$Message: DB CR,LF,7,'Auxiliary device not Ready?',CR,LF,0 03214 ; 03214 ; 03214 ; 03216 AUXOUT: ;=== BIOS entry point (standard) ;====================================</pre>		
03211       ;       C = data byte         034AD       000407417503213       AUXOUT\$Busy\$Hessage:       DB       CR.LF.7.'Auxiliary device not Ready?',CR,LF.0         03215       ;====================================		
04AD 0D0A07417503213 AUXOUT\$BusyNtessage: DB CR.LF.7. 'Auxiliary device not Ready?'.CR.LF.0 03215 ;====================================	03211	
<pre>03214 ; 03216 AUXOUT: ;</pre> 03216 AUXOUT: ; 03216 AUXOUT: ; 03216 AUXOUT: ; 03216 AUXOUT: ; 03217 ; 04D1 11AD04 03219 LXI D,AUXOUT&BusyMessage ; Message to be output if time 03200 ;# 03200 ;# 03200 ;# 03301 ; 03302 ; List status 03303 ; 03303 ; 03304 ; This routine sets the A register to indicate whether the 03305 ; List Device(s) is/are ready to accept output data. 03306 ; As more than one device can be used for list output, this 03307 ; routine returns a Boolean AND of all of their statuses. 03309 ; Entry parameters: none 03311 ; Exit parameters 03312 ; 03312 ; A = 000H if one or more list devices are not ready 03313 ; A = 00FH if all list devices are ready 03314 ; A = 00FH if all list devices are not ready 03315 ; 03317 ;====================================		
<pre>03215 i======= 03216 AUXOUT: ; &lt;=== BIOS entry point (standard) 0320 UXI D,AUXOUT\$Busy\$Message ; Message to be output if time 0320 UXI D,AUXOUT\$Busy\$Message ; Message to be output if time 0320 i= 0320 i= 03300 ; 03300 ; 03311 ; 03311 ; 03312 ; 03312 ; 03312 ; 03312 ; 03313 ; 04D7 2A6200 0 04D7 2A6200 0 0320 UHLD CB\$List\$Output ;Get list redirection word 04D7 2A6200 0 0320 UHLD CB\$List\$Output ;Get list redirection word 04D7 2A6200 0 0320 ; 03301 ; 03302 ; 03400 ;# 03400 ; 03400 ; 04000 ; 04000 ; 04000 ; 040000 ; 040000000000</pre>		AUXUU/#BUSY#MESSage: DB CR,LF,/,'AUXIIIary device not keady?',LK,LF,U
03217       ;====================================	03215	
04CE 2A5E00       03218       LH.D       CB#Auxiliary#Output       for aux. redirection word         04D1 11AD04       03220       LXI       D,AUXOUT#Busy#Message       for aux. redirection word         04D4 C3A205       03222       JMP       Multiple#Output#Byte       for aux. redirection word         03300       r       03300       r       runs out       for aux. redirection word         03301       JMP       Multiple#Output#Byte       for aux. redirection word       for aux. redirection word         03301       JMP       Multiple#Output#Byte       for aux. redirection word       for aux. redirection word         03302       List status       03301       for aux. redirection word       for aux. redirection word         03302       List status       03301       for aux. redirection word       for aux. redirection word         03302       List bevice(s)       Status       for aux. redirection word       for aux. redirection word         03303       T       List status       for aux. redirection word       for aux. redirection word         03303       T       for aux. redirection word       for aux. redirection word       for aux. redirection word         03305       for aux. redirection word       for aux. redirection word       for aux. redirection word <tr< th=""><th>03216</th><th></th></tr<>	03216	
OAD1 11ADO4O3219 0320LXID,AUXOUT\$Busy\$Message; Message to be output if time ; runs out04D4 C3A20503221 03202JMPMultiple\$Output\$Byte;03300:#		
04D4 C3A205 04D4 C3A205 04D4 C3A205 04D4 C3A205 03222 ; 03300 ;# 03301 ; 03302 ; List status 03303 ; 03304 ; This routine sets the A register to indicate whether the 03305 ; List Device(s) is/are ready to accept output data. 03306 ; As more than one device can be used for list output, this 03306 ; As more than one device can be used for list output, this 03306 ; As more than one device can be used for list output, this 03306 ; As more than one device can be used for list output, this 03307 ; routine returns a Boolean AND of all of their statuses. 03308 ; 03308 ; 03308 ; 03309 ; Entry parameters: none 03310 ; 03312 ; 03312 ; 03314 ; A = 00FH if all list devices are not ready 03314 ; A = 0FFH if all list devices are ready 03315 ; 03316 ; 03317 ;====================================		LXI D,AUXQUT\$Busy\$Message :Message to be output if time
04D7 2A6200 04D7		runs out
<pre>03300 ;# 03301 ; 03302 ; List status 03303 ; 03303 ; 03304 ; This routine sets the A register to indicate whether the 03305 ; List Device(s) is/are ready to accept output data. 03306 ; As more than one device can be used for list output, this 03307 ; routine returns a Boolean AND of all of their statuses. 03308 ; 03309 ; Entry parameters: none 03310 ; 03311 ; Exit parameters 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 0FFH if all list devices are ready 03315 ; 03317 ;====================================</pre>		
03301 ; 03302 ; List status 03303 ; 03304 ; This routine sets the A register to indicate whether the 03305 ; List Device(s) is/are ready to accept output data. 03306 ; As more than one device can be used for list output, this 03307 ; routine returns a Boolean AND of all of their statuses. 03308 ; 03309 ; Entry parameters: none 03310 ; 03311 ; Exit parameters 03312 ; 03312 ; 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 0FH if all list devices are ready 03315 ; 03316 ; 03317 ;====================================	03222	5 2 <b>#</b>
03303 ; 03304 ; This routine sets the A register to indicate whether the 03305 ; List Device(s) is/are ready to accept output data. 03306 ; As more than one device can be used for list output, this 03307 ; routine returns a Boolean AND of all of their statuses. 03308 ; 03309 ; Entry parameters: none 03310 ; 03311 ; Exit parameters 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 00FH if all list devices are ready 03315 ; 03315 ; 03316 ; 03317 ;====================================	03301	;
03004 ; This routine sets the A register to indicate whether the 03305 ; List Device(s) is/are ready to accept output data. 03306 ; As more than one device can be used for list output, this 03307 ; routine returns a Boolean AND of all of their statuses. 03308 ; 03309 ; Entry parameters: none 03310 ; 03311 ; Exit parameters 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 0FFH if all list devices are ready 03315 ; 03316 ; 03317 ;====================================		
03305 ; List Device(s) is/are ready to accept output data. 03306 ; As more than one device can be used for list output, this 03307 ; routine returns a Boolean AND of all of their statuses. 03309 ; Entry parameters: none 03310 ; 03311 ; Exit parameters 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 00FH if all list devices are ready 03315 ; 03315 ; 03316 ; 03317 ;==== BIOS entry point (standard) 03318 LISTST: ;<=== BIOS entry point (standard) 04D7 2A6200 03320 LHLD CB\$List\$Dutput ;Get list redirection word 04DA C37905 03321 JMP Get\$Composite\$Status 03300 ;# 03401 ; List output 03402 ; 03400 ;# 03403 ; This routine outputs a data byte to the list device. 03404 ; It is similar to CONQUT except that it uses the watchdog 03405 ; timer to detect if the printer stays busy for more 03406 ; than 30 seconds at a time. It outputs a message to the console 03407 ; if this happens. 03409 ; Entry parameters 03401 ; 03409 ; Entry parameters 03401 ; 03401 ; 03401 ; 03403 ; This routine outputs a data byte to the list device. 03406 ; than 30 seconds at a time. It outputs a message to the console 03407 ; if this happens. 03409 ; Entry parameters 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03402 ; 03403 ; 03409 ; 03409 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03402 ; 03403 ; 03409 ; 03409 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03402 ; 03401 ; 03402 ; 03403 ; 03409 ; 03409 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03401 ; 03402 ; 03401 ; 03402 ; 03401 ; 03401 ; 03402 ; 03403 ; 03401 ; 03402 ; 03403 ; 03405 ; 03405 ; 03405 ; 03405 ; 03405 ; 03405 ; 03406 ; 03406 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407 ; 03407		
0407 ; routine returns a Boolean AND of all of their statuses. 03308 ; 03309 ; Entry parameters: none 03311 ; Exit parameters 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 0FFH if all list devices are ready 03315 ; 03317 ;====================================	03305	List Device(s) is/are ready to accept output data.
03310       ;       Chify parameters         03311       ;       Exit parameters         03312       ;       A = 000H if one or more list devices are not ready         03313       ;       A = 0FFH if all list devices are ready         03314       ;       A = 0FFH if all list devices are ready         03315       ;       03316         03316       ;	03306	; As more than one device can be used for list output, this
03310       ;       Chify parameters         03311       ;       Exit parameters         03312       ;       A = 000H if one or more list devices are not ready         03313       ;       A = 0FFH if all list devices are ready         03314       ;       A = 0FFH if all list devices are ready         03315       ;       03316         03316       ;	03307	; routine returns a poolean AND of all of their statuses.
03310 ; 03311 ; Exit parameters 03312 ; 03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 0FFH if all list devices are ready 03315 ; 03316 ; 03317 ;==== BIOS entry point (standard) 03318 LISTST: ;<=== BIOS entry point (standard) 0319 ;====================================	00007	; Entry parameters: none
0312 ; 0312 ; 0313 ; 0314 ; A = 00FH if all list devices are not ready 03314 ; 03315 ; 03316 ; 03316 ; 03317 ;====================================	03310	
03313 ; A = 000H if one or more list devices are not ready 03314 ; A = 0FFH if all list devices are ready 03315 ; 03316 ; 03317 ;====================================	03311	; Exit parameters
03314       ;       A = OFFH if all list devices are ready         03315       ;         03316       ;         03317       ;====         03318       LISTST:         03319       ;====         03319       ;====         03319       ;====         03319       ;====         04D7 2A6200       03320       LHLD       CB%List%Output       ;Get list redirection word         04DA C37905       03321       JMP       Get%Composite%Status         03400       ;#	03312	
03316 ; 03317 ;====================================	03314	A = OFFH if all list devices are ready
03317       ;====================================	03315	
03318       LISTST:       ;<=== BIOS entry point (standard)         03319       ;====================================	03316	
0310 ;====================================	03318	LISTST: ;<=== BIOS entry point (standard)
03322 ; 03400 ;# 03401 ; List output 03402 ; 03403 ; This routine outputs a data byte to the list device. 03404 ; It is similar to CONCUT except that it uses the watchdog 03405 ; timer to detect if the printer stays busy for more 03406 ; than 30 seconds at a time. It outputs a message to the console 03406 ; than 30 seconds at a time. It outputs a message to the console 03407 ; if this happens. 03408 ; 03409 ; Entry parameters 03410 ; C = data byte	03319	;=================
03322 ; 03400 ;# 03401 ; List output 03402 ; 03403 ; This routine outputs a data byte to the list device. 03404 ; It is similar to CONCUT except that it uses the watchdog 03405 ; timer to detect if the printer stays busy for more 03406 ; than 30 seconds at a time. It outputs a message to the console 03406 ; than 30 seconds at a time. It outputs a message to the console 03407 ; if this happens. 03408 ; 03409 ; Entry parameters 03410 ; C = data byte		LHLD CB\$List\$Output ;Get list redirection word .MP Get\$Composite\$Status
03400 ;# 03401 ; List output 03402 ; 03403 ; This routine outputs a data byte to the list device. 03404 ; It is similar to CONOUT except that it uses the watchdog 03405 ; timer to detect if the printer stays busy for more 03406 ; than 30 seconds at a time. It outputs a message to the console 03407 ; if this happens. 03408 ; 03409 ; Entry parameters 03410 ; 03411 ; C = data byte	03322	•
03402 ; 03403 ; This routine outputs a data byte to the list device. 03404 ; It is similar to CONOUT except that it uses the watchdog 03405 ; timer to detect if the printer stays busy for more 03406 ; than 30 seconds at a time. It outputs a message to the console 03407 ; if this happens. 03409 ; Entry parameters 03410 ; 03410 ; 03411 ; C = data byte	03400	5 <b>*</b>
03403;This routine outputs a data byte to the list device.03404;It is similar to CONOUT except that it uses the watchdog03405;timer to detect if the printer stays busy for more03406;than 30 seconds at a time. It outputs a message to the console03407;if this happens.03408;03409;Entry parameters03410;C = data byte		
03404 ; It is similar to CONOUT except that it uses the watchdog 03405 ; timer to detect if the printer stays busy for more 03406 ; than 30 seconds at a time. It outputs a message to the console 03407 ; if this happens. 03408 ; 03409 ; Entry parameters 03410 ; C = data byte	03403	: This routine outputs a data byte to the list device.
03405; timer to detect if the printer stays busy for more 03406; than 30 seconds at a time. It outputs a message to the console 03407; if this happens. 03408; 03409; Entry parameters 03410; 03411; C = data byte	03404	It is similar to CONDUT except that it uses the watchdog
03408 ; 03409 ; Entry parameters 03410 ; 03411 ; C = data byte	03405	; timer to detect if the printer stays busy for more
03408 ; 03409 ; Entry parameters 03410 ; 03411 ; C = data byte	03406	; than so seconds at a time, it outputs a message to the console if this happens.
03409 ; Entry parameters 03410 ; 03411 ; C = data byte	03408	
03411 ; C = data byte	03409	
		· · · · · · · · · · · · · · · · · · ·
· •		
	L	

Figure 8-10. (Continued)

0400	00040750	7203413 03414	LIST\$Busy\$ ;	Message:	DB	CR,LF,	7, "Printer not Ready?",CR,LF,0	
		03415	; =========	*********	= =			
		03416	LIST:			; <===	BIOS entry point (standard)	
		03417		********				
	2A6200	03418			ist\$Output		;Get list redirection word	
04F8	11DD04	03419	L)	I D,LI	ST\$Busy\$Me	ssage	Message to be output if time	
	C3A205	03420	٩٢				; runs out	
0468	C3A205	03421 03422		P Mult	iple\$Outpu	t\$Byte		
		03500	; ; #					
		03501		quest <b>use</b> r	abaiaa			
		03502	7	quest user	choice			
		03503		is routine	displays	an error	message, requesting	
		03504	; a	choice of:	013010/3	an error	message, requesting	
		03505						
		03506		R	Retry the	operati	on that caused the error	
		03507	;				and attempt to continue	
		03508	,				and return to CP/M	
		03509	;					
		03510	; Th	is routine	accepts a	charact	er from the console,	
		03511	; ce	nverts it i	to upperca	se and r	eturns to the caller	
		03512	; wi	th the resp	ponse in ti	he A reg	ister.	
		03513	; 					
		03514	RUC\$Messag					
	0D0A	03515		DB	CR,LF		Datum I Ima	
500	202020202	03516		DB	-	enter R	~ Retry, I - Ignore, A - Abort : 7,0	
		03518 03519	; Request\$Us					
52F	CD2D03	03519		er≨Choice: LL CONSI	<b>.</b>		Cobble up any type-shead	
	CA3B05	03520	JZ		ı Buffer\$Emp	•	;Gobble up any type-ahead	
	CD3A03	03521	02 CA			. 7		
	C32F05	03522	JM		N est\$User\$Ci	hoice		
550	001,00	03523	On	, negue	estauserali	JOICE		
		03525	RUC\$Buffer	SEmpty:				
53B	21FE04	03526	LX		S\$Message		Display prompt	
	CD5305	03527	CA		ut\$Error\$M	essage	reserve, prompt	
		03528	0.0					
541	CD3A03	03529	CA	LL CONIN	J		;Get console character	
544	CD3BOE	03530	CA		Wpper		Make uppercase for comparisons	
547	32B00D	03531	ST	A Disk‡	Action\$Co	nfirm	;Save in confirmatory message	
54A	F5	03532	PU				;Save for later	
		03533						
	21B00D	03534	LX		sk\$Action\$(			
54E	CD5305	03535	CA	uL Outpu	it\$Error\$Me	essage		
		03536						
551		03537	PO				Recover action code	
552	C:9	03538	RE	r				
		03539	;					
		03600	;#					
		03601	;					
		03602		tput error	message			
		03603	;					
		03604					nessage to all the currently	
		03605					t those being used to receive	
		03606					avoid "deadly embrace" situations	
		03607	; wh	are the pri	inter's be:	ing busy	for too long causes an error message	
		03608				nsole out	tput is being directed to the	
		03609		inter as we	211.			
		03610 03611	; 	in subvasit	na naka		st of the CONCUT subscription	
		03612					ost of the CONOUT subroutine. NNOUT using a private	
		03612		r memory ec try point.		inters Cl	succe datua a bilagia	
		03613	, en	, point.				
		03615	, : En	try paramet	lore			
		03616	, 20	., paramet				
		03617	,	HI ->	00-byte	ermina*	ed error message	
		03618		nu -7	OO-Dyte i	. er milfielt f	in ciint messañe	
		03618	/ Output&Exec	or\$Message:				
553	F5	03620	PU:				;Save message address	
	2A5A00	03621	LH		onsole\$Outp	out	;Get console redirection bit map	
224		03622	XC				, conserv (carecton bit map	
			ĹĤ		ist\$Output		Length that we descend a set with	
557		03623						
557	2A6200	03623 0362 <b>4</b>	LH	LU (B¥L1	ist#Output		;Get list redirection bit map ;HL = list, DE = console	

Figure 8-10. (Continued)

	03626			; bit map that are set to 1 in the
	03627			; list bit map
055B 7C	03628	MOV	A, H	Get MS byte of list
055C 2F	03629	CMA		; 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
	03636			; devices
0563 B4	03637	ORA	н	;Ensure that at least one device
0564 CA6A05	03638	JZ	OEM\$Device\$Present	; is selected
0567 210100	03639	LXI	H, 0001H	;Otherwise use default of device O
	03640	DEM#Device\$Pre	sent:	
	03641	OEM\$Next\$Chara	cter:	
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	Α	Check if end of message
056E C8	03646	RZ		;Yes, exit
056F D5	03647	PUSH	D	;Save message address for later
0570 E5	03648	PUSH	Ĥ,	;Save special bit map
	03649		·	;Data character is in A
0571 CDD103	03650	CALL	CONQUT\$0EM\$Entry	Enter shared code
0574 E1	03651	POP	н	Recover special bit map
0575 C36A05	03652	JMP	0EM\$Next\$Character	
-	03653	;		
	03654			
	03655	;		
	03656		mposite status	
	03657			
	03658		outine sets the A regis	ter to indicate whether the
	03659			to accept output data.
	03660	a As mor	e than one device can t	e used for output, this
	03661	; routin	e returns a Boolean AND	) of all of their statuses.
	03662	7		
	03663	7 Entry	parameters	
	03664	;		
	03665	;	HL = I/O redirection	bit map for output device(s)
	03666	7		
	03667	; Exit p	arameters	
	03668	;		
	03669	;	A = 000H if one or mo	ere list devices are not ready
	03670	;	A = OFFH if all list	devices are ready
	03671	;		
0578 00	03672	GCS\$Status:	DB 0 ;Comp	osite status of all devices
	03673	;		
	03674	7		
	03675	Get\$Composite\$	Status:	
0579 3EFF	03676	MVI	A, OFFH	Assume all devices are ready
057B 327805	03677	STA	GCS\$Status	Preset composite status byte
	03678			
057E 116400	03679	LXI	B,CB\$Device\$Table\$Add	
	03680	PUSH	D	Put onto stack ready for loop
	03681	PUSH	н	Save bit map
0382 E5	03682	GCS\$Next\$Devic	e:	
0582 E5 0583 E1	03682 03683	POP	e: H	Recover redirection bit map
0582 E5 0583 E1 0584 D1	03682	GCS\$Next\$Devic POP POP		Recover redirection bit map Recover device table addresses pointer
0582 E5 0583 E1 0584 D1 0585 CD6F06	03682 03683 03684 03685	POP	н	
0582 E5 0583 E1 0584 D1 0585 CD6F06	03682 03683 03684	POP POP	H D	<pre>#Recover device table addresses pointer #Get device table in DE</pre>
0382 E5 0583 E1 0384 D1 0385 CD6F06 0388 B7	03682 03683 03684 03685	POP POP CALL ORA	H D Select\$Device\$Table A	<pre>#Recover device table addresses pointer</pre>
0382 E5 0583 E1 0584 D1 0385 CD6F06 0588 B7 0589 CA9905	03682 03683 03684 03685 03685 03687 03687	POP POP CALL ORA JZ	H D Select\$Device\$Table A GCS\$Exit	<pre>#Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit</pre>
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0589 CA9905 058C C5	03682 03683 03684 03685 03685 03685 03687 03688 03689	POP POP CALL ORA JZ PUSH	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B	<pre>;Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map</pre>
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0589 CA9905 058C C5 058C E5	03682 03683 03684 03685 03685 03687 03688 03689 03689	POP POP CALL ORA JZ PUSH PUSH	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H	<pre>#Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit</pre>
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0586 B7 0586 C5 0580 E5 0586 C505F06	03682 03683 03684 03685 03685 03687 03688 03689 03689 03690 03691	POP POP CALL ORA JZ PUSH PUSH CALL	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready	Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0588 CA9905 0580 C5 0580 E5 0580 CD0F06 0591 217805	03682 03683 03684 03685 03685 03687 03689 03689 03690 03691 03692	POP POP CALL ORA JZ PUSH PUSH CALL LXI	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H	;Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0580 C5 0580 C5 0580 E5 0580 E5 0580 CD0F06 0591 217805 0594 A6	03682 03683 03684 03685 03685 03687 03689 03689 03689 03690 03691 03692 03693	POP POP CALL ORA JZ PUSH PUSH CALL LXI ANA	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready	Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0580 C5 0580 C5 0580 E5 0580 E5 0580 CD0F06 0591 217805 0594 A6	03682 03683 03684 03685 03685 03687 03689 03689 03690 03691 03692	POP POP CALL ORA JZ PUSH PUSH CALL LXI	H D Select#Device#Table A GCS#Exit B ;Yes - B H Check\$Output#Ready H,GCS#Status	<pre>;Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready ;AND together with previous devices</pre>
0582 E5 0583 E1 0584 D1 0585 CD6F06 0585 CA9905 0580 C5 0580 E5 0580 E5 0580 E5 0580 E5 0581 217805 0591 217805 0594 A6	03682 03683 03685 03685 03687 03687 03689 03690 03691 03692 03693 03694 03695	POP POP CALL ORA JZ PUSH PUSH CALL LXI ANA MOV	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready H,GCS\$Status M	Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready ;AND together with previous devices ; status
0582 E5 0583 E1 0584 D1 0585 CD6F06 0585 CA9905 0580 C5 0580 E5 0580 E5 0580 E5 0580 E5 0581 217805 0591 217805 0594 A6	03682 03683 03685 03685 03685 03687 03687 03689 03690 03691 03691 03692 03693 03694	POP POP CALL ORA JZ PUSH PUSH CALL LXI ANA	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready H,GCS\$Status M	Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready ;AND together with previous devices ; status
0582 E5 0583 E1 0584 D1 0585 CD6F06 0585 CA9905 0580 C5 0580 E5 0580 E5 0580 E5 0580 E5 0581 217805 0591 217805 0594 A6	03682 03683 03685 03685 03687 03687 03689 03690 03691 03692 03693 03694 03695	POP POP CALL ORA JZ PUSH PUSH CALL LXI ANA MOV	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready H,GCS\$Status M M,A	;Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready ;AND together with previous devices ; status ;Save composite status
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0589 CA9905 0580 C5 0580 E5 0580 C5 0580 E5 0580 C0506 0591 A6 0594 A6 0595 77 0596 C38305	03682 03683 03684 03685 03685 03685 03687 03689 03691 03692 03691 03692 03693 03694 03695 03695	POP POP CALL ORA JZ PUSH PUSH CALL LXI ANA MOV	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready H,GCS\$Status M M,A	<pre>;Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready ;AND together with previous devices ; status ;Save composite status</pre>
0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0589 CA9905 0580 E5 0580 E5 0580 E5 0580 E5 0581 217805 0594 A4 0595 77 0596 C38305 0599 3A7805	03682 03683 03684 03685 03687 03688 03689 03690 03691 03692 03693 03693 03695 03695 03695 03695	POP POP CALL ORA JZ PUSH PUSH CALL LXI ANA MOV JMP	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready H,GCS\$Status M M,A	<pre>;Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready ;AND together with previous devices ; status ;Save composite status</pre>
0581 D5 0582 E5 0583 E1 0584 D1 0585 CD6F06 0588 B7 0580 C5 0580 C5 0580 E5 0580 E5 0591 217805 0594 A6 0595 77 0596 C38305 0599 3A7805 0599 C9	03682 03683 03684 03685 03685 03687 03689 03689 03691 03692 03691 03692 03693 03694 03695 03694 03695 03697 03698	POP POP CALL ORA JZ PUSH CALL LXI ANA ANA MOV JMP ; GCS\$Exit:	H D Select\$Device\$Table A GCS\$Exit B ;Yes - B H Check\$Output\$Ready H,GCS\$Status M M,A GCS\$Next\$Device	Recover device table addresses pointer ;Get device table in DE ;Check if a device has been ; selected (i.e. bit map not all zero) ;No, exit ;Save redirection bit map ;Save device table addresses pointer ;Check if device ready ;AND together with previous devices ; status ;Save composite status ;Loop back for next device

Figure 8-10. (Continued)

	03702	;				
	03800	;#				
	03801	;				
	03802		le output	t byte		
	03803	;				
	03804	; This re	outine o	utputs a	data by	te to the all of the
	03805	t device:	s specif:	ied in t	he I/O r	edirection word.
	03806	; It is a	similar '	to CONOU	T except	that it uses the watchdog
	03807	; timer t	to deteci	t if any	of the	devices stays busy for more
	03808	; than 30	0 seconds	s at a t	ime. It	outputs a message to the console
	03809	; if this	s happens	5.		
	03810	7				
	03811		parameter	r s		
	03812	;				
	03813	7			ection b	
	03814	;		nessage	το be ou	tput if time runs out
	03815		$U = da^{2}$	ta byte		
0708 =	03816 03817	; MOB\$Maximum\$Bu;		EQU	1800	abhamban ad alamh ddat i to to t
0708 =	03817	nusanaximumasu	sy	EQU	1800	Number of clock ticks (each at
	03818					; 16.666 milliseconds) for which the
059E 00	03819	MOB\$Character:		DB	0	; device might be busy ;Character to be output
059F 0000	03821	MOB\$Busy\$Messad		DB	ŏ	;Character to be output ;Address of message to be
0000	03822		34.0	2.00	v	; output if time runs out
05A1 00	03823	MOB\$Need\$Messag	ae:	DB	0	; Flag used to detect that the
	03824				~	; watchdog timer timed out
	03825	;				, waterweg timer times out
	03826	Multiple\$Output	t\$Byte:			
05A2 79	03827	MOV	A,C			;Get data byte
05A3 320807	03828	STA		kimum\$Bu	5 Y	;Save copy
05A6 EB	03829	XCHG				;HL -> timeout message
05A7 229F05	03830	SHLD	MOB\$But	sy\$Messa	ge	Save for later use
05AA EB	03831	XCHG				;HL = bit map again
	03832					
05AB 116400	03833	LXI		evice\$Ta	ble\$Addr	
05AE D5	03834	PUSH	D			Save on stack ready for loop
05AF E5	03835	PUSH	н			Save I/O redirection bit map
	03836	MOB\$Next\$Device				
05B0 E1	03837	POP	н			Recover redirection bit map
05B1 D1	03838	POP	D		+ \	Recover device table addresses pointer
05B2 CD6F06	03839	CALL	Select	\$Device\$	lable	;Get device table in DE
0585 87 0586 CAEC05	03840 03841	ORA JZ	A MOB\$Exi			;Check if any device selected
UJBO LAECUS	03841	32	NUDPEXI			
05B9 C5	03842	PUSH	R	;<- Ye	P	;Save device table addresses pointer
0584 E5	03843	PUSH	в Н	,\~ Te		;Save device table addresses pointer ;Save redirection bit map
	03845	1 0001				FORTE LEGALECTACH DAL MEP
	03846	, MOB\$Start\$Watch	ndoas			
05BB AF	03847	XRA	A A			;Reset message needed flag
05BC 32A105	03848	STA		ed\$Messa	ge	,
05BF 010807	03849	LXI		¶aximum\$		;Time delay
05C2 210906	03850	LXI		Vot\$Read		;Address to go to
05C5 CD6D08	03851	CALL	Set\$Wa		-	;Start timer
	03852			-		
	03853	MOB\$Wait: '				
05C8 3AA105	03854	LDA	MOB\$Nee	ed\$Messa	ge	;Check if watchdog timed out
05CB B7	03855	ORA	A			
05CC C2EE05	03856	JNZ		tput\$Mes		;Yes, output warning message
05CF CDOF06	03857	CALL		Dutput\$R	eady	;Check if device ready
05D2 CAC805	03858	JZ	MOB\$Wa:	1 T		;No, wait
	03859	;				
05D5 F3	03860 03861	DI				; Interrupts off to avoid
0504 010000	03861	LXI	в.0			; involuntary reentrance ;Turn off watchdog
05D6 010000 05D9 CD6D08	03862	CALL	B,0 Set\$Wai	tebdog		; (HL setting is irrelevant)
0007 CD0D08	03863	UMLL	Ge (#W8)	cenalog		) (OF BELLING 12 TILETEAGUE)
05DC 3A9E05	03865	LDA	MOBSCH	aracter		;Get data byte
05DF 4F	03866	MOV	C, A			, cet data o/te
05E0 CD2608	03867	CALL		\$Data\$By	te	;Output the data byte
05E3 FB	03868	EI				
05E4 CD3A06	03869	CALL	Process	s\$Etx\$Pr	otocol	;Deal with ETX/ACK protocol
05E7 C3B005	03870	JMP		kt\$Devic		
	03871	7				
	03872	MOB\$Ignore\$Exi	tı			;Ignore timeout error
05EA E1	03873	POP	н			;Balance the stack
05EB D1	03874	POP	D			

	02075	· ·		
	03875 03876	; MOB\$Exit:		
05EC 79	03877	MOV	A, C	CR/M Noosyastics N
05ED C9	03878	RET	A, C	;CP/M "convention"
VJED C#	03879			
	03880	, MOB\$Output\$Mes		
05EE 2A9F05	03881	LHLD	MOB\$Busy\$Message	
05F1 CD5305	03882	CALL	Output\$Error\$Mes	Display warning message
03F1 003303	03883	MOB\$Request\$Ch	output serror sner	sage ; on selected console devices
05F4 CD2F05	03884	CALL	Request\$User\$Cho	pice ;Display message and get
Jor4 Coar 00	03885	CALL	Requestauser acht	; action character
5F7 FE52	03886	CPI	'R'	; action character ;Retry
SF9 CABB05	03887	JZ	MOB\$Start\$Watchd	
5FC FE49	03888	CPI	· indestar tewatcht	
	03889	JZ		; Ignore
SFE CAEAOS	03890	CPI	MOB\$Ignore\$Exit 'A'	
601 FE41	03890			; Abort
603 CA360E		JZ	System\$Reset	; Give BDOS function O
606 C3F405	03892	JMP	MOB\$Request\$Choi	ce
	03893	<b>;</b>		
	03894	MOB\$Not\$Ready:		Watchdog timer routine will call this
	03895			<pre>; routine if the device is busy</pre>
	03896			; for more than approximately 30 seconds
	03897			;Note: This is an interrupt service routine
609 3EFF	03898	MVI	A, OFFH	Set request to output message
60B 32A105	03899	STA	MOB\$Need\$Message	
60E C9	03900	RET	•	Return to the watchdog routine
	03901	;		
	04000	;#		
	04001		output ready	
	04002	;		
	04003	: This r	outine checks to s	ee if the specified device is ready
	04004		eive output data.	······································
	04005			o see if the device has been suspended
	04006		otocol reasons and	
	04007	) (U) (P)		
	04008		This routine does	NOT check if the USART itself is ready.
	04009			in the output data byte routine itself.
	04010	:		In the output data byte fourthe itself.
	04011		parameters	
	04012	1		
	04013	;	DE -> device tat	le le
	04014	;		•••
	04015		arameters	
	04016	1 EALC P	al and tel s	
	04017	,	A = 000H (Zero-1	lag set) : Bevice not ready
	04018	;		lag clear) : Device ready
	04019		A - OFFA (Lero-	ing clears : Device ready
	04020	, Check\$Output\$R	adv.	
60F 210E00	04021	LXI	H,DT\$Status	;Get device status
612 19	04022	DAD	n, Disstatus D	;HL ~> status byte
613 7E	04022	MOV	A.M	;HL ~> status byte ;Get status byte
		MOV		
614 47	04024 04025	ANI	B,A	Take a copy of the status byte
615 E601			DT\$Output\$Susper	
617 C23806	04026	JNZ	COR\$Not\$Ready	;Yes, indicate not ready
	04027			
61A 3E04	04028	MVI	A, DT\$Output\$DTR	Check if DTR must be high to send
61C A0	04029	ANA	B	Mask with device status from table
61D CA3406	04030	JZ	COR\$Ready	;No, device is logically ready
	04031			<b></b>
620 210000	04032	LXI	H, DT\$Status\$Port	;Set up to read device status
623 19	04033	DAD	α	
624 7E	04034	MOV	A, M	;Get status port number
625 322906	04035	STA	COR\$Status\$Port	;Set up instruction below
	04036			
628 DB	04037	DB	IN	
	04038	COR\$Status\$Por	t :	
629 00	04039	DB	0	;< Set up by instruction above
62A 4F	04040	MOV	C,A	;Save hardware status
	04041			
62B 210400	04042	LXI	H, DT\$DTR\$Ready	;Yes, set up to check chip status
62E 19	04043	DAD	D	; to see if DTR is high
62F 7E	04044	MOV	Ă, M	;Get DTR high status mask
	04045	ANA	<b>H</b> ,H	;Get DIR nigh status mask ;Test chip status
630 A1		JZ	COR\$Not\$Ready	;)est cnip status ;DTR low, indicate not ready
	04046	• 02		,
630 A1 631 CA3806	04046 04047 04048	; COR\$Ready:		······································

Figure 8-10. (Continued)

0634 3EFF	04049	MVI	A, OFFH	Tradicate device ready for output
0634 3EFF	04049	ORA	A, UFFH A	;Indicate device ready for output
0637 C9	04051	RET	7	
	04052	;		
	04053	COR\$Not\$Ready:		;Indicate device not ready for output
0638 AF	04054	XRA	A	
0639 C9	04055	RET		
	04056	7		
	04200	; #		
	04201	;		
	04202		s ETX/ACK protocol	
	04203 04204	; This u	outine maintains ETX/ACK	nvatanal
	04204			a characters have been output
	04206			is output and the device
	04207		to output suspended state	
	04208	ACK ch	aracter is received (unde	r interrupt control) will
	04209	; output	be resumed to the device	•
	04210	,		
	04211	; Entry ;	parameters	
	04212	;		
	04213	3	DE -> device table	
	04214	· · · · ·		
	04215	•	arameters	
	04216	,		(and warnt if pagageney)
	04217 04218	1	message count downdated	(and reset if necessary)
	04218	7 Process\$Etx\$Pro	storol:	
63A 210E00	04220	LXI	H,DT\$Status	;Check if ETX/ACK protocol enabled
063D 19	04221	DAD	D	, then it protocol chapted
063E 7E	04222	MOV	Ă, M	
063F E610	04223	ANI	DT\$Output\$Etx	
641 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	н	Save address of count for later
0647 4E	04228	MOV	С,М	;Get LS byte
0648 23	04229	INX	H	
0649 46	04230	MOV	в, м	;Get MS byte
064A OB	04231 04232	DCX MOV	в	
064B 78 064C B1		ORA	A,B C	
064D C25706	04233 04234	JNZ	c PEP\$Save\$Count	;Check if count now zero :No
0650 211200	04235	LXI		Yes, reset to message length
0653 19	04236	DAD	D	, ies, ieset to message tength
0654 4E	04237	MOV	č, M	;Get LS byte
0655 23	04238	INX	н	, oet 20 0, te
0656 46	04239	MOV	в, м	;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	H	
06 <b>5A</b> 70	04244	MOV	м, в	
	04245	;		Beerlah 1. K. Shekker, and K. K. B.
065B B7	04246	ORA	A	Reestablish whether count hit O
065C CO 065D 0E03	04247 04248	RNZ MVI	C.ETX	;No, no further processing required ;Yes, send ETX to device
065F F3	04248	10M	tey ta 1 A	
065F F3 0660 CD2608	04249	CALL	Output\$Data\$Byte	;Avoids involuntary reentrance
0663 FB	04251	EI	Gulbar angraan) (s	
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	A,M	;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	7 <u> </u>		
	04400	7 <b>#</b>		
	04401 04402	; Calaat	device table	
		; Select	device table	
		-		
	04403		utina coane a téchit com	d and depending on which is the
				d, and depending on which is the rresponding device table address.

Figure 8-10. (Continued)

260

	04407	; Entry	parameters		
	04408	;			
	04409	;	HL ≕ Bit map		
	04410	;	DE -> Table of	device	table addresses
ļ	04411	;	The fi	rst addr	ess in the list is called
	04412	;			ignificant bit of the bit map is
	04413	;	nonzer	o, and s	o on.
	04414	,			
	04415	; Exitp	arameters		
]	04416	;			
	04417	;			device table addresses
	04418	;	DE = Selected		able address
	04419	,	HL = Shifted b		1
	04420	1			t was found
	04421 04422	1	Zero If D	it map n	ow entirely 0000
1	04423	, ; Note:			then the first entry in the
	04424		table addresses		
	04425	; uevice		**** 04	retained in DEt
	04426	, Select\$Device\$	Table:		
066F 7C	04427	MOV	A,H	:Get m	ost significant byte of bit map
0670 B5	04428	ORA	L		if HL completely 0
0671 C8	04429	RZ	-		n indicating no more bits set
0672 7D	04430	MOV	A,L		if the LS bit is nonzero
0673 E601	04431	ANI	1		
0675 C28006	04432	JNZ	SDT\$Bit\$Set		return corresponding address
0678 13	04433	INX	D	;No, u	pdate table pointer
0679 13	04434	INX	D		
067A CDDB08	04435	CALL	SHLR		HL right one bit
067D C36F06	04436	JMP	Select\$Device\$	Table	;Check next bit
	04437	SDT\$Bit\$Set:		_	
0680 E5	04438	PUSH	H		shifted bit map
0681 42	04439	MOV	B,D	;Take	copy of table pointer
0682 4B	04440	MOV	C,E		- 4 4
0683 EB 0684 5E	04441 04442	XCHG MOV	~ ~	1 ML ->	address in table
0685 23	04442	INX	E,M H		
0686 56	04444	MOV	D.M	• DE>	selected device table
0000 00	04445	nov	0,11		p registers for another
	04446			; ent	
0687 E1	04447	POP	н		er shifted bit map
0688 CDDBO8	04448	CALL	SHLR		bit map right one bit
068B 03	04449	INX	в		e DT address table pointer to
0680 03	04450	INX	B	; ent	ry
068D 3E01	04451	MVI	A, 1	;Indic	ate that a one bit was found
068F B7	04452	ORA	A	; and	registers are set up correctly
0690 C9	04453	RET			
	04454	;			
	04600	;*			
í	04601	;			
	04602		put character		
	04603	;			
	04604				ut character from the device
1	04605 04606			r tabie I	handed over as an input
	04606	; parame	ter.		
	04608	; Get\$Input\$Char	acter:		
0691 211900	04609	LXI	H,DT\$Character	Count	;Check if any characters have
0694 19	04610	DAD	D		; been stored in the buffer
1	04611	GIC\$Wait:	-		, erenee en end werren
0695 FB	04612	EI			;Ensure that incoming chars. will
1	04613				; be detected
0696 7E	04614	MOV	Α,Μ		;Get character count
0697 B7	04615	ORA	A		
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\$	<b>Buffer</b>	;Returns HL -> character
0440 75	04622		A M		; and with get offset updated
06A2 7E	04623	MOV	A,M		Get the actual data character
06A3 F5	04624	PUSH	PSW		;Save until later
06A4 211900	04625 04626	LXI	H,BT\$Character\$	Cour+	;Check downdated count of chars. in
06A7 19	04627	DAD	D	count	; buffer, checking if input should be
1	VIUE/	000	-		, littly checking it input should be
h					

Figure 8-10. (Continued)

	04920			
0702 11CE02	04921	LXI	D, DT\$2	;Device 2
0705 CD1607	04922	CALL	Service\$Device	
0708 3E20	04923 04924	MVI	4 104501	
070A D3D8	04925	OUT	A,IC\$EOI IC\$OCW2\$Port	Tell the interrupt controller chip
070C D1	04926	POP	B	; that the interrupt has been serviced ;Restore registers
070D C1	04927	POP	В	Shestore registers
070E F1	04928	POP	PSW	
070F 2A8422	04929	ԼՅԱՆ	PI\$User\$Stack	;Switch back to user's stack
0712 F9	04930	SPHL		
0713 E1	04931	POP	н	
0714 FB	04932	EI		Relenable interrupts in the CPU
0715 C9	04933	RET		Resume pre-interrupt processing
	04934 05000	; ;*		
	05001	; "		
	05002		e device	
	05003	;		
	05004	; This ro	outine performs t	he device interrupt servicing,
	05005	; checkir	ng to see if the	device described in the specified
	05006	; device	table (address i	n DE) is actually interrupting,
	05007			haracter. Depending on which data character
	05008 05009	; is inpu (shutti	it, this routine	will either stack it in the input buffer
	05010			stream if the buffer is nearly full), me the output to the device.
	05011	t or with	, suspend of resu	me the odtput to the device.
	05012		arameters	
	05013	;		
	05014	;	DE -> device ta	ble
	05015	;		
	05016	Service\$Device;		
0716 210000	05017	LXI	H, DT\$Status\$Por	
0719 19 071A 7E	05018 05019	DAD MOV	D A,M	; interrupting
071B 321F07	05020	STA	n,n SD\$Status\$Port	;Get status port number ;Store in instruction below
0/10 321-0/	05020	210	SD#Status#Fort	Store in instruction below
071E DB	05022	DB	IN	;Input status
	05023	SD\$Status\$Port:		,,
071F 00	05024	DB	0 ;< Se	t up by instruction above
	05025	;		
0720 210300	05026	LXI	H, DT\$Input\$Read	y ;Check if status indicates data ready
0723 19	05027 05028	DAD ANA	D M	
0724 A6 0725 C8	05028	RZ	61	;Mask with input ready value ;No, return to interrupt service
0/20 00	05030			;Check if any errors have occurred
0726 210700	05031	LXI	H,DT\$Detect\$Err	
0729 19	05032	DAD	D	; interrupting
072A 7E	05033	MOV	Α,Μ	;Get status port number
072B 322F07	05034	STA	S <b>D\$E</b> rror <b>\$</b> Port	Store in instruction below;
0705 BB	05035			· • • • • • •
072E DB	05036 05037	DB SD#Error#Port.	IN	;Input error status
072F 00	05037	SD\$Error\$Port: DB	0 1< Se	t up by instruction above
V/AF VV	05038	;	0 / X 0e	c up by instruction above
0730 210800	05040	ĻXI	H,DT\$Detect\$Err	or\$Value ;Mask with error bit(s)
0733 19	05041	DAD	B	
0734 A6	05042	ANA	M	
0735 CA4707	05043	JZ	SD\$No\$Error	;No bit(s) set
0738 210900	05044	LXI	H,DT\$Reset\$Erro	r\$Port ;Set up to reset error
073B 19	05045	DAD MOV	D A.M	· Cod woodd oowd ownhow
073C 7E 073D 32 <b>4607</b>	05046 05047	STA	A,M SD\$Reset\$Error\$	;Get reset port number Port    ;Store in instruction below
0740 210A00	05048	LXI	H,DT\$Reset\$Erro	
0743 19	05049	DAD	D	
0744 7E	05050	MOV	A, M	;Get reset interrupt value
	05051			
0745 D3	05052	DB	OUT	
	05053	SD\$Reset\$Error\$	Port	
0746 00	05054	DB	0 ;< Se	t up in instruction above
	05055	CDENALCHAR		
	05056	SB\$No\$Error: LXI	H,DT\$Data\$Port	;Input the data character (this may
	05057			
0747 210100 074A 19	05057 05058		0	
0747 210100 074A 19 074B 7E	05057 05058 05059	DAD MOV		; be garbled if an error occurred)
074A 19	05058	DAD	D	

Figure 8-10. (Continued)

074F					
074F		05061			
				***	
÷ · • ·	DB .	05062	DB	IN	;Input data character
		05063	SD\$Data\$Port:		
0750	00	05064	DB	0 ; <set by<="" td="" up=""><td>instruction above</td></set>	instruction above
		05065			
0751	47	05066	MOV	B.A	Take copy of data character above;
	210E00	05067	LXI	H, DT\$Status	Check if either XÓN or ETX protocols
0755		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		05072	ANI	DT\$Output\$Xon	;Check if XON/XOFF is active
	C26E07	05073	JNZ	SD\$Check\$if\$Xon	;Yes, check if XON char. input
U/SE	L20EU/		UNZ	SUPCHECKPITPLON	
		05074			;No, assume ETX/ACK active
0761	3E06	05075	MVI	A, ACK	;Check if input character is ACK
0763	B8	05076	CMP	В	
	C28107	05077	JINZ	SD\$No\$Protocol	;No, process character as data
0/04	020107	05078	SD\$Output\$Desus		
			SD#Output#Desus	peno:	;Yes, device now ready
		05079			; to accept more data, so indicate
		05080			; output to device can resume
		05081			;The moninterrupt driven output
		05082			; routine checks the suspend bit
07/7	75	05083	MOU		
0767			MOV	A,M	;Get status/protocol byte again
0768		05084	ANI		Suspend Preserve all bits BUT suspend
076A		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			
		05088			VON/VOEF eveterel active
			SD\$Check\$if\$Xon	7 ·	;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
07/5	0511		MI 1 *		
076E		05094	MVI	A, XON	;Check if XON character input
0770		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		05098	CMP	R	,
	C28107	05099	JNZ	SD\$No\$Protocol	No, process character as data
0///	C2010/				Device needs pause in output of
		05100	SD\$Output\$Suspe	na:	
		05101			; data, so indicate output suspended
077A	7E	05102	MOV	Α,Μ	;Get status/protocol byte again
077B	F601	05103	ORI	DT\$Output\$Suspend	;Set suspend bit to 1
077D	77	05104	MOV	M, A	Save back in device table
	C3D907	05105	JMP	SD\$Exit	Exit to interrupt service without
0//2	000/0/	05106	One	SDWEXIC	Exit to interrupt service without
					; saving the input character
		05107	;		
		05108	SD\$No\$Protocol:		
	211800	05109	LXI	H,DT\$Buffer\$Length\$Mask	;Check if there is still space
0784		05110	DAD	D	; in the input buffer
0785		05111	MOV	Ă.M	;Get length - 1
0704	20				
0786	30	05112	INR	Α	;Update to actual length
	211900	05113	LXI	H,DT\$Character\$Count	;Get current count of characters
078A		05114	DAD	D	; in buffer
078B	BE	05115	CMP	M	Check if count = length
0780	CAEB07	05116	JZ	SD\$Buffer\$Full	;Yes, output bell character
		05117	PUSH	B	;Save data character
078F			LXI	H, DT\$Put\$Offset	;Compute address of character in
078F 0790	211600	05118			: input buffer
0790		05119			
0790	211600 CDF007		CALL	Get\$Address\$In\$Buffer	;HL -> character position
0790 0793	CDF007	05119 05120			HL -> character position
0790 0793 0796	CDF007 C1	05119 05120 05121	POP	в	;HL -> character position ;Recover input character
0790 0793	CDF007 C1	05119 05120 05121 05122		B M, B	;HL -> character position ;Recover input character ;Save character in input buffer
0790 0793 0796	CDF007 C1	05119 05120 05121 05122 05123	POP	B M, B ; Update	<pre>;HL -&gt; character position ;Recover input character ;Save character in input buffer ;number of characters in input</pre>
0790 0793 0796	CDF007 C1	05119 05120 05121 05122 05123 05123	POP	B M,B ;Update ; buff	<pre>;HL -&gt; character position ;Recover input character ;Save character in input buffer ; number of characters in input er, checking if input should</pre>
0790 0793 0796 0797	CDF007 C1 70	05119 05120 05121 05122 05123 05123 05124 05125	POP MOV	B M,B ;Update ; buff ; be t	<pre>;HL -&gt; character position ;Recover input character ;Save character in input buffer ;number of characters in input</pre>
0790 0793 0796 0797	CDF007 C1	05119 05120 05121 05122 05123 05123	POP	B M,B ;Update ; buff ; be t	<pre>;HL -&gt; character position ;Recover input character ;Save character in input buffer ; number of characters in input er, checking if input should</pre>
0790 0793 0796 0797 0797	CDF007 C1 70 211900	05119 05120 05121 05122 05123 05124 05125 05126	POP MOV LXI	B M,B ;Update ; buff ; be t H,DT\$Character\$Count	<pre>;HL -&gt; character position ;Recover input character ;Save character in input buffer ; number of characters in input er, checking if input should</pre>
0790 0793 0796 0797 0797	CDF007 C1 70 211900 19	05119 05120 05121 05122 05123 05123 05124 05125 05126 05127	POP MOV LXI DAD	B M,B ;Update ; buff ; be t H,DT\$Character\$Count D	;HL -> character position ;Recover input character ;Save character in input buffer : number of characters in input er, checking if input should emporarily halted
0790 0793 0796 0797 0797 0798 0798 0798	CDF007 C1 70 211900 19 34	05119 05120 05121 05122 05123 05124 05125 05126 05127 05128	POP MOV LXI DAD INR	B M,B ;Update ; buff H,DT\$Character\$Count D M	<pre>&gt;HL -&gt; character position &gt;Recover input character . &gt;Save character in input buffer number of characters in input er, checking if input should emporarily halted &gt;Update character count</pre>
0790 0793 0796 0797 0797 0798 0798 0798 0790	CDF007 C1 70 211900 19 34 7E	05119 05120 05121 05122 05123 05124 05125 05126 05126 05127 05128 05129	POP MOV LXI DAD INR MOV	B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M A,M	;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 ;Get updated count
0790 0793 0796 0797 0797 0798 0798 0798 0798 0796 0795	CDF007 C1 70 211900 19 34 7E 211A00	05119 05120 05121 05122 05123 05124 05125 05126 05127 05128 05129 05130	POP MOV LXI DAD INR MOV LXI	B M,B ;Update ; buff H,DT\$Character\$Count D M	<ul> <li>HL -&gt; character position</li> <li>Recover input character</li> <li>Save character in input buffer</li> <li>number of characters in input</li> <li>er, checking if input should</li> <li>emporarily halted</li> <li>Update character count</li> <li>SCH updated count</li> <li>Check if current count matches</li> </ul>
0790 0793 0796 0797 0797 0798 0798 0798 0790	CDF007 C1 70 211900 19 34 7E 211A00	05119 05120 05121 05122 05123 05124 05125 05126 05126 05127 05128 05129	POP MOV LXI DAD INR MOV	B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M A,M	;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 ;Get updated count
0790 0793 0796 0797 0798 0798 0798 0798 0799 0795 0795	CDF007 C1 70 211900 19 34 7E 211A00 19	05119 05120 05121 05122 05123 05124 05125 05125 05126 05127 05128 05127 05128 05129 05130	POP MOV LXI DAD INR MOV LXI DAD	B M,B ;Update ; buff H,DT\$Character\$Count D M A,M H,DT\$Stop\$Input\$Count	<ul> <li>HL -&gt; character position</li> <li>Recover input character</li> <li>Save character in input buffer</li> <li>number of characters in input</li> <li>er, checking if input should</li> <li>emporarily halted</li> <li>Update character count</li> <li>SCH updated count</li> <li>Check if current count matches</li> </ul>
0790 0793 0796 0797 0797 0798 0798 0798 0796 0796 0796 0796 0796	CDF007 C1 70 211900 19 34 7E 211A00 19 BE	05119 05120 05122 05123 05124 05125 05126 05127 05128 05129 05130 05131	POP MOV DAD INR MOV LXI DAD CMP	B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M A,M H,DT\$Stop\$Input\$Count D M	;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 ;Get updated count ;Check if current count matches ; buffer-full threshold
0790 0793 0796 0797 0797 0798 0798 0798 0796 0796 0796 0796 0796	CDF007 C1 70 211900 19 34 7E 211A00 19	05119 05120 05121 05122 05123 05124 05125 05126 05127 05128 05129 05130 05131 05132	POP MOV LXI DAD INR MOV LXI DAD	B M,B ;Update ; buff bet H,DT\$Character\$Count D M A,M H,DT\$Stop\$Input\$Count D	<pre>;HL -&gt; 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 ;Get updated count ;Check if current count matches ; buffer-full threshold ;Not at threshold, check if control</pre>
0790 0793 0796 0797 0798 0798 0798 0798 0798 0796 0796 0792 0741 0742 0743	CDF007 C1 70 211900 19 34 7E 211400 19 BE C2CE07	05119 05120 05121 05122 05123 05124 05125 05126 05126 05127 05128 05129 05129 05130 05131 05132	POP MOV LXI DAD INR MOV LXI DAD CMP JNZ	B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M A,M H,DT\$Stop\$Input\$Count D M SD\$Check\$Control	<ul> <li>jHL -&gt; character position</li> <li>;Recover input character</li> <li>;Save character in input buffer</li> <li>number of characters in input</li> <li>er, checking if input should</li> <li>emporarily halted</li> <li>;Update character count</li> <li>;Get updated count</li> <li>;Check if current count matches</li> <li>; buffer-full threshold</li> <li>;Not at threshold, check if control</li> <li>; character input</li> </ul>
0790 0793 0796 0797 0798 0798 0798 0798 0798 0796 0796 0792 0741 0742 0743	CDF007 C1 70 211900 19 34 7E 211A00 19 BE C2CE07 210E00	05119 05120 05121 05122 05123 05124 05125 05126 05127 05128 05129 05130 05131 05132	POP MOV DAD INR MOV LXI DAD CMP	B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M A,M H,DT\$Stop\$Input\$Count D M	<pre>;HL -&gt; 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 ;Get updated count ;Check if current count matches ; buffer-full threshold ;Not at threshold, check if control</pre>

0744 70	05107	MOU		<b></b>
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	No
0784 210800	05143	LXI	H,DT\$RTS\$Control\$Port	;Yes, get control port number
07B7 19	05144	DAD	D	
07B8 7E	05145	MOV	A, M	
0789 320207	05146	STA	SD\$Drop\$RTS\$Port	Store in instruction below
07BC 210C00				store in instruction below
0780 210000	05147	LXI	H,DT\$Drop\$RTS\$Value	
07BF 19	05148	DAD	D	
07C0 7E	05149	MOV	A, M	;Get value needed to drop RTS
1	05150			
0701 03	05151	DB	OUT	
0/01 03				
	05152	SD\$Drop\$RTS\$Po		
07C2 00	05153	DB	0 ;<- Se	t up in instruction above
	05154			;Drop into input XON test
	05155	SD\$Check\$Input	\$Xon: :Check	if XON/XOFF protocol being used
	05156			temporarily suspend input
07C3 F1	05157	POP	PSW , 10	
				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
1	05162			Josephi udie Dyle
1		CD#Chaultan	-1	· · · · · · · · · · · · · · · · · · ·
1	05163	SD\$Check\$Contr		if control character (other than
1	05164		; CR,	LF, or TAB) input, and update
1	05165			nt of control characters in buffer
07CE CD0808	05166	CALL	Check\$Control\$Char	Check if control character
07D1 CAD907	05167	JZ	SD\$Exit	
0701 040707				<pre>\$No,it is not a control character</pre>
07D4 211C00	05168	LXI	H,DT\$Control\$Count	
07D7 19	05169	DAD	D	
0708 34	05170	INR	M	;Update count of control chars.
	05171			
	05172	SD\$Exit:	· Reset	hardware interrupt system
0709 210500	05173	LXI	H, DT\$Reset\$Int\$Port	Harbware Interrupt System
07DC 19	05174		B	
		DAD	-	<b>.</b>
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	H,DT\$Reset\$Int\$Value	yorore in instruction below
07E6 19	05181			
		DAD	D	
07E7 7E	05182	MOV	A, M	;Get reset interrupt value
	05183			
07E8 D3				
0760 03	05184	DB	OUT	
0/28 03	05184 05185	DB SD\$Reset\$Int\$Po		
	05185	SD\$Reset\$Int\$Po	ort:	instruction above
07E9 00	05185 05186	SD\$Reset\$Int\$Po DB	ort:	instruction above
	05185 05186 05187	SD\$Reset\$Int\$Po	ort:	instruction above ;Return to interrupt service routine
07E9 00	05185 05186 05187 05188	SD\$Reset\$Int\$Po DB RET ;	ort: O ;< Set up in	;Return to interrupt service routine
07E9 00 07EA C9	05185 05186 05187 05188 05188	SD\$Reset\$Int\$Po DB RET ; SD\$Buffer\$Full;	ort: O ;< Set up in	Return to interrupt service routine; Input buffer completely full
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05190	SD\$Reset\$Int\$Po DB RET ; SD\$Buffer\$Full; MVI	ort: O ;< Set up in C,BELL	Return to interrupt service routine Input buffer completely full Send bell character as desperate
07E9 00 07EA C9	05185 05186 05187 05188 05189 05190 05191	SD\$Reset\$Int\$Po DB RET ; SD\$Buffer\$Full;	ort: O ;< Set up in	Return to interrupt service routine Input buffer completely full Send bell character as desperate
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05190	SD\$Reset\$Int\$Po DB RET ; SD\$Buffer\$Full; MVI	ort: O ;< Set up in C,BELL	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05190 05191 05192	SD\$Reset\$Int\$Po DB RET ; SD\$Buffer\$Full; MVI	ort: O ;< Set up in C,BELL	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05190 05191 05192 05193	SD\$Reset\$Int\$Pc DB RET ; SD\$Buffer\$Full: MVI JMP ;	ort: O ;< Set up in C,BELL	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05199 05191 05192 05193 05300	SD\$Reset\$Int\$Pc DB RET } SD\$Buffer\$Fult MVI JMP ; ;#	ort: O ;< Set up in C,BELL	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05189 05190 05190 05191 05192 05193 05300 05301	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ;# ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05190 05191 05192 05193 05300 05301 05302	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ;# ;	ort: O ;< Set up in C,BELL	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05189 05190 05190 05191 05192 05193 05300 05301	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ;# ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05190 05191 05192 05193 05300 05301 05302	SD\$Reset\$Int\$Pc DB RET ; SD\$Buffer\$Full: MP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte Iress in buffer	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to caller will be done by subroutine
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05189 05199 05190 05191 05192 05193 05300 05301 05302 05303	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get add ; This rc	ort: O ;< Set up in C,BELL Output\$Data\$Byte Wress in buffer outine computes the addre	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05189 05190 05190 05192 05193 05300 05301 05302 05303 05304 05305	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full MVI JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte Iress in buffer	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to caller will be done by subroutine
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05190 05191 05192 05193 05300 05300 05300 05303 05304 05305 05306	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte Press in buffer utine computes the addre in a device buffer.	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to caller will be done by subroutine
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05190 05190 05190 05192 05193 05300 05301 05302 05303 05304 05305 05306 05307	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; Get add ; ; This rc ; access ; Entry F	ort: O ;< Set up in C,BELL Output\$Data\$Byte Wress in buffer outine computes the addre	Return to interrupt service routine Input buffer completely full Send bell character as desperate measure. Note JMP return to caller will be done by subroutine
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05191 05192 05192 05193 05300 05301 05302 05303 05304 05305 05306 05307 05308	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte lress in buffer wutine computes the addre in a device buffer. warameters	Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine sss of the next character to
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05190 05191 05192 05300 05300 05300 05303 05304 05306 05306 05308 05309	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get adc ; ; This rc ; access ; Entry P	ort: O ;< Set up in C,BELL Output\$Data\$Byte mess in buffer in a device buffer. arameters DE -> appropriate device	<pre>;Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ce table</pre>
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05199 05192 05193 05392 05300 05302 05303 05300 05305 05306 05306 05306 05306 05309	SD\$Reset\$Int\$Po DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte ress in buffer outine computes the addre in a device buffer. arameters DE -> appropriate device HL = offset in the device	Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ce table vice table of either the
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05190 05191 05192 05300 05300 05300 05303 05304 05306 05306 05308 05309	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get adc ; ; This rc ; access ; Entry P	ort: O ;< Set up in C,BELL Output\$Data\$Byte mess in buffer in a device buffer. arameters DE -> appropriate device	Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ce table vice table of either the
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05189 05199 05192 05193 05300 05302 05300 05302 05303 05304 05305 05306 05306 05306 05307	SD\$Reset\$Int\$Po DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte ress in buffer outine computes the addre in a device buffer. arameters DE -> appropriate device HL = offset in the device	Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ce table vice table of either the
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05190 05190 05192 05192 05301 05300 05300 05303 05304 05304 05305 05304 05305 05306 05307 05308 05310	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get add ; ; This rc ; access ; Entry F ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte utine computes the addre in a device buffer. arameters DE -> appropriate device HL = offset in the device Get\$Offset or t	Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ce table vice table of either the
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05189 05199 05199 05199 05199 05300 05300 05300 05300 05300 05304 05306 05306 05306 05307 05306 05309 05310 05311 05312	SD\$Reset\$Int\$Po DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte ress in buffer outine computes the addre in a device buffer. arameters DE -> appropriate device HL = offset in the device	Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to the table vice table of either the
07E9 00 07EA C9 07EB 0E07	05185 05187 05187 05188 05190 05190 05190 05190 05192 05193 05300 05300 05300 05300 05300 05303 05304 05305 05307 05305 05307 05309 05310 05310	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get adc ; ; This rc ; access ; Entry p ; ; ; ; ; Exit pa	ort: O ;< Set up in C,BELL Output\$Data\$Byte Iress in buffer nutine computes the addre in a device buffer. arameters DE -> appropriate device HL = offset in the device Get\$Offset or to rameters	Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to the table vice table of either the
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05189 05199 05199 05190 05192 05192 05300 05300 05300 05300 05300 05300 05300 05300 05300 05310 05311 05311 05313	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get add ; ; This rc ; access ; Entry F ; ; ; Entry F ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte less in buffer outine computes the addre in a device buffer. -arameters DE -> appropriate device Get\$Offset or t rameters DE unchanged	<pre>;Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ess of the next character to the next character to the suble rice table rice table of either the the Put\$Offset</pre>
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05190 05192 05192 05192 05300 05300 05300 05300 05303 05304 05306 05305 05306 05306 05307 05306 05307 05310 05310 05311 05312 05314 05315	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get adc ; ; This rc ; access ; Entry p ; ; ; ; ; Exit pa	ort: O ;< Set up in C,BELL Output\$Data\$Byte Iress in buffer nutine computes the addre in a device buffer. arameters DE -> appropriate device HL = offset in the device Get\$Offset or to rameters	<pre>;Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ess of the next character to the next character to the suble rice table rice table of either the the Put\$Offset</pre>
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05187 05189 05190 05192 05192 05300 05300 05300 05300 05300 05304 05304 05304 05304 05306 05307 05308 05310 05311 05313 05315 05315	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; Get add ; This rc ; access ; Entry p ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte utine computes the addre in a device buffer. warameters DE -> appropriate device HL = offset in the device Get\$Offset or to rameters DE unchanged HL -> address in charace	<pre>;Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ess of the next character to the next character to the suble rice table rice table of either the the Put\$Offset</pre>
07E9 00 07EA C9 07EB 0E07	05185 05186 05187 05188 05190 05190 05192 05192 05192 05300 05300 05300 05300 05303 05304 05306 05305 05306 05306 05307 05306 05307 05310 05310 05311 05312 05314 05315	SD\$Reset\$Int\$Pc DB RET SD\$Buffer\$Full: MVI JMP ; ; ; ; Get add ; ; This rc ; access ; Entry F ; ; ; Entry F ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ort: O ;< Set up in C,BELL Output\$Data\$Byte utine computes the addre in a device buffer. warameters DE -> appropriate device HL = offset in the device Get\$Offset or to rameters DE unchanged HL -> address in charace	<pre>;Return to interrupt service routine ;Input buffer completely full ;Send bell character as desperate ; measure. Note JMP return to ; caller will be done by subroutine ess of the next character to ess of the next character to the next character to the suble rice table rice table of either the the Put\$Offset</pre>

Figure 8-10. (Continued)

07F0 19	05319		DAD	D	;HL -> get/put offset in dev. table
07F1 E5	05320		PUSH	н	Preserve pointer to table
07F2 4E	05321		MOV	С,М	;Get offset value
07F3 0600	05322		MVI	B, 0	;Make into word value
	05323			;Update	offset value, resetting to
	05324				end of buffer
07F5 79	05325		MOV	A, C	;Get copy of offset
07F6 3C 07F7 211800	05326 05327		INR LXI		;Update to next position
07FA 19	05328		DAD	H,DT\$Buffer\$Length\$Mask D	
07FB A6	05329		ANA	M	;Mask LS bits with length - 1
07FC E1	05330		POP	Ĥ	Recover pointer to offset in table
07FD 77	05331		MOV	M. A	;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	ם	;HL -> address of buffer in table
0802 7E	05334		MOV	A, M	;Get LS byte of address
0803 23	05335		INX	H	;HL -> MS byte of address
0804 66 0805 6F	05336 05337		MOV MOV	H,M	;H = MS byte
0806 09	05338		DAD	L,A B	;L = LS byte ;Add on offset to base
0807 09	05339		RET	0	;Aud on offset to base
	05340				
	05341	;			
	05400	;*			
i	05401	1			
	05402	;	Check c	ontrol character	
	05403	,			
	05404	;	This ro	utine checks the charact	er in A to see if it is a
	05405 05406	1			, LF, or TAB. The result is
	05406	7	returne	d in the Z-flag.	
	05408	í	Entry p	arameters	
	05409	;		al dime cel s	
	05410			A = character to be che	cked
	05411	;			
	05412	;	Exit pa	rameters	
	05413	;			
	05414	3			ot contain a control character
	05415	3		or if it is CR,	LF, or TAB
	05416 05417				
	05418	;		CR, LF, or TAB.	control character other than
	05419	, Check\$C	ontrol\$C		
0808 3E1F	05420		MVI	$A_r = 1$	Space is first noncontrol char.
080A B8	05421		CMP	B	
080B DA2408	05422		JC	CCC\$No	Not a control character
080E 3EOD	05423		MVI	A, CR	;Check if carriage return
0810 88	05424		CMP	B	
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	05428		.17	B CCC\$No	Not really a control character
081A 3E09	05429		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		ĦVI	A, 1	;Indicate a control character
0822 B7	05433		ORA	A	
0823 C9	05434		RET		
0004 45	05435	CCC\$No:	-	•	;Indicate A does not contain
0824 AF 0825 C9	05436 05437		XRA RET	A	; a control character
0020 07	05438		116.1		
	05500	, ; #			
	05501	,			
	05502	;	Output	data byte	
	05503	;			
	05504	;			routine that outputs a single
	05505	;			ry) to the device specified in
	05506 05507	;		ice table.	have been re-entrants because
	05508	;			have been re-entrant; however numbers. Therefore, to use it
	05509	,			pts enabled, the instruction
I	05510	;		e must be:	
	05511	;			
	05512	;			upts off
	05513	;		CALL Output\$Data\$Byt	e



## 266 The CP/M Programmer's Handbook

	05514	;	EI	;Inte	rrupts on
	05515	;			
	05516		ure to do thi	s may cause	involuntary re-entrance.
	05517	;			
	05518	; Entr	y parameters		
	05519	;			
	05520	;	C = chara	cter to be c	uteut
	05521		DE -> dev:	ice table	
	05522				
	05523	Output\$Data\$	Bytes		
0826 C5	05524	PUSH			
0827 210200	05525	LXI			;Save registers
			H, DT\$Outpu	lt#Ready	;Get output ready status mask
082A 19	05526	DAD	D		
082B 46	05527	MOV	B, M		
082C 210000	05528	LXI	H,DT\$Statu	is\$Port	;Get status port number
082F 19	05529	DAD	D		
0830 7E	05530	MOV	Α, Μ		
0831 323508	05531	STA	ODB#Statu:	\$Port	Store in instruction below
	05532	ODB\$Wait\$unt	ilsReady:		fotore in instruction below
	05533				
0834 DB	05534	DB	IN		- David address
0004 00	05535	ODB\$Status\$P			;Read status
0835 00	05536	DB	0;	Set up	in instruction above
	05537		_		
0836 AO	05538	ANA	B		;Check if ready for output
0837 CA3408	05539	JZ		intil\$Ready	; No
083A 210100	05540	LXI	H, DT\$Data\$		;Get data port
083D 19	05541	DAD	D		· · - · - ·
083E 7E	05542	MOV	Ã.M		
083F 324408	05543	STA	ODB\$Data\$F	Port	Store in instruction below
				ort	
0842 79	05544	MOV	A,C		;Get character to output
	05545				
0843 D3	05546	DB	OUT		
	05547	ODB\$Data\$Port	t:		
0844 00	05548	DB	0 :<	Set up	in instruction above
	05549				
0845 C1	05550	POP	в		Restore registers
0846 C9	05551	RET	D		AVESTOLE LEGISCELS
0040 07		. NET			
	05552	,			
	05700	;#			
	05701	;			
	05702	7 7			
		; ; ; Input	t status routi	ne	
	05702	; ; ; Inpu; ;	t status routi	ne	
	05702 05703 05704	1			in the A register indicating whether
	05702 05703 05704 05705	) ) This	routine retur	ns a value	in the A register indicating whether
	05702 05703 05704 05705 05706	; ; This ; one (	routine retur or more data c	ns a value haracters i	s/are waiting in the input buffer.
	03702 05703 05704 05705 05706 05707	7 This 7 One 0 7 Some	routine retur or more data c products, suc	ns a value haracters i h as Micros	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead
	03702 05703 05704 05705 05706 05707 05708	; ; This ; one ; Some ; by co	routine retur or more data c products, suc onstantly "gob	ns a value haracters i h as Micros bling" char	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming
	05702 05703 05704 05705 05706 05707 05708 05709	This cone Some Contine	routine retur or more data c products, suc onstantly "gob rol-S, -Q or -	ns a value haracters i h as Micros bling" char C has been	s/are waiting in the input buffer. Sft BASIC, defeat normal type-ahead acters in order to see if an incoming "eccived. In order to preserve
	05702 05703 05704 05705 05706 05707 05708 05709 05710	J     This       J     One       J     Some       J     Some       J     Dy cold       J     Conti       J     type	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t	ns a value haracters i h as Micros bling" char. C has been hese circum	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return
	03702 05703 05704 05705 05706 05707 05708 05709 05710 05711	; ; one of ; Some ; by co ; Conti ; type ; can,	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option	ns a value haracters i h as Micros bling" char C has been hese circum selected by	stare waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05711 05712	This This Some Some Dy co Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Coni	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t <b>as an</b> option he input buffe	ns a value haracters i h as Micros bling" char. C has been hese circum selected by r contains	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	03702 05703 05704 05705 05706 05707 05708 05709 05710 05711	This This Some Some Dy co Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Coni	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option	ns a value haracters i h as Micros bling" char. C has been hese circum selected by r contains	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05711 05712	This This Some Some Dy co Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Conti Coni	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t <b>as an</b> option he input buffe	ns a value haracters i h as Micros bling" char. C has been hese circum selected by r contains	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05709 05710 05711 05712 05713 05714	Image: This is a constraint of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second of the second	routine retur or more data of products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in	ns a value haracters i h as Micros bling" char. C has been hese circum selected by r contains	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05713 05714 05715	<pre>     This     This     cone (     Some     Some     conti     type     cont     if ti     Micro     f     Entr: </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t <b>as an</b> option he input buffe	ns a value haracters i h as Micros bling" char. C has been hese circum selected by r contains	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05711 05712 05713 05714 05715	<pre>     This     one (     Some (     Some (     Some (     Cont)     Cont)     Cont(     type,     can,     if th     Micro     f     Entr:     f </pre>	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters	ns a value tharacters i th as Micros bling" char C has been hese circum selected by cr contains to allowing	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05711 05712 05713 05714 05715 05716 05717	<pre>     This     This     cone (     Some     Some     conti     type     cont     if ti     Micro     f     Entr: </pre>	routine retur or more data of products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in	ns a value tharacters i th as Micros bling" char C has been hese circum selected by cr contains to allowing	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05711 05712 05713 05714 05715 05716	<pre>3 This 3 One of 5 Some 5 Dy or 5 Conti 5 Can, 5 if ti 7 Micro 7 7 8 Entry 5 8</pre>	routine retur or more data c products, suc onstantly "gob rol-S, -@ or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi	ns a value tharacters i th as Micros bling" char. C has been hese circum selected by cr contains to allowing	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05713 05714 05715 05716 05717 05718	<pre>3 This 3 One of 5 Some 5 Dy or 5 Conti 5 Can, 5 if ti 7 Micro 7 7 8 Entry 5 8</pre>	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters	ns a value tharacters i th as Micros bling" char. C has been hese circum selected by cr contains to allowing	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05709 05709 05710 05711 05712 05713 05714 05715 05716 05717 05718	<pre>3 This 3 One of 5 Some 5 Dy or 5 Conti 5 Can, 5 if ti 7 Micro 7 7 8 Entry 5 8</pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters	the a value tharacters i th as Micross bling" char. C has been these circum selected by r contains to allowing to allowing	stare waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead.
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05713 05714 05715 05716 05717 05718	<pre>3 This 3 One of 5 Some 5 Dy or 5 Conti 5 Can, 5 if ti 7 Micro 7 7 8 Entry 5 8</pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters	the a value tharacters i th as Micross bling" char. C has been these circum selected by r contains to allowing to allowing	s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools
	05702 05703 05704 05705 05706 05707 05709 05709 05710 05711 05712 05713 05714 05715 05716 05717 05718	<pre>     This     This     cone (     Some     Some     cont     type     cont     type     cont     fif th     Micro      Entry      Exit      Exit </pre>	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i	the a value tharacters i th as Micross bling" char. C has been these circum selected by r contains to allowing to allowing	stare waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead.
	05702 05703 05704 05705 05706 05707 05706 05707 05709 05710 05712 05712 05713 05714 05715 05716 05716 05719 05720 05721	This This Some of Some Some Cont Cont Cont Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Cont Type Cont Type Cont Type Cont Cont Cont Cont Cont Cont Cont Cont	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i	tharacters i tharacters i th as Micross bling" char C has been hese circum selected by tr contains to allowing ce table	stare waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead.
	05702 05703 05704 05705 05706 05707 05709 05710 05711 05712 05713 05713 05715 05716 05716 05716 05719 05720 05720 05721 05722	This This Some of Some Some Cont Cont Cont Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Type Cont Cont Type Cont Type Cont Type Cont Cont Cont Cont Cont Cont Cont Cont	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i	tharacters i tharacters i th as Micross bling" char C has been hese circum selected by tr contains to allowing ce table	stare waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead.
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05713 05714 05715 05716 05717 05718 05717 05720 05720 05722 05723 05724	This cont cont cont cont cont cont cont cont	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i bu	tharacters i tharacters i th as Micross bling" char C has been hese circum selected by tr contains to allowing ce table	stare waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead.
	05702 05703 05704 05705 05706 05707 05709 05709 05710 05712 05713 05714 05715 05714 05715 05716 05716 05719 05720 05720 05720 05722 05723	<pre>     This     This     Cone     Some     Some     Conting     type     can,     fift     Micro     Fift     Fift     Fift     Some     Some</pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i bu	ths a value tharacters i th as Micross bling" char: C has been these circum selected by r contains to allowing to allowing to allowing to allowing	stare waiting in the input buffer. off BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user,.return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input
	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05713 05714 05715 05716 05717 05717 05717 05718 05720 05720 05722 05722 05722 05723	<pre>     This     This     Tone d     Some     Some     Dy cd     Conti     type:     Can,     if th     Micro     f     Entr:     f     Exit     f     Exit     c     Exit     LXI </pre>	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe cooft BASIC in y parameters DE -> devi parameters A = 000H i bu atus: H,DT\$Statu	ths a value tharacters i th as Micross bling" char: C has been these circum selected by r contains to allowing to allowing to allowing to allowing	stare waiting in the input buffer. off BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input scheck if fake mode enabled
084A 19	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05712 05713 05714 05715 05716 05717 05716 05717 05720 05722 05722 05722 05722 05722 05724 05725 05726	<pre>     This     Tohis     cone     Some     by cc     conti     type     can,     if th     Micro      Entry      Exit      f      Get\$Input\$St.     LXI     DAD </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffg osoft BASIC in y parameters DE -> devi parameters A = 000H j bu atus: H, DT\$Statu D	ths a value tharacters i th as Micross bling" char: C has been these circum selected by r contains to allowing to allowing to allowing to allowing	stare waiting in the input buffer. off BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user,.return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input ;Check if fake mode enabled ;HL -> status byte in table
084A 19 084B 7E	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05713 05714 05715 05716 05717 05717 05717 05718 05720 05720 05722 05722 05722 05723	<pre>     This     This     Some     Some     by cc     Conti     type     Cent;     Conti     type     Cent;     fit     Micro     F     Entr:     Some     Exit     Some     LXI     DAD     MOV </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H j bu atus: H, DT\$Statu D A,M	ins a value tharacters i th as Micross bling" char C has been hese circum selected by tr contains tr contains to allowing ce table f no charac ifer	<pre>s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input ;Check if fake mode enabled ;HL -&gt; status byte in table ;Get status byte</pre>
084A 19 084B 7E	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05712 05713 05714 05715 05716 05717 05716 05717 05720 05722 05722 05722 05722 05722 05724 05725 05726	<pre>     This     Tohis     cone     Some     by cc     conti     type     can,     if th     Micro      Entry      Exit      f      Get\$Input\$St.     LXI     DAD </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffg osoft BASIC in y parameters DE -> devi parameters A = 000H j bu atus: H, DT\$Statu D	ins a value tharacters i th as Micross bling" char C has been hese circum selected by tr contains tr contains to allowing ce table f no charac ifer	stare waiting in the input buffer. off BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user,.return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input ;Check if fake mode enabled ;HL -> status byte in table
084A 19 084B 7E 084C E601	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05712 05715 05716 05717 05717 05720 05720 05722 05723 05724 05725 05726 05727 05728	<pre>     This     This     Tone of     Some     Some     by cc     cont     type     can,     if tt     Micro      f     Entr:     f      Exit     f      Cet\$Input\$St LXI     DAD     MOV     ANI </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i bu atus: H,DT\$Statu D, A,M DT\$Fake\$T}	ins a value tharacters i th as Micross C has been these circum selected by tr contains to allowing the table of no charac offer	<pre>s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming 'eccived. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input ;Check if fake mode enabled ;HL -&gt; status byte in table ;Get status byte ;Isolate status bit</pre>
084A 19 084B 7E 084C E601	05702 05703 05704 05705 05706 05707 05709 05709 05710 05712 05713 05713 05715 05716 05716 05716 05716 05720 05721 05722 05723 05724 05725 05724 05725 05724 05725 05728 05729 05729 05729	<pre>     This     This     Some     Some     by cc     Conti     type     Cent;     Conti     type     Cent;     fit     Micro     F     Entr:     Some     Exit     Some     LXI     DAD     MOV </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H j bu atus: H, DT\$Statu D A,M	ins a value tharacters i th as Micross C has been these circum selected by tr contains to allowing the table of no charac offer	<pre>s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input ;Check if fake mode enabled ;HL -&gt; status byte in table ;Get status byte</pre>
084A 19 084B 7E 084C E601	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05713 05714 05715 05716 05717 05717 05718 05719 05720 05722 05722 05722 05722 05722 05722 05722 05722 05722 05722 05725 05726 05727 05728 05729 05729 05729	<pre>     This     This     Tone of     Some     Some     by cc     cont     type     can,     if tt     Micro      f     Entr:     f      Exit     f      Cet\$Input\$St LXI     DAD     MOV     ANI </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i bu atus: H,DT\$Statu D, A,M DT\$Fake\$T}	ins a value tharacters i th as Micross bling thar C has been hese circum selected by tr contains to allowing ce table if no charac offer is\$2 peahead tatus	<pre>s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input ;Check if fake mode enabled ;HL -&gt; status byte in table ;Get status byte ;Isolate status bit ;Fake mode disabled</pre>
084A 19 084B 7E 084C E601	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05712 05713 05714 05715 05716 05715 05716 05717 05720 05721 05722 05723 05724 05725 05726 05726 05726 05727 05728 05729 05730	<pre>     This     This     Tone of     Some     Some     by cc     cont     type     can,     if tt     Micro      f     Entr:     f      Exit     f      Cet\$Input\$St LXI     DAD     MOV     ANI </pre>	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i bu atus: H,DT\$Statu D, A,M DT\$Fake\$T}	the action of the second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second secon	<pre>s/are waiting in the input buffer. boft BASIC, defeat normal type-ahead boters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input ;Check if fake mode enabled ;HL -&gt; status byte in table ;Get status byte ;Isolate status bit ;Fake mode disabled mode only indicates data</pre>
084A 19 084B 7E 084C E601 084E CA5B08	05702 05703 05704 05705 05706 05707 05709 05709 05710 05712 05713 05713 05714 05715 05714 05715 05714 05715 05716 05717 05718 05720 05720 05721 05723 05725 05725 05726 05725 05726 05727 05728 05729 05729 05729	<pre>     This     This     Some     Some     by cc     Conti     type:     Can,     if th     Hierd     Entr:     f      Get*Input*St:     LXI     DAD     MOV     ANI     JZ </pre>	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i bu atus: H,DT\$Statu D A,M DT\$Fake\$Ty GIS\$True\$S	ins a value tharacters i th as Micross Ding" char C has been hese circum selected by r contains it allowing ce table (f no charac iffer is\$2 peahead tatus ; ;Fake ;read	<pre>stare waiting in the input buffer. oft BASIC, defeat normal type-ahead laters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input :Check if fake mode enabled ;HL -&gt; status byte in table ;Get status byte ;Isolate status byte ;Isolate status bit ;Fake mode disabled mode only indicates data y if control chars. in buffer</pre>
084A 19 084B 7E 084C E601 084E CA5B08	05702 05703 05704 05705 05706 05707 05708 05709 05710 05712 05712 05712 05713 05714 05715 05716 05717 05718 05717 05720 05722 05722 05722 05722 05722 05724 05725 05726 05727 05726 05727 05728 05729 05729 05731 05733	; This ; This ; Some ; Some ; by cc ; Cant; ; type ; can, ; if ti ; Micro ; ; Entr; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	routine retur or more data c products, suc onstantly "gob rol-S, -Q or - -ahead under t as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H j bu atus: H,DT\$Statu D A,M DT\$Fake\$Ty GIS\$True\$S	ins a value tharacters i th as Micross Ding" char C has been hese circum selected by r contains it allowing ce table (f no charac iffer is\$2 peahead tatus ; ;Fake ;read	<pre>s/are waiting in the input buffer. oft BASIC, defeat normal type-ahead acters in order to see if an incoming 'eccived. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input fcheck if fake mode enabled ;HL -&gt; status byte in table ;Get status byte ;Isolate status bit ;Fake mode disabled mode only indicates data ; fc control chars. in buffer ;Check if any control characters</pre>
0847 210F00 084A 19 084B 7E 084C E601 084E CA5B08 0851 211C00 0855 AF	05702 05703 05704 05705 05706 05707 05709 05709 05710 05712 05713 05713 05714 05715 05714 05715 05714 05715 05716 05717 05718 05720 05720 05721 05723 05725 05725 05726 05725 05726 05727 05728 05729 05729 05729	<pre>     This     This     Some     Some     by cc     Conti     type:     Can,     if th     Hierd     Entr:     f      Get*Input*St:     LXI     DAD     MOV     ANI     JZ </pre>	routine retur or more data c products, sud onstantly "gob rol-S, -Q or - -ahead under as an option he input buffe osoft BASIC in y parameters DE -> devi parameters A = 000H i bu atus: H,DT\$Statu D A,M DT\$Fake\$Ty GIS\$True\$S	ins a value tharacters i th as Micross Ding" char C has been hese circum selected by r contains it allowing ce table (f no charac iffer is\$2 peahead tatus ; ;Fake ;read	<pre>stare waiting in the input buffer. oft BASIC, defeat normal type-ahead laters in order to see if an incoming received. In order to preserve stances, the input status return the user, return "data waiting" only a Control-S, -Q or -C. This fools type-ahead. ters are waiting in the input :Check if fake mode enabled ;HL -&gt; status byte in table ;Get status byte ;Isolate status byte ;Isolate status bit ;Fake mode disabled mode only indicates data y if control chars. in buffer</pre>

Figure 8-10. (Continued)

0856 E	RA	05737	ORA	м	;Set flags according to count
0857 0		05738	RZ		Return indicating zero
		05739	GIS\$Data\$Rea	dv.	proton indicating itro
0858 A	AF	05740	XRA		;Cheap O
0859 3		05741	DCR	A	;Set A = OFFH and flags NZ
085A C		05742	RET	-	Return to caller
0,000		05743			filetani to carrer
		05744	GIS\$True\$Sta	11151	1
		05745	010111 001010		, True status, based on any characters
		05746			;ready in input buffer
085B 2	2ASDOF	05747	LHLD	CB#Forced#Input	Check if any forced input waiting
		05748	MOV	A, M	;Get next character of forced input
085E 7 085F B	87	05749	ORA	A	Check if nonzero
0860 0		05750	JNZ	GIS\$Data\$Ready	Yes, indicate data waiting
0000 0	220000	05751	0112	OIS#Data#ReaDy	, tes, indicate data waiting
0863 2	211900	05752	LXI	H,DT\$Character\$	Count ;Check if any characters
0866 1		05753	DAD	n	; in buffer
0867 7		03754	MOV	Å, M	;Get character count
0868 8		05755	ORA	A	yber character count
0869 0		05756	RZ	M	;Empty buffer, A = 0, Z-set
					tempty buffer, $A = 0$ , 2-set
086A C	232808	05757	JMP	GIS\$Bata\$Ready	
		03758	1		
		05759	7		
		05900	; #		
		05901	1	• / ··· • • ··· • ···	
		05902		time clock process	ing
		05903		• • • • •	
		05904			to the RTC\$Interrupt routine each time
		05905			ks. The tick count is downdated to see
		05906			s elapsed. If so, the ASCII time in
		05907		configuration block	is updated.
		05908	;		
		05909	; With	each tick, the wat	chdog count is downdated to see if control
		05910	; must	be "forced" to a p	reviously specified address on return
		05911	; from	the RTC interrupt.	The watchdog timer can be used to pull
		05912	; cont	rol out of what wou	ld otherwise be an infinite loop, such
		05913		aiting for the prin	
		05914	7		
		05915	;		
		05916	; Set	watchdog	
		05917	;	-	
		05918		is a noninterrupt	level subroutine that simply sets the
		05919	; watc	hdog count and addr	155
		05920	•	-	
		05921	•	y parameters	
		05922	;	, ,	
		05923	,	BC = number of	lock ticks before watchdog should
		05924	,	"time ou	
		05925	, 1		which control will be transferred when
		05926	1		times out
		05927	;		
		05928	; Set\$Watchdog	•	
086D F	- 3	03928	Set#Watchdog DI	•	Avoid interference from interrupts
086E 2		05930	SHLD	RTC\$Watchdog\$Ad	
0865 2		05930	MOV		aress joel Quuress
0872 6		05931	MOV	H, B	
				L,C	
0873 2		05933	SHLD	RTC\$Watchdog\$Co	int ;Set count
0876 F		05934	EI		
0877 C	<i>.</i> 9	05935	RET		
		05936	;		
		05937	;		
		06000	;#		
		06001	;		
		06002			l is received here each time the
		06003			time clock ticks
		06004	RTC\$Interrup		
0878 F		06005	PUSH		Save other registers
0879 2		06006	SHLD	PI\$User\$HL	Switch to local stack
087C 2		06007	LXI	н, о	
087F 3		06008	DAD	SP	;Get user's stack
		06009	SHLD		Save it
		06010	LXI	SP.PI\$Stack	Switch to local stack
0880 2					ywaten tw total state
0880 2 0883 3		06011	Dilen		
0880 2 0883 3 0886 C	:5	06011	PUSH		
0880 2 0883 3	:5	06012	PUSH		
0880 2 0883 3 0886 C	5 )5				;Downdate tick count

Figure 8-10. (Continued)

0888	2 35	06015		BCR	м		
	C2B008	06016		JNZ	RTC\$Check\$Watcl	dee.	:Is not at 0 yet
0000	. 020000	06017		CANZ	Ricecheckenater	1009	;One second has elapsed so
0005	3ABD00	06018		LDA	RTC\$Ticks\$per\$5		; reset to original value
		06019		MOV	M.A	second	; reset to original value
0892				MUV	m, A		
		06020					;Update ASCII real time clock
	11A10F	06021		LXI	D, Time\$in\$ASCI		;DE -> 1 character after ASCII time
0896	21BD00	06022		LXI	H,Update\$Time\$	End	;HL ~> 1 character after control table
		06023	RTC\$Upd	late\$Digi	t:		
0899		06024		DCX	D		;Downdate pointer to time in ASCII
089A	2B	06025		DCX	н		Downdate pointer to control table
0895	7E	06026		MOV	A, M		;Get next control character
0890	B7	06027		ORA	A		Check if end of table and therefore
0890	CAB008	06028		JZ	RTC\$Clock\$Updat	ted	; all digits of clock updated
	FA9908	06029		JM	RTC\$Update\$Dig:		Skip over ":" in ASCII time
0843		06030		LDAX	n n		;Get next ASCII time digit
0844		06031		INR	Ă		;Update it
		06032			n		
0845	12	06032		STAX CMP	M		; and store it back
0846	BE						Compare to maximum value
	C2B008	06034		JNZ	RTC\$Clock\$Upda	led	No carry needed so update complete
	3E30	06035		MVI	A, 101		Reset digit to ASCII O
08AC		06036		STAX	D		; and store back in ASCII time
OSAD	C39908	06037		JMP	RTC\$Update\$Dig:	it	;Go back for next digit
		06038	1				
1		06039		ck\$Updat			
·		06040		ck\$Watch			
0880		060	041	LHLD	RTC\$Watchdog\$Co	ount	;Get current watchdog count
0883		06042		DCX	н		;Downdate it
0884	70	06043		MOV	А,Н		Check if it is now OFFFFH
0885		06044		ORA	A		
	FACBOS	06045		JM	RTC\$Dog\$Not\$Set		;It must have been 0 beforehand
0889	B5	06046		ORA	L		Check if it is now 0
08BA	C2C808	06047		JNZ	RTC\$Dog\$NZ		;No, it is not out of time
		06048					
l		06049					;Watchdog time elapsed, so "call"
		06050					; appropriate routine
	210508	06051		LXI		D - 4	
0800		06052		PUSH	H,RTC\$Watchdog1	Return	;Set up return address
					••		; ready for return
	2AC100	06053		LHLD	RTC\$Watchdog\$Ac	dress	;Transfer control as though by CALL
08C4	E9	06054		PCHL			
		06055	RTC\$Wat	chdog\$Re	turni		<pre>\$Control will come back here from</pre>
		06056					the user's watchdog routine
0805	C3CB08	06057		JMP	RTC\$Dog\$Not\$Set		Behave as though watchdog not active
		06058					
<b>I</b>		06059	RTC\$Dog	\$NZ :			
0808	22BF00	06060		SHLD	RTC\$Watchdog\$Co	unt	Save downdated count
		06061	RTC\$Dog	\$Not\$Set	1		; (Leaves count unchanged)
0808	3E20	06062		MVI	A, IC\$EOI		Reset the interrupt controller chip
0800	0308	06063		OUT	IC\$0CW2\$Port		
		06064					
OBCF	D1	06065		POP	D		Restore registers from local stack
0800		06065		POP	B		Avertote LeArriets (LOW TOCAT RIGCK
	248422	06067					- Culture hands to second a stands
0801		06067		LHLD SPHL	PI\$User\$Stack		;Switch back to user's stack
	248622	06069		LHLD	PI\$User\$HL		;Recover user's registers
08D8		06070		POP	PSW		
08D9		06071		EI			Re-enable interrupts
08DA	C9	06072		RET			
		06073.	;				
		06200	:#				
		06201	;				
		06202	3	Shift HL	. Right one bit		
		06203	;				
		06204	SHLR				
OBDB	B7	06205		ORA	A	;Clear	Carry
OBDC		06206		MOV	A,H	Bet MS	
OBDD		06207		RAR			set from previous carry
1		06208					goes into carry
OBDE	47	06209		MOV	H, A		ifted MS byte back
OBDE		06210		MOV	A,L	;Get LS	
0850		06211		RAR	nr -		= bit 0 of MS byte
				MOV			
08E1 08E2		06212			L,A	Frut Da	ck into result
	U7	06213 06214		RET			
VULA							
VOLA							
		06215	;				
COLL			; ;#				

Figure 8-10. (Continued)

0630	Hig	h level diskette drivers	· · · · · · · · · · · · · · · · · · ·
0630	2 ;		
0630: 0630-		se drivers perform the f	ollowing functions:
0630	5 j SEL		disk and return the address of
0630/ 0630		the appropriate dis TRK Set the track number	k parameter header r for the next read or write
0630			er for the next read or write
0630			rite) address for the next read or write
0631		TRAN Translate a logical	sector number into a physical
0631 0631		E Set the track to O be on Track O	so that the next read or write will
0631		De on track o	
0631	l i In	addition, the high level	drivers are responsible for making
0631	5; the	5 1/4" floppy diskettes	that use a 512-byte sector appear
0631/ 0631			a 128-byte sector. They do this g code. This blocking/deblocking
0631			etail later in this listing,
0631	9; jus	t prior to the code itse	1f.
0632			
0632			
0632		k parameter tables	
0632	4 ;		
0632			these describe the physical
0632			drives. In this example BIOS, drives; standard single-sided,
0632	3; sin	gle-density 8", and doub	le-sided, double-density 5 1/4"
0632		i-diskettes.	
0633 0633		atandard OV distates d	o not need to use the blocking/
0633			1/4" drives do. Therefore an additional
0633	3 ; byt		the disk parameter block to
0633			each logical disk's physical
0633		kette type 15, and wheth	er or not it needs deblocking.
0633			
0633		k definition tables	
0633			
0634			eter headers, with one entry d disk parameter blocks with
0634		her one parameter block	per logical disk, or the same
0634		ameter block for several	logical disks.
0634 0640			
0640			
0640		ter\$Headers:	;Described in Chapter 3
0640			-1. 4. 200 4240 -32-1-244-1
0640 8E3 AE09 0640		Floppy\$5\$Skewtable	sk A: (5 1/4" diskette) ;5 1/4" skew table
BE5 0000000000640		0,0,0	Reserved for CP/M
BEB 8022 0640		Directory#Buffer	<b></b>
BED 3409 0640 BEF 8023 0640		Floppy\$5\$Parameter\$ Disk\$A\$Workarea	Block
8F1 1024 0641		Disk\$A\$Workarea Disk\$A\$A1location\$V	ector
0641	1 ,		
0641			sk Br (5 1/4" diskette)
BF3 AE09 0641		Floppy\$5\$Skewtable 0.0.0	;Shares same skew table as A: ;Reserved for CP/M
8FB B022 0641		Directory\$Buffer	;Reserved for CP/m ;Shares same buffer as A:
8FD 3409 0641		Floppy\$5\$Parameter\$	
0641 0641		Disk\$B\$Workarea	;Private work area
0901 2624 0641		Disk\$B\$Allocation\$V	Private allocation vector
0641		ilosical di	sk C: (8" floppy)
0443	1 DW	Floppy\$8\$Skewtable	;8" skew table
0642 0903 F609 0642	2 DW	0,0,0	Reserved for CP/M
0903 F609 0642 0905 00000000000642		Directory\$Buffer	;Shares same buffer as A:
903         F609         0642           905         00000000000642           908         B022         0642	3 DW		
0903         F609         0642           0903         0000000000642         000000000642           0908         B022         0642           0908         B022         0642           0908         0642         0642	3 DW 4 DW	Floppy\$8\$Parameter\$	
0903         F609         0642           0905         0000000000642         0908           0908         8022         0642           0909         4409         0642           0906         7623         0642	3 DW 4 DW 5 DW		;Private work area
0903 F609 0642 9905 000000000642 090B B022 0642 090D 4409 0642 090F F023 0642 0911 3C24 0642 0642	3 DW 4 DW 5 DW 6 DW 7 ;	Floppy\$8\$Parameter\$ Disk\$C\$Workarea Disk\$C\$Allocation\$V	Private Work area Vector Private allocation vector
0903 F609 0642 0905 000000000642 0908 B022 0642 0908 F023 0642 090F F023 0642 0911 3C24 0642 0642 0642	3 DW 4 DW 5 DW 6 DW 7 ; 8	Floppy\$8\$Parameter\$ Disk\$C\$Workarea Disk\$C\$Allocation\$V ;Logical di	;Private work area Vector ;Private allocation vector Lsk D: (S" floppy)
0903 F609 0642 0905 000000000642 090B B022 0642 090D 4409 0642 090F F023 0642 0911 3C24 0642 0642	3 DW 4 DW 5 DW 6 DW 7 ; 8 9 DW	Floppy\$8\$Parameter\$ Disk\$C\$Workarea Disk\$C\$Allocation\$V	Private Work area Vector Private allocation vector

Figure 8-10. (Continued)

						. <u> </u>		
091D	4409	06432		DW	Floopvs	8\$Paramet	ersBlock	;Same DPB as C:
	0024	06433		DW		Workarea		Private work area
0921	5B24	06434		DW		Allocatio	n\$Vector	Private allocation vector
		06435						
		06436				<i>i</i> Logical	disk M:	(memory disk)
		06437	M\$Disk\$	DPH1				
	0000	06438		DW	0			;No skew required
	000000000			DW	0,0,0			Reserved for CP/M
0928	B022	06440		DW	Director	ry\$Buffer		
092D	5409	06441		DW	M#Disk#F	Parameter	\$Block	
092F	0000	06442		DW	0			Disk cannot be changed, therefore
		06443						1 no work area is required
0931	7A24	06444		DW	M#Disk##	Allocatio	n\$Vector	
		06445	;					
		06446	;					
		06447	;	Equates	for disk	k paramet	er block	
		06448	;					
		06449		Disk Typ	Pes			
		06450	;					
0001	=	06451	Floppy\$	5	EQU	1	5 1/4"	mini floppy
0002	=	06452	Floppy\$8		EQU			PY (SS SD)
0003		06453	M\$Disk		EQU		Memory	
		06454	;					
		06455	;	Blocking	g/deblock	king indi	etor	
		06456	;			·		
0080	-	06457	Need\$Deb	locking	EQU	1000\$000	DB	Sector size > 128 bytes
		06458	3					
		06600	2 <b>H</b>					
		06601	;					
		06602	3	Disk par	ameter b	locks		
		06603	,					
		06604	;	5 1/4"	nini flop	voc		
		06605	÷					
		06606	•				Extra b	yte prefixed to indicate
		06607						type and blocking required
0933	81	80660		DB	Flopey\$5	5 + Need\$1	eblocki	
		06609						ameter block has been amended
		06610				-		flect the new layout of one
		06611				1	track	per diskette side, rather
		06612				1	than v	viewing one track as both
		06613				1		on a given head position.
		06614				1		also been adjusted to reflect
		06615				1	one "r	new" track more being used for
		06616				1	the Cl	P/M image, with the resulting
		06617						e in the number of allocation
		06618						s and the number of reserved
		06619					: track	5.
		06620	Floppy\$5		ter\$Block			
0934		06621		DW	36			e sectors per track
0936		06622		DB	4		Block s	
0937		06623		DB	15		Block m	
0938		06624		DB	1		Extent	
093 <b>9</b>	ABOO	06625		DW	171			allocation block number
093B		06626		DW	127			of directory entries - 1
093D		06627		DB	1100\$000			for reserving 1 alloc. block
093E	00	06628		DB	0000\$000	DOB :	for f	ile directory
093F	2000	06629		DW	32			anged work area size
0941		06630		DW	3			of tracks before directory
		06631	;					·
		06632						
		06633	;	Standard	1 8" Flop	эру		
		06634				1	Extra b	yte prefixed to DPB for
		06635				1	this v	ersion of the BIOS
0943	02	06636		DB	Floppy\$8			es disk type and the fact
		06637					; that n	o deblocking is required
		06638			ter\$Block			
0944		06639		DW	26			per track
0946	03	06640		DB	3		Block sl	
0947	07	06641		DB	7		Block ma	
0948	00	06642		DB	0		Extent #	
0949		06643		DW	242			allocation block number
094B		06644		DW	63			of directory entries - 1
094D		06645		DB	1100\$000		Bit map	for reserving 2 alloc. blocks
094E		06646		DB	0000\$000			ile directory
094F		06647		DW	16			anged work area size
0951	0200	06648		DW	2	1	Number (	of tracks before directory

<u> </u>				
	06649	;		
	06650	∳ M\$Disk		
	06651	1		
	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			<pre>y used to select the bank</pre>
0953 03	06660	DB	M\$Disk	;Type is M\$Disk, no deblocking
	06661	M\$Disk\$Paramete		an an mar Marsall Prack Arrests an
0954 COC		DW	192	;Sectors per "track". Each track is
	06663		•	; 24K of memory
0956 03	06664	DB DB	3 7	;Block shift (1024 byte allocation) :Block mask
0957 07	06665			
0958 00	06666	DB	0	;Extent mask ;Maximum allocation block number
0959 000		DW TW	192	;Number of directory entries -1
0958 3FC		DB	63 1100\$0000B	Bit map for reserving 2 allocation blocks
095D CO	06669			
095E 00	06670	DB	0000\$0000B	; for file directory Dick correct be abarred therefore no
095F 000	00 06671	DW	0	Disk cannot be changed, therefore no
	06672	<b>D</b> 11	•	; work area
0961 000		DW	0	No reserved tracks
	06674	1		
0004 =	06675	Number\$of\$Logic	al\$Disks	EQU 4
	06676	;		
	06800	;#		
	06801			1
	06802	SELDSK:	;Select disk	
	06803			D for drive A, 1 for B, etc.
	06804			rn the address of the appropriate
	06805			sk parameter header in HL, or 0000H
	06806			the selected disk does not exist.
0963 210	06807	LXI	; н,о	;Assume an error
				;Assume an error ;Check if requested disk valid
0966 79	06809 06810	MOV	A,C	tenery to leducated diak Adiio
0967 FEC		CPI	1M1 - 1A1	:Check if memory disk
0967 FEC		JZ	SELDSK#M#Disk	;UNECK IT MEMORY DISK ;Yes
V707 LAT	06812	01	SELDOKANADISK	\$ 1 % #
096C FEG		CPI	Number#of#Log	ical\$Disks
096E D0		RNC		Return if > maximum number of disks
0702 00	06815	PINC I		FINESHITE AT 2 HIMANAHIMAHI HADAUGI WE WASKS
096F 322		, STA	Selected\$Disk	Save selected disk number
570P 322	06818	oin	Serec (Enthrigh	Set up to return DPH address
0972 6F	06819	MOV	L,A	Make disk into word value
0973 260		MVI	H, O	FILLS WITH THE WORLD THEE
07/3 200	06821			sCompute offset down disk parameter
	06822			; header table by multiplying by
	06823			; parameter header length (16 bytes)
0975 29	06823	DAD	н	
0976 29	06825	DAD	н	;*4
0977 29	06826	DAD	H	;*8
0978 29	06827	DAD	н	1*16
0979 115		LXI	D,Disk\$Parame	
0979 11E 097C 19	06829	DAD	D, Diskerarenne D	IDE -> appropriate DPH
097D E5	06830	PUSH	Ĥ	;Save DPH address
5770 E3	06831	;		,
	06832	,		Access disk parameter block to
	06833			# extract special prefix byte that
	06834			identifies disk type and whether
	06835			; deblocking is required
	06836			• • • • • • • • • • • • • • • • • • •
		LXI	D, 10	Get DPB pointer offset in DPH
097E 110		DAD	D	DE -> DPB address in DPH
097E 110			Ē,M	;Get DPB address in DE
0981 19	06838	MOV		
0981 19 0982 5E	06838 06839	MOV INX	н	
0981 19 0982 5E 0983 23	06838 06839 06840		H D.M	
0981 19 0982 5E 0983 23 0984 56	06838 06839 06840 06841	INX MOV		;DE -> DPB
0981 19 0982 5E 0983 23	06838 06839 06840 06841 06842	INX		\$DE → DPB
0981 19 0982 5E 0983 23 0984 56	06838 06839 06840 06841	INX MOV	D, M	;DE -> DPB
0981 19 0982 5E 0983 23 0984 56 0985 EB	06838 06839 06840 06841 06842 06843 06843	INX MOV XCHG	D, M	
0981 19 0982 5E 0983 23 0984 56 0985 EB	06838 06839 06840 06841 06842 06843	INX MÖV XCHG SELDSK\$Set\$Disk	D,M \$Type: H	;DE -> DPB ;DE -> prefix byte ;Get prefix byte
0981 19 0982 5E 0983 23 0984 56 0985 EB	06838 06839 06840 06841 06843 06843 06844 06845 06845	INX MOV XCHG SELDSK\$Set\$Disk DCX	D, M \$Type:	;DE -> prefix byte

Figure 8-10. (Continued)

098A 32360A	06848	ST		ted\$Disk\$T	VDA		for use in 1	ou loval driver	
098D 7E	06849	MO		LEGADISKAT			opy of prefi	ow level driver	
098E E680	06850	AN		Deblocking					
							e deblockin		
0990 32350A	06851	ST		ted\$Disk\$D				ow level driver	
0993 E1	06852	PO			;Recover	UPH PO	ointer		
0994 C9	06853	RE	Ŧ						
	06854	;							
	06855	SELDSK\$M\$D				;M\$Disk	selected		
0995 212309	06856	LX	I H, M\$E	)isk\$DPH		;Return	) correct pa	rameter header	
0998 C38609	06857	JM	P SELDS	K\$Set\$Disk	\$Type	;Resume	normal pro-	cessing	
	06858	;							
	07000	;#							
	07001	1							
	07002	; Se	t logical t	rack for n	ext read	or writ	e		
	07003		· · · · · · · · · · · · · · · · · · ·						
	07004	SETTRK							
099B 60	07005	MO	∨ н,в		:Selecte	d track	in BC on e	ntry	
0990 69	07006	MO			, derecte		th bo on e		
099D 222E0A	07007	SH		ted\$Track	· Cave fo		level driver		
09A0 C9	07008	RE		LEOFIT ACK	Joave IC	37 IOW I	level uriver		
07H0 C7	07009								
		,							
	07100	; #							
	07101	;							
	07102		t logical s	ector for	next read	i or wri	ite		
	07103	7							
	07104	;							
	07105	SETSEC:			;Logical	l sector	' in C on en	try	
09A1 79	07106	MO							
09A2 32300A	07107	ST	A Selec	ted\$Sector	;Save fo	or low 1	level driver		
09A5 C9	07108	RE							
	07109	;							
	07200	; #							
	07201	1							
	07202		t diek DMA	(Input /Out	out) addr	ess for	next read	or write	
	07203			(Inpat/oat	Fat		next read	or write	
09A6 0000	07204	, DMA\$Addres	s: DW	0		;DMA ad	drace		
0748 0000	07204	Drimamuures	51 DW	v		JUMM ad	laress		
	07206	SETDMA:							
09A8 69	07208	MO					on entry		
					;Move to	HL to	save		
09A9 60	07208	MO			~ ~				
09AA 22A609	07209	SH		ddress	;Save to	or low i	level driver		
09AD C9	07210	RE	T						
	07211	;							
	07300	; #							
	07301	;							
	07302	; Tr	anslate log	ical secto	r number	to phys	sical		
	07303	;							
	07304	; Se	ctor transl	ation tabl	es				
	07305	; Th	ese tables	are indexe	d using t	the logi	ical sector :	number,	
	07306	; an	d contain t	he corresp	onding ph	nysical	sector numb	er.	
	07307	;		-					
	07308	Floppy\$5\$S	kewtable:	;Each p	hysical s	sector c	contains fou	r	
	07309				te sector				
	07310	:	Physi	cal 128b	Logical		Physica	1 512-byte	
09AE 00010203	07311	ĎB		,02,03	;00,01,0		0		
09B2 10111213	07312	DB		, 18, 19	;04,05,0		4	; ;	
09B6 20212223	07313	DB		3, 34, 35	;08,09,1		s	5	
09BA OCODOEOF	07314	DB		3, 14, 15	;12,13,1		3	) Head	
09BE 1C1D1E1F	07315	DB		9,30,31	;16,17,1		7	) 0	
09C2 08090A0B	07316	DB		2,10,11	;20,21,2		2	) V	
09C6 18191A1B	07317	DB		5,26,27	;24,25,2		6	Ś	
09CA 04050607		DB		5,06,07	;24,23,2			,	
09CE 14151617	07318 07319	DB					1	,	
070E 1413161/	07319		20,21	,22,23	;32,33,3	54,90	5	·	
0000 04050/05		;	A/ A-		. 34 . 7 . 7		~	•	
09D2 24252627	07321	DB		, 38, 39	;36,37,3		0	3	
09D6 34353637	07322	DB		3,54,55	;40,41,4		4	3	
09DA 44454647	07323	DB		7,70,71	;44,45,4		8	3	
09DE 30313233	07324	DB		9,50,51	;48,49,5		3	] Head	
09E2 40414243	07325	DB		5,66,67	;52,53,5	54,55	7	3 1	
09E6 2C2D2E2F	07326	DB	44,45	5,46,47	;56,57,5	58,59	2	3	
09EA 3C3D3E3F	07327	DB	60,61	,62,63	:60,61,6		6	3	
09EE 28292A2B	07328	DB		, 42, 43	164,65,6		1	3	
09F2 38393A3B	07329	DB		,58,59	;68,69,7		5	j	
	07330	,				• • -	•		
	07331								
	07332	Floppy\$8\$S	kewtable:	:Standa	rd 8" Dri	iver			
				,			·		

Figure 8-10. (Continued)

		A7999			A1 A2 A	3 04 05	04 07 09	00 10	Lenies] sectors
0054	01070D1319	07333		DB			06,07,08 05,11,17		Logical sectors ;Physical sectors
0760		07335		100	01,0/,1	3, 17, 23,	03, 11, 17	,23,03	providel sectors
		07336		;	11 12 1	2 14 18	16, 17, 18	19 20	Logical sectors
	090F150208			DB			14,20,26		Physical sectors
UMUU					09,15,2	1,02,00,	14,20,20	,00,12	FRYSICAL SECTORS
		07338		3	<b>a</b> 1 <b>a</b> 2 <b>a</b>		24	Logion	1 sectors
				1		3, 24, 25,			
OAOA	1218040A10			DB	18,24,0	4,10,16,	22	Physic	al sectors
		07341							
		07400	3 <b>#</b>						
		07401	;						
		07402	SECTRAN	1		Transl	ate logi	cal sect	or into physical
		07403				30n ent	ry, BC =	logical	sector number
		07404				<b>3</b> -	DE -	> approp	riate skew table
		07405				3			
		07406				ton exi	t, HL =	physical	sector number
0A10	EB	07407		XCHG				skew tab	
0A11		07408		DAD	в				sector number
0A12		07409		MOV	Ē. M				ector number
0A13		07410		MVI	н, о				-bit value
0A15		07411		RET					
0413	.,	07412	;						
		07500	; #						
		07501							
		07502	!						
			J LIONE -				ha		and dials to turnels O
		07503	HOME						cal disk to track O
		07504				Before	doing t	nis, a c	heck must be made to see
		07505				- 1 I I	ne physi	cal disk	buffer has information in
		07506				, it t	hat must	De writ	ten out. This is indicated by
		07507							uffer, that is set in the
		07508				; debl	ocking c	ode.	
		07509				;			
0A16	3A2COA	07510		LDA	Must#Wr	ite\$Buff	er		if physical buffer must
0A19	87	07511		ORA	A			; be w	ritten to a disk
OA1A	C2200A	07512		JNZ	HOME\$No	\$Write			
OA1D	322B0A	07513		STA	Data\$In	\$Disk\$Bu	ffer		indicate that buffer
		07514						🔅 is n	ow unoccupied
		07515	HOME\$No\$	Write:					
0A20		07516		MVI	C,0			;Set to	track O (logically,
	CD9809	07517		CALL	SETTRK			) no a	ctual disk operation occurs)
0A25	C9	07518		RET					
		07519							
		07520	1						
		07600	;*						
		07601	3	Data wri	itten to	or read	from th	e mini-f	loppy drive is transferred
		07602	í						ete track in length,
		07603		0 4 512	hytes	It is de	clared a	t the en	d of the BIOS, and has
		07604	1						e "hidden" in it.
		07605	:						
		07605	1	The blac	king/de	blocking	oode -*	tempts *	o minimize the amount
		07607	1						and track
		07608	1				he physi		
		07609	•	tearrent)	17.F#\$10	111 W 111 T		Cel Dutt 20-but-	er. CP/M PrestorP
			1	AT 4 744	u reque	st becur	s UT & L		CP/M "sector"
		07610	!						no disk access occurs
		07611	1						B-byte CP/M 'sector'
		07612	3						isk access will occur,
		07613	1	UNLESS 1	rne BDOS	indicat	es that	11 15 W	iting to the directory.
		07614	;					iate wri	te to disk of the entire
		07615	1	track in	n the ph	ysical b	utfør.		
		07616	;						
		07617	1						
0800		07618		on#Block		EQU	2048		
0009		07619	Physical	l\$Sec\$Per	#Track	EQU	9		ed to reflect a "new"
		07620							k is only one side of the
		07621						🕴 disk	
0200	*	07622	Physical	\$Sector	Size	EQU	512		s the actual sector size
		07623							the 5 1/4" mini-floppy diskett
		07624							diskettes and memory disk
		07625							128-byte sectors
		07626							e the physical disk buffer for
		07627						1 5 1/	4" diskettes
		07628	CPMesant	Per \$Phy	Lical.	EQU	Physics	18Sector	\$Size/128
0004	-					EQU			<pre>sical*Physical\$Sec\$Per\$Track</pre>
0004			COMAGe - 4						
0024	=	07629	CPM\$Sec						
0024	=	07629 07630	Bytes#Pe	er#Track		EQU	Physica	1\$Sec\$Pe	r\$Track*Physical\$Sector\$Size
0024	-	07629	Bytes\$Pe Sector\$P	er#Track			Physica		r\$Track*Physical\$Sector\$Size

······································					
	07633	1			
	07634				values handed over by the BDOS
	07635				is the write operation.
	07636				/unallocated indicates whether the
	07637 07638				to write to an unallocated allocation
	07639		1 610	CK (11 0)	nly indicates this for the first
	07640		; 128 ; tha	the set	ctor write), or to an allocation block ready been allocated to a file.
	07641				indicates if it wishes to write to
	07642			file di	
	07643		1		
0000 =	07644	Write\$Allocated	EQU	0	
0001 =	07645	Write\$Directory	EQU	1	
0002 =	07646	Write\$Unallocated	EQU	2	<pre>s&lt;== ignored for track buffering</pre>
	07647	3			
0A26 00	07648	Write\$Type:	DB	0	Contains the type of write
	07649				; indicated by the BDOS
	07650 07651	1			
	07652	; In\$Buffer\$Dk\$Trk;			bles for physical sector currently
	07652	INABUTTERADKAILKI			DistBuffer in memory
0A27 00	07654	In\$Buffer\$Disk:	DB	, <u>.</u>	<ol> <li>these are moved and compared</li> </ol>
0A28 0000	07655	In\$Buffer\$Track;	DW	ŏ	;) as a group, so do not alter
	07656			-	; these lines
0A2A 00	07657	In\$Buffer\$Disk\$Type:	DB	0	Disk type for sector in buffer
	07658	1			
0A2B 00	07659	Data#In#Disk#Buffer:	DB	0	#When nonzero, the disk buffer has
	07660			_	<pre>y data from the disk in it</pre>
0A2C 00	07661	Must\$Write\$Buffer:	DB	0	Nonzero when data has been written
	07662				<pre># into Disk\$Buffer but not yet</pre>
	07663				; written out to disk
	07664 07665	F Selected\$Dk\$Trk:			
	07666	Selectedankaliki		yvaria:	bles for selected disk, track and sector ected by SELDSK, SETTRK and SETSEC)
0A2D 00	07667	Selected\$Disk:	DB	0	;) These are moved and compared
0A2E 0000	07668	Selected#Track:	DW	ŏ	;) as a group so do not alter order
0012E 0000	07669		2.4	Ū	Fr as a group so do not after order
0A30 00	07670	Selected#Sector:	DB	0	Not part of group but needed here
	07671				•••••••••••••••••••••••
0A31 00	07672	Selected\$Physical\$Secto	or: DB	0	Selected physical sector derived
	07673				<pre># from selected (CP/M) sector by</pre>
	07674				<pre>shifting it right the number of</pre>
	07675				; bits specified by Sector\$Bit\$Shift
	07676				
	07677 07678	7			
0A32 00	07679	Disk\$Error\$Flag:	DB	0	Nonzero to indicate an error
UNDE UU	07680	araveti oi 4L16Ai	00	v	that could not be recovered
	07681				; by the disk drivers. The BDOS
	07682				; will output a "Bad Sector" message
0A33 00	07683	Disk\$Hung\$Flag:	DB	0	Nonzero if a watchdog timeout
	07684			-	1 occurs
0258 =	07685	Disk‡Timer	EQU	600	Number of 16.66 ms clock ticks
	07686				) for a 10 second timeout
	07687	7		-	
	07688			;Flags	used inside the deblocking code
	07689				
0A34 00	07690	Read\$Operation:	DB	0	Nonzero when a CP/M 128-byte
0495 00	07691	Selected#Disk#Deblock:	ne	0	; sector is to be read ;Nonzero when the selected disk
0A35 00	07692 07693	Setectedanizkan6010cki	08	0	; needs deblocking (set in SELDSK)
0A36 00	07693	Selected\$Disk\$Type:	DB.	0	Indicates 8" or 5 1/4" floppy or
0.000 00	07695			-	# MeDisk selected, (set in SELDSK)
	07696	,			
	07800	2 🗰			
	07801	1			
	07802	; Read in the 126	B-byte_C	P/M secto	or specified by previous calls
	07803	1 to Select Disk,	Set Tr	ack and S	Sector. The sector will be read
	07804		is speci	ried in t	the previous Set DMA Address call.
	07805	The second second second second			aine analous lavany than 130 buta-
	07806				sing sectors larger than 128 bytes, o "unpack" a 128-byte sector from
	07807 07808	; deblocking code : the physical se		e useu ti	U UNPACK & IZO-DYLE SECTOR FROM
	07809	READ:			
0A37 3A350A	07810		d\$Disk\$	Deblock	Check if deblocking needed
OA3A B7	07811	ORA A			; (flag was set in SELDSK call)

	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 til the actual data transfer as though
		07816 07817				was the first write to an unallocated
		07817 07818			y it r al	was the first write to an unallocated location block
OAGE	9501	07818		MVI	A.1	JINDICATION DIOCK
	32340A	07820		STA	A,1 Read\$Operation	is to be performed
0440	JEJTVA	07820		314	VEENAODEL ETTOIL	7 IS CODE PERIOTNED
0A43 :	3500	07822		MVI	A,Write#Allocated	Fake deblocking code into believing
OA45	32260A	07823		STA	Write\$Type	; that this is a write to an
		07824				; allocated allocation block
0448	C35C0A	07825		JMP	Perform#Read#Write	Use common code to execute read
		07826	1			
		07900	;*			
		07901	,	Write	a 128-byte sector from	the current DMA address to
		07902	i i		eviously selected disk,	
		07903	;	•		
		07904	1			1 have set register C to indicate
		07905		whethe	r this write operation	is to an already allocated allocation
		07906	;	block	(which means a preread	of the sector may be needed), or
		07907	i	to the	directory (in which ca	se the data will be written to the
		07908	;		mmediately).	
		07909	;			
		07910	,			take place immediately. In all other
		07911	,			from the DMA address into the disk
		07912	;	buffer	, and only be written o	ut when circumstances force the
		07913	;	transf	er. The number of physic	cal disk operations can therefore
		07914	3	be red	uced considerably.	
		07915	,			
		07916	WRITE:			
	3A350A	07917		LDA	Selected#Disk#Deblock	
OA4E		07918		ORA	A	; (flag set in SELDSK call)
OA4F	CA2A0B	07919		JZ	Write\$No\$Deblock	
		07920				• ·· · · · · · · · · · · · · · · · · ·
0A52		07921		XRA	<b>A</b>	Indicate that a write operation
	32340A	07922		STA	Read\$Operation	; is required (i.e NOT a read)
0A56		07923		MOV	A,C	Save the BDOS write type
0A57	E601	07924		ANI	1	<pre>s but only distinguish between </pre>
- ·		07925				s write to allocated block or
UA39	32260A	07926		STA	Write\$Type	# directory write
		07927	3			
		07928	7			
		08000	<b>; #</b>			
		08001	1	*De e d é l		to overwhe heath woods and
		08002 08003	Perform	iake soži		to execute both reads and
OASC	AF	08003		XRA		128-byte sectors.
	AF 32320A	08004		XKA STA	A JASSU Disk\$Error\$Flag j oc	me that no disk errors will
vnou	JEJEUM	08005		91A	DISKAELLOLALISG 1 OC	Lur
0440	3A300A	08005		LDA	Salastad&Sector +Conv	ert selected 128-byte sector
0463		08007		RAR		to physical sector by dividing by 4
0463		08009		RAR	, 10	re buildness sector of gratgrid of 4
0465		08010		ANI	3EH s Reno	ve any unwanted bits
	32310A	08011		STA	Selected#Physical#Sec	
vnu/		08012		0.4	water toot, in sich oec	
0466	21280A	08012		LXI	H.Data\$In\$Disk\$Buffer	Check if disk buffer already has
		08013		MOV	A,M	; data in it
OA4P	15			MVI	A, H M, 1	; Unconditionally indicate that
OA6D				114.1	117 8	; the buffer now has data in it)
0A6D 0A6E		08015				y the warrer now here the the ty
0A6E	3601	08016		ORA		•Nid it indeed have data in it?
0A6E 0A70	3601 B7	08016 08017		ORA	A Read&Track&into#Puffe	<pre>plid it indeed have data in it? r tNo. proceed to read a physical</pre>
0A6E 0A70	3601	08016 08017 08018		ORA JZ	A Read#Track#into#Buffe	r ;No, proceed to read a physical
0A6E 0A70	3601 B7	08016 08017 08018 08019				
0A6E 0A70	3601 B7	08016 08017 08018 08019 08020			3	r ;No, proceed to read a physical ; track into the buffer
0A6E 0A70	3601 B7	08016 08017 08018 08019 08020 08021			; ;The	r ;No, proceed to read a physical ; track into the buffer buffer does have a physical track
0A6E 0A70	3601 B7	08016 08017 08018 08019 08020 08021 08022			; ;The	r ;No, proceed to read a physical ; track into the buffer
0A6E 0A70 0A71	3601 87 CA870A	08016 08017 08018 08019 08020 08021 08022 08023		JZ	ş şThe şin ş	r ;No, proceed to read a physical y track into the buffer buffer does have a physical track it. Check if it is the right one
0A6E 0A70 0A71	3601 B7 CA870A 11270A	08016 08017 08018 08019 08020 08021 08022 08023 08023		JZ	; ;The ; in D,In\$Buffer\$Dk\$Trk	<pre>r ;No, proceed to read a physical</pre>
0A6E 0A70 0A71 0A74 0A77	3601 B7 CA870A 11270A 212D0A	08016 08017 08018 08019 08020 08021 08022 08023 08024 08025		JZ LXI LXI	3 ;The ; in D,In\$Buffer\$Dk\$Trk H,Selected\$Dk\$Trk	<pre>;No, proceed to read a physical ; track into the buffer buffer does have a physical track it. Check if it is the right one ;Check if track in buffer is the ; same as that selected earlier</pre>
0A6E 0A70 0A71 0A74 0A77 0A7A	3601 B7 CA870A 11270A 212D0A CDE10A	08016 08017 08018 08019 08020 08021 08022 08023 08023 08024 08025 08026		JZ LXI LXI CALL	; ;The ; in D,In\$Buffer\$Dk\$Trk H,Selected\$Dk\$Trk Compare\$Dk\$Trk	<pre>r ;No, proceed to read a physical y track into the buffer buffer does have a physical track it. Check if it is the right one ;Check if track in buffer is the y same as that selected earlier ;Compare ONLY disk and track</pre>
0A6E 0A70 0A71 0A74 0A77 0A7A	3601 B7 CA870A 11270A 212D0A	08016 08017 08018 08019 08020 08021 08022 08023 08024 08025 08025		JZ LXI LXI	3 ;The ; in D,In\$Buffer\$Dk\$Trk H,Selected\$Dk\$Trk	<pre>;No, proceed to read a physical ; track into the buffer buffer does have a physical track it. Check if it is the right one ;Check if track in buffer is the ; same as that selected earlier</pre>
0A6E 0A70 0A71 0A74 0A77 0A7A	3601 B7 CA870A 11270A 212D0A CDE10A	08016 08017 08018 08019 08020 08021 08022 08023 08024 08025 08026		JZ LXI LXI CALL	; ;The ; in D,In\$Buffer\$Dk\$Trk H,Selected\$Dk\$Trk Compare\$Dk\$Trk	<pre>;No, proceed to read a physical ; track into the buffer buffer does have a physical track it. Check if it is the right one ;Check if track in buffer is the ; same as that selected earlier ;Compare ONLY disk and track ;Yes, it is already in buffer</pre>
0A6E 0A70 0A71 0A74 0A77 0A7A	3601 B7 CA870A 11270A 212D0A CDE10A	08016 08017 08018 08019 08020 08021 08022 08023 08024 08025 08025		JZ LXI LXI CALL	; ;The ; in D,In\$Buffer\$Dk\$Trk H,Selected\$Dk\$Trk Compare\$Dk\$Trk	<pre>r ;No, proceed to read a physical y track into the buffer buffer does have a physical track it. Check if it is the right one ;Check if track in buffer is the y same as that selected earlier ;Compare ONLY disk and track</pre>

Figure 8-10. (Continued)

0483	87	08032	ORA	A	
	C4E50B	08033	CNZ		# must be written out first
0404	042000		UNZ	Write\$Physical	;Yes, write it out
		08034	7		
		08035	Read\$Track\$int	o\$Buffer:	
0A87	CDCEOA	08036	CALL	Set\$In\$Buffer\$D	k\$Trk ;Set in buffer variables from
		08037			<pre>selected disk, track</pre>
		08038			
		08039			
0494	CDEAOB	08040	<b>C41</b>		s buffer now
			CALL	Read\$Physical	Read the track into the buffer
OABD	MF	08041	XRA	A	Reset the flag to reflect buffer
OASE	322C0A	08042	STA	Must\$Write\$Buff	er ; contents
		08043	ş		
		08044	Track\$In\$Buffe	ri	\$Selected track and
		08045			; disk is already in the buffer
		08046			(Convert the selected CP/M (128-byte)
		08047			
		08048			<pre># sector into a relative address down # the buffer</pre>
0401	3A300A	08049	LDA		3 the buffer
				Selectedasector	:Get selected sector number
0A94	OF .	08050	MOV	L,A	Multiply by 128 by shifting 16-bit value
0A95	2600	08051	MVI	н, о	jleft 7 bits
0A97		08052	DAD	н	1* 2
0A98	29	08053	DAD	н	1 4 4
0A99		08054	DAD	H	** 8
0A9A		08055	DAD	н	1* 16
0A9B		08056	DAD	Ĥ	1× 32
0A9C		08057	DAD	H	
0A9D		08058	DAD	н	1# 64
0470	#.7			п	;* 128
A+05		08059	;		<b>.</b>
	11A40F	08060	LXI	D,Disk\$Buffer	#Get base address of disk buffer
OAA1	19	08061	DAD	D	#Add on sector number # 128
		08062			<pre>#HL -&gt; 128-byte sector number start</pre>
		08063			<pre># address in disk buffer</pre>
0AA2	EB	08064	XCHG		<b>;DE -&gt; sector in disk buffer</b>
0AA3	266609	08065	LHLD	DMA\$Address	JGet DMA address set in SETDMA call
0AA6		08066	XCHG	DINPHOUTESS	Yoet Dram address set in SciDna Call
VINO	60	08067	XCH0		#Assume a read operation, so
					DE -> DMA address
		08068			; HL -> sector in disk buffer
0AA7	0E10	08069	MVI	C,128/8	Because of the faster method used
		08070			to move data in and out of the
		08071			# disk buffer, (eight bytes moved per
		08072			<pre>: loop iteration) the count need only</pre>
		08073			t be 1/8 of normal
		08074			sAt this point,
		08075			C = loop count
		08076			) DE -> DMA address
		08077			JUC "/ DAM BOORESS
0440	343404		1 54		HL -> sector in disk buffer
	3A340A	08078	LDA	Read#Operation	Determine whether data is to be moved
OAAC		08079	ORA	A	; out of the buffer (read) or into the
OAAD	C2850A	08080	JNZ	Buffer\$Move	; buffer (write)
		08081			;Writing into buffer
		08082			;(A must be 0 get here)
OABO	30	08083	INR	<b>A</b>	;Set flag to force a write
	322C0A	08084	STA	Must\$Write\$Buffe	
OAB4		08085	XCHG	husten itespuite	r ; of the disk buffer later on. ;Make DE -> sector in disk buffer
VADA	~~	08086	AU10		
			-		; HL -> DMA address
		08087			
		08088	<b>;</b>		
		08089	Buffer\$Move:		
OAB5	CDF80A	08090	CALL	Move\$8	;Moves 8 bytes * C times from (HL)
		08091			; to (DE)
		08092			
		08093			:
OABS	3A260A	08094	LDA	Write\$Type	;If write to directory, write out
OABB		08095	CPI	WriteSDirectory	; buffer immediately
	343204	08096	LDA	Dieb&Error&Else	; Bet error flag in case delayed write or read
OACO		08097	RNZ	STAKARLU OLALIGA	
UACO	0		rinz		Return if delayed write or read
		08098			<b>)</b>
OAC1		08099	ORA	A	;Check if any disk errors have occured
OAC2	CO	08100	RNZ		;Yes, abandon attempt to write to directory
		08101			;
0AC3	AF	08102	XRA	A	Clear flag that indicates buffer must be
	322C0A	08103	STA	Must\$Write\$Buffe	r ; written out
	CDE50B	08104	CALL		Write buffer out to physical track
	3A320A	08105	LDA		Return error flag to caller
OACD		08106	RET		process error lang to talker
UNCD		08107			
		00107	,		

	08108	;		
	08109			
	08110	Set\$In\$Buffer\$	Dk\$Trk:	Indicate selected disk, track
	08111			now residing in buffer
OACE 3A2DOA	08112	LDA	Selected\$Disk	
0AD1 32270A	08113 08114	STA	In\$Buffer\$Disk	
OAD4 2A2EOA	08114	LHLD	Selected\$Track	
0AD7 22280A	08116	SHLD	In\$Buffer\$Track	
VND/ ZEZUVN	08117	anco	Inspanser strack	
OADA 3A360A	08118	LDA	Selected\$Disk\$7	ype ;Also reflect disk type
OADD 322AOA	08119	STA	In\$Buffer\$Disk\$	
	08120			
OAEO C9	08121	RET		
	08122	;		
	08123	<b>†</b>		• • • • • • • • • •
	08124 08125	Compare\$Bk\$Trk	:	Compares just the disk and track
OAE1 OE03	08125	MVI	С,З	; pointed to by DE and HL ;Disk (1), track (2)
UNEI UEUU	08127	Compare\$Dk\$Trk		DISK (I), (Fack (2)
OAE3 1A	08128	LDAX	n .	;Get comparitor
OAE4 BE	08129	CMP	ň	Compare with comparand
OAE5 CO	08130	RNZ		Abandon comparison if inequality found
OAE6 13	08131	INX	D	;Update comparitor pointer
0AE7 23	08132	INX	н	;Update comparand pointer
OAE8 OD	08133	DCR	С	;Count down on loop count
OAE9 C8	08134	RZ		;Return (with zero flag set)
OAEA C3E3OA	08135 08136	JMP	Compare\$Dk\$Trk\$	LOOP
	08130	,		
	08138	, Move\$Dk\$Trk:		Moves the disk, track
	08139	HOTE OF A THE A		; variables pointed at by HL to
	08140			; those pointed at by DE
OAED OE03	08141	MVI	С,З	;Disk (1), Track (2)
	08142	Move\$Dk\$Trk\$Lo		
OAEF 7E	08143	MOV	A, M	;Get source byte
0AF0 12	08144	STAX	D	Store in destination
0AF1 13	08145	INX	D	;Update pointers
0AF2 23 0AF3 0D	08146	INX DCR	н	·
OAF4 C8	08147 08148	RZ	С	Count down on byte count
OAF5 C3EFOA	08149	JMP	Move\$Dk\$Trk\$Loc	;Return if all bytes moved
	08150	1	1010102000	
	08300	;#		
	08301	1		
	08302	; Move e	ight bytes	
	08303	•		
	08304 08305			t bytes in a block, C times, from
	08305			drop through" coding to speed
	08305	; upexe	cution.	
	08308		Parameters	
	08309			•
	08310	,	C = number of E	-byte blocks to move
	08311	1	DE -> destinati	on address
	08312	;	HL -> source ad	dress
	08313	1		
0450 75	08314 08315	Move\$8: MOV	A M	Cod bude dues course
0AF8 7E 0AF9 12	08315	MUV STAX	A,M D	;Get byte from source
0AFA 13	08315	INX	D	;Put into destination ;Update pointers
OAFB 23	08318	INX	Ĥ	tobaste boturgie
OAFC 7E	08319	MOV	A, M	;Get byte from source
OAFD 12	08320	STAX	D	Put into destination
OAFE 13	08321	INX	D	Update pointers
OAFF 23	08322	INX	н	
0800 7E	08323	MOV	A, M	;Get byte from source
OB01 12	08324	STAX	D	Put into destination
0B02 13 0B03 23	08325 08326	INX INX	D H	;Update pointers
0B03 23 0B04 7E	08326	MOV	н А.М	;Get byte from source
0B05 12	08328	STAX	D	Put into destination
OB06 13	08329	INX	Ď	Update pointers
OB07 23	08330	INX	Ĥ	
0B08 7E	08331	MOV	Α,Μ	;Get byte from source
OB09 12	08332	STAX	D	;Put into destination
				and the second second second second second second second second second second second second second second second

OBOA 13	08333	INX D		;Updat	e pointer	S			
OBOB 23	08334	INX H							
OBOC 7E	08335	MOV A,M			yte from				
OBOD 12	08336	STAX D			nto desti				
OBOE 13	08337	INX D		;Updat	e pointer	· S			
OBOF 23	08338	INX H							
OB10 7E	08339 08340	MOV A,M STAX D			yte from .nto desti				
OB11 12 OB12 13	08340	INX D			e pointer.				
OB12 13	08342	INX H		, opuat	e pointer	•			
OB14 7E	08343	MOV A, M		:Get b	yte from	source			
OB15 12	08344	STAX D			nto desti				
OB16 13	08345	INX D			e pointer				
OB17 23	08346	INX H							
	08347								
OB18 OD	08348	DCR C				loop counter			
0B19 C2F80A	08349	JNZ Move	\$8	;Repea	t until d	ione			
OB1C C9	08350	RET							
	08351 08352	1							
	08500	; ;#							
	08501	,							
	08502		to the di-	sk contr	ollers or	this computer system⊾			
	08503								
	08504	; There are tw	o "smart"	disk con	trollers	on this system, one			
	08505					one for the 5 1/4"			
	08506	; mini-diskett							
	08507	;							
	08508	; The controll	ers are "h	ard-wire	d" to mor	itor certain locations			
	08509	; in memory to	detect wh	en they	are to pe	rform some disk			
	08510					ocation 0040H, and 0 0045H. These are			
	08511 08512	; the 5 1/4" c	dick cont	100KS at	i i dealiidi a i i f the	most significant			
	08512								
	08514	; look at the word following the respective control bytes.							
	08515	; This word must contain the address of a valid disk control							
	08516	; table that specifies the exact disk operation to be performed.							
	08517	,							
	08518	; Once the operation has been completed, the controller resets							
	08519				and this	indicates completion			
	08520	; to the disk	driver cod	e.					
	08521	,	-			· · · · · · · · · · · · · · · · · · ·			
	08522	; The controll	er also se	ts a ret	urn code	in a disk status block.			
	08523	; Both control	ters use t	ne same Sie stat	us block	(0043H) for this. is less than 80H, then			
	08524 08525	; If the first ; a disk error	bas occur	red. For	this sin	mple BIOS, no further details			
	08526	; of the statu	s settings	are rel	levant. No	ote that the disk controller			
	08527	: has built-in	retry log	ic. read	s and wri	tes are attempted ten			
	08528	times before	the contr	olier re	eturns an	error.			
	08529	;							
	08530	; The disk con				below. Note that the			
	08531					ntrol tables to be			
	08532	; chained toge	ther so th	at a sec	uence of	disk operations can			
	08533	; be initiated	. In this	BIOS thi	is feature	e is not used. However,			
	08534	· · · · · · ·				pointers in the			
	08535	; disk control : in order to				the main control bytes			
	08536	; in order to	indicate t	ne end (	on the cha	31114			
0040 =	08537 08538	; Disk\$Control\$8	EQU	40H	:8"	ntrol byte			
0040 =	08538	Command\$Block\$8	EQU	41H		ol table pointer			
0041 -	08539	1	L		,				
0043 =	08541	, Disk\$Status\$Block	EQU	43H	;8" ANI	0 5 1/4" status block			
	08542	;							
0045 =	08543	Disk\$Control\$5	EQU	45H		" control byte			
0046 =	08544	Command\$Block\$5	EQU	46H	;Contre	ol table pointer			
	08545	7							
ļ	08546	*							
	08547	; Floppy Disk	Control Ta	DIES					
	08548	; []		DB	0	;Command			
OB1D 00	08549	Floppy\$Command:		EQU	01H	y commerte			
0001 =	08550	Floppy\$Read\$Code Floppy\$Write\$Code		EQU	02H				
0002 = 0B1E 00	08551 08552	floppy\$Unit:		DB	0	;Unit (drive) number = 0 or 1			
OBIE 00	08552	Floppy#Head:		DB	ŏ	:Head number = 0 or 1			
0B10 00	08554	Floppy#Track:		DB	ŏ	;Track number			
0B21 00	08555	Floppy\$Sector:		DB	ō	Sector number			
1							_		

0B22 0000	08556	Floppy\$Byte\$Co		DW	0	Number of bytes to read/write
OB24 0000	08557	Floppy\$DMA\$Add		DW	0	Transfer address
OB26 0000	08558 08559	Floppy\$Next\$St	atus#Block:	DW	0	<pre>;Pointer to next status block ; if commands are chained.</pre>
0828 0000	08560	FloonveNevteCo	ntrol\$Location:	nω	0	; if commands are chained. ;Pointer to next control byte
0020 0000	08561	1 10000 1000		24	•	; if commands are chained
	08562	3				,
	08700	;#				
	08701	1				
	08702	;				
	08703	Write\$No\$Deblo	ck:			ts of disk buffer to
	08704				ect sec	
OB2A 3E02	08705	MVI	A,Floppy\$Write			write function code
OB2C C3310B	08706	JMP	Common\$No\$Deblc			common code
	08707	Read\$No\$Debloc	K:			sly selected sector
0B2F 3E01	08708	MVI		; into		
082F 3E01	08709		A,Floppy\$Read\$0	.00e	juet r	read function code
0B31 321D0B	08710 08711	Common\$No\$Deb1	Floppy\$Command			function code
0831 321008	08712	514	FIOPPy#Command	set co	nondeb	blocked command table
	08713			toer at	nonder	STOCKED COMMAND (ADTe
0B34 3A360A	08714	LDA	Selected\$Disk\$	[vne	• Check	k if memory disk operation
0B37 FE03	08715	CPI	M\$Disk		yonecr	A IT memory disk operation
0B39 CA7A0B	08716	JZ	M\$Disk\$Transfer	Yes, i	t is Ma	5Disk
	08717	~~		,,		
	08718	No\$Deblock\$Ret	r <b>y :</b>	;Re-ent	ry poir	nt to retry after error
0B3C 218000	08719	LXI	H, 128	Bytes		
B3F 22220B	08720	SHLD	Floppy\$Byte\$Cou			
OB42 AF	08721	XRA	Α		ppy onl	ly has head O
OB43 321FOB	08722	STA	Floppy\$Head			
	08723			,		
OB46 3A2DOA	08724	LDA	Selected\$Disk			ntroller only knows about
	08725					d 1 so Selected\$Disk must
	08726			3 be o	onverte	ed
OB49 E601	08727	ANI	01H		nto O d	
OB4B 321EOB	08728	STA	Floppy\$Unit	;Set ur	it numb	ber
	08729			7		
OB4E 3A2EOA	08730	LDA	Selected\$Track	<b>.</b>		
0B51 32200B	08731	STA	Floppy\$Track	;Set tr	ack nur	mber
	08732	LDA	C-1	;		
0854 3A300A 0857 322108	08733 08734	STA	Selected#Sector Floppy#Sector		etor nu	
0657 322106	08735	318	FIOPPY#Sector	joet se	recor no	
085A 2AA609	08736	LHLD	DMA\$Address	Tranet	er dire	ectly between DMA Address
OB5D 22240B	08737	SHLD	Floppy#DMA\$Add	, a		8" controller.
0200 222 /0D	08738	01120		1	,	
	08739				isk conf	troller can accept chained
	08740			; dis	contro	ol tables, but in this case,
	08741			they	are no	ot used, so the "Next" pointers
	08742			; must	be pot	inted back at the initial
	08743				rol by	tes in the base page.
OB60 214300	08744	LXI	H, Disk\$Status\$			<b>;Point next status back at</b>
<b>0B63 22260B</b>	08745	SHLD	Floppy\$Next\$Sta	atus\$Bloc	:k	; main status block
	08746					1
OB66 214000	08747	LXI	H,Disk\$Control			Point next control byte
<b>0B69 22280B</b>	08748	SHLD	Floppy\$Next\$Co	ntro1\$Loo	ation	; back at main control byte
	08749				1.0.1	
OB6C 211DOB	08750	LXI	H, Floppy\$Comman		;Poin	t controller at control table
0B6F 224100	08751	SHLD	Command\$Block\$8	5		
<b>0B72 214000</b>	08752 08753	LXI	H,Disk#Control		Acti	vate controller to perform
0872 214000 0875 3680	08753	MVI	M, 80H	**		vate controller to perform eration
0B77 C33B0C	08755	JMP	Wait\$For\$Disk\$	Complete	, 00	eracion
08// 033800	08756	811		Joinple Ce		
	08757	;				
	08900	; #				
	08901		disk driver			
	08902	1				
	08903		outine must use a	an intera	nediarv	buffer, since the
	08904		dress in bank ("			
	08905					the M\$Disk itself.
	08906	; prace ; The M\$	Disk\$Buffer is al	nove the	48K max	rk, and therefore
	08907					ss of which bank/track
	08908	; is sel		Space I		
		,				
	08909	;				

Figure 8-10. (Continued)

<u> </u>				
	08911 08912		iting, the 128-by	yte sector must be processed:
	08912		• Maria	
	08914	1		DMA\$Address -> M\$Disk\$Buffer
		3	2. Select corre	ect track (+1 to get bank number)
ļ	08915 08916	1		M\$Disk\$Buffer -> M\$Disk image
		3	<ol> <li>Select bank</li> </ol>	0
	08917	,		
	08918 08919		ading, the proces	551NG 15:
	08920	7		
	08920	7	1. Select corre	
		;		M\$Disk image -> M\$Disk\$Buffer
	08922 08923		3. Select Bank	
		2	4. Move sector	M\$Dísk\$Buffer -> DMA\$Address
	08924	7		
	08925	; If the	re is any risk of	any interrupt causing control
	08926	; to be	transferred to an	address below 48K, interrupts must
	08927	; bedis	abled when any ba	ank other than 0 is selected.
	08928	· · · · · · · · ·		
	08929	M\$Disk\$Transfe	ri	
0B7A 3A30		LDA	Selected#Sector	Compute address in memory
0B7D 6F	08931	MOV	L,A	; by muliplying sector * 128
0B7E 2600		MVI	н, о	
0880 29	08933	DAD	н	:* 2
0B81 29	08934	DAD	н	1¥ 4
0B82 29	08935	DAD	н	** 8
0883 29	08936	DAD	н	;* 16
0884 29 0885 29	08937	DAD	н	:* 32
0B85 29 0B86 29	08938	DAD DAD	H	;* 64
0000 27	08940	DAD	п	;* 128
0887 3A26		1.54	CalestadeTassi	Commute shiph held of heals and
000/ 3H20	08942	LDA	Selected\$Track	Compute which half of bank sector
OB8A 47	08942	MOV	B,A	; is in by using LS bit of track
0B8B E601		ANI	1 1	Save copy for later
OBSD CA94		JZ	M\$Disk\$Lower\$Ha	;Isolate lower/upper indicator
0000 0074	08946	52	H#DISK#COWER #Ha	11
0890 1100		LXI	D,(48 * 1024) /	2 ;Upper half, so bias address
0B93 19	08948	DAD	n	z ;opper mari, so bras address
	08949		-	
	08950	M\$Disk\$Lower\$Ha	alf:	;HL -> sector in memory
OB94 78	08951	MOV	A, B	Recover selected track
0895 1F	08952	RAR		;Divide by 2 to get bank number
0B96 3C	08953	INR	A	Bank 1 is first track
OB97 47	08954	MOV	B,A	Preserve for later use
	08955			
0898 3A1D		LDA	Floppy\$Command	Check if reading or writing
0898 FE02	08957	CPI	Floppy\$Write\$Co	de
OB9D CABE	08 08958	JZ	M\$Disk\$Write	;Writing
	08959			;Reading
	08960			
OBAO CDDD		CALL	Select\$Bank	;Select correct memory bank
OBA3 1130		LXI	D,M\$Disk\$Buffer	;DE -> M\$Disk\$Buffer, HL -> M\$Disk image
OBA6 OE10	08963	MVI	C,128/8	Number of 8-byte blocks to move
OBA8 CDF8		CALL	Move\$8	
08AB 0600	08965			
		MVI	B,O	;Revert to normal memory bank
OBAD CDDD	0B 08967 08968	CALL	Select\$Bank	
OBBO 2AA6				· Bot user/s DMA address:
0BB3 1130			DMA\$Address D_M\$Disk\$Ruffer	;Get user's DMA address
OBBS 1130	08971	XCHG	D,M\$Disk\$Buffer	;DE -> User's DMA, HL -> M\$Disk buffer
OBB7 OE10		MVI	C,128/8	;DE -> User's DMA, HL -> Mabisk butter ;Number of 8-byte blocks to move
OBB9 CDF8		CALL	Move\$8	Anguner of C-Dire Diocks to Wove
	08974			
OBBC AF	08975	XRA	Α	;Indicate no error
OBBD C9	08976	RET		
	08977			
	08978	M\$Disk\$Write:		;Writing
OBBE E5	08979	PUSH	н	Save sector's address in M\$Disk image
OBBF 2AA6	09 08980	LHLD	DMA\$Address	Move sector into M\$Disk\$Buffer
OBC2 1130	23 08981	LXI	D,M#Disk#Buffer	
0BC5 0E10		MVI	C,128/8	Number of 8-byte blocks to move
OBC7 CDF3		CALL	Move\$8	;(Does not use B register)
	08984		_	;B = memory bank to select
OBCA CDDD		CALL	Select\$Bank	
	08986			
· · · · · · · · · · · · · · · · · · ·				

Figure 8-10. (Continued)

		· · · · · · · · · · · · · · · · · · ·			
OBCD D1	08987	POP	D		r sector's M\$Disk image address
OBCE 213023	08988	LXI	H,M\$Disk\$Buffe	r	
OBD1 OE10	08989	MVI	C,128/8		
OBD3 CDF80A	08990	CALL	Move\$8	;Move 1	nto M\$Disk image
	08991				
OBD6 0600	08992	MVI	B,O	;Select	Dank U
OBD8 CDDDOB	08993 08994	CALL	Select\$Bank		
OBDB AF	08995	XRA	<b>A</b>	*Indica	te no error
OBDC C9	08996	RET	-	, Indica	
	08997	1			
	09100	;*			
	09101	; Select	bank		
	09102	1			
	09103	; This r	outine switches	in the re	quired memory bank.
	09104				t controls bank selection
	09105			it. ines	e are preserved across
	09106 09107	t banks	elections.		
	09108		parameter		
	09109	; <u> </u>	Patratile (et		
	09110	;	B = bank numbe	r	
	09111	,		•	
0040 =	09112	Bank\$Contro1\$P	ort EQU	40H	
00F8 =	09113	Bank\$Mask	EQU	1111\$10	00B ;To preserve other bits
	09114	;			
	09115	Select\$Bank:			<b>.</b>
OBDD DB40	09116	IN	Bank\$Contro1\$P	ort	;Get current setting in port
OBDF E6F8	09117	ANI ORA	Bank\$Mask B		Preserve all other bits Set bank code
OBE1 BO	09118	OUT	Bank\$Control\$P	~**	;Select the bank
OBE2 D340 OBE4 C9	09119 09120	RET	Denkacontroiar	071	Joerect the bank
0BE4 07	09121	;			
	09200	7#			
	09201	1			
	09202				
	09203	Write\$Physical	:		contents of disk buffer to
ADER 0500	09204	MVI	A . E 1		ect sector
OBE5 3E02 OBE7 C3ECOB	09205 09206	JMP	A,Floppy\$Write Common\$Physica		;Get write function code
UBE/ LJECUB	09205	Read\$Physical:	CommonsPhysica		reviously selected sector
	09208	Neddel II/ Sicell			disk buffer
OBEA 3E01	09209	MVI	A,Floppy\$Read\$		Get read function code
	09210	;			
	09211	Common\$Physica	1:		
OBEC 321DOB	09212	STA	Floppy\$Command	;Set co	mmand table
	09213				
	09214	1		<b>.</b> .	
	09215	Deblock\$Retry:		;Re-ent	ry point to retry after error
OBEF 3A2A0A OBF2 FE01	09216 09217	LDA CPI	In\$Buffer\$Disk Floppy\$5	\$IYPC	;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy
OBF4 CAFDOB	09218	JZ	Correct\$Disk\$T		iYes
OBF7 3E01	09219	MVI	A, 1	785	;No, indicate disk error
OBF9 32320A	09220	STA	Disk\$Error\$Fla	a	
OBFC C9	09221	RET		•	
	09222	Correct#Disk#T	ypei	;Set up	disk control table
	09223			1	
OBFD 3A270A	09224	LDA	In\$Buffer\$Disk		t disk number to 0 or 1
0C00 E601	09225	ANI	1	) for	disk controller
OCO2 321EOB	09226	STA	Floppy\$Unit		
	09227	LHLD	Ta #Duddau#7		hand and Aurali symbols
0C05 2A280A 0C08 7D	09228 09229	MOV	A,L		head and track number umbered tracks will be on
0C09 E601	09230	ANI	м, L 1		0, odd numbered on head 1
OCOB 321FOB	09231	STA	Floppy\$Head		ad number
	09232	• • • •		,	
OCOE 7D	09233	MOV	A,L		this is single byte value
OCOF 1F	09234	RAR			r track (carry off from ANI above)
0C10 32200B	09235	STA	Floppy\$Track		
	09236				
0C13 3E01	09237	MVI	A, 1		Start with sector 1 as a whole
0C15 32210B	09238	STA	Floppy\$Sector		; track will be transferred
0018 210012	09239 09240	1 X T	H. Bytes\$Per\$Tr	ack	; ;Set byte count for complete
OC18 210012 OC18 22220B	09240	SHLD	Floppy\$Byte\$Co		; track to be transferred
SOID REELUD	09242	ONED			; CACK to be transferred
	¥7 = 7£	·····			· · · · · · · · · · · · · · · · · · ·

Figure 8-10. (Continued)

OC1E 21A40F OC21 22240B	09243	LXI	H,Disk\$Buffer	;Set transfer address to be
0C21 22240B	09244	SHLD	Floppy\$DMA\$Addr	ss ; disk buffer
	09245			;
	09246			As only one control table is in
	09247			; use, close the status and busy
	09248			; chain pointers back to the
	09249			; main control bytes
0024 214300	09250	LXI	H,Disk\$Status\$B	
OC27 22260B	09251	SHLD	Floppy\$Next\$Sta	
0C2A 214500	09252	LXI	H, Disk\$Control\$	
0C2D 22280B	09253	SHLD	Floppy\$Next\$Con	
VC20 222000	09254	300	FICEPYPNEX(\$CC	rolación
OC30 211DOB	09255	LXI	H,Floppy\$Comman	Set up command block pointer
0C33 224600	09256	SHLD	Command\$Block\$5	
	09257			
0036 214500	09258	LXI	H,Disk\$Control\$	Activate 5 1/4" disk controller
0039 3680	09259	MVI	M, 80H	
	09260	;		
	09261	Wait\$For\$Disk\$C	Complete:	;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 AF	09265	XRA	Α	Ensure hung flag clear
0C3C 32330A	09266	STA	n Disk\$Hung\$Flag	Privatie unua itea cteai
0000 0200VA	09266	314	Prevendiget 198	
0005 015700	07267	LXI	H Distant	Contain and a balance of the
0C3F 21570C	09268		H,Disk\$Timed\$Ou	
0C42 015802	09269	LXI	B,Disk\$Timer	;Time delay
0C45 CD6D08	09270	CALL	Set\$Watchdog	
	09271	Disk\$Wait\$Loop:		
0C48 7E	09272	MOV	A, M	;Get control byte
OC49 B7	09273	ORA	A	
OC4A CA5DOC	09274	JZ	Disk\$Complete	;Operation done
	09275			
0C4D 3A330A	09276	LDA	Disk\$Hung\$Flag	Also check if time expired
0C50 B7	09277	ORA	A	FRIDE CHECK IT CIME EXPIRED
0C51 C2B40D	09278	JNZ	Disk\$Error	thill be not to dob
0031 020400		CINZ	DISKACTOF	;Will be set to 40H
	09279	JMP		
0C54 C3480C	09280	JMP	Disk\$Wait\$Loop	
	09281			<b>.</b> . <b>.</b>
	09282	Disk\$Timed\$Out:		;Control arrives here from watchdog
	09283			; routine itself so this is effectively
	09284			; part of the interrupt service routine.
0C57 3E40	09285	MVI	A, 40H	;Set disk hung error code
0C59 32330A	09286	STA	Disk\$Hung\$Flag	; into error flag to pull
	09287			; control out of loop
0050 09	09288	RET		Return to watchdog routine
	09289			······
	09290	Disk\$Complete:		
0050 010000	09291	LXI	B.0	;Reset watchdog timer
0030 010000		LAI	5,0	
	09292	<b></b> .	0-14U-1 · ·	;HL is irrelevant here
0C60 CD6D08	09293	CALL	Set\$Watchdog	
	09294			
0063 344300	09295	LDA	Disk\$Status\$Blo	
0C66 FE80	09296	CPI	80H	<pre>\$Check if any errors occurred</pre>
DC69 DAB40D	09297	JC	Disk\$Error	;Yes
	09298	;		
	09299	Disk\$Error\$Igno	ore:	
OC6B AF	09300	XRA	A	; No
0C6C 32320A	09301	STA	Disk\$Error\$Flag	;Clear error flag
0C6F C9	09302	RET	is a second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second second secon	
	09303	NET.		
		_		
	09304	7 - 11		
	09400	;#		
	09401	; Disker	ror message hand	ing
	09402	7		
	09403	;		
	09404	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			; points to the correct message text.
		DB	40H	· ····································
0070 40	09410			
	09410			
0C71 9D0C	09411	DW	Disk\$Msg\$40	
0C70 40 0C71 9D0C 0C73 41 0C74 A20C				

Figure 8-10. (Continued)

<b></b>			
0C76 42 09414	DB	42H	
0C77 ACOC 09415	DW	Disk\$Ms	3\$42
0C77 ACOC 09415 0C79 21 09416	DB	21H	
	DW		- 4 3 1
		Disk\$Ms	J#21
0C7C 22 09418	DB	22H	
0C7D C10C 09419	DW	Disk\$Ms	1\$22
0C7F 23 09420	DB	23H	
0080 0800 09421	DW	Disk\$Ms	3\$23
0082 24 09422	DB	24H	
			- 404
0C83 DA0C 09423	DW	Disk\$Ms	3924
0085 25 09424	DB	25H	
0C86 E60C 09425	DW	Disk\$Ms	n \$ 25
	DB		120
0C88 11 09426		11H	
0C89 F90C 09427	DW	Disk\$Ms	3\$11
OC8B 12 09428	DB	12H	
0C8C 070D 09429	DM	Disk\$Ms	3912
0C8C 070D 09429 0C8E 13 09430	DB	13H	
0C8F 140D 09431	DW	Disk\$Ms	9513
	DB	14H	
0091 14 09432			
0092 2200 09433	DW	Disk\$Ms	g\$14
0094 15 09434	DB	158	
			- 4 1 2
0C95 310D 09435	DW	Disk#Ms	4+1-J
0097 16 09436	DB	16H	
0C98 3D0D 09437	DW	Disk\$Ms	3\$16
	DB	0	
		*	;<== Terminator
0C9B 4D0D 09439	DW	Disk\$Ms	s\$Unknown ;Unmatched code
09440			
0003 = 09441	, DEM\$Entry\$Size	EQU	3 ;Disk error message table entry size
	DEMPENTRYPSIZE	EWU	3 DISK error message table entry Size
09442	7		
09443	; Message	texts	
09444			
		-	
0C9D 48756E670009445	Disk\$Msg\$40:	DB	'Hung',O ;Timeout message
OCA2 4E6F74205209446	Disk\$Msg\$41:	DB	Not Ready',0
OCAC 577269746509447	Disk\$Msg\$42:	DB	'Write Protected',0
0CBC 446174610009448	Disk\$Msg\$21:	DB	'Data',0
OCC1 466F726D6109449	Disk\$Msg\$22:	DB	'Format',0
OCC8 4D6973736909450	Disk\$Msg\$23:	DB	'Missing Data Mark',0
		DB	'Bus Timeout',O
OCDA 427573205409451	Disk\$Msg\$24:		
OCE6 436F6E747209452	Disk#Msg#25:	DB	'Controller Timeout',0
OCF9 447269766509453	Disk\$Msg\$11:	DB	'Brive Address',0
		DB	
0D07 486561642009454	Disk\$Msg\$12:		'Head Address', O
0D14 547261636B09455	Disk\$Msg\$13:	DB	Track Address1,0
0D22 536563746F09456	Disk\$Msg\$14:	DB	'Sector Address',0
		DB	'Bus Address',0
0D31 427573204109457	Disk\$Msg\$15:		Bus Huuress ;0
0D3D 496C6C656709458	Disk\$Msg\$16:	DB	'Illegal Command',0
OD4D 556E6B6E6F09459	Disk\$Msg\$Unknow	n:	DB (Unknown),0
09460			
	· · · · · · · · · · · · · · · · · · ·		
09461	Disk‡EM\$1:		;Main disk error message part 1
0D55 070D0A 09462		DB	BELL, CR, LF
OD58 4469736B2009463		DB	'Disk '.O
			,-
09464	;		
09465			;Error text output next
09466			
09467	, Disk\$EM\$2:		;Main disk error message part 2
	D1989C0021		
OD5E 204572726F09468		DB	1 Error (1
0066 0000 09469	Disk\$EM\$Status:	DB	0,0 ;Status code in Hex.
0D68 290D0A202009470		DB	<pre>/)',CR,LF,' Drive '</pre>
	DiskEMADuring		
0076 00 09471	Disk\$EM\$Drive:	DB	0 ;Disk drive code, A,B
0077 202048656109472		DB	, Head '
OD7E 00 09473	Disk\$EM\$Head:	DB	0 ;Head number
OD7F 2C2054726109474		DB	, Track
0087 0000 09475	Disk\$EM\$Track:	DB	0,0 ;Track number
0089 202053656309476		DB	1, Sector 1
0092 0000 09477	Disk\$EM\$Sector:	DB	0,0 ;Sector number
	DISK#ENFORCTOP'		
0D94 2C204F706509478		DB	<pre>/, Operation - /</pre>
0DA2 00 09479		DB	0 ;Terminator
09480			
	, Disk\$EM\$Read:	DB	'Read.',0 ;Operation names
ODA3 526561642E09481			
ODA9 577269746509482	Disk\$EM\$Write:	DB	'Write.',O
09483	;		
09484			
	7	fium-	
09485	Disk\$Action\$Con [*]		
ODBO 00 09486		DB	0 ;Set to character entered by user
ODB1 OD0A00 09487		DB	CR, LF, O
09488	;		
09489		ror proc	rssor

Figure 8-10. (Continued)

			······	
	09490	1		
	09491		outine builds and output	s an error message.
	09492		ser is then given the opp	
	09493	;	iel is chen given the opp	or can't y to
	09494	,	P vetry the energia	ion that caused the error
		,		
	09495	,	1 ignore the error	and attempt to continue
	09496	,	A abort the program	n and return to CP/M.
	09497	7		
	09498	Disk\$Error:		
ODB4 F5	09499	PUSH	PSW \$Prese	erve error code from controller
0DB5 21660D	09500	LXI	H,Disk\$EM\$Status	Convert code for message
ODBS CD440E	09501	CALL	CAH	Converts A to hex.
	09502			
ODBB 3A270A	09503	LDA	In\$Buffer\$Disk	Convert disk id. for message
			'A'	
ODBE C641	09504	ADI		;Make into letter
0DC0 32760D	09505	STA	Disk\$EM\$Drive	
	09506			
ODC3 3A1FOB	09507	LDA	Floppy\$Head	;Convert head number
0DC6 C630	09508	ADI	101	
ODC8 327EOD	09509	STA	Disk\$EM\$Head	
0000 32/200	09510	OTH	Diskfelifieds	
0000 040000		1.04	ElennutTunek	·Convert track number
ODCB 3A200B	09511	LDA	Floppy\$Track	;Convert track number
ODCE 21870D	09512	LXI	H,Disk\$EM\$Track	
ODD1 CD440E	09513	CALL	CAH	
1	09514			
ODD4 3A210B	09515	LDA	Floppy\$Sector	;Convert sector number
ODD7 21920D	09516	LXI	H, Disk\$EM\$Sector	
ODDA CD440E	09517	CALL	CAH	
	09518	UNC -		
				Dubuuh diwat and of parama
ODDD 21550D	09519	LXI	H,Disk\$EM\$1	;Output first part of message
ODEO CD5305	09520	CALL	Output\$Error\$Message	
1	09521			
ODE3 F1	09522	POP	PSW	Recover error status code
ODE4 47	09523	MOV	B,A	For comparisons
ODE5 216DOC	09524	LXI	H,Disk\$Error\$Messages	
ODED 216DOC			III DISKOLI I OI THESSAGES	;HL -> table ~ one entry
	09525			Get entry size for loop below
ODE8 110300	09526	LXI	D, DEM\$Entry\$Size	Get entry size for loop below
	09527	Disk\$Error\$Ne>		
ODEB 19	09528	DAD	D	Move to next (or first) entry
	09529			
ODEC 7E	09530	MOV	A, M	;Get code number from table
ODED B7	09531	ORA	Α.	Check if end of table
ODEE CAFSOD	09532	JZ	Disk\$Error\$Matched	;Yes, pretend a match occurred
ODF1 B8	09533	CMP	B	Compare to actual code
ODF2 CAFSOD	09534	JZ	Disk\$Error\$Matched	Yes, exit from loop
		JMP		
ODF5 C3EBOD	09535	JHP	Disk\$Error\$Next\$Code	;Check next code
1	09536	;		
1	09537	Disk\$Error\$Ma	tched:	
0DF8 23	09538	INX	н	;HL -> address of text
ODF9 5E	09539	MOV	E,M	;Get address into DE
ODFA 23	09540	INX	H	
ODFB 56	09541	MOV	п, м	
				AND IN A REAL
ODFC EB	09542	XCHG		;HL -> text
ODFD CD5305	09543	CALL	Output\$Error\$Message	;Display explanatory text
1	09544			
OE00 215E0D	09545	LXI	H,Disk\$EM\$2	;Display second part of message
0E03 CD5305	09546	CALL	Output\$Error\$Message	
	09547			
0E06 21A30D	09548	LXI	H,Disk\$EM\$Read	;Choose operation text
0000 210000	09549	_~··		; (assume a read)
0500 044000	07347	LDA	Floppy\$Command	Get controller command
OEO9 3A1DOB	09550			yoet controller command
OEOC FEO1	09551	CPI	Floppy\$Read\$Code	
OEOE CA14OE	09552	Σل	Disk\$Error\$Read	;Yes
OE11 21A90D	09553	LXI	H,Disk\$EM\$Write	;No, change address in HL
1	09554	Disk#Error#Rea	ad 1	
0E14 CD5305	09555	CALL	Output\$Error\$Message	Display operation type
	09556			
1		, Disk\$Error\$Red	west#Action:	Ask the user what to do next
	09557			Display prompt and wait for input
0E17 CD2F05	09558	CALL	Request\$User\$Choice	
1	09559			; Returns with A = uppercase char.
OE1A FE52	09560	CPI	<r<sup>2</r<sup>	;Retry?
0E1C CA2COE	09561	JZ	Disk\$Error\$Retry	
OE1F FE41	09562	CPI	'A'	; Abor t
0E21 CA360E	09563	JZ	System\$Reset	
0E24 FE49	09564	CPI	×1×	;Ignore
OE26 CA6BOC	09565	JZ	Disk\$Error\$Ignore	
VERO CHODUC	0,000			

Figure 8-10. (Continued)

0E29 C3170E	09566 09567	JMP.	Disk\$Error\$Re	quest\$Action
	09568	; Disk\$Error\$Ret	~ ~ •	The decision on where to return
	09569	DISKBEFFOR WREC	r <b>y i</b>	; depends on whether the operation
	09570			; failed on a deblocked or
	09571			; nondeblocked drive.
0E2C 3A350A	09572	LDA	Selected\$Disk	\$Deblock
0E2F B7	09573	ORA	Α	
OE30 C2EFOB	09574	JNZ	Deblock\$Retry	
		JMP		•'
0E33 C33C0B	09575	JMP	No\$Deblock\$Re	(ry
	09576	,		
	09577	System\$Reset:		;This is a radical approach, but
	09578			; it does cause CP/M to restart.
DE36 0E00	09579	MVI	C.0	;System reset
0E38 CD0500	09580	CALL	BDOS	
	09581			
	09582			
		1		
	09583	;		
	09584	; Atou	pper	
	09585	;		
	09586	; Conver	ts the contents	of the A register to an upper-
	09587	; case l	etter if it is	currently a lowercase letter.
	09588	1		
	09589		parameters	
			Haidmeidia	
	09590	;	<b>.</b>	
	09591	;	A = character	to be converted
	09592	;		
	09593		arameters	
	09594	;		
	09595	,	A = converted	character
			A - converteu	
	09596			
	09597	A\$To\$Upper:		
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
OE40 DO	09601	RNC		No need to convert
0E41 E65F	09602	ANI	5FH	Convert to uppercase
			JEN	, convert to appercase
OE43 C9	09603	RET		
	09604	1		
	09605	; Conver	t A register to	hexadecimal
	09606	,		
	09607		ubroutine conve	rts the A register to hexadecimal.
	09608	;		
	09609		parameters	
			parameters	
	09610	;		
	09611	;	A = value to	be converted and output
	09612	,	HL -> buffer	area to receive two characters of output
	09613	;		· · · · · · · · · · · · · · · · · · ·
	09614		arameters	
			ar and Let B	
	09615	;		· · · · · · · · · · · · · · · · · · ·
	09616	;	HL -> Dyte fo	llowing last hex byte output
	09617	;		
	09618	CAH:		
0E44 F5	09619	PUSH	PSW	;Take a copy of the value to be converted
OE45 OF	09620	RRC		Shift A right four places
0E46 0F	09621	RRC		,
		RRC		
OE47 OF	09622			
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\$Convert:		
			000000000000	alaalada I.C. Kauna bid-
OE4D E6OF	09629	ANI	0000\$1111B	;Isolate LS four bits
0E4F C630	09630	ADI	101	;Convert to ASCII
OE51 FE3A	09631	CPI	<b>797 + 1</b>	;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
0230 2007			,	Founder L LO & Letter
	09634	CAH\$Numeric:		<b>o</b>
OE58 77	09635	MOV	M, A	;Save character
0E59 23	09636	INX	н	;Update character pointer
0E5A C9	09637	RET		
	09638			
		_		
	09639	;		
	09640 09700	; ;#		

Figure 8-10. (Continued)

	09701	;				
	09702	; Disk co	ontrol table imag	es for w	arm boot	
	09703 09704	; Boot\$Contro1\$Pa				
0E5B 01	09705	DB	1		Read function	
0E5C 00	09706	DB	0		;Unit (drive) number	
0E5D 00	09707	DB	ŏ		;Head number	
0E5E 00	09708	DB	0		Track number	
0E5F 02	09709	DB	2		Starting sector number	
0E60 0010	09710	DW	8×512		Number of bytes to read	
0E62 00C4	09711	DW DW	CCP\$Entry		Read into this address	
OE64 4300 OE66 4500	09712 09713	DW	Disk\$Status\$Blo Disk\$Contro1\$5	ic k	Pointer to next status block	
0200 4000	09714	Boot\$Control\$Pa			;Pointer to next control table	
0E68 01	09715	DB	1		Read function	
0E69 00	09716	DB	ō		;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 0E6F 00D4	09720 09721	DW	3*512	E101	Number of bytes to read	
0E71 4300	09722	DW	CCP\$Entry + (8* Disk\$Status\$Blo		;Read into this address	
0E73 4500	09723	DW	Disk\$Control\$5	CK	;Pointer to next status block ;Pointer to next control table	
02/0 4000	09724	2.4	DISK#Control#O		, orner to next control table	
	09725	;				
	09726	;				
	09800	; #				
	09801	; UDOOT-	allaum bi A - A			
	09802 09803	WBOOT:	;Warm boot entr		the CCP and BDOS must be reloaded	
	09803				In this BIOS, only the 5 1/4"	
	09805				11 be used, therefore this code	
	09806		; is h	ardware	specific to the controller. Two	
	09807				d control tables are used.	
0E75 318000	09808	LXI	SP,80H			
0E78 115B0E	09809	LXI	D,Boot\$Contro1\$	Part1	;Execute first read of warm boot	
OE7B CD8AOE	09810 09811	CALL	Warm\$Boot\$Read		Load drive 0, track 0,	
0E7E 11680E	09812	LXI	D.Boot\$Control\$	Part 7	; head 0, sectors 2 - 8 ;Execute second read	
OE81 CD8AOE	09813	CALL	Warm\$Boot\$Read	rartz	Load drive 0. track 0.	
	09814				; head 1, sectors 1 - 3	
OE84 CDDFOE	09815	CALL	Patch\$CPM		;Make custom enhancements patches	
0E87 C36C02	09816	JMP	Enter\$CPM		;Set up base page and enter CCP	
	09817	j Mariant David Chandra		- 0		
	09818 09819	Warm\$Boot\$Read:			ry, DE -> control table image ontrol table is moved into	
	07820				main disk control table and	
	09821				the controller activated.	
0E8A 211D0B	09822	LXI	H,Floppy\$Comman	d	;HL -> actual control table	
0E8D 224600	09823	SHLD	Command\$Block\$5		Fell the controller its address	
	09824				Move the control table image	
	09825		a	<b>.</b>	<pre>; into the control table itself.</pre>	
OE90 OEOD	09826 09827	MVI Warm\$Boot\$Move:	C,13	;Set by	te count	
0E92 1A	09827	LDAX	D	Got in	age byte	
0E93 77	09829	MOV	M,A		into actual control table	
0E94 23	09830	INX	H		pointers	
0E95 13	09831	INX	D			
0E96 0D	09832	DCR	С		down on byte count	
0E97 C2920E	09833	JNZ	Warm\$Boot\$Move	;Contin	ue until all bytes moved	
	09834			-		
0E9A 214500 0E9D 3680	09835 09836	LXI MVI	H,Disk\$Control\$ M.80H	2	;Activate controller	
VE7D 366V	09836	Wait\$For\$Boot\$C				
OE9F 7E	09838	MOV	A,M		;Get status byte	
OEAO B7	09839	ORA	Α		;Check if complete	
OEA1 C29FOE	09840	JNZ	Wait\$For\$Boot\$Co	omplete	;No	
	09841				;Yes, check for errors	
0EA4 3A4300	09842	LDA	Disk\$Status\$Bloc	c K		
OEA7 FE80 OEA9 DAADOE	09843 09844	CPI JC	80H Warm\$Boot\$Error		Ver an error occurred	
OEAC C9	09844	RET	warm#poot#cfror		;Yes, an error occurred	
CENC C7	07845	:				
	09847	Warm\$Boot\$Error				
0EAD 21860E	09848	LXI	H,Warm\$Boot\$Erro	or\$Messa	3e	
OEBO CD5F02	09849	CALL	Display\$Message			

Figure 8-10. (Continued)

0EB3 C3750E	09850 09851	JMP	WBOOT	;Restart warm boot
	09852	, Warm\$Boot\$Error\$	Message:	
OEB6 0D0A5761				oot Error - retrying′,CR,LF,O
	09854	;		
	09855	1		
	10000	; #		
	10001	;		
	10002	Ghost\$Interrupt:		rol will only arrive here under the most
	10003			usual circumstances, as the interrupt
	10004			ntroller will have been programmed to
	10005		; su	ppress unused interrupts.
	10006		;	
OED8 F5	10007		PSW	;Save pre-interrupt registers
OED9 3E20	10008		A, IC\$EOI	;Indicate end of interrupt
OEDB D3D8	10009		IC\$OCW2\$Port	
OEDD F1	10010		PSW	
OEDE C9	10011	RET		
	10012	7		
	10013	7		
	10100	;#		
	10101	7		
	10102	; Patch CP	/M	
	10103	;		
	10104			me very special patches to the
	10105		BDOS in order	to make some custom enhancements
	10106	ŧ		
	10107	; Public f		
	10108			disk systems it is extremely useful
	10109	;		the disk using the user number features.
	10110			ecomes wasteful of disk space because
	10111	7		es of common programs must be stored in
	10112	•		a. This patch makes User O public
	10113			om any other user area.
	10114	;	*** WARNING *	
	10115		Files in User	O MUST be set to system and read/only
	10116	,	status to avo	id their being accidentally damaged.
	10117			e side effects associated with public
	10118	;		tch can be turned on or off using
	10119	;	a flag in the	long term configuration block.
	10120 10121	;		
	10121	; User pro		/M's USER command and user numbers
	10122	1		t is all too easy to become confused
	10123	;	in general, i	ich user number you are "in." This
	10124	•		s the CCP to display a prompt which
	10125	1		y the default disk id., but also the
	10127	,	current user	number, and an indication of whether
	10128	7	public files	
	10128	;	PODIC LINES	ALA FUGR1684
	10129	1		P3B> or 3B>
	10130	1		
	10132	:		When public files are enabled.
	10132	,		wien bante ittes die endnien:
	10134	, Fouster	for public fi	les
	10135	,		
D35E =	10136	PF\$BDOS\$Exit\$Poi	int EQU	BDOS\$Entry + 758H
D37C =	10137	PF\$BD0S\$Char\$Mat		BDOS\$Entry + 776H
D361 =	10138	PF\$BDOS\$Resume\$F		BDOS\$Entry + 75BH
000D =	10139	PF\$BDOS\$Unused\$E		13
	10140	1		
	10141	;		
	10142		for user prom	et
	10143	1		• -
C788 =	10144	UP\$CCP\$Exit\$Poir	nt EQU	CCP\$Entry + 388H
C78B =	10145	UP\$CCP\$Resume\$Pc		CCP\$Entry + 38BH
C513 =	10146	UP\$CCP\$Get\$User	EQU	CCP\$Entry + 113H
C5D0 =	10147	UP\$CCP\$Get\$Disk1		CCP\$Entry + 1BOH
C48C =	10148	UP\$CCP\$CONOUT	EQU	CCP\$Entry + 8CH
0.00 -	10149	1	230	
	10150	,		
	10150		he interventi	on points
	10152			
	10153	, Patch\$CPM:		
OEDF SECS	10154	MVI	A, JMP	:Set up opcode
0EE1 325ED3	10155	STA	PF\$BDOS\$Exit\$	
		018		

Figure 8-10. (Continued)

0EE4 3288C7	10156	STA	UP\$CCP\$Exit\$Poi	nt
0EE7 21F40E	10157	LXI	H,Public#Patch	
OEEA 225FD3	10158	SHLD	PF\$BDOS\$Exit\$Po	
OEED 21110F	10159	LXI	H, Prompt\$Patch	;Get address of intervening code
0EF0 2289C7	10160	SHLD	UP\$CCP\$Exit\$Poi	nt + 1
	10161			
0EF3 C9	10162	RET		;Return to enter CP/M
	10163 10164	;		
		;		
	10165 10166	; Public\$Patch:		0
		Public#Paten:		;Control arrives here from the BDOS
	10167			The BDOS is in the process of scanning
	10168			; down the target file name in the
	10169			; search next function
	10170 10171			; HL -> the name of the file searched for
	10172			; DE -> directory entry ; B = character count
	10172			; D - character count
0EF4 3A4200	10174	LDA	CDeDubliceEiles	Check if public files are to be enabled
0EF7 B7	10175	ORA	CB#Public#Files	Scheck it public tiles are to be enabled
OEF8 CAOBOF	10176	JZ	No\$Public\$Files	-No
UEFO CAUBUF	10177	32	MOALODIICALII62	; NO
OEFB 78	10178	MOV	A,B	:Get character count
OEFC B7	10179	ORA	м, <b>р</b> А	;Get character count ;Check if looking at first byte
	10180	UNA		; (that contains the user number)
OEFD C20BOF	10181	JNZ	NosPublicsFiles	; (that contains the user numper) ;No, ignore this patch
32, 5 32080i	10182	ONL	The College Hes	Anet should fully berein
0F00 1A	10183	LDAX	D	;Get user number from directory entry
OF01 FEE5	10184	CPI	0E5H	Check if active directory entry
OFO3 CAOBOF	10185	JZ		;Yes, ignore this patch
	10186			, ter, ignore this pater
0F06 7E	10187	MOV	A. M	;Get user number
0F07 B7	10188	ORA	A	;Check if User O
OFOS CA7CD3	10189	JZ	PF\$BDOS\$Char\$Ma	
	10190			
	10191	No\$Public\$Files	:	Replaced patched out code
0F08 78	10192	MOV	А, В	;Check if count indicates that
OFOC FEOD	10193	CPI	PF\$BD0S\$Unused\$1	Bytes ; registers are pointing at
	10194			; unused bytes field of FCB
OFOE C361D3	10195	JMP	PF\$BDOS\$Resume\$	
	10196	1		,
	10197	Prompt\$Patch:		Control arrives here from the CCP
	10198			The CCP is just about to get the
	10199			; drive id. when control gets here.
	10200			The CCP's version of CONOUT is used
	10201			; so that the CCP can keep track of
	10202			; the cursor position.
	10203			
OF11 3A4200	10204	LDA	CB\$Public\$Files	;Check if public files are enabled
OF14 B7	10205	ORA	A	
OF15 CA1DOF	10206	JZ	UP\$Private\$File:	s ;No
	10207			
0F18 3E50	10208	MVI	A, 'P'	
OF1A CD8CC4	10209	CALL	UP\$CCP\$CONOUT	;Use CCP's CONOUT routine
	10210			
	10211	UP\$Private\$File		• · · · · · ·
OFID CD13C5	10212	CALL		;Get current user number
OF20 FEOA	10213	CPI	9 + 1	;Check if one or two digits
0F22 D2300F	10214	JNC	UP\$2\$Digits	
0F25 C630	10215	ADI	101	;Convert to ASCII
APA3 63000	10216	UP\$1\$Digit:	UDACODACOUCUT	Out-ut the should be
OF27 CD8CC4	10217	CALL	UP\$CCP\$CONOUT	;Output the character
OF2A CDDOC5	10218	CALL	UP\$CCP\$Get\$Disk	
0F2D C38BC7	10219 10220	JMP	UP\$CCP\$Resume\$P	pint ;Return to CCP
	10220	; UP\$2\$Digits:		
0F30 C626	10221	OF#2#Digits: ADI	<pre>/01 - 10</pre>	Subtract 10 and convert to ASCII
0F30 C626	10222	PUSH	PSW 10	;Subtract 10 and convert to ASCII ;Save converted second digit
0F32 F5	10223	MVI	A, 11	;Save converted second digit ;Output leading '1'
0F35 CD8CC4	10224	CALL	UP\$CCP\$CONOUT	logran regard r
0F38 F1	10225	POP	PSW	:Recover second digit
0F39 C3270F	10228	JMP	PSW UP\$1\$Digit	;Output remainder of prompt and return to
VES7 CSEIVE	10228	One	0. 41401910	: the CCP
	10228			
	10230			
	10300	, ; #		
		F 17		

Figure 8-10. (Continued)

1	10301	•		
1	10302	; Configu	iration block get addres	15
	10303	;		
1	10304	🤉 This re		lity programs running in the TPA.
1	10305			it returns the address of a specific
1	10306		in the configuration bl	
1	10307		In the contractor bi	
	10308	3 Du 1144		y programs need not know the exact
1	10309		of the configuration bl	LOCK.
	10310			
	10311		parameters	
	10312	;		
	10313	;		ode (in effect, this is the
	10314	,	subscript of t	the object's address in the
	10315	1	table below)	
	10316			
	10317	, 2332322228282828		
	10318	CB#Get#Address		:<=== BIOS entry point (private)
	10319	1======================================		
OF3C F5	10320	PUSH	PSW	;Save user's registers
				JOAVE USER S REGISTERS
OF3D C5	10321	PUSH	B	
OF3E D5	10322	PUSH	D	
	10323			
OF3F 69	10324	MOV	L,C	;Make code into a word
OF40 2600	10325	MVI	н,о	
OF42 29	10326	DAD	н	Convert code into word offset
0F43 114F0F	10327	LXI	D,CB\$Object\$Table	Get base address of table
0F46 19	10328	DAD	D	<pre>;HL -&gt; object's address in table</pre>
0F47 5E	10329	MOV	Ĕ,M	Get LS byte
0F48 23	10330	INX	Н.	
0F49 56	10330	MOV	D, M	:Get MS byte
			D, M	
OF4A EB	10332	XCHG		;HL = address of object
	10333		_	
OF4B D1	10334	POP	D	;Recover user's registers
OF4C C1	10335	POP	в	
OF4D F1	10336	POP	PSW	
1	10337			
OF4E C9	10338	RET		
	10339	;		
	10400	; <b>#</b>		
	10401			
1	10402	, CB\$Object\$Table		
	10402	0.89001651414010		Code
			1	
1	10404	<b></b>		VV
OF4F 8FOF	10405	DW	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 BDOF	10408	DW	CB\$Forced\$Input	;04 forced input pointer
0F57 4300	10409			
		DW	CB\$Startup	;05 system startup message
		DW		
0559 5800	10410		CB\$Startup	;05 system startup message ; Redirection words
0F59 5800	10410 10411	DW	CB\$Startup CB\$Console\$Input	;03 system startup message ; Redirection words ;06
OF58 5A00	10410 10411 10412	DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output	;05 system startup message ; Redirection words ;06 ;07
0F5B 5A00 0F5D 5C00	10410 10411 10412 10413	DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input	j05 system startup message ; Redirection words ;06 ;07 ;08
0F5B 5A00 0F5D 5C00 0F5F 5E00	10410 10411 10412 10413 10414	DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input CB\$Auxiliary\$Output	;05 system startup message ; Redirection words ;07 ;08 ;09
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000	10410 10411 10412 10413 10414 10415	DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input CB\$Auxiliary\$Output CB\$List\$Input	;05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10
0F5B 5A00 0F5D 5C00 0F5F 5E00	10410 10411 10412 10413 10414 10415 10416	DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input CB\$Auxiliary\$Output	;05 system startup message ; Redirection words ;07 ;08 ;09
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200	10410 10411 10412 10413 10414 10415 10416 10417	DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input	;05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10 ;11
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200	10410 10411 10412 10413 10414 10415 10416 10417 10418	DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Input CB\$List\$Output	<pre>j05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10 ;10 ;11 sses ;12</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200	10410 10411 10412 10413 10414 10415 10415 10416 10417 10418 10419	DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Output CB\$Device\$Table\$Addres CB\$12\$24\$Clock	<pre>;05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10 ;11 sses ;12 ;13 Selects 12/24 hr. format clock</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200	10410 10411 10412 10413 10414 10415 10416 10417 10418	DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Input CB\$List\$Output	<pre>j05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10 ;10 ;11 sses ;12</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F65 6400 0F67 B500 0F67 BD00	10410 10411 10412 10413 10414 10415 10415 10416 10417 10418 10419	DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Output CB\$Device\$Table\$Addres CB\$12\$24\$Clock	<pre>;05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10 ;11 sses ;12 ;13 Selects 12/24 hr. format clock</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F69 BD00 0F69 BF00	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10421	DW DW DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Auxiliary\$Input CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Output CB\$Device\$Table\$Addres CB\$12\$24\$Clock RTC\$Ticks\$per\$Second RTC\$Watchdog\$Count	<pre>105 system startup message 106 107 108 109 109 110 111 sses ;12 113 Selects 12/24 hr. format clock 114 115</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F63 6400 0F67 B500 0F67 B500 0F68 BD00 0F6B BF00 0F6B C100	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10421 10422	DW DW DW DW DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Output CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$Device\$Table\$Addres CB\$12\$24\$Clock RTC\$Watchdog\$Count RTC\$Watchdog\$Count	<pre>;05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10 ;11 sses ;12 ;13 Selects 12/24 hr. format clock ;14 ;15 ;16</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F67 B500 0F68 BF00 0F6B BF00 0F6B C100 0F6F C300	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10421 10422 10423	DW DW DW DW DW DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$LostIlary\$Output CB\$List\$Input CB\$List\$Output CB\$Device\$Table\$Addres CB\$Device\$Table\$Address CB\$L2\$24&Clock RTC\$Tick\$\$per\$Second RTC\$Watchdog\$Count RTC\$Watchdog\$Address CB\$Function\$Ke\$Table	<pre>105 system startup message 106 107 108 109 109 10 10 111 sses ;12 113 Selects 12/24 hr. format clock 114 115 115 116 117</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F63 6400 0F67 B500 0F67 B500 0F68 BD00 0F6B BF00 0F6B C100	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10421 10422 10423 10424	DW DW DW DW DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Output CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$Device\$Table\$Addres CB\$12\$24\$Clock RTC\$Watchdog\$Count RTC\$Watchdog\$Count	<pre>;05 system startup message ; Redirection words ;06 ;07 ;08 ;09 ;10 ;11 sses ;12 ;13 Selects 12/24 hr. format clock ;14 ;15 ;16</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F67 B500 0F68 BF00 0F6B BF00 0F6B C100 0F6F C300 0F71 1B02	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10421 10422 10423 10424	DW DW DW DW DW DW DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Input CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Output CB\$Device\$Table\$Addres CB\$12\$24\$Clock RTC\$Tick\$per\$Second RTC\$Watchdog\$Count RTC\$Watchdog\$Cddress CB\$Function\$Key\$Table CONOUT\$Escape\$Table	<pre>;05 system startup message ; Redirection words ;06 ;07 ;08 ;10 ;11 sses ;12 ;13 Selects 12/24 hr. format clock ;14 ;15 ;16 ;17 ;18</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F68 B500 0F68 B500 0F68 B500 0F6B C100 0F67 C300 0F71 1802 0F73 8400	10410 10411 10412 10413 10415 10415 10415 10416 10417 10418 10417 10418 10420 10421 10422 10423 10424 10425 10426	DW DW DW DW DW DW DW DW DW DW DW DW DW	CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$List@Input CB\$List@Input CB\$List@Input CB\$Device\$Table\$Address CB\$12\$24\$Clock RTC\$Ticks\$per\$Second RTC\$Watchdog\$Count RTC\$Watchdog\$Address CB\$Function\$Key\$Table CONOUT\$Escape\$Table DO\$Initialize\$Stream	<pre>105 system startup message 106 107 108 109 109 110 111 sses ;12 13 Selects 12/24 hr. format clock 114 115 116 117 118 ;19</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F67 B500 0F67 B500 0F68 BD00 0F68 BF00 0F6B C100 0F67 C300 0F71 1B02 0F73 8400 0F75 9100	10410 10411 10412 10413 10414 10415 10416 10417 10418 10417 10420 10421 10422 10423 10424 10425 10424	DW DW DW DW DW DW DW DW DW DW DW DW DW D	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Output CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$Device\$Table\$Addres CB\$I2\$24\$Clock RTC\$Ticks\$per\$Second RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Address CB\$Function\$Key\$Table CONOUT\$Escape\$Table DO\$Initialize\$Stream D0\$Baud\$Rate\$Constant	<pre></pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F63 6200 0F63 6200 0F65 6400 0F67 B500 0F69 B500 0F69 B500 0F6B BF00 0F6B C300 0F71 1B02 0F73 8400 0F75 9100	10410 10411 10412 10413 10414 10415 10415 10415 10417 10418 10417 10420 10421 10422 10423 10424 10425 10425 10425	DW DW DW DW DW DW DW DW DW DW DW DW DW D	CB\$Startup CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$Console\$Output CB\$LostInput CB\$List\$Output CB\$List\$Output CB\$Device\$Table\$Address CB\$L2\$24&Clock RTC\$Vick\$Per\$Second RTC\$Vick\$Per\$Second RTC\$Viction\$Key\$Table CONOUT\$Escape\$Table Do\$Initialize\$Stream D0\$Baud\$Rate\$Cconstant D1\$Initialize\$Stream	<pre>105 system startup message 106 107 108 109 109 10 111 sses ;12 113 Selects 12/24 hr. format clock 114 115 116 117 118 119 120 121</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F67 B500 0F68 BF00 0F6B BF00 0F6F C300 0F74 1802 0F73 8400 0F77 9100 0F77 9400	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10422 10423 10422 10423 10424 10425 10426 10427		CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Auxiliary\$Dutput CB\$Auxiliary\$Dutput CB\$List\$Input CB\$List\$Input CB\$List\$Output CB\$Device\$Table\$Address CB\$12\$24\$Clock RTC\$Ticks\$per\$Second RTC\$Watchdog\$Address CB\$Function\$key\$Table CONOUT\$Escape\$Table D0\$Initialize\$Stream D0\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant	<pre>105 system startup message 106 system startup message 106 107 108 109 109 110 111 sses :12 113 Selects 12/24 hr. format clock 114 115 116 117 118 119 120 121 122</pre>
0F5B 5A00           0F5D 5C00           0F5F 5E00           0F61 6000           0F63 6200           0F65 8500           0F67 8500           0F67 8500           0F68 8500           0F68 8500           0F68 8500           0F67 8500           0F67 8500           0F67 8500           0F73 8400           0F75 9100           0F77 9400           0F78 A400           0F78 A400	10410 10411 10412 10413 10414 10415 10416 10417 10418 10417 10420 10421 10421 10422 10423 10424 10425 10425 10426 10425 10426 10429 10430	DW DW DW DW DW DW DW DW DW DW DW DW DW D	CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$Lost1iary\$Output CB\$List\$Input CB\$List\$Input CB\$Device\$Table\$Addres CB\$List\$Output CB\$Device\$Table\$Addres CB\$List\$Cock RTC\$Tick\$\$per\$Second RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count CB\$Function%key\$Table CONOUT\$Escape\$Table Do\$Initialize\$Stream D0\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Initialize\$Stream	<pre>105 system startup message 106 107 108 109 100 111 sses ;12 113 Selects 12/24 hr. format clock 114 115 115 116 117 118 119 120 121 122 123</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F67 B500 0F68 BF00 0F6B BF00 0F6F C300 0F74 1802 0F73 8400 0F77 9100 0F77 9400	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10422 10423 10422 10423 10424 10425 10426 10427		CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$Lost@Input CB\$Lost@Input CB\$Lost@Input CB\$Device\$Table\$Addres CB\$Device\$Table\$Addres CB\$Lost@Cont RTC\$Watchdog\$Address CB\$Function\$Key\$Table CONOUT\$Escape\$Table DO\$Initialize\$Stream D0\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant	<pre>105 system startup message 106 system startup message 106 107 108 109 100 111 115 114 115 114 115 116 117 118 119 120 121 122 123 124</pre>
0F5B 5A00           0F5D 5C00           0F5F 5E00           0F61 6000           0F63 6200           0F65 8500           0F67 8500           0F67 8500           0F68 8500           0F68 8500           0F68 8500           0F67 8500           0F67 8500           0F67 8500           0F73 8400           0F75 9100           0F77 9400           0F78 A400           0F78 A400	10410 10411 10412 10413 10414 10415 10416 10417 10418 10417 10420 10421 10421 10422 10423 10424 10425 10425 10426 10425 10426 10429 10430	DW DW DW DW DW DW DW DW DW DW DW DW DW D	CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Input CB\$Device\$Table\$Address CB\$List\$Cock RTC\$Ticks\$per\$Second RTC\$Watchdog\$Address CB\$Function\$Key\$Table CONOUT\$Escape\$Table DO\$Initialize\$Stream D0\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant	<pre>105 system startup message 106 system startup message 106 107 108 109 109 110 111 10 111 10 114 115 114 115 116 117 118 119 120 121 122 123 124 125</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F67 B500 0F68 BF00 0F6B C100 0F6B C100 0F6F C300 0F71 1B02 0F73 8400 0F75 9100 0F77 9400 0F77 9400 0F78 B100	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10422 10422 10422 10423 10424 10425 10426 10427 10426 10427 10426	DW DW DW DW DW DW DW DW DW DW DW DW DW D	CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$Lost@Input CB\$Lost@Input CB\$Lost@Input CB\$Device\$Table\$Addres CB\$Device\$Table\$Addres CB\$Lost@Cont RTC\$Watchdog\$Address CB\$Function\$Key\$Table CONOUT\$Escape\$Table DO\$Initialize\$Stream D0\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant	<pre>105 system startup message 106 system startup message 106 107 108 109 100 111 115 114 115 114 115 116 117 118 119 120 121 122 123 124</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F68 B500 0F68 B500 0F68 B500 0F60 C100 0F67 C300 0F71 1802 0F73 8400 0F77 9400 0F77 9400 0F77 8400 0F77 B100 0F7F 4002 0F7F 4002	10410 10412 10412 10413 10414 10415 10416 10417 10428 10421 10421 10422 10423 10424 10425 10426 10427 10428 10429 10430	DW DW DW DW DW DW DW DW DW DW DW DW DW D	CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Auxiliary\$Output CB\$List\$Input CB\$List\$Input CB\$List\$Input CB\$Device\$Table\$Address CB\$List\$Cock RTC\$Ticks\$per\$Second RTC\$Watchdog\$Address CB\$Function\$Key\$Table CONOUT\$Escape\$Table DO\$Initialize\$Stream D0\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant	<pre>105 system startup message 106 system startup message 106 107 108 109 109 110 111 10 111 10 114 115 114 115 116 117 118 119 120 121 122 123 124 125</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F67 B500 0F68 BF00 0F6B C100 0F6F C300 0F77 1B02 0F73 8400 0F77 9100 0F77 9400 0F77 9400 0F78 A400 0F78 B400 0F77 B100 0F7F B100 0F7F 802 0F78 B80F	10410 10411 10412 10413 10414 10415 10416 10417 10418 10419 10420 10422 10423 10422 10423 10424 10425 10427 10428 10429 10430 10431		CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$List@Input CB\$List@Input CB\$List@Input CB\$List@Output CB\$Device\$Table\$Address CB\$12\$24\$Clock RTC\$Histchdog\$Cont RTC\$Histchdog\$Cont RTC\$Histchdog\$Address CB\$Function\$Key\$Table CONOUT\$Escape\$Table DO\$Initialize\$Stream D0\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D1\$Baud\$Rate\$Constant D2\$Initialize\$Stream D2\$Baud\$Rate\$Constant D2\$Initialize\$Stream D2\$Baud\$Rate\$Constant D2\$Initialize\$Stream D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant D2\$Baud\$Rate\$Constant Interrupt\$Vector LTCB\$Offset LTCB\$Length	<pre>105 system startup message 106 system startup message 106 107 108 109 109 110 111 sses :12 113 Selects 12/24 hr. format clock 114 115 116 117 118 119 120 121 122 123 124 125 124 125 126 127</pre>
0F5B 5A00 0F5D 5C00 0F5F 5E00 0F61 6000 0F63 6200 0F65 6400 0F67 B500 0F68 B500 0F68 B500 0F68 B500 0F60 C100 0F67 C300 0F71 1802 0F73 8400 0F77 9400 0F77 9400 0F77 8400 0F77 B100 0F7F 4002 0F7F 4002	10410 10411 10412 10413 10414 10415 10415 10415 10416 10417 10418 10420 10421 10422 10423 10424 10425 10426 10426 10429 10430 10431 10432		CB\$Startup CB\$Console\$Input CB\$Console\$Input CB\$Console\$Output CB\$Console\$Output CB\$Lost@Output CB\$List@Output CB\$List@Output CB\$Lest@Output CB\$Lest@Output CB\$Lest@Cock RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Watchdog\$Count RTC\$Wa	<pre>105 system startup message 106 system startup message 106 107 108 109 10 111 11 11 11 11 11 11 11 11 11 11 11</pre>

Figure 8-10. (Continued)

OF87 A421		0436		DW	Multi\$Co	ommand\$B	uffer	; 31
		0437 ;						
		)500 ; <b>†</b>	ŧ					
		)501 ;		The shou	rt term o	configura	ation blo	ck.
	10	0502 ;						
	10	503 ;		This cou	ntains va	ariables	that can	be set once CP/M
	10	504 ;		has been	n initiat	ted, but	that are	never preserved
		505 ;						next. This part of
		506 ;						last initialized bytes
		507 ;		in the 1		on brock		last initialized bytes
		508						
				The tue	values b		. used by	utility sussume that
				The two	values	Jerow are		utility programs that
					reau in	the long	g term co	nfiguration block from disk.
				Ine BIU	5 starts	on a 250	s-byte pa	se boundary, and therefore
		512 ;						tor boundary in the reserved
		513 ;						ram can then, using the
		514 ;						, determine how many 128-byte
		515 ;		sectors	need to	be read	in by th	e formula:
		516 ;						
		)517 ;			(LCTB\$01	ffset + L	_TCB\$Leng	th) / 128
		)518 ;						
		)519 ;						om the start of the BIOS to
		520 ;		where th	ne first	byte of	the long	term configuration block
		521 1						length, the utility can
		)522 ;						over the disk image
		523 ;						and then write the
	10	524 ;		updated	LTCB bac	k onto t	the disk.	
	10	)525 ;						
OF89 BED9	<b>7</b> 10		CB\$Off	set:	DW	BIOS\$Ent	try - Lone	3\$Term\$CB
OF88 E601			CB\$Len					- Long\$Term\$CB
		528 ;		-				-
		529		Forced i	input poi	inter		
		530 ;						
		531 ;		IT CONTR	N ever fi	inds that	this po	inter is pointing to a nonzero
		532 ;		byte. th	nen this	hvte wil	ll he ini	ected into the console input
		533						ed on the console. The
		534 ;						ext byte in memory.
		535						
OF8D 4300			SEorce	d\$Input:		DW	CB#Starts	
0,00 4000		537 ;				2		-F
		538 ;						
			tei			+Current	t system (	late
0F8F 3130	2F313710	540		DB	10/17/8	21.LF	:Unless	otherwise set to the contrary
		541						is the release date of the system
		542						, it will be set by the DATE utility
0F98 00		543		DB	0			terminator
0, 70, 00		544			•		,	
			me\$in\$	45011.		• Current	t system '	lime
0F99 3030		546 HH		DB	1001	, current	Hours	
0F9B 3A		1546 nn		DB	· · ·		, 1041 \$	
0F9C 3030	, ,,	548 MM		DB	1001		;Minutes	
0F9E 3A		1348 mm 1549		DB DB	.00		, minutes	
0F9F 3030		550 SS		DB	100/		:Seconds	
UF 7F 3030				DD ASCII\$Er				n updating the time
OFA1 OA		552					,useu whi	in apparting the time
0FA2 00		553		DB	0		100-but-	terminator
UPM2 00				00	v		,00-0,10	
		554 ; 555 ;						
				. #E1				ins two flags that are used
	10		meşuat	e\$Flags:				
						: to ir		
		557						ether the time and/or date
	10	557 558				; have	been set	either programmatically or
	10 10	557 558 559				; have ; by us	been set sing the	either programmatically or FIME and DATE utilities. These
	10 10 10	557 558 559 560				; have ; by us ; flags	been set ing the can be f	either programmatically or FIME and DATE utilities. These tested by utility programs that
	10 10 10 10	557 558 559 560 561			<u>,</u>	; have ; by us ; flags	been set ing the can be f	either programmatically or FIME and DATE utilities. These
0FA3 00	10 10 10 10 10	557 558 559 560 561 562		DB	0	; have ; by us ; flags ; need	been set ing the can be to have	either programmatically or FIME and DATE utilities. These tested by utility programs that
0001 =	10 10 10 10 10	557 558 559 560 561 562 563 Ti	me\$Set		EQU	<pre>; have ; by us ; flags ; need 0000\$000</pre>	been set sing the can be to have 01B	either programmatically or FIME and DATE utilities. These tested by utility programs that
	10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da			EQU	; have ; by us ; flags ; need	been set sing the can be to have 01B	either programmatically or FIME and DATE utilities. These tested by utility programs that
0001 =	10 10 10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da 565	me\$Set		EQU	<pre>; have ; by us ; flags ; need 0000\$000</pre>	been set sing the can be to have 01B	either programmatically or FIME and DATE utilities. These tested by utility programs that
0001 =	10 10 10 10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da 565 565	me\$Set te\$Set		EQU	<pre>; have ; by us ; flags ; need 0000\$000</pre>	been set sing the can be to have 01B	either programmatically or FIME and DATE utilities. These tested by utility programs that
0001 =	10 10 10 10 10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da 565 565 565 566 ; 700 ;#	me\$Set te\$Set		EQU EQU	; have ; by us ; flags ; need 0000\$000	been set ing the can be to have 01B 00B	either programmatically or FIME and DATE utilities. These tested by utility programs that
0001 =	10 10 10 10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da 565 566 ; 700 ; <b>#</b> 701 ;	me\$Set te\$Set		EQU	; have ; by us ; flags ; need 0000\$000	been set ing the can be to have 01B 00B	either programmatically or FIME and DATE utilities. These tested by utility programs that
0001 =	10 10 10 10 10 10 10 10 10 10 10	557 558 559 560 561 562 564 Da 565 564 F 565 566 ; 700 ; <b>#</b> 701 ; 702 ;	me\$Set te\$Set	Uninitia	EQU EQU Alized bu	; have ; by us ; flags ; need 0000\$000 0000\$001	been set sing the f s can be f to have f 01B 00B	either programmatically or TIME and DATE utilities. These ested by utility programs that the correct time and date set.
0001 =	10 10 10 10 10 10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da 565 565 565 565 566 ; 7700 ; # 7701 ; 7702 ; 7703 ;	me\$Set te\$Set	Uninitia With the	EQU EQU alized bu	; have ; by us ; flags ; need 0000\$000 000\$000	been set sing the ' s can be ' to have ' DIB OB COB	either programmatically or FIME and DATE utilities. These ested by utility programs that the correct time and date set.
0001 =	10 10 10 10 10 10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da 565 566 ; 700 ; # 701 ; 702 ; 702 ; 703 ; 704 ;	me\$Set te\$Set	Uninitia With the bytes of	EQU EQU alized bu e excepti f code, a	<pre>; have ; by us ; flags ; need 0000\$000 0000\$000 uffer are .on of th ull of th</pre>	been set sing the s can be s to have s DIB COB eas ne main D: ne other o	either programmatically or IIME and DATE utilities. These ested by utility programs that the correct time and date set. isk\$Buffer, which contains a few uninitialized variables
0001 =	10 10 10 10 10 10 10 10 10 10 10 10 10	557 558 559 560 561 563 Ti 564 Da 565 565 565 565 565 7700 ;# 7701 ; 7703 ; 7703 ; 7703 ;	me\$Set te\$Set	Uninitia With the bytes of occur he	EQU EQU alized bu excepti code, a ere. This	; have ; by us ; flags ; need 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 000 0000\$000 0000\$000 000 0000 0000\$000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 0000	been set sing the ' can be ' to have ' DIB HOB HOB HOB HOB HOB HOB HOB HOB HOB HO	either programmatically or FIME and DATE utilities. These lested by utility programs that the correct time and date set. isk\$Buffer, which contains a few uninitialized variables of reducing the number of
0001 =	10 10 10 10 10 10 10 10 10 10 10 10 10	557 558 559 560 561 562 563 Ti 564 Da 565 566 ; 700 ; # 701 ; 702 ; 702 ; 703 ; 704 ;	me\$Set te\$Set	Uninitia With the bytes of occur he	EQU EQU alized bu excepti code, a ere. This	; have ; by us ; flags ; need 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 0000\$000 000 0000\$000 0000\$000 000 0000 0000\$000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 0000	been set sing the ' can be ' to have ' DIB HOB HOB HOB HOB HOB HOB HOB HOB HOB HO	either programmatically or IIME and DATE utilities. These ested by utility programs that the correct time and date set. isk\$Buffer, which contains a few uninitialized variables

Figure 8-10. (Continued)

OFA4 21A4 - OFA4	-	10707 10708 10709 10800 10801 10803 10803 10804 10805 10806 10807 10808 10809 10810	; ; ; <b>*</b> ; ; ; ; ; ;	he cold t can b herefor	boot e over	initial written		le is on:	be kept on the d	Jisk.
21A4 :		10709 10800 10801 10802 10803 10803 10804 10805 10806 10807 10808 10809 10810 10811	f ;# ; T ; I ; I ; T ; ; ; ; ; ; ;	t can b herefor	e over	written			ly needed once.	
21A4 :		10800 10801 10802 10803 10804 10805 10806 10807 10808 10809 10810	;# ; T ; I ; T ; ; Disk\$buff	t can b herefor	e over	written			ly needed once.	
21A4 :		10801 10802 10803 10804 10805 10806 10807 10808 10809 10810 10811	; T ; I ; T ; T ; ; Disk\$buff	t can b herefor	e over	written			ly needed once.	
21A4 :		10802 10803 10804 10805 10806 10807 10808 10809 10810 10811	; T ; I ; T ; ; Disk\$buff	t can b herefor	e over	written			ly needed once.	
21A4 :	-	10803 10804 10805 10806 10807 10808 10809 10810 10811	i I ; T ; ; Disk\$buff	t can b herefor	e over	written			ay needed Unce.	
21A4 :	-	10804 10805 10806 10807 10808 10809 10810 10811	; T ; ; Disk\$buff	herefor				se høær 4	evecuted.	
21A4 :		10805 10806 10807 10808 10809 10810 10811	; ; Disk\$buff		•, 10				in disk buffer.	
21A4 :	=	10806 10807 10808 10809 10810 10810		'er:			Deu. Tuerne	e criter mat.	In disk burrer.	
21A4 :	-	10807 10808 10809 10810 10811		er:						
21A4 :	-	10808 10809 10810 10811			DS	Physi	as14Sector4	Size a I	Physical\$Sec\$Per\$1	Track
	-	10809 10810 10811	,		53	,				II WCK
	-	10810 10811						+Save t	he location counte	er
		10811	After\$Dis	k\$Buffe	~	EQU	•		rrent value of loc	
OFA4			;	K+Buille	•	240	•	, • - cu	inent value of iot	
VI    1		10812	,			ORG	Disk\$Buf	for	Wind the location	on counter back
		10813				••••			,	
		10814	, Initializ		<b>m</b> •	. This	stream of	data is	used by the	
		10815							e. It has the foll	lowing
		10816					rmat:			
		10817								
		10818				;	DB	Port nu	mber to be initial	lized
		10819				,	DB		of byte to be out	
		10820				;	DB		x,xx data to be out,	
		10821				í	:			
		10822				í	:			
		10823				÷	DB	Port nue	mber of OOH termin	nates
		10824				÷				
		10825				÷				
		10826	,							
		10827		nitiali	zation	stream	declared h	iere		
OFA4		10828				11\$Port			59 interrupt cont	roller
OFA5		10829	Ē		1					
OFA6		10830	r	)B	IC\$ICW	11				
0.710		10831	-			-				
OFA7 I		10832	n	B	ICSICW	2\$Port				
OFAS (		10833	Ē		1					
OFA9		10834			IC\$ICW	12				
		10835								
OFAA J	09	10836	D	)B	IC\$OCW	11\$Port				
OFAB	01	10837	E	)B	1					
OFAC I	FC	10838	D	B	IC\$OCW	11				
		10839								
OFAD	33	10840	D	)B	83H			;Program	m the 8253 clock o	generator
OFAE (	01	10841	D	)B	1					
OFAF :		10842	0	)B	00\$11\$	010\$0B		;Counter	r O, periodic inte	errupt, mode 2
		10843								
OFBO (		10844			80H			;RTC use	es channel O	
OFB1 (		10845			2					
OFB2 (		10846	c	W	17921				* 930 nanoseconds	
		10847							66 milliseconds).	60 ticks/sec.
OFB4 (		10848	0	)B	0		;Port nu	imber of	O terminates	
		10849	;							
		10850	;							
		10851	Signon\$Me							
	13502F4D20				CP/M		_			
OFBE 3	3030	10853			VERSIO	IN	;Current	t versio	n number	
OFCO 2	20	10854		B						
OFC1		10855			MONTH		;Current	t date		
OFC3 2		10856			11					
OFC4	3236	10857			DAY					
OFC6	2F	10858			<u></u>					
OFC7		10859			YEAR					
		10860			CR, LF,			-		
	156E68616E						S', CR, LF, LF			
	469736B20			)B 18	· U1SK	LONTIGU	ration : ,C	K, LF, LF		
	2020202020					A: 0.35	Mbyte 5" F	loppy ,	UR,LF CB / F / F	
	2020202020			B	,	BI 0.35	Mbyte 5" F	TOPPY',	UR,LF,LF CD LF	
	2020202020					0.24	Mbyte 8" F Mbyte 8" F	ioppy',	LK,LF	
	2020202020			)B		DI 0.24	- moyte 8" #	TOPPY',	UR;LF / CD   E   E	
1090 3	2020202020			)B	-	mi 0.19	Mbyte Memo	ary UISK	, UR, LF, LF	
1000		10868	,		^					
1080 (		10869	-	98	0					
		10870	1		6 m	en cele				
		10871	•	lessages	TOP M	19U1SK				
		10872	;							

Figure 8-10. (Continued)

		10873	M\$Disk\$Setup			
108E	2020202020		DB	<ul> <li>Ma</li> </ul>		contains valid information.',CR,LF,O
1000	2020202020	10875	M\$Disk\$Not\$S DB			n initialized to empty state.',CR,LF,O
		10877	, 55	114	DISK HAS DEE	n Initialized to empty state: Jok, Er Jo
		10878	M\$Disk\$Dir\$E	ntry:		my directory entry used to determine
		10879				f the M\$Disk contains valid information
LOF3		10880	DB DB	15 (MCD/ -)	;Use	r 15
	4D24446973 A0A020	10882	DB	1M\$Disk	· ·+80H, · ·	;System and read/only
		10883	DB	0,0,0,0		, oystem and read/only
	0000000000	10884	DB		0,0,0,0,0,0,0,	0,0,0,0,0,0
		10885	1			
0004		10886	Default\$Disk	EQU	0004H ;Def	ault disk in base page
		10888	BOOT:	Fintered	directly fr	om the BIOS JMP Vector
		10889				nsferred here by the CP/M
		10890			strap loader	
		10891		;	<b>.</b> .	
		10892			; [n]	tialize system s routine uses the Initialize\$Stream
		10894			1111 ( b =	eclared above
		10895			, -	
113		10896	DI			able interrupts to prevent any
		10897				de effects during initialization
	21A40F	10898	LXI		lize\$Stream	;HL -> data stream
1117	CD1903	10899	CALL	output\$E	3yte\$Stream	;Output it to the specified ; ports
		10900				, porta
111A	CDEE02	10902	CALL	General	CIO\$Initiali	zation ;Initialize character devices
		10903				
	21850F	10904	LXI		\$Message	;Display sign-on message on console
1120	CD5F02	10905	CALL	Display <b>\$</b>	Message	
1122	CDDFOE	10905	; CALL	Patch\$CF	- Mak	e necessary patches to CCP and BDOS
	CDDI OL	10908	Onet	i acciteci		or custom enhancements
		10909				
		10910				tialize M\$Disk
		10911 10912				the M\$Disk directory has the pecial reserved file name "M\$disk"
		10912				with lowercase letters and marked
		10914				YS and R/O), then the M\$Disk is
		10915			; a	ssumed to contain valid data.
		10916				the "M\$Disk" file is absent, the
		10917 10918			9 M	<pre>\$Disk Directory entry is moved into \$2.200 Birls into \$2.200 /pre>
		10918			, ,	he M\$Disk image, and the remainder of he directory set to OE5H.
1126	0601	10920	MVI	B.1		ect bank 1
	CDDDOB	10921	CALL			hich contains the M\$Disk directory
		10922				
		10923				ck if M\$Disk directory entry present
	210000	10924	LXI	H,0 D MeDicl		rt address for first directory
	11F310 0E20	10925 10926	LXI MVI	0,M\$015K C,32	sDir\$Entry:	gth to compare
	~~=~	10927	M\$Disk\$Test:	0,02	,	g
1133		10928	LDAX			byte from initialized variable
1134		10929	CMP	M		pare with M\$Disk image
1135	C24F11	10930	JNZ INX	M\$Disk\$N D	lot\$Setup	;Match fails
1138 1139		10931	INX	H		
1139 113A		10932	DCR	Ċ		
	CA4111	10934	JZ	M\$Disk\$9	Setup ;All	bytes match
	C33311	10935	JMP	M\$Disk\$7	ſest	
		10936	;			
	210510	10937	M\$Disk\$Setup			an Taform user
1141	218E10	10938 10939	LXI	n, mauisk	.*>e.rdh.au6229	ge ;Inform user
		10940	, M\$Disk\$Setup	\$Done:		
1144	CD5F02	10941	CALL		Message	
		10942				
1147		10943	XRA	A		default disk drive to A:
	320400	10944	STA EI	Default		errupts can now be enabled
1148	гø	10945	E.1		1111	CITURES CON NOW DE ENGUIRO
	C36C02	10947	JMP	Enter\$CF	۲M :Go	into CP/M
1140						

Figure 8-10. (Continued)

	10949	M\$Disk\$Not\$Setu					
114F 110000	10950	LXI	D,0			Bisk directory entry into	
1152 21F310	10951	LXI		k\$Dir\$En	try	; M\$Disk image	
1155 OEO4	10952	MVI	C,32/8			;Number of 8-byte blocks to move	
1157 CDF80A	10953	CALL	Move\$8				
	10954	;			DE N		
	10955					next byte after M\$Disk directory y in image	
	10956 10957	MVI	A, 0E5H		; entr	;Set up to do memory fill	
115A 3EE5	10957	STAX	A, UESH			Store first byte in "source" area	
115C 12 115D 62	10959	MOV	Й, D			;Set HL to DE +1	
	10960	MOV	L,E			, Set HE to be th	
115E 6B 115F 23	10961	INX	L, L				
1160 OEFC	10962	MVI		10241 -	32) / 8	;Two allocation blocks	
1160 VEFC	10963		0,112 4	1024/	02/ / 0	; less 32 bytes for M\$Disk entry	
1162 CDF80A	10964	CALL	Move\$8			;Use Move\$8 to do fill operation	
	10965					,	
1165 210010	10966	LXI	H.M\$Dis	k\$Not\$Se	tup\$Mess	age	
1168 C34411	10967	JMP		Setup\$Do		;Output message and enter CP/M	
	10968	;					
	10969	7					
116B 00	10970	DB	0		; Dummy		
	10971	Last#Initialize	ed\$Byte:		;<== ad	dress of last initialized byte	
	10972	,					
	10973	; End of	cold boo	ot initia	lization	code	
	10974	;					
21A4	10975	ORG	After\$E	)isk\$Buff	er	Reset location counter	
	10976	;					
21A4	10977	Multi\$Command\$	Buffer:	DS	128	;This can be used to insert long	
	10978					; command sequences into the	
	10979					console input stream by setting	
	10980					; the forced input pointer here	
	10981 10982	; D0\$Buffer\$Leng		EQU	32	Must be binary number	
0020 = 2224	10982	DOSBuffer:			er\$Lengt		
2224	10983	DOBBUTTER:	05	DOBBUIL	erscengt	.11	
0020 =	10985		• •	EQU	32	;Must be binary number	
2244	10985	DisBuffer:	DS		er\$Lengt		
2277	10987	bittbarrer.	00	DIODU	er twengt		
0020 =	10988	D2\$Buffer\$Leng	* 5	FOU	32	;Must be binary number	
2264	10989	D2\$Buffer:	DS	D25Buff	er\$Lengt		
2204	10990						
	10991		reas for	the char	acter dr	ivers	
	10992			••••			
2284	10993	PI\$User\$Stack:	DS	2	;Storag	e area for user's stack pointer	
	10994					an interrupt occurs	
2286	10995	PI\$User\$HL:	DS	2		area for user's HL	
2288	10996		DS	40		area for use by interrupt service	
	10997	PI\$Stack:				ines to avoid overflowing the	
	10998				; user	's stack area	
	10999	1				<b></b>	
22B0	11000	Directory\$Buff	eri	DS	128	;Disk directory buffer	
	11001	<b>;</b>				• • • • • • • • • • • • • •	
2330	11002	M\$Disk\$Buffer:		DS	128	#Intermediary buffer for	
	11003					9 M\$Disk	
	11004	t Déale a					
	11005		ork areas	•			
	11006 11007	; There	are used	hy the F	nos to d	letect any unexpected	
	11007		are used of diete	uy ine s itas Th	BDOS W	vill automatically set	
	11008					-only status.	
	11010	y such et	CHANARO	GI SKELLE			
2380	11011	, Disk\$A\$Workare	a 1	DS	32	: A:	
2300	11012	Disk\$B\$Workare		DS	32	; B:	
23F0	11012	Disk\$C\$Workare		DS	16	, D. ; C:	
2400	11013	Disk\$D\$Workare	 a:	DS	16	; D:	
	11015	:			. –	• •	
	11016	;					
	11017		llocation	n vectors			
	11018	,					
	11019	; These	are used	by the E	BDOS to m	maintain a bit map of	
			allocatio	n blocks	are use	ed and which are free.	
	11020	; which	arrocacic				
		; which ; One by	te is use	d for ei	ght allo	ocation blocks, hence the	
	11020	; Oneby	te is use	d for ei	ght allo	cation blocks, hence the tion blocks/8)+1.	
	11020 11021	; Oneby	te is use	d for ei	ght allo (allocat	cation blocks, hence the tion blocks/8)+1.	
2410	11020 11021 11022	; One by ; expres	te is use sion of t	ed for ei the form	ght allo	cation blocks, hence the	



2426	11025	Disk\$B\$Allocation\$Vector	DS	(174/8)+1	; B:
	11026	;			
2430	11027	Disk\$C\$Allocation\$Vector	DS	(242/8)+1	; C:
245B	11028	Disk\$D\$Allocation\$Vector	DS	(242/8)+1	5 D1
	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)

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 2 deal with this class of errors. . 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. 0000 = B\$System\$Reset EQU 0 **;BDOS system reset function** 0005 = BDOS 5 **;BDOS entry point** EQU 0000 00 Printer#Timeout#Flag: DB Ő. ;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 = Printer\$Delay\$Count EQU 1800 ;Given a clock period of 16.666 ms ; 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\$Mes				
0003	0D0A 5072696E74 436865636E			CR,LF 'Printe			for too long,',CR,LF ine and ready.',CR,LF,O
004E	00	; 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	79 324E00			A,C Printer	\$Charac	ter	;Save the data character
	010807	Printer	Retry: LXI	B,Print	er\$Dela	iy\$Count	;This is the count of the number ; of clock ticks before the watchdog : subroutine call
	217E00 CDA300			H,Print Set\$Wat		d\$Out	; <== this address ;Sets the watchdog running
005C	CDA300	Printer:		Get\$Pri	nter\$St	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
005F	C26C00		JNZ	Printer	\$Ready		; waiting for XON, ACK, or DTR to ; come high ;The printer is now ready
0062	340000		LDA	Printer	\$Timeou	it\$Flag	<pre>;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)</pre>
0065 0066	B7 C28400		ora JNZ	A Display	\$Busy\$M	lessage	;Yes, so display message to ; indicate an error has occurred
0069	C35C00		JMP	Printer	\$Wait		Otherwise, check if printer is ; now not busy
		Printer	Ready:				;The printer is now ready to output ; a character, but before doing so, ; the watchdog timer must be reset
	010000 CDA300		DI LXI CALL EI	B,0 Set\$Wat	chdo <b>g</b>		;Ensure no false timeout occurs ;This is done by setting the count ; to zero
0077	3A4E00 11A300 CDA300		LXI	Printer D,Print Output\$	er\$Devi	ce\$Table	;Get character to output ;DE -> device table for printer ;Output the character to the printer
007D	C9	,	RET				;Return to the BIOS's caller
		;	\$Timed\$Ou	it:			;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
	3EFF 320000 C9		MVI STA RET	A,OFFH Printer	\$Timeou	it\$F1ag	;Set 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
                                Printer$Timeout$Flag
                        STA
0088 210100
                        LXI
                                H, Printer$Busy$Message
                                                         ;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
:Check if Retry
0091 FE52
                        CPI
                                1R1
0093 CA5300
                        JZ
                                Printer$Retry
0096 FE41
                        CPI
                                                         ;Check if Abort
0098 CA9E00
                        JZ
                                Printer$Abort
009B FE49
                        CPI
                                11.
                                                         :Check if Ignore
009D C8
                        R7
               .
Printer$Abort:
009E 0E00
                                C,B$System$Reset
                        MVI
                                                         :Issue system reset
00A0 C30500
                        JMP
                                BDOS
                                                         ;No need to give call as
                                                            control will not be returned
                                                         .
                        Dummy subroutines
                        These are shown in full in Figure 8-10. The line numbers in
                        Figure 8-10 are shown in the comment field below
               Printer*Device*Table:
                                                 ;Line 01300 (example layout)
               Request$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
```

Figure 9-1. Serial printer error handling (continued)

## **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.

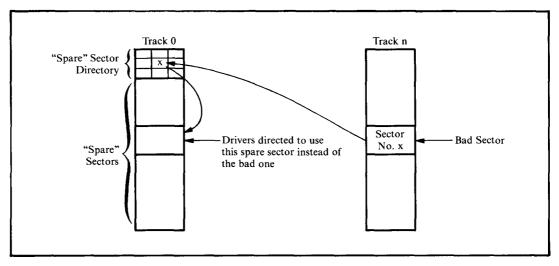


Figure 9-2. Sector sparing

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.

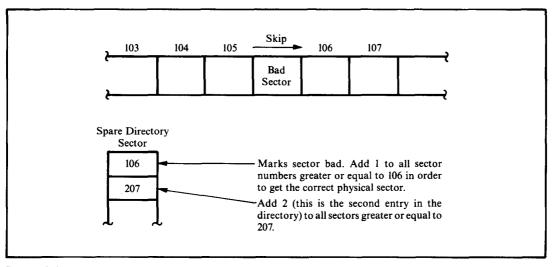


Figure 9-3. Sector skipping

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.

		ment ba	d sector			ions to be made in order nt using sector sparing
	;					
0000 =	False	EQU	0			
FFFF =	True s	EQU	Not Fal	se		
0000 =	Sector#Sparing	EQU	False			
FFFF =	Sector\$Skipping ; ;	EQU	Not Sec	tor <b>s</b>	iSpari	ng
	Addition	al equa	tes and o	defi	nitio	ns
	, Spare\$Directorie	51		;Ta		f spare directory addresses ;Note: The directories themselves ; are declared at the end of the ; BIOS
0000 D500	DW	Spare\$D:	irectory	\$0		Physical disk O
0002 9701	DW	Spare\$D	irectory	\$1		Physical disk 1
	Spare\$Dir\$In\$Mem	orvi		3F1	ags u	sed to indicate whether spare
0004 00	DB	0				tory for a given physical disk
0005 00		Ó				een loaded into memory. Set by SELDSK
	;					
0000 =	Spare\$Track		EQU	0		;Track containing spare directory ; sectors
0004 =	Spare\$Sector		EQU	4		Sector containing directory
0005 =	First\$Spare\$Sect	or	EQU	Spa		ctor + 1
	; Variable ;					
	Selected\$Spare\$D	lirectory				
0006 0000			DW	0		Pointer to directory
0008 00	Selected\$Disk:		DB	0		Logical disk number
0009 00	Disk\$Type:		DB	0		Floppy/hard disks
000A 00	Deblocking\$Requi		DB	Ô		;Deblocking flag
0008 00	Selected\$Physica	l\$Disk:	DB	0		Physical disk number
0000 0000	Disk\$Track:	DW	0	;)	These	variables are part of the command
000E 00		DB	ŏ			handed over to the disk controller

Figure 9-4. Bad sector management

```
Maximum$Track
8000 =
                                          FOU
                                                   32768 ;Used as a terminator
0012 =
                                          EQU
                Sectors$Per$Track
                                                   18
0000 =
                                          EQU
                First$Sector$On$Track
                                                   ٥
                Disk$Parameter$Headers:
                         Standard DPH Declarations
                         1-----
                ;
                1
                         Equates for disk parameter block
                :
                ş
                :
                         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.
                1
                         Disk types
                                                 vvvv--- Physical disk number
0010 =
                Floppy$5
                                          0$001$0000B
                                 EQU
                                                          ;5 1/4" mini floppy
0020 =
                Floppy$8
                                  EQU
                                          0$010$0000B
                                                           $8" floppy (SS SD)
0030 =
                M$Disk
                                  EQU
                                          0$011$0000B
                                                           Memory disk
0040 =
                H$Disk$10
                                  EQU
                                          0$100$0000B
                                                            ;Hard disk - 10 megabyte
0070 =
                .
Disk$Type$$Mask
                                          EQU
                                                   0$111$0000B
                                                                    ;Masks to isolate values
000F =
                Physical$Disk$Mask
                                          EQU
                                                   0$000$1111B
                         Blocking/deblocking indicator
0080 =
                Need$Deblocking EQU
                                          1$000$0000B
                                                           #Sector size > 128 bytes
                :
                .
                         Disk parameter blocks
                .
                .
                         ; Standard DPB's for A: and B:
                ş
                                                   ;Logical disk C:
                                                   FExtra byte indicates disk type
                                                   ; deblocking requirements and physical
                                                   ; disk drive.
                                 H$Disk$10 + Need$Deblocking + 0 ; Physical drive 0
000F C0
                         DB
                Hard$5$Parameter$Block$C:
                         2
                         Standard format parameter block
                         ; ---
                ;
                ;
0010 CO
                         DB
                                 H$Disk$10 + Need$Deblocking + 0 ; Physical drive 0
                Hard$5$Parameter$Block$D:
                         ٠
                         ;Standard format parameter block
0004 =
                Number$of$Logical$Disks
                                                  FOU
                                                           4
                1
                SELDSK:
                                          Select disk in register C
                                          ;C = 0 for drive A, 1 for B, etc.
                                          ;Return the address of the appropriate
; disk parameter header in HL, or 0000H
                                          ; if the selected disk does not exist.
0011 210000 0014 79
                                                  ;Assume an error
;Check if requested disk valid
                        LXI
                                 н, о
                        MOV
                                 A.C
0015 FE04
                        CPI
                                 Number$of$Logical$Disks
0017 D0
                        RNC
                                                  ;Return if > maximum number of disks
```



0018 320800	STA	Selected\$Disk	Save selected disk number
001B 6F	MOV	L,A	;Set up to return DPH address ;Make disk into word value
0010 2600	MVI	Н, О	ANAKE DISK INTO WOLD VALUE
			;Compute offset down disk parameter
			; header table by multiplying by
001F 00	<b>D</b> +D		; parameter header length (16 bytes)
001E 29 001F 29	DAD DAD	H H	;*2 ;*4
0020 29	DAD	н	:*8
0021 29	DAD	H	;*16
0022 110F00	LXI	D,Disk\$Paramete	
0025 19	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
0027 110A00	LXI	D, 10	; ;Get DPB pointer offset in DPH
002A 19	DAD	D	;DE -> DPB address in DPH
002B 5E	MOV	Ē,M	;Get DPB address in DE
0020 23	INX	H .	
002D 56 002E EB	MOV XCHG	D,M	
VVZE EB	XUHG		;DE -> DPB
	SELDSK\$Set\$Dis	k\$Type:	
002F 2B	DCX	н	;DE -> prefix byte
0030 7E	MOV	A, M	;Get prefix byte
0031 E670	ANI	Disk\$Type\$Mask	;Isolate disk type
0033 320900 0036 7E	STA MOV	Disk\$Type A,M	;Save for use in low-level driver ;Get another copy of prefix byte
0037 E680	ANI	Need\$Deblocking	
0039 320A00	STA	Deblocking\$Requ	
			Additional code to check if spare
			; directory for given disk has already
			; been read in.
			••••••••••••••••••••••••••••••••••••••
003C 7E 003D E60F	MOV ANI	A,M Physical\$Disk\$№	;Get physical disk number
003F 320B00	STA		al\$Disk ;Save for low-level drivers
0042 SF	MOV	E,A	;Make into word
0043 1600	MVI	D,0	Manager - Mater
0045 210400 0048 19	L X I DAD	H,Spare\$Dir\$In\$	Memory ;Make pointer into table
		-	
0049 7E	MOV	Α, Μ	;Get flag
004A B7	ORA	A	Conversion allocated in the second
004B C27700 004E 34	JNZ	Dir\$In\$Memory M	;Spare directory already in memory ;Set flag
00 /E 04	1 1417		1 mar 1 7 2 2
004F 210000	LXI	H,Spare\$Directo	
0052 19	DAD	D	; spare directory (added twice
0053 19	DAD	D	; as table has word entries)
			;HL -> word containing directory addr.
0054 5E	MOV	Е,М	
0055 23 0056 56	INX MOV	н D.M	;Spare directory address in DE
0056 56 0057 EB	XCHG	111	;Spare directory address in DE ;HL -> spare directory
0058 220600	SHLD	Selected\$Spare\$	Directory ;Save for use in physical
			; drivers later on
	LXI	D,Spare\$Track	Track containing spare directory
005B 110000	LDA	Selected\$Physic	
005E 3A0B00		<b>D A</b>	
005E 3A0B00 0061 47	MOV	B,A	
005E 3A0B00 0061 47 0062 3E04	MVI	A, Spare\$Sector	Sector containing spare directory
005B 110000 005E 3A0B00 0061 47 0062 3E04 0064 0E18 0064 0ED500			

Figure 9-4. (Continued)

0069 2A0600 006C 11C000 006F 19	LHLD LXI DAD	Selected\$Spare\$Direc D,Spare\$Length D	tory ;Set end marker ; at back end of spare directory
0070 110080		D. D.Maximum\$Track	;Use maximum track number
0073 73	MOV	M,E	Jose maximum track number
0073 73 0074 23	INX	H	
0075 3602	MVI	M, D	
	Dir\$In\$Memory:		
0077 E1	POP	H ;Rec	over DPH pointer
0078 C9	RET		
	;		
		low-level disk driver	s, the following code must be
	; insert	ed just before the dis	k controller is activated to
		e a read or a write co	ommand,
0079 240000	, LHLD	Disk\$Track	;Get track number from disk
007C EB	XCHG		; controller command table ;DE = track
007D 2A0600	LHLD	Selected\$Spare\$Direc	tory ;HL -> spare directory
0080 2B	DCX	H	Back up one entry
0081 2B	DCX	н	; (3 bytes)
0082 2B	DCX	Ĥ	
0083 3A0E00	LDA	Disk\$Sector	;Get sector number
0086 4F	MOV	C,A	;Save for later
0087 06FF	MVI	B, OFFH	;Set counter (biased -1)
	Check\$Next\$Ent		
0089 23	INX	н	;Update to next (or first) entry
	Check\$Next\$Ent		
008A 23	INX	н	
008B 23	Check\$Next\$Ent INX	г <u>ү</u> 2 ; Н	
008C 04	INR	в	;Update count
	İF	Sector\$Sparing	
		ouccos vopal ing	;If sparing is used, the
			<pre>; end of the table is indicated ; by an entry with the track number</pre>
			; = to maximum track number
	LXI	D, Maximum\$Track	Get maximum track number
	CALL JZ	CMPM Not <b>\$Bad\$</b> Sector	;Compare DE to (HL), (HL+1) ;End of table reached
	ENDIF	HOT POLOPOEC (OF	fend of table reached
			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
			<pre>; maximum track number is encountered</pre>
			; at the end of the table.
OOBD EB	XCHG		;DE -> table entry
008E 2A0C00	LHLD	Disk\$Track	;Get requested track
0091 EB 0092 CDCD00	XCHG	CHEM	<pre>#DE = req. track, HL -&gt; table entry</pre>
0072 CDCD00	CALL	CMPM	;Compare req. track to table entry
	IF	Sector#Sparing	;Use the following code for
	16.1 T		; sector sparing
	JNZ INX	Check\$Next\$Entry H	Firack does not match
	INX	H	;HL -> MS byte of track ;HL -> sector
	MOV	A,C	;Get requested sector
	CMP	M	;Compare to table entry
	JNZ	Check\$Next\$Entry2	Sector does not match
			<b>-</b>
			;Track and sector match, so
			; Frack and sector match, so ; substitute spare track and ; appropriate sector



		L	XI	H, Spare\$Track	<pre>;Get track number used for spare ; sectors</pre>
		S	HLD	Disk\$Track	;Substitute track
			VI DD	A,First\$Spare\$Sector B	;Get first_sector number ;Add on matched directory ; entry number
			TA NDIF	Disk\$Sector	;Substitute sector
		I	F	Sector\$Skipping	;Use the following code for ; sector skipping ;The object is to find the ; entry in the table which ; is greater or equal to the ; requested sector/track
0098	CA9E00 D2AC00 C38900	J	z NC MP	Tracks\$Match Compute\$Increment Check\$Next\$Entry	:Possible match of track and sector ;Requested track ;Requested track > table entry
		Tracks\$Ma	tch:		
009E 009F	23		NX NX	H H	;HL -> MS byte of track ;HL -> sector
0040	77		ον	M,A	Get sector from table
00A5	B9 CAAB00 D2AC00 C38B00	ں س	MP Z NC MP	C Sectors\$Match Compute\$Increment Check\$Next\$Entry2	;Compare with requested sector ;Track/sector matches ;Req. trk/sec < spare trk/sec ;Move to next table entry
		Sectors#M	atchs		
OOAB	04		NR	B	;If track and sectors match with ; a table entry, then an additional ; sector must be skipped
		Compute\$1	ncreme	nt:	
					B contains number of cumulative; number of sectors to skip;
OOAC			ov	A,C	;Get requested sector
00AD 00AE	80 0612		DD VI	B B,Sectors\$Per\$Track	;Skip required number ;Determine final sector number ; and track increment
	CDC300 320E00		ALL TA	DIV\$A\$BY\$B Disk\$Sector	; and track increment ;Returns C = quotient, A = remainder ;A = new sector number
00B6	59	м	ον	E,C	;Make track increment a word
	1600		VI	D, O	<b>.</b>
00B9	2A0C00		HLD AD	Disk\$Track D	;Get requested track ;Add on increment
	220000	S	HLD NDIF	Disk\$Track	;Save updated track
		Not\$Bad\$S	ector:		
	C3D500		MP	Read\$Write\$Disk	;Either track/sector were not bad, ; or requested track and sector have ; been updated. ;Go to physical disk read/write
0000	030300	;			,00 to physical disk redu/write
		,	F	Sector\$Skipping	;Subroutine required for skipping ; routine
		÷			
		; D	IV\$A\$E ivide	8Y\$B A by B	
		; т		outine divides A by B, r remainder in A.	eturning the quotient in C
			ntry p	arameters	
		;		A = dividend B = divisor	
		; ; E	vit na	Irameters	

Figure 9-4. (Continued)

```
;
                                        A = remainder
                   ;
                                        C = quotient
                   DIV$A$BY$B:
                                       'c,o
                                                            :Initialize quotient
00C3 0E00
                             MVI
                   DIV$A$BY$B$Loop:
                                        С
                                                            ;Increment quotient
                              INR
00C5 0C
                                        Ē
                                                            Subtract divisor
00C6 90
                              SUB
00C7 F2C500
                              JP
                                        DIV$A$BY$B$Loop ;Repeat if result still +ve
OOCA OD
                              DCR
                                        С
                                                            ;Correct quotient
00CB 80
00CC C9
                              ADD
                                        в
                                                            ;Correct remainder
                              RET
                              ENDIF
                   .
                              CMPM
                   ;
                              Compare memory
                   ş
                   ;
                             This subroutine compares the contents of DE to (HL) and (HL+1) returning with the flags as though the subtraction (HL) – DE
                   $
                   3
                              were performed.
                             Entry parameters
                                        HL -> word in memory
                   3
                                        DE = value to be compared
                    3
                   .
                              Exit parameters
                   ;
                   ;
                                        Flags set for (HL) - DE
                   .
                   CMPM:
                                                                       ;Get MS byte
OOCD 7E
                              MOV
                                        A.M
00CE BA
00CF C0
00D0 23
00D1 7E
                              CMP
                                        n
                              RNZ
                                                                       ;Return now if MS bytes unequal
                                                                       ;HL -> LS byte
;Get LS byte
                              INX
                                        н
                              MOV
                                        A, M
OOD2 BB
                              CMP
                                        Е
00D3 2B
                              DCX
                                        н
                                                                       ;Return with HL unchanged
00D4 C9
                              RET
                   2
                   Absolute$Read:
                                                  ;The absolute read (and write) routines
                                                  ; access the specified sector and track
                                                  ; without using bad sector management.
                              Entry parameters
                   ;
                   ;
                                        HL -> Buffer
DE = Track
                                        A = Sector
                                        B = Physical disk drive number
                                        C = Number of bytes to read / 8
                              Set up disk controller command block with parameters in
                              registers, then initiate read operation by falling through into Read$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
                   ;
                    2
                             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
                    1
                    1
                    1
                              searching of the directory).
```

Figure 9-4. (Continued)

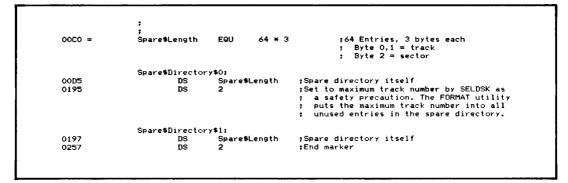


Figure 9-4. Bad sector management (continued)

## **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.

	<ul> <li>for disk error</li> <li>in Figure 8-</li> <li>This version</li> <li>depending on</li> <li>Code that ren</li> <li>abbreviated.</li> </ul>	ors than f 10. outputs a the natur mains unch	that show a recomme re of the nanged fi	-friendly error processor wn in the enhanced BIOS ended course of action e error detected. rom Figure 8-10 has been ations needed to get
	; an error free			
0001 = 0002 =	; Floppy\$Read\$Code Floppy\$Write\$Code	EQU EQU	01H 02H	;Read command for controller ;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)
0043 =	Disk\$Status\$Block	EQU	43H	;Address in memory where controller ; returns status ;Values from controller command table
0001 00	Floppy\$Command:	DB	0	
0002 00	Floppy\$Head:	DB	0	
0003 00	Floppy\$Track:	DB	0	
0004 00	Floppy\$Sector:	DB	0	

Figure 9-5. User-friendly disk-error processor

```
0005 00.
                Deblocking$Required:
                                          DB
                                                   0
                                                            Flag set by SELDSK according
                                                            to selected disk type
0006 00
                DiskSErrorSFlag:
                                          DB
                                                   ۵
                                                            Error flag returned to BDOS
0007 00
                ,
In$Buffer$Disk:
                                          DB
                                                            :Logical disk Id. relating to current
                                                   0
                                                            ; disk sector in deblocking buffer
                :
                         Equates for Messages
                :
                .
0007 =
                BELL
                         EQU
                                  07H
                                          $Sound terminal bell
000D =
                CR
                         FOU
                                  ODH
                                          Carriage return
000A =
                         EQU
                                  OAH
                $
0005 #
                         EQU
                                  5
                                          :BDOS entry point (for system reset)
                'anns
                No$Deblock$Retry:
                         ; Omitted code to set up disk controller command table
                         ; and initiate the disk operation
0008 C31500
                         IMP
                                  Wait$For$Disk$Complete
                Write$Physical:
                                                   ;Write contents of disk buffer to
                                                     correct sector
ie ;Get write function code
000B 3E02
                         MUT
                                  A,Floppy$Write$Code
                                  CommonSPhysical :00 to common code
000D C31200
                         JMP
                Read$Physical:
                                                   Read previously selected sector
                                                   ; into disk buffer
0010 3E01
                         HUT
                                  A, Floppy $Read$Code
                                                           ;Get read function code
                Common $Physical:
0012 320100
                         STA
                                  Floppy$Command ;Set command table
                Deblock$Retry:
                                                   :Rementry point to retry after error
                         9 Omitted code sets up disk controller command block
9 and initiates the disk operation
                         .
                                                                    Wait$For$Disk$Complete:
                                                   ;Wait until disk status block indicates
                                                   ; operation has completed, then check
; if any errors occurred
                                                   ;On entry HL -> disk control byte
                         XRA
0015 AF
                                                            Ensure hung flag clear
0016 320000
                                  Disk$Hung$Flag
                         STA
0019 213100
0010 015802
                         LXI
                                                            ;Set up watchdog timer
                                  H,Disk#Timed#Out
                                  B, Disk#Timer
                         LXI
                                                            Time delay
001F CD3803
                         CALL
                                  Set#Watchdog
                Disk$Wait$Loop:
0022 7E
0023 B7
                         MOV
                                  A.M
                                                            ;Get control byte
                         ORA
0024 CA3700
                         J7
                                  Disk$Complete
                                                            ;Operation done
0027 3A0000
                         LDA
                                  Disk$Hung$Flag
                                                            Also check if timed out
002A B7
                         ORA
002B C29F02
                         JNZ
                                  Disk$Error
                                                            Will be set to 40H
002E C32200
                         JMP
                                  Disk$Wait$Loop
                Disk$Timed$Out:
                                                   #Control arrives here from watchdog
                                                   ; routine itself -- so this is effectively
                                                      part of the interrupt service routine
;Set disk hung error code
                                                   ;
0031 3E40
0033 320000
                         MVI
                                  A, 40H
                         STA
                                  Disk$Hung$Flag
                                                            ; into error flag to pull
                                                               control out of loop
0036 C9
                         RET
                                                            Return to watchdog routine
```

	Disk\$Complete:	~ ^	-Dessi unishdag timor
010000	L×1	B,0	;Reset watchdog timer ;HL is irrelevant here
CD3B03	CALL	Set\$Watchdog	• • • • • • • • • • • • • • • • • • • •
344300	LDA	Disk\$Status\$Blo	ck ;Complete now check status
FE80	CPI	BOH	;Check if any errors occurred
DA9F02	JC	Disk\$Error	;Yes
	; Disk\$Error\$Ignd	ore:	
AF	XRA	Α	;No
320600	STA	Disk\$Error\$Flag	;Clear error flag
69	KC I		
	7		
		ror message hand	ling
	1		
	Disk\$Error\$Mess	ages:	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
			<pre>; error description message,</pre>
			-1.44
			sk\$Msg\$40
	DW		sksMcas4 <u>1</u>
42	DB	42H	2.4.1.2.2.4.7
E301A400	DW		sk\$Msg\$42
21		21H Disktonius and Di	-1.444474
07028400			SK≱M2G≱<1
18028900	DW		sk\$Msq\$22
23	DB	23H	
1B02C000	DW		sk\$Msg\$23
			-LeMtOA
		25H	2K4U2G444
3D02DE00	DW ·		sk\$Msg\$25
11	DB	11H	
			sk\$Msg\$11
			= 2 ± M= - = ± 1 0
	DB	13H	2241129415
53020001	DW	Disk\$Advice7,Di	sk\$Msg\$13
	DB		-1.4.4
			sk\$Msg\$I4
53022901	DB		sk\$Msg\$15
16	DB	16H	
53023501	DW		
			;<== Terminator sk\$Msg\$Unknown ;Unmatched code
33024001	,		
=	DEMSEntry\$Size	EQU 5	;Entry size in error message table
	5 7		
		e texts	
4975454700	) DieksMens40:	ne 'Hune'.	0 ;Timeout message
46/3620/5/	2Disk\$Msg\$41:		ady',0
			Protected', 0
			,^,O 19 Data Mark',O
4067/3/363	ADiskomsgoza: ADiskomsgoza:		meout',0
			oller Timeout',0
4472697665	5Disk\$Msg\$11:	DB 'Drive	Address',0
4865616420	)Disk\$Msg\$12:	DB 'Head A	Nddress',0
	40 40 40 80019500 47 320600 C9 47 320600 C9 40 80019500 41 C9019500 42 801400 22 1802000 23 1802000 24 1802000 25 30020200 10 53022F00 11 53022F00 12 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 13 53022F00 15 53022F00 13 53022F00 13 53022F00 13 53022F00 14 53022501 16 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 53022F00 15 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 5772697455 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 577267 5	010000 LXI CD3B03 CALL 3A4300 LDA FE80 CPI DAFF02 JC	010000 LXI B,0 CD3B03 CALL Set\$Watchdog 3A4300 LDA Disk\$Status\$B10 FE80 CPI 80H DA9F02 JC Disk\$Error Disk\$Error\$Ignore: AF XRA A 320600 STA Disk\$Error\$Flag C9 RET j Disk\$Error\$Messages: 40 DB 40H B0019500 DW Disk\$Advice1, Di j Disk\$Error\$Messages: 40 DB 40H B0019500 DW Disk\$Advice2, Di 41 DB 41H C9019A00 DW Disk\$Advice3, Di 21 DB 42H E301A400 DW Disk\$Advice3, Di 21 DB 21H 0702B400 DW Disk\$Advice3, Di 22 DB 22H 1802B900 DW Disk\$Advice5, Di 23 DB 23H 1802B900 DW Disk\$Advice6, Di 24 DB 24H 3002D200 DW Disk\$Advice6, Di 25 DB 25H 100 DB 25H 1002D200 DW Disk\$Advice6, Di 25 DB 25H 100 DB 25H 100 DB 10H 11 DB 11H 5302Ff00 DW Disk\$Advice7, Di 12 DB 12H 5302FF00 DW Disk\$Advice7, Di 13 DB 13H 53022001 DW Disk\$Advice7, Di 14 DB 14H 53021A01 DW Disk\$Advice7, Di 15 DB 25H 53022501 DW Disk\$Advice7, Di 14 DB 14H 53022101 DW Disk\$Advice7, Di 15 DB 15H 53022501 DW Disk\$Advice7, Di 14 DB 14H 53022100 DW Disk\$Advice7, Di 15 DB 15H 53022501 DW Disk\$Advice7, Di 16 DB 16H 53022501 DW Disk\$Advice7, Di 17 DB 15H 53022501 DW Disk\$Advice7, Di 18 OF 53024501 DW Disk\$Advice7, Di 19 DS 19B 0 10 DS 19B 0 11 DS 19B 0 13 DB 13H 53022501 DW Disk\$Advice7, Di 14 DB 14H 5302100 DW Disk\$Advice7, Di 15 DB 25H 53022501 DW Disk\$Advice7, Di 16 DB 16H 53022501 DW Disk\$Advice7, Di 17 Message texts 18 Format 405772607305Nisk\$Msg\$42: DB 'Hung', 4267742052Disk\$Msg\$42: DB 'Hung', 42757305Disk\$Msg\$42: DB 'Contro

Figure 9-5. (Continued)

011A 536563746FDisk\$Msg\$14: DB 'Sector Address'.0 0129 4275732041Disk\$Msg\$15: 0135 496C6C6567Disk\$Msg\$16: DB 'Bus Address',0 DB 'Illegal Command',0 0145 556E6B6E6FDisk\$Msg\$Unknown: ΠŘ 'Unknown',0 Disk#EM#1: #Main disk error message -- part 1 0140 07000A **DB** BELL, CR, LF 0150 4469736820 **D**B Disk 🖓 🛈 ;Error text output next Disk\$EM\$2: Main disk error message -- part 2 Error ( 0156 204572726F DB 015E 0000 Disk#EM#Status: 0,0 ;Status code in hex DB DA, CR, LF, 0160 290D0A2020 **n**R Drive ' Disk drive code, A,B... , Head ' 016E 00 1 016F 2C20486561 0 Disk\$EM\$Drive: DB DR 0176 00 1 0177 2C20547261 . Disk\$EM\$Head: DB o ;Head number Track DB 017F 0000 Disk\$EM\$Track: DB 0,0 ;Track number 0181 2020536563 DB ', Sector ;Sector number 018A 0000 Disk\$EM\$Sector: DB 0,0 018C 2C204F7065 DB · Operation -019A 00 DB 0 **#Terminator** 019B 526561642EDisk#EM#Read: DB 'Read, ', O süperation names 01A1 5772697465DisksEMsWrite: DB Write, ',0 01A8 0D0A202020Disk\$Advice0: DB CR. LE. 4 4,0 01B0 4368656368Disk#Advice1: DB 'Check disk loaded, Retry',0 01C9 506F737369Disk\$Advice2: DВ 'Possible hardware problem',0 01E3 5772697465Disk#Advice3: ΠB 'Write enable if correct disk, Retry',0 0207 5265747279Disk#Advice41 DB 'Retry several times',0 "Reformat disk or use another disk",0 "Hardware error, Retry",0 "Hardware or Software error, Retry",0 021B 5265666F72Disk\$Advice5: DB 023D 4861726477Disk\$Advice6: DB 0253 4861726477Disk\$Advice7: DB 0275 2C206F7220Disk\$Advice9: DB ', or call for help if error persists', CR, LF Disk\$Action\$Confirm: 029B 00 029C 0D0A00 **DR** ;Set to character entered by user n CR.LF.O DB Disk error processor This routine builds and outputs an error message. The user is then given the opportunity to: R -- retry the operation that caused the error A -- 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 02A0 215E01 H,Disk\$EM\$Status LXI ;Convert code for message 02A3 CD3B03 CALL CAH :Converts A to hex In\$Buffer\$Disk ;Convert disk id. for message 0246 340700 IDA 02A9 C641 02AB 326E01 ;Make into letter ADI STA Disk\$EM\$Drive 02AE 3A0200 02B1 C630 LDA Floppy\$Head ;Convert head number 101 ADI 0283 327601 STA Disk\$EM\$Head 0286 3A0300 LDA Floppy\$Track ;Convert track number 02B9 217F01 H, Disk#EM#Track LXI 02BC CD3B03 CAH CALL 02BF 3A0400 02C2 218A01 LDA Floppy\$Sector ;Convert sector number LXI H.Disk\$EM\$Sector 02C5 CD3B03 CALL CAH 02C8 214D01 LXI H,Disk\$EM\$1 ;Output first part of message 02CB CD3B03 CALL Output#Error#Message

Figure 9-5. (Continued)

02CE F1 POP PSW Recover error status code 02CF 47 MOV B, A :For comparisons 02D0 214500 IXI H,Disk\$Error\$Messages ~ DEM\$Entry\$Size ;HL -> table -- one entry 0203 110500 1 8 7 D.DEM\$Entry\$Size For loop below Disk\$Error\$Next \$Code: 02D6 19 DAD n #Move to next (or first) entry 02D7 7E MOV A. M ;Get code number from table ;Check if end of table 02D8 B7 ORA 02D9 CAE302 JZ Disk\$Error\$Matched ;Yes, pretend a match occurred 02DC 88 CMP :Compare to actual code R 02DD CAE302 JZ Disk#Error#Matched Yes, exit from loop Check next code 02E0 C3D602 JMP Disk\$Error\$Next\$Code . Disk\$Error\$Matched: 02E3 23 INX н ;HL -> advisory text address 02E4 5E MOV E.M 02E5 23 TNX н 02E6 56 MOV :DE -> advisory test D.M 02E7 D5 PUSH n 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 02EC EB XCHG. ;HL -> text O2ED CD3B03 Output\$Error\$Message ;Display explanatory text CALL 02F0 215601 LXI H, Disk\$EM\$2 ;Display second part of message 02F3 CD3B03 CALL Output#Error#Message 02F6 219B01 LXI H,Disk\$EM\$Read ;Choose operation text (assume a read) ;Get controller command 02F9 3A0100 02FC FE01 Floppy\$Command Floppy\$Read\$Code CP L 02FE CA0403 .17 Disk\$Error\$Read :Yes H,Disk\$EM\$Write ;No, change address in HL LXI 0301 21A101 Disk\$Error\$Read: 0304 CD3B03 CALL Output#Error#Message ;Display operation type 0307 21A801 LXI ;Display leading blanks H. Disk\$AdviceO Output#Error#Message 030A CD3B03 CALL 030D E1 POP Recover advisory text pointer 030E CD3B03 CALL Output\$Error\$Message 0311 217502 LXI H,Disk\$Advice9 Display trailing component 0314 CD3B03 CALL Output#Error#Message Disk\$Error\$Request\$Action: ;Ask the user what to do next Display prompt and get single 0317 CD3B03 CALL Request\$User\$Choice character response (folded to . uppercase) 031A FE52 CPI **1**R1 Retry 031C CA2C03 031F FE41 Disk*Error*Retry JΖ CPT 'A' : Abort? 0321 CA3603 System\$Reset . 17 0324 FE49 CPI ۲ Y : Ignore? 0326 CA4500 JZ Disk\$Error\$Ignore 0329 C31703 JMP Disk\$Error\$Request\$Action The decision on where to return to depends on whether the operation Disk\$Error\$Retry: failed on a deblocked or nondeblocked drive 2 032C 3A0500 LDA Deblocking\$Required 032F B7 ORA 0330 C21500 JNZ Deblock\$Retry 0333 C30800 JMP No\$Deblock\$Retry

Figure 9-5. (Continued)

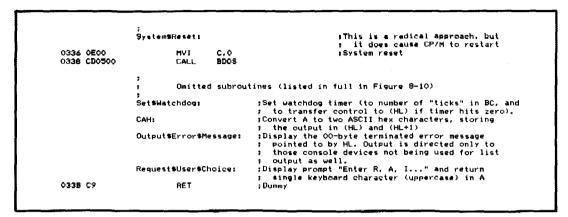


Figure 9-5. User-friendly disk-error processor (continued)

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
                           Next instruction
```

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,

```
0210
           CALL
                      Debug$Routine
0213
           DR
                      OADH, 'Caption string',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

Value displayed
bit registers
Condition Flags
Register A
Register B
Register C
Register D
Register E
Register H
Register L
Memory
Bytes starting and ending at the addresses specified by the two word values following the code value.
bit registers
Register pair BC
Register pair DE
Register pair HL
Stack Pointer
Byte values
Byte addressed by BC
Byte addressed by DE
Byte addressed by HL
Ford values
Word addressed by BC
Word addressed by DE
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:

Debug
5
NOT Debug
OUT
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									
00001									
00003		,							
00004		;	Debug (	Subroutines					
00005		*							
00006		;<	NOTE:						
00007		;			me left are included purely				
00008		1		erence the code from th are deliberately induce					
00010		3			d discontinuities low space for expansion.				
00011		i i	411 111	traineers are wroter to at	THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PERSONNELSES OF THE PE				
00012		i	Becaus	e of the need to test t	hese routines thoroughly,				
00013		i	and in	case you wish to make	any changes, the testbed				
00014		:			itself has been left in				
00015		!	in thi	s figure.					
00016		1	Dates	tarthad					
00017 00018		;	neodâ	testbed					
00018	0100	,	ORG	100H					
00020		START:	<b>#</b>						
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				
00024	0109 3EAA		MVI	A 044H	Simple test of A register display; Preset a value in the A register				
00025	0109 3EAA 010B 01CCBB			A, OAAH B, OBBCCH	Prefill all other registers, partly				
00028	OTOB OTCEBB			D. ODDEEH	: to check the debug display, but				
00028	0111 2111FF			H. OFF11H	; also to check register save/restore				
00100		ş #		-					
00101		;	Test r	egister display					
00102	A ==		A- /		- Pul M #1				
00103	0114 B7		ORA	A	ySet M-flag, clear Z-flag, set E-flag				
00104	0115 37 0116 CD5205		STC	DBéDisplay	sSet carry sCall the debug routine				
00105	0116 CD5205 0119 00		DB	DB\$D15play DB\$F	year the depug rodtine				
00108	0114 466C61677	3	DB	'Flags',0					
00108		3	55	-					
00109	0120 CB5205		CALL	DB\$Display	;Call the debug routine				
00110	0123 02	-	DB						
00111	0124 4120526563		DB	'A Register',O					
00112	0125 005005	;	CAL 1	DRADicolas	*Call the debug routing				
00113 00114	012F CD5205 0132 04		CALL DB	DB\$Display DB\$B	;Call the debug routine				
00115	0133 422052656	7	DB	′B Register′,0					
00116		;							
00117	013E CD5205		CALL	DB\$Display	;Call the debug routine				
00118	0141 06	_	DB	DB\$C					
00119	0142 432052656	-	DB	'C Register',0					
00120	0145 65565-	,	CALL	<b>DB&amp;</b> Bienlay	;Call the debug routine				
00121	014D CD5205 0150 08		CALL DB	DB\$Bisplay DB\$D	rears the omoug rudting				
00122	0150 08	7	DB	'D Register',0					
00123		<b>`</b> ,	~~						
00125	015C CD5205		CALL	DB\$Display	;Call the debug routine				
00126	015F 0A		DB	DB\$E					
00127	0160 4520526563		DB	'E Register',0					
00128	0140 00004-	;	<b></b> .		and the definition of the				
00129 00130	016B CD5205 016E OC		CALL DB	DB\$Display DB\$H	;Call the debug routine				
00130	016E 0C 016F 4820526563	7	DB	UB\$H 'H Register',0					
00132		, ,							
00133	017A CD5205	-	CALL	DB\$Display	;Call the debug routine				
00134	017D OE		DB	DB\$L					
00135	017E 4C20526563	7	DB	'L Register',0					
00200		;*	<b>T</b> *	amony Dime Direct					
00201		!	rest M	emory Dump Display					
00202	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 #11,0	; of 10H				
00207	0140 0	1		DBAD4					
00208	01A0 CD5205		CALL	DB\$Display DB\$M					
00209	01A3 18 01A4 00011F01		db Dw	DB\$M 100H, 11FH	;Dump memory ;Check start and end on displayed				
00210	01A8 4D656D6F72	2	DB	'Memory Dump #2',0	; line boundaries				
00212	100000F//	2 ;			,				
		·							

Figure 10-2. Debug subroutines

00213	0187 CD5205	CALL	DB\$Display	
00214	01BA 18	DB	DB\$M	;Dump memory
00215	01BB 01010001	DW	101H, 100H	Check error handling where
00216	01BF 4D656D6F72	DB	'Memory Dump #3',0	; start > end address
00217	····· <b>·</b> ··· <b>·</b>		includer y and the year	y start / enu address
00218	01CE CD5205	CALL	DB\$Display	
00219	01D1 18	DB	DB\$M	1 Dump memory
00220	01D2 00010001	DW	100H, 100H	Check end-case of single byte
00221	01D6 4D656D6F72	DB	'Memory Dump #4',0	i output
00300	;#			
00301	,	Test r	egister pair display	
00302	7			
00303	01E5 CD5205	CALL	DB\$Display	Call the debug routine
00304	01E8 10	DB	DB\$BC	
00305	01E9 4243205265	DB	'BC Register',0	
00306	;			
00307	01F5 CD5205	CALL	DB\$Display	Call the debug routine
00308	01F8 12	DB	DB\$DE	,
00309	01F9 4445205265	DB	'DE Register',0	
00310	1			
00311	0205 CD5205	CALL	DB\$Display	Call the debug routine
00312	0208 14	DB	DB\$HL	
00313	0209 4840205265	DB	'HL Register',0	
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	1#			
00401	3	Test b	yte indirect display	
00402	,			
00403	022E CD5205	CALL	DB\$Display	;Call the debug routine
00404	0231 1A-	DB	DB\$B\$BC	,
00405	0232 4279746520	DB	'Byte at (BC)',0	
00406				
00407	023F CD5205	CALL	DB\$Display	;Call the debug routine
00408	0242 10	DB	DB\$B\$DE	,call the bebug routine
00409	0243 4279746520	DB	'Byte at (DE)',O	
00410	;			
00411	0250 CD5205	CALL	DB#Display	;Call the debug routine
00412	0253 1E	DB	DB\$B\$HL	
00413	0254 4279746520	DB	'Byte at (HL)',0	
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)',O	
00426	· • •			
00427	028C CB5205	CALL	DB\$Display	Call the debug routine
00428	028F 24	DB	DB\$W\$HL	
00429	0290 576F726420	DB	'Word at (HL)',0	
00500	;#			1
00501	;	Test D	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	ODH,OAH,'This message	should NOT appear ,0
00506				
00507	02C4 CD1505	CALL	DB\$On	
00508	02C7 CDD607	CALL	DB\$MSG I	
00509	02CA 0D0A446562	DB	ODH, OAH, 'Debug output	has been re-enabled.',0
00600	;#			
00601	;	Test p	ass count logic	
00602	;			

Figure 10-2. (Continued)

00603	0355	CD1D05		CALL	DBSOff		
00603		CD1005 CD2405		CALL	DB\$UTT DB\$Set\$F		Disable debug output
				DW	30	-455	;Set pass count
00605	021-4	1E00		DW	30		
00606			\$				
00607	02F6	3E22		MVI	A, 34		Set loop counter greater than pass
00608							; counter
00609			Test#Pa:	ss#Loop:			
00610		CD3505		CALL	DB#Pass		Decrement pass count
00611	02FB	CDD607		CALL	DB\$MSGI		pDisplay in-line message
00612		OD0A546865	,	DB		. This message st	nould display 5 times',0
00613	0324	30		DCR	A		
00614		C2F802		JNZ	Test#Pa	ss\$Loop	
00700			;#				
00701			;	Test det	oug input		
00702			-	iest uer	ada ruba.	c/output	
00703	0000	CD1D05	;	CALL	DB\$Off		Charle that debug INCOUT
	0320	CDIDUS		CALL	DB#011		;Check that debug IN/OUT
00704							; must still occur when debug
00705					_		; output is disabled.
00706	032B			RST	4		;Debug input
00707	0320			DB	11H		;Port number
00708	032D	EF		RST	5		;Debug output (value return from input)
00709	032E	22		DB	22H		Port number
00710							
00711	032F	C30000		JMP	0		Warm boot at end of testbed
00712			3				
00713			;				
00714			1	Bummy ve	lues for	· byte and word o	disòlavs
00715	0332	BC	Byte\$BC:		DB	OBCH	
00716	0333		Byte\$DE:		DB	ODEH	
00717	0334		Byte#HL:		DB	OF1H	
00718	0334	F 1	DYCEPHE		00	OF TH	
	0000	OCOB	,	_	DW	OROCIU	
00719			Word\$BC			OBOCH	
00720			Word\$DE		DW	ODOEH	
00721	0339	010F	Word\$HL:		DW	OF01H	
00722			1				
00723		99999999999			DW		эн, 9999н, 9999н, 9999н, 9999н, 9999н
00724		99999999999	>				
					DW		эн, 9999н, 9999н, 9999н, 9999н, 9999н
00725	035B	99999999999	<b>,</b>		DW		9H, 9999H, 9999H, 9999H, 9999H, 9999H 9H, 9999H, 9999H, 9999H, 9999H, 9999H
00725 00726	035B	99999999999		ack:			
	0358	99999999999	) Test\$St	ack:			
00726	0358	99999999999	) Test\$Sti 7	ack:			
00726 00727 00728	0358	99999999999	) Test\$St	ack:			
00726 00727 00728 00729		<del>99999999999</del> 9	7 Test\$St ; ;		DW		<del>рн, 9999н, 9999н, 9999н, 9999н, 9999</del> н, 9999н
00726 00727 00728 00729 00730	035B 0400	<b>99999</b> 999999	7 Test\$St ; ;	ack: ORG			эН, 9999Н, 9999Н, 9999Н, 9999Н, 9999Н
00726 00727 00728 00729 00730 00731		99999999999	) Test\$St ; ;		DW		<del>рн, 9999н, 9999н, 9999н, 9999н, 9999</del> н, 9999н
00726 00727 00728 00729 00730 00731 00732			7 Test\$St; ; ;		DW		эН, 9999Н, 9999Н, 9999Н, 9999Н, 9999Н
00726 00727 00728 00729 00730 00731 00732 00800			7 Test\$St; ; ; ; ;#		DW		эН, 9999Н, 9999Н, 9999Н, 9999Н, 9999Н
00726 00727 00728 00729 00730 00731 00732 00800 00801			) Test\$St; ; ; ; ; ; ;	ORG	DW 400H	9999H, 9999H, 9999	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings
00726 00727 00728 00729 00730 00731 00732 00800 00801 00802			) Test\$St ; ; ; ; ; ; ;	ORG	DW	9999H, 9999H, 9999	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings
00726 00727 00728 00729 00730 00731 00732 00800 00801 00802 00803			) Test\$St; ; ; ; ; ; ; ; ; ;	ORG	DW 400H	9999H, 9999H, 9999	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings
00726 00727 00728 00730 00731 00731 00732 00800 00801 00802 00803 00804			) Test\$St; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su	DW 400H ubroutine	9999H, 9999H, 9999	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings
00726 00727 00728 00730 00730 00731 00732 00800 00801 00802 00803 00805			) Test\$St; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates	DW 400H ubrouting for DB\$I	9999H,9999H,9999 	pH,9999H,9999H,9999H,9999H,9999H γTo avoid unnecessary listings γ when only the testbed changes
00726 00727 00728 00729 00730 00731 00732 00800 00801 00801 00803 00804 00805 00806			) Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These eq	400H ubroutine	9999H,9999H,9999 s Display codes e the offsets do	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00729 00730 00730 00732 00800 00800 00800 00803 00803 00804 00805 00806 00806			) Test\$St; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These eq	400H ubroutine	9999H,9999H,9999 	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00801 00800 00801 00802 00803 00804 00805 00806 00807 00806	0400		) Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These eq for vari	DW 400H ubroutine for DB\$I quates an ious subr	9999H,9999H,9999 s Display codes re the offsets do routines to be us	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00730 00730 00730 00801 00800 00801 00802 00803 00804 00805 00806 00807	0400	-	) Test\$St, ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These eq for var: EQU	400H ubroutine for DB\$I quates ar lous subr	9999H,9999H,9999 Isplay codes re the offsets de routines to be un	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00732 00800 00801 00802 00803 00804 00805 00806 00806 00806 00809 00809 00810	0400 0000 0002	-	) Test\$St; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Debug su Equates These ec for vari EQU EQU	400H 400H 40routine for DB\$I quates an ious subr 00 02	9999H,9999H,9999 The offsets do routines to be un Flags fA register	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00730 00730 00730 00801 00800 00801 00802 00803 00804 00805 00806 00807	0400	-	) Test\$St, ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These eq for var: EQU	400H ubroutine for DB\$I quates ar lous subr	9999H,9999H,9999 Isplay codes re the offsets de routines to be un	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00732 00800 00801 00802 00803 00804 00805 00806 00806 00806 00809 00809 00810	0400 0000 0002		) Test\$St; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Debug su Equates These ec for vari EQU EQU	400H 400H 40routine for DB\$I quates an ious subr 00 02	9999H,9999H,9999 Display codes the offsets do routines to be un ;Flags ;A register ;B	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00730 00801 00800 00801 00802 00803 00804 00805 00806 00805 00806 00809 00810 00811	0400 0000 0002 0004	-	) Test\$St, ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These ec for vari EQU EQU EQU	DW 400H ubrouting for DB\$1 quates ar lous subr 00 02 04	9999H,9999H,9999 Display codes re the offsets do routines to be un ;Flags ;A register ;B	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00729 00730 00730 00801 00800 00801 00803 00804 00805 00806 00805 00806 00807 00806 00807 00806 00807 00808 00807 00810 00811 00812 00813	0400 0000 0002 0004 0006 0006		) Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These et for var: EQU EQU EQU EQU EQU	DW 400H ubroutine for DBSI quates ar ious subr 00 02 04 06 08	9999H,9999H,9999 Tes the offsets do routines to be us sFlags sA register sB sC sD	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00730 00730 00730 00801 00802 00800 00803 00804 00805 00806 00805 00806 00807 00808 00809 00810 00811 00813 00814	0400 0000 0002 0004 0006 0008		7 Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates for var: EQU EQU EQU EQU EQU EQU EQU	DW 400H ubroutine for DBSI uates as lous subr 00 02 04 06 08 10	9999H,9999H,9999 Display codes re the offsets do routines to be un ;Flags ;A register ;B ;C ;D ;E	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00731 00732 00800 00801 00802 00803 00804 00805 00806 00806 00806 00807 00808 00809 00810 00811 00812 00813 00814 00815	0400 0000 0002 0004 0006 0008 000A 000C		) Test\$St, ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug st Equates These ec for vari EQU EQU EQU EQU EQU EQU EQU EQU	DW 400H 	9999H,9999H,9999 Display codes re the offsets do outines to be un ;Flags ;A register ;B ;C ;D ;E ;H	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00801 00800 00801 00802 00803 00804 00805 00806 00807 00806 00807 00806 00807 00808 00801 00811 00812 00813 00814 00815 00916	0400 0000 0002 0004 0006 0008 000A 000C		) Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These et for var: EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 400H broutine for DBSI ious subr 00 02 04 06 08 10 12 14	9999H,9999H,9999 Display codes re the offsets de routines to be un ;Flags ;A register ;B ;C ;D ;E ;H	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00731 00732 00800 00801 00802 00803 00804 00805 00806 00805 00806 00806 00806 00806 00806 00809 00810 00811 00812 00813 00814 00815 00816 00817	0400 0000 0002 0004 0006 0008 0000 0000 0000		<pre>&gt; Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;</pre>	ORG Debug su Equates These ec for vari EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 400H for DB\$1 juates ar jous subr 00 02 04 06 08 10 12 14 16	9999H,9999H,9999 ss Display codes re the offsets do ooutines to be un ;Flags ;A register ;B ;E ;H ;L ;BC	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00730 00730 00730 00801 00800 00803 00803 00804 00805 00804 00805 00806 00806 00806 00806 00806 00806 00806 00808 00808 00808 00808 00808 00808 00810 00811 00815 00815 00817 00818	0400 0002 0004 0006 0008 0000 0002 0008 0000 0008		Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Debug su Equates These ec for vari Equ Equ Equ Equ Equ Equ Equ Equ Equ Equ	2004 4004 ubroutine for DB\$1 yuates ar ious subr 00 02 04 06 06 06 06 08 10 12 14 16 18	9999H,9999H,9999 ss Display codes re the offsets do routines to be un ;Flags ;A register ;B ;C ;D ;E ;B ;E ;BC ;DE	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00730 00730 00730 00801 00802 00800 00802 00803 00804 00805 00806 00805 00806 00807 00808 00809 00810 00811 00813 00814 00815 00916 00819	0000 0002 0004 0006 0008 0006 0008 0006 0006 0006		Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates for var: EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 	9999H,9999H,9999 Display codes re the offsets do routines to be un ;Flags ;A register ;B ;C ;D ;E ;H ;E ;HL ;BC ;HL	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00730 00730 00730 00801 00800 00803 00804 00805 00806 00806 00806 00806 00806 00806 00806 00807 00808 00809 00810 00811 00812 00813 00814 00815 00815 00818 00819 00820	0400 0000 0002 0004 0006 0000 0000 0000		) Test\$St, ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These ed for vari Equ Equ Equ Equ Equ Equ Equ Equ Equ Equ	200 4000H 4000H for DB\$1 yuates ar jous subr 00 00 00 00 00 00 00 00 12 14 16 18 20 22	9999H,9999H,9999 ss Display codes re the offsets do ooutines to be un Flags 7A register 7B 7E 7E 7E 7B 7B 7B 7B 7B 7B 7B 7B 7B 7B	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00730 00730 00730 00801 00800 00801 00802 00803 00804 00805 00806 00807 00806 00807 00806 00810 00811 00812 00813 00814 00815 00815 00816 00816 00816 00816	0400 0000 0002 0004 0006 0008 0006 0002 0006 0002 0004 0006 0012 0014 0014		) Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Debug su Equates These et for var: EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 	9999H,9999H,9999 Display codes re the offsets do routines to be un ;Flags ;A register ;B ;C ;D ;L ;BC ;DE ;HL ;Stack pointer ;Memory	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00801 00801 00802 00803 00804 00805 00806 00805 00806 00806 00806 00807 00806 00807 00810 00811 00812 00813 00814 00815 00816 00817 00818 00819 00821 00821 00821 00821 00821	0400 0000 0002 0004 0006 0000 0000 0000		7 Test\$St. ; ; ; ;; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates for vari EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H for DB\$1 juates arrows autes arrows	9999H,9999H,9999 ss Display codes re the offsets do outines to be un ;Flags ;A register ;B ;E ;H ;E ;H ;E ;HL ;Stack pointer ;Menory ;(BC)	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00730 00730 00730 00801 00800 00803 00803 00804 00805 00804 00805 00806 00805 00806 00807 00808 00809 00810 00811 00812 00813 00814 00815 00815 00815 00815 00817 00818 00821 00820	0400 0000 0002 0004 0006 0000 0000 0000		Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These et for vari Equ Equ Equ Equ Equ Equ Equ Equ Equ Equ	DW 4000H 	9999H,9999H,9999 Display codes re the offsets do routines to be un ;Flags ;A register ;B ;C ;D ;L ;BC ;DE ;HL ;Stack pointer ;Memory	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00729 00730 00730 00801 00801 00802 00803 00804 00805 00806 00805 00806 00806 00806 00807 00806 00807 00810 00811 00812 00813 00814 00815 00816 00817 00818 00819 00821 00821 00821 00821 00821	0400 0000 0002 0004 0006 0000 0000 0000		7 Test\$St. ; ; ; ;; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These et for vari Equ Equ Equ Equ Equ Equ Equ Equ Equ Equ	DW 4000H for DB\$1 juates arrows autes arrows	9999H,9999H,9999 ss Display codes re the offsets do outines to be un ;Flags ;A register ;B ;E ;H ;E ;H ;E ;HL ;Stack pointer ;Menory ;(BC)	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00730 00730 00730 00801 00800 00803 00803 00804 00805 00804 00805 00806 00805 00806 00807 00808 00809 00810 00811 00812 00813 00814 00815 00815 00815 00815 00817 00818 00821 00820	0400 0000 0002 0004 0006 0000 0000 0000		Test\$St. ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These e for var: EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 	9999H,9999H,9999 ss Display codes re the offsets do routines to be un ;Flags ;A register ;B ;Flags ;A register ;B ;B ;B ;B ;B ;B ;B ;B ;HL ;Stock pointer ;Memory ;(BC) ;(BL) ;(HL)	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00729 00730 00730 00801 00800 00803 00804 00803 00804 00805 00804 00805 00806 00809 00810 00811 00812 00813 00814 00815 00814 00815 00815 00816 00817 00815 00819 00820 00821 00822 00823 00824 00825	0400 0000 0002 0004 0008 0008 0000 0012 0012 0012 0014 0018 0014 0016 0012		Test\$St.       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	ORG Debug su Equates These ec for vari EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H for DB\$1 lous sub 00 02 04 06 08 10 12 14 16 18 12 22 24 26 28 30	9999H,9999H,9999 ss Display codes re the offsets do ooutines to be us ;Flags ;A register ;B ;C ;D ;E ;H ;L ;E ;H ;E ;H ;E ;H ;E ;H ;C ; ;E ;H ;C ; ; ; ; ; ; ; ; ; ; ; ; ;	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00729 00730 00730 00801 00800 00803 00803 00803 00804 00805 00806 00805 00806 00807 00806 00807 00806 00807 00806 00807 00810 00811 00812 00815 00815 00815 00815 00815 00815 00821 00822 00823 00824 00825 00825 00826	0400 0000 0002 0004 0006 0000 0012 0014 0016 0012 0018 0012		Test\$St.;       ;;;;       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	Debug su Equates These et for vari EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 	9999H,9999H,9999 is Display codes re the offsets de routines to be un ;Flags ;A register ;B ;C ;D ;E ;HL ;BC ;DE ;HL ;Stack pointer ;Memory ;(BC) ;(DE) ;(HL) ;(DE+1),(BC) ;(DE+1),(DE)	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00801 00800 00800 00803 00804 00805 00806 00805 00806 00806 00807 00806 00807 00808 00809 00811 00812 00813 00814 00815 00814 00815 00819 00820 00821 00822 00823 00824 00825 00826 00827	0400 0000 0002 0004 0008 0008 0000 0012 0012 0012 0014 0018 0014 0016 0012		Test\$St ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Debug su Equates These et for vari EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 4000H for DB\$1 auates an out on the subr out out on the out out out out out out out out out out	9999H,9999H,9999 ss Display codes re the offsets do ooutines to be us ;Flags ;A register ;B ;C ;D ;E ;H ;L ;E ;H ;E ;H ;E ;H ;E ;H ;C ; ;E ;H ;C ; ; ; ; ; ; ; ; ; ; ; ; ;	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses
00726 00727 00728 00729 00730 00730 00801 00800 00803 00804 00803 00804 00805 00804 00805 00806 00807 00808 00809 00810 00811 00812 00813 00814 00815 00815 00815 00816 00817 00818 00817 00828 00824 00822 00823 00824 00827 00826	0400 0000 0002 0004 0006 0000 0012 0014 0016 0012 0018 0012		Test\$St.       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	Debug su Equates These et for vari EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 	9999H,9999H,9999 is Display codes re the offsets de routines to be un ;Flags ;A register ;B ;C ;D ;E ;HL ;BC ;DE ;HL ;Stack pointer ;Memory ;(BC) ;(DE) ;(HL) ;(DE+1),(BC) ;(DE+1),(DE)	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00729 00730 00730 00730 00800 00800 00800 00800 00803 00804 00805 00806 00806 00806 00807 00806 00807 00810 00811 00812 00813 00814 00815 00816 00817 00816 00821 00823 00823 00824 00825 00826 00827	0400 0000 0002 0004 0006 0000 0012 0014 0016 0012 0018 0012		Test\$St ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ORG Debug su Equates These ex for var: EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 	9999H,9999H,9999 is Display codes re the offsets de routines to be un ;Flags ;A register ;B ;C ;D ;E ;HL ;BC ;DE ;HL ;Stack pointer ;Memory ;(BC) ;(DE) ;(HL) ;(DE+1),(BC) ;(DE+1),(DE)	pH,9999H,9999H,9999H,9999H,9999H ;To avoid unnecessary listings ; when only the testbed changes 
00726 00727 00728 00730 00730 00730 00801 00800 00803 00804 00803 00804 00805 00806 00805 00806 00805 00806 00807 00808 00809 00810 00811 00812 00813 00814 00815 00814 00815 00816 00817 00820 00820 00823 00824 00825 00824 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00826 00827 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00807 00810 00812 00812 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 00827 008270	0400 0000 0002 0004 0006 0000 0012 0012 0014 0018 0014 0016 0012 0012 0014 0016 0012 0012 0014		<pre>&gt; Test\$\$t, ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;</pre>	Debug su Equates These et for vari EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H for DB\$1 quates arrows aubroutine for DB\$1 quates arrows aubrous subr 00 02 04 06 08 04 06 08 10 12 14 16 18 18 20 22 4 24 26 28 30 32 34 36	9999H,9999H,9999 rs Display codes re the offsets do outines to be un Flags A register B C D E HL Stack pointer Memory (BC) (DE) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC)	ры, 9999н, 9999н, 9999н, 9999н, 9999н ; To avoid unnecessary listings ; when only the testbed changes own the table of addresses sed.
00726 00727 00728 00729 00730 00730 00730 00800 00800 00800 00800 00803 00804 00805 00806 00806 00806 00807 00806 00807 00810 00811 00812 00813 00814 00815 00816 00817 00816 00821 00823 00823 00824 00825 00826 00827	0400 0000 0002 0004 0006 0000 0012 0014 0016 0012 0018 0012		Test\$St.       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	ORG Debug su Equates These ex for var: EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	DW 4000H 	9999H,9999H,9999 rs Display codes re the offsets do outines to be un Flags A register B C D E HL Stack pointer Memory (BC) (DE) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC) (BC)	эн, 9999н, 9999н, 9999н, 9999н, 9999н ;To avoid unnecessary listings ; when only the testbed changes own the table of addresses

Figure 10-2. (Continued)

00832	0028	-	RST5	EQU	28H	;Address for RST 5 - OUT instruction
00833			1			
00834	0001		B#CONIN	EQU	1	BDOS CONIN function code
00835	0002		B\$CONOUT	EQU	2	<pre>\$BDOS CONOUT function code</pre>
00836	000A		BSREADCONS	EQU	10	;BBOS read console function code
00837	0005	=	BDOS	EQU	5	BDOS entry point
00838			7		_	
00839	0000		False	EQU	0	
00840	FFFF		True	EQU	NOT Fals	i e
00841			3			
00842						Figurates to specify how DB\$CONOUT
00843						<pre># and DB#CONIN should perform</pre>
00844				FOU	F	; their input/output
00845	0000			EQU	False	;)
00846	0000		DB\$BIOS\$IO	EQU	False	;) Only one must be true
00847	FFFF		DB\$BDOS\$I0	EQU	True	;)
00848			;			Fruches (an art) of 10
				500		Equates for polled I/O
00850	0001			EQU	01H	;Console status port
00851	0002	-	DB\$Data\$Port	EQU	02H	;Console data port
00852			; 	FOU		on a factorization data usadu
00853	0002		Det angle at the aut		0000\$001	
00854	0001	-	DB\$Output\$Ready	260	0000#000	1B ;Ready for output
00855			ĩ			;Data for BIOS I/O
00856	0400	<b>C</b> 2	BIOS\$CONIN:	DB	JMP	;Data for B105 170 ;The initialization routine sets these
00857	0400		BIGSPLONINI	DW	0	; two JMP addresses into the BIOS
00858 00859			BIOS\$CONOUT:	DB	JMP	; two one addresses into the bids
00859	0403 0404	0000	D103#C0N0011	DM	0 0	
00860	0404			0.44	•	
00862			; Main deb		bler	l constants
00863			, nain bec		SOLES GIN	Constants
00864	0406	00	, DB\$Flag:	DB	0	<pre>#Main debug control flag</pre>
00865	0400	00			•	; 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			;			
00879			DB\$Save\$HL:			;Save area for HL
00880	0409		DB\$Save\$L:	DB	0	
00881	040A	00	DB\$Save\$H:	DB	0	
00882						
00883	040B		DB\$Save\$SP:	DW	0	Save area for stack pointer
00884	040D		DB\$Save\$RA:	DW	•	Save area for return address
00885	040F	0000	DB\$Call\$Address:	1	DW	0 ;Starts out the same as DB\$Save\$RA
00886						; but DB\$Save\$RA gets updated during
00887						; debug processing. This value is
00888						; output ahead of the caption
00889			DB\$Start\$Address		_	;Start address for memory display
00890	0411	0000		DW	0	
00891			DB\$End\$Address:		•	;End address for memory display
00892	0413	0000		DW	0	No. 1 do concentration
00893			DB\$Display\$Code:		•	;Display code requested
00894	0415	00	_	DB	0	
00895			1			
00896			3			Stack area
00897 00898		99999999999	<b>`</b>	DW	00000	; Stack area 999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H
00898		999999999999		DW DW		, , , , , , , , , , , , , , , , , , ,
00899		999999999999		DW DW		7770, 77770, 77770, 77770, 77770, 77770, 77770 7991, 99991, 99991, 99991, 99991, 99991, 99991
00900	0436	77777777777777777777777777777777777777	DB\$Save\$E:	DB	0	*E register
00901	0446		DB\$Save\$D:	08	ŏ	;c register ;D register
00902	0448		DB\$Save\$C:	DB	ŏ	;C register
00903	0448		DB\$Save\$B:	DB DB	ŏ	jB register
00905	0449 044A	00	DB\$Save\$F:	DB	ŏ	;Flags
00905	044A	00	DB\$Save\$A:	08	ŏ	;A register
00908	0470		DB\$Stack:		-	;Debug stack area
00908						; The registers in the stack area are PUSHed
						; onto the stack and accessed directly.
00909						

Figure 10-2. (Continued)

00910						
00911		, Regis	ter canti	on message	25	
00912		j news				
00913			able belo	w, indexe	d by the	Display\$Code is used to access
00914				aption st		
00915		1				
00916		DB\$Register\$C	aptions:			
00917	044C 7204	DW	DB\$F\$R	C	;Flags	
00918	044E 7804	DW	DB\$A\$R		;A regi	ster
00919	044C 7204 044E 7804 0450 7A04	DW	DB\$B\$R		;B	
00920	0452 7C04	DW	DB\$C\$R		÷C	
00921	0454 7E04	DW	DB#D#R		; D	
00922	0456 8004		DB\$E\$R		;E	
00923	0458 8204	DW	DB\$H\$R		įН	
00924	045A 8404	DW	DB\$L\$R		\$L.	
00925	045C 8604	DW	DB\$BC\$		) BC	
00926	045E 8904	DW	DB\$DE\$		;DE	
00927	0460 8004	DW	DB\$HL\$		j HL_	
00928	0462 8F04	DW	DB#SP\$		‡Stack	pointer
00929	0464 9204	DW	DB\$M\$R		3 Memor y	,
00930	0466 A604		DB\$B\$B		; (BC)	
00931	0468 AB04		DB\$B\$D		;(DE)	
00932	046A B004		DB\$B\$H		;(HL)	
00933	046C B504 046E C104	DW	DB\$W\$B		;(BC+1)	
00934	046E C104	DW	DB\$W\$D		;(DE+1)	
00935	0470 CD04	DW	DB\$W\$H	L\$RC	;(HL+1)	,(HL)
00936		<b>;</b>				
00937	0472 466C	616773DB\$F\$RC:	DB	'Flags',	,0	\$Flags
00938	0478 4100		DB	Ά',0 'Β',0 'C',0 'D',0		:A register
00939	047A 4200		DB	B',0		1B
00940	047C 4300	DB\$C\$RC:	DB	·U·,0		10
00941	047E 4400	DB\$D\$RC:	DB	·B·,0		1D
00942	0480 4500		DB	'E',0		ŧE
00943	0482 4800		DB	1H1,0 1L1,0		3H
00944	0484 4000	DB¢L¢RC:	DB	1BC1,0		۶L ۶C
00945	0486 4243	00 DB#BC#RC:	DB	BC ,0		) BC
00946 00947	0489 4445		DB DB	'DE',0 'HL',0		#DE
				ML',0		;HL
00948	048F 5350	00 DB\$SP\$RC: 617274DB\$M\$RC:	DB	'SP',0		;Stack pointer
00949	0472 33/4	61727408\$M\$RC: 4329008\$8\$8C\$RC:	DB	'Start, '(BC)',(		ress ',0 ;Memory ;(BC)
00951	044B 2844	452900DB\$B\$DE\$RC:	DB	(DE)//	Ś	; (DE)
00952		4C2900DB\$B\$HL\$RC:	DB	(DE),( (HL),(	Ś	\$ (DE) \$ (HL)
00953	0485 2842	432B31DB\$W\$BC\$RC1	DB	<pre>(BC+1)</pre>	(BC) / A	;(HL) ;(BC+1),(BC)
00954	0401 2844	452B31DB\$W\$DE\$RC:	DB	(DE+1)		(DE+1), (DE)
00955	04CD 2849	4C2B31DB\$W\$HL\$RC:	DB	(HL+1)		;(DE+1),(DE) ;(HL+1),(HL)
00956		1				
00957		Flags	message			
00958		1				
00959	04D9 4378	5A784DDB\$Flags\$Msg:	DB	'CxZxM×E	ExIx'.0	Compatible with DDT's display
00960		;				· · · · · · · · · · · · · · · · · · ·
00961			masks us	ed to test	t user's	flag byte
00962		3				_ • •
00963		DB#F1ag#Masks:				
00964	04E4 01		DB	000000000	)1B	:Cerry
00965	04E5 40		DB	01009000		3 Zeró
00966	04E6 80		DB	10000000		Minus
00967	04E7 04		DB	00000010	OOB	;Even parity
00968	04E8 10		DB	00019000		#Interdigit carry (aux carry)
00969	04E9 00		DB	0		Terminator
01100		;#				
01101		; DB\$In:	••			
01102			routine i	nitializes	s the de	bug package.
01103		,				
01104		DB\$Init:				
01105		IF	DB\$BIO	S\$I0		;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			;Get CONIN offset in jump vector
01109		SHLD	BIOSSC	ONIN + 1		;Set up address
01110		MVI	L, OCH			;Get CONDUT offset in jump vector
01111		SHLD	BIOS\$C	ONOUT + 1		
01112		ENDIF				
01113					IMP: 1-	······
01114				;Set up	UMP ins	tructions to receive control
01115	A454 858-		4 1145	; when		instruction is executed
01116	04EA 3EC3	MVI	A, JMP		7 Set JN	P instructions at RST points
L						

Figure 10-2. (Continued)

01117	04EC 322000	STA	RST4	
01118	04EF 322800	STA	RST5	
	04F2 211A08			•••
01119		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				
	A455 00			
01124	04FE C9	RET		
01200		;#		
01201		) DB\$CON	INU	
01202				e next character from the console,
01203		; but co	out the returns to	'z" to uppercase letters.
		, Dut eu	moverting a to	z" to uppercase letters.
01204		;		
01205		BB\$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		;*	DDTHT HOTOFFCI	ford to appar and retain
01301				
		F DB\$CON		
01302		7 Thisr	outine returns th	e next character from the console.
01303		; Accord	ling to the settir	g of equates, it uses simple
01304			1/0. the BDOS (4	unction 2) or the BIOS.
01305		; ;	170, the 2000 th	diction 2) of the bios.
01306				
			arameters	
01307		,		
01308		;	A = character f	rom console
01309		;		
01310		DB\$CONIN:		
01311				<b>. . .</b> .
*****		IF	DB\$Polled\$IO	Simple polled input
01312		IN	DB\$Status\$Port	<pre>\$Check if incoming data</pre>
01313		ANI	DB\$Input\$Ready	-
01314		JZ	DBCONIN	:No
01315		IN	DB\$Data\$Port	
01316		PUSH		;Input data character
			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		thecover 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	
	0307 630300		BUUS	;BDOS returns to our caller
01326		ENDIF		
01327				
01328		IF	DB\$BIOS\$IO	;Use BIOS for input
01329		JMP	BIOSSCONIN	
		Orin	BIOSECONIN	This was set up during BIOS
01330				; initialization
01331		ENDIF		
01332				
01400		;#		
01401		DB\$CON	0.017	
01402				e character in the C register to the
01403		; consol	e, using simple p	olled I/O, the BDOS or the BIOS.
01404		7	· - ·	
01405			parameters	
01406		:		
		2	A = byte to be	Julian
01407		F		
01408		DB\$CONOUT:		
01409	050A 3A0604	LDA	DB\$Flag	Check if debug output enabled
01410	050D B7	ORA	A	· · · · · · · · · · · · · · · · · · ·
01411	050E C8	RZ		· Tomoro output if displad
	JUL 00	R4		;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\$Ready	
01416		JZ	DB\$CONOUT	; No
01417		MOV		
			A,C	;Get data byte
01418		OUT	DB\$Data\$Port	
01419		RET		
01420		ENDIF		
01421				
			BRADDOCTO	
		IF	DB\$BDOS\$IO	;Use BDOS for output
01422	050F 59	MOV	E,C	<pre>#Move into correct register</pre>
01423		MVI	C, B\$CONOUT	-
01423 01424	0510 0E02	. 11412	ROOS	IBDOS returns to our caller
01423 01424 01425		JMP	BDOS	BDOS returns to our caller
01423 01424 01425 01426	0510 0E02	JMP ENDIF	BDOS	BDOS returns to our caller
01423 01424 01425 01426 01427	0510 0E02	ENDIF		
01423 01424 01425 01426	0510 0E02		BDOS DB\$B10S\$10	
01423 01424 01425 01426 01427	0510 0E02	ENDIF		;BDOS returns to our caller ;Use BIOS for output

Figure 10-2. (Continued)

				······	
01429 01430			IOV .	A,C	Move into correct register
01430			MP	BIOS\$CONOUT	Set up during debug initialization
01431		;#	NDIF		
01500					
01502		; 1 D	8\$On		
01502				utine enables all debug	output by setting the
01504				nonzero.	A A A A A A A A A A A A A A A A A A A
01505				UNITARI VI	
01506		DB\$On:			
01507	0515 F5		USH	PSW	Preserve registers
01508	0516 3EFF		Ň	A, OFFH	
01509	0518 320604		TA	DB\$Flag	;Set control flag on
01510	051B F1		OP	PSW	
01511	051C C9	R	ET		
01600		3#			
01601		;			
01602		; D	B\$Off		
01603		I T	his rou	utine disables all debu	g output by setting the
01604		; D	B\$Flag	to zero.	
01605		;			
01606		DB\$Off:			
01607	051D F5		USH	PSW	#Preserve registers
01608	051E AF		RA	A	
01609	051F 320604		TA	DB\$Flag	;Clear control flag
01610	0522 F1		OP	PSW	
01611	0523 C9		ET		
01700		3 W			
01701		1	D#0-1		
01702			B#Set #F		- • · · · · · · · · · · · · · · · · · ·
01703					nter. Subsequent calls to DB\$Pass
01704					it reaches 0, debug output
01705			s enabl	lea.	
01706		! ~	-114		
01708			ailing	sequence	
01708		,		CALL DB#Set#Pass	
01710		1		DW Pass\$Count\$Val	
01711				DW Fessacountaval	ue la la la la la la la la la la la la la
01712		7 DB\$Set\$Pa			
01713	0524 220904		HLD	DB\$Save\$HL	Preserve user's HL
01714	0527 E1			H	Recover return address
01715	0528 05		USH	D	Preserve user's DE
01716	0529 5E			Ĕ,M	Get LS byte of count
01717	052A 23			H	JUpdate pointer
01718	052B 56			D. M	iGet MS byte
01719	052C 23			Ĥ	;HL points to return address
01720	0520 EB		CHG		HL = pass counter
01721	052E 220704			DB\$Pass\$Count	Set debug pass counter
01722	0531 EB	x	CHG		HL points to return address
01723	0532 D1			D	Recover user's DE
01724	0533 E3		THL		Recover user's HL and set
01725					; return address on top of stack
01726	0534 C9	RE	ET		
01800		1#			
01801		1			
01802			B\$Pass		
01803				itine decrements the de	
01804					takes no further action.
01805		; I1	f the r	esult is zero, it sets	the debug control flag nonzero
01806				e debug output.	
01807		;			
01808		DB\$Pass:			
01809	0535 F5			PSW	;Save user's registers
01810	0536 E5			н	
01811	0537 2A0704			DB\$Pass\$Count	;Get pass count
01812	053A 2B	DC		н	
01813	053B 7C	MC		A, H	;Check if count now negative
01814	053C B7	OF		A	
	053D FA4705	UP UP		DB\$P4\$\$\$x	;Yes, take no further action
01815	0540 220704			DB\$Pass\$Count	;Save downdated count
01815 01816		OF		L	<pre>sCheck if count now zero</pre>
01815 01816 01817	0343 85		7	DB\$Pass\$ED	;Yes, enable debug
01815 01816 01817 01818	0343 83 0344 CA4A03	JZ			
01815 01816 01817 01817 01818 01819	0344 CA4A03	DB\$Pass\$xi	1		•
01815 01816 01817 01818 01819 01819 01820	0544 CA4A03 0547 E1	DB\$Pass\$x1 PC	DP -	Ħ	∮ ;Recover user's registers
01815 01816 01817 01817 01818 01819 01820 01821 01822	0344 CA4A03	DB\$Pass\$xi	)P )P	H PSH	)  Recover user's registers

Figure 10-2. (Continued)

· · · · · · · · · · · · · · · · · · ·				
01823		1		
01824		DB\$Pass\$Ed:		;Enable debug
01825	054A 3EFF 054C 320604	MVI	A,OFFH DB\$Flag	· Cot dobug contuct flag
01827	054F C34705	STA JMP	BB\$Pass\$x	;Set debug control flag
01900	0046 034703	;#	DDFF455FX	
01901		;		
01902		DB\$Disp	lay	
01903				ry debug display routine.
01904		;		
01905		f Calling	sequence	
01906		7	CALL DE	3\$Display
01908		,		isplay\$Code
01909		:		Caption String',0
01910		,		
01911		;	Display co	de identifies which register(s) are to be
01912		;	displayed.	- -
01913		,		
01914		;		display code specifies a block of memory
01915		,	the sequen	nce 1s:
01918		7	CALL DE	3\$Display
01918		;		splay\$Code
01919		,		art\$Address,End\$Address
01920		;		Caption String',0
01921		;		
01922		DB\$Display:		
01923		1		
01924 01925	0552 220904	DB\$Display\$Enab: SHLD	DB\$Save\$HL	. ;Save user's HL
01925	0332 220904	SHLU	DD#04Ve#nL	Save user's HL
01927	0555 E3	XTHL		;Get return address from stack
01928	0556 220004	SHLD	DB\$Save\$RA	
01929	0559 E5	PUSH	н	Save return address temporarily
01930	055A 2B	DCX	н	;Subtract 3 to address call instruction
01931	055B 2B	DCX	н	; itself
01932	055C 2B	DCX	H	
01933	055D 220F04 0560 E1	SHLD	DB\$Call\$Ad H	
01934	0360 EI	POP	-	Recover return address
01936	0561 F5	PUSH	PSW	Temporarily save flags to avoid
01937		10011		; them being changed by DAD SP
01938	0562 210000	£XI	н, о	Preserve stack pointer
01939	0565 39	DAD	SP	
01940	0566 23	INX	н	Correct for extra PUSH PSW needed
01941	0567 23	INX	H	; to save the flags
01942	0568 220804 0568 F1	SHLD POP	DB\$Save\$SP PSW	:Recover flags
01943	0368 F1	PUP	F3W	;Recover flags
01945	0560 314004	LXI	SP, DB\$Stac	ck ;Switch to local stack
01946	0000 011001	241	.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	yourten to socar stack
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		<b>-</b>		
01951	0572 2A0D04	LHLD	DB\$Save\$RA	A ;Get return address
01952 01953	0575 7E 0576 321504	MOV STA	A,M DB\$Display	;Get display code #SCode
01953	0579 23	INX	H	:Update return address
01955		4170		Jopanta (statil dubledd
01956	057A FE18	CPI	DB\$M	;Check if memory to be displayed
01957	0570 029105	JNZ	DB\$Not\$Men	
01958	057F 5E	MOV	E,M	;Get DE = start address
01959	0580 23	INX	H	
01960 01961	0581 56 0582 23	MOV INX	D, M H	
01961	0582 23 0583 EB	XCHG	••	;HL = start address
01963	0584 221104	SHLD	DB\$Start\$A	
01964	0587 EB	XCHG		;HL -> end address
01965	0588 5E	MOV	E,M	;Get DE = end address
01966	0589 23	INX	н	
01967	058A 56	MOV	D, M	
01968	058B 23	INX	н	
01969	058C EB 058D 221304	XCHG SHLD	DB\$End\$Add	;HL = end address, DE -> caption
01970	0580 221304 0590 EB	XCHG	2072107800	;HL -> caption string
				,
L				

Figure 10-2. (Continued)

r				
01972		DB\$Not\$Memory:		
01973 01974		7 7 Oktout	preamble and caption str	ring
01975			rmat for everything excep	
01976		1		
01977		1 ñnnn i	Caption String : RC = vv	/ • •
01978		· · · · ·	· · · · · · · · · · · · · · · · · · ·	•
01979		; Call A		llue (on (A, B, C)
01980		;	Register Capti	ION (M, B, C)
01982		, A carr	iage return. line feed is	s output at the start of the
01983		7 messag	e - but NOT at the end.	
01984		;		
01985		; Memory	displays look like s	
01986		·	on String : Start, End	
01987 01988				hh hh hh hh hh i cccc cccc cccc
01989		,		
01990				
01991	0591 E5	PUSH	н	Save pointer to caption string
01992	0592 CDC107	CALL	DB\$CRLF	Display carriage return, line feed
01993	0395 CD7C07	CALL	DB\$D1splay\$CALLA	plisplay DB#Call#Address in hex.
01995	0598 E1	POP	н	Recover pointer to caption string
01996		DB#Display#Cap		HL -> caption string
01997	0399 7E	MOV	Α,Μ	;Get character
01998	059A 23	INX	н	
01999	059B B7	ORA		Check if end of string
02000	059C CAA805	JZ	DB\$End\$Caption	;Yes
02002	059F E5	PUSH	н	Save string pointer
02003	05A0 4F	MOV	C,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		; DB\$End\$Caption		
02009	05A8 220D04	SHLD	DB\$Save\$RA	;Save updated return address
02010				
02011	05AB CDC807	CALL	DB\$Colon	;Display ( : (
02012				
02013	0545 341504	LDA	DB#Display#Code	Display register caption
02014	05AE 3A1504 0581 5F	MOV	E,A	;Get user's display code ;Make display code into word
02016	05B2 1600	MVI	D, O	make display code into word
02017	0584 D5	PUSH	D	;Save word value for later
02018				
02019	0585 FE18	CPI	DB\$M	Memory display is a special case
02020	05B7 CACF05	JZ	DB\$Display\$Mem\$Caption	;Yes
02021	05BA 214C04	LXI	H,DB\$Register\$Captions	;Make pointer to address in table
02023	058D 19	DAD	D	;HL -> word containing address of
02024				; register caption
02025	OSBE SE	MOV	E, M	;Get LS byte of address
02026	05BF 23 05C0 56	INX MOV	H	DE -> vanistov antien stving
02027	05C1 EB	XCHG	D, M	;DE -> register caption string ;HL -> register caption string.
02029	05C2 CDEE07	CALL	DB\$MSG	;HL -/ register caption string. ;Display message addressed by HL
02030				
02031	05C5 CDD607	CALL	DB\$MSGI	;Display in-line message
02032	05C8 203D2000	DB	1 = 1,0	
02033	05CC C3ED05	JMP	DB\$Select\$Routine	;Bo to correct processor
02034		₽ DB\$Displey\$Mem	SCaption:	;The memory display requires a special
02036				; caption with the start and end
02037				; addresses
02038	05CF 219204	LXI	H, DB\$M\$RC	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	05E4 2A1304	LHLD	DB\$End\$Address	;Get end address
02040	VJE4 241304	LUCD	00721107A001 633	yoet end douress
<b></b>				

Figure 10-2. (Continued)

					······································	
02049		CD8707		CALL	DB\$DHLH	;Display HL in hex.
02050	05EA	CDC107		CALL	DB\$CRLF	;Display carriage return, line feed
02051						Drop into select routine
02052			DB\$Sele	-+ + D+	iner	, biop thto select rodtine
02053	05ED		DD#9616	POP	D	Description of the Discharge of the
					-	Recover word value Display\$Code
02054	OSEE	210A06		LXI	H,DB\$Display\$Ta	
02055	05F1	19		DAD	D	;HL -> address of code to process
02056						; display requirements
02057	05F2	55		MOV	E,M	;Get LS byte of address
02058	05F3			INX	н	;Update pointer
02059	05F4	56		MOV	D, M	;Get MS byte of address
02060	05F5	EB		XCHG		;HL -> code
02061						
02062	A557	LIEDOE		LXI	D,DB\$Exit	·Entre limb - stants
		11FB05				<b>;Fake link on stack</b>
02063	05F9	D5		PUSH	D	
02064	05FA	E9		PCHL		;"CALL" display processor
02065						
02066			DB\$Exit			;Return to the user
		-	DD#CXIC			
02067	05FB			POP	D	Recover user's registers saved
02068	05FC	C1		POP	B	; on local debug stack
02069	05FD	F1		POP	PSW	
02070		2A0804		LHLD	DB\$Save\$SP	;Revert to user's stack
02071	0601			SPHL	207047070	JUCTOR C SC SINGLE P PLACE
02072	0602	2A0D04		LHLD	DB\$Save\$RA	;Get updated return address (bypasses
02073						; in-line parameters)
02074	0605	E3		XTHL.		Replace on top of user's stack
02075		2A0904		LHLD	DB\$Save\$HL	;Get user's HL
					DD454AEBUL	
02076	0609	UY		RET		;Transfer to correct return address
02077						
02078						
02079			DB\$Disp	lay\$Tab	le:	
02080	060A	2006		DW	DP\$F	;Flags
02081	0600			DW	DP\$A	
						;A register
02082	060E	5A06		DW	DP\$B	; B
02083	0610	6006		DW	DP\$C	;C
02084	0612	6606		DW	DP\$D	;D
	0012	20000		DW	DP\$E	
02085	0614	0000				ŧE
02086	0616	7206		DW	DP\$H	;H
02087	0614 0616 0618	7806		DW	DP\$L	۶L
02088	061A	7E06		DW	DP\$BC	, BC
02089		8406		DW	DP\$DE	; DE
02090	061E	8A06		DW	DP\$HL	;HL
02091	0620	9006		DW	DP\$SP	;Stack pointer
02092	0622 0624	9606		DW	DP\$M	Memory
02073	0624	4907		DW	DP\$B\$BC	; (BC)
	0626	5007		DW	DP\$B\$DE	
02094						; (DE)
02095	0628			DW	DP\$B\$HL	;(HL)
02096	062A	5E07		DW	DP\$W\$BC	;(BC+1),(BC)
02097	062C			DW	DP\$W\$DE	;(DE+1),(DE)
02098	062E			DW	DP\$W\$HL	
	<b>UOXE</b>	/20/		DW	DF ###FIC	;(HL+1),(HL)
02200			;#			
02201			;	Debug (	display processing	routines
02202			;			
02203			DP\$F:			:Flags
02203						The flags are displayed in the same way that
02205						; DDT uses: C1ZOMOEOIO
02206		3A4A04		LDA	DB\$Save\$F	;Get flags
02207	0633			MOV	B,A	;Preserve copy
02208		21DA04		LXI	H, DB\$Flags\$Msg	
02209	0427	11E404			D DRAFI and Mark	;DE -> table of flag mask values
	003/	****	DDATES		STODE TEGALISES	ADE & FORTE AL LIER WERK AGINES
02210			DB\$F\$Ne		_	
02211	063A			LDAX	D	;Get next flag mask
02212	063B	B7		ORA	A	;Check if end of table
02213	0630	CA4E06		Jz	DB\$F\$Display	;Yes, display the results
02214	0000	0.172.00		~-	set thispid)	liest graptal the leaders
	a / a =				_	
02215	063F			ANA	B	;Check if this flag is set
02216	0640	3E31		MVI	A, 11	;Assume yes
02217	0642	C24706		JNZ	DB\$F\$NZ	;Yes, it is set
02218	0645			MVI	A, '0'	;No, it is clear
	0043	0200	DDecen		., .	,,
02219			DB\$F\$NZ			
02220	0647	77		MOV	M,A	;Store 'O' or '1' in message text
02221	0648	23		INX	н	;Update pointer to next 0/1
	0649			INX	H	
				INX	n	;Update flag mask pointer
02222		13		1144	U	Schoole Ites meas terring
02222 02223	064A					
02222		C33A06		JMP	DB\$F\$Next	
02222 02223 02224			DB\$F\$Di		DB\$F\$Next	Display results
02222 02223	06 <b>4</b> B		DB\$F\$Di		DB\$F\$Next H,DB\$Flags\$Msg	;Display results

02227	0651 C3EE07		JMP	DB\$M\$Q	Display message and return
02228 02229		1 DP\$A:			
02230	0654 3A4B04	DEAN	LDA	;A register DB\$Save\$A	3Get saved value
02231	0657 C39107		JMP	DB\$DAH	Display it and return
02232		1		_	
02233 02234	065A 3A4904	DP\$B:	LDA	;B BB\$Save\$B	;Get saved value
02235	065D C39107		JMP	DB\$DAH	;Display it and return
02236		;			
02237 02238	0660 3A4804	DP\$C:		C BRECourse	-0-4
02238	0663 C39107		LDA JMP	DB\$Save\$C DB\$DAH	≇Get saved value ≇Display it and return
02240		;	0.11		jorspra, it and return
02241		DP#D:		; D	
02242 02243	0666 3A4704 0669 C39107		LDA JMP	DB\$Save\$D DB\$DAH	;Get şaved value ;Display it and return
02244	000/ 03/10/		OFIC	00 <b>7</b> 000	joispiay it and return
02245		DP\$E:		;E	
02246 02247	066C 3A4604 066F C39107		LDA JMP	DB#Save#E DB#DAH	;Get saved value
02248	0000 (3910)		JMF	DBADUU	Display it and return
02249		DP\$H:		1H	
02250	0672 3A0A04		LDA	DB\$Save\$H	;Get saved value
02251 02252	0675 C39107	-	JMP	DB\$DAH	;Display it and return
02253		DP\$L:		ŧL	
02254	0678 3A0904		LDA	DB\$Save\$L	;Get saved value
02255 02256	067B C39107		JMP	DB\$DAH	Display it and return
02257		DP\$BC:		; BC	
02258	067E 2A4804		LHLD	DB\$Save\$C	;Get saved word value
02259 02260	0681 C38707		JMP	DB\$DHLH	;Display it and return
02260		; DP\$DE:		; DE	
02262	0684 244604		LHLD	DB#Save#E	rGet saved word value
02263	0687 C38707		JMP	DB‡DHLH	Display it and return
02264 02265		; DP\$HL:		;HL	
02266	068A 2A0904		LHLD	DB\$Save\$HL	;Get saved word value
02267	068D C38707		JMP	DB\$DHLH	Display it and return
02268 02269		: DP\$SP:		;Stack Pointer	
02270	0690 2A0B04	DEASEI	LHLD	DB\$Save\$SP	;Get saved word value
02271	0693 C38707		JMP	DB\$DHLH	Display it and return
02272		; DP\$M:		- M	
02273 02274	0696 2A1304	DP\$11	LHLD	;Memory DB\$End\$Address	;Increment end address to make
02275	0699 23		INX	н	; arithmetic easier
02276	069A 221304		SHLD	DB\$End\$Address	
02277 02278	069D 2A1104		LHLD	DB\$Start\$Address	
02279	06A0 CD3A07		CALL		;Compare HL to End\$Address
02280	06A3 BAD106		JC	DB\$M\$Address\$QK	
02281 02282	06A6 CDD607 06A9 0D0A2A2A2		CALL DB	DB\$MSGI	}Errör start > end R – Start Address > End **′,0
02283	06CD C9	.0	RET	UDH, VAH, *** ERRU	R - Start Address > End ##",U
02284		,			
02285 02286	OGCE CDC107	DB#M\$Ne:	Kt\$Line: CALL	DB#CRLF	
02287		DB\$M\$Ada	iress\$0K		;Output carriage return, line feed ;Bypass CR,LF for first line
02288	06D1 CDD607		CALL	DB\$MSGI	;Indent line
02289	06D4 202000		DB	r r,0	
02290 02291	06D7 2A1104 06DA CD8707		LHLD CALL	DB\$Start\$Address DB\$DHLH	;Get start of line address ;Display in hex
02292			UNCC	DDFDACA	forspray in nex
02293	OGDD CDC807		CALL	DB\$Colon	;Display ′ : ′
02294 02295	06E0 2A1104		LHLD	DB\$Start\$Address	
02296	VOLV LAIIUT	DB\$M\$Ne;	<t\$hex\$b;< td=""><td></td><td></td></t\$hex\$b;<>		
02297	06E3 E5		PUSH	н	Save memory address
02298 02299	06E4 CDD007 06E7 E1		POP	DB\$Blank H	sOutput a blank
02300	06E8 7E		MQV	H A.M	Recover current byte address
02301	06E9 23		INX	Н	Update memory pointer
02302 02303	06EA E3 06EB CD9107		PUSH CALL	H DB‡DAH	Save for later
02303	OSEE EI		POP	UB\$UAH H	pDisplay in hex. Recover memory updated address
					Surrest went, dhared entites

				Company III was and address.
02305	OGEF CD3A07	CALL	DB\$M\$Check\$End	Compare HL vs.end address
02306	06F2 CAFE06	JZ	DB\$M\$Display\$ASCII	yYes, end of area
02307	06F5 7D	MOV	A,L	;Check if at start of new line,
02308	06F6 E60F	ANI	0000\$1111B	<pre># (is address XXXOH?)</pre>
02309	06F8 CAFE06	JZ	DB\$M\$Display\$ASCII	;Yes
02310	06FB C3E306	JMP	DB\$M\$Next\$Hex\$Byte	;No, loop back for another
02311				
02312		DB\$M\$Display\$A	SCII	;Display bytes in ASCII
02313	06FE CDC807	CALL	DB\$Colon	Display :
02314	0701 2A1104	LHLD	DB\$Start\$Address	Start ASCII as beginning of line
02315	0,01 241104	DB\$M\$Next\$ASC1	IsRutes	,
02316	0704 7E	MOV	A, M	;Get byte from memory
02317	0705 E5	PUSH	H	Save memory address
	0705 E5	ANI	0111#1111B	Remove parity
02318		MOV		Prepare for output
02319	0708 4F		C,A	
02320	0709 FE20	CPI		;Check if non-graphic
02321	070B D21007 070E 0E2E	JNC	DB\$M\$Display\$Char	;Char >= space
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, '.'	Force to '.'
02327		•	•	
02328		DB\$M\$Not\$DEL:		
02329	0717 CD0A05	CALL	DB\$CONOUT	;Display character
02329	071A E1	POP	H	Recover memory address
		INX	Н	JUpdate memory pointer
02331	071B 23			Update memory pointer
02332	071C 221104	SHLD	DB\$Start\$Address	
02333	071F CD3A07	CALL	DB\$M\$Check\$End	Check if end of memory dump
02334	0722 CA3707	JZ	<b>DB\$M\$E</b> xit	:Yes, done
02335	0725 7D	MOV	A,L	;Check if end of line
02336	0726 E60F	ANI	0000\$1111B	; by checking address = XXXOH
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
	072E C20407	JNZ		
02340	072E C20407		DB\$M\$Next\$ASCII\$By	
02341		CALL	DB\$Blank	;Yes, output blank
02342	0734 C30407	JMP	DB\$M\$Next\$ASCII\$By	te ;Go back for next character
02343				
02344		3		
02345		DB\$M\$Exit:		
02346	0737 C3C107	JMP	DB\$CRLF	;Output carriage return, line feed
02347				; and return
02348				
02349		DB\$M\$Check\$End	<b>!</b>	;Compares HL vs End\$Address
02350	073A D5	PUSH	מ	;Save DE (defensive programming)
02351			-	;DE = current address
00050	073B EB	XCHG	DDeEnd&Address	
02352	0738 EB 0730 2A1304	XCHG LHLD	DB\$End\$Address	;Get end address
02353	0738 EB 073C 2A1304 073F 7A	XCHG LHLD MOV	A, D	
02353 02354	073B EB 073C 2A1304 073F 7A 0740 BC	XCHG LHLD MOV CMP	A, D H	;Get end address ;Compare MS bytes
02353 02354 02355	073B EB 073C 2A1304 073F 7A 0740 BC	XCHG LHLD MOV CMP JNZ	A, D H DB\$M\$Check\$End\$X	;Get end address ;Compare MS bytes ;Exit now as they are unequal
02353 02354 02355 02356	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B	XCHG LHLD MOV CMP JNZ MOV	A, D H DB\$M\$Check\$End\$X A, E	;Get end address ;Compare MS bytes
02353 02354 02355 02356 02356 02357	073B EB 073C 2A1304 073F 7A 0740 BC	XCHG LHLD MOV CMP JNZ MOV CMP	A,D H DB\$M\$Check\$End\$X A,E L	;Get end address ;Compare MS bytes ;Exit now as they are unequal
02353 02354 02355 02356 02356 02357 02358	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD	XCHG LHLD MGV CMP JNZ MGV CMP DB\$M\$Check\$End	A,D H DB\$M\$Check\$End\$X A,E L	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes
02353 02354 02355 02356 02356 02357	0738 EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0746 EB	XCHG LHLD MOV CMP JNZ JNZ MOV CMP DB\$M\$Check\$Enc XCHG	A,D H DB\$M\$Check\$End\$X A,E L	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address
02353 02354 02355 02356 02356 02357 02358	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD	XCHG LHLD MGV CMP JNZ MGV CMP DB\$M\$Check\$End	A,D H DB\$M\$Check\$End\$X A,E L	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes
02353 02354 02355 02356 02357 02358 02359 02360	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD	XCHG LHLD MOV CMP JNZ JNZ MOV CMP DB\$M\$Check\$Enc XCHG	A,D H DB\$M\$Check\$End\$X A,E L I\$X1	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE
02353 02354 02355 02356 02357 02358 02359 02360 02361	0738 EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0746 EB	XCHG LHLD MOV CMP JNZ MP DB\$M\$Check\$Enc XCHG POP	A,D H DB\$M\$Check\$End\$X A,E L I\$X1	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address
02353 02354 02355 02355 02357 02358 02359 02360 02361 02362	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD	XCHG LHLD MOV CMP JNZ MOV CMP DB\$M\$Check\$Enc XCHG POP RET ;	A,D H DB\$M\$Check\$End\$X A,E L L SX: D	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE
02353 02354 02355 02356 02357 02358 02359 02360 02361 02362 02363	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 EB 0745 EB 0746 C9	XCHG LHLD MOV CMP JNZ DB\$M\$Check\$EC XCHG POP RET ; DP\$B\$BC1	A,D H DB\$M\$Check\$End\$X A,E L I\$X: D ; (BC)	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Recover DE ;Return with condition flags set
02353 02354 02355 02356 02357 02358 02359 02360 02361 02362 02363 02364	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0746 EB 0747 D1 0748 C9	XCHG LHLD MOV CHP JNZ MOV CHP DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BC: LHLD	A,D H DB\$M\$Check\$End\$X A,E L \$X1 D ;(BC) DB\$Save\$C ;G	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value
02353 02354 02355 02355 02357 02358 02359 02360 02361 02362 02364 02364 02365	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0748 C9 0749 2A4804 074C 7E	XCHG LHLD MOV CMP JNZ MOV CMP DB\$M\$Check\$End XCHG POP RET ; DP\$B\$BC: LHLD MOV	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D JB\$Save\$C 1G A,M 1G	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it
02353 02354 02355 02355 02357 02358 02359 02360 02361 02362 02363 02364 02365 02365	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0746 EB 0747 D1 0748 C9	XCHG LHLD MOV CHP JNZ MOV CHP DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BC: LHLD	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D JB\$Save\$C 1G A,M 1G	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value
02353 02354 02355 02355 02357 02358 02359 02360 02361 02362 02363 02364 02365 02364 02365	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0748 C9 0749 2A4804 074C 7E	XCHG LHLD MOV CHP JNZ MOV CHP DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BC: LHLD MOV JHP ;	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D 1\$(BC) DB\$Save\$C A,M JB\$SAVe DB\$DAH JD	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it
02353 02354 02355 02355 02356 02359 02360 02360 02362 02363 02364 02365 02364 02365 02364 02365 02364	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0745 C9 0749 2A4804 0747 7E 0749 C39107	XCHG LHLD MOV CMP JNZ DB\$M\$Check\$EC POP RET ; DP\$B\$BC: LHLD MOV NOV ; DP\$B\$DE:	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C A,M DB\$Save\$C JG DB\$Save\$C JG DB\$Save JG DB\$DAH JC DB\$DAH JC DB\$DAH JC DB\$DAH	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return
02353 02354 02355 02355 02355 02357 02358 02359 02361 02362 02363 02364 02365 02364 02365 02366 02366 02369	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 074D C39107	XCHG LHLD MOV CMP JNZ DB\$M\$CheckSEnc XCHG POP RET ; DP\$B\$BEC: LHLD jDP\$B\$DE: LHLD	A,D H DB\$M\$Check\$End\$X A,E L I\$X1 D DB\$Save\$C A,M JG DB\$Save\$C J(DC) DB\$Save\$E 3G CDC) DB\$Save\$E 3G	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value
02353 02354 02355 02355 02357 02358 02357 02360 02361 02362 02363 02364 02364 02364 02364 02364 02365 02367 02368 02367	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 0749 C39107 0750 2A4604 0753 7E	XCHG LHLD MOV CMP JNZ MOV CMP DB\$M\$Check\$End XCHG POP RET ; DP\$B\$BC: LHLD MOV JMP ; DP\$B\$DE: LHLD MOV	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it</pre>
02353 02354 02355 02355 02355 02357 02358 02359 02361 02362 02363 02364 02365 02364 02365 02366 02366 02369	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 074D C39107	XCHG LHLD MOV CMP JNZ DB\$M\$CheckSEnc XCHG POP RET ; DP\$B\$BEC: LHLD jDP\$B\$DE: LHLD	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G	;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value
02353 02354 02355 02355 02357 02358 02357 02360 02361 02362 02363 02364 02364 02364 02364 02364 02365 02367 02368 02367	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 0749 C39107 0750 2A4604 0753 7E	XCHG LHLD MOV CMP JNZ DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BC: LHLD MOV JMP ; UP\$B\$DE: LHLD MOV JMP ;	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it</pre>
02353 02354 02355 02355 02357 02358 02357 02360 02361 02362 02363 02364 02365 02364 02365 02364 02365 02364 02365 02369 02371 02371	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 0749 C39107 0750 2A4604 0753 7E	XCHG LHLD MOV CMP JNZ DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BC: LHLD MOV JMP ; UP\$B\$DE: LHLD MOV JMP ;	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it</pre>
02353 02354 02355 02355 02358 02359 02360 02361 02362 02363 02364 02365 02364 02365 02366 02366 02369 02370 02371 02372 02373	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0745 C9 0749 2A4804 0747 D1 0748 C9 0749 2A4804 0747 7E 0740 C39107 0750 2A4604 0753 7E 0754 C39107	XCHG LHLD MOV CMP JNZ DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BC: LHLD MOV NOV NOV NOV NOV JMP ; DP\$B\$DE: LHLD JMP ; DP\$B\$HL:	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D ;(BC) DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G DB\$DAH ;D ;(HL)	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return</pre>
02353 02354 02355 02356 02357 02358 02359 02360 02361 02362 02363 02364 02365 02366 02365 02366 02368 02369 02370 02371 02372 02373 02374	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 0740 C39107 0750 2A4604 0753 7E 0754 C39107	XCHG LHLD MOV CHP JNZ DB\$M\$Che ck\$Enc XCHG POP RET ; DP\$B\$BE: LHLD MOV JHP ; DP\$B\$DE: LHLD MOV JHP ; DP\$B\$HL: LHLD	A,D H DB\$M\$Check\$End\$X A,E L I\$X: D DB\$Save\$C ;G A,M ;G DB\$SAH ;D DB\$SAH ;D DB\$SAH ;D JB\$SAH ;D JB\$SAH ;D JB\$SAH ;D JB\$SAH ;D	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value</pre>
02353 02354 02355 02355 02357 02358 02357 02360 02361 02362 02363 02364 02363 02364 02364 02364 02364 02364 02369 02370 02371 02372 02373 02374 02375	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0747 D1 0746 C9 0749 2A4804 074C 7E 0749 C39107 0750 2A4604 0753 7E 0754 C39107 0757 2A0904 075A 7E	XCHG LHLD MOV CMP JNZ MOV CMP DB\$M\$Check\$End XCHG POP RET ; DP\$B\$BC: LHLD MOV JMP ; DP\$B\$DE: LHLD MOV JMP ; DP\$B\$HL: LHLD MOV	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C ;G A,M ;G DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G DB\$DAH ;D ;(HL) DB\$Save\$HL ;G A,M ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it</pre>
02353 02354 02355 02355 02358 02359 02360 02361 02362 02363 02364 02365 02365 02366 02367 02368 02369 02370 02371 02371 02373 02374	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 0740 C39107 0750 2A4604 0753 7E 0754 C39107	XCHG LHLD MOV CHP JNZ DB\$M\$Che ck\$Enc XCHG POP RET ; DP\$B\$BE: LHLD MOV JHP ; DP\$B\$DE: LHLD MOV JHP ; DP\$B\$HL: LHLD	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C ;G A,M ;G DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G DB\$DAH ;D ;(HL) DB\$Save\$HL ;G A,M ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value</pre>
02353 02354 02355 02355 02357 02358 02357 02360 02361 02362 02363 02363 02364 02365 02364 02365 02364 02365 02364 02365 02367 02370 02371 02375 02375 02375	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0747 D1 0746 C9 0749 2A4804 074C 7E 0749 C39107 0750 2A4604 0753 7E 0754 C39107 0757 2A0904 075A 7E	XCHG LHLD MOV CHP JNZ DB\$M\$Check5Enc XCHG POP RET JP\$B\$BEC: LHLD MOV JMP ; DP\$B\$DE: LHLD MOV JMP ; DP\$B\$HL: LHLD MOV JMP	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D DB\$Save\$C ;G A,M ;G DB\$Save\$C ;G A,M ;G DB\$DAH ;D JB\$Save\$E ;G A,M ;G DB\$DAH ;D JB\$Save\$HL ;G A,M ;G DB\$Save\$HL ;G A,M ;G DB\$Save\$HL ;G A,M ;G DB\$Save\$HL ;G A,M ;G DB\$DAH ;D	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it</pre>
02353 02354 02355 02355 02359 02359 02360 02361 02362 02363 02364 02365 02364 02365 02366 02366 02366 02369 02370 02371 02372 02373 02375	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0745 EB 0745 C9 0749 2A4804 0747 D1 0748 C9 0749 2A4804 0747 C7 0750 2A4604 0750 2A4604 0753 7E 0754 C39107 0757 2A0904 0758 C39107	XCHG LHLD MOV CHP JNZ DB\$M\$CheK\$Enc XCHG POP RET ; DP\$B\$BE: LHLD MOV JNP ; DP\$B\$DE: LHLD MOV JNP ; DP\$B\$HL: LHLD MOV JNP ; DP\$B\$HL: LHLD MOV JNP	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D DB\$Save\$C ;G A,M ;G DB\$DAH ;D j(DE) DB\$Save\$E ;G A,M ;G DB\$DAH ;D j(HL) DB\$Save\$HL ;G A,M ;G DB\$DAH ;D j(BC) DB\$DAH ;D	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return</pre>
02353 02354 02355 02355 02357 02358 02359 02360 02361 02362 02363 02364 02365 02364 02365 02364 02365 02364 02365 02364 02367 02376 02371 02372 02374 02375 02376 02376 02377 02378	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0746 C9 0749 2A4804 074C 7E 074D C39107 0750 2A4604 0753 7E 0754 C39107 0757 2A0904 0758 C39107	XCHG LHLD MOV CMP JMZ DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BEC: LHLD MOV JMP ; DP\$B\$DE: LHLD MOV JMP ; DP\$B\$HL: LHLD MOV JMP ; DP\$B\$HL: LHLD	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D DB\$Save\$C ;G A,M ;G DB\$DAH ;D j(DE) DB\$Save\$E ;G A,M ;G DB\$DAH ;D j(HL) DB\$Save\$E ;G A,M ;G DB\$DAH ;D j(BC+1),(BC) DB\$Save\$C ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et saved word value</pre>
02353 02354 02355 02355 02357 02358 02357 02360 02361 02362 02363 02364 02363 02364 02364 02364 02365 02364 02369 02370 02373 02374 02375 02375 02378 02377 02378 02377 02378 02379	073B EB 073C 2A1304 073F 7A 0740 BC 0741 C24407 0744 7B 0745 BD 0745 BD 0745 EB 0747 D1 0748 C9 0749 2A4804 074C 7E 0749 C39107 0750 2A4404 0753 7E 0757 2A0904 0757 2A0904 0758 C39107 0755 C39107	XCHG LHLD MOV CMP JNZ DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BC: LHLD MOV JMP ; DP\$B\$DE: LHLD MOV JMP ; DP\$B\$HL: LHLD MOV JMP ; DP\$B\$HL: LHLD MOV	A,D H DB\$M\$Check\$End\$X A,E L (\$X: D DB\$Save\$C ;G A,M ;G DB\$Save\$C ;G A,M ;G DB\$DAH ;D ;(DE) DB\$Save\$E ;G A,M ;G DB\$DAH ;D ;(HL) DB\$Save\$HL ;G A,M ;G DB\$DAH ;D ;(BC+1),(BC) DB\$Save\$C ;G C,M ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return</pre>
02353 02354 02355 02355 02357 02358 02359 02360 02361 02362 02363 02364 02365 02364 02365 02364 02365 02364 02365 02364 02367 02376 02371 02372 02374 02375 02376 02376 02377 02378	073B EB 073C 2A1304 073C 7A 0740 BC 0741 C24607 0744 7B 0745 BD 0745 BD 0746 EB 0747 D1 0746 C9 0749 2A4804 074C 7E 074D C39107 0750 2A4604 0753 7E 0754 C39107 0757 2A0904 0758 C39107	XCHG LHLD MOV CMP JMZ DB\$M\$Check\$Enc XCHG POP RET ; DP\$B\$BEC: LHLD MOV JMP ; DP\$B\$DE: LHLD MOV JMP ; DP\$B\$HL: LHLD MOV JMP ; DP\$B\$HL: LHLD	A,D H DB\$M\$Check\$End\$X A,E L (\$X1 D DB\$Save\$C ;G A,M ;G DB\$DAH ;D j(DE) DB\$Save\$E ;G A,M ;G DB\$DAH ;D j(HL) DB\$Save\$E ;G A,M ;G DB\$DAH ;D j(BC+1),(BC) DB\$Save\$C ;G	<pre>;Get end address ;Compare MS bytes ;Exit now as they are unequal ;Compare LS bytes ;HL = current address ;Recover DE ;Return with condition flags set et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et byte addressed by it isplay it and return et saved word value et saved word value</pre>

	02382	0763	54		MOV	D.M			
						D, M			
	02383	0764			XCHG			ord to be displayed	
	02384	0765	C38707		JMP	DB\$DHLH	;Displa	y it and return	
	02385			;					
£	02386			DP\$W\$DE	:	;(DE+1),(DE)			
	02387	0768	264604		LHLD	DB\$Save\$E	:Get sa	ved word value	
	02388	076B			MOV	E,M		rd addressed by it	
		076C					,041 40		
	02389				INX	Н			
	02390	076D			MOV	D, M			
	02391	076E			XCHG			ord to be displayed	
	02392	076F	C38707		JMP	DB\$DHLH	;Displa	y it and return	
	02393			:					
	02394			DP\$W\$HL		:(HL+1),(HL)			
	02395	0772	240904		LHLD	DB\$Save\$HL	. Got ca	ved word value	
	02396	0775			MOV	E,M		rd addressed by it	
							tost wo	ra addressed by It	
	02397	0776			INX	н			
	02398	0777	56		MOV	D, M			
	02399	0778			XCHG		;HL = w	ord to be displayed	
	02400	0779	C38707		JMP	DB\$DHLH		y it and return	
	02401			;				,	
	02500			:*					
	02501				DRADIAN	lay\$CALLA			
				1					
	02502			;	Inis ro	utine displays th	he DB\$Ca	ll\$Address in hexadecimal,	
	02503			7	followe	d by " : ".			
1	02504			;					
	02505			DB\$Dien	ley\$CALL	A:			
	02506	077C	ES.		PUSH	Ĥ	. Save -	aller's W	
1	02507						12678 G	aller's HL	
1			2A0F04		LHLD	DB\$Call\$Address	JUNE TH	Y VALL ADDress	
	02508		CD8707		CALL	DB¢DHLH		y HL in hex.	
	02509	0783			POP	н	Recove	r caller's HL	
	02510	0784	C3C807		JMP	DB\$Colon	(Displa	y " : " and return	
	02511			,					
	02600			;#					
	02601								
				F					
	02602			;	DB\$DHLH				
	02603			\$	Display	HL in hex.			
	02604			;					
	02605				Entry P	arameters			
	02606								
1	02607					HL = value to be		und .	
E						HL - Value to be	e nizhia	yeu	
	02608			<u> </u>					
	02609			DB\$DHLH					
	02610	0787	E5		PUSH	н	;Save i	nput value	
1	02611	0788	7C		MOV	A, H	:Get MS	byte first	
	02612	0789	CD9107		CALL	DB\$DAH	i Displa	A in hex.	
	02613	078C			POP	H		r input value	
1	02614	078D			MOV		Bet LS		
						A,L			
	02615	U/BE	C39107		JMP	DB\$DAH	ADJ2614	v it and return	
1	02616			;					
	02700			s #					
1	02701			1					
	02702				DB\$DAH				
1	02703			í		A register in he	vadecim	a1	
1	02704			;					
1	02705			-	F-1				
1				;	Entry P	arameters			
1	02706			;					
1	02707			,		A = value to be	convert	ed and output	
1	02708			;					
ł.	02709			DB\$DAH:					
1	02710	0791	F5		PUSH	PSW	sTake a	copy of the value to be converted	
1	02711	0792			RRC			A right four places	
1							1 211110	a italit iodi histori	
1	02712	0793			RRC				
1	02713	0794			RRC				
1	02714	0795			RRC				
1	02715		CDB407		CALL	DB\$Nibble\$To\$Hex	c	Convert LS 4 bits to ASCII	
1	02716		CDOA05		CALL	DB\$CONOUT		Display the character	
	02717	079C			POP	PSW		Get original value again	
	02718	0700	CDB407		CALL	BB\$Nibble\$To\$Hex		Convert LS 4 bits to ASCII	
1							•		
1	02719	0740	C30A05	-	JMP	DB\$CONOUT		Display and return to caller	
	02800			3 <b>#</b>					
1	02801			;					
1	02802			1	DB\$CAH				
1	02803			1	Convert	A register to be	xadecima	al ASCII and store in	
	02804			i		ed address.			
1	02805			-					
	02806			•	Est				
1				,	Entry P	arameters			
1	02807			;					

-				
02808			A	a convertion and evidence
02808		;		e converted and output rea to receive two characters of output
02810		,	HE -> Duiller a	rea to receive two characters of output
02811			arameters	
02812		1 6010 9	arameters	
02813		1	HL -> hyte fol	lowing last hex.byte output
02814		,		
02815		DB\$CAH:		
02816	07A3 F5	PUSH	PSW	;Take a copy of the value to be converted
02817	07A4 OF	RRC		;Shift A right four places
02818	07A5 OF	RRC		
02819	07A6 OF	RRC		
02820	07A7 OF	RRC		
02821	07A8 CDB407	CALL	DB\$Nibble\$To\$H	
02822	07AB 77	MOV	M, A	;Save in memory
02823	07AC 23	INX	H	;Update pointer
02824	07AD F1	POP	PSW	;Get original value again
02825	07AE CDB407	CALL MOV	DB\$Nibble\$To\$H	ex ;Convert to ASCII hex. ;Save in memory
	07B1 77	INX	M,A H	
02827	07B2 23 07B3 C9		н	;Update pointer
02828	0/83 (9	RET		
02901		;		
02902			subroutines	
02903		1	sabi out the s	
02904		,		
02905			ble\$To\$Hex	
02906				tine that converts the least
02907		t signif	icant four bits	of the A register into an ASCII
02908			haracter in A an	
02909		1		
02910		; Entry	parameters	
02911		;		
02912		;	A = nibble to	be converted in LS 4 bits
02913		;		
02914		; Exitp	arameters	
02915		3		
02916		;	A,C ≃ ASCII h	ex. character
02917		<u>I</u>		
02918		DB\$Nibble\$To\$H		
02919	07B4 E60F	ANI	0000\$1111B	Isolate LS four bits
02920	07B6 C630	ADI	'0' '9' + 1	Convert to ASCII
02921	0788 FE3A	CPI JC		;Compare to maximum ;No need to convert to A -> F
02922	07BA DABF07	ADI	DB\$NTH\$Numeric 7	Convert to a letter
02923	07BD C607	DB\$NTH\$Numeric		JConvert to a letter
02925	07BF 4F	MOV	C, A	For convenience of other routines
02926	0700 09	RET	C, H	For conventence of other routines
02927	0/00 0/	1121		
02928				
02929		;		
02930		; DB\$CRL	F	
02931				lay carriage return, line feed.
02932				
02933		DB\$CRLF:		
02934	07C1 CDD607	CALL	DB\$MSGI	;Display in-line message
02935	07C4 0D0A00	DB	ODH, OAH, O	· · · · · · · · · · · · · · · · · · ·
02936	07C7 C9	RET		
02937	•	1		
02938		# DB\$Col		
02939			routine to disp	lay ' : '.
02940		1		
02941		DB\$Colon:		
02942	07C8 CDD607	CALL	DB\$MSGI	;Display in-line message
02943	07CB 203A2000	DB	1:1,0	
02944	07CF C9	RET		
02945		; . DDfD1.	- 14	
02946		; DB\$Bla		1
02947 02948		; Simple	routine to disp	147 •
02948		; DB\$Blank:		
02949	07D0 CDD607	CALL	DB\$MSGI	;Display in-line message
02951	0703 2000	DB	· ·,0	tershirt vi tile messede
02952	07D5 C9	RET	, .	
03100		;#		
03101		1		
03102			e processing sub	routines
1				

Figure 10-2. (Continued)

-							
	03103		;				
	03104			DREMSOT	(	∎ in-line	
	03105		÷	Output		a termini	ited message that follows the
	03106				MSGOUTI		ited message (nat follows the
			,	CALL TO	MS60011		
Ł	03107		7				
	03108		7	Calling	sequence	t	
E.	03109		3				
	03110				CALL	DB\$MSGI	
	03111		,		DB	Message	1.0
	03112					t instruc	
	03113		•			. Instruc	
				<b>F</b>			
L	03114		7	EXIT PA	rameters		
	03115		;		HL -> 1	nstructio	on following message
	03116		\$				
Ł	03117		;				
	03118		DB\$MSGI	7			
	03119						;Get return address of stack, save
	03120						; user's HL on top of stack
	03121	07D6 E3		XTHL			THL -> message
	03122	0,20 20		AE			
	03123	0707 F5		PUSH	PSW		\$Save all user's registers
							togan all after a Ledistela
1	03124	07D8 C5		PUSH	в		
1	03125	07D9 D5		PUSH	D		
1	03126		DB¢MSGI	\$Next:			
L	03127	07DA 7E		MOV	Α,Μ		;Get next data byte
	03128	07DB 23		INX	н		Update message pointer
1	03129	07DC B7		ORA	Â		Check if null byte
1	03130	07DD C2E507		JNZ	DB\$MSGI		;No, continue
1	03131				0000000	-	free wells strate
Т		07E0 D1		POP	D		;Recover user's registers
1	03132						SUCCOVEL AREL & LEGISTEL2
1	03133	07E1 C1		POP	B		
1	03134	07E2 F1		POP	PSW		
	03135	07E3 E3		XTHL.			;Recover user's HL from stack, replacing
	03136						; it with updated return address
	03137	07E4 C9		RET			Return to address after 00-byte
	03138						; after in-line message
	03139		DB\$MSGI	r.			
	03140	07E5 E5	0040301	PUSH	н		;Save message pointer
	03141	07E6 4F		MOV	C,A		Ready for output
	03142	07E7 CD0A05		CALL	DB#CONO	UT	
	03143	07EA E1		POP	н		;Recover message pointer
	03144	07EB C3DA07		JMP	DB¢MSGI	♦Next	;Go back for next char.
	03145						
L	03146		3				
	03147			DB\$MSG			
	03148		;				
1			;	σάτρατι	nu11-0yt	e termina	ated message
1	03149		;				
	03150		;	Calling	sequence	2	
	03151		;				
	03152		;	MESSAGE	:	DB	'Message',0
1	03153		;				
1	03154		;		LXI	H, MESSAG	3E
1	03155		;		CALL	DB\$MSG	
	03156						
1	03157		i i	Exit per	rameters		
	03158		,			11 6444	terminator
	03159						AAL WALLARS OF
1	03159						
1			,				
1	03161		DB\$MSG:				
1	03162	07EE F5		PUSH	PSW		;Save user's registers
1	03163	07EF C5		PUSH	В		
1	03164	07F0 D5		PUSH	D		
1	03165		DB\$MSG\$				
1	03166	07F1 7E		MOV	A, M		Get next byte for output
1	03167	07F2 B7		ORA	A A		Check if 00-byte terminator
1	03168	07F3 CA0008		JZ	DB\$MSG\$	<b>v</b>	
1		07F3 CM0008				^	FExit
	03169	07F6 23 07F7 E5		INX	н		;Update message pointer
1	03170	U/F/ ED		PUSH	H .		Save updated pointer
1	03171	07F8 4F		MOV	C,A		Ready for output
1	03172	07F9 CD0A05		CALL	DB\$CONO	UT	
1	03173	07FC E1		POP	н		Recover message pointer
1	03174	07FD C3F107		JMP	DB\$MSG\$	Next	;Go back for next character
1	03175		;				
1	03176		DB\$MSG\$	X :			
1	03177	0800 D1		POP	D		;Recover user's registers
1	03178	0801 C1		POP	B		fileseise ment a tegarteta
1	03179	0802 F1		POP	PS₩		
1	031/9	V8U2 F1		PUP	r Sw		
1							
—							

Figure 10-2. (Continued)

03180	0803 C9		RET			
03300	0003 07	;#	NE I			
03301		; *				
03302		;	Debug i	nput rout	ine	
03303		,	20009 1	input rout	1116	
03304		;	This ro	utine hel	os debug	code in which input instructions
03305		÷				e opcode of the IN instruction
03306		,				lue of OE7H (RST 4).
03307		, T			27 2 12	
03308		;	This ro	utine pic	ks up th	e port number contained in the byte
03309		÷				everts it to hexadecimal, and
03310				s the mes		iter to new bell with the
03311		,				
03312		÷		Input fr	om port	XX :
03313		,				
03314		;	It then	accepts	two char	acters (in hex.) from the keyboard,
03315		;				in A, and then returns control
03316		;				e port number
03317		,				
03318		;	*****			
03319		;	WARNING	- This r	outine u	ses both DB\$CONOUT and BDOS calls
03320		;	*****			
03321		;				
03322	0804 496E707			DB	'Input f	rom Port 1
03323	0814 5858203	A20DBIN\$Po	rti	DB	^XX : ^,	0
03324		;				
03325		;				
03326		DB\$Inpu				
03327	081A 220904		SHLD	DB\$Save\$		;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			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	H		Update return address to bypass port number
03335	0825 220004		SHLD	DB\$Save\$		Save return address
03336	0828 C5		PUSH	B		Save remaining registers
03337	0829 D5 0828 F5		PUSH PUSH	D PSW		· Paus anut number for later
03339	USZA PO		FUSH	raw		;Save port number for later
03340						
03340	0828 CDB108		CALL		Cave 60a	Save current state of debug flag
03342	COND CDD100		CALL	DDAL 1484		
03343						; and enable debug output
03344	082E CDC107		CALL	DB\$CRLF		Display carriage return, line feed
03345	0831 CD7C07		CALL			Display call address
03346	0834 F1		POP	PSW		Recover port number
03347	0835 211408		LXI	H, DBINSP	ort	recover port number
03348	0838 CDA307		CALL	DBSCAH		Convert to hex, and store in message
03349	0838 210408		LXI	H, DBINSM		;Output prompting message
03350	083E CDEE07		CALL	DB\$MSG	a s s a y u	tograd brown ring message
03351	0841 0E02		MVI	C, 2		;Get 2 digit hex.value
03352	0843 CDCF08		CALL	DBSGHV		
03353	0846 70		MOV	A,L		;Returns value in HL ;Get just single byte
03354						year year shigte wite
03355	0847 CDBF08		CALL	DB\$F1ao4	Restore	Restore debug output to previous state
03356						
03357	084A D1		POP	D		Recover registers
03358	084B C1		POP	B		· · · · · · · · · · · · · · · · · · ·
03359	084C 2A0904		LHLD	DB\$Save\$		;Get previous HL
03360	084F E5		PUSH	н		Put on top of stack
03361	0850 2A0D04		LHLD	DB\$Save\$	RA	;Get return address
03362	0853 E3		XTHL			;TOS = return address, HL = previous value
03363	0854 C9		RET			
03500		;#				
03501		;				
03502		;	Debug o	utput rou	tine	
03503		;				
03504		;				code in which output instructions
03505		;				e opcode of the OUT instruction
03506		;	must be	replaced	by a va	lue of OEFH (RST 5).
03507		;				
03508		;				e port number contained in the byte
03509		;				verts it to hexadecimal, and
03510		;	display	s the mes	sage :	
03511		;				
L						

03512 03513		1	Output to port	XX : AA
03513		J Where	AA is the content	s of the A register prior to the
03515			being executed.	
03516				d to the byte following the port number.
03517		;		
03518			**	
03519				uses both DB\$CONOUT and BDOS calls
03520		; *****	**	
03521		;		
03522		1		
03523		7075DBO\$Message:	DB 'Output	to Port 1
03524		3A20DBO#Port:	DB (XX :	
03525	0869 414100	DBO#Value:	DB 'AA',Q	
03526		1		
03527		) DB\$Output;		
03528 03529	0860 220904		DB\$Save\$HL	;Save user's HL
03527	086F E1	POP	H H	; Recover address of port number
03530	0870 2B	DCX	н	Backup to point to RST
03532	0871 220F04			; Save for later display
03533	0874 23	INX	H	Restore to point at port number
03534	0875 324804		DB\$Save\$A	Preserve value to be output
03535	0878 7E	MOV	A,M	Get port number
03536	0879 23	INX	н	;Update return address to bypass port number
03537	087A 220D04	SHLD	DB\$Save\$RA	;Save return address
03538	087D C5	PUSH	8	Save remaining registers
03539	087E D5	PUSH	D	
03540	087F F5	PUSH	PSW	;Save port number for later
03541				
03542	0880 CDB108	CALL	DB\$Flag\$Save\$O	;Save current state of debug flag
03543				; and enable debug output
03544				
03545	0883 CDC107		DB\$CRLF	Display carriage return, line feed
03546	0886 CD7C07	CALL POP	DB\$D15play\$CAL PSW	A:Display call address TRecover port number
03547 03548	0889 F1 088A 216408		H.DBO\$Port	INSCALS FALL HANDEL
03545	088D CDA307		DB\$CAH	;Convert to hex, and store in message
03550	0000 CDH007	CALL	BBQCAN	
03551	0890 3A4804	LDA	DB\$Save\$A	
03552	0893 216908		H, BBO\$Value	Convert value to be output
03553	0896 CDA307	CALL	DB\$CAH	Convert to hex, and store in message
03554				
03555	0899 215508	LXI	H,DBO\$Message	;Output prompting message
03556	089C CDEE07	CALL	DB\$MSG	
03557				
03558	089F CDBF08	CALL	DB\$F1ag\$Restor	Restore debug flag to previous state
03559				
03560	08A2 D1	POP	D	Recover registers
03561	08A3 C1	POP	B	
03562	08A4 2A0904		BB\$Save\$HL	JGet previous HL JPut on top of stack
03563	08A7 E5	PUSH	H DB\$Save\$RA	;Put on top of stack :Get return address
03564 03565	OBAB 2A0D04 OBAB E3		DDA94A6 AUH	;uet return adoress ;TOS = return address, HL = previous value
03565	OBAD E3		DB\$Save\$A	Recover A (NOTE: FLAG NOT RESTORED)
03565	08AC 3A4804	RET	00704VE7M	THECOTEL A CHOIET FERD NOT REGIONEDY
03367	COMP C7	;#		
03701		; <del>*</del>		
03702		: DB\$F1	ag\$Save\$On	
03703		s This	routine is only u	ed for DB\$IN/OUT.
03704		7 It sa	ves the current s	tate of the debug control flag,
03705		3 D\$F1a	g, and then enable	es it to make sure that
03706		DB\$IN	OUT output alway	s goes out.
03707		1		
03708	0880 00	DB\$Flag\$Previ	ous: DB	0 Previous flag value
03709		3	-	
03710		DB\$Flag\$Save\$		Paus sellen/s mesisters
03711	08B1 F5	PUSH	PSW DB#E1 +=	;Save caller's registers
03712	0882 3A0604		DB\$Flag	;Get current value
03713	0885 328008		DB\$Flag\$Previo A,OFFH	is ;Save it ;Set flag
00714	0888 3EFF 088A 320604	MVI STA	A,OFFH DB\$Flag	1947 ITAA
03714		POP	PSW	
03715	A885 E(		FOW	
03715 03716	OBBD F1			
03715 03716 03717	0880 F1 088E C9	RET		
03715 03716				

03802		; DB\$F1a	g\$Restore	
03803			outine is only use	
03804				ntrol flag, DB\$Flag, to
03805		; its fo	ormer state.	
03806 03807				
03808	08BF F5	BB\$Flag\$Restor		
03808		PUSH	PSW	<b>A I I I I I I I I I I</b>
03809	08C0 3AB008 08C3 320604	LDA STA	DB\$Flag\$Previous	Get previous setting
03811	08C6 F1	POP	DB\$F1ag PSW	Set debug control flag
03812	0807 09	RET	PSW	
03812	0867 69	REI		
03814		1		
03900		, ;#		
03901		; **		
03902			x. value	
03903		i oet ne		
03904			ubroutine outputs	a prompting message, and then reads
03905		the ke	whoard in order to	get a hexadecimal value.
03906				c in that the first non-hex value
03907		termin	ates the input. Th	a maximum number of digits to be
03908		: conver	ted is specified a	s an input parameter. If more than the
03909		t maximu	m number is enter	d, only the last four are significant.
03910				-,, the cast roat are significant,
03911		; **********	***	****
03912		;	WARN	
03913		7 DB\$GHV		ne BDOS to perform a read console
03914				I if you use this routine from
03915		7 within	an executing BIOS	•
03916				"我就就你就没我就没就就能能能能能能能能能能能能能能能能能能能。"
03917		;		
03918		; Entry	parameters	
03919		3		
03920		7	HL -> 00-byte te	rminated message to be output
03921		;		wadecimal digits to be input
03922		;		
03923		;		
03924		DB\$GHV\$Buffer:		Input buffer for console characters
03925		DB\$GHV\$Max\$Cou		
03926	08C8 00	DB	0	Set to the maximum number of chars.
03927				; to be input
03928		DB\$GHV\$Input\$C		
03929	0809 00	DB		Set by the BDOS to the actual number
03930				of chars. entered
03931		DB\$GHV\$Data\$By		
03932	08CA	DS	5	Buffer space for the characters
03933		;		
03934		;		
03935	A005 70	DB\$GHV:		
03936	08CF 79	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\$OK	;Carry set if A < 5
03939	08D5 3E04	MVI	A,4	Force to only four characters
03940	0007 000000	DB\$GHV\$Count\$0		
03941	08D7 32C808	STA	DB\$GHV\$Max\$Count	Set up maximum count in input buffer
03942	08DA CDEE07 08DD 11C808	CALL	DB\$MSG	;Output prompting message
03943		LXI	D, DB\$GHV\$Buffer	Accept characters from console
03944	08E0 0E0A	MVI	C, B\$READCONS	Function code
03945	08E2 CD0500	CALL	BDOS	
03946	ADEE ATAA		C. Deconort	
03947	08E5 0E02	MVI	C, B\$CONOUT	;Output a line feed
03948		MVI	E, OAH	
	08E7 1E0A			
03949	08E7 1E0A 08E9 CD0500	CALL	BDOS	
03949 03950	08E9 CD0500			- <b>T</b> - 4 <b>4</b> 4 - <b>T</b> - 1
03949 03950 03951	08E9 CD0500 08EC 210000	LXI	н, о	;Initial value
03949 03950 03951 03952	08E9 CD0500 08EC 210000 08EF 11CA08	LXI	H,0 D,DB\$GHV\$Data\$By	les ;DE -> data characters
03949 03950 03951 03952 03953	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908	LXI LXI LDA	H,0 D,DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou	tes ;DE -> data characters nt ;Get count of characters input
03949 03950 03951 03952 03953 03954	08E9 CD0500 08EC 210000 08EF 11CA08	LXI LXI LDA MOV	H,0 D,DB\$GHV\$Data\$By	les ;DE -> data characters
03949 03950 03951 03952 03953 03954 03955	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F	LXI LXI LDA MOV DB\$GHV\$Loop:	H,O D,DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C,A	tes \$DE -> data characters ht \$Get count of characters input \$Keep count in C
03949 03950 03951 03952 03953 03954 03955 03956	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F6 0D	LXI LDA MOV DB\$GHV\$Loop: DCR	H,0 D,DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou	tes DE -> data characters DE -> data characters Det count of characters input Steep count in C Downdate count
03949 03950 03951 03952 03953 03954 03955 03955 03956 03957	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F6 0D 08F7 F8	LXI LXI LDA MOV DB\$GHV\$LOOP: DCR RM	H,0 D,DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C,A C	tes fDE -> data characters t fDE -> data characters input fGet count of characters input fKeep count in C fDowndate count fReturn when all done (HL has value)
03949 03950 03951 03952 03953 03954 03955 03956 03957 03958	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F5 0D 08F6 0D 08F7 F8 08F6 1A	LXI LXI DA DB\$GHV\$Loop: DCR RM LDAX	H,O D,DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C,A C D	tes jDE -> data characters t gGet count of characters input gKeep count in C gDowndate count gReturn when all done (HL has value) gGet next character from buffer
03949 03950 03951 03953 03953 03954 03955 03956 03956 03958 03958	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F6 0D 08F7 F8 08F7 F8 08F8 1A 08F9 13	LXI LXI DB\$GHV\$LOOPI DCR RM LDAX INX	H,O D,DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C,A C D D	<pre>iss jDE -&gt; data characters it ;DE count of characters input ;Keep count in C ;Downdate count ;Return when all done (HL has value) ;Get next character from buffer ;Update buffer pointer</pre>
03949 03950 03951 03952 03953 03954 03955 03955 03955 03956 03957 03959 03959 03960	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F6 0D 08F7 F8 08F8 1A 08F9 13 08FA CD1809	LXI LXI DB\$GHV\$LOOP: DCR RM LDAX INX CALL	H, O D, DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C, A C D DB\$A\$To\$Upper	tes jDE -> data characters ;Get count of characters input ;Keep count in C ;Downdate count ;Return when all done (HL has value) ;Get next character from buffer ;Update buffer pointer ;Convert A to uppercase if need be
03949 03950 03951 03952 03953 03955 03955 03955 03955 03957 03958 03959 03960 03961	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F6 0D 08F7 F8 08F7 F8 08F7 IA 08F9 13 08FA CD1B09 08FD FE30	LXI LDA MOV DB\$GHV\$Loop: DCR RM LDAX INX CALL CPI	H,O D,DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C,A C D D	<pre>iss iDE -&gt; data characters it iDE -&gt; data characters input if count of characters input if count in C if count if Return when all done (HL has value) if the next character from buffer if update buffer pointer if convert A to uppercase if need be if check if less than 0</pre>
03949 03950 03951 03952 03953 03954 03955 03956 03957 03959 03959 03960 03961 03962	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F6 0D 08F7 F8 08F8 1A 08F9 13 08FA CD1B09 08FD FE30 08FD FE30	LXI LXI MOV DB\$GHV\$Loop1 DCR RM LDAX INX CALL CPI RC	H, O D, DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C, A C D D DB\$A\$To\$Upper 'O'	<pre>iss fDE -&gt; data characters it fGet count of characters input fGet count in C fDowndate count fReturn when all done (HL has value) fGet next character from buffer fUndate buffer fConvert A to uppercase if need be fCheck if less than 0 fyes, terminate</pre>
03949 03950 03951 03952 03953 03954 03955 03955 03956 03958 03958 03959 03959 03961	08E9 CD0500 08EC 210000 08EF 11CA08 08F2 3AC908 08F5 4F 08F6 0D 08F7 F8 08F7 F8 08F7 IA 08F9 13 08FA CD1B09 08FD FE30	LXI LDA MOV DB\$GHV\$Loop: DCR RM LDAX INX CALL CPI	H, O D, DB\$GHV\$Data\$By DB\$GHV\$Input\$Cou C, A C D DB\$A\$To\$Upper	<pre>isE =&gt; data characters ifE =&gt; data characters input ifEet count of characters input ifEet count in C ifEeturn when all done (HL has value) ifEet next character from buffer ifUpdate buffer pointer ifConvert A to uppercase if need be ifCheck if less than 0</pre>

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 D0	RNC		;Yes, terminate			
03969	090B D637	SUI	'A' - 10	Convert A through F to numeric			
03970	090D C31209	JMP	DB\$GHV\$Shift\$Left	\$4 ;Combine with current result			
03971		7					
03972		DB\$GHV\$Hex\$Digit:					
03973	0910 D630	SUI	101	Convert to binary			
03974		DB\$GHV\$Shift\$Left\$4:					
03975	0912 29	DAD	н	;Shift HL left four bits			
03976	0913 29	DAD	н				
03977	0914 29	DAD	н				
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		; Atou					
04103				the A register to an uppercase			
04104		; letter	if it is currently	a lowercase letter			
04105		- <b>1</b>					
04106		; Entry parameters					
04107		;					
04108		3	A = character to	be converted			
04109		7					
04110		; Exite	arameters				
04111		7					
04112		7	A = converted cha	racter			
04113		1					
04114		DB\$A\$To\$Upper:					
04115	091B FE61	CPI		Compare to lower limit			
04116	091D D8	RC		No need to convert			
04117	091E FE7B	CPI		Compare to upper limit			
04118	0920 D0	RNC		No need to convert			
04119	0921 E65F	ANI	5FH ;	Convert to uppercase			
04120	0923 C9	RET					
1							

Figure 10-2. Debug subroutines (continued)

```
B>dd1 figl0-2.hex(gr)
DDT VERS 2.0
NEXT PC
O924 0000
-sl000(gr)
0116 : Flags : Flags = Cl2OMIE1IO
0120 : A Register : A = AA
0127 : B Register : B = BB
013E : C Register : C = CC
014D : D Register : D = DD
015C : E Register : C = BE
016B : H Register : H = FF
017A : L Register : H = FF
017A : L Register : L = 11
0189 : Memory Dump #1 : Start, End Address : 0108, 0128
0109 : 05 3E AA 01 CC BB 11 EE : .>W. L:.n
0110 : DD 21 11 FF B7 37 CD 52 05 00 46 6C 61 67 73 00 : 11.. 77MR ..F1 ags.
0120 : CD 52 05 02 41 20 52 65 67 : HR.. A Re s
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. .>W. L:.n
0110 : DD 21 11 FF B7 37 CD 52 05 04 66 C6 61 67 73 00 : 11.. 77MR ..F1 ags.
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. .>W. L:.n
01B7 : Memory Dump #3 : Start, End Address : 0101, 0100
** ERROR - Start Address > End **
01C0 : 31 : 1
```

Figure 10-3. Console output from debug testbed run

```
O1E5 : BC Register :
                      BC ≈
                           BBCC
01F5 : DE Register : DE = DDEE
0205 ; HL Register : HL = FF11
0215 : SP Register : SP = 0369
O22E : Byte at (BC) : (BC) = BC
O23F : Byte at (DE) : (DE) = DE
0250 : Byte at (HL) : (HL) = F1
026A : Word at (BC) : (BC+1),(BC) = 080C
027B : Word at (DE) : (DE+1), (DE)
                                    = ODOE
028C ; Word at (HL) ; (HL+1), (HL)
                                    = OF01
Debug output has been re-enabled.
This message should display 5 times
This message should display 5 times
This message should display 5 times
This message should display 5 times
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.

		•	The	-1-+	a consiste of three components:		
		The complete source file consists of three components:					
		;			code shown here		
		;		<ol> <li>The characte</li> <li>The debug pa</li> </ol>	r I/O drivers destined for the BIOS ckage shown in Figure 10-2.		
FFF 000		TRUE FALSE	EQU EQU	OFFFFH NOT TRUE			
FFF	=	DEBUG	EQU	TRUE	For conditional assembly of RST ; instructions in place of IN and		
030	-	RSTé	EQU	30H	OUT instructions in the drivers suse RST 6 for fake incoming character interrupt		
100			ORG	100H			
		START:	LXI	SP, Test\$Stack	:Use a local stack		
	31D101 CDD101		CALL	DB\$Init	;Initialize the debug package		
	3EC3		MVI	A, JMP	:Set up RST 6 with JMP opcode		
	323000		STA	RST6	year of net o still of opened		
	21D101		LXI	H,Character#Int	errupt ;Set up RST 6 JMP address		
	223100		SHLD	RST6 + 1			
		:	Make re	neated entry to	character interrupt routine		
		:	to ener	reated entry to	ers can be captured and stored in		
		;		t buffer			
		; Testbed!					
	3EAA	.62rne0;	MVI	A, OAAH	;Set registers to known pattern		
	OICCBB		LXI	B, OBBCCH	· · · · · · · · · · · · · · · · · · ·		
	11EEDD		LXI	D, ODDEEH			
119	2111FF		LXI	H, OFF11H			
iić	2111FF F7		RST	6	;Fake interrupt for incoming character		
110	CDD101		CALL	DB\$MSGI	;Display in-line message		
120	0D0A456E74	ŧ	DB	ODH, OAH, 'Enter	I to Input Char., O to Output, D to enter		
	4444542034		DB	'DDT : ',O			
159	CDD101		CALL	DB\$CONINU	;Get uppercase character		
	FE49		CPI	11	CONIN?		
15E	CA7201		JZ	Go\$CONIN			
161	FE44		CPI	1D1	;DDT?		
163	CA6E01		JZ	Go\$DDT			
166	FE4F		CPI	101	; CONOUT?		
	CA9101		JZ	Go\$CONOUT			
16B	C31101		JMP	Testbed\$Loop	;Loop back to interrupt again		
		Go\$DDT:	RST	7	;Enter DDT (RST 7 set up by DDT)		
16E	FF C31101		JMP	7 Testbed#Loop	JENCEL DOI (MOL / SEC OF DY DDI)		
101	031101	Go\$CONII		'ssingatrooh			
172	CDD101		CALL	CONST	;Get console status		
	CA1101		JZ	Testbed#Loop	No data waiting		
178	CDD101		CALL	CONIN	;Get data from buffer		
178	CDD101		CALL	DB\$Display	;Display character returned		
17E			DB	DB\$A	; in A register		
	434F4E494E	E	DB	'CONIN returned	14,0		
18E	C37201		JMP	Go#CONIN	;Repeat CONIN loop until no chars. ; waiting		
		; Go\$CONO	117 •				
191	CDD101	0040040	CALL	CONST	;Get console status		
	CA1101		JZ	Testbed\$Loop	';No data waiting		
	CDD101		CALL	CONIN			
19A	4F		MOV	C, A	Ready for output		
	CDD101		CALL	CONOUT	;Output to console		
	C39101		JMP	Go#CONOUT	Repeat while there is still data		
	9999999999	;	DW	9999H, 9999H, 99	99H, 9999H, 9999H, 9999H, 9999H, 9999H		
141							
)1A1	9999999999	9	D₩	9999H,9999H.99	, , , , , , , , , , , , , , , , , , ,		
01B1	99999999999	9	DW DW	9999H,9999H,99 9999H,9999H,99	99H, 9999H, 9999H, 9999H, 9999H, 9999H 99H, 9999H, 9999H, 9999H, 9999H, 9999H		

Figure 10-4. Testbed for character I/O drivers in the BIOS

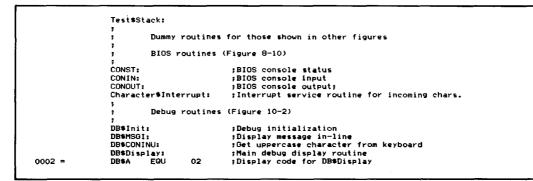


Figure 10-4. Testbed for character I/O drivers in the BIOS (continued)

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
                    81
                         Data Port
                                          80
     Status Port
     Output Ready
                    01
                        Input Ready
                                          02
     DTR high
                    40
     Reset Int. Prt D8
                         Reset Int. Val. 20
     Status Byte 1
          Output Suspended
          Output Xon Enabled
     .
```

:			
Buffer Base	OESC		
Put Offset	05	Get Offset	01
Char, Count	04	Control Count	00
Data Buffer			
41 42 43 44	45 00 0	0 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 41 H, 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 c	lock driver in the BIOS.
,			
, ,	The com	plete source fil	e consists of three components:
, ,			
,		1. The testbed	
1		2. The real tim	e clock driver destined for the BIOS.
;		3. The debug pa	ckage shown in Figure 10-2.
	RUE EQU	OFFFFH	
0000 = F	ALSE EQU	NOT TRUE	
FFFF = D	EBUG EQU	TRUE	
_ FFTT =	500 EWU	TRUE	For conditional assembly of RST
			<pre>instructions in place of IN and OUT instructions in the drivers.</pre>
0030 = R	ST6 EQU	30H	OUT instructions in the drivers. Use RST 6 for fake clock tick.
1			Jose Nor & for Take Clock (ICK.
0100	ORG	100H	
s.	TART:		
0100 318B01	LXI	SP, Test\$Stack	;Use local stack
0103 CD8B01	CALL	DB\$Init	;Initialize the debug package
0106 3EC3	MVI	A, JMP	;Set up RST 6 with JMP opcode
0108 323000	STA	RST6	
0108 218801	LXI	H,RTC\$Interrupt	;Set up RST 6 JMP address
010E 223100	SHLD	RST6 + 1	
0111 C31D01	JMP	Testbed\$Loop	<pre>;&lt;=== REMOVE THIS JMP WHEN READY TO</pre>
			; TEST WATCHDOG ROUTINES
0110 010000	,	D	
0114 013200 0117 214201	LXI	B,50	50 ticks before timeout
011A CD8B01	LXI	H,WD\$Timeout	Address to transfer to
	CALL	Set\$Watchdog	;Set the watchdog timer
2			
;	Make ve	nested entry to I	RTC interrupt routine
1 1			correctly updated
	to ensu	re that clock is	correctly updated
Í í	stbed\$Loop:		
011D JEAA	MVI	A, OAAH	;Set registers to known pattern
011F 01CCBB	LXI	B, OBBCCH	
0122 11EEDD	LXI	D, ODDEEH	
0125 2111FF	LXI	H, OFF11H	
0128 F7	RST	6	Fake interrupt clock
0129 CD8B01	CALL	DB\$MSGI	;Display in-line message
012C 436C6F636B	DB	'Clock =',0	
0134 218801	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
0134 CR0701			<pre>p a line feed character in it)</pre>
013A CD8B01 013D 0D00	CALL	DB¢MSGI	;Display in-line message
0130 0000	DB	ODH, O	;Carriage return
013F C31D01	JMP	Testbed\$Loop	
	One	iestnanaFoob	
1 :	Control	arrives here whe	n the watchdog timer times
	out		
	STimeouts		
0142 CD8B01	CALL	DB\$MSGI	
0145 0D0A576174	DB	ODH, OAH, 'Watchdo	g timed out',0
015A C9	RET		Return to watchdog routine
,			
0158 9999999999	DW	9999H, 9999H, 9999	н, 9999н, 9999н, 9999н, 9999н, 9999н
016B 9999999999	DW	9999H, 9999H, 9999	н, 9999н, 9999н, 9999н, 9999н, 9999н
017B 9999999999	DW	9999H, 9999H, 9999	н, 9999н, 9999н, 9999н, 9999н, 9999н
1	st#Stack:		
	_		<b>A</b>
	Dummy ro	outines for those	shown in other figures
	PTOC -		10)
! !	BIUS YOU	itines (Figure 8-	107
	C\$Interrupt:		nt revuice routing for slade tick
	t\$Watchdog:	sinterru	pt service routine for clock tick chdog timer
	me\$In\$ASCII:		tring of HH:MM:SS, LF, O
] ;		JHOUII S	CANE OF INTERNACO, LE, V
	Dehua vo	outines (Figure 1	0-2)
	beddy ru	NEVENCES VERSUITE A	v =/
) ĎB	\$Init:	:Debua i	nitialization
DB	\$MSGI:		message in-line
	\$MSG:	Display	message
		//	

Figure 10-5. Testbed for real-time-clock driver in the BIOS

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.

		;			ivers in the BIOS	
		1	The complete source file consists of three components:			
		;	1. The testbed code shown here			
		1		2. The Disk I/O 3. The debug pac	drivers destined for the BIOS kage shown in Figure 10-2.	
FFF	-	TRUE	EQU	OFFFFH		
000		FALSE	EQU	NOT TRUE		
FFF	-	DEBUG	EQU	TRUE	;For conditional assembly of RST ; instructions in place of IN and ; OUT instructions in the drivers.	
100		START:	ORG	100H		
	314704 CD4704		LXI CALL	SP,Test\$Stack DB\$Init	;Use a local stack ;Initialize the debug package	
		;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	Make cal		TTRK, SETSEC and SETDMA,	
		; Testbeds	Loop:			
106	314704		LXI	SP, Test\$Stack	;Use local stack	
	3A1202		LDA	Logical\$Disk	;Set up for SELDSK call	
10C	4F CD4704		MOV CALL	C, A SELDSK		
			CALL	DB\$Diselay	:Display return value in HL	
113			DB	DB\$HL.		
114	5345404453	3	DB	'SELDSK returned	31,0	
	223201		SHLD	DPH\$Start	;Set up to display disk parameter header	
127 12A	111000 19		LXI DAD	D,16 D	;Compute end address	
	223401		SHLD	DPH\$End	;Store into debug call	
12E 131	CD4704 18		CALL DB	DB\$Display DB\$M	:Display DPH ;Memory	
132	0000	DPH\$Stan	∙t: DW	0		
		DPH\$End:		0		
	0000 53656C6563	3	DW DB	Selected DPH',	>	
	2A1302		LHLD	Track H	;Call SETTRK	
146			PUSH POP	B	;SETTRK needs track in BC	
	CD4704		CALL	SETTRK		
	3A1502		LDA	Sector	Call SETSEC	
14E	4F CD4704		MOV CALL	C,A SETSEC	;SETSEC need sector in C	
					Cat DMA adduced	
	011702 CD4704		LXI CALL	B,Test\$Buffer SETDMA	;Set DMA address	
158	3A1602		LDA	Write#Disk	;Check if reading or writing	
15B 15C	B7 C2D101		ORA JNZ	A Test\$Write		
	CD4704		CALL	Read\$No\$Deblock	INNH or Read\$Physical depending on which	
162	CD4704		CALL	DB\$Display	;### drivers you are testing ;Display return code	
165	02	~	DB	DB\$A		
	546573742	0	DB	'Test Read retui		
	CD0102 CA0601		CALL JZ	Check\$Ripple Testbed\$Loop	;Check if ripple pattern in buffer ;Yes, it is correct	
	CD4704		CALL	DB\$MSGI	Indicate problem	
0182 0183	14 526970706	c	DB DB	DB\$HL 'Ripple pattern	;Display HL (points to offending byte) incorrect. HL -> failure.^,0	
	CD4704		CALL	DB\$Display	JDisplay test buffer	
)1AF	CD1800		CALL	DB\$M	; Memory	
1182	1702		DW	Test\$Buffer		

Figure 10-6. Testbed for disk I/O drivers in the BIOS

```
0184 0002
                          DW
                                    Test$Buffer$Size
01B6 436F6E7465
                          DB
                                    Contents of Test$Buffer',0
01CE C30601
                          JMP
                                   Testbed$Loop
                 Test$Write:
01D1 CDF201
                          CALL
                                   Fill$Ripple
                                                     Fill the test buffer with ripple pattern
01D4 CD4704
                          CALL
                                   Write$No$Deblock;*** or Write$Physical depending on which
                                                     ;*** drivers you are testing
01D7 CD4704
                          CALL
                                   DBSDisplay
                                                      Display return code
01DA 02
01DB 5465737420
                          DB
                                   DB$A
                          DB
                                    'Test Write returned',0
01EF C30601
                          JMP
                                   Testbed$Loop
                 Fill$Ripple:
                                                     Fills the Test$Buffer with a pattern
                                                     ; formed by putting into each byte, the
; least significant 8-bits of the byte's
                                                         address.
01F2 010002
                                   B, Test$Buffer$Size
                          LXI
01F5 211702
                          LXI
                                   H. Test$Buffer
                 FR$Loop:
                          MOV
01F8 75
                                                     ;Set pattern value into buffer
                                   M.L
01F9 23
                          INX
                                   H
                                                     ;Update buffer pointer
O1FA OB
                          DCX
                                   в
                                                     ;Down date count
01FB 79
                          MOV
                                   A,C
                                                     Check if count zero
OIFC BO
                          ORA
                                   B
01FD C2F801
                          JNZ
                                   FR$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
                          LXI
                                   B, Test$Buffer$Size
0204 211702
                         LXI
                                   H, Test$Buffer
                 CR$Loop:
0207 7D
                          MOU
                                   A,L
                                                     ;Get correct value
0208 BE
                          CMP
                                   м
                                                     ;Compare to that in the buffer
0209 CO
                          RNZ
                                                     #Mismatch, nonzero already indicated
#Update buffer pointer
020A 23
                          INX
                                   н
020B 0B
                          DCX
                                   в
                                                     ;Downdate count
0200 79
                          MOV
                                   A,C
                                                     ;Check count zero
020D B0
                          ORA
020E C20702
                          JNZ
                                   CR$Loop
                                                     ;Repeat until zero
0211 C9
                          RET
                                                     ;Zero flag will already be set
                 ;
                         Testbed variables
                 :
0212 00
                Logical$Disk:
                                   DB
                                            0
                                                     A = 0, B = 1, \dots
0213 0000
                                                     ;Disk track number
                 Tracks
                                   DW
                                            0
0215 00
                                                     ;Disk sector number
                 Sectors
                                   DB
                                            0
0216 00
                 Write$Disk:
                                   DB
                                            0
                                                     ;NZ to write to disk
                                            EQU
0200 =
                 Test$Buffer$Size
                                                     512
                                                              ;<=== Alter as required
                                            Test$Buffer$Size
0217
                 Test$Buffer:
                                  DS
0417 99999999999
0427 99999999999
0437 99999999999
                          DW
                                   9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H
                          DW
                                   9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H
                         DW
                                   9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H
                 Test$Stack:
                         Dummy routines for those shown in other figures
                         BIOS routines (Figure 8-10)
                 SELDSK:
                                            ;Select logical disk
                 SETTRK:
                                            ;Set track number
                 SETSEC:
                                            ;Set sector number
                 SETDMA:
                                            Set DMA address
                 Read$No$Deblock:
                                            ;Driver read routines
                 Read$Physical:
                 Write$No$Deblock:
                                            ;Driver write routines
                 Write$Physical:
```

Figure 10-6. (Continued)

```
Debug routines (Figure 10-2)
                 DR&Init:
                                              :Debug initialization
                 DR&MSGI+
                                              ;Display message in-line
;Main debug display routine
                 DB$Display:
0002 =
                 DB$A
                           EQU
                                    02
                                              Display codes for DB$Display
                                    20
0014 =
                 DB$HL
                           EQU
0018 =
                 DB$M
                           EOU
```

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.

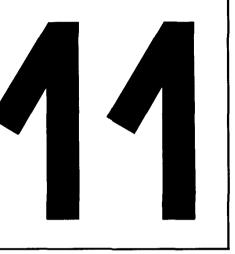
Check that the function key decode logic is working correctly. With the A> prompt displayed, press a function key. The CONIN driver should inject the correct function key string and it should appear on the terminal. For example, with the BIOS in Figure 8-10, pressing PF1 on the VT-100 terminal should 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 */
                                                                     ß
int code; /* Code that specifies the object
                  whose address is required */
/* Exit Parameters
  Address returned by the BIOS routine */
char *retval;
                    /* Value returned by the BIOS */
       retval = biosh(CBGADDR,code);
/* printf("\nget_cba : code %d address %4x",code,retval); */
      return retval;
} /* End of get_cba(code) */
/∗
      Character manipulation functions */
strscn(string,key)
                          /* String scan */
/* 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;
                   /* String to be searched */
/* Key string to be searched for */
char *key;
/* Exit parameters
  Pointer to key string within searched string, or
  zero if key not found
                                                                     b
while (*string)
                   /* For all non-null chars. in string */
       if ((*string == *key) &&
                                  /* First char. matches */
          (sstrcmp(string,key) == 0) /* Perform substring
                                    compare on rest */
         >
             return string;
                                  /* Substring matches,
                                     return pointer *,
                                 /* Move to next char. in string */
       string++;
      3
return 0:
                                  /* Indicate no match found */
} /* End of strsen */
ustrcmp(string1,string2)
                             /* Uppercase string compare */
C
/* This function is similar to the normal stremp 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;
                        /* Pointer to first string */
char *string2;
                         /* 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-mull, and
match their counterparts in string 2, */
while (string1[count] == string2[count])
        if (string1[++count] == (\0/) /* Last char. in string 1 */
                                          /* Indicate equality */
                 return O:
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 strcmp, except that the
length of the substring controls how many characters are compared. */
/* Entry parameters */
                         /* Pointer to main string */
/* Pointer to substring */
char *string;
char *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] == '\0') /* 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 */
                       /* Pointer to main string */
/* Pointer to substring */
char *string;
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 0;
return substring[count] - string[count];
                                                        /* "Compare" chars. */
} /* End of usstremp */
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 OxE5 (this value is used to indicate an inactive directory entry). */
/* Entry parameters */
struct scb *scb;
                          /* Pointer to search control block */
/* Pointer to file name */
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 wmaski
/* 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[O] == 0xE5)
                  return NAME NE; /* Indicate inequality */
         /* First char. of mask is not "?" */
else
         if (mask[0] != name[0]) /* User numbers do not match */
                  return NAME_NE; /* Indicate inequality */
         1
/* No, check the name (and, if the length is such, the extent) */
     (count = 1; /* Start with first name character */
count <= scb -> scb_length; /* For all required characters */
for (count = 1;
      count++)
                                      /* Move to next character */
         if (maskEcount] == '?') /* 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] & 0x7F))
                  { /* 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);
                  1
         /* 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. */
return NAME EQ;
                         /* Indicate mask and name are "equal" */
} /* End of comp_fname */
conv_fname(fcb,fn)
/* This function converts the contents of a file control
block into a printable string "D:FILENAME.TYP." */
/* Entry parameters */
                                    /* Pointer to file control block */
/* Pointer to area to receive name */
struct fcb *fcb;
char #fn;
         /* If the disk specification in the
FCB is 0, use the current disk */
*fn++ = (fcb -> fcb_disk) ? (fcb -> fcb_disk + ('A'-1)) :
    (bdos(GETDISK) + 'A');
                                                                                             g
                                             /* Insert disk id. delimiter */
#fn++ = ':':
movmem(&fcb -> fcb_fname,fn,8);
                                              /* Move file name */
                                              /* Update pointer */
fn += 8;
*fn++ = '.'
                                             /* Insert file name/type delimiter */
/* Move file type */
movmem(&fcb -> fcb_fname+8,fn,3);
                                              /* Remove any attribute bits */
/* Remove any attribute bits */
*fn++ &= 0x7F;
*fn++ &= 0x7F;
*fn++ &= 0x7F;
*fn = 1\0';
                                             /* Remove any attribute bits #/
                                             /* 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 */
                                    /* Disk id. (A = 0, B = 1) */
short disk:
                                   /* 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');
                                     /* Update pointer to file name */
fn += 5;
movmem(&dir -> de_fname,fn,8); /* Move file name */
                                    /* Update pointer */
/* Insert file name/type delimiter */
fn += 8;
*fn++ = '.';
movmem(&dir -> de_fname+8,fn,3); /* Move file type */
                                    /* Remove any attribute bits */
*fn++ &= 0x7F;
#fn++ &= 0x7F;
#fn++ &= 0x7F;
#fn = 1\01;
                                    /* Remove any attribute bits */
                                    /* Remove any attribute bits */
/* Terminator */
```

Figure 11-1. (Continued)

```
Ϋ́h
3 /* End of conv_dfname */
get_nfn(amb_fname,next_fname) /* Get next file name */
                      /Hennesses
                                                        _____/
/* This function sets the FCB at "next fname" to contain the
   directory entry found that matches the ambiguous file name
   in "amb_fname."
   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)
×/
char bdos_func;
                       /* Set to either search first or next */
                       /* Pointer to file name in directory entry */
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)) )
        £
       return 0:
                      /* Indicate no match */
        3
/* Move file name and type */
movmem(pfname,&next_fname -> fcb_fname,11);
                      /* Indicate match found */
return 1:
} /* 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 */
                                                                            j
struct_fcb *fcb; /* pointer to file control block */
short bdos_code; /* either SEARCHF or SEARCHN */
/* Exit parameters
   0 = no match found
   NZ = pointer to entry matched (currently in buffer)
```

Figure 11-1. (Continued)

```
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 */
j
                               /* No match found */
  (r_code == 255)
       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 #/
struct _drb #drb;
                      /* Bisk request block (disk, track, sector, buffer) */
/* Exit parameters
  0 = No data available
1 = Data available
                                                                                k
-
if (!set_disk(drb))
                       /* Call SELDSK, SETTRK, SETSEC */
/* If SELDSK fails, indicate
       return Ox
                          no data available */
if (bios(DREAD))
                       /* Execute BIOS read */
       return O;
                       /* Indicate no data available if error returned */
return 11
                       /* Indicate data available */
} /* End of rd_disk */
*****************************
                            ______
                      /* Write disk (via BIOS) */
wrt_disk(drb)
/* 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
                                                                                I
   1 = Data written OK
if (!set_disk(drb))
                       /* Call SELDSK, SETTRK, SETSEC, SETDMA */
return O;
if (bios(DWRITE))
                       /* If SELDSK fails, indicate no data written */
                       /* Execute BIOS write */
                       /* Indicate error returned */
       return O:
return 1)
                       /* Indicate data written */
3 /# 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 */
struct _drb *drb;
                      /* Disk request block (disk, track, sector, buffer) */
/* Exit parameters
   0 = Invalid disk (do not perform read/write)
   1 = BIOS now set up for read/write
× /
£
       /* 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). */
awtab; /* Skewtab -> disk parameter header -> skew table */
short ##skewtab;
short phy_sec;
                      /* Physical sector */
                                                                             m
        /* Call the SELDSK BIOS entry point. If this returns
a 0, then the disk is invalid. Otherwise, it returns
           a pointer to the pointer to the skew table */
if ( !(skewtab = biosh(SELDSK,drb -> dr_disk)).
                              /* Invalid disk */
       return O;
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 BLOS */
phy_sec = biosh(SECTRN,drb -> dr_sector,*skewtab); /* Get physical sector */
bios(SETSEC,phy_sec); /* Set sector */
bios(SETDMA,drb -> dr_buffer); /* Set buffer address */
return 1:
                       /* Indicate no problems */
} /* End of setp_disk */
/¥
       Directory Management Functions
                                              */
/* Get next directory entry */
get_nde(dir_pb)
/* 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.
~ /
                              /* Directory not yet opened */
if(|dir_pb -> dp_open)
        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)

```
dir_pb -> dp_entry = dir_pb -> dp_buffer + DIR_BSZ;
dir_pb -> dp_write = 0;  /# 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_bb -> dp_entry < dir_pb -> dp_buffer + DIR_BSZ)
         - 6
                                             /* Return pointer to next entry */
         return dir_pb -> dp_entry;
         /* 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
                  dir_pb -> dp_write = 0; /* Reset the flag */
if(!rw_dir(W_DIR,dir_pb)) /* Write the directory sector */
                           err_dir(W_DIR,dir_pb); /* Report error on writing */
                           exitO;
                  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 Op
                                                      /# Indicate no more entries #/
                  /* Update sector (and if need be track and sector) */
         if (++dir_pb -> dp_sector == dir_pb -> dp_sptrk)
                  ++dir_pb -> dp_track;
                                                      /* Update track */
                  dir_pb -> dp_sector = 0;
                                                      /# Reset sector #/
         ъ
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;
3 /* 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 */
```

Figure 11-1. (Continued)

```
/* Exit parameters
    0 = Error, directory not opened
    1 = Directory open for processing
                                        /* CP/M disk parameter block */
struct _dpb #dpb;
          /* Get disk parameter block address for the disk specified in
             the directory parameter block #/
if ((dpb = get_dpb(dir_pb -> dp_disk)) == 0)
                             /# Return indicating no DPB for this disk #/
         return 0;
          /# Set the remaining fields in the parameter block #/
/* Set the remaining fields in the parameter Diock */
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_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 */
                                                                                                        0
          /* 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);
return 1;
                              /* Indicate that directory now opened */
} /* 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) */
                                                                                                        р
drb.dr_disk = dir_pb -> dp_disk;
                                                 /* Set up disk request */
drb.dr_track = dir_pb -> dp_track;
drb.dr_sector = dir_pb -> dp_sector;
drb.dr_buffer = dir_pb -> dp_buffer;
if (read_op)
         if (!rd_disk(&drb))
                                    /* Issue read command */
/* Indicate error -- no data available */
                   return Or
         3
....
          if (!wrt_disk(&drb))
                                       /* Issue write.command */
                                      /* Indicate error -- no data written */
                   return Or
return 1:
                                       /# Indicate operation complete #/
} /* End of rd_dir */
```

Figure 11-1. (Continued)

```
----
err_dir(opcode,dir_pb)
                               /* Display directory error
                      /* 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");
                breaks
        case O_DIR:
                printf("Opening");
                break;
        defaults
                printf("Unknown Operation (%d) on",opcode);
        ъ
printf(" Directory on disk %cs. ",dir_pb -> dp_disk + 'A');
} /* End of err_dir */
       /#==
/**=
                                   /# 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
        difilename.typ
        *ifilename.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 #scbi
char *fname;
short user;
                        /* User number to search for */
                                                                                 r
short extent;
                        /* Extent number to search for */
                        /* Number of bytes to compare */
int length;
None.
#/
/* Exit parameters
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 l= '1')
                                       /* Until ":" reached */
```

Figure 11-1. (Continued)

```
<sup>1</sup> # 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 l= 1 << (toupper(#fname) - 'A');
                                        /* Move to next character */
                         ++fname;
                         3
                                         /* Bypass colon */
                 ++fname:
else
        /* Use only current default disk */
                 /* Set just the bit corresponding to the current disk */
        adisks = 1 << bdos(GETDISK);
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
                                                                                       r
   bits in the bit map. */
for (disk = 0)
                        /* Start with disk A: */
     disk < 16;
                        /* Until disk P: */
/* Use next disk */
     disk++)
        £
        if ( !((1 << disk) & adisks))
                continue:
                                        /* Avoid selecting unspecified disks */
        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) ^ OxFFFF);
        2
scb -> scb_adisks = adisks;
                                /* Set bit map in SCB */
                               /* Set user number */
/* Set extent number */
/* Set number of bytes to compare */
scb -> scb_userno = user;
scb -> scb_extent = extent;
scb -> scb_length = length;
} /* 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 -
                                                                                       s
¥/
$
        /* 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:
```

Figure 11-1. (Continued)

t

u

```
<u>م</u>
            2
                з
                    4 5
                           6
                               7
                                   •
                                      9 10 11 12 13 14 15 Used Free
        1
                         202
                                199 101 211
A: 123
           20 98
                                                                 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 usernos
                              /# Current user number #/
unsigned dsum:
                              /* Sum of entries for given disk */
printf("\n
               0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Used Free"):
for (disk = 0:
                              /* Start with disk A: */
/* Until disk P: */
    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 */
       dsum = 0:
                              /* Reset sum for this disk */
       for (userno = 0)
                              /* Start with user 0 */
            userno < 16;
                              /* Until user 15 */
            USerno++)
                              /* Next user number */
               dsum += disk_map[disk][userno]; /* Build sum */
       if (dsum)
                      /* Check if any output for this disk,
and if not, display d: None */
               userno < 16;
                                     /* Until user 15 */
                    userno++)
                                     /* Next user number */
                      ....
                                         • > =
                              printf("
                      ъ
               3
       else
                      /* No output for this disk */
               £
               printf( " -- None --
       printf(" %4d %4d",disk_map[disk]fUSED_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 */
/* 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 */
```

Figure 11-1. (Continued)

The CP/M Programmer's Handbook

```
bdos(SETDISK,disk):
                                                    /* Use BDOS SETDISK function */
     return bdos(GETDPARM);
                                                    /* Get the disk parameter block */
                                                                                                    u
     } /* 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 */
    char *string;
                                         /* Pointer to string for entry */
    /* Exit parameters
       None.
    ¥/
    entry -> _ct_code = code;
entry -> _ct_sp = string;
                                                /* Set _ct_code */
/* Set string pointer */
    } /* end of ct_inti */
    unsigned
    ct parc(table.string)
                                         /* Parameter - return code */
    /* 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
                                "APRICOTS"
                       з
       A response of "O" would return code = 2, but "A" or "AP" would
be ambiguous, "APR" or "APP" would be required. */
    struct _ct *table;
                                         /* Pointer to table */
    char *string;
                                         /* Pointer to key string */
```

Figure 11-1. (Continued)

384

```
/* Matched code to return */
int acode:
int mcount;
                             /* Count of number of matches found */
mcode = CT_SNF;
                             /* Assume error */
                             /* Reset match count */
mcount = 0;
while(table -> _ct_code != CT_SNF) /* Not at end of table */
       /* Compare keyboard response to table entry using
         uppercase substring compare. #/
       if (usstrcmp(table -> _ct_sp,string) == 0)
                                                                       w
              mcount++;
                             /# Update match count */
              mcode = table -> _ct_code;
                                        /* Save code */
              3
       table++;
                            /* Move to next entry */
                             /* Only one match found */
if (mcount == 1)
                             /* Return matched code */
       return mcode;
else
                             /* Illegal or ambiguous */
       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 */
struct _ct *table;
                    /* Pointer to table */
char #string;
                     /* 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 */
       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 */
struct _ct #table;
                           /* Pointer to table */
                                                                       y
/* Exit Parameters
       None.
×/
£
while(table -> _ct_code != CT_SNF)
                                   /# Not end of table #/
       printf("\n\t\t%s",table -> _ct_sp);
                                        /# Print string #/
                                    /* Move to next entry */
       table++:
       ł
```

Figure 11-1. (Continued)

**386** The CP/M Programmer's Handbook

```
putchar((\n');
                                       /* Add final return */
                                                                              у
} /* 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;
                             /* Pointer to table */
char *string;
                              /* Pointer to string */
/* Exit parameters
   Index of entry matching string, or
CT_SNF if string not found.
                                                                              7.
int index;
                               /* Current value of index */
index = 0;
                               /* Initialize index */
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 */
з
char *ct_stri(table,index) /* Get string according to index */
          /* This function returns a pointer to the string in the
   table entry specified by the index. */
/* Entry parameters */
                             /* Pointer to table */
struct _ct *table;
int index;
                                                                             88
                              /* Index into table */
        _ct #entry; /* Entry pointer */
entry = table[index]; /* Point to entry */
struct _ct #entry;
        return entry -> _ct_sp; /* Return pointer to string */
} /# 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. */
/* Entry parameters */
                                                                              b b
struct _ct *table;
unsigned code;
                              /* Pointer to table */
                              /* Code value */
while(table -> _ct_code != CT_SNF) /* Until end of table */
        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(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 */
                     /# Pointer to a bit vector #/
struct by #by;
unsigned bytes;
                     /* Number of bytes in bit vector */
/* Exit parameter
   NZ = vector created
                                                                        cc
   0 = insufficient memory to create vector
if(!(bv -> bv_bits = alloc(bytes)))
                                      /* Request memory */
       return O;
                                      /* Request failed */
                                     /* Set length */
bv -> bv_bytes = bytes;
by -> by end = by -> by bits + bytes; /* Set pointer to end */
                                     /# Fill with 0's #/
by fill(by.0):
return 1:
} /* End by_make */
/* Fill bit vector with value */
bv_fill(bv,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 */
                    /* Pointer to bit vector */
struct _bv *bv;
                                                                        d d
char value:
                      /* Value to fill vector with */
/* Exit parameters
  None.
w/
                  length
                                value */
/₩
      address
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
                                                                        ee
   to one-bit. */
/* Entry parameters */
                              /* Pointer to bit vector */
struct _bv *bv;
unsigned bitnum;
                              /* Bit number to be set */
```

Figure 11-1. (Continued)

```
/* Exit parameters
   None.
#7
unsigned byte offset;
                               /# Byte offset into the bit vector #/
if ((byte_offset = bitnum >> 3) > bv -> bv_bytes)
        return O;
                       /* Bitnum is "off the end" of the vector */
/* Set the appropriate bit in the vector. The byte offset
                                                                                  ee
   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>
return 1;
                       /* Indicate completion */
/# End of by_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 by #by:
unsigned bitnum;
/* Exit parameters
_wit
None.
*/
                                                                                  ff
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 */
/* 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. */
/* Exit Parameters
   NZ = one or more bits are set in the vector 0 \approx all bits are off
                                                                                  gg
* 1
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 */
```

Figure 11-1. (Continued)

```
з
                            /* Indicate vector is zero */
return Or
                                                                              gg
} /# End of by_nz #/
by 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
char *bits1, *bits2, *bits3;
                             /* Working pointers to bit vectors */
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
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;
bits2 = bv2 -> bv_bits;
                             /* Initialize working pointers */
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 */
} /* 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 #/
char #title;
                             /# Title for the display #/
struct _bv *bv;
                             /* Pointer to the bit vector */
```



```
/# Exit parameters
   None.
#/
                                 /* Working pointer */
char #bits;
                                  /* Count used for formatting display */
/* Count for processing bits in a byte */
unsigned byte_count;
unsigned bit_count;
char byte_value;
                                            /* Value to be displayed */
printf("\nBit Vector : %s",title);
                                            /* Display title */
bits = bv -> bv_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 */
                                                                                                 ii
                                            /* 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 */
        }
printf("
                   * > 2
                                                       /* Separator */
                         /* Update byte count */
        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 */ ______#
```

**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 (by_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 */
          31
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
#define DELETEF 19
#define GETDISK 25
                                 /# Search next #/
                                 /# Delete file #/
                                                                                                                         c
                                 /# Get default disk (currently logged in) #/
#define SETDMA 26
                                 /* Set DMA (Read/Write) Address */
#define GETDPARM 31
                                 /* Get disk parameter block address */
#define GETUSER 32
                                 /# Get current user number #/
Adefine SETUSER 32
                                 /# Set current user number #/
/# 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. */
                                 /* Console status #/
/* Console input #/
#define CONST
                      2
#define CONIN
                      з
#define CONOUT 4
                                 /# Console output #/
#define LIST5#define AUXOUT6#define AUXIN7
                                /* List output */
                                 /* Auxiliary output */
                               /# Auxiliary input #/
#define HOME
                                 /* Home disk */
                      A
                                                                                                                         d
#define SELDSK 9
                                 /# Select logical disk #/
                                /* Set track */
/* Set sector */
/* Set DMA address */
/* Disk read */
#define SETTRK 10
#define SETSEC 11
#define SETDMA 12
#define DREAD 13
#define DWRITE 14
#define LISTST 15
                                 /* Disk write */
                                 /* List status #/
#define SECTRN 16
#define AUXIST 17
#define AUXOST 18
                                 /* Sector translate */
                                 /* Auxiliary input status */
/# Auxiliary output status */
                                 /* "Private" entries in jump vector */
#define CIOINIT 19
                                 /# Specific character I/O initialization #/
#define SETDOG 20
#define CBGADDR 21
                                 /* Set watchdog timer */
                                 /* 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
#define TIME_SET 0x01
#define DATE_SET 0x02
                                           /* Time in ASCII */
/* Date, time flags */
/* This bit NZ means date has been set */
/* This bit NZ means time has been set */
                                                                                                                        e
#define CB_FIP 3
                                            /* Forced input pointer */
#define CB_SUM 4
                                            /# System start-up message #/
#define CB_CI 5
#define CB_CO 6
#define CB_AI 7
                                            /# Console input #/
                                            /# Console output #/
/# Auxiliary input #/
                                            /# Auxiliary output #/
#define CB_AO 8
```

Figure 11-2. (Continued)

#define CB_LI 9 #define CB_LO 10 /# List input #/ /* List output */ #define CB_DTA 11 /* Device table addresses */ #define CB_C1224 12 #define CB_RTCTR 13 /* Clock 12/24 format flag */ /* Real time clock tick rate (per second) */ #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 */ #define CB_BO_BRC 19 /* Device 0 baud rate constant */ e #define CB_D1_IS 20 /* Device 1 initialization stream */ #define CB_D1_BRC 21 /# Device 1 baud rate constant #/ #define CB_D2_IS 22 /* Device 2 initialization stream */ #define CB_D2_BRC 23 /* Device 2 baud rate constant */ /* Interrupt vector */ /* Long term config. block offset */ /* Long term config. block length */ #define CB IV 24 #define CB_LTCBO 25 #define CB_LTCBL 26 #define CB_PUBF 27 /* Public files flag */ #define CB_MCBUF 28 #define CB_POLLC 29 /* Multi-command buffer */ /* Polled console flag */ /* Device numbers and names for physical devices */ /* NOTE: Change these definitions for your computer system */ #define T_DEVN 0 /* Terminal */ f #define M_DEVN 1 /* Modem */ #define P_DEVN 2 /* 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. */ #define FK_LENGTH 16
#define FK_ENTRIES 18 /* Length of string (not including fk_term) */ /* Number of function key entries in table */ h struct _fkt /* Function key table */ char fk_input[FK_ILENGTH]; /* Lead-in character is not in table */ char fk_inputtFK_ILENGIHJ; /# Leau-in character string #/
char fk_term; /# Output character string #/
char fk_term; /# Safety terminating character #/ 3 2 /* 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 XON or ETX protocol. #/ i #define DT_ODTR 0x8004 /* Output DTR high to send (OR ed in) */ /* Output XON */ /* Output ETX/ACK */ #define DT_OXON 0x0008 #define DT_OETX 0x0010 /# Input RTS (OR-ed in) #/ /# Input XON #/ #define DT_IRTS 0x8040 #define DT_IXON 0x0080

Figure 11-2. (Continued)

```
#define ALLPROTO OxDC
                                        /* All protocols combined */
                                        /* Device table */
struct _dt
          char_dt_f1[14];
                                        /# Filler #/
                                                                                                                 i
          char dt_st1;
char dt_st2;
unsigned dt_f2;
unsigned dt_etxml;
                                        /* Status byte 1 --- has protocol flags */
/* Status byte 2 */
                                        /# Filler #/
                                        /* 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 */
                             /# Name not equal (and comparison ambiguous) #/
#define NAME_NE 3
/# Structure for standard CP/M file control block #/
                                        /* Define the overall length of an FCB */
#define FCBSIZE 36
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 */
          char fcb_ranrec[3];
                                        /* Record for random read/write */
          31
/* 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 #/
          short dp_uren,
short dp_end; /* NZ when at end us used to disk */
short dp_write; /* NZ to write current sector to disk */
struct _dir *dp_entry; /* Pointer to directory entry in buffer */
is a buffer EDIR_BSZ]; /* Directory sector buffer */
is a buffer EDIR_BSZ]; /* Directory sector buffer */
                                                                                                                 1
                                    /* Current logical disk */
          int dp_track;
                                       /* Start track */
                                      /* Start sector */
/* Number of directory entries */
          int dp_sector;
int dp_nument;
                                       /* Entries remaining to process */
          int dp_entrem;
                                       /* Number of sectors per track */
/* Number of allocation blocks per dir. entry */
          int dp_sptrk;
          int dp_nabpde;
unsigned dp_nab;
                                       /* Number of allocation blocks */
          int dp_absize;
                                        /* Allocation block size (in Kbytes) */
          31
/* 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. */
#define W_DIR
                   0
                              /* Writing directory */
                             /* Reading directory */
#define R_DIR 1
#define O_DIR 2
                              /* Opening directory */
```

Figure 11-2. (Continued)

```
/* Disk parameter block maintained by CPM */
struct _dpb
         í
         unsigned dpb_sptrk;
                                     /* Sectors per track */
/* Block shift */
         short dpb_bshift;
         short dpb_bmask;
                                     /* Block mask */
                                                                                                    n
         short dpb_emask;
                                     /* Extent mask */
         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_rabl;
unsigned dpb_diskca;
unsigned dpb_trkoff;
         3 :
/* Disk directory entry format */
struct _dir f
         char de_userno;
                                     /* User number or 0xE5 if free entry */
         char de_fname[11];
int de_extent;
                                     /* File name [8] and type [3] */
                                     /* Extent number of this entry */
                                     /* Number of 128-byte records used in last
         int de_recont;
                                                                                                    0
                                           allocation block */
         union
                                     /* Allocation blocks can be either */
                                     /* single or double bytes */
                  short de_short[16];
unsigned de_long[8];
                  } _dirab;
         3:
/* Disk request parameters for BIOS-level read/writes */
struct _drb
         short dr_disk;
                                     /* Logical disk A = 0, B = 1... */
                                                                                                    р
         unsigned dr_track;
                                     /* Track (for SETTRK) */
         unsigned dr_sector;
char *dr_buffer;
                                     /* Sector (for SETSEC) */
                                     /* Buffer address (for SETDMA) */
         ) :
/* Search control block used by directory scanning functions */
struct _scb
                                     /* User number(s) to match */
         short scb_userno;
         char scb_fname[11];
                                     /* File name and type */
         short scb_extent;
                                     /* Extent number */
                                                                                                    q
         char unused[19];
                                     /* Dummy bytes to make this look like
                                          a file control block #/
         short scb_length;
short scb_disk;
                                     /* Number of bytes to compare */
                                     /* 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 */
struct _ct
                           /* Define structure of code table */
                                                                                                    r
         unsigned _ct_code;
char *_ct_sp;
                                     /* Code value */
                                     /* String pointer */
         31
```

Figure 11-2. (Continued)

c

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:

char *ptr_to_date; /* declare date pointer*/
ptr_to_date = biosh(CB_DATE); /* get address */

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
     ş
    char *pdt[16];
                       /* Array of 16 pointers to device tables */
    } *ppdt;
                       /* Pointer to array of 16 pointers */
struct _dt *dt;
                       /* Pointer to device table */
ppdt = get_cba(CB_DTA);
                            /* Set pointer to array of pointers */
dt = ppdt -> pdt[device_no]; /* Set pointer to specified device
                                table */
if (!dt)
    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 --> -Totals-

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  $16 \times 18$  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:

```
٥
          1
              2
                  3
                                10 11 12 13 14 15 Used Free
A:
      1
          1
                                                          15 241
                                                          245 779
в:
     66 20
             74
                 50
                      з
    -- None ---
C:
                                                           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).

## **404** The CP/M Programmer's Handbook

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 bv_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 by_test function is effectively the reverse of by_set. It accesses the specified bit and returns its value to the calling program.

The bv_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 by_disp's output:

```
Bit Vector : Allocation Blocks in Use

0: 0000 0000 0001 1000 1000 0001 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 1101 1000 0001 1110

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 _feb feb;
                         /* 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 */
     3
```

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><u>erase(CR></u>

ERASE Version 1.0 02/23/83 (Library 1.0)

Usage :

ERASE (d:)file_name.typ

P3A><u>erase *.com<CR></u>

ERASE Version 1.0 02/23/83 (Library 1.0)

Searching for file(s) matching A:???????.COM.

Erase A:UNERASE .COM y/n? <u>n</u>

Erase A:TEMP1 .COM y/n? <u>y</u> <== Will be Erased!

Erase A:TEMP2 .COM y/n? <u>n</u>

Erase A:TEMP3 .COM y/n? <u>n</u>

Erase A:TEMP4 .COM y/n? <u>n</u>

Erasen A:ERASE .COM y/n? <u>n</u>

Erasen A:ERASE .COM y/n? <u>n</u>

Erasen A:TEMP4 .COM erased.

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>
                              /* Ambiguous name file control block */
/* Used for BDOS search functions */
struct _fcb amb_fcb;
struct _fcb fcb;
                                     /* Formatted for display: d:FILENAME.TYP */
char file_name[20];
                                     /* Current logical disk at start of program */
/* ERASE saves the FCB's of the all the
files that need to be erased in the
following array */
short cur_disk;
#define MAXERA 1024
struct _fcb era_fcb[MAXERA];
int ecount;
                                     /* Count of number of files to be erased */
int count;
                                     /* Used to access era_fcb during erasing */
main(argc,argv)
                         /# Argument count #/
/# Argument vector (pointer to an array of char. #/
short arge:
short argc;
char *argv[];
printf("\nERASE Version %s (Library %s)",VN,LIBVN);
                                     /* Check usage */
chk_use(argc);
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))
        printf(" <== Will be erased!");
                 /* 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 #/
        ъ
                 /* 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 */
                          /* Check usage */
chk use(argc)
/* 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 != 2)
         £
         printf("\nUsage 1");
         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"
/* LINERASE --
    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 _dirpb dir_pb;
struct _dir *dir_entry;
struct _scb scb;
struct _scb scb;
struct _dpb dpb;
struct _bv file_bv;
struct _bv file_bv;
struct _bv extents;
                                     /* Pointer to directory entry */
/* Search control block */
                                      /* SCB set up to match all files */
                                     /* 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 */
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 */
                                      /* Used to access the allocation block numbers
int count:
                                            in each directory entry #/
int users
                                      /* User in which the file is to be revived */
main(argc.argv)
                            /* Argument count */
short arect
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. \ast/
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(strsen(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 */
                                        /* No user number specified */
if (argc == 2)
          user = bdos(GETUSER, 0xFF);
                                               /* Get current user number */
else
ŧ
          user = atoi(argv[2]);
                                                /* Get specified number */
         if (user > 15)
                   printf("\nUser number can only be 0 - 15.");
                    exit();
1
/* 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.",
         .....do dire
argv[1]);
exit();
2
          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))
- E
          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 */
by 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)

1

```
/* 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]);
                   3
         3
1
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) */
/* 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. */
unsigned abno:
                           /* Allocation block number */
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();
         3
/* 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))
        OxE5), the file name and the type). */
if (comp_fname(scb,dir_entry) == NAME_EQ)
{
                  /* Compare user number (which can legitimately be
                  ++mcount;
                                             /* Update match count */
                  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];
                                    /* Assume 16 1-byte numbers */
                           else
                                    .
```

Figure 11-4. (Continued)

```
abno = dir_entry -> _dirab.de_short[count];
                             if (abno) by_set(by,abno); /# Set the bit #/
                   3
         1
return acount;
                             /* Return number of dir. entries matched */
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:
/* 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 i; 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.tvp
         difilename.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 #/
struct _scb #scb;
char #fname;
                            /# Pointer to search control block #/
                             /* Pointer to the file name */
/* User number to be matched */
short user;
                             /* Number of bytes to compare */
int length;
/* Exit parameters
   Disk number to be searched. (A = 1, B = 2...)
¥/
short disk_id;
                             /* Bisk 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 2 з 5 11 12 13 14 15 **Used Free** . . . A: 1 1 8 23 233 20 74 55 252 772 B: 66 з 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 2 з 4 5 11 12 13 14 15 Used Free 1 . . . A: 5 23 233 Bı 61 5 13 252 772 4 C: -- None --16 112 P3B>find H.com names<CR> FIND Version 1.0 02/11/83 (Library 1.0) Searching disk : B .COM 0/B:CLINK 0/B:CC .COM 2/B:CLIB 0/B:CC2 .COM . COM 1/B:CPM61 . COM 1/B: MOVCPM . COM 1/B:PSWX . COM 0/B:SUBMIT . COM 2/B:CDB . COM 1/B:CPM60 .COM O/B:DDT . COM O/B:EREMOTE .COM 0/B: SPEEDSP . COM O/B:PIP . COM 0/B:PROTOSP . COM O/B:RX . COM O/BITXA . COM . COM 0/B:EPRIV . COM 0/B:EPUB O/B:WSC . COM 0/B:X . COM 0/B:CRCK O/B:XSUB .COM . COM 0/B:DU . COM . COM 0/B: REMOTE .COM 0/B: QERA O/B:FINDALL .COM 0/B: MOVEF . COM 0/B:LOCAL .COM O/B: DUMP .COM 0/B: MRESET .COM 0/B:ELOCAL .COM 0/B: FDUMP 0/B: PUTCPMF5. COM O/B: TEST **O/B:INVIS** . COM .COM .COM . COM . COM . COM 0/B:L80 0/B:LIST O/B: PUB . COM 0/B:LOAD O/B: MAC . COM 0/B:SCRUB . COM 0/B:RXA . COM O/B:STAT .COM 0/B: TX . COM 0/B: ERASEALL.COM O/B:WM .COM 0/B:MSFORMAT.COM . COM 0/B: STATUS 0/B:UNERA . COM **O/B:MSINIT** . COM .COM 0/B:VIS .COM . COM .COM 0/B:WSVTIP O/B:XD O/B:NEWVE O/B: DDUMF . COM . COM O/B: FORMATMA. COM 0/B:PRIV 0/B: FCOMP . COM 0/B: DDUMPA .COM 0/B: PUTSYS1C. COM 0/B: DSTAT .COM O/B:DDUMPNI .COM .COM O/B:ASM 2/BICDBTEST .COM 0/B: OLDSYS . COM 0/B:E . COM 2/B:F/C .COM 3/B: ERASE 3/B: FUNKEY .COM .COM 3/B:DATE .COM 3/B:FIND .COM Press Space Bar to continue.... . COM 3/B: SPACE . COM 3/B: UNERASE . COM 3/B: MAKE . COM 3/B: MOVE 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. */ #include <LIBRARY.H>

 struct _dirpb dir_pb;
 /# Directory management parameter block #/

 struct _dir #dir_entry;
 /# Pointer to directory entry (somewhere in dir_pb) #/

 struct _scb scb;
 /# Search control block #/

 char file_name[20];
 /# 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 #/
int mcount:
                                     /* Match count (no. of file names matched) */
int dmcount;
                                     /* Per disk match count */
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. #/
                         18]; /* Disk A -> P, users 0 -> 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)
short arger
                         /# Argument count #/
                         /* Argument vector (pointer to an array of chars.) */
char #argv[];
/* Reset disk map */
dm_clr(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 As */
scb_scb_disk < 16; /* Until logical disk P: */
scb.scb_disk++) /* Move to next logical disk */
for (scb.scb_disk = 0;
£
         /* Check if current disk has been selected for search */
if (!(scb.scb_adisks & (1 << scb.scb_disk)))
continue; /* No,so bypass this disk */
dir_pb.dp_disk = scb.scb_disk; /* Set to disk to be searched*/
decount = 0;
                                     /* Reset disk matched count #/
         >_flag) /* If file names are to be displayed */
putchar('\n'); /* Move to column 1 */
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]++;
         else
                  /* 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 */
```

```
14 (
               (dir_entry -> de_userno != 0xE5) &&
(dir_entry -> de_extent == 0) &&
(comp_fname(scb,dir_entry) == NAME_EQ)
             )
                    £
                                        /* Update matched counts */
                    mcount++;
                    dmcount++;
                                       /* Per disk count */
                    if (map_flag)
                                       /* Check map option */
                              4
                                        /* Update disk map */
                              disk_map[scb.scb_disk][dir_entry -> de_userno]++;
                              1
                    else
                                        /* Bisplay names */
                              conv_dfname(scb.scb_disk,dir_entry,file_name);
printf("%s ",file_name);
                              /* Check if need to start new line */
if (!(dmcount % 4))
                                        £
                                        Putchar((\n/))
                                                 if (++1count > 18)
                                                            lcount = 0;
printf("\nPress Space Bar to continue....");
                                                            setchar():
                                                           putchar ((\n');
                                                  3
                                        3
                   }
} /# End of directory #/
          } /# All disks searched #/
if (map_flag)
printf("\n
                                Numbers show files in each user number.");
printf("\n
                                                                                                Dir. Entries");
                                             --- User Numbers ---
dm_disp(disk_map,scb.scb_adisks);
                                                 /* Display disk map */
if (meount == 0)
printf("\n --- File Not Found --- ");
bdos(SETDISK,cur_disk); /* Reset to current disk */
                             /* check usage */
chk use(argc)
/* This function checks that the correct number of
parameters has been specified, outputting instructions
   if not.
¥7
/* Entry parameter */
int argc; /* Count of the number of arguments on the command line */
int arge:
4
/* 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 | i | argc > 3)
printf("\nUsage :");
printf("\n\\FIND 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.");
exitO;
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)
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
               3
                    4
        1
A: 18 202
               38
                                                           258 1196
B: 692 432 656 548 36
                                                          2364 996
C: 140
                                                           140 204
```

```
#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;
                                     /* Pointer to directory entry */
/* Search control block */
struct _dir *dir_entry;
struct _scb scb;
struct _dpb dpb;
                                      /* CP/M's disk parameter block */
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 */
/* 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.
¥/
unsigned disk_map[16][18];
                                    /* Disk A -> P, users O -> 15, free, used */
                                    /* "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 #/
char #argv[];
                            /* Argument vector (pointer to an array of chars.) */
```

Figure 11-6. SPACE.C, a utility that displays how much disk storage is used or available

```
printf("\nSPACE Version %s (Library %s)",VN,LIBVN);
chk_use(argc); /* Check usage */
cur_disk = bdos(GETDISK); /* Get current default disk */
chk_use(argc);
cur_disk = bdos(GETDISK);
dm_clr(disk_map);
                                 /* Reset disk map */
                                 /* Special version : set disks,
ssetscb(scb,argv[1]);
                                    name, type */
                                 /* Starting with logical disk A: */
for (scb.scb_disk = 0;
     scb.scb_disk < 16;</pre>
                                /* Until logical disk P: */
/* Move to next logical disk */
     scb.scb_disk++)
        Ŧ
        /* Check if current disk has been selected for search */
if (!(scb.scb_adisks & (1 << scb.scb_disk)))</pre>
                continue:
                                /* No. so bypass this disk #/
        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)
                                       /* Bypass inactive entries */
                        continue;
                for (count = 0)
                                         /* Start with the first alloc. block */
                     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);
                                 /* Assume 16 1-byte numbers */
                        else
                                 3
                                 /* All allocation blocks processed */
                з
                         /* 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 */
                /* Free space = (# of alloc. blks * # of kbyte per blk)
                 used Kbytes
        - (dir_pb.dp_nument >> 5);
/* All disks processed */
                                                 /* Same as / 32 */
        ъ
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;
/* Exit parameters
    None.
*/
ŧ
int disk;
                                /* Disk number currently being checked */
                               /* Bit map for active disks */
unsigned adisks;
adisks = 0;
                                /* Assume no disks to search */
if (*ldisks)
                               /* Some values specified */
           if (*1disks == '*')
                                         /* Check if "all disks" */
                     Ŧ
                     adisks = 0xFFFF:
                                                    /* Set all hits */
          else
                                          /* Set specific disks */
                     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');
                               ++ldisks:
                                                    /* Move to next character */
                     3
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;
      disk++)
                               /* Use next disk */
          if ( !((1 << disk) & adisks))
                     continue;
                                                    /* Avoid selecting unspecified disks */
```

Figure 11-6. (Continued)

```
/* Make BIOS SELDSK call */
          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) * OxFFFF);
          1
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 */
int arge:
                 /* Count of the number of arguments on the command line */
s.
          /* 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();
} /* 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 difilename.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
B: SPACE
            COM : UNERASE COM : MAKE
                                          COM : MOVE
                                                           COM
            COM : ASSIGN COM : SPEED
                                          COM : PROTOCOL COM
B: TIME
P3B>move <u>*.com 0</u> <u>names<CR></u>
MOVE Version 1.0 02/10/83 (Library 1.0)
Moving file(s) 3/B:??????.COM -> User 0.
```

0/B:ERASE .COM O/B:FUNKEY .COM O/B:DATE .COM O/B:FIND .COM O/B:UNERASE .COM O/B:MAKE .COM O/B:MOVE . COM 0/B:SPACE . COM .COM 0/B:ASSIGN .COM 0/B:SPEED .COM 0/B:PROTOCOL.COM 0/B:TIME P3B>user OKCR> POB>dir B: ERASE COM : FUNKEY COM : DATE COM : FIND COM B: SPACE COM : UNERASE COM : MAKE 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;
                                    /* Pointer to directory entry */
                                   /* Search control block */
struct _scb scb;
#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 */
short name_flag;
                                   /* NZ to display names of files moved */
short cur_disk;
                                   /* Current logical disk at start of program */
int from_user;
int to_user;
                                   /* User number from which to move files */
/* User number to which files will be moved */
int meount;
                                  /* 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 */
short arge:
char *argv[];
                         /* Argument vector (pointer to an array of chars.) */
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)
         Ŧ
         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]);
         3
else
                  /* No source user specified */
         ÷
```

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.");
          exitOr
          /* 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: */
      scb.scb_disk < 16;
                                          /* 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)))
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);
          lcount++:
                                          /* Update line count */
          /# Reset disk matched count #/
          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++:
                               decount++:
                               if (name_flag) /* Check map option */
                                          conv_dfname(scb.scb_disk,dir_entry,file_name);
                                                    printf("%s ",file_name);
                                         /* Check if need to start new line */
if (!(dmcount % 4))
                                                    £
                                                    putchar((\n');
                                                    if (++1count > 18)
```

Figure 11-7. (Continued)

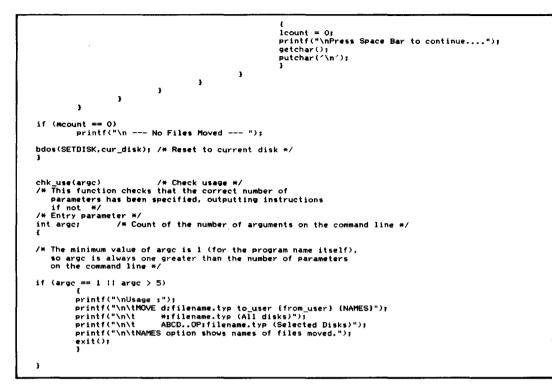


Figure 11-7. (Continued)

### **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)
```

# 430 The CP/M Programmer's Handbook

```
mcount++;
                                                   /* Update matched counts */
                               if (invisible)
                                         { /* Set ms bits */
dir_entry -> de_fname[8] != 0x80;
                                         dir_entry -> de_fname[9] != 0x80;
                               ....
                                         /# 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)
                                        conv_dfname(scb.scb_disk,dir_entry,file_name);
printf("\n\t%s made %s in User %d.",
                                                  file_name, operation, to user);
                                         ٦
                               3
                    } /* All directory entries processed */
/* All disks processed */
          3
if (meount == 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 #/
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 == 3 || argc == 4)
          returns
....
          £
          printf("\nUsage :"):
          printf("\n\tMAKE difilename.typ INVISIBLE (NAMES)");
          printf("\n\t
                                                 VISIBLE");
          printf("\n\t
                                #:filename.typ (All disks)");
          printf("\n\t ABCD..OPifilename.typ (Selected Disks)");
printf("\n\tNAMES option shows names of files processed.");
          exit();
}
```

#### 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. Usage is : SPEED physical-device baud-rate, or 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"

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
 #define B1200
                     0x37
                                          /* 1200 baud */
                                          /* 2400 baud */
#define B2400
#define B4800
                     0x34
                                           /* 4800 baud */
                     0x3C
#define B9600 0x3E
#define B19200 0x3F
                                          /* 9600 baud */
                                           /* 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 */
                                /* set up the code tables for this program */
setup()
           /* Initialize the physical device table */
ct_init(ct_pdev[3],T_DEVN,PN_T); /* Terminal */
ct_init(ct_pdev[3],P_DEVN,PN_P); /* Printer */
ct_init(ct_pdev[3],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_brt31,B4800, #4800");
ct_init(ct_brt51,B4800, #4800");
ct_init(ct_brt51,B9200, "9600");
ct_init(ct_brt61,B19200, "19200");
ct_init(ct_brt73,CT_SNF,"*"); /* Terminator */
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 */
char *ppdev;
unsigned retval;
                                                     /* Return value */
/* Get code for ASCII string */
           4
           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 */
```

Figure 11-9. (Continued)

```
char *pbaud;
                                                        /* Pointer to character string */
 retval = ct_parc(ct_br,pbaud); /* Return value */
if (retval == CT_SNF) /* If string not found that is the string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string string 
                    Ŧ
                   printf("\n\007Baud Rate '%s' is invalid or ambiguous.",
                                     pbaud);
                   exit();
                   2
                                                       /# Return code #/
 return retval:
 3
                                                      /# Set the baud rate of the specified device */
 set_baud(pdevc,baudc)
                                                        /* Physical device code */
 int odevct
 short baude;
                                                        /# 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_re = baude;
                   /# Then call the BIOS initialization routine #/
bios(CIOINIT,pdevc);
 show_baud()
                                                      /# Show current baud rate #/
 int pdevn;
                                                       /* Physical device number */
 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 */
 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
3
chk_use(argc)
                                                      /* Check correct usage */
int arge:
                                                      /* 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>protoco1<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 m o e 128<CR>
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_dpev[MAXPDEV + 2]; /* Physical device table */
struct _ct ct_io[3]; /* Input, output */
           /* Parameters on the command line */
#define PDEV argvL11  /* Physical device */
#define IO argv[2]  /* Input/output */
#define PROTO argv[3]  /* Protocol */
#define PROTOL argv[4]  /* Protocol message length */
main(argc, argv)
int arges
char #argv[];
f
                  /* Display sign-on message */
printf(VN);
setup();
                     /* Set up code tables */
/# Check if request to show current settings #/
if (usstrcmp("SHOW",argv[1]))
                               /# No -- assume a set is required #/
          ŧ
           get_proto(get_io(IO), PROTO),
                     PROTOL >;
                                         /# Protocol message length #/
show_proto();
3 /* end of program */
setup()
                                /# Set up the code tables for this program #/
           /* Initialize the physical device table */
ct_init(ct_pdevE0],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_pdevE3],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_oproto[2], 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_oprotoE31,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[1],DT_ODTR, "Output DTR");
ct_init(ct_dproto[1],DT_ODTR, "Output DTR");
ct_init(ct_dproto[1],DT_OXON, "Output XON");
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_SNE "#");
ct_init(ct_dproto[5],CT_SNF,"*");
ъ
unsigned
                             /* Get physical device */
get pdev(ppdev)
/* This function returns the physical device code
specified by the user in the command line. */
char *ppdev; /* Pointer to character string */
                              /* Return value */
unsigned retval:
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 code */
return retval;
1
unsigned
                            /* Get input/output parameter */
/* Pointer to character string */
get_io(pio)
char *pio;
unsigned retval;
                                         /* Return value */
÷
          printf("\n\007Input/Output direction '%s' is invalid or ambiguous.",
          pio);
printf("\nLegal values are : ");
          ct_disps(ct_io);
                                       /* Display all values */
          exit();
          1
return retval;
                                        /# Return code */
3
unsigned
get_proto(output,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();
```

Figure 11-10. (Continued)

```
3
                                    /* INPUT specified */
else
         £
         /* Get code for ASCII string */
retval = ct_parc(ct_iproto,pproto);
                                           /# If string not found #/
         if (retval == CT_SNF)
                  £
                  printf("\n\007Input Protocol '%s' is invalid or ambiguous.",
         pproto):
                  printf("\nLegal Input Protocols are ; ");
                  ct_disps(ct_iproto); /* Display valid protocols */
                  exit();
         . 2
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 #/
3 1
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 (Idt)
                           /* Check if pointer in array is valid */
         £
         printf("\nError -- Array of Device Table Addresses is not set for device #%d.",
                 pdevc);
         exit();
if (protoc & Ox8000)
                          /* 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 & Ox7F) == DT_OETX) /* If ETX/ACK, check for message
                                      length */
         if (isdigit(*pplength))
                                             /* Check if length present */
                  ŧ
                           /* Convert length to binary and set device
                  table field. */
dt -> dt_etxml = atoi(pplength);
         1
3
show_proto()
                          /* Show the current protocol settings #/
struct _ppdt
char *pdt[16];
                          /* Array of 16 pointers to the device tables */
3 :
struct _ppdt #ppdt;
struct _dt #dt;
int pdevc;
                                    /* Pointer to the device table array */
                                    /* Pointer to a device table */
/* Physical device code */
struct _ct #dproto;
                                    /# Pointer to display protocols #/
ppdt = get_cba(CB_DTA); /* Set pointer to array of pointers */
         /# For all physical devices #/
```

Figure 11-10. (Continued)

```
for (pdevc = 0; pdevc <= MAXPDEV; pdevc++)
        ŧ
        /* 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;
                        ъ
                        /* Set pointer to display protocol table */
                dproto = ct_dproto;
                while (dproto -> _ct_code != CT_SNF)
                        £
                                 /* Check if protocol bit set */
                        if (dproto -> _ct_code & dt -> dt_st1)
{ /* Display protocol */
                                 printf("%s ",dproto -> _ct_sp);
                                        /* Move to next entry */
                         ++dproto:
                         /* Check if ETX/ACK protocol and
                dt -> dt_etxml);
                3
        3
3
chk_use(argc)
                        /* Check for correct usage */
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();
        ъ
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><u>assign<CR></u>
ASSIGN Vn 1.0 02/17/83
ASSIGN sets the Input/Output redirection.
ASSIGN logical-device INPUT physical-device
ASSIGN logical-device OUTPUT physical-devi {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 e<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_pdevEMAXPDEV + 2];
                                           /* Physical device table */
/* Names of logical devices */
#define LN_C "CONSOLE"
#define LN_A "AUXILIARY"
#define LN_A
#define LN_L
                 "LIST"
                                  /* Logical device table */
struct _ct_ct_ldev[4];
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 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 #/
```

# Figure 11-11. ASSIGN.C, a utility that assigns a logical device's input and output to two physical devices

440 The CP/M Programmer's Handbook

```
/* NOTE : the number of physical devices to
                       process is given by argc - 3 */
          set_assign(get_ldev(LDEV),get_io(IO),argc = 3,argv);
show_assign();
1
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[2],2,PN_M);
                                      /* Modem */
ct_init(ct_pdev[3],CT_SNF, "*"); /* Terminator */
          /* Initialize the logical device table */
ct_init(ct_idevI0].0,LM_C); /# Terminal */
ct_init(ct_idevI1].1,LM_A); /# Auxiliary */
ct_init(ct_idevI2].2,LN_L); /# List */
ct_init(ct_idevI3].CT_SNF,"*"); /# Terminator */
/* Terminator */
ł
unsigned
/* Get logical device */
/* This function returns the logical device code
    specified by the user in the command line. */

                            /* Pointer to character string */
char *pldev;
ŧ
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_io(pio)
                            /* Get input/output parameter */
                            /* Pointer to character string */
char *pio:
unsigned retval;
                                       /* Return value */
retval = ct_parc(ct_io,pio); /* Get code for ASCII string */
if (retval == CT_SNF) /* If string not found */
         £
         printf("\n\007Input/output direction '%s' is invalid or ambiguous.",
         pio);
printf("\nLegal values are : ");
         ct_disps(ct_io); /* Display all values */
         exit();
return retval;
                                      /* Return code */
set_assign(ldevc,output,argc,argv)
int_ldevc; /*
                                                /* Set assignment (I/B redirection) */
                                      /* Logical device code */
                                       /* I/O redirection code */
int output;
                                       /* count of arguments to process */
int argc;
char *argv[];
                                       /* Replica of parameter to main function */
unsigned *redir;
                                       /* Pointer to redirection word */
/* Physical device code */
int pdevc;
unsigned rd_val;
                                       /* Redirection value */
          /* 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);
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 #/
                   exitOr
                    1
                    /* Repeat this loop for as long as there are
more parameters (for output only) #/
          .1...
                    /* Build new redirection value by OR ing in
                   a one-bit shifted left pdevc places. #/
rd_val != (1 << pdevc);</pre>
          } while (--argc && output);
                            /* Set the value into the config. block */
#redir = rd_val;
1
                                      /# Show current baud rate #/
show_assign()
int rd_code;
                                       /* Redirection code for get_cba */
int ldevn;
                                       /* Logical device number */
                                       /# Physical device number #/
int pdevn;
                                       /# Redirection value #/
unsigned rd_val;
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 area
         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_LO; 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. */
           printf("\n\t%s %s is assigned to - "
                      ct_strc(ct_ldev,(rd_code - CB_CI) >> 1),
ct_strc(ct_io,((rd_code & 0x01) ^ 1)));
                       /* 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) );
                      3
           3
3
chk_use(argc)
                                 /* Check for correct usage */
int arge;
                                 /* Argument count on commmand line */
Ŧ
if (argc == 1)
           ÷.
          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;
                             /* Variables to hold month, day, year */
/* Match count of numeric values entered */
int mm,dd,yy;
int mcount;
                             /* Count used to add leading O's to date */
int counts
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 #/
                              /* Check if help requested (or needed) */
if (argc != 2)
          show_use();
                             /* Display correct usage and exit */
if (usstromp("SHOW",argv[1])) /* Check if not SHOW option */
                    /* Convert specified time into month, day, year */
          mcount = sscanf(argv[i], "%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);
                                       /# Display correct usage and exit #/
                    show_use();
          if (dd > 31 || dd < 1)
                    printf("\nDay = %d is illegal.",dd);
                    show_use(); /* Display correct usage and exit */
                    ł
          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;
date[9] = '\0';
                                  /* Terminate with line feed */
                                        /* New string terminator */
                     /* Change " 1/ 2/ 3" into "01/02/03" */
           for (count = 0; count < 7; count+=3)
                     if (datefcount] == < <>
                              date[count] = '0';
                     ъ
                     /* Turn flag on to indicate that user has set date */
           #date_flag != DATE_SET;
printf("\n\tCurrent Date is %s",date);
show_use()
                              /* Display correct usage and exit */
printf("\nDATE sets the system date. Usage is :");
print("\n\tDATE mm/dd/y");
print("\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>
char *time; /* Pointer to the time in the config. block */
char *time_set; /* Pointer to the time set flag */
int hh.mm.ss; /* Variables to hold hours, minutes, seconds */
                            /* Match count of numeric values entered #/
/* Count used to add leading zeros to time #/
int mcount;
int count;
main(argc,argv)
int arec:
char #argv[];
£
printf(VN);
                             /* Display sign-on message */
time = get_cba(CB_TIMEA); /* Set pointer to time */
time_flag = get_cba(CB_DTFLAGS); /* Set pointer to the time-set flag */
hh = mm = ss = 0; /# Initialize the time if seconds or
                               minutes are not specified */
          if (arge != 2)
if (usstromp("SHOW", argvEi])) /* Check if not SHOW option */
          £
                   /* Convert time into hours, minutes, seconds */
          mcount = sscanf(argv[1], "%d:%d:%d",&hh,&mm,&ss);
                   bunt) /* Input not numeric */
show_use(); /* Display correct usage and exit */
          if (imeount)
          if (hh > 12)
                                      /* Check valid hours, minutes, seconds */
                   £
                   printf("\n\007Hours = %d is illegal.", hh);
                    show_use(); /# Display correct usage and exit #/
                    ъ
```

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 #/
               ł
       if (ss > 59)
               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':
               ъ
                /* 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 hhimm(iss)}");
printf("\n\tTIME SHOW (to display current time)\n");
exitOr
ъ
```

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.)
                   C
                           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>
                                 /* Function key number to be programmed */
int fnum;
                                 /* String for function key */
/* Pointer to function key table */
char fstring[20]:
struct _fkt *pfk;
main(argc,argv)
int argc;
char #argv[];
5
if (argc == 1 | | argc > 3)
        show_use();
pfk = get_cba(CB_FKT); /* Set pointer to function key table */
if (usstrcmp("SHOW",argv[1]))
         if (!isdigit(argv[1][0]))
                 ş
                 printf("\n\007'%s' is an illegal function key.".
                         arev[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();
                 ъ
        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);
        ъ
else
                /* SHOW function specified */
        6
        printf(VN);
                                  /* Display sign-on message */
        show_fun();
        3
3
get_fs(string)
                         /* Get function string from command tail */
char string[];
                         /* Pointer to character string */
                         /* Pointer to command tail */
/* Count of TOTAL characters in command tail */
/* String length */
char *tail;
short tcount;
int slen:
tail = 0x80;
                         /* Command line is in memory at 0080H */
tcount = *tail++;
                         /* Set TOTAL count of characters in command tail */
slen = 0;
                         /* Initialize string length */
while(tcount--)
                         /* For all characters in the command tail */
         ÷
        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 */
         5
         printf("\n\007No leading quotes found.");
         show_use() #
         1
                          /# Adjust tail count #/
++trounts
                          /* For all remaining characters in tail */
while(tcount---)
         if (*tail == '"')
                  £
                  string[slen] = '\0';  /* Add terminator */
                                 /* Exit from loop */
                 breaks
                  3
         string[slen] = #tail++; /# Move char. from tail into string #/
         if (string[slen] == '<')
                 string[slen] = OxOA;
         ++slen:
        3
if (Itcount)
                          /* No terminating quotes found */
         ŧ
        printf("\n\007No trailing quotes found.");
         show_use();
return slen;
                          /* Return string length */
ъ
show_fun()
                          /* Bisplay settings for all function keys */
struct _fkt *pfkt;
                          /* Local pointer to function keys */
                          /* Count to access function keys */
/* Pointer to "<" character (LINE FEED) */</pre>
int count;
char #lf;
pfkt = get_cba(CB_FKT); /* Set pointer to function key table */
for (count = 0; count <= FK_ENTRIES; count++)
         £
        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"))
                          *1f = '<':
                          2
                 printf("\n\tKey #%d = '%s'",count,fstring);
        ++pfkt;
                          /* Move to next entry */
        ъ
3
show_use()
£
printf("\nFUNKEY sets a specific function key string.");
printf("\n\tFUNKEY key-number \042string to be programmed<\042 ");
printf("\n\t (Note : '<' is changed to line feed.)");</pre>
                      (Note : '<' is changed to line feed.)");
printf("\n\t
                                  key-number is from 0 to %d.)",
                         ¢
FK_ENTRIES-1);
printf("\n\t
                         C
                                  string can be up to %d chars.)",
        FK LENGTH);
printf("\n\tFUNKEY SHOW
                                (displays settings for all keys)");
exitOr
3
```

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

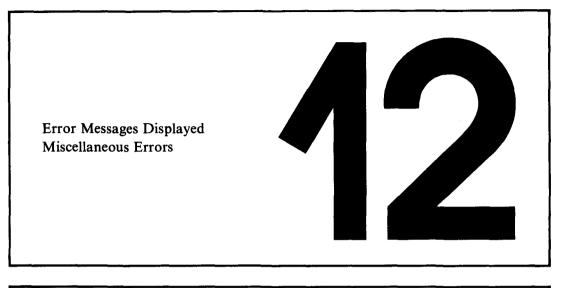
This would load the long term configuration block from a previously saved image.

#### 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>stat *.*<cr>

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 nonexis-

tent 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.

- 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.

Meaning

# CANNOT CLOSE, READ/ONLY? (SUBMIT)

х #

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 reformat 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^Z]" 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 string Z ]" 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.

1				$\begin{array}{c} b7 \longrightarrow \\ b6 \longrightarrow \\ b5 \longrightarrow \end{array}$	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	Row Col.	0 1	1	2	3	4	5	6	7	
0	0	0	0	0	NUL	DLE	SP	0	@	Р	`	р	
0	0	0	1	1	SOH	DC1	!	1	Α	Q	a	q	
0	0	1	0	2	STX	DC2	"	2	В	R	b	r	
0	0	1	1	3	ETX	DC3	#	3	С	S	с	s	
0	1	0	0	4	EOT	DC4	\$	4	D	Т	d	t	
0	1	0	1	5	ENQ	NAK	%	5	Ε	U	e	u	
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v	
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w	
	0	0	0	8	BS	CAN	(	8	Н	X	h	x	
1	0	0	1	9	HT	EM	)	9	I	Y	i	У	
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z	
	0	1	1	11	VT	ESC	+	;	K		k	{	
1 1	1	0	0	12	FF	FS	•	<	L	$\left  \right\rangle$	1		
1 1	1	0	1	13	CR	GS	-	=	М	] ]	m	}	
	1	1	0	14	SO	RS		>	Ν	^	n	~	
	1	1	1	15	SI	US	/	?	0	-	0	DEL	
NUL	Null	1			DCI	Dev	vice cor	ntrol 1					
SOH	Start of heading							Dev	Device control 2				
STX	Start of text							Dev	Device control 3				
ETX									Device control 4				
EOT End of transmission								Neg	Negative acknowledge				
ENQ Enquiry									Synchronous idle				
ACK Acknowledge									End of transmission block				
BEL Bell or alarm								Car	Cancel				
BS	Backspace							Enc	End of medium				
НТ	Horizontal tabulation							Sut	Substitute				
LF								Esc	Escape				
VT	Vertical tabulation							File	File separator				
FF	For	Form feed							Group separator				
CR	Carriage return								Record separator				
so	Shift out								Unit separator				
SI	Shift in								Space				
DLE	Data link escape							Del	Delete				

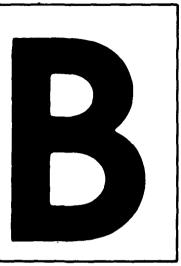
Table A-1. ASCII Character Codes

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	?
10	001 0000	DLE	40	100 0000	
11	001 0001	DC1	41	100 0001	Α
12	001 0010	DC2	42	100 0010	В
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	1
1A	001 1010	SUB	4A	100 1010	J
1B	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	R
23	010 0011	#	53	101 0011	S
24	010 0100	\$	54	101 0100	Т
25	010 0101	%	55	101 0101	U
26	010 0110	&	56	101 0110	v
27	010 0111	,	57	101 0111	W
28	010 1000	(	58	101 1000	X
29	010 1001	)	59	101 1001	Y
2A	010 1010	+	5A	101 1010	Z
2B	010 1011	+	5B	101 1011	]
2C	010 1100	,	5C	101 1100	1
2D	010 1101	-	5D	101 1101	]
2E	010 1110		5E	101 1110	۸
2F	010 1111	/	5F	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	j	7A	111 1010	z
6B	110 1011	k	7B	111 1011	{
6C	110 1100	1	7C	111 1100	Ì
6D	110 1101	m	7D	111 1101	3
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.

## **DDT 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.
Hx,y	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^2	<b>Cinsertstring</b> ^ <b>Zendstring</b> ^ <b>Z</b> 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.		
+/— <b>nK</b>	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.
- **Rilename** < 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	<ul> <li>Type lines. Displays "n" lines.</li> <li>+ displays the "n" lines after the CP</li> <li>- displays the "n" lines before the CP.</li> </ul>
	If the CP is not at the beginning of a line 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.
+/ <b>-</b> 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 " $+/-V$ ").
+/ <b>n</b>	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**<**CI**> 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 old 2.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

DnDeletes all characters after the "n"th column.EEchoes the copying to the console as it is being performed.FRemoves form feed characters during transfer.GnDirects PIP to copy a file from user area "n."
F Removes form feed characters during transfer.
•
Gn Directs PIP to copy a file from user area "n."
H Checks for proper Intel Hex File format.
I Ignores any :00 records in Intel Hex File transfers.
L Translates uppercase letters to lowercase.
N Adds a line number to each line transferred.
O 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 ^A 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.
- **STAI 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.

- 478 The CP/M Programmer's Handbook
  - **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**

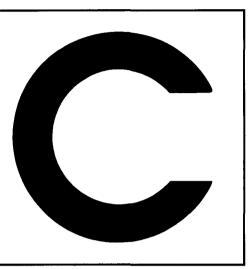
**TYPE x:filename.typ**<**cr>** Displays the contents of file filename.typ from drive x: on the console.

## **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	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	Evenless 44
No.	Name	Parameter(s)	Parameter(s)	Explanation
02	CONSOLE 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 program until a character is received.
04	PUNCH OUTPUT	E = ASCII character	None	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.
specif	fically returne	ed in the A regis	ter. Some man	te H register in the A register if nothing is to be sufacturers, specifically Microsoft, make use of ation between the H and A registers.

such information to reduce movement of information between the H and A registers.

## Appendix C: Summary of BDOS Calls 481

 Table C-1.
 (Continued)

Function		Entry	Exit	Explanation		
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 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.						

BDOS Function codes (continued)

#### RIOS (continued)

## 490 The CP/M Programmer's Handbook

#### **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:

#### 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

**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, 35 Number 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: Layout of standard CP/M diskette, 37

Diskette format: Concepts, 9

**Display 5-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_ctr: 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 DPR: See Disk Parameter Block DPH: 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

DO: Suggestion for utility program, 448 DPB: See Disk Parameter Block

**DPH:** See Disk Parameter Header

## E

ED: Editor, manual, 6

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 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 н 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

Heath/Zenith: Special version of CP/M, 55 Host Buffer: In the BIOS, 152

Host sector size: In the BIOS, 152

## I

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-J, 48 List Device Errors: Problems with BDOS Function 5, 78 Loading CP/M: Overview, 11 Loading programs: Via the CCP, 54 Loadsys: 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 Ο 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

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: See 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

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 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 Usstremp: Uppercase substring compare, 373 Ustremp: 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

#### 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

## X

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

**ZSID:** Z80 Symbolic Interactive Debugger, 185, 350 . 

## THE PROGRAMMER'S CP/M HANDBOOK

The most comprehensive source of information on CP/M[®] published to date, especially for the CP/M system programmer. **The Programmer's CP/M[®] Handbook** contains descriptions of CP/M's internal structure, the CP/M file system, the Console Command Processor, Basic Disk Operating System, Basic Input/Output System, utility programs, and much more. Illustrated examples are presented with detailed comments on source code for customizing individual CP/M systems, and utility programs are shown with annotated C language source code.

**The Programmer's CP/M[®] Handbook** is a valuable and long-awaited tool for the serious programmer.

Andy Johnson-Laird has more than twenty years of experience in the computer field and was engaged by Digital Research, Inc., the creators of CP/M, to customize CP/M-86[®] for the IBM PC. Mr. Johnson-Laird is the president and founder of Johnson-Laird, Inc. and Control-C Software, Inc., two Portland, Oregon, software houses.

• CP/M and CP/M-86 are registered trademarks of Digital Research, Inc.

• IBM is a registered trademark of IBM.



ISBN 0-88134-103-7