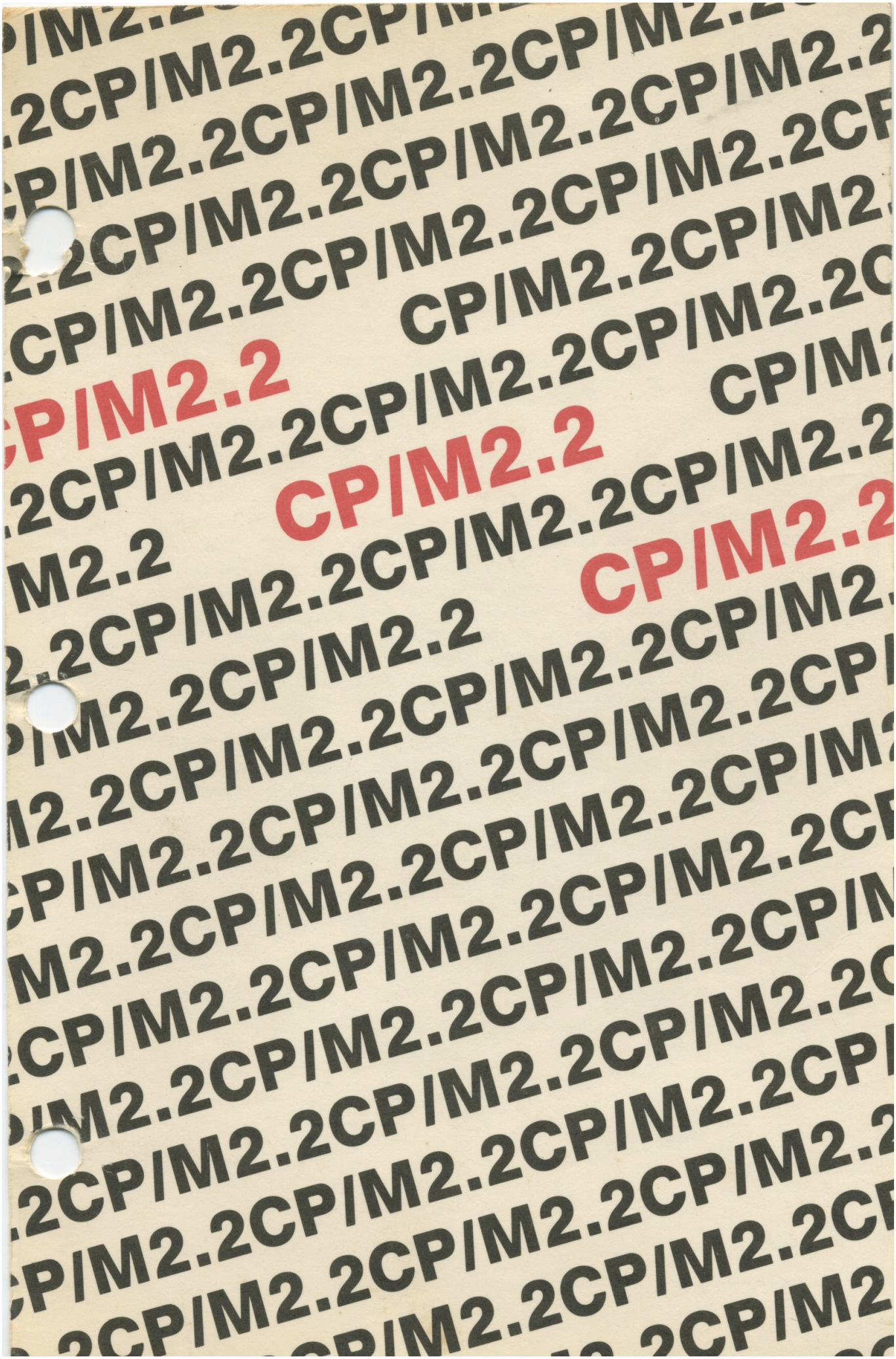

CP/M2.2
CP/M2.2
CP/M2.2

COLECO

ACKNOWLEDGEMENTS

Writer, PART A and PART B

Chopeta Lyons

Contributing Editors, PART C and PART D

Barbara Spear
Chopeta Lyons

Technical Consultants and Contributors

Mark Callahan
Steven Munnings
Phil Waller
Lem Miller
Teri Zalegowski
Ian Spence
Bob Reid
John Reese

014415

Printed in the U.S.A.

TABLE OF CONTENTS

Section 3: CP/M Assembler

Section 4: CP/M Dynamic Debugging Tool

INDEX

90-DAY LIMITED WARRANTY

Coleco warrants to the original consumer purchaser in the United States of America that the physical components of this Medium ("medium" refers to the digital data pack or disk) will be free of defects in the material and workmanship for 90 days from the date of purchase under normal in-house use.

Coleco's sole and exclusive liability for defects in material and workmanship of the Medium shall be limited to repair or replacement at an authorized Coleco Service Center. This warranty does not obligate Coleco to bear the cost of transportation charges in connection with the repair or replacement of defective parts.

This warranty is invalid if the damage or defect is caused by accident, act of God, consumer abuse, unauthorized alteration or repair, vandalism or misuse.

Any implied warranties arising out of the sale of the Medium including the implied warranties of merchantability and fitness for a particular purpose are limited to the above 90-day period. In no event shall Coleco be liable to anyone for incidental, consequential, contingent or any other damages in connection with or arising out of the purchase or use of the Medium. Moreover, Coleco shall not be liable for any claim of any kind whatsoever by any other party against the user of the Medium.

This limited warranty does not extend to the programs contained in the Medium and the accompanying documentation (the "Programs"). Coleco does not warrant the Programs will be free from error or will meet the specific requirements or expectations of the consumer. The consumer assumes complete responsibility for any decisions made or actions taken based upon information obtained using the Programs. Any statements made concerning the utility of the Programs are not to be construed as express or implied warranties.

Coleco makes no warranty, either express or implied, including any implied warranties of merchantability and fitness for a particular purpose, in connection with the Programs, and all Programs are made available solely on an "as is" basis.

In no event shall Coleco be liable to anyone for incidental, consequential, contingent or any other damages in connection with or arising out of the purchase or use of the Programs and the sole and exclusive liability, if any, of Coleco, regardless of the form of action, shall not exceed the purchase price of the Medium. Moreover, Coleco shall not be liable for any claim of any kind whatsoever by any other party against the user of the Programs.

This warranty gives you specific legal rights, and you may have other rights which vary from State to State. Some states do not allow the exclusion or limitation of incidental or consequential damages or limitations on how long an implied warranty lasts, so the above limitations or exclusions may not apply to you.

SERVICE POLICY

Please read your Owner's Manual carefully before using your Medium. If your Medium fails to operate properly, please refer to the trouble-shooting checklist in the Operating Tips Manual or the Operating Procedures in the Disk Drive Manual. If you cannot correct the malfunction **after** consulting these manuals, please call Customer Service on Coleco's **toll-free service hotline: 1-800-842-1225 nationwide.** This service is in operation from 8:00 a.m. to 10:00 p.m. Eastern Time, Monday through Friday.

If Customer Service advises you to return your Medium, please return it postage prepaid and insured, with your name, address, proof of the date of purchase and a brief description of the problem to the Service Center you have been directed to return it to. If your Medium is found to be factory defective during the first 90 days, it will be repaired or replaced at no cost to you. If the Medium is found to have been consumer damaged or abused and therefore not covered by the warranty, then you will be advised, in advance, of repair costs.

If your Medium requires service after expiration of the 90-day Limited Warranty period, please call Coleco's toll-free service hotline for instructions on how to proceed: **1-800-842-1225 nationwide.**

IMPORTANT:
SAVE YOUR RECEIPTS SHOWING DATE OF PURCHASE.

PREFACE

WHO IS THIS MANUAL FOR?

This manual is designed to be useful to novices as well as to experienced programmers. It is divided so that if you are experienced and ready to use the ASSEMBLER, you can go directly to the customized Reference Manual, Part C.

Or, if you are a novice just starting out, you can begin at the beginning. The first part gives some simple background on computers and the role of operating systems such as CP/M 2.2®. This very basic information intends only to acquaint the unfamiliar user with some background. It is by no means a full description of how a computer functions.

Part A is written so that those who have done little more than look at a computer terminal can run applications through CP/M 2.2. It discusses disk care, making a backup copy of the original CP/M 2.2 data pack or disk, and using write-protect tabs. Part A also explains both how to **boot** CP/M 2.2 into ADAM™ and then how to **load** the CP/M 2.2 applications programs that you buy.

Read **Part B,** if you are interested in more than just loading applications programs and want to use some of the CP/M 2.2 utilities. These chapters teach some useful and simple CP/M 2.2 commands and list the error messages you might encounter. They also provide the information you'll need for making copies of disks, data packs or files; renaming files; erasing files, or checking how full is your disk or data pack.

The bibliography at the end of Part B suggests several commercially published books that are natural supplements to the material presented here. Recommended are several books for the programmer who wants to learn assembly language programming.

Part C is the bulk of this manual, a CP/M 2.2 Reference Manual for the experienced programmer. If you get hooked on CP/M 2.2, you might want to explore its more sophisticated aspects and re-program CP/M 2.2 to suit your needs. If you're at this level, you already have some background with assembly-level programming.

Part D contains the appendices—there you'll see the Glossary, the Summary of Commands and CP/M 2.2 Messages. Once you are familiar with the CP/M 2.2 commands discussed in the manual, use the appendices for easy reference.

The last part is an index to the manual.

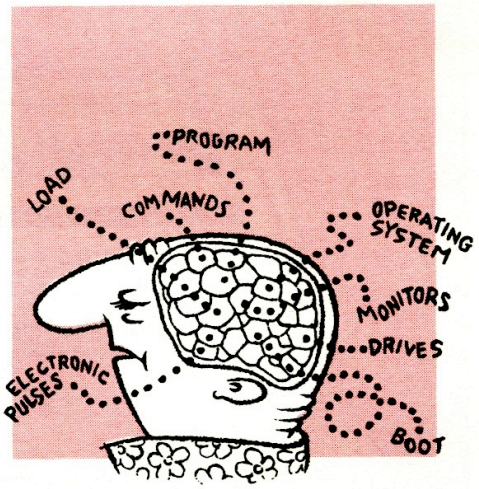
Use these five parts to fit your personal use of CP/M 2.2. You are the best judge of where to begin.

PART A:
OPERATING PROCEDURES

PART A:
OPERATING PROCEDURES

INTRODUCTION

WHAT IS CP/M?



Despite all its high-faluting and fancy packaging, the microcomputer, at its simplest, is a bunch of mechanical and electronic devices that pass electrical pulses to one another. These electrical pulses interact in a highly complex manner that rivals the synapses of the human brain.

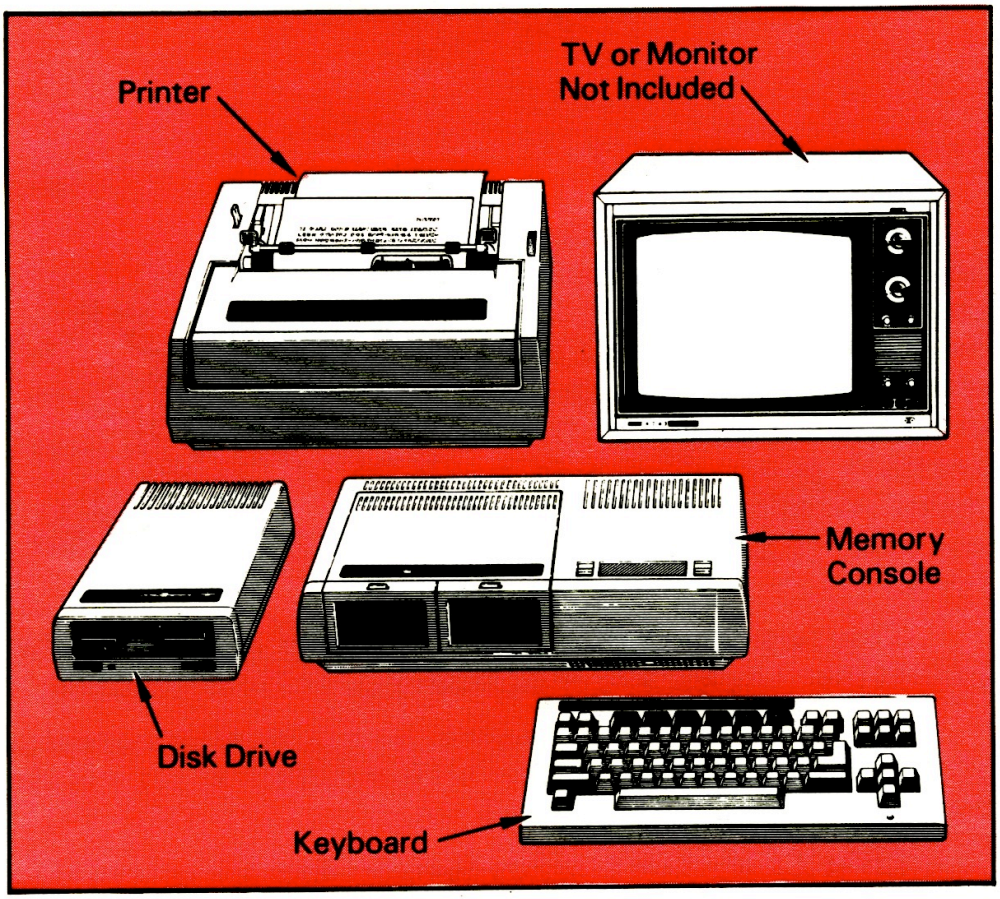
That beige box, sitting patiently on your desk or table, organizes this complex flow of impulses into useful information.

To help it, inside your computer is an **operating system,** a program or set of programs that helps the different parts of the computer communicate with each other.

CP/M 2.2 (Control Program Monitor) does just exactly that: It is a program that controls and monitors the operations of a small computer—such as your ADAM.

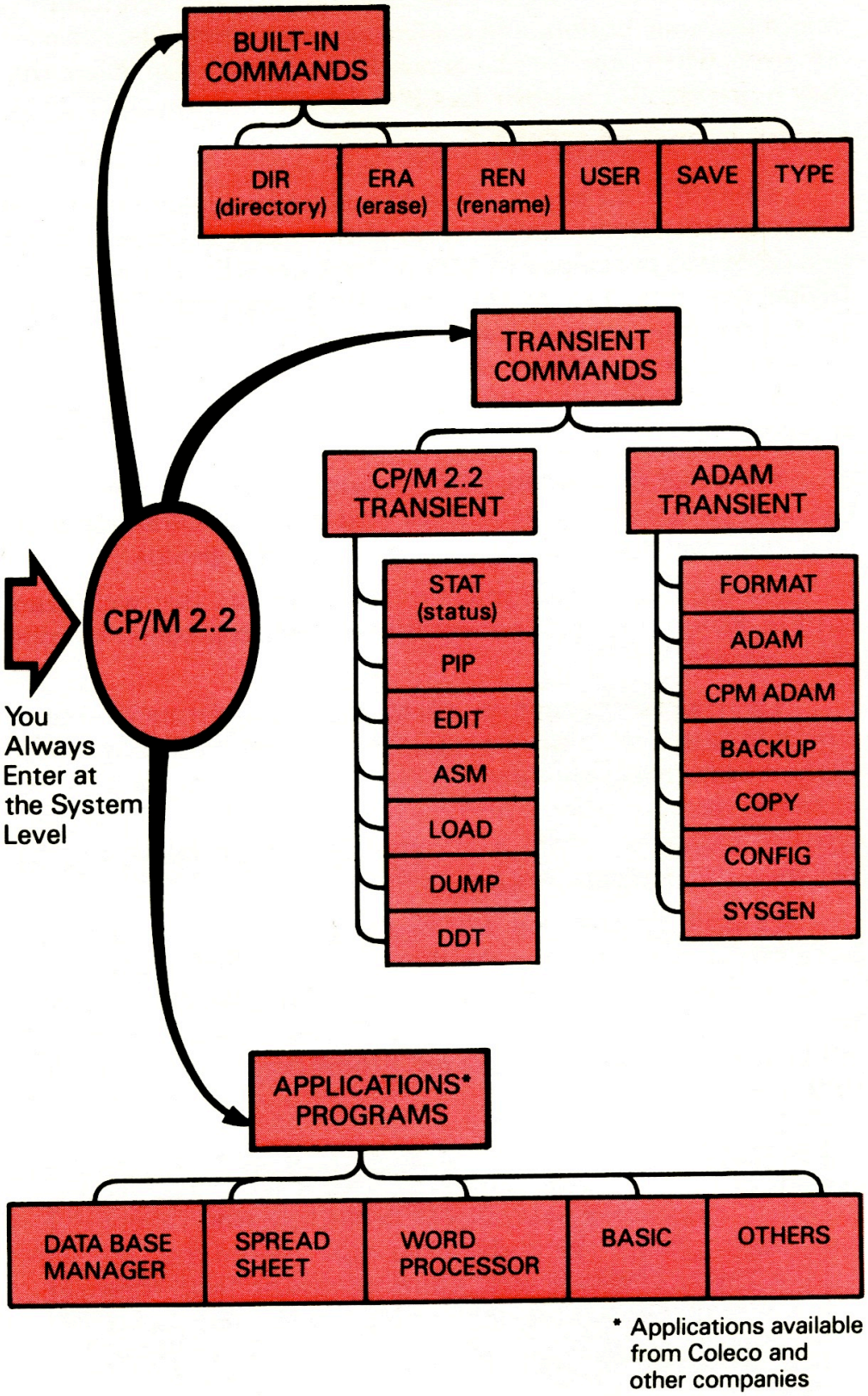
But at first glance, it might seem that CP/M 2.2 doesn't do much since its operations are often invisible. Yet, after you **boot** CP/M 2.2 (transfer an operating system program stored on data pack or disk to the computer), you can **load** the applications program (transfer it to the computer already booted with an operating system). An applications program is a set computer programs used to accomplish a specific user task.

Separate CP/M 2.2 software applications use already established CP/M 2.2 routines. This way, the applications programs do not need their own programs to move data to and from keyboard, drives, television or monitor screen, disk, data pack, printer, and even to and from a telephone modem.



CP/M controls ADAM and its components

What are some of the visible aspects of CP/M 2.2? People talk about **built-in** and **transient** commands in CP/M 2.2. The built-in features are standard facilities—such as ERA (which erases a disk file). These are loaded when you load the CP/M Operating System. The transient commands are subsets of CP/M 2.2 that are called in from the disk or data pack for more complicated jobs such as debugging. Some of these subsets are discussed in Part B. Others, which involve more in-depth understanding of CP/M 2.2, are discussed in Part C. The diagram on the next page will help you visualize the different types and levels of CP/M 2.2 commands.



Types of Commands in CP/M 2.2

If you want to learn them, Part B provides self-teaching guides for some built-in and some transient commands. For now, however, when you see the Smart Keys appear, remember that they represent all the **built-in** CP/M 2.2 commands.

Background of CP/M 2.2

This CP/M 2.2 jargon—built-in and transient commands and a host of other terms—might seem alien to you. It helps to know that CP/M was developed in 1975 by Dr. Gary Kildall who started Digital Research, Inc. At that time, small computers hadn't exploded onto the consumer scene. But a few years later with the commercial availability of small personal computers, CP/M was the only well-known standard small computer operating system around.

Since that time, many CP/M enthusiasts have designed utilities and programming languages that require CP/M for their operation. And hobbyists developed strong user groups.

But the CP/M jargon refers to those devices that were standard in 1975, practically ancient history—when it comes to computer science.

These earlier computers might have put information out to a printer or to a teletype or to the video screen. And many of the CP/M words reflect this early heritage. The command TTY, for example, means teletype. In most likelihood, you will not use a teletype with your ADAM. The command RDR, as another example, means Paper Tape Reader.

It's doubtful that you will be doing much paper tape reading with your ADAM either. But this jargon shouldn't throw you at all. Once you become familiar with CP/M 2.2 and the tasks you wish to perform, you'll be thinking ERA and DIR and some REN all the time!! And when you get really proficient, you'll know SYSGEN and ASM.

If you are interested in contacting any of these user groups, the three most well known are: (1) CPMUG Lifelines; 1651 Third Ave.; New York, New York 10028; (2) SIG/M; Box 97; Iseline, New Jersey 08830; (3) BDS C User's Group; Box 287 Yates, Center KS 66783

CP/M Applications and Utilities

Applications: With CP/M 2.2 you can take advantage of the wealth of existing software. Professional programmers have devised useful application programs. You can purchase these separately to run with your ADAM CP/M 2.2. Some programs move words around on a screen and print them, such as word-processing application programs. Some manipulate numbers as spread sheets do. Loading CP/M 2.2 into your ADAM allows you to run any ADAM Compatible CP/M 2.2 application program as long as it is ***ON AN ADAM CP/M 2.2 FORMATTED DATA PACK OR DISK.***

Utilities: CP/M 2.2 has several utilities (CP/M 2.2 programs loaded off the data pack or disk) right with the operating system itself. By using CP/M 2.2 and its utilities you can keep your data pack or disk files organized and up-to-date.

In fact, CP/M 2.2 is like an extraordinarily efficient home file manager. Use CP/M 2.2 to prepare blank data packs or disks for CP/M 2.2 programs in a process called formatting. Then CP/M 2.2, with its random access capabilities, can quickly go to any file on your disk or data pack. CP/M 2.2 knows how to get information to the printer as well as how to route information from one of ADAM's data pack drives to a disk drive. CP/M 2.2 knows an efficient method of copying files, disks, and data packs. And CP/M 2.2 can even take those files you've made using either the ADAM word processor or BASIC and convert them. Then you can use them in a CP/M based word processor or a CP/M 2.2 based BASIC.

Much of this is **transparent** to you—that is, you won't see CP/M 2.2 operating at all. You will notice your computer operates smoothly and efficiently.

Version 2.2

Since that first version of CP/M developed in 1975, there have been several others. One current version produced by Digital Research Inc. is known in computer circles as CP/M 2.2. Coleco's CP/M 2.2 can understand all programs written in CP/M 2.2 (as long as the program is on an ADAM CP/M 2.2 data pack or disk). When you buy programs or read about CP/M for your ADAM, make sure you are dealing with CP/M 2.2 and ADAM compatible software because not all versions of CP/M can understand one another completely.

THE CARE AND HANDLING OF DISKS

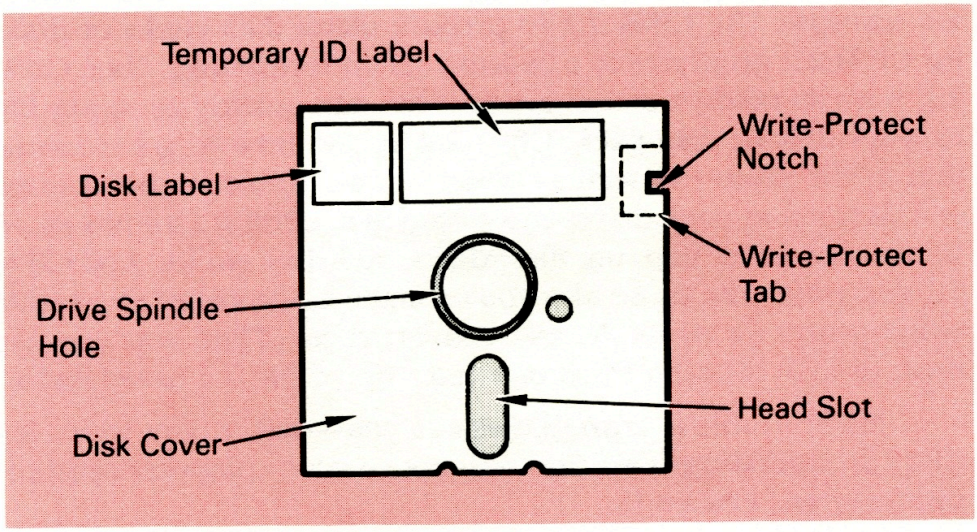
Knowing Your Disks

Not all disks are compatible or interchangeable between systems. When you purchase blank disks, make sure they are:

- Single-sided (Or you can purchase double-sided disks, but you will use only one side.)
- Double-density
- 5¼″

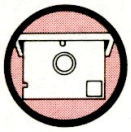
NOTE: When you purchase software on disk to be used with CP/M 2.2, make sure the software is ADAM-compatible.

Remember that although disks are extremely durable, they cannot withstand abuse. A coffee spill on any of the sensitive exposed areas, such as the head slot, is a disaster.

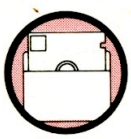


Computer disks must be handled wisely. The instructions below give tips for good disk care.

DO'S



1. Always insert and remove disks carefully.



2. Store disks in their paper sleeves.

3. Keep disks in a clean place, away from electrical currents, dust, liquids and extreme heat and cold.
4. Before you turn off the computer or pull the plug, always remove the disk.
5. Keep extra copies of disks separate from the originals. Be sure to make backups.

DON'TS

1. Never touch the shiny exposed recording surface on a disk. The oil from your fingers can ruin a disk.

2. Keep disks away from food and drink. A soft drink spill can destroy months of work.

3. Keep disks away from magnets and magnetic fields. Many electrical appliances such as refrigerators, telephones, vacuum cleaners, televisions, even your ADAM memory console and printer have magnets.

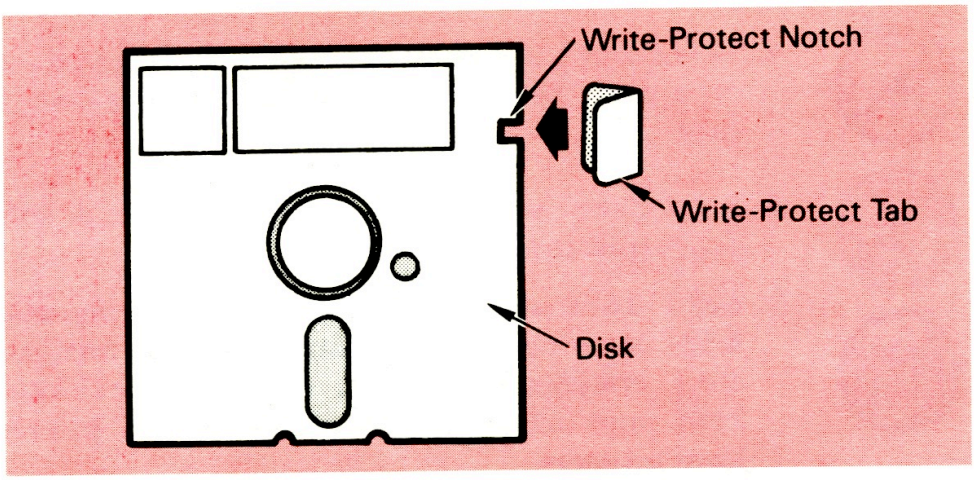
4. Keep disks away from extremes of temperature. Temperatures below 50°F and above 125°F can damage your disks. Avoid placing disks on components that get warm such as the Memory Console, printer or television.
5. Never put the disk on top of your ADAM, even for just a minute. The heat and the magnets can damage the disk destroying anything stored on it.

6. Never bend a disk. Do not use staples or paper clips on disks.

7. Never use sharp or pointed objects on the surface of the disk. It's a good idea to write on your disks with a felt-tip pen only. Never use a pencil or ballpoint pen.
8. Never attempt to remove a disk from the drive when the Drive-In-Operation light is on.
9. Never turn the Disk Drive power off while a disk is in the drive.

Write-Protect Tabs

A sheet of write-protect tabs comes with every ADAM disk product. To write protect a disk, carefully remove an adhesive tab. Fold the tab—adhesive side to disk—over the write-protect notch so that it completely covers the notch. All four corners must be flat. Improper application of the write-protect tab can cause disk jamming in the drive.



Write-Protecting a Disk

If you remove the write-protect tab, you can write to the disk, alter its files, reformat it, or do any of the disk management operations that **write** to the disk.

TURNING ON ADAM AND LOADING CP/M 2.2

Before you read this section, start ADAM and have your CP/M 2.2 data pack or disk at hand. Follow the instructions as you read them and watch ADAM respond. You'll get comfortable with CP/M 2.2 as you learn by doing.

If you have experimented with SmartBasic™, you already know how to make ADAM understand a different language. When you put the SmartBasic data pack into the data pack drive and pull the Computer Reset Switch, ADAM knows the language of SmartBasic.

Teaching ADAM to understand CP/M 2.2 is just as easy. Turn on ADAM.

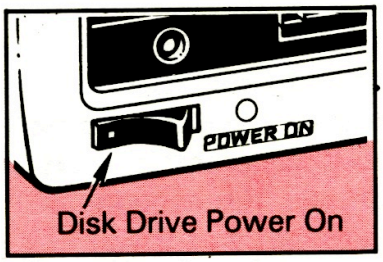
Using a Data Pack

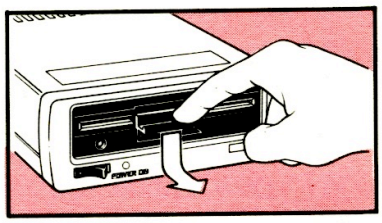
Turn computer ON. **Insert the CP/M 2.2 digital data pack.** **Pull Computer RESET Switch.**

Using a Disk

With a disk drive, the procedure is somewhat different. The instructions below show you how.

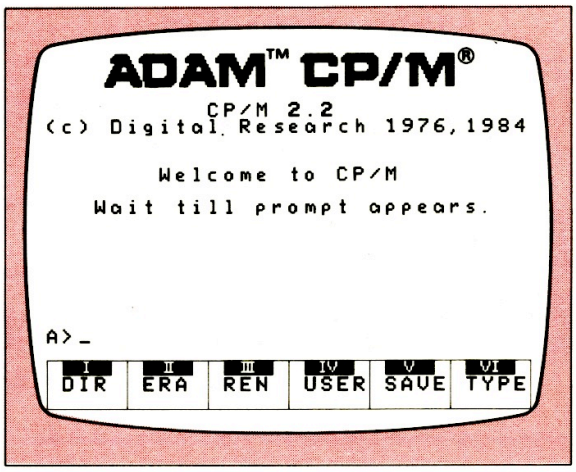
1. Make sure there are no data packs or disks in the drives.
2. Make sure the disk drive switches are set properly.
3. Turn the disk drive power **ON.**
4. Turn the ADAM **ON.**
5. Insert the CP/M 2.2 disk into the first disk drive, label side up, write-protect notch on the left side.
6. Press the drive latch down, pulling out slightly at the bottom.
7. Pull the Computer Reset Switch.





NOTE: If your keyboard is disconnected, CP/M 2.2 lets you know with a string of ^@ ^@ across the screen.

Whether you use a disk or a data pack, the following CP/M 2.2 title screen appears in a few seconds.



CP/M 2.2 Title Screen

The CP/M 2.2 prompt arrow > lets you know that ADAM is waiting for your command. The letter **A** lets you know that all commands will be directed to Drive A, the drive from which you booted your CP/M 2.2. Notice the smart key labels at the bottom of the screen. Each Smart Key label stands for one of CP/M 2.2's built-in commands. The words in these labels may at first seem strange. But sections in Parts A and B explain what these Smart Keys can do for you. For easy reference, look to the Summary of Commands in Part D, the Appendices.

Loading a CP/M 2.2 Application Program

All CP/M 2.2 application programs run only after you have booted CP/M 2.2. Any CP/M 2.2 application program for ADAM will be on data pack or disk. Even if you purchase an CP/M 2.2 application that contains CP/M 2.2, you will need to boot CP/M 2.2 first.

NOTE: Anytime you reboot (pull Computer Reset or hold the CONTROL Key while pressing the C key), there must be a CP/M 2.2 system on the data pack or disk in Drive A.

REMINDER: Make sure all your CP/M 2.2 applications program purchases are ADAM compatible.

To load an application program, follow the procedures outlined below.

1. Follow the instructions on page A8 to turn on ADAM and boot CP/M 2.2.
2. If the CP/M 2.2 program you wish to run is on a data pack, insert the program data pack, close the door, and type the drive letter (if other than A, see B23) and the program name. Then, press return. For example, if you are operating out of the drive you loaded CP/M 2.2 and you wish to run a word processing program called COMPOSE, you type in A > COMPOSE and press return.
3. If your CP/M 2.2 program is on a disk, insert the disk, notched side on the left, label side up, and pull the latch down, then out. Type the name of the program. Then, press return.

Turning Off the Smart Keys

When the application program is ready for you, the screen will change. The CP/M 2.2 title screen scrolls off the top of your video screen and the beginning of your application program appears. The CP/M 2.2 Smart Key labels might remain on the screen, or the application program may have its own Smart Keys. If you no longer wish to see the Smart Keys, you can turn them off by holding down the shift key and pressing the UNDO key (at the end of the Smart Key row). Turning off the Smart Keys gives you three more lines. You can still use the CP/M 2.2 commands, however, just by typing in the command and pressing return. If you want to turn the Smart Keys on again, just hold down the shift key and press UNDO again.

Turning Off ADAM

When you are done using ADAM, follow these steps to turn the system off:

- Remove all disks and data packs from the drives.
- Turn the power switch on the front of the disk drive(s) to **OFF.**
- Turn the power switch on the back of the printer to **OFF.**
- Turn the TV off.

MAKING A BACKUP COPY OF CP/M 2.2

CP/M 2.2 is an incredibly powerful operating system, making it a wonderful tool. You don't want to lose it from a coffee spill or another accident.

To prevent such disasters, your first CP/M operation should be to make a backup of your CP/M 2.2 disk or data pack. Then you can use only the backup. If you are an audiophile, you probably do this with record albums by making audio cassettes from your LP. Then, you use the cassette, storing the album in a cool, safe place. Do the same with your valuable software.

But be forwarned that the procedure for backing up a disk or data pack is complicated. If you are unfamiliar with computers and have only the vaguest notions of backups and formatting, read through Part B to get a general idea of how CP/M 2.2 works. Then, when you actually make your CP/M 2.2 backup, you won't walk in cold.

To make a backup, you will need a fresh disk or data pack. Have one handy. You will then do two operations:

1. You will format the spare disk or data pack to hold CP/M 2.2 information. **Turn** to page 27 in Part B and **follow** the instructions there for **FORMAT. Verify** the disk or data pack so that you know you will be putting your backup of CP/M 2.2 on undamaged media.
2. Now, only when you have finished formatting your spare disk or data pack, **turn** to the **BACKUP** instructions on page B30.

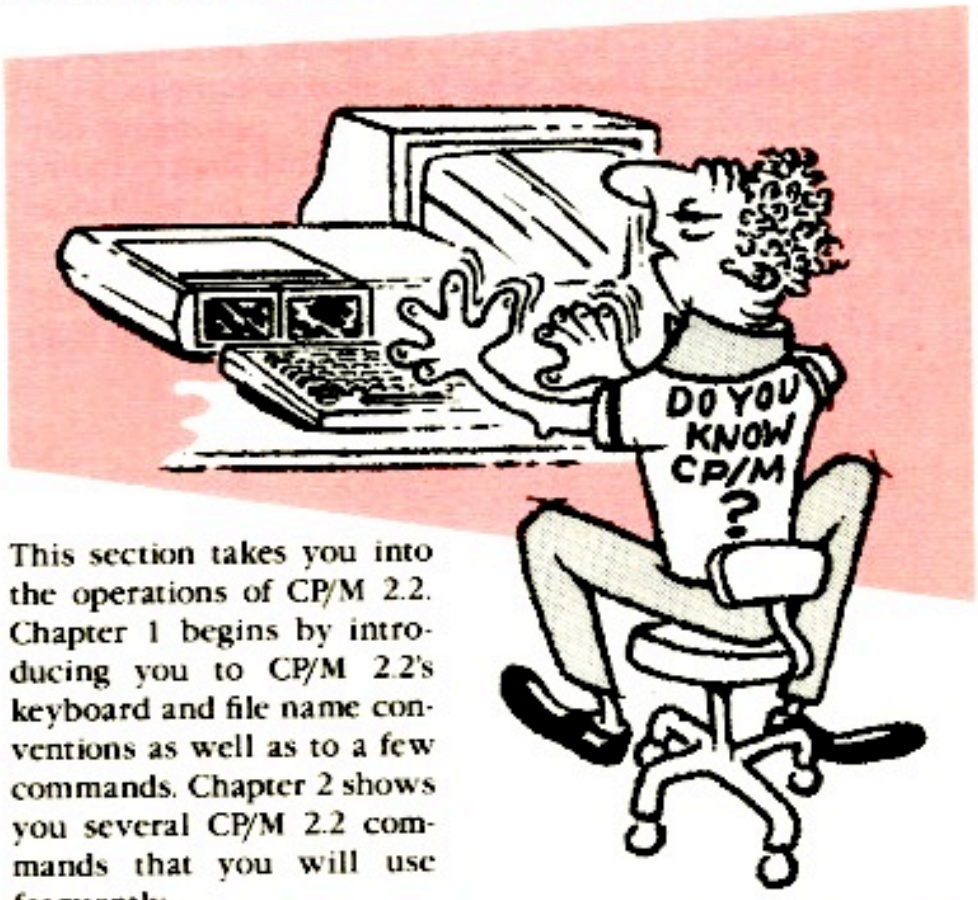
   Follow the step-by-step instructions there.

Once you have finished, store your original CP/M 2.2 data pack or disk in its sleeve or plastic case. Place it in your "vault" somewhere you intend to keep important originals of programs and data files. If you do not already have a "vault," reread the pages A6 to A7 on the Care and Handling of Disks and begin a storage vault now.

PART B:
SELF STUDY GUIDE

PART B:
SELF-STUDY GUIDE

TALKING TO ADAM IN CP/M 2.2



This section takes you into the operations of CP/M 2.2. Chapter 1 begins by introducing you to CP/M 2.2's keyboard and file name conventions as well as to a few commands. Chapter 2 shows you several CP/M 2.2 commands that you will use frequently.

If you want to do more than use CP/M 2.2 to load your other application programs, read on . . . .

CHAPTER 1: GETTING INTO CP/M 2.2

Right off, Chapter 1 gets you using CP/M. It shows you the keyboard, how to move across the 80 columns, how to erase characters; how to do **carriage returns** [indicated by the symbol <cr> throughout this and the Reference Manual].

But don't just read about these conventions, try them.

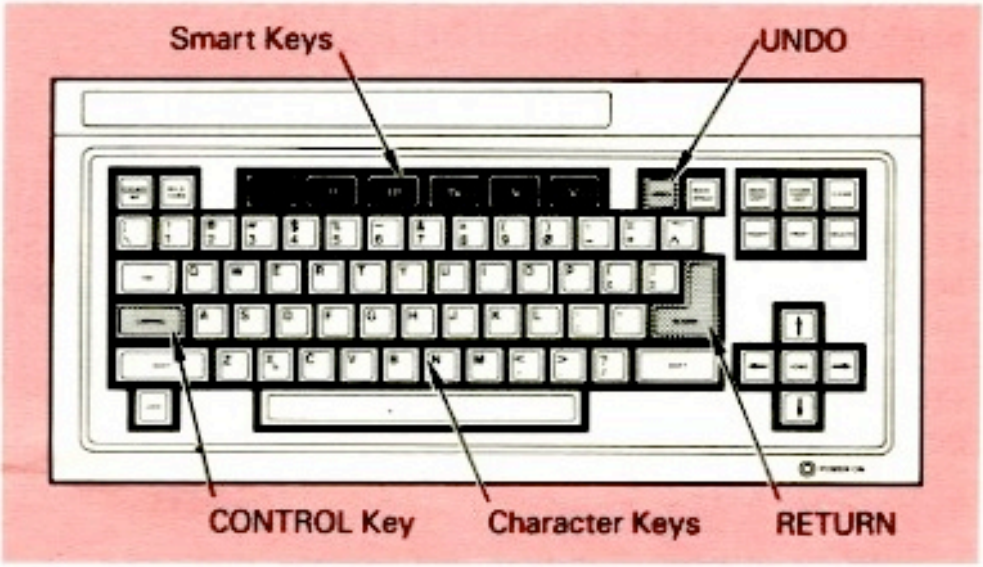
Then Chapter 1 teaches you how to use the built-in DIR (directory) command and shows what you need to know to use it: understanding a file name, fixing typing errors, viewing the contents of a file, stopping the screen from scrolling, understanding what the error messages you might get in these circumstances mean, getting help, and printing.

Want to master these? Begin by turning on your ADAM and loading CP/M 2.2. If you don't remember how, turn to pages 8–10 in Part A.

The ADAM Keyboard

CP/M 2.2 has different functions for several of your ADAM keyboard keys. Although you are probably quite familiar with the ADAM keyboard, review the illustration below to get an idea of some of the keys you might not have used often in SmartWriter or SmartBasic.

Notice the CONTROL Key, the RETURN and the UNDO Key, as well as the arrow keys and the other standard ADAM keyboard features.



Control Key Functions

With CP/M 2.2 you use several CONTROL Key functions. These require that you hold down the CONTROL Key while you press the specific key. For example, CONTROL C is a **warm boot** —a method of restarting the CP/M 2.2 program after it has been booted off of your disk or data pack. You execute a warm boot by holding the CONTROL key and pressing the C key.

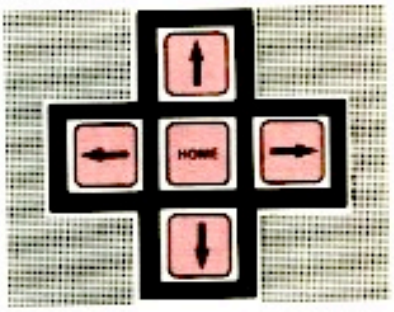
The CONTROL Key is indicated by the ^ character. Sometimes after executing a particular CONTROL Key command, you will see this character on the screen. And, if you run into **^H** command, for example, in the documentation, you know to hold the CONTROL Key while you press the H character key.

The 80-Column Screen

To get a feel of CP/M's 80 columns, hold down the space bar. As you do so, the blue cursor moves across the screen and then the screen shifts. You'll see only the blue cursor on a blank screen, with Smart Key labels across the bottom. If you hold the space bar down long enough, you will return to the CP/M 2.2 title screen.

You can see 32 characters across the screen at one time, but when the cursor reaches the edge of the screen, the screen adjusts to let you see more. It's as if you were looking through a window 32 characters wide and 20 lines high.

You can move the window in any direction to see different sections of text. You can move the window to the right by pressing the CONTROL Key and the right arrow key (⟶). You can move the window to the left by pressing ^ and the left arrow key (⟵). To get back to the title screen or to the left edge of the 80 columns, press ^J.



Typing In A Message

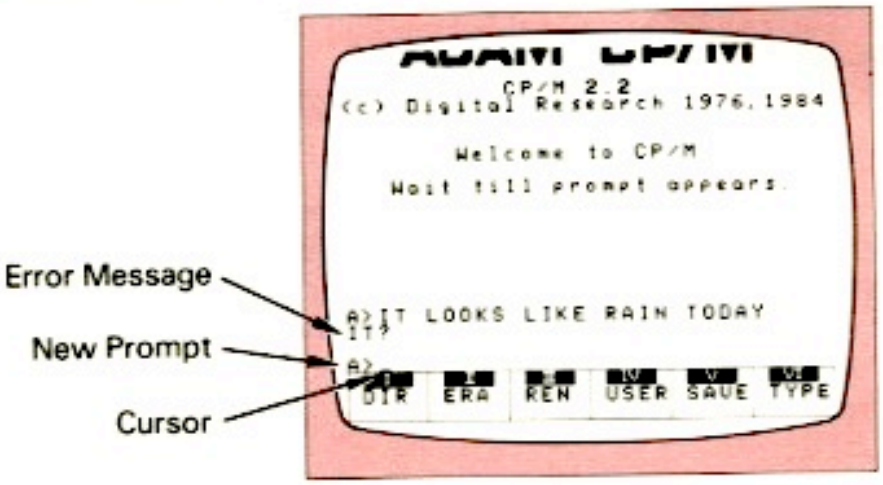
Use the keyboard to type in a message, something like "It looks like rain today." The blue cursor advances as you type, showing you where the next character will appear. If you make a mistake in typing, just press the backspace key. The cursor backs up and erases the character to the left. (Pressing ^ H also backs up the cursor and erases the character to the left.)

When you finish typing the line, press return, **<cr>**.

CP/M 2.2 goes off looking for a file name beginning with **IT**. When it cannot find that file, CP/M 2.2 responds with the question: **IT?**

If CP/M 2.2 does not understand what you are trying to say, it repeats your message, (up to the first space) but follows it with a question mark.

If your message before the first space is long, CP/M 2.2 only repeats the first 127 characters.



Talking CP/M

There's a problem, here, of course. ADAM can't read your English. To get this conversation going somewhere, you need to learn some conventions of CP/M 2.2. To begin, how are you going to correct typing mistakes?

Erasing Characters

There are two ways of erasing characters—the backspace and ^ H.

Try the ^ H. Type some words. Now, press ^ H. You can hold the CONTROL Key down and press the H for each letter you want to erase. Or, you can hold both the CONTROL Key and the H down continuously. Watch the cursor erase everything right back to the A > prompt.

Carriage Returns

There are three methods for carriage returns—the signal to CP/M 2.2 that you are finished talking, that CP/M 2.2 can now read your message or execute the instructions you gave it.

Method 1: RETURN: Type a couple of words, anything, then press the return key to send CP/M 2.2 your message.

Method 2: ^ J: You can also return (cursor moves to the left edge of the screen) by holding the CONTROL key and pressing J.

Type another line of gibberish and try it.

Method 3: ^ M: The third method for carriage returns is to press the CONTROL key and simultaneously press M.

Each of the above methods for carriage returns are equally effective. You will use the carriage return <cr> most often.

Dropping down a line: If you just want to drop down a line on the screen, you can use ^E. It takes you down one line and back to the left screen edge. However, remember that ^E does not send a message to the CP/M 2.2. The command gives you a new prompt line instead.

WHAT CP/M 2.2 CAN DO FOR YOU

CP/M 2.2 lets you take advantage of ADAM's capabilities. It's good at organizing, storing, transmitting, and otherwise handling information efficiently.

One of a microcomputer's many strong points is storing information. Most people store information in file folders in a file cabinet. They write names on the individual file folders to identify what is in each folder. Then they can find the information easily.

The computers' file folders are electronic files on a media like data packs or disks. With CP/M 2.2 and ADAM you can go to another file without losing your place in the first. You can jump back and forth between files, no matter what "drawer" you have stored them in. You make up your own names for the computer files. When you want to see the information contained in a file, ask for that file by name. Later on, you will learn how to handle a file, once you know how to call for it.

The next two sections discuss how to handle files. The major points discussed are:

- **DIR** How to see the names of the files on a data pack or disk with the DIR (I)—**directory**—command.
- **TYPE** How to see the information contained in an **ASCII** file on the screen using the built-in TYPE (VI) command.
- **REN** How to change the name of a file with the REN (III)—rename—command.
- **^P** How to direct information that appears on the screen also to be printed by using ^P.
- **HELP** How to see the words needed for different commands using the TYPE command.

THE DIR COMMAND

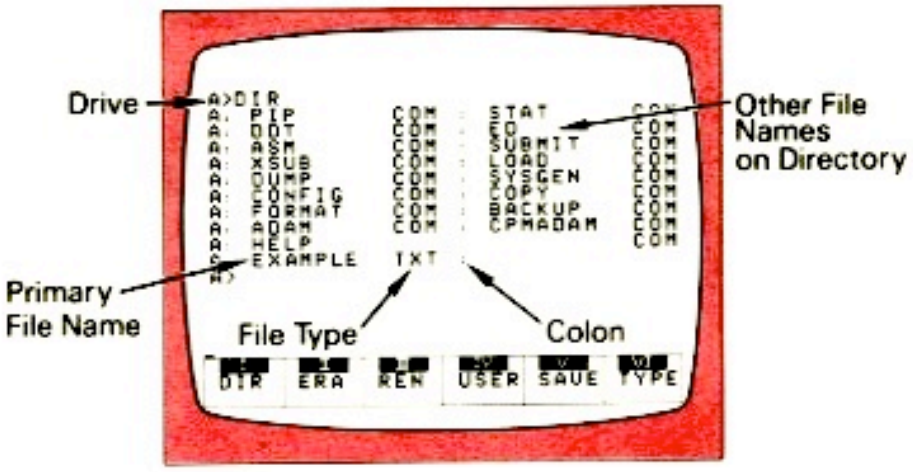
The CP/M 2.2 data pack or disk already has some files stored on it. Ask CP/M 2.2 to show you the names of these files using DIR.

Step 1

- If you do not already have ADAM on and CP/M 2.2 booted, follow the instructions in Part A, page A8 to do so. Leave the CP/M 2.2 data pack or disk in the drive.
- The CP/M 2.2 prompt arrow (>) shows that CP/M 2.2 is ready for a command. The prompt reminds you that everything you do will be directed toward the files in Drive A, the same drive that you used to bring up CP/M 2.2.

Step 2

- Notice that Smart Key label 1 says "DIR". Press Smart Key 1 on the top row of the keyboard and "DIR" appears beside the prompt on your screen. Now press the return key. Pressing the return key always tells CP/M 2.2 to perform a command. In this case, you've commanded CP/M 2.2 to show you the directory of the disk or data pack in Drive A. CP/M 2.2 responds with a list of file names that looks like the illustration below.



CP/M 2.2 Directory

The **directory** lists the files stored on the CP/M 2.2 data pack or disk. For now, don't worry about what's in these files—just look at their names. On the sample file name, notice the drive name: **A**. This name indicates that these are the files on the data

pack or disk in Drive A. Notice that there is a colon after the drive label in the file names. There is an arrow > after the drive label in line prompts.

Understanding A File Name

Each file name has two parts: The primary file name and the file type. In the illustration on page B8, the last entry has EXAMPLE as its primary file name that refers to the specific file. The second part is the file type that tells you about the content of the file and its uses. If the file type is COM, for example, the file contains executable instructions to the computer. The file type TXT in the file EXAMPLE.TXT means that the file contains text. Both parts of the file name should be used.

You can even make up your own file types. Maybe you want to use the first three letters of your name or your initials to easily distinguish between files you created and files created by someone else. Or you could identify all the files that pertain to a certain subject with an appropriate file type, say, for example history files, you might use PAPER1.HST.

Some Tips About Creating File Names

- A primary file name can be 1 to 8 characters.
- File type can be 1 to 3 characters. You don't have to create a file type, but if you do, include the file type when you get that file off of the data pack or disk.
- When you create or refer to a file name, type a period between the primary file name and the file type.
- CP/M 2.2 won't stop you from putting in more characters than are allowed. But it will only use the first eight characters you type for the primary file name and the first three for file type.
- CP/M 2.2 does not pay attention to the difference between upper and lower case letters. So if you type in a file name hello, CP/M 2.2 considers it to be the same as HELLO.
- If two files have the same primary file name but different file types, they are considered to be two different files. For example, HELLO.COM is a different file than HELLO.TXT.
- CP/M 2.2 does not recognize some characters. You can type them, but CP/M 2.2 may eliminate them when storing the file. Some programs will not recognize the name. The characters CP/M 2.2 does not recognize are the following: < > . , : ; = ? * _

Below are examples of names that CP/M 2.2 won't recognize:

NAME	WHY CP/M 2.2 MAY NOT RECOGNIZE IT
COUNTRYSNOW	Too many characters. CP/M 2.2 recognizes and uses only the first 8 characters.
CT,SNOW	Contains a file name character not recognized by CP/M
CT SNOW	Contains a space in file name
< > CTSNOW	Contains a file name character not recognized by CP/M
CTSNOW*	Contains a file name character that has special meaning for CP/M

Unrecognizable File Names.

It is best to use file names that are memory joggers, devices that help you remember what is in the file. Naming your term-paper files as 1, 2, and 3 isn't as helpful as naming them TERMPAP1.TXT, TERMPAP2.TXT, and so on.

For more detailed information about file types, refer to Part C.

Fixing Typing Errors

You already know that if you make a typing error, you can use the backspace key or ^H to erase a character. CP/M 2.2 has some other handy features that make it just as easy to erase a line or make it invisible. The first two ways to erase are summarized again on B11. Methods 3 and 4 have not been discussed yet, so as you read about them, try them.

Method 1: Backspace: Type a character that you do not want. Press the backspace key. Your unwanted character is quickly erased.

Method 2: ^H: Try pressing the CONTROL and H keys simultaneously to erase the unwanted character.

Method 3: ^X: To see how this method works, type some gibberish. Don't press return, since CP/M 2.2 won't understand gibberish. To quickly erase the whole line, hold down the CONTROL key and press X. The line disappears and the cursor moves back to the beginning of the clean line. It's as if the gibberish never existed.

Method 4: ^U: Another way to erase a line is with ^U. Again, type something, a string of characters or a few words.

Then hold down the CONTROL key and press U. This time the cursor moves to a new line, but the meaningless characters—sometimes called garbage—stay on the screen and a number sign # appears beside it. The number sign indicates that the garbage characters will be ignored.

```
A> dneotr0cl;spwsz #
A>
```

Typing Capitals

Use the shift lock to type in the upper case. To go back to lower case, push lock again.

If you want to capitalize only letters (and leave numbers as they are), use CP/M 2.2's special feature. Just press CONTROL Key + up arrow and all the letters you type appear as capitals on the screen. Numbers remain. To undo this feature, again press ^ up arrow key.

THE TYPE COMMAND

Viewing the Contents of a File

If you are curious about what's in the files listed on the directory screen, use the TYPE (VI) command to look at them.

With the CP/M 2.2 TYPE (VI) command, you can see the contents of any file that uses ASCII characters. If the program file is an executable one such as a COM file, you will see non-ASCII text on your screen.

ASCII is a common acronym that you'll see frequently in computer related writing. The letters stand for American Standard Code for Information Interchange. The ASCII code is a commonly agreed upon information code. There are 128 codes; 96 displayable characters—upper and lower case alphabet, punctuation marks, and numbers. There are also 32 non-displayable characters. The code for any text—letters, reports, address files—or for any SmartBasic program is ASCII code. See Part D for a complete list of the ASCII code.

To try out the TYPE command, use the file name EXAMPLE.TXT and follow the step-by-step instructions below to view the contents of that file.

Step 1

- Press TYPE (VI). "TYPE" appears on the screen.

Step 2

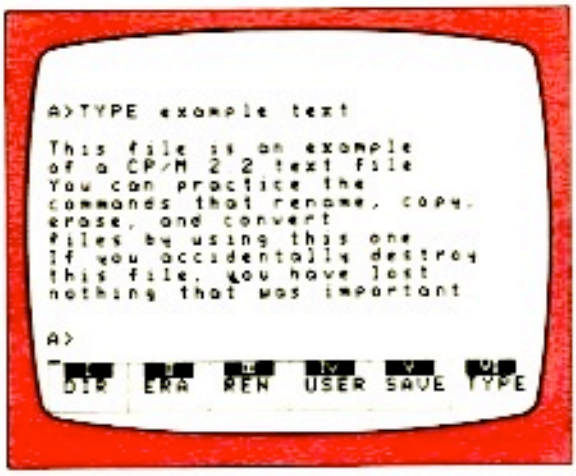
- Type in the name of the file you wish to see. Be sure to include the period between the primary file name and the file type.

  The command line should look like this:

  A > TYPE EXAMPLE. TXT

Step 3

- Now press return to tell CP/M 2.2 to perform the command. CP/M 2.2 takes a minute to find the file named EXAMPLE.TXT on the data pack or disk. Soon, the contents of the file appear on the screen.



Contents of The EXAMPLE.TXT File

As you can see, this file is an example of a text file. You can use this file to try out CP/M 2.2 commands with no fear of losing something or changing an important file.

Stopping the Screen Scroll

When you practice using the TYPE command, you might wish to freeze the screen so that you can read the file's contents instead of merely watching it sweep by.

You can stop the display by pressing ^S. This combination of key presses acts just like a pause button. Press any key to start the display again.

Error Messages

If you mistakenly typed the file name incorrectly, CP/M 2.2 responds with an error message: The incorrect file name as you typed it, followed by a question mark. But when you ask for something from the directory, CP/M 2.2 might give you some other error messages. These error messages are summarized below.

NOTE: A general rule of thumb if you are using a data pack drive and an error message appears is to pop open your data pack drive, adjust your data pack, and close the drive again. Now retry your command.

DIT?

This message appears when CP/M 2.2 could not find the command you typed. In this particular instance, CP/M 2.2 could not find the COM file named DIT—you probably wanted DIR and typed DIT.

NO FILE	This message appears if you press or type DIR and then type a name for a file that does not exist. This message appears with other commands as well.
Missing Media Abort, Retry	Another error you might see when you are first experimenting with CP/M 2.2. This one occurs if you forgot to put in the data pack or disk.
Missing Block Adjust Media Abort, Retry	bad section of tape or disk. It also appears if a non CP/M 2.2 data pack or disk is in the drive. Before retrying, be sure to adjust media.
Unknown Drive Ignore, Abort	You receive this error message when you have told CP/M 2.2 to go to a drive that does not exist. Have you typed the drive label correctly? Check your drive switches and connections?
Bad System, Retry on A: ___	This message occurs whenever you have done a warm boot and the disk or data pack in drive A does not have the CP/M 2.2 Operating System on it. Replace disk or data pack with one that has CP/M 2.2 and press ^C.
Bdos Err on ___: Select	CP/M 2.2 receives a command specifying a nonexistent drive. CP/M 2.2 terminates the current program as soon as you press any key. Press return or ^C to recover. **NOTE: The only safe response to the BDOS error message is to type ^C. CP/M 2.2 will reload. (When the computer is on and running and you wish to reload a program, the procedure is called a warm boot.)**

To re-do the command correctly, type the name of the command and the file. In this case, type TYPE EXAMPLE.TXT. Correct any mistakes using backspace, ^H, ^U or ^X, and then press RETURN.

Getting Help

Use the TYPE command just shown to get help on how to make ADAM do a particular task.

Step 1

- Press or type TYPE.

Step 2

- Type in HELP and press return. The words you need to talk to CP/M 2.2 appear on the screen. These are the names of the CP/M 2.2 commands, the "syntax" of CP/M 2.2. As you learn them, you are learning the CP/M 2.2 language.

Step 3

- If the screen scrolls by too quickly, remember that you can press ^S to stop it.

If you want to give CP/M 2.2 a particular command, you need to only follow the syntax given to you on the HELP screen. In this text, the < > around a portion of the command show that this portion is required for CP/M 2.2 to understand your message. If you leave out required parts of a command, CP/M 2.2 won't know what to do. CP/M 2.2 will give you an error message or a question mark, and you'll have to try again.

If you type a COM file (a command) and press return, you instruct CP/M 2.2 to find the command file on your data pack or disk, load it into the ADAM Memory Console, and then execute it. For now, don't give CP/M 2.2 any commands. Some are **destructive;** that is, you can lose important files if you aren't sure of what you are commanding CP/M 2.2 to do.

Later, though, when you are confident and ready to command CP/M 2.2 to perform a task, just type the command you want and then press the return key.

The table below gives you the **syntax**—the relationships of characters and operation—for the most frequently used CP/M 2.2 commands. These words are the same as those that appear on the HELP screen.

You already know that the portions of the command that are in < > are required parts of the command. As you look at the table below, you will also notice that some portions of the commands are in brackets [ ]. The brackets show that these portions are optional. You would include a drive, for example, only if you wanted to specify a different drive from which to get files.

```
DIR [dr:] [file]
TYPE [dr:] <file to type>
REN <new> = <old>
ERA [dr:] <file to erase>
USER <user 0 to 15>
STAT [command line]
PIP [command line]
SAVE <pages> [dr:] <file>
DUMP [dr:] <file to dump>
ED [dr:] <file to edit>
LOAD [dr:] <hex file>
ASM [dr:] <file to assemble>
COPY [dr:] <existing file> [dr:] <new file>
A: [file]
b: [file]
FORMAT  <cr>
BACKUP  <cr>
CONFIG  <cr>
SYSGEN  <cr>
ADAM  <cr>
CPMADAM  <cr>
```

The Syntax For the CP/M 2.2 commands

Not all of the commands shown in this table are discussed in this Self Study Guide. For explanations of these commands, turn to either Part C or Part D.

A word of caution—except for the EXAMPLE.TXT file, the files on the CP/M 2.2 data pack or disk are the powerful CP/M 2.2 utility files. Until a practice session introduces a file, or you feel comfortable with CP/M 2.2, avoid files with which you are unfamiliar. **If you erase or change one of these files, you will not be able to get the original back.**

Printing With ^P

Sometimes you'll want a permanent, printed copy of the contents of a file. ^P instructs the printer to print everything that you type or that would normally go to the screen. Combining the TYPE command with ^P gives you an easy way to print a file.

Step 1

- Hold down the CONTROL Key.

Step 2

- Press the P key as you hold down the CONTROL key.

Step 3

- Follow the steps for the TYPE command: Press Smart Key TYPE (VI);
  Type in the name of the file—EXAMPLE.TXT
- Then press return.
- As you go through the TYPE command, everything you do on the screen is printed. As EXAMPLE.TXT appears on screen, it is printed at the same time.

Step 4

- If what is being printed will be more than one page, watch the printer carefully. Use the ^S (the same one you use to stop the screen from moving) to pause. When you are ready to begin printing again, hit any key.

To stop the printer, press ^P again. Or, when the printer and the display stop automatically because they have finished a job, you need to "turn off" the printer. Press ^P. CP/M 2.2 now knows to stop sending information to the printer even though it will continue sending information to the television screen.

NOTE: If you print a file containing lines longer than 80 characters, the full lines are not printed. ADAM's printer stops at the 80th character, skips the rest of the line, and starts again with the next line in the file.

THE REN COMMAND

The REN (III) command lets you change the name of a file easily. However, REN (rename) does not change the contents of the file, only the name. To see how REN works, try changing the name of the EXAMPLE.TXT file.

Step 1

- First, press REN (III). Just as with DIR and TYPE, the name of the command appears on the screen.

Step 2

- After the command name appears on the screen, you need to type in some specific instructions that tell CP/M 2.2 the name of the file you want renamed and what you wish to rename it.
- Type in the new name for the file.

MYFILE. TXT

Step 3

- Press the equals sign (Shift + the + key)

MYFILE. TXT =

Step 4

- Type in the name of the file as it exists now. Compare your onscreen command line to the illustration below:

A > REN MYFILE. TXT = EXAMPLE. TXT

RENAME Command line

Step 5

- Does your line compare?
- If it does, press return and CP/M 2.2 goes to work on your command! When CP/M 2.2 has finished the rename command, the prompt reappears on the screen.

Use the DIR command that you learned earlier to make sure that the file has been renamed correctly. If you don't remember the directory command, turn back to page B8 of this chapter for a quick review.

When the list of files appears on the screen, you will see a file name MYFILE.TXT. But also notice that there is no longer a file named EXAMPLE.TXT. Just as you requested, CP/M 2.2 changed the name of the file from EXAMPLE.TXT to MYFILE.TXT.

To prove that CP/M 2.2 did not change the content of the file, use the TYPE command. Type the command TYPE MYFILE.TXT. Now, the information that *used to be* in EXAMPLE.TXT should be displayed on the screen. The content of the file has not changed, but the name has.

TROUBLE-SHOOTING?

Error Messages Help You Understand What Went Wrong!

As you know, sometimes you may not get the results you expected, and you don't know what to do about it. The first step is to read any error messages that CP/M 2.2 may have sent you. You met some of these earlier in the TYPE command instructions.

If you don't understand the error message and still don't know what to do, refer to the CP/M 2.2 Message Appendix in Part D. This appendix explains the message and tells you what to do to get going again.

However, some quick tips: If you press return and CP/M 2.2 does not respond right away, it may be busy with some other task. The keyboard remembers up to 10 keystrokes while CP/M 2.2 is busy. When CP/M 2.2 is finished, those characters are displayed. If the data pack is moving or the disk drive in use light is on, you can tell that CP/M 2.2 is occupied with another task. Check the drive you indicated in your last command. Is it the one you want CP/M 2.2 to read?

Or, sometimes a computer "**hangs**" on a program—CP/M 2.2 can get stuck in a program before completing it. To get out of a hang, you may have to press ^C. However, this procedure only works if the program or application is waiting for you to enter new information. If CP/M 2.2 is busy with a task, it can't "hear" ^C. Use this technique sparingly, because you take the risk of losing some information that was being handled by the hung program when you use ^C.

There's another method, but it's an eleventh hour bail out. If you are really hung in a program, pull the Computer Reset Switch on the ADAM Memory Console.

This will reboot your system, but it is potentially destructive of files and data, particularly if you are in the middle of a backup operation.

SUMMARY

Now that you have completed this section, you've started to learn CP/M 2.2. Here's a summary of the commands and operations that you have learned.

- **DIR** displays the names of the files on a data pack or disk.
- **TYPE** displays the contents of a file.
- **REN** changes the name of a file.
- **TYPE HELP** reminds you of the syntax of a command.
- **^P** prints each character as it is displayed on the screen.
- **^U** and **^X** erase a line before you press return.
- **^H** is a destructive backspace used to erase unwanted characters.
- **^J** and **^M** are carriage returns.
- **^S** pauses the screen or printer movement.
- **Error Messages** tell you what went wrong in a command line.
- **Smart Keys** are shortcuts, letting you access the Built-In Commands with one keystroke. But you can also type in commands, yourself.

CHAPTER 2: USING CP/M 2.2

Several CP/M 2.2 commands must be handled with care. Now that you have met some sturdy ones, you are ready to tackle some of the more sophisticated commands. These require more care in use as they are potentially volatile and destructive.

Remember, no matter how smart and ingenious the computer is, it can only do what it is told to do. In the process of following the instructions you give it, CP/M 2.2 can irretrievably change or destroy information. Practice using these sensitive commands on disposable files—files that can be destroyed such as EXAMPLE.TXT. To learn anything new, the novice has to make a few mistakes—it's part of the learning process. But it's a good idea to minimize the risks.

This chapter covers the following:

- **Drives** How to use more than one drive with CP/M 2.2.
- **^C** ^C lets you "log in" a new disk.
- **FORMAT** How to prepare a disk or data pack for use with CP/M 2.2. The blank disks and the ADAM data packs you buy in the store must be specially formatted to understand CP/M 2.2. CPM's FORMAT command does this task.
- **BACKUP** How to make an extra copy of an entire CP/M 2.2 data pack or disk with the BACKUP command.
- **COPY** How to make a copy of one file at a time with the COPY command.
- **ERASE** How to get rid of a file you don't need with the ERASE command.
- **STAT** How to find out how much room is left on a data pack or disk with the STAT command.
- **ADAM** How to convert ADAM files to CP/M 2.2 files.
- **CPMADAM** How to convert CP/M 2.2 files to ADAM files.

USING MORE THAN ONE DRIVE

Up to this point, you've worked with files in Drive A. But maybe you have more than one drive—maybe you have one data pack drive and one disk drive, or one disk drive and two data pack drives, or some other combination. Naturally, more than one drive makes handling and transferring information more convenient.

Drive Labels

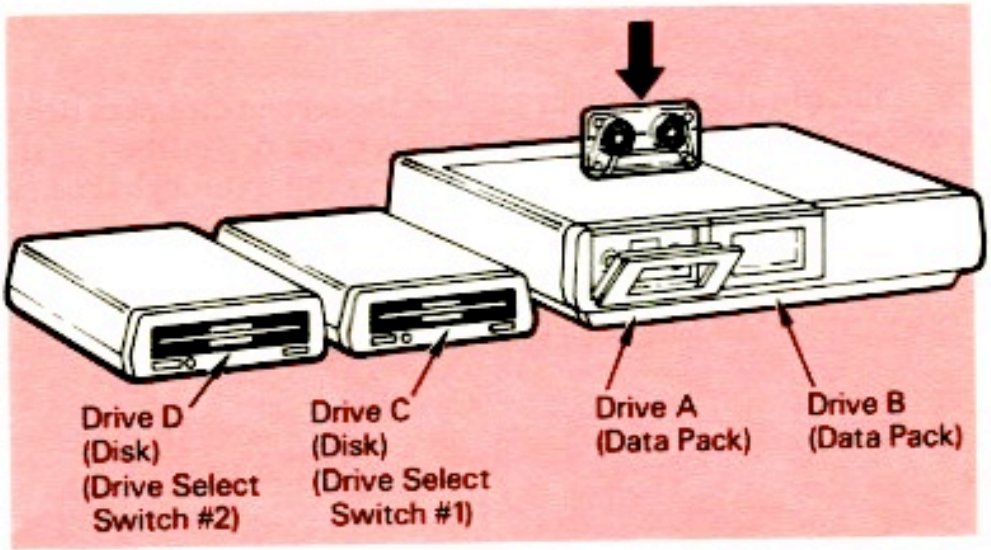
ADAM assumes that you want to deal with DRIVE A (whatever drive you used to bring up CP/M 2.2) until you say otherwise. However, to tell CP/M 2.2 to use another drive involves first knowing how CP/M 2.2 labels the drives. Alphabetical characters refer to the particular drives. But these change depending on the combination of drives you have and the way you use them. CP/M 2.2 always names the drive that brings the CP/M 2.2 program up as "Drive A." For example, if you bring up CP/M 2.2 from the data pack drive on the left, then that drive is Drive A. The drives are labeled as follows:

The data pack on the left, from which you booted CP/M 2.2, is **Drive A.**

The data pack on the right is **Drive B.**

If you add disk drives, then the first disk drive you physically attached is **Drive C.**

The second disk drive you attached is **Drive D.**



Drive labels when CP/M 2.2 is loaded from the leftmost data pack drive.

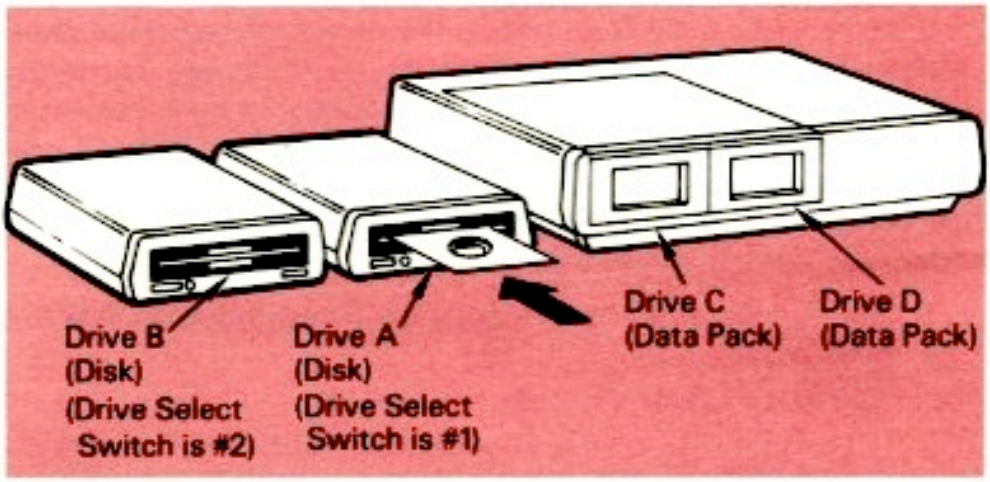
Since ADAM comes with a data pack drive, you will probably use configuration illustrated above at first. However, remember that if you change the drive from which you load CP/M 2.2, the labels for the drives also change. For example, if you bring up CP/M 2.2 from the first disk drive, the drives are labeled as follows:

The first disk drive, from which you booted CP/M 2.2, is **Drive A**.

A second disk drive is **Drive B**.

The data pack drive on the left is **Drive C**.

The data pack drive on the right is **Drive D**.



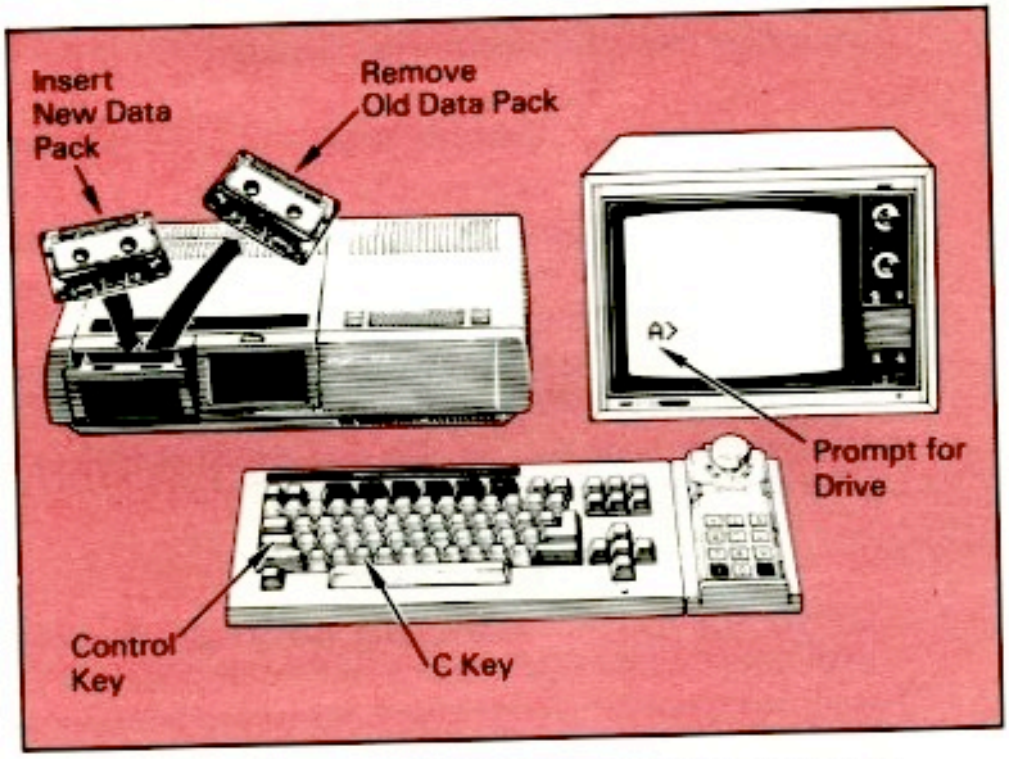
Drive labels when CP/M 2.2 is loaded through the first disk drive.

You can also boot CP/M 2.2 from the second data pack drive or from the second disk drive. However, the drive labels again change. It is recommended that you always boot through the first disk or the first data pack drive.

"Logging in" Disks or Data Packs

If you have been working with one data pack or disk in a drive, and then insert a different data pack or disk in the same drive, you must inform CP/M 2.2 of the switch. To do so, when the CP/M 2.2 drive prompt appears, simply hold the CONTROL key while pressing C. This **warm boot** allows CP/M 2.2 to recognize the different data pack or disk.

If a **BAD SYSTEM** error message appears, you tried to log a data pack or disk that was not formatted with the CP/M 2.2. operating system. Be sure your data pack or disk is properly formatted with **SYSGEN** (see B27).



Warm Boot to tell CP/M 2.2 of new data pack.

If you change data packs or disks without logging in, CP/M 2.2 behaves unpredictably. It may not let you write any information to the data pack or disk or it may capriciously write over existing files.

Forgetting to log in a data pack or disk can raise havoc.

Telling Drive Prompts Where To Get Information

Anytime you are directing information to or getting information from a specific drive, you use the drive label and a colon. For example, to tell CP/M 2.2 that you want to do something with Drive B:

Step 1

- Type B (the letter that corresponds to the drive).
- Type a colon and then press return.

```
B:
```

Step 2

- If you have a Drive B, you'll see the prompt line change to B >_. If you don't have a Drive B or your drive switches are not on, then you'll see an error message:

```
A >B:
     Unknown Drive
     Ignore, Abort
```

If you press I—to ignore the error message—without correcting the problem, you'll get another message:

```
Bdos Err on B: Select
```

Your best bet now is to do a warm boot, then correct the problem with your drive.

- If there is no data pack or disk in Drive B, you'll see a third error message:

```
Missing Media
Abort, Retry
```

Correct the problem—maybe you haven't got the drive latch shut just right—and press A <cr> or R <cr>.

In sum, the CP/M 2.2 prompt always keeps you informed as to the drive currently operating. The onscreen prompt will be A if the A drive is operating; B if the B drive is operating. If you want CP/M 2.2 to switch drives, you can do so whenever the CP/M 2.2 prompt appears at the start of a new line.

THE FORMAT COMMAND

Anytime you buy a blank digital data pack or a blank disk to use with your CP/M 2.2 operating system, you will need to format it. You can also reformat old disks and data packs in order that they can be used with CP/M 2.2. Formatting initializes the data pack or disk so that CP/M 2.2 can read and write to it. If not formatted with CP/M 2.2, the new blank data pack or disk behaves unpredictably.

Formatting does not put the CP/M 2.2 Operating System on the data pack or disk. To put the system on your media, you need to do an operation called SYSGEN. See Part C.

Although it isn't necessary to put SYSGEN on your media, we recommend it.

WARNING: ALL INFORMATION ON A DISK OR DATA PACK WILL BE ERASED BY THE FORMAT PROCESS.

Step 1

- Turn on ADAM and bring up CP/M 2.2 if you have not already done so.

Step 2

- Type FORMAT. Press return.
  On the screen appear your instructions.

```
ADAM CP/M 2.2 FORMAT UTILITY
FORMAT MEDIA ON DRIVE
(A, B, C, or D)?
```

Step 3

- Type in the letter that indicates the drive into which you will insert the blank data pack or disk.
- Press Return. Onscreen appear your instructions.
- **NOTE:** If you are formatting a disk in a disk drive, the message will be the same as illustrated below but it will tell you to insert a disk instead of a data pack.

```
INSERT TAPE TO BE FORMATTED INTO DRIVE A:
PRESS RETURN WHEN READY TO FORMAT
```

Formatting a Data Pack

Step 4

- Insert the data pack or disk into the drive you indicated that you wish to format for CP/M 2.2 files.
- Press Return.
- If you are formatting a data pack, onscreen appears the message: FORMATTING BLOCK 000

  As CP/M 2.2 gets the new data pack ready for CP/M 2.2, the number of the block appears on the screen. As CP/M 2.2 formats more blocks of the data pack, the numbers increase. When CP/M 2.2 finishes formatting the data pack, another instruction appears on the screen.
- If you are formatting a disk, the formatting block message does *not* appear.

Step 5

- If you format a disk or data pack, you may wish to also verify that the data pack or disk is indeed formatted. Once CP/M 2.2 is finished formatting, the following instruction appears on the screen.

```
VERIFY MEDIA
(Y or N)
VERIFYING BLOCK  000
```

Verifying Either Data Pack or Disk

Step 6

- If you want to verify that the disk or data pack is formatted, type a Y and press return. CP/M 2.2 will go over every part of the disk or data pack that it thought it formatted and check to see that it did. Once CP/M 2.2 finishes verifying the disk, the following instructions appear on the screen:

```
FORMAT ANOTHER
(Y or N)?
```

Formatting even more?

Step 7

- Type Y if you wish to format more data packs or disks. If you type Y and press return, CP/M 2.2 will again ask you for the drive. Insert your new data pack or disk. Now type the drive label and press return.
- If you type N, CP/M 2.2 tells you to insert your data pack or disk and press return.
- Once you do, CP/M 2.2 gives you the prompt line again.

THE BACKUP COMMAND

As a computer user, you know that information is a valuable resource. You also know that accidents can destroy that information. To avoid losing the only version of computer files, you can make extra copies of important CP/M 2.2 data packs and disks. This procedure is called "backing up." The extra copies are called "backups."

Note that it is especially important to back up the data pack or disk containing your CP/M 2.2 program—a little preventative medicine. If you accidentally ruin your CP/M 2.2 data pack or disk, a backup copy will save you the money you'd have to spend to replace the damaged one. Later you might wish to learn the special procedure for backing just the CP/M 2.2 operating system —but none of the utilities or files. This sophisticated procedure is called SYSGEN and is discussed fully in the Reference Manual.

For now, learn the simpler procedure for backing up entire data packs or disks. **BACKUP** will be easy to learn since it is quite similar to that of FORMAT.

When you back up, the blank data pack or disk onto which you put the backup needs to be CP/M 2.2 formatted. If you haven't already formatted your blank data pack or disk, do so now.

The BACKUP sequence you use will vary, depending upon the number of drives and the media (data pack or disk).

WARNING: Do not try to back up a data pack file that is longer than 160 KBytes onto a disk. If your **target**—that is, the storage device *onto which* you plan to write information—is smaller than your **source**—the storage device *from which* you read information, CP/M 2.2 gives you the error message—Target Drive Smaller than Source Drive.

BACKUP with One Drive

When you back up with only one drive, you have to do some juggling—putting the source (your original) data pack or disk into the drive, taking it out and then putting in the target data pack or disk—the empty media onto which you put your backup.

BACKUP is done in chunks of 48 blocks of information. (Data packs can hold as many as 255 blocks and disks, 160 blocks.) When you are backing up a data pack or disk with one drive, you will have to repeat the procedure of inserting the source media, reading 48 blocks of information off of it, taking it out, putting the target data pack or disk in, writing the 48 blocks of information onto it, taking it out, inserting the source media, reading another 48 blocks off of it, taking it out and so on, until the entire disk or data pack is backed up.

To back up a data pack with one drive:

Step 1

- Turn on ADAM and bring up CP/M 2.2 if you have not already done so.

Step 2

- Type BACKUP. Press return. The onscreen will appear your instructions:

```
ADAM CP/M 2.2 BACK UP UTILITY
BACK UP TO DRIVE
(A, B, C, or D)?
```

Step 3

- Type in the letter label for the drive in which you will place your source data pack. Press return. In this case, you want to back up a data pack from Drive A onto a blank data pack that will be Drive A (since you are working with one drive). Press return.

Step 4

- The following onscreen message appears:

```
INSERT SOURCE MEDIA
PRESS RETURN WHEN READY
```

Put the data pack or disk you wish to back up in your first drive, Drive A. Press return. CP/M 2.2 tells you the stages of the backup process right on the screen.

```
READING          000
```

CP/M 2.2 will read up to 48 blocks of your "source media," the data pack or disk that you wish to back up. When the instruction says READING 047, CP/M 2.2 gives you new instructions:

```
INSERT TARGET MEDIA
PRESS RETURN WHEN READY
WRITING 000
```

After you insert the target media and press return, CP/M 2.2 writes the first 48 blocks of your original to the target data pack or disk. When the number reaches 047, CP/M 2.2 repeats the first message.

Step 5

- Continue this operation until the entire data pack or disk is backed up.

Step 6

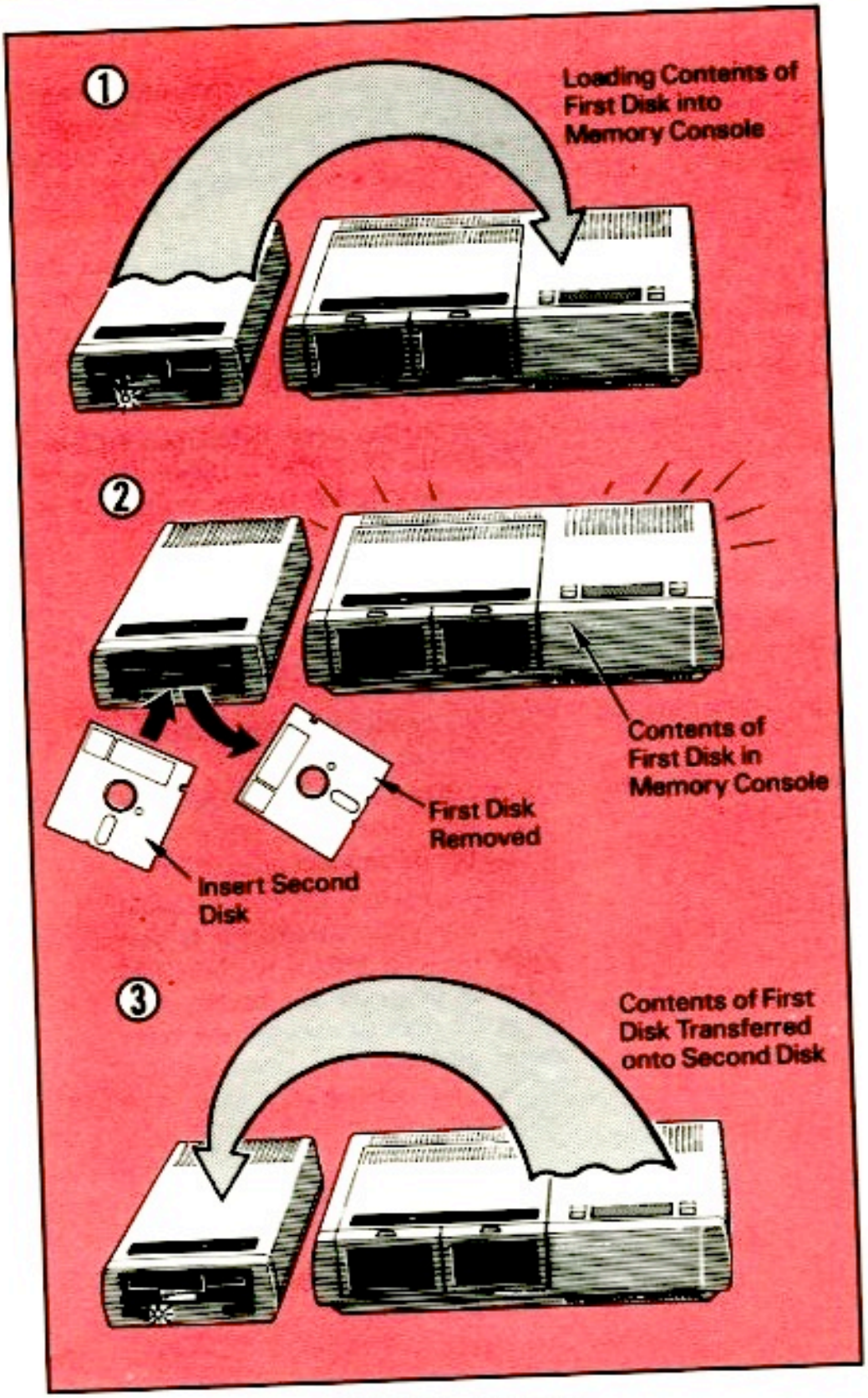
- The backup process takes awhile. This is normal for any backup. You might even have time for a cup of coffee if you want.

Step 7

- When the backup process is finished, CP/M 2.2 gives you the following message:

```
BACKUP ANOTHER
(Y or N)?
```

If you press Y <cr>, the procedure begins again. If you press N <cr>, then you return to the first level of CP/M 2.2.



CP/M 2.2's Back Up

BACKUP With Two Drives

To back up a data pack or disk with two drives, follow steps 1 through 4 above, under BACKUP with One Drive. However, CP/M 2.2 will not prompt you to insert either the source or the target media. It just asks you to press return when you have your target and source in place.

Once you press return to read the information from the source onto the target, CP/M 2.2 goes busily to work until it is finished. You'll see messages that continually update the status of the process. As CP/M 2.2 reads in the contents from the source, the numbers on the screen increase—up to 255 on data packs.

As the contents are written to the target drive, you will see a message on the screen "WRITING 000." The numbers increase as CP/M 2.2 writes and counts how much data is written. It is normal for the backup process to take awhile.

Error Messages

When you back up a disk or data pack with CP/M 2.2, you might receive one of the following error messages.

ERROR IN READING DRIVE X	CP/M 2.2 cannot read the drive you have indicated.
ERROR IN WRITING TO DRIVE X	CP/M 2.2 cannot write to the drive you have indicated. Have you typed the drive label correctly?
Abort Retry or Continue (A, R, or C)	For some reason, CP/M 2.2 cannot carry out your command. If you abort (A), you return to the CP/M 2.2 main program. If you retry (R), CP/M 2.2 tries to execute the command again. If you continue (C), CP/M 2.2 continues with the operation.

Target Drive Smaller Than Source Drive	The files you are trying to transfer are too big for the data pack or disk you are trying to put them on. Break down the source files and transfer them in smaller units.
Not a Legal CP/M Format	You are trying to copy from or to a disk or data pack that is not CP/M 2.2 formatted. Either format the media (remember that all information presently stored on it will be lost in the format process) or use a media that is already formatted for CP/M 2.2.

THE COPY COMMAND

You now know how to make a backup copy of an entire disk or data pack. But if you only want to make a copy of a particular file, use the COPY command. You can put a copy of a file on another disk or data pack, or put the copy on the same disk or data pack. Copies of files are useful as backups, but copies also have other functions. For example, you might want to create a new file very similar to an old one. Instead of retyping the file, you could copy the old file with a new name, then make the few needed changes. You'd get the new file with a minimum of work.

Make a copy of the file MYFILE.TXT—that's the file that used to be called EXAMPLE.TXT. You renamed it in Part B, Chapter 1.

Copying To The Same Data Pack or Disk

To copy a file on the same disk or data pack:

Step 1

- Type COPY. Leave a space. Type the file name you want to copy, in this case, MYFILE.TXT.

```
A > COPY MYFILE.TXT
```

Step 2

- Then, leave a space and type the name of the copy. Use MYCOPY.TXT. The name of the copy *must* be different than the name of the original file.

```
A > COPY MYFILE.TXT MYCOPY.TXT
```

Step 3

- Press return.
- When CP/M 2.2 finishes copying, the prompt line appears. To check to see that CP/M 2.2 indeed has done so, use the directory command. On the directory you should see both the MYFILE.TXT and MYCOPY.TXT file names.

Copying From One Drive To Another

Step 1

- Make sure the data pack or disk to which you are copying the file is CP/M 2.2 formatted.
- Insert the data pack or disk into the second drive.

Step 2

- Type COPY. Leave a space. Type the name of the file you want to copy. Leave a space. Type the label of the drive into which you have inserted the data pack or disk on which you wish to put the copy of the file. Type a colon. Your entry should look like this:

```
A > COPY MYCOPY.TXT B:
```

- Since you are working with two drives, you do not have to use different file names. The name can be the same as the name of the original file, because they will not be on the same data pack or disk.

Step 3

- Press Return.
- Use DIR to check to see if COPY was a success.

Copying the CP/M 2.2 Operating System

To copy CP/M 2.2 itself requires a special command. Refer to the command SYSGEN in part C, the Reference Manual.

Error Messages

DISK IS FULL FILES CLOSED	Although you want to make a copy, there is no room on your disk.
CANNOT FIND SOURCE	CP/M 2.2 cannot find the file you want to copy. Check for typing errors.
CANNOT CREATE TARGET	CP/M 2.2 cannot make a copy. You may have a faulty disk or data pack.
CANNOT COPY FILE ONTO ITSELF	Use a different file name for the copy.

THE ERA COMMAND

Like a file drawer, a data pack or disk can accumulate outdated, useless files. How can you get rid of them and open up more space for new files? Simply by telling CP/M 2.2 to ERA (II) a file.

Step 1

- Start by pressing ERA (II). As usual, the command appears on the screen. The command line contains two items:
- [DRIVE:] Check the prompt to see which drive you are currently using. If the file you want to erase is in the same drive, you don't need to do anything. If it is in another drive, type the appropriate drive label, see page B23.
- <FILE TO ERASE> Type in the file name of the file you want to erase. For practice, type MYCOPY.TXT. Be sure to include both parts of the file name, with a period between them.

```
A > ERA MYCOPY.TXT
```

Step 2

- Since this is a destructive command, better recheck the file name to make sure this is the file you want to erase. Once you press return to tell CP/M 2.2 to erase this file, there's no getting it back.
- Think twice! Okay? Now, press return.

Step 3

- When CP/M 2.2 is done erasing MYCOPY.TXT, the prompt reappears.
- Do a directory command (See page B8) to see that MYFILE.TXT is removed from the directory listing.

THE STAT COMMAND

An efficient office manager checks to see if the files are crowded and if it's time to order another cabinet. With the STAT command, you can do the same and check if your data pack or disk is getting full. If you need a new "cabinet," you get another disk or data pack. But, if you do not keep tabs on the disk's or data pack's status, you might try to store a file onto an overloaded one. Then, CP/M 2.2 would send you an error message, and you would have to stop what you're doing and load in a new data pack or disk. Or, you would have to erase files on the overloaded one. In either case, you run the risk of losing information in the process.

To avoid getting into this precarious situation, occasionally check how much space is left on the data pack or disk with the STAT command.

Checking Remaining Space

To check how much space is left on a data pack or disk:

Step 1

- With the data pack or disk in Drive A, type STAT and press return.
- ADAM responds with the following message.

```
A > STAT
    A:R/W, SPACE: 144K
A >
```

Understanding the KBYTES Message

What does this message mean? The "A" is no mystery to you now; it's the designation for Drive A. And you know what STAT is, the name of the command you've just given CP/M 2.2. The rest of the message indicates the space remaining: 144K. 144K bytes are left on the data pack or disk.

What does 144K bytes mean to you? One byte holds one character and one K is 1,024 bytes. So, 144K translates to approximately 147,456 bytes or characters. Considering that a double-spaced typewritten page takes up about 2K, that's quite a bit of space.

What about the first part of the message, R/W? You didn't specifically ask for this information, but CP/M 2.2 gives it to you as a standard part of the response to a STAT command. R/W stands for read/write, and indicates that you can get (read) information from the data pack or disk, and put (write) new information on it. The other possible response is R/O, which stands for Read Only. You can only get (read) information from a Read Only disk or data pack: you cannot put information on it or change the information that is already there.

Checking Space Used By A File

You can use a slightly different form of the STAT command to find out how much space is used by one file. For example, to find out how much space the PIP.COM file uses:

Step 1

- With the CP/M 2.2 data pack or disk in Drive A, type STAT.
- Type PIP.COM and press return.
- The command should look like this: `A > STAT PIP.COM`

Step 2

- CP/M 2.2 responds with this message:

```
 RECS  BYTES       EXT  ACC
   62     8K         1  R/W  A:PIP.COM
BYTES  REMAINING   ON    A:  104K
```

What do all these figures and letters mean to you? Well, the columns labeled RECS BYTES and EXT give different measurements of the space used by the file PIP.COM. It's a little like getting the distance between your house and your cousin's in kilometers, miles and by how the crow flies. The BYTES measurements is the easiest to understand.

The BYTES column says that PIP.COM uses 8K. 8K translates to approximately 8192 characters. But one important note to remember: The PIP.COM file might not actually contain 8192 characters, it may contain less. CP/M 2.2 has set aside that much space for the PIP.COM file. It's as if you had an entire file drawer set aside for rent checks, and you are only using a quarter of the space in the

drawer. For efficiency, CP/M 2.2 deals with blocks of space 1K or larger. So even the smallest file will be given 1K of space, never less.

If you want to write protect a data pack, disk or even just a file, you can do so with CP/M 2.2. Read the relevant sections in Part C.

Writing to a Protected File

If you try to write to a protected file, data pack or disk, CP/M 2.2 sends the following message:

```
Write Protected
Abort or Retry
(A or R)?
```

Take note of this warning. If you have changed your mind and you still want to write, you can continue with your command. The first key you press will tell CP/M 2.2 to change the designation to read/write.

File References

You've now had experience manipulating files with CP/M 2.2. Are you ready for some shortcuts?

Using what you have learned so far, if you wanted to copy most, but not all, of the files on a disk, you would have to do the COPY command over and over again.

But as a quick shortcut, you can use what CP/M 2.2 calls an ambiguous file name to make the job easier. By taking this shortcut, you are using the way CP/M 2.2 finds files to your advantage.

Your CP/M 2.2, operating inside your system, finds a file by matching the file name you type in, character for character, with a file name on the directory of the data pack or disk.

For short cuts, you can take advantage of the CP/M 2.2's file managing method. With CP/M 2.2, you can substitute some special symbols for some of the characters in the file name. The special symbols are like wild cards—they match any character or group of characters. By adding these symbols, you are creating an ambiguous file name that can refer to several files. You can use the asterisk (*) or the question mark (?) to do this.

The asterisk takes the place of the whole (or part of a) primary file name or the whole file type. An ambiguous file name of * * matches every file in the directory.

The file directory might look like the illustration below:

```
PIP     COM  STAT    COM
DDT     COM  SC      COM
HELP    TXT  HELP1   TXT
HELP2   TXT  HELP15  TXT
SC      OVL
```

The File Directory Screen.

So if you wanted to copy all the command files, you would use the file name *.COM. This ambiguous file will match every file with a file type of COM. Likewise, SC.* matches every file with the primary name of SC.

A question mark stands for any single character in the same position as the ?.

Say, for example, you wanted all COM files that had only three characters or less in the primary file name. The ambiguous file name ???.COM would match all these files. By checking the directory you can see that the files included in this ambiguous file would be PIP.COM and DDT.COM.

If the ambiguous file name was HELP?.TXT, the files in the directory that would match are:

```
HELP   TXT
HELP1  TXT
HELP2  TXT
```

All of these files start with HELP, followed by one other character and a file type of TXT.

HELP15.TXT does not match because of the "5" in the file name. To cover that file, you would have to do HELP??.TXT.

WARNING:

Ambiguous file names can be the most destructive of destructive commands!

If given this command. . . . . 

. . . . . CP/M 2.2 would dutifully erase every COM file on the disk or data pack, probably smiling cheerfully to itself over a job well done.

How NOT to use Ambiguous Files!

Error Messages

NO FILE FOUND	Check command line for typing errors. Retype command.
INVALID ASSIGNMENT	Check your file name or your drive indication. You might have misspelled the file name.
INVALID DISK ASSIGNMENT	This error message might appear if you type in anything after the drive specification. You can type in R/O, but nothing else is valid.

THE ADAM COMMAND

You might wish to use old ADAM SmartWriter data packs or disks with a word processing program that uses the CP/M 2.2 operating system. Or you might wish to use your BASIC programs with a CP/M based BASIC. You can still use these data files by converting them to CP/M 2.2 data files.

Step 1

- Insert the ADAM data pack with the files that you wish converted to CP/M 2.2 into a drive.

Step 2

- Type in ADAM. Press Return. Onscreen appear the following instructions:

```
ADAM to CP/M FILE COPY UTILITY
ADAM TAPE/DISK IS DRIVE
(A, B, C, D) ?
```

Step 3

- Type in the letter that corresponds to the drive in which you have the ADAM data pack or disk.
- CP/M 2.2 checks ADAM data pack or disk and displays the directory.
- A new instruction appears:

```
TRANSFER ADAM FILE?
```

Step 4

- Type in the name of the file you wish to convert to a CP/M 2.2 file. (Remember, the original ADAM file will remain intact on its own data pack or disk.)

```
ADAM FILE TYPE
(H, h, A, or a)?
```

- More instructions appear onscreen.

```
NAME CP/M FILE ?
```

Step 5

- Type in the name that you wish the converted ADAM file to have in CP/M 2.2. Press Return. Then type in the file type. If the ADAM file is a SmartBasic file, type in A (unless you want the earlier version of the same file—the backup, then type a). If the ADAM file is a word processing file, a letter you typed or a termpaper, type in H (unless you want the earlier version of the same file, the backup, then type h).

Step 6

- After the computer finishes converting your files, a new instruction appears:

```
ANOTHER FILE TO TRANSFER?
(Y or N)?
```

- If you type Y <cr>, then the computer begins the process over again, starting with step 1. If you type N <cr>, the computer gives you a new prompt line.
- When the old ADAM files are called into CP/M 2.2, you might notice some unrecognizable characters—garbage. These characters *were used* to tell SmartWriter what margins, spacing, and tabs the file required. Now that you've converted the file to CP/M 2.2, these characters are unnecessary and can be deleted.

CPMADAM COMMAND

Sometimes, you might wish to use files that you created under the CP/M 2.2 operating system with ADAM's SmartWriter. The CPMADAM utility does just that. It translates files created under the CP/M 2.2 operating system to be accessible by SmartWriter or by SmartBasic. Note, however, that any CP/M 2.2 file that has special codes in it will appear with "garbage" instead of the special codes once you do the conversion.

Step 1

- Type in CPMADAM. Press Return. Onscreen appear the following instructions:

```
CP/M to ADAM FILE COPY UTILITY
ADAM TAPE/DISK IS DRIVE
(A, B, C, D) ?
```

Step 2

- Insert into a drive the ADAM data pack that you wish to receive the CP/M 2.2 file. Type in the letter of that drive. Press return. More instructions appear onscreen.

Step 3

- Type in the CP/M 2.2 file name. Press Return.

Step 4

- Type in the name you wish the file to have as a CP/M 2.2 file. Press Return. CPMADAM transfers your file as an A-type—an ASCII file. If you want the file to be an a-type (a backup ASCII), an H-type (program), or an h-type (a backup program) file, then type the file type in square brackets like this: [H] directly behind the file name.

Step 5

- CP/M 2.2 asks if you want to transfer another file. You respond by typing Y or N, then pressing return.

Error Messages

You may receive any of the following error messages while you are doing ADAM or CPMADAM commands.

CANNOT FIND ADAM FILE	The file you named is not on the ADAM disk or data pack.
ERROR READING DRIVE X Abort or Retry (A or R)?	The drive may have a faulty connection.
DRIVE IS FULL	No more data can be sent to this drive because the data pack or disk is full.
CANNOT CLOSE CPM FILE	CP/M 2.2 cannot finish converting your file. Use another data pack or disk and begin again.
INSUFFICIENT DRIVE OR DIRECTORY SPACE	The conversion of files cannot be completed because of storage limitations.
ERROR WRITING TO DRIVE X	There may be a bad block or sector on the data pack or disk that you are trying to write to. Try another.
CP/M FILE NOT FOUND	The file you named is not on the CP/M 2.2 disk or data pack. Check for typing mistakes.
NO RECORDS IN CP/M FILE	ADAM found the file, but there's nothing in it.
ADAM TAPE/DISK OR DIRECTORY FULL	There is no more room on the ADAM data pack or disk. Try another one.

The following three commands are mentioned here for consistency, but fuller discussions are detailed in Part C.

CONFIG COMMAND

CONFIG is a command that you can play with to change parameters such as the screen color, the cursor color and shape, the color of the Smart Keys. But you can also use it to modify the CONTROL Key plus special key (↑, ↓, Command Keys, etc.) functions, the baud rate—the rate at which signals are sent—and the text of Smart Key labels. If you want to change parameters, turn to Section 1, Part C, and follow the syntax provided.

SAVE COMMAND

The SAVE (V) command is a primitive method of storing the contents of the memory. You will rarely use SAVE unless you are assembly language programming. See Part C.

USER COMMAND

The USER (VI) command lets you subdivide your data pack or disk into sections. The USER command allows you to give your files a unique user ID—a number between 0 and 15 that you have chosen. All files that you save can be given this ID. See Part C for instructions.

Summary

Now that you have completed this section, you know several helpful utilities and instructions. Here's a summary of the commands that have been covered.

- **CONTROL C** tells CP/M 2.2 that you've put in a new disk or data pack. It also "warm boots" the system.
- You can tell CP/M 2.2 which drive to address by typing the drive label and a colon, then pressing return.
- **FORMAT** lets you take an empty data pack or disk and make it ready to work with the CP/M operating system.
- **BACKUP** lets you make a backup data pack or disk.
- **COPY** lets you make a copy of one file.
- **ERASE** lets you get rid of a file that you don't need.
- **STAT** tells you how much room you have left on your data pack or disk.
- **ADAM** lets you take ADAM files and convert them to CP/M 2.2 files.
- **CPMADAM** lets you take CP/M 2.2 files and convert them to ADAM files.
- The purpose of the two commands **SAVE** and **USER** has been explained. Refer to Part C for a full discussion.
- **CONFIG** lets you recognize the parameters of CP/M 2.2. Refer to Part C for a full discussion.

Selected Bibliography for Further Reading

The books recommended below are suggested as references that take you beyond this beginning guide and into CP/M. The short list is just to get you started. You will find an array of helpful books on the market.

Dennon, Jack D. **CP/M Revealed.** Rochelle Park, New Jersey: Hayden Book Company, Inc. 1982.

Hogan, Thom. **Osborne CP/M® User Guide Second Edition.** Berkeley, California: Osborne/McGraw-Hill, 1982.

Zaks, Rodnay. **The CP/M Handbook with MP/M.** SYBEX, Inc. 1982.

And to learn ASSEMBLY LANGUAGE PROGRAMMING . . .

Short, Kenneth L. **MicroProcessors and Programmed Logic.** Englewood Cliffs, NJ: Prentice Hall, 1981.

PART C:
REFERENCE MANUAL

Section 1
CP/M Features and Facilities

1.1 Introduction

CP/M® is a monitor control program for microcomputer system development. It provides an environment for program construction, storage, and editing, along with assembly and program checkout facilities.

CP/M on the ADAM™ is a 56 Kilobyte system, supports 2 Digital Data Pack Drives, 5-1/4″ disk drives, a RAM disk (64K memory expander) and parallel or serial ports (through a RS-232 interface).

The CP/M monitor provides rapid access to programs through a comprehensive file management subsystem. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file subsystem, a large number of programs can be stored in both source and machine-executable form.

CP/M 2.2 is a high-performance operating system that uses table-driven techniques.

Features of CP/M 2.2 on ADAM include user specification of one to five logical drives, each containing up to 256k bytes. Any particular file can reach the full drive size. The directory can contain up to 64 entries, and each file can be tagged with Read-Only and system attributes. Users of CP/M 2.2 are physically separated by user numbers. Files can be copied from one user area to another. Powerful relative-record random access functions provide direct access to any of the 65536 records.

CP/M supports ED, a powerful context editor, ASM™ , an Intel-compatible assembler, and DDT™, debugger subsystems.

CP/M is logically divided into several distinct parts:

- BIOS (Basic I/O System), hardware-dependent
- BDOS (Basic Disk Operating System)
- CCP (Console Command Processor)
- TPA (Transient Program Area)

The BIOS provides the primitive operations necessary to access the disk and data pack drives and to interface standard peripherals: RS232 serial, keyboard, video, and user-defined peripherals. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies that provide fully dynamic file construction while minimizing head movement across the disk during access. The BDOS has entry points that include the following primitive operations, which the program accesses:

- SEARCH looks for a particular file by name.
- OPEN opens a file for further operations.
- CLOSE closes a file after processing.
- RENAME changes the name of a particular file.
- READ reads a record from a particular file.
- WRITE writes a record to a particular file.
- SELECT selects a particular drive for further operations.

The CCP provides a symbolic interface between the console and the remainder of the CP/M system. The CCP reads the console device and processes commands, which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers.

The TPA, Transient Program Area, holds programs that are loaded from the disk or data pack under the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

Any CP/M subsystem can be overlaid by an executing program. That is, once a user program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A bootstrap loader is accessible to a program if the BIOS portion is not overlaid. Thus, the program can branch to the bootstrap loader at the end of execution and reload the complete CP/M monitor from disk or data pack.

Because the CP/M operating system is partitioned into distinct modules, it is easily modified to any nonstandard environment by changes to the peripheral drivers.

1.2 Functional Description

You interact with CP/M primarily through the CCP, which reads and interprets commands entered through the console. The CCP can address up to four different

drives, labeled A, B, C, D, and the RAM disk, M. A disk or data pack is logged-in if the CCP is currently addressing the drive. To indicate which drive is the currently logged drive, the CCP prompts the operator with the drive name followed by the symbol >, indicating that the CCP is ready for another command. The CP/M system is loaded from drive A, and the CCP displays the following message:

```
ADAM™ CP/M®
CP/M 2.2
(c) Digital Research 1976, 1984
Welcome to CP/M
Wait till prompt appears


A>
```

CP/M automatically logs in drive A, prompts you with A>, indicating that CP/M is currently addressing drive A, and waits for a command. Commands are implemented at two levels: built-in commands and transient commands.

1.2.1 General Command Structure

Built-in commands are a part of the CCP program. Transient commands are utility programs that are loaded into the TPA from disk or data pack and executed. The following are built-in commands which appear on the Smart Key labels.

- DIR lists filenames in the directory. (I)
- ERA erases specified files. (II)
- REN renames the specified file. (III)
- SAVE saves memory contents in a file. (V)
- TYPE types the contents of a file on the logged disk or data pack. (VI)

The Smart Keys also include USER (IV) which allows you to change the current user number.

1.2.2 File References

A file reference identifies a particular file or group of files on a disk or data pack attached to CP/M. These file references are either unambiguous (ufn) or ambiguous (afn). An unambiguous file reference uniquely identifies a single file. An ambiguous file reference is satisfied by a number of different files.

File references consist of two parts: the primary filename and the filetype. The filetype is optional, and is usually generic. For example, the filetype ASM denotes that the file is an assembly language source file. The primary filename distinguishes each particular ASM source file. The two names are separated by a period, as shown in the following example:

```
filename.typ
```

In this example, `filename` is the primary filename of eight characters or less, and `typ` is the filetype of three characters or less. As mentioned above, the name

```
filename
```

is also allowed. It has a filetype consisting of three blanks.

The characters used in specifying an unambiguous file reference cannot contain any of the following special characters:

```
< > . , ; : = ? *  _ \
```

All alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol ? can be interspersed throughout the primary and secondary names. The ? symbol matches any character of a filename in the ? position. Thus, the ambiguous reference

```
X?Z.C?M
```

matches the following unambiguous filenames

```
XYZ.COM
```

and

```
X3Z.CAM
```

In an ambiguous reference, the * character replaces all or part of a filename or filetype. Therefore:

```
* *
```

equals the ambiguous file reference

```
????????.???
```

while

```
filename *
```

and

```
* typ
```

are abbreviations for

```
filename.???
```

and

```
????????.typ
```

respectively. As an example,

```
A>DIR *.*
```

is interpreted by the CCP as a command to list the names of all files in the directory. The following example searches only for a file by the name X.Y

```
A>DIR X.Y
```

Similarly, the command

```
A>DIR X?Y.C?M
```

causes a search for all unambiguous filenames on the disk that satisfy this ambiguous reference.

The following file references are valid unambiguous file references:

```
X
X.Y
XYZ
XYZ.COM
GAMMA
GAMMA.1
```

As an added convenience, you can generally specify the drive name along with the filename. In this case, the drive name is given as A, B, C, D, or M followed by a colon (:). The specified drive is then logged-in before the file operation occurs. Thus, the following are valid file references with drive name prefixes:

```
A:X.Y
D:XYZ.COM
B:XYZ
B:X.A?M
C:GAMMA
M:*.ASM
```

NOTE: All alphabetic lower-case letters in file and drive names are translated to upper-case when they are processed by the CCP. Thus, lower-case alphabetics are treated as if they are upper-case in command names and file references.

1.3 Switching Disks and Digital Data Packs

You can switch the currently logged drive by typing the drive name, A, B, C, D, or M followed by a colon when the CCP is waiting for console input. The following sequence of prompts and commands can occur after the CP/M system is loaded from drive A:

```
CP/M VER 2.2
A>DIR                          List all disk or data pack files on drive A.
A:SAMPLE ASM SAMPLE PRN
A>B:                           Switch to drive B.
B>DIR *.ASM                    List all ASM files on B.
B:DUMP ASM : FILES ASM
B>A:                           Switch back to drive A.
```

1.4 Built-in Commands

The following abbreviations are used in the description below, of the built-in commands:

ufn	unambiguous file reference
afn	ambiguous file reference

1.4.1 ERA Command

Syntax:

ERA afn

The ERA (erase) command removes files from the currently logged-in disk or data pack. The files that are erased are those that satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

`ERA X.Y`	The file named **X.Y** on the currently logged disk or data pack is removed from the disk directory and the space is returned.
`ERA X.*`	All files with primary name **X** are removed from the current directory.
`ERA *.ASM`	All files with secondary name **ASM** are removed from the current directory.

`ERA X?Y.C?M`	All files on the current directory that satisfy the ambiguous reference `X?Y.C?M` are deleted.
`ERA *.*`	Erases all files on the current directory. In this case, the CCP prompts the console with the second chance message `ALL FILES (Y/N)?` which requires a `Y` response before files are actually removed.
`ERA b:*.PRN`	All files on drive B that satisfy the ambiguous reference `????????.PRN` are deleted, independently of the currently logged drive.

1.4.2 DIR Command

Syntax:

DIR afn

The DIR (directory) command causes the names of all files that satisfy the ambiguous filename afn to be listed on the console device. As a special case, the command

```
DIR
```

lists the files on the currently logged disk or data pack (the command `DIR` is equivalent to the command `DIR *.*`). The following are valid DIR commands:

```
DIR X.Y
DIR X?Z.C?M
DIR ??.Y
```

Similar to other CCP commands, the afn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place:

```
DIR B:
DIR B:X.Y
DIR B:*.A?M
```

If no files on the selected disk or data pack satisfy the directory request, the message `NO FILE` appears at the console. (If you have an ADAM SmartBASIC™ tape or SmartWRITER™ files on a disk or digital data pack, CP/M gives a `NO FILE` message.) This is also true for supergame digital data packs.

1.4.3 REN Command

Syntax:

REN ufn1=ufn2

The REN (rename) command allows you to change the names of files. The file satisfying ufn2 is changed to ufn1. The currently logged disk or data pack is assumed to contain the file to rename (ufn2). The following are examples of the REN command:

`REN X.Y=Q.R`	The file `Q.R` is changed to `X.Y`.
`REN XYZ.COM=XYZ.XXX`	The file `XYZ.XXX` is changed to `XYZ.COM`.

You can precede either ufn1 or ufn2 (or both) by an optional drive address. If ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to exist on the drive as well. If both ufn1 and ufn2 are preceded by drive names the same drive must be specified for both. The following REN commands illustrate this format:

`REN X.ASM=Y.ASM`	The file `Y.ASM` is changed to `X.ASM`
`REN ZAP.BAS=ZOT.BAS`	The file `ZOT.BAS` is changed to `ZAP.BAS`
`REN A.ASM=A.BAK`	The file `A.BAK` is renamed to `A.ASM`

If ufn1 is already present, the REN command responds with the error `FILE EXISTS` and does not perform the change. If ufn2 does not exist on the specified disk or data pack, the message `NO FILE` is printed at the console device.

1.4.4 SAVE Command

Syntax:

SAVE n ufn

The SAVE command places n pages (256-byte blocks) onto disk or data pack from the TPA and names this file ufn. In the CP/M distribution system, the TPA starts at 100H (hexadecimal) which is the second page of memory. The SAVE command must specify 2 pages of memory if the user's program occupies the area from 100H through 2FFH. The machine code file can be subsequently loaded and executed. The following are examples of the SAVE command:

`SAVE 3 X.COM`	Copies 100H through 3FFH to `X.COM`.
`SAVE 40 Q`	Copies 100H through 28FFH to `Q`. Note that 28 is the page count in 28FFH, and that 28H = 2*16+8=40 decimal.
`SAVE 4 X.Y`	Copies 100H through 4FFH to `X.Y`.

SAVE allows you to use illegal characters in the file name, but these may not be accepted by other programs. The SAVE command can also specify a disk drive in the ufn portion of the command, as shown in the following example:

`SAVE 10 B:ZOT.COM`	Copies 10 pages, 100H through 0AFFH, to the file `ZOT.COM` on drive B.

1.4.5 TYPE Command

Syntax:

TYPE ufn

The TYPE command displays, on the console device, the content of the ASCII source file ufn on the currently logged disk or data pack. The following are valid TYPE commands:

```
TYPE X.PLM
TYPE XXX
```

The TYPE command expands tabs, CTRL-I characters, assuming tab positions are set at every eighth column. The ufn can also reference a drive name.

`TYPE B:X.PRN`	The file `X.PRN` from drive B is displayed.

1.4.6 USER Command

Syntax:

USER n

The USER command allows maintenance of separate files in the same directory. In the syntax line, n is an integer value in the range 0 to 15. On cold start, you are automatically logged into user area number 0. You can issue the USER command at any time to move to another logical area within the same directory. Drives that are logged-in while addressing one user number are automatically active when you move to another. A user number is simply a prefix that accesses particular directory entries on the active disks.

The active user number is maintained until changed by a subsequent USER command, or until a cold start when user 0 is again assumed.

1.5 Line Editing and Output Control

The CCP allows certain line-editing functions while typing command lines. The CTRL-key sequences are obtained by holding down the control key, then pressing the appropriate letter key. CCP command lines are up to 255 characters in length. They are not executed until the RETURN key is pressed.

1-1. Line-editing Control Characters

Character	Meaning
CTRL-C	Reboots CP/M system when pressed at start of line.
CTRL-E	Physical end of line; carriage is returned, but line is not sent until the carriage return key is pressed.
CTRL-H	Backspaces one character position.
CTRL-J	Terminates current input (line feed).
CTRL-L	Clears ADAM's screen.
CTRL-M	Terminates current input (carriage return).
CTRL-P	Copies all subsequent console output to the currently assigned list device (usually the SmartWRITER printer). Output is sent to the list device and the console device until the next CTRL-P is pressed, at which point, output to the list device is cancelled.
CTRL-R	Retypes current command line; types a clean line following character deletion with rubouts.
CTRL-S	Stops the console output temporarily. Program execution and output continue when you press any character at the console, for example another CTRL-S. This feature stops output on high speed consoles, such as CRTs.
CTRL-U	Deletes the entire line typed at the console.
CTRL-V	Erases Smart Key labels, but Smart Keys remain active.
CTRL-X	Same as CTRL-U.
CTRL-Z	Ends input from the console (used in PIP and ED).
BACKSPACE	Deletes the last character typed at the console.
Shift + UNDO	Toggles the Smart Key labels on and off and cancels their activity.

1.6 Transient Commands

Transient commands are utility programs that are loaded from the currently logged disk or data pack and executed in the TPA. The transient commands (utility programs) that come with CP/M 2.2 are listed below. You can easily define additional functions (see Section 1.6.3).

Table 1-2. CP/M Transient Commands (Standard)

Command	Function
STAT	Lists the number of bytes of storage remaining on the currently logged disk or data pack, provides statistical information about particular files, and displays or alters device assignment.
ASM	Loads the CP/M assembler and assembles the specified program from disk or data pack.
LOAD	Loads the file in Intel HEX machine code format and produces a file in machine executable form which can be loaded into the TPA. This loaded program becomes a new command under the CCP.
DDT	Loads the CP/M debugger into TPA and starts execution.
PIP	Loads the Peripheral Interchange Program for subsequent file and peripheral transfer operations.
ED	Loads and executes the CP/M text editor program.
SUBMIT	Submits a file of commands for batch processing.
DUMP	Dumps the contents of a file in hex.
SYSGEN	Writes the operating system to a user-specified drive.

Table 1-3. Transient ADAM Utilities

Utility	Function
COPY	Loads COPY program for copying files.
FORMAT	Loads FORMAT program for preparing disks and digital data packs for use with CP/M.
BACKUP	Duplicates disks on data packs.
ADAM	Loads the program which lets you convert EOS files to CP/M files.
CPMADAM	Loads the program which lets you convert CP/M files to EOS files.
CONFIG	Loads the program which lets you change the color of Smart keys, the ASCII values of the editing keys, and the communications parameters.

Transient commands and transient ADAM utilities are specified in the same manner as built-in commands. Additional commands are easily defined. For convenience, a transient command can be preceded by a drive name which causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

```
B:STAT
```

causes CP/M to temporarily log in drive B for the source of the STAT transient, and then return to the original logged drive for subsequent processing.

1.6.1 STAT Command

Syntax:

STAT
STAT "command line"

The STAT command provides statistical information about file storage and device assignment. Special forms of the command line allow the current device assignment to be examined and altered. The various command lines that can be specified are:

`STAT`

Calculates the storage remaining on all active drives, and prints one of the following messages:

```
d: R/W, SPACE: nnnK
```

```
d: R/O, SPACE: nnnK
```

for each active drive d:, where R/W indicates the drive can be read or written, and R/O indicates the drive is Read-Only (a drive becomes R/O if explicitly set to Read-Only, as shown below, or if disks are inadvertently changed without performing a warm start). The space remaining on the disk or data pack in drive d: is given in kilobytes by nnn.

`STAT d:`

If a drive name is given, then the drive is selected before the storage is computed. Thus, the command `STAT B:` could be issued while logged into drive A, resulting in the message

```
BYTES REMAINING ON B: nnnK
```

`STAT afn`

The command line can also specify a set of files to be scanned by STAT. The files that satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading:

```
RECS BYTES EXT D:FILENAME.TYP
rrr bbbK ee d:filename.typ
```

where rrrr is the number of 128-byte records allocated to the file, bbb is the number of kilobytes allocated to the file (bbb=rrrr*128/1024), ee is the number of 16K extensions (ee=bbb/16), d: is the drive name containing the file, filename is the eight-character primary filename, and typ is the three-character filetype. After the list of the individual files, the storage usage is summarized.

`STAT d:afn`

The specified drive is first selected, and STAT afn is executed.

`STAT d:=R/O` This form sets the drive given by d to Read-Only, remaining in effect until the next warm or cold start takes place. When a disk or data pack is Read-Only, the message

```
BDOS ERR ON d:  READ-ONLY
```

appears if there is an attempt to write to the Read-Only disk. CP/M waits until a key is pressed before performing an automatic warm start; then the disk becomes R/W.

The STAT command allows you to control the physical-to-logical device assignment. See the IOBYTE function described in Sections 5 and 6. There are four logical peripheral devices that are, at any particular instant, each assigned one of several physical peripheral devices. The following is a list of the four logical devices:

- CON: is the system console device, used by CCP for communication with the operator. Defaults to keyboard for input, 9928 video for output.
- RDR: ADAM's input default to the RS-232.
- PUN: ADAM's input default to the RS-232.
- LST: is the output list device. Defaults to the SmartWRITER™ printer.

The actual devices attached to any particular computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR device, could actually be a high speed reader, teletype reader, or cassette tape. To allow some flexibility in device naming and assignment, several physical devices are defined in Table 1-4.

Table 1-4. Physical Devices

Device	Meaning
TTY:	Teletype device (slow speed console)
CRT:	Cathode ray tube device (high speed console)
BAT:	Batch processing (console is current RDR, output goes to current LST device)
UC1:	User-defined console
PTR:	Paper tape reader (high speed reader)
UR1:	User-defined reader #1
UR2:	User-defined reader #2
PTP:	Paper tape punch (high speed punch)
UP1:	User-defined punch #1
UP2:	User-defined punch #2
LPT:	Line printer
UL1:	User-defined list device #1

NOTE: The physical device names may not correspond to devices that the names imply. For example, you can implement the PTP device as a cassette write operation. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M.

The command,

```
STAT VAL:
```

produces a summary of the available status commands, resulting in the output:

```
Temp R/O Disk d:$R/O
Set Indicator: filename.typ $R/O $R/W $SYS $DIR
Disk Status: DSK: d:DSK
Iobyte Assign:
```

which gives an instant summary of the possible STAT commands and shows the permissible logical-to-physical device assignments:

```
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

The logical device to the left takes any of the four physical assignments shown to the right. The current logical-to-physical mapping is displayed by typing the command:

```
STAT DEV:
```

This command produces a list with each logical device on the left and the current corresponding physical device on the right. For example:

```
CON: = CRT:
RDR: = UR1:
PUN: = PTP:
LST: = TTY:
```

The current logical-to-physical device assignment can be changed using CONFIG or by typing a STAT command in the form:

```
STAT ld1 = pd1, ld2 = pd2, ... , ldn = pdn
```

where ld1 through ldn are logical device names and pd1 through pdn are compatible physical device names. For example, ld1 and pd1 appear on the same line in the VAL: command shown above. The following example shows valid STAT commands that change the current logical-to-physical device assignments:

```
STAT CON:=CRT:
STAT PUN:=TTY:, LST:=LPT:, RDR:=TTY:
```

The command form,

```
STAT d:filename.typ $S
```

where d: is an optional drive name and filename.typ is an unambiguous or ambiguous filename, produces the following output display:

```
 Size      Recs      Bytes      Ext Acc

   48        48        6K      1 R/O A:ED.COM
   55        55       12K      1 R/O (A:PIP.COM)
65536       128       16K      2 R/W A:X.DAT
```

where the $S parameter causes the Size field to be displayed. Without the $S, the Size field is skipped, but the remaining fields are displayed. The Size field lists the virtual file size in records. The Recs field sums the number of virtual records in each extent. For files constructed sequentially, the Size and Recs fields are identical. The Bytes field lists the actual number of bytes allocated to the corresponding file. The minimum allocation unit is determined at configuration time; thus, the number of bytes corresponds to the record count plus the remaining unused space in the last allocated block for sequential files. Random access files are given data areas only when written, so the Bytes field contains the only accurate allocation figure. In the case of random access, the Size field gives the logical end-of-file record position and the Recs field counts the logical records of each extent. Each of these extents, however, can contain unallocated holes even though they are added into the record count.

The Ext field counts the number of physical extents allocated to the file. The Ext count corresponds to the number of directory entries given to the file. Depending on allocation size, there can be up to 128K bytes (8 logical extents) directly addressed by a single directory entry. In a special case, there are actually 256K bytes that can be directly addressed by a physical extent.

The Acc field gives the R/O or R/W file indicator, which you can change using the commands shown. The four command forms,

```
STAT d:filename.typ $R/O
STAT d:filename.typ $R/W
STAT d:filename.typ $SYS
STAT d:filename.typ $DIR
```

set or reset various permanent file indicators. The R/O indicator places the file, or set of files, in a Read-Only status until changed by a subsequent STAT command. The R/O status is recorded in the directory with the file so that it remains R/O through intervening cold start operations. The R/W indicator places the file in a permanent Read-Write status. The SYS indicator attaches the system indicator to the file, while the DIR command removes the system indicator. The filename.typ may be ambiguous or unambiguous. Files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by d: is optional.

When a file is marked R/O, subsequent attempts to erase or write into the file produce the following BDOS message at your screen:

```
BDOS Err on d: File R/O
```

STAT DSK: lists the characteristics of the default drive (usually A). The drive characteristics of the disk drive are listed in the following format.

```
A>stat dsk:

   A:  Drive Characteristics

1176:  128 Byte Record Capacity
 147:  Kilobyte Drive Capacity
  64:  32 Byte Directory Entries
  64:  Checked Directory Entries
 128:  Records/ Extent
   8:  Records/ Block
   8:  Sectors/ Track
  13:  Reserved Tracks
```

STAT [D:] DSK: lists the drive characteristics of the disk named by d: The drive characteristics of the digital data drive are listed in the following format:

```
A>stat c: dsk:

   C:  Drive Characteristics
1944:  128 Byte Record Capacity
 243:  Kilobyte Drive  Capacity
  64:  32 Byte  Directory Entries
  64:  Checked  Directory Entries
 128:  Records/ Extent
   8:  Records/ Block
   8:  Sectors/ Track
  13:  Reserved Tracks
```

where d: is the selected drive, followed by the total record capacity (65536 is an eight-megabyte drive), followed by the total capacity listed in kilobytes. The directory size is listed next, followed by the checked entries. The number of checked entries is usually identical to the directory size for removable media, because this mechanism is used to detect changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, because the media are not changed without at least a cold or warm start.

The number of records per extent determines the addressing capacity of each directory entry (1024 times 128 bytes, or 128K in the previous example). The number of records per block shows the basic allocation size (in the example, 128 records/block times 128 bytes per record, or 16K bytes per block). The number of physical sectors per track and the number of reserved tracks are displayed.

For logical drives that share the same physical disk, the number of reserved tracks can be quite large because this mechanism is used to skip lower-numbered disk areas allocated to other logical disks. The command form

```
STAT DSK:
```

produces a drive characteristics table for all currently active drives. The final STAT command form is

```
STAT USR:
```

which produces a list of the user numbers that have files on the currently addressed disk. The display format is

```
Active User:   0
Active Files:  0 1 3
```

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In this case, the active user number is 0 (default at cold start) with three user numbers that have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging in with USER 1 or USER 3 commands, followed by a DIR command at the CCP level.

1.6.2 ASM Command

Syntax:

ASM ufn

The ASM command loads and executes the CP/M 8080 assembler. The ufn specifies a source file containing assembly language statements, where the filetype is assumed to be ASM and is not specified. The following ASM commands are valid:

```
ASM X
ASM GAMMA
```

The two-pass assembler is automatically executed. Assembly errors that occur during the second pass are printed at the console.

The assembler produces a file:

X.PRN

where X is the primary name specified in the ASM command. The PRN file contains a listing of the source program with embedded tab characters if present, along with the machine code generated for each statement and any diagnostic error messages. The PRN file is listed at the console using the TYPE command, or sent to a peripheral device using PIP (see Section 1.6.4). Note that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns; for example, program addresses and hexadecimal machine code. The PRN file serves as a backup for the original source file. If the source file is accidentally removed or destroyed, the PRN file can be edited by removing the leftmost 16 characters of each line (see Section 2). This is done by issuing a single editor macro command. The resulting file is identical to the original source file and can be renamed from PRN to ASM for subsequent editing and assembly. The file

X.HEX

is also produced, which contains 8080 machine language in Intel HEX format, suitable for subsequent loading and execution.

The source file for assembly is taken from an alternate disk by prefixing the assembly language filename by a disk drive name. The command

ASM B:ALPHA

loads the assembler from the currently logged drive and processes the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

1.6.3 LOAD Command

Syntax:

LOAD ufn

The LOAD command reads the file ufn, which is assumed to contain HEX format machine code, and produces a memory image file that can subsequently be executed. The filename ufn is assumed to be of the form:

X.HEX

and only the filename X needs to be specified in the command.
The LOAD command creates a file named

```
X.COM
```

that marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the filename X immediately after the prompting character > printed by the CCP.

Generally, the CCP reads the filename X following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system directory for a file by the name

```
X.COM
```

If the filename is found, the machine code is loaded into the TPA, and the program executes. You only need to LOAD a hex file once. It can then be executed any number of times by typing the primary name. This way, you can invent new commands in the CCP. Initialized disks contain the transient commands (utility programs) as COM files, which are optionally deleted. The operation takes place on an alternate drive if the filename is prefixed by a drive name. Thus,

```
LOAD B:BETA
```

brings the LOAD program into the TPA from the currently logged disk or data pack and operates on drive B after execution begins.

NOTE: The file being LOADed must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) that begin at 100H of the TPA. The addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard COM files that operate in the TPA. Programs that occupy regions of memory other than the TPA are loaded using DDT.

1.6.4 PIP

Syntax:

PIP
PIP destination=source#1, source#2, ..., source #n

PIP is the CP/M Peripheral Interchange Program that implements the basic media conversion operations necessary to load, print, copy, and combine disk and data pack files. The PIP program is initiated by typing one of the following forms:

```
PIP
PIP command line
```

In both cases PIP is loaded into the TPA and executed. In the first form, PIP reads command lines directly from the console, prompted with the * character, until an empty command line is typed (for example, you type a single carriage return). Each successive command line causes some media conversion to take place according to the rules shown below.

In the second form of the PIP command, the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

```
destination = source#1, source#2, ..., source#n
```

where destination is the file or peripheral device to receive the data, and source#1, ..., source#n is a series of one or more files or devices that are copied from left to right to the destination.

When multiple files are given in the command line (for example, n>1), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (CTRL-Z) at the end of each file (see the O parameter to override this assumption). Lower-case ASCII alphabetics are internally translated to upper-case. The total command line length cannot exceed 255 characters. CTRL-E can be used to force a physical carriage return for lines that exceed the console width.

The destination and source elements are unambiguous references to CP/M source files with or without a preceding drive name. That is, any file can be referenced with a preceding drive name that defines the particular drive where the file can be obtained or stored. When the drive name is not included, the currently logged drive is assumed.

The destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If it already exists, the destination file is removed if the command line is properly formed. It is not removed if an error condition arises. The following command lines are valid inputs to PIP:

`X=Y`	Copies to file `X` from file `Y`, where `X` and `Y` are unambiguous filenames; `Y` remains unchanged.
`X=Y,Z`	Concatenates files `Y` and `Z` and copies to file `X`, with `Y` and `Z` unchanged.
`X.ASM=Y.ASM,Z.ASM`	Creates the file `X.ASM` from the concatenation of the `Y` and `Z.ASM` files.
`NEW.ZOT=B:OLD.ZAP`	Moves a copy of `OLD.ZAP` from drive `B` to the currently logged drive; names the file `NEW.ZOT`.
`B:A.U=B:B.V,A:C.W,D.X`	Concatenates file `B.V` from drive `B` with `C.W` from drive `A` and `D.X` from the logged drive; creates the file `A.U` on drive `B`.

For convenience, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated PIP forms are

```
PIP d:=afn
PIP d1=d2:afn
PIP ufn=d2:
PIP d1:ufn=d2:
```

The first form copies all files from the currently logged drive that satisfy the afn to the same files on drive d, where d = A,B,C,D,M. The second form is equivalent to the first, where the source for the copy is drive d_2, where d_2 = A,B,C,D,M. The third form is equivalent to the command PIP d_1:ufn=d_2:ufn which copies the file given by ufn from drive d_2 to the file ufn on drive d_1. The fourth form is equivalent to the third, where the source disk or data pack is explicitly given by d_2.

The source and destination disks or data packs must be different in all of these cases. If an afn is specified, PIP lists each ufn that satisfies the afn as it as being copied. There is a file with the same name as the destination file, it is removed after successful completion of the copy and replaced by the copied file.

The following PIP commands give examples of valid copy operations:

B:=*.COM	Copies all files that have the file typ COM to drive B from the current drive.
A:=B:ZAP.*	Copies all files that have the primary name ZAP to drive A from drive B.
ZAP.ASM=B:	Same as ZAP.ASM=B:ZAP.ASM
B:ZOT.COM=A:	Same as B:ZOT.COM=A:ZOT.COM
B:=GAMMA.BAS	Same as B:GAMMA.BAS=GAMMA.BAS
B:=A:GAMMA.BAS	Same as B:GAMMA.BAS=A:GAMMA.BAS

PIP allows reference to physical and logical devices that are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The following is a list of logical devices given in the STAT command

```
CON: (console)
RDR: (reader)
PUN: (punch)
LST: (list)
```

while the physical devices are

```
TTY: (console, reader, punch, list) RS232
CRT: (console, list), UC1: (console) Keyboard, video
PTR: (reader), UR1: (reader), UR2: (reader)
PTP: (punch), UP1: (punch), UP2: (punch)
LPT: (list), UL1: (list)
```

The BAT physical device is not included, because this assignment is used only to indicate that the RDR and LST devices are used for console input/output.

The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M and are easily altered through CONFIG. The current physical device mapping is defined by IOBYTE; see Section 6 for a discussion of this function. The destination device must be capable of receiving data, for example, data cannot be sent to the punch, and the source devices must be capable of generating data, for example, the LST device cannot be read.

The following list describes additional device names that can be used in PIP commands.

- NUL: sends 40 nulls (ASCII 0s) to the device.
- EOF: sends a CP/M end-of-file (ASCII CTRL-Z) to the destination device (sent automatically at the end of all ASCII data transfers through PIP).
- INP: is a special PIP input source that can be patched into the PIP program. PIP gets the input data character-by-character, by CALLing location 103H, with data returned in location 109H (parity bit must be zero).
- OUT: is a special PIP output destination that can be patched into the PIP program. PIP CALLs location 106H with data in register C for each character to transmit. Note that locations 109H through 1FFH of the PIP memory image are not used and can be replaced by special purpose drivers using DDT (see Section 4).
- PRN: is the same as LST:, except that tabs are expanded at every eighth character position, lines are numbered, and page ejects are inserted every 60 lines with an initial eject (same as using PIP options [t8np]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (CTRL-Z for ASCII files, and end-of-data for non-ASCII disk files). Data from each device or file are concatenated from left to right until the last data source has been read.

The destination device or file is written using the data from the source files,and an end-of-file character, CTRL-Z, is appended to the result for ASCII files. If the destination is a disk or data pack file, a temporary file is created (with a **$$$** filetype) that is changed to the actual filename only on successful completion of the copy. Files with the extension COM are always assumed to be non-ASCII.

The copy operation can be aborted at any time by pressing any key on the keyboard. PIP responds with the message ABORTED to indicate that the operation has not been completed. If any operation is aborted, or if an error occurs during processing, PIP removes any pending commands that were set up while using the SUBMIT command.

PIP performs a special function if the destination is a disk file with type HEX (an Intel hex-formatted machine code file), and the source is an external peripheral device. The PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records.

When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. When corrective action has been taken, press the return key. PIP attempts another read. You can also enter the record manually with the ED program after the file has been constructed.

PIP allows the end-of-file to be entered from the console if the source file is a logical device. In this case, the PIP program reads the device and monitors the keyboard. If CTRL-Z is typed at the keyboard, the read operation is terminated normally.

The following are valid PIP commands:

```
PIP LST: = X.PRN
```

Copies X.PRN to the LST device and terminates the PIP program.

```
PIP
```

Starts PIP for a sequence of commands. PIP prompts with *

```
*CON:=X.ASM,Y.ASM,Z.ASM
```

Concatenates three ASM files and copies to the CON device.

```
*X.HEX=CON:,Y.HEX,PTR:
```

Creates a HEX file by reading the CON until a CTRL-Z is typed, followed by data from Y.HEX and PTR until a CTRL-Z is encountered.

```
PIP PUN:=NUL:,X.ASM,EOF:,NUL:
```

Sends 40 nulls to the punch device; copies the X.ASM file to the punch, followed by an end-of-file, CTRL-Z, and 40 more null characters.

```
<cr>
```

A single carriage return stops PIP.

You can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). Table 1-4 describes valid PIP parameters.

Table 1-4. PIP Parameters

Parameter	Meaning
B	Block mode transfer. Data is buffered by PIP until an ASCII x-off character, CTRL-S, is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data that can be buffered depends on the memory size of the system. PIP issues an error message if the buffers overflow.
Dn	Deletes characters that extend past column n in the transfer of data to the destination from the character source. This parameter is generally used to truncate long lines that are sent to a narrow printer or console device.
E	Echoes all transfer operations to the console as they are being performed.
F	Filters form-feeds from the file. All embedded form-feeds are removed. The P parameter can be used simultaneously to insert new form-feeds.
Gn	Gets file from user number n (n in the range 0-15).
H	Transfers HEX data. All data is checked for proper Intel hex file format. Nonessential characters between hex records are removed during the copy operation. The console is prompted for corrective action in case errors occur.
I	Ignores :00 records in the transfer of Intel hex format file. The I parameter automatically sets the H parameter.
L	Translates upper-case alphabetics to lower-case.
N	Adds line numbers to each line transferred to the destination, starting at one and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, leading zeroes are included and a tab is inserted following the number. The tab is expanded if T is set.
O	Transfers non-ASCII object files. The normal CP/M end-of-file is ignored.
Pn	Includes page ejects at every n lines with an initial page eject. If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form-feed suppression takes place before the new page ejects are inserted.

(continued)

Table 1-4. PIP Parameters (continued)

Parameter	Meaning
Qs^Z	Quits copying from the source device or file when the string s, terminated by CTRL-Z, is encountered.
R	Reads system files.
Ss^Z	Start copying from the source device when the string s, terminated by CTRL-Z, is encountered. The S and Q parameters can be used to abstract a particular section of a file, such as a subroutine. The start and quit strings are always included in the copy operation. If you specify a command line after the PIP command keyword, the CCP translates strings following the S and Q parameters to upper-case. If you do not specify a command line, PIP does not perform the automatic upper-case translation.
Tn	Expands tabs, CTRL-I characters, to every nth column during the transfer of characters to the destination from the source.
U	Translates lower-case alphabetics to upper-case during the copy operation.
V	Verifies that data has been copied correctly by rereading after the write operation (the destination must be a disk file).
W	Writes over R/O files without console interrogation.
Z	Zeros the parity bit on input for each ASCII character.

The following examples show valid PIP commands that specify parameters in the file transfer.

```
PIP X.ASM=B:[v]
```

Copies **X.ASM** from drive B to the current drive and verifies that the data was properly copied.

```
PIP LPT:=X.ASM[nt8u]
```

Copies **X.ASM** to the LPT device; numbers each line, expands tabs to every eighth column, and translates lower-case alphabetics to upper-case.

```
PIP PUN:=X.HEX[i],Y.ZOT[h]
```

First copies **X.HEX** to the PUN: device and ignores the trailing :00 record in **X.HEX** continues the transfer of data by reading **Y.ZOT**, which contains HEX records, including any :00 records it contains.

```
PIP X.LIB=Y.ASM[sSUBRI:^z qJMP L3^z]
```

Copies from the file **Y.ASM** into the file **X.LIB**. . The command starts the copy when the string **SUBR1:** has been found, and quits copying after the string **JMP L3** is encountered.

```
PIP PRN:=X.ASM[p50]
```

Sends **X.ASM** to the **LST** device with line numbers, expands tabs to every eighth column, and ejects pages at every 50th line. The assumed parameter list for a **PRN** file is nt8p60; p50 overrides the default value.

Under normal operation, PIP does not overwrite a file that is set to a permanent R/O status. If an attempt is made to overwrite an R/O file, the following prompt appears:

```
DESTINATION FILE IS R/O, DELETE (Y/N)?
```

If you type **Y**, the file is overwritten. Otherwise, the following response appears:

```
** NOT DELETED **
```

The file transfer is skipped, and PIP continues with the next operation in sequence. To avoid the prompt and response in the case of R/O file overwrite, the command line can include the W parameter, as shown in this example:

```
PIP A:=B:*.COM[W]
```

The W parameter copies all nonsystem files to the A drive from the B drive and overwrites any R/O files in the process. If the operation involves several concatenated files, the W parameter need only be included with the last file in the list, as in this example:

```
PIP A.DAT=B.DAT,F:NEW.DAT,G:OLD.DAT[W]
```

Files with the system attribute can be included in PIP transfers if the R parameter is included; otherwise, system files are not recognized. For example, the command line:

```
PIP ED.COM=B:ED.COM[R]
```

reads the ED.COM file from the B drive, even if it has been marked as an R/O and system file. The system file attributes are copied, if present.

To copy files into another user area, PIP.COM must be located in that user area. Use the following procedure to make a copy of PIP.COM in another user area.

`USER 0`	Log in user 0.
`DDT PIP.COM (note PIP size s)`	Load PIP to memory.
`GØ`	Return to CCP.
`USER 3`	Log in user 3.
`SAVE s PIP.COM`	

In this procedure, s is the integral number of memory pages, 256-byte segments, occupied by PIP. The number s can be determined when PIP.COM is loaded under DDT, by referring to the value under the NEXT display. If, for example, the next available address is 1D00, then PIP.COM requires 1C hexadecimal pages, or 1 times 16 + 12 = 28 pages, and the value of s is 28 in the subsequent save. Once PIP is copied in this manner, it can be copied to another disk belonging to the same user number through normal PIP transfers.

1.6.5 ED Command

Syntax:

ED ufn

The ED program is the CP/M system context editor that allows creation and alteration of ASCII files in the CP/M environment. Complete details of ED's operation are given in Section 2. ED allows you to create and operate upon source files that are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage return/line-feed sequence). Line length is defined as the number of characters typed between carriage returns. There is no practical restriction on line length, but no single line can exceed the size of the working memory.

The ED program has a number of commands for character string searching, replacement, and insertion that are useful for creating and correcting programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 51 - 56K), the file size that can be edited is not limited, since data are easily paged through this work area.

ED creates the specified source file and opens the file for access, if it does not exist. If the source file does exist, you can append data for editing (see the A command). The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context, allowing easy access to particular portions of a large file (see the N command).

If you type the following command line:

```
ED X.ASM
```

the ED program creates an intermediate work file with the name

X.$$$

to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original unedited file, and the X.ASM file contains the newly edited file. You can always return to the previous version of a file by removing the most recent version and renaming the previous version. If the current X.ASM file has been improperly edited, the following sequence of commands reclaim the backup file.

`DIR X.*`	Checks to see that **BAK** file is available.
`ERA X.ASM`	Erases most recent version.
`REN X.ASM=X.BAK`	Renames the **BAK** file to **ASM**.

You can abort the edit at any point (reboot, power failure, CTRL-C, or CTRL-Q command) without destroying the original file. In this case, the BAK file is not created and the original file is always intact.

The ED program lets you edit the source on one disk and create the back-up file on another disk. This form of the ED command is

```
ED ufn d:
```

where ufn is the name of the file to edit on the currently logged disk and d is the name of an alternate drive. The ED program reads and processes the source file and writes the new file to drive d using the name ufn. After processing, the original file becomes the back-up file. If you are addressing disk A, the following command is valid.

```
ED X.ASM b:
```

This edits the file X.ASM on drive A, creating the new file X.$$$ on drive B. After a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.$$$ is renamed to B:X.ASM. For convenience, the currently logged disk becomes drive B at the end of the edit. Note that if a file named B:X.ASM exists before the editing begins, the following message appears on the screen:

```
FILE EXISTS
```

This message is a precaution against accidentally destroying a source file. Erase the existing file and then restart the edit operation.

Similar to other transient commands (utility programs), editing can take place on a drive different from the currently logged disk by preceding the source filename with a drive name. The following are examples of valid edit requests:

`ED A:X.ASM`	Edits the file `X.ASM` on drive `A`, with new file and back-up on drive `A`.
`ED B:X.ASM A:`	Edits the file `X.ASM` on drive `B` to the temporary file `X.$$$` on drive `A`. After editing, this command changes `X.ASM` on drive B to `X.BAK` and changes `X.$$$` on drive `A` to `X.ASM`.

1.6.6 SYSGEN Command

Syntax:

SYSGEN

The SYSGEN utility allows the transfer of the CP/M operating system to a formatted disk or data pack. The SYSGEN program prompts the console for commands by interacting as shown.

```
SYSGEN <cr>
```

Initiates the SYSGEN program.

```
SYSTEM GENERATOR
```

SYSGEN sign-on message.

```
Enter Source Drive Name
(Or Return To Reboot)?
```

Respond with the drive name (one of the letters A, B, C, or D) of the data pack or disk containing a CP/M system, usually A.

Place a disk or data pack containing the CP/M operating system on drive d (d is one of A, B, C, or D). Type the drive name and press the return key when ready. System is copied to memory. SYSGEN then prompts with the following:

```
Enter Destination Drive Name
(Or Return To Reboot)?
```

Answer with the drive name. You cannot use the RAM disk (M:) as a destination drive. Otherwise, press the return key and the system reboots from drive A. Typing drive name d causes the above message to be repeated.

Place a formatted disk or data pack into drive d; press the return key when ready. After ADAM writes the system to the disk or digital data pack, the following prompt appears:

```
SYSTEM WRITTEN SUCCESSFULLY
```

or

```
SYSTEM NOT WRITTEN
```

If successful, the new disk is initialized in drive d.

The DESTINATION prompt is repeated until the return key is pressed, so that more than one disk or data pack can be initialized.

Upon completion of a successful system generation, the new disk or data pack contains the operating system and the built-in commands. You must copy the appropriate COM files (utility programs) from an existing CP/M disk to the newly constructed disk using the COPY or PIP utility.

You can copy all files from an existing disk by typing the following PIP command:

```
PIP B: = A:*.*[v]
```

This command copies all files from drive A to drive B and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

A SYSGEN does not destroy the files that already exist on a disk or data pack; it only constructs a new operating system. If a disk is being used only on drives B, C, and D and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place.

SYSGEN can also write the system to an ADAM SmartBASIC™ tape. This destroys the tape for future SmartBASIC use.

1.6.7 SUBMIT Command

Syntax:

SUBMIT ufn parm#1 ... parm#n

The SUBMIT command allows CP/M commands to be batched for automatic processing. The ufn given in the SUBMIT command must be the filename of a file on the currently logged disk or data pack, with an assumed file type of SUB. The SUB file contains CP/M prototype commands. Actual parameters parm#1 ... parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands is processed sequentially by CP/M.

The prototype command file is created using the ED program, with interspersed $ parameters of the form:

```
$1 $2 $3 ... $n
```

corresponding to the number of actual parameters that will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 ... parm#n are paired with the formal parameters $1 ... $n in the prototype commands. If the number of formal and actual parameters do not correspond, the SUBMIT function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

$$$.SUB

on the logged disk or data pack. When the system reboots, at the termination of the SUBMIT, this command file is read by the CCP as a source of input rather than the console. If the SUBMIT function is performed on any disk or data pack other than drive A, the commands are not processed until the disk or data pack is inserted into drive A and the system reboots. You can abort command processing at any time by pressing the shift and delete keys when the command is read and echoed. In this case, the $$$.SUB file is removed and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs that execute under CP/M can abort processing of command files when error conditions occur by erasing any existing $$$.SUB file.

To put dollar signs into a SUBMIT file, type $$ which reduces to a single $ within the command file.

The last command in a SUB file can initiate another SUB file, allowing chained batch commands:

Suppose the file ASMBL.SUB exists on disk or data pack and contains the prototype commands

```
ASM $1
DIR $1.*
ERA *.BAK
PIP $2:=$1.PRN
ERA $1.PRN
```

then, you issue the following command:

```
SUBMIT ASMBL X PRN
```

The SUBMIT program reads the ASMBL.SUB file, substituting X: for all occurrences of $1 and PRN for all occurrences of $2. This results in a $$$.SUB file containing the commands:

```
ASM X
DIR X.*
ERA *.BAK
PIP PRN:=X.PRN
ERA X.PRN
```

which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file on an alternate drive if the filename is preceded by a drive name. Submitted files are only acted upon when they appear on drive A. Thus, it is possible to create a submitted file on drive B that is executed at a later time when inserted in drive A.

An additional utility program called XSUB extends the power of the SUBMIT facility to include line input to programs as well as the CCP. The XSUB command is included as the first line of the SUBMIT file. When it is executed, XSUB self-relocates directly below the CCP. All subsequent SUBMIT command lines are processed by XSUB so that programs that read buffered console input, BDOS Function 10, receive their input directly from the SUBMIT file. For example, the file SAVER.SUB can contain the following SUBMIT lines:

```
XSUB
DDT
I$1.COM
R
GO
SAVE 1 $2.COM
```

a subsequent SUBMIT command, such as

```
A>SUBMIT SAVER PIP Y
```

substitutes X for $1 and Y for $2 in the command stream. The XSUB program loads, followed by DDT, which is sent to the command lines PIP.COM, R, and G0, thus returning to the CCP. The final command SAVE 1 Y.COM is processed by the CCP.

The XSUB program remains in memory and prints the message

```
(xsub active)
```

on each warm start operation to indicate its presence. Subsequent SUBMIT command streams do not require the XSUB, unless an intervening cold start occurs.

1.6.8 DUMP Command

Syntax:

DUMP ufn

The DUMP program prints the contents of the disk file (ufn) to the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long printouts can be aborted by pressing the shift and delete keys during printing.

1.6.9 FORMAT Utility

Syntax:

FORMAT

The format command initializes a disk or digital data pack for use of the CP/M operating system. If a disk or digital data pack is not formatted for CP/M, it will default to the EOS operating system that is built into ADAM.

After formatting a disk, a prompt appears giving you the opportunity to verify the formatting. A "Y" response will begin the verification process.

1.6.10 BACKUP Utility

Syntax:

BACKUP

The BACKUP utility lets you copy the entire contents of a disk or data pack. System files, if present, are copied. The BACKUP utility copies a disk or digital data pack exactly; it does not reorganize the files to maximize space allocation.

The disk or digital data pack containing the original files (those to be backed up) is called the SOURCE media. The blank disk or digital data pack onto which the files are copied is called the TARGET media. You must use the CP/M FORMAT utility to prepare the blank disk or digital data pack before you can use it in a BACKUP operation.

Disks can hold a maximum of 160 blocks of data. Digital data packs can hold up to 255 blocks of data. Because of the difference in capacity, a digital data pack can only be backed-up onto another digital data pack; it cannot be backed-up onto a disk. A full disk, however, can be backed up onto a digital data pack.

The BACKUP utility accommodates either a single or multi-drive system. With a single drive system, you will use drive A for both your source and target media.

```
BACK UP TO DRIVE
(A,B,C,D)? A
```

The next screen message prompts you to insert your source media. Insert the disk or digital data pack you want to backup, then press the RETURN key.

```
INSERT SOURCE MEDIA
PRESS RETURN WHEN READY
```

As the backup of a digital data pack proceeds, a status report is displayed on the screen. ADAM can read up to 48 blocks of material into active memory. The prompt

```
INSERT TARGET MEDIA
PRESS RETURN WHEN READY
```

appears and you must remove the source disk or digital data pack and insert the target disk or digital data pack. When you press the RETURN key, ADAM writes the information it has collected onto the target disk or digital data pack. With a digital data pack, a status message is displayed during this process. When the data is on the target media, the INSERT SOURCE MEDIA prompt appears. This process continues until all of the data from the source media is copied onto the target media.

With a multi-drive system, the drive from which you call the backup program is the source. You then have the opportunity to specify which drive contains the target media. Because both source and target media are accessible to the system, the backup is performed automatically.

ERROR Messages

There are several messages that may be displayed during the backup process.

```
ERROR IN READING DRIVE d OR
ERROR IN WRITING TO DRIVE d
ABORT RETRY of CONTINUE
(A,R, or C)
```

These messages indicate that the system cannot access one of the disks or digital data packs. Check to be sure you have inserted your source and target media correctly.

```
** TARGET DRIVE
SMALLER THAN SOURCE DRIVE
```

If you try to use backup to copy the contents of a digital data pack onto a disk, you will get this error message. To transfer the contents of a digital data pack to a disk, you must use a combination of SYSGEN and PIP or COPY.

1.6.11 COPY

Syntax:

COPY afn destination:
COPY ufn destination:

The COPY command lets you duplicate selected files. Files can be copied from one disk or digital data pack to another or onto the same disk or digital data pack.

To put a duplicate copy of a file on the same disk or digital data pack as the original file, you must rename the copied file using the format:

COPY filename A filename B

The operating system can only be copied using the sysgen command.

ERRORS:

`Incorrect File Names`

This means that the filenames you specified do not exist.

`Cannot Access Source`

This means that the source drive you specified is inactive or invalid.

`Cannot Copy File onto Itself`

You must rename a file to copy it onto the same disk or digital data pack.

`Storage device is full`

There is not enough space available on the disk or digital data pack you specified to accommodate the file or files you want to copy.

1.6.12 ADAM Utility

Syntax:

ADAM

ADAM converts ADAM EOS data files to CP/M 2.2 data files.

Steps:

1. ADAM to CP/M FILE COPY UTILITY
   ADAM TAPE/DISK IS DRIVE
   (A,B,C,D)?

   Type the letter indicating the drive that contains the ADAM data pack/disk and press return.

2. PLACE ADAM TAPE/DISK IN THE
   DEFAULT DRIVE
   PRESS RETURN WHEN READY

NOTE: If CP/M data packs are in both drives, the supposed ADAM VOLUME lists as graphic characters. Put the appropriate ADAM data pack into your drive and proceed with the instructions.

3. VOLUME = Volume name

   [ADAM file names and types scroll onto the screen.]

   TRANSFER ADAM
   FILE?

   Type in the name of the ADAM file and press return.

NOTE: Illegal characters may be put in a file name, but in most cases they will be ignored by CP/M when it makes up the file name. Remember that ADAM file names must be accurate, in upper- and lower case. CP/M file names can be inexact in terms of upper- and lower case.

4. ADAM FILE TYPE
   (H,h,A OR a) ?

   The file type appears to the left of the file's name in the directory. Type the letter corresponding to the file type and press Return.

5. NAME CPM FILE?

6. PLACE CPM TAPE/DISK IN THE
   DEFAULT DRIVE
   PRESS RETURN WHEN READY

7. ANOTHER FILE TO TRANSFER
   (Y/N)?

NOTE: Steps 2 and 6 appear only if you have not already placed disk or data pack in drive.

1.6.13 CPMADAM Utility

Syntax:

CPMADAM

To convert data files created under the CPM operating system to ADAM EOS data files.

Steps:

WITH ONE DRIVE

1. CP/M TO ADAM FILE COPY UTILITY
   ADAM TAPE/DISK IS DRIVE
   (A,B,C,D)?

   Type the letter indicating the drive that contains the ADAM data pack or disk and press return.

2. CPM FILE NAME?
3. ADAM FILE NAME?

   If the ADAM file will be used in LOGO, type only capital letters.

```
4. PLACE ADAM TAPE/DISK IN THE
   DEFAULT DRIVE
   PRESS RETURN WHEN READY

5. PLACE CPM TAPE/DISK IN THE
   DEFAULT DRIVE
   PRESS RETURN WHEN READY

6. ANOTHER FILE TO TRANSFER
   (Y or N)?
```

NOTE: Default file type is A. If you wish another file type, when you are typing in the ADAM file name, type the desired file type in brackets [H] immediately after the file name.

Example: `Letter.txt[H]`

FROM ONE DRIVE TO ANOTHER

The steps are the same as they are for CPMADAM with one drive, but steps *4* and *5* are eliminated.

1.6.14 CONFIG Command

Syntax:

CONFIG

CONFIG lets you change some CP/M 2.2 parameters: keyboard translations; Smart Key color, text, and action, character colors; cursor shape and color; background color; serial card settings; and the I/O default.

CONFIG is menu driven. Whenever you wish to change a CP/M parameter, press the number corresponding to the option you want and follow the instructions that appear on the screen. After you type CONFIG and press the carriage return, the Main Menu appears:

MAIN MENU:

```
ADAM Configuration
       for CP/M 2.2
1) Exit Configure
2) Read Tables From Drive
3) Read Tables From Memory
4) Write Tables To Drive
5) Write Tables To Memory
6) Edit Tables

Input Option (1-6)?
```

To change any parameter, first load the tables into temporary memory by selecting option 2 or 3. [If you select option 3, the transfer to memory is instantaneous.] Now choose option 6) Edit Tables to make the changes you want. If you failed to load the tables into memory before choosing 6, Config now reports,

```
No Tables Loaded Yet
```

Once you finish Editing, Return to the Main Menu (press Option 1) to write your changes out to the drive or to the current memory. Any tables in which the screen or peripheral device characteristics have been changed MUST be written to the drive. To see the effects of your change, reboot CP/M (^C).

EDIT TABLES:

```
Option 6: Editable Features
1) Return to Main Menu
2) Keyboard Translations
3) Smart Key Menu
4) Character Colors
5) Cursor Changes
6) Background Color
7) Serial Card Settings
8) Set Default I/O Byte
9) Initial Smart Key State

Input Option (1-9)?
```

The EDIT TABLES provide you with various CP/M parameters that you can configure: Type the number of the parameter you wish to change and press RETURN.

Selection 2—Keyboard Translations

All of the Smart Keys and Editing keys are presented one at a time, each key with its current Hexadecimal value.

```
Key __ has Value __
Change, Skip or Quit
(C,S or Q)?
```

C: Pressing C lets you change the specified key's values. Type in the desired Hexadecimal value (00-FF). Appendix B lists the values for each key. Once you finish, CONFIG lists the next Smart Key.

S: Pressing S lets you see the next key's stated value and make a selection.

Q: Pressing Q lets you return to the Edit Menu.

The Keys in turn are Shift Smart Keys VI to I, Smart Keys VI to I, Shift Tab, Home-Left (in combination), Home-Down, Home-Right, Home-Up, Up-Left, Down-Left, Down-Right, Up-Right, CTRL-Down, Left, Down, Right, Up, Shift-Delete, Shift-Clear, Shift-Print, Shift-Insert, Shift-Store, Shift-Move, Shift-Wildcard, Delete, Clear, Print, Insert, Store, Move, Undo, Wildcard, Home.

Selection 3—Smart Key Menu

This utility presents choices to change the actions and the label for each of the smart keys. The first menu allows you to choose the smart key you want to change.

SMART KEY MENU:

```
Smart Key Modifier
1) Return to Edit Menu
2) Change Smart Key I
3) Change Smart Key II
4) Change Smart Key III
5) Change Smart Key IV
6) Change Smart Key V
7) Change Smart Key VI

Input Option (1-7)?
```

Press the number corresponding to the Smart key you wish to change. Then press Return. CONFIG shows the screen:

```
Data For Smart Key V
⟶ SAVE ⟵
Enter New Return String
(Control DELETE Exists String Edit)
⟶
```

To change the function of the key, type in its new function and hold down the CONTROL key while pressing DELETE.

WARNING: In the Smart Key Menu, type exactly what you want. The destructive backspace and ^C are disabled in this Utility.

If you wish to put an action followed by the Carriage Return, you may type in both. For instance, if you decided to type in PIP and Return, you would type PIP followed by pressing CONTROL and M. The Screen would show PIP^M.)

As soon as you press CONTROL DELETE, the next instruction appears:

```
The Text Shown For The Key
⟶ SAVE ⟵
⟶      ⟵
Enter Two, Five Character Strings For
Displayed Smart Key (One For Each Line)
⟶
```

Type the new expression you wish to appear on the Smart Key label panel. If you wish to center it, leave a leading space. To move to the next line, fill the line with spaces. CONFIG automatically advances to the next line.

```
Display Color is Blue
Enter Color
(Blue or Yellow)?
```

Type blue or yellow and press Return. If you choose yellow, the number of the Smart Key will not appear in the panel. When you press the Return Key, CONFIG returns to the Smart Key Modifier menu.

Selections 4, 5, and 6

In these three parameters you can change the colors of the characters, cursor, and background. You input your option from the available colors:

```
COLOR MENU:  0-Transparent
             1-Black
             2-Medium Green
             3-Light Green
             4-Dark Blue
             5-Light Blue
             6-Dark Red
             7-Cyan
             8-Medium Red
             9-Light Red
             A-Dark Yellow
             B-Light Yellow
             C-Dark Green
             D-Magenta
             E-Gray
             F-White

             Input Option (1-F)?
```

Selection 4—Character Colors

Characters require four color selections; the Clear Bit (color that appears in the square around the letter. If this is set to 0 (transparent), it will show the regular background color, Set Bit, Inverse Clear Bit, and Inverse Set Bit.

Selection 5—Cursor Changes

You can change both the cursor color and its shape. Change the shape of the cursor through typing a new bit map:

```
This is the Cursor Graphic

00000000
00000000    Enter 8 Bytes For The New Cursor
00000000    Example 00001111
00000000
00000000
00000000
11111100
11111100
```

To change the shape of the cursor, you enter each line, substituting 1's wherever you wish the cursor to show up. Fill each line and then the cursor automatically advances to the next line. Mistakes may be corrected by backspacing.

Selection 6

You can change the screen background color by choosing option 6 and then pressing the number on the COLOR MENU that corresponds to the color you desire.

Selection 7—Serial Card Features

If you have purchased an RS-232 Serial Interface for ADAM, you may control many of the parameters used in communications:

```
SERIAL MENU:  Serial Card Features
              1) Return to Edit Menu
              2) Set Baud Rate
              3) Set Character Length
              4) Set Parity
              5) Set Num. Of Stop Bits
              6) Set Type Of Handshaking
              7) Set Other Functions

              Input Option (1-7)?
```

Option 2 allows you to set the communications (BAUD) rate to as low as 50 baud, or as high as 19.2 kilobaud.
Option 3 allows setting of Character Bit length to 5, 6, 7, or 8 bits.
Option 4 provides choices of Odd, Even, or No parity.
Option 5 allows setting of 1 or 2 stop bits.
Option 6 provides RTS-CTS, XON-XOFF, or no handshaking protocol.
Option 7 gives the choices of Clearing the Receive Buffer, DTR On, or DTR Off.

Selection 8—Set Default I/O Byte

The Input/Output Byte selects the physical devices that are activated when messages are directed through the logical devices CONsole, Reader (RDR), PUNch, or List (LST). The choices are:

```
CON: may be TTY: CRT: BAT: UC1:
RDR: may be TTY: PTR: UR1: UR2:
PUN: may be TTY: PTP: JP1: JP2:
LST: may be TTY: CRT: LPT: UL1:
```

Selection 9—Initial Smart Key State

When CP/M is booted, the Smart Keys appear at the bottom of the screen. If you want to use the space taken up by the Smart Keys, use CONFIG to have the whole text screen available. Select Off At Boot to get the extra space.

Remember to write your reconfigured tables to the drive or to memory before you reboot to see the results.

1.7 BDOS Error Messages

There are three error situations that the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

```
BDOS ERR ON d: error
```

where d is the drive name and error is one of the three error messages:

```
BAD SECTOR
SELECT
READ ONLY
```

The BAD SECTOR message indicates that the controller has detected an error in reading or writing the disk or digital data pack. This error is generally caused by a malfunctioning controller or a worn disk or data pack. If you find that CP/M reports this error more than once a month, check the condition of your controller, disks and data packs.

NOTE: Pressing the RETURN key can destroy disk integrity if the operation is a directory write. Be sure you have adequate back-ups.

The SELECT error occurs when there is an attempt to address a drive other than those supported by the BIOS. In this case, the value of d in the error message gives the selected drive. The system reboots following any input from the console.

The READ ONLY message occurs when there is an attempt to write to a disk, data pack, or file that has been designated as Read-Only in a STAT command or has been set to Read-Only by the BDOS. Reboot CP/M by using the warm start procedure, CTRL-C, or by performing a cold start whenever the disks or data packs are changed. If a changed disk or data pack is to be read but not written, BDOS allows the disk to be changed without the warm or cold start, but internally marks the drive as Read-Only. The status of the drive is subsequently changed to Read-Write if a warm or cold start occurs. On issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.

Section 2
The CP/M Editor

2.1 Introduction to ED

ED is the context editor for CP/M, and is used to create and alter CP/M source files. To start ED, type

```
ED filename
```

or

```
ED filename.typ
```

Generally, ED reads segments of the source file given by filename or filename.typ into the central memory, where you edit the file. It is written back to disk or digital data pack after alteration. If the source file does not exist before editing, it is created by ED and initialized as an empty file. The overall operation of ED is shown in Figure 2-1.

2.1.1 ED Operation

ED operates upon the source file, shown in Figure 2-1 by x.y, and passes all text through a memory buffer where the text can be viewed or altered.

The number of lines that can be maintained in the memory buffer varies with the line length, but has a total capacity of about 13000 characters in ADAM's 55K CP/M system.

Edited text material is written into a temporary work file. Upon termination of the edit, the memory buffer is written to the temporary file, followed by any remaining (unread) text in the source file. The name of the original file is changed from x.y to x.BAK so that the most recent edited source file can be reclaimed if necessary. See the CP/M commands ERASE and RENAME. The temporary file is then changed from x.**$$$** to x.y, which becomes the resulting edited file.

The memory buffer is logically between the source file and working file, as shown in Figure 2-2.

Figure 2-1. Overall ED Operation

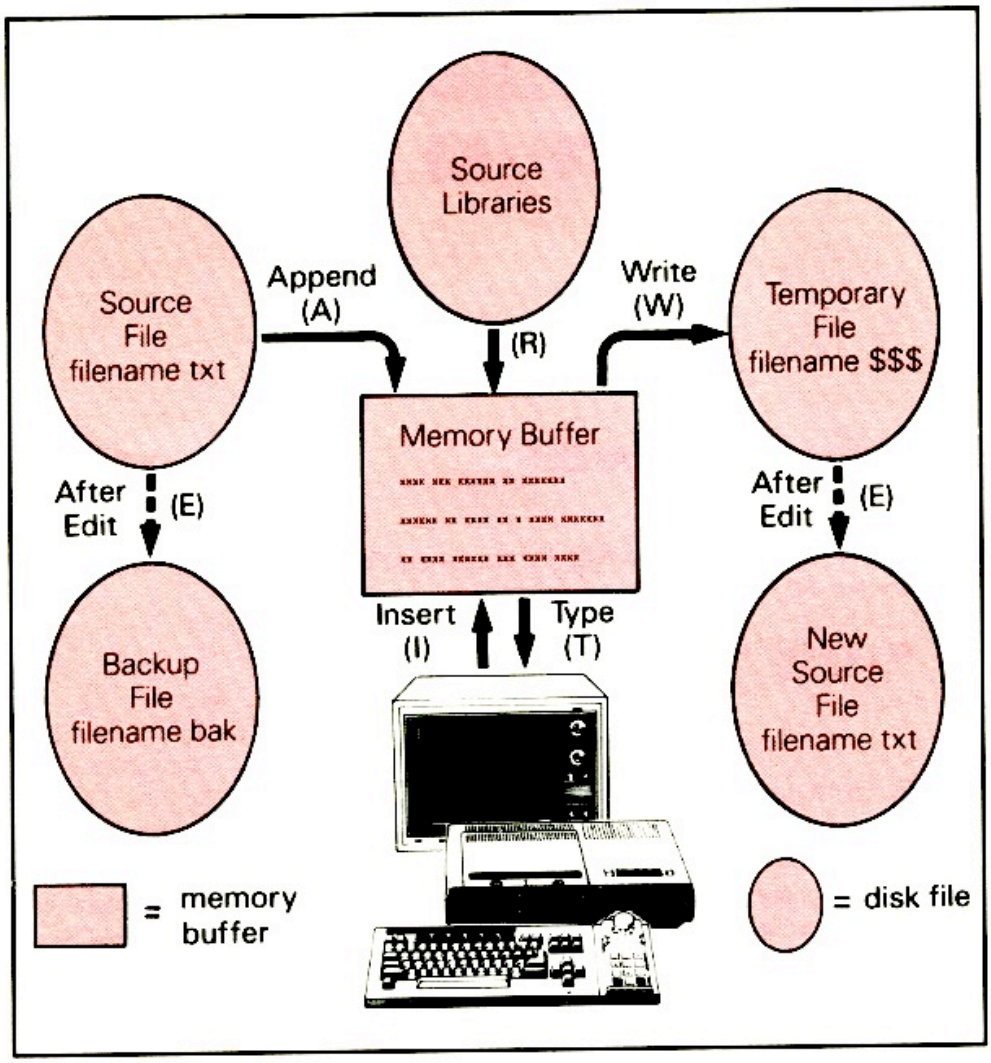
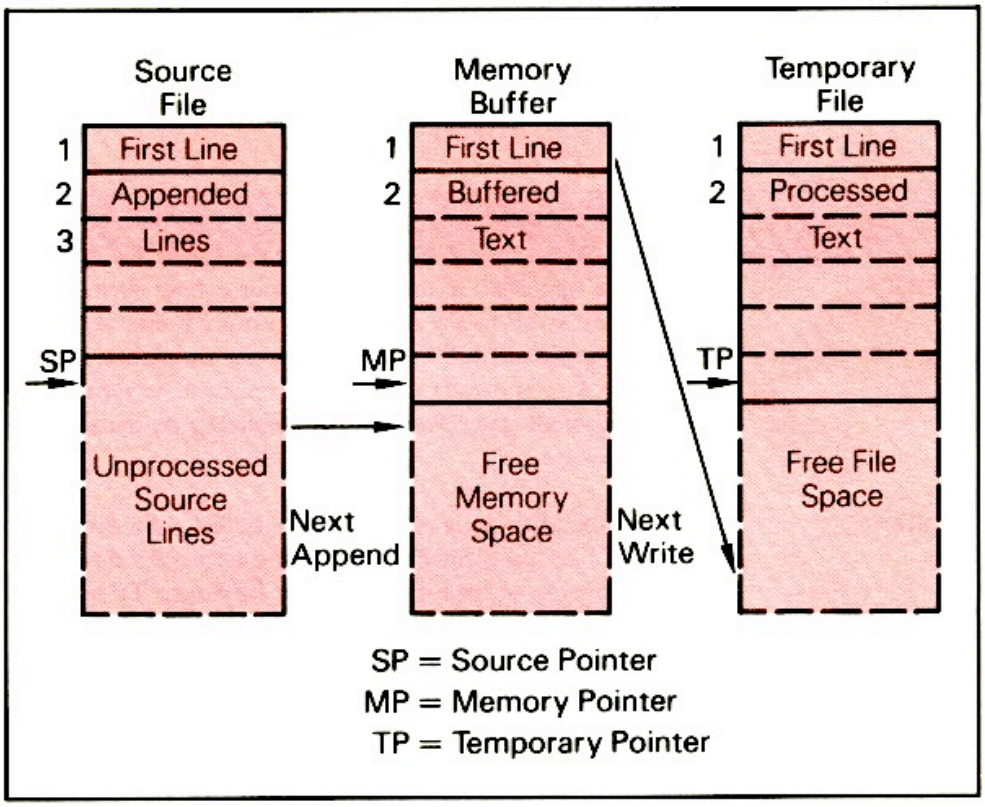


Figure 2-2. Memory Buffer Organization



2.1.2 Text Transfer Functions

Given that n is an integer value in the range 0 through 65535, several single-letter ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file. Single letter commands are shown in upper-case, but can be typed in either upper- or lower-case.

Table 2-1. ED Text Transfer Commands

Command	Result
nA	Appends the next n unprocessed source lines from the source file at SP to the end of the memory buffer at MP. Increment SP and MP by n. If upper-case translation is set (see the U command) and the A command is typed in upper-case, all input lines will automatically be translated to upper-case.
nW	Writes the first n lines of the memory buffer to the temporary file free space. Shift the remaining lines n+1 through MP to the top of the memory buffer. Increment TP by n.
E	Ends the edit. Copy all buffered text to temporary file and copy all unprocessed source lines to temporary file. Rename files.
H	Moves to head of new file by performing automatic E command. The temporary file becomes the new source file, the memory buffer is emptied, and a new temporary file is created. The effect is equivalent to issuing an E command, followed by a reinvocation of ED, using x.y as the file to edit.
O	Returns to original file. The memory buffer is emptied, the temporary file is deleted, and the SP is returned to position 1 of the source file. The effects of the previous editing commands are thus nullified.
Q	Quits edit with no file alterations, returns to CP/M.

There are a number of special cases to consider. If the integer n is omitted in any ED command where an integer is allowed, then 1 is assumed. Thus, the commands A and W append one line and write one line, respectively. In addition, if a pound sign # is given in the place of n, then the integer 65535 is assumed (the largest value for n that is allowed). Because most source files can be contained entirely in the memory buffer, the command #A is often issued at the beginning of the edit to read the entire source file to memory. Similarly, the command #W writes the entire buffer to the temporary file.

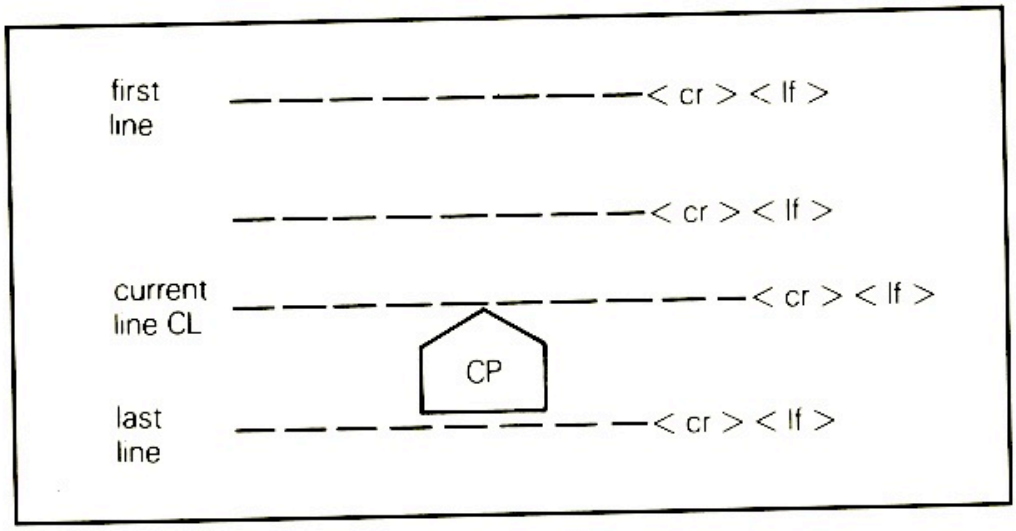
Two special forms of the A and W commands are provided as a convenience. The command 0A fills the current memory buffer at least half full, while 0W writes lines until the buffer is at least half empty. An error is issued if the memory buffer size is exceeded. You can then enter any command, such as W, that does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

2.1.3 Memory Buffer Organization

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an imaginary character pointer (CP) that moves throughout the memory buffer under your command.

The memory buffer appears logically as shown in Figure 2-3, where the dashes represent characters of the source line of indefinite length, terminated by carriage return (<cr>) and line-feed (<lf>) characters, and CP represents the imaginary character pointer. Note that the CP is always located ahead of the first character of the first line, behind the last character of the last line, or between two characters. The current line CL is the source line that contains the CP.

Figure 2-3. Logical Organization of Memory Buffer



2.1.4 Line Numbers and ED Start-up

ED produces absolute line number prefixes that are used to reference a line or range of lines. The absolute line number is displayed at the beginning of each line when ED is in insert mode (see the I command in Section 2.1.5). Each line number takes the form

```
nnnnn:
```

where nnnnn is an absolute line number in the range of 1 to 65535. If the memory buffer is empty or if the current line is at the end of the memory buffer, nnnnn appears as 5 blanks.

You can reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. The line denoted by the absolute line number must be in the memory buffer (see the A command). Thus, the command

```
345:T
```

is interpreted as move to absolute 345, and type the line. Absolute line numbers are produced only during the editing process and are not recorded with the file. The line numbers will change following a deleted or expanded section of text.

You can also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute number by a colon. Thus, the command

```
:400T
```

is interpreted as type from the current line number through the line whose absolute number is 400. Combining the two line reference forms, the command

```
345::400T
```

is interpreted as move to absolute line 345, then type through absolute line 400. Absolute line references of this sort can precede any of the standard ED commands.

Line numbering is controlled by the V (Verify Line Numbers) command. Line numbering can be turned off by typing the -V command.

If the file to edit does not exist, ED displays the following message:

```
NEW FILE
```

To move text into the memory buffer, you must enter an i command before typing input lines and terminate each line with a carriage return. A single CTRL-Z character returns ED to command mode.

2.1.5 Memory Buffer Operation

When ED begins, the memory buffer is empty. You can either append lines from the source file with the A command, or enter the lines directly from the console with the insert command. The insert command takes the following form:

```
I
```

ED then accepts any number of input lines. You must terminate each line with a <cr> (the <lf> is supplied automatically). A single CTRL-Z, denoted ^Z, returns ED to command mode. The CP is positioned after the last character entered. The following sequence:

```
I<cr>
NOW IS THE<cr>
TIME FOR<cr>
ALL GOOD MEN<cr>
^Z
```

leaves the memory buffer as

```
NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>
```

Generally, ED accepts command letters in upper- or lower-case. If the command is upper-case, all input values associated with the command are translated to upper-case. If the I command is typed, all input lines are automatically translated internally to upper-case. The lower-case form of the i command is most often used to allow both upper- and lower-case letters to be entered.

Various commands can be issued that control the CP or display source text in the vicinity of the CP. The commands shown below with a preceding n indicate that an optional unsigned value can be specified. When preceded by $\pm$, the command can be unsigned, or have an optional preceding plus or minus sign. As before, the pound sign # is replaced by 65535. If an integer n is optional, but not supplied, then n=1 is assumed. Finally, if a plus sign is optional, but none is specified, then + is assumed.

Table 2-2. Editing Commands

Command	Action
± B	Move CP to beginning of memory buffer if + and to bottom if —.
± nC	Move CP by ± n characters (moving ahead if +), counting the <cr><lf> as two characters.
± nD	Delete n characters ahead of CP if plus and behind CP if minus.
± nK	Kill (remove) ± n lines of source text using CP as the current reference. If CP is not at the beginning of the current line when K is issued, the characters before CP remain if + is specified, while the characters after CP remain if — is given in the command.
± nL	If n = 0, move CP to the beginning of the current line, if it is not already there. If n ≠ 0, first move the CP to the beginning of the current line and then move it to the beginning of the line that is n lines down (if +) or up (if —). The CP will stop at the top or bottom of the memory buffer if too large a value of n is specified.
± nT	If n = 0, type the contents of the current line up to CP. If n = 1, type the contents of the current line from CP to the end of the line. If n>1, type the current line along with n — 1 lines that follow, if + is specified. Similarly, if n>1 and — is given, type the previous n lines up to the CP. Any key can be depressed to abort long type-outs.
± n	Equivalent to ± nLT, which moves up or down and types a single line.

2.1.6 Command Strings

Any number of commands can be typed contiguously (up to the capacity of the console buffer) and are executed only after you press the <cr>. Table 2-3 summarizes the CP/M console line-editing commands used to control the input command line.

Table 2-3. Line-editing Controls

Command	Result
CTRL-C	Reboots the CP/M system when typed at the start of a line.
CTRL-E	Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
CTRL-H	Backspaces one character position.
CTRL-J	Terminates current input (line-feed).
CTRL-M	Terminates current input (carriage return).
CTRL-R	Retypes current command line: types a clean line character deletion with rubouts.
CTRL-U	Deletes the entire line typed at the console.
CTRL-X	Same as CTRL-U.
CTRL-Z	Ends input from the console (used in PIP and ED).
backspace	Deletes the last character typed at the console; or the character before the cursor.

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. In the following example, the command strings on the left produce the results shown to the right. Use lower-case command letters to avoid automatic translation of strings to upper-case.

Command String	Effect
B2T<cr>	Move to beginning of the buffer and type two lines: NOW IS THE TIME FOR The result in the memory buffer is NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
5C0T<cr>	Move CP five characters and type the beginning of the line NOW I. The result in the memory buffer is NOW IS THE<cr><lf>
2L-T<cr>	Move two lines down and type the previous line TIME FOR. The result in the memory buffer is NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
-L#K<cr>	Move up one line, delete 65535 lines that follow. The result in the memory buffer is NOW IS THE<cr><lf>
I<cr> TIME TO<cr> INSERT<cr> ^Z	Insert two lines of text with automatic translation to upper-case. The result in the memory buffer is

Command String	Effect
	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>
-2L#T<cr>	Move up two lines and type 65535 lines ahead of CP NOW IS THE.The result in the memory buffer is NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>
<cr>	Move down one line and type one line INSERT. The result in the memory buffer is NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>

2.1.7 Text Search and Alteration

ED has a command that locates strings within the memory buffer. The command takes the form

nF s <cr>

or

nF s ^Z

where s represents the string to match, followed by either a <cr> or CTRL-Z, denoted by ^Z. ED starts at the current position of CP and attempts to match the string. The match is attempted n times and, if successful, the CP is moved directly after the string. If the n matches are not successful, the CP is not moved from its initial position. Search strings can include CTRL-L, which is replaced by the pair of symbols <cr><lf>.

The following commands illustrate the use of the F command:

Command String	Effect
B#T<cr>	Move to the beginning and type the entire buffer. The result in the memory buffer is NOW IS THE <cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
FS T<cr>	Find the end of the string S T. The result in the memory buffer is NOW IS THE<cr><lf>
FIs^Z0TT	Find the next I and type to the CP; then type the remainder of the current line ME FOR. The result in the memory buffer is NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is

I s ^Z

or

I s<cr>

where s is the string to insert. If the insertion string is terminated by a CTRL-Z, the string is inserted directly following the CP, and the CP is positioned directly after the string. The action is the same if the command is followed by a <cr> except that a <cr><lf> is automatically inserted into the text following the string. The following command sequences are examples of the F and I commands:

Command String	Effect
BITHIS IS ^Z<cr>	Insert THIS IS at the beginning of the text. The result in the memory buffer is THIS IS NOW THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
FTIME^Z-4DIPLACE^Z<cr>	Find TIME and delete it; then insert PLACE. The result in the memory buffer is THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> ALL GOOD MEN<cr><lf>
3F0↕Z-3D5D1 CHANGES^Z<cr>	Find third occurrence of O (that is, the second O in GOOD), delete previous 3 characters and the subsequent 5 characters; then insert CHANGES. The result in the memory buffer is THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> ALL CHANGES<cr><lf>
-8CISOURCE<cr>	Move back 8 characters and insert the line SOURCE <cr><lf>. The result in the memory buffer is THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> ALL SOURCE<cr><lf> CHANGES<cr><lf>

ED also provides a single command that combines the F and I commands to perform simple string substitutions. The command takes the following form:

```
nS  s1^Zs2 <cr>
```

or

```
nS s1^Zs2  ^Z
```

and has exactly the same effect as applying the following command string a total of n times:

```
F s1u^Z-kDIs2 <cr>
```

or

```
F s1^Z-kDIs2 ^Z
```

where k is the length of the string. ED searches the memory buffer starting at the current position of CP and successively substitutes the second string for the first string until the end of buffer, or until the substitution has been performed n times.

As a convenience, a command similar to F is provided by ED that automatically appends and writes lines as the search proceeds. The form is

```
n N s <cr>
```

or

```
n N s ^Z
```

which searches the entire source file for the nth occurrence of the strings (F fails if the string cannot be found in the current buffer). The operation of the N command is precisely the same as F except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory content is written (that is, an automatic #W is issued). Input lines are then read until the buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until the string has been found n times, or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command, takes the form

n J $s_1^{Z}s_2^{Z}s_3$<cr>

or

n J $s_1^{Z}s_2^{Z}s_3$^Z

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string s_1. If found, insert the string s_2, and move CP to follow s_2. Then delete all characters following CP up to, but not including, the string s_3, leaving CP directly after s_2. If s_3 cannot be found, then no deletion is made. If the current line is

NOW IS THE TIME<cr><lf>

the command

JW ^ZWHAT^Z^l<cr>

results in

NOW WHAT <cr>< lf>

^l (CTRL-L) represents the pair <cr><lf> in search and substitute strings.

The number of characters ED allows in the F, S, N, and J commands is limited to 100 symbols.

2.1.8 Source Libraries

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is

```
R filename ^Z
```

or

```
R filename <cr>
```

where filename is the primary filename of a source file on the disk or data pack with an assumed filetype of LIB. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command

```
RMACRO<cr>
```

is issued, ED reads from the file MACRO.LIB until the end-of-file and automatically inserts the characters into the memory buffer.

ED also includes a block move facility implemented through the X (Transfer) command. The form

```
nX
```

transfers the next n lines from the current line to a temporary file called

```
X$$$$$$.LIB
```

which is active only during the editing process. You can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred lines accumulate one after another in this file and can be retrieved by typing

```
R
```

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer. Note that the X command does not remove the transferred lines from the memory buffer, although a K command can be used directly after the X, and the R command does not empty the transferred LIB file. That is, given that a set of lines has been transferred with the X command, they can be reread any number of times back into the source file. The command

```
0X
```

is provided to empty the transferred line file.

Upon normal completion of the ED program through Q or E, the temporary LIB file is removed. If ED is aborted with a CTRL-C, the LIB file exists if lines have been transferred, but is generally empty (a subsequent ED invocation erases the temporary file).

2.1.9 Repetitive Command Execution

The macro command M allows you to group ED commands together for repeated evaluation. The M command takes the following form:

```
n M CS <cr>
```

or

```
n M CS ^Z
```

where CS represents a string of ED commands, not including another M command. ED executes the command string n times if n>1. If n=0 or 1, the command string is executed repetitively until an error condition is encountered (for example, the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line that is changed:

```
MFGAMMA^Z-5DIDELTA^Z0TT<cr>
```

or equivalently

```
MSGAMMA^ZDELTA^Z0TT<cr>
```

2.2 ED Error Conditions

On error conditions, ED prints the message BREAK X AT C where X is one of the error indicators shown in Table 2-4.

Table 2-4. Error Message Symbols

Symbol	Meaning
?	Unrecognized command.
>	Memory buffer full (use one of the commands D, K, N, S, or W to remove characters); F, N, or S strings too long.
#	Cannot apply command the number of times specified (for example, in F command).
0	Cannot open LIB file in R command.

If there is a disk error, CP/M displays the following message:

```
BDOS ERR on d: BAD SECTOR
```

You can ignore the error by pressing RETURN at the console (in this case, the memory buffer data should be examined to see if they were incorrectly read), or you can reset the system with a CTRL-C and reclaim the back-up file if it exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information. For example, type the following:

```
TYPE x.BAK
```

where x is the file being edited. Then remove the primary file

```
ERA x.y
```

and rename the BAK file

```
REN x.y=x.BAK
```

The file can then be reedited, starting with the previous version.

ED also takes file attributes into account. If you attempt to edit a Read-Only file, the message

```
** FILE IS READ/ONLY **
```

appears at the console. The file can be loaded and examined, but cannot be altered. You must end the edit session and use STAT to change the file attribute to R/W. If the edited file has the system attribute set, the following message:

```
'SYSTEM' FILE NOT ACCESSIBLE
```

is displayed and the edit session is aborted. Again, the STAT program can be used to change the system attribute.

2.3 Control Characters and Commands

Table 2-5 summarizes the control characters and commands available in ED.

Table 2-5. ED Control Characters

Control Character	Function
CTRL-C	System reboot
CTRL-E	Physical <cr><lf> (not actually entered in command)
CTRL-H	Backspace
CTRL-J	Logical tab (cols 1, 9, 16, ...)
CTRL-L	Logical <cr><lf> in search and substitute strings
CTRL-R	Repeat line
CTRL-U	Line delete
CTRL-X	Line delete
CTRL-Z	String terminator
BACKSPACE	Character delete

Table 2-6 summarizes the commands used in ED.

Table 2-6. ED Commands

Command	Function
nA	Append lines
± B	Begin or bottom of buffer
± nC	Move character positions
± nD	Delete characters
E	End edit and close files (normal end)
nF	Find string
H	End edit, close and reopen files
I	Insert characters, use i if both upper- and lower-case characters are to be entered.
nJ	Place strings in juxtaposition
± nK	Kill (remove) lines
± nL	Move down/up lines
nM	Macro definition
nN	Find next occurrence with autoscan
O	Return to original file
± nP	Move and print pages
Q	Quit with no file changes
R	Read library file
nS	Substitute strings
± nT	Type lines

(continued)

Table 2-6 summarizes the commands used in ED.

Table 2-6. ED Commands (continued)

Command	Function
± U	Translate lower- to upper-case if U, no translation if —U
± V	Verify line numbers, or show remaining free character space
0V	A special case of the V command, OV, prints the memory buffer statistics in the form: free/total where free is the number of free bytes in the memory buffer (in decimal) and total is the size of the memory buffer
nW	Write lines
nZ	Wait (sleep) for approximately n seconds
± n	Move and type (± nLT).

Because of common typographical errors, ED requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The following commands:

- E(end)
- H(head)
- O(original)
- Q(quit)

must be typed as single letter commands.

The commands I, J, M, N, R, and S should be typed as i, j, m, n, r, and s if both upper- and lower-case characters are used in the operation, otherwise all characters are converted to upper-case. When a command is entered in upper-case, ED automatically converts the associated string to upper-case, and vice versa.

Section 3
CP/M Assembler

3.1 Introduction

The CP/M assembler reads assembly-language source files from the disk or data pack and produces 8080 machine language in Intel hex format. To start the CP/M assembler, type

```
ASM filename
```

or

```
ASM filename.parms
```

In both cases, the assembler assumes there is a file on the disk or data pack with a filetype of ASM

filename.ASM

which contains an 8080 assembly-language source file. The first and second forms shown above differ only in that the second form allows parameters to be passed to the assembler to control source file access, and hex file destination and print file destinations.

In either case, the CP/M assembler loads and prints the message:

```
CP/M ASSEMBLER VER 2.2
```

In the case of the first command, the assembler reads the source file with assumed filetype ASM and creates two output files

```
filename.HEX
filename.PRN
```

The HEX file contains the machine code corresponding to the original program in Intel hex format, and the PRN file contains an annotated listing showing generated machine code, error flags, and source lines. If errors occur during translation, they are listed in the PRN file and at the console.

The form ASM filename parms is used to direct input and output files. The parms portion of the command is three letters that specify the origin of the source file, the destination of the hex file, and the destination of the print file. The form is

```
filename.p1p2p3
```

where p1, p2, and p3 are single letters. P1 can be

```
A,B,C,D,M
```

which designates the drive that contains the source file. P2 can be

```
A,B,C,D,M
```

which designates the drive that will receive the hex file; or, P2 can be

```
Z
```

which skips the generation of the hex file.

P3 can be

```
A,B,C,D,M
```

which designates the drive that will receive the print file. P3 can also be specified as

```
X
```

which places the listing at the console; or

```
Z
```

which skips generation of the print file. Thus, the command

```
ASM X.AAA
```

indicates that the source, X.HEX, and print, X.PRN, files are also to be created on disk A. This form of the command is implied if the assembler is run from disk A. Given that you are currently addressing disk A, the above command is the same as

```
ASM X
```

The command

```
ASM X.ABX
```

indicates that the source file is to be taken from disk A, the hex file is to be placed on disk B, and the listing file is to be sent to the console. The command

```
ASM X.BZZ
```

takes the source file from disk B and skips the generation of the hex and print files. This command is useful for fast execution of the assembler to check program syntax.

The source program format is compatible with the Intel 8080 assembler. Macros are not implemented in ASM. There are certain extensions in the CP/M assembler that make it somewhat easier to use. These extensions are described below.

3.2 Program Format

An assembly-language program acceptable as input to the assembler consists of a sequence of statements of the form

```
line# label operation operand ;comment
```

where any or all of the fields may be present in a particular instance. Each assembly-language statement is terminated with a carriage return and line-feed (the line-feed is inserted automatically by the ED program), or with the character !, which is treated as an end-of-line by the assembler. Thus, multiple assembly-language statements can be written on the same physical line and separated by exclamation points.

The line# is an optional decimal integer representing the source program line number. ASM ignores this field.

The label field takes either of the following forms:

```
identifier
identifier:
```

The label field is optional, except where noted in particular statement types. The identifier is a sequence of alphanumeric characters where the first character is alphabetic. Identifiers can be freely used to label elements such as program steps and assembler directives, but cannot exceed 16 characters in length. All characters are significant in an identifier, except for the embedded dollar symbol $, which can be used to improve readability of the name. Lower-case alphabetics are treated as upper-case. The following are all valid instances of labels:

```
x       xy      long$name
x:      yxl:    longer$named$data:
xly2    Xlx2    x234$5678$9012$3456:
```

The operation field contains either an assembler directive or pseudo operation, or an 8080 machine operation code. The pseudo operations and machine operation codes are described in Section 3.3.

Generally, the operand field contains an expression formed out of constants and labels, along with arithmetic and logical operations on these elements. The complete details of properly formed expressions are given in Section 3.3.

The comment field contains arbitrary characters following the semicolon symbol until the next real or logical end-of-line. These characters are read and listed, but otherwise ignored by the assembler. The CP/M assembler also treats statements that begin with an * in column one as comment statements that are listed and ignored in the assembly process.

The assembly-language program is a sequence of statements, terminated by an optional END statement. All statements following the END are ignored by the assembler.

3.3 Forming the Operand

The operand field consists of simple operands, labels, constants, and reserved words, combined in properly formed expressions by arithmetic and logical operators. The expression computation is carried out by the assembler during the assembly process. Each expression must produce a 16-bit value during the assembly. Further, the number of significant digits in the result must not exceed the intended use. If an expression is to be used in a byte move immediate instruction, the most significant 8 bits of the expression must be zero. The restriction on the expression significance is given with the individual instructions.

3.3.1 Labels

As discussed above, a label is an identifier that occurs on a particular statement. In general, the label is a value determined by the type of statement that it precedes. If the label occurs on a statement that generates machine code or reserves memory space (for example, a MOV instruction or a DS pseudo operation), the label is given the value of the program address that it labels. If the label precedes an EQU or SET, the label is given the value that results from evaluating the operand field. Except for the SET statement, an identifier can label only one statement.

When a label appears in the operand field, its value is substituted by the assembler. This value can then be combined with other operands and operators to form the operand field for a particular instruction.

3.3.2 Numeric Constants

A numeric constant is a 16-bit value in one of several bases. The base is called the radix and is denoted by a trailing radix indicator. The following are radix indicators:

- B is a binary constant (base 2).
- O is a octal constant (base 8).
- Q is a octal constant (base 8).
- D is a decimal constant (base 10).
- H is a hexadecimal constant (base 16).

Q can be used for octal numbers because the letter O is easily confused with the digit 0. Any numeric constant that does not have a radix indicator is a decimal constant.

A constant is a sequence of digits, followed by an optional radix indicator. The digits must be in the appropriate range for the radix. Binary constants must be composed of 0 and 1 digits, octal constants can contain the digits 0-7, while decimal constants contain decimal digits. Hexadecimal constants contain decimal digits as well as hexadecimal digits A(10D), B(11D), C(12D), D(13D), E(14D), and F(15D). The leading digit of a hexadecimal constant must be a decimal digit to avoid confusing a hexadecimal constant with an identifier. A leading 0 will always suffice. A constant composed in this manner must evaluate to a binary number that can be contained within a 16-bit counter, otherwise it is truncated on the right by the assembler.

Embedded $ signs are allowed within constants to improve their readability. If the radix indicator is in lower case, it is translated to upper-case. The following are valid examples of numeric constants:

```
1234        1234D       1100B       1111$0000$1111$0000B
1234H       0FFEH       33770       33$77$22Q
3377o       0fe3h       1234d       0ffffh
```

3.3.3 Reserved Characters

There are several reserved character sequences that have predefined meanings in the operand field of a statement. The names of 8080 registers are given below. When they are encountered, they produce the values shown to the right.

Table 3-1. Reserved Characters

Character	Value
A	7
B	0
C	1
D	2
E	3
H	4
L	5
M	6
SP	6
PSW	6

Lower-case characters have the same values as their upper-case equivalents. Machine instructions can also be used in the operand field; they evaluate to their internal codes. In the case of instructions that require operands, where the specific operand becomes a part of the binary bit pattern of the instruction, the value of the instruction is the bit pattern of the instruction with zeros in the optional fields.

When the **$** symbol is not embedded within identifiers and numeric constants in the operand field, its value becomes the address of the next instruction not including the instruction contained within the current logical line.

3.3.4 String Constants

String constants represent sequences of ASCII characters and are enclosed within single quotes. All strings must be fully contained within the current physical line and must not exceed 64 characters in length. Exclamation points can be used in strings. The single quote can be used within a string by typing two single quotes. CP/M sees the two single quotes as one quote. You must use two keystrokes to type the two quotes. The double quote symbol does not work. In most cases, the string length is restricted to either one or two characters (the DB pseudo operation is an exception), in which case the string becomes an 8- or 16-bit value, respectively. Two-character strings become a 16-bit constant, with the first character as the high-order byte and the second character as the low- order byte.

The value of a character is its corresponding ASCII code. Within strings, both upper- and lower-case characters can be represented.

Note: Only graphic printing ASCII characters are allowed within strings.

Valid strings:	How assembler reads the strings:
'A' 'AB' 'ab' 'c'	A AB ab c
''a''' ''' '''	a'''
'Walla Walla Wash.'	Walla Walla Wash.
'She said "Hello" to me.'	She said "Hello" to me
'I said "Hello" to her.'	I said "Hello" to her

3.3.5 Arithmetic and Logical Operators

The operands described in Section 3.3 can be combined in normal algebraic notation using any combination of properly formed operands, operators, and parenthetical expressions. The operators recognized in the operand field are described in Table 3-2.

Table 3-2. Arithmetic and Logical Operators

Operators	Meaning
a + b	unsigned arithmetic sum of a and b
a − b	unsigned arithmetic difference between a and b
+ b	unary plus (produces b)
− b	unary minus (identical to 0 − b)
a * b	unsigned magnitude multiplication of a and b
a / b	unsigned magnitude division of a by b
a MOD b	remainder after a / b.
NOT b	logical inverse of b (all 0s become 1s, 1s become 0s), where b is considered a 16-bit value
a AND b	bit-by-bit logical AND of a and b
a OR b	bit-by-bit logical OR of a and b
a XOR b	bit-by-bit logical EXCLUSIVE OR of a and b
a SHL b	the value that results from shifting a to the left by an amount b, with zero fill
a SHR b	the value that results from shifting a to the right by an amount b, with zero fill

In each case, a and b represent simple operands (labels, numeric constants, reserved words, and one- or two-character strings) or fully enclosed parenthetical subexpressions, like those shown in the following examples:

```
10+20   10h+37Q   LI/3   (L2+4) SHR 3
('a' AND 5fh) + '0' ('B'+B) OR (PSW+M)
(1+(2+c)) SHR (A−(B+1))
```

All computations are performed at assembly time as 16-bit unsigned operations. Thus, −1 is computed as 0−1, which results in the value 0ffffh (that is, all 1s). The resulting expression must fit the operation code in which it is used. For example, if the expression is used in an ADI (add immediate) instruction, the high-order 8 bits of the expression must be zero. As a result, the operation ADI−1 produces an error message (−1 becomes 0ffffh, which cannot be represented as an 8-bit value), while ADI(−1) AND 0FFH is accepted by the assembler because the AND operation zeros the high-order bits of the expression.

3.3.6 Precedence of Operators

As a convenience, ASM assumes that operators have a relative precedence of application that allows you to write expressions without nested levels of parentheses. The resulting expression has assumed parentheses that are defined by the relative precedence. The order of application of operators in nonparenthetical expressions is listed below. Operators listed first have highest precedence (they are applied first in a nonparenthetical expression), while operators listed last have lowest precedence. Operators listed on the same line have equal precedence, and are applied from left to right as they are encountered in an expression.

```
* / MOD SHL SHR
- +
NOT
AND
OR XOR
```

Thus, the expressions shown to the left below are interpreted by the assembler as the parenthetical expressions shown to the right.

```
a*b+c                          (a*b)+c
a+b*c                          a+(b*c)
a MOD b*c SHL d                ((a MOD b)*c) SHL d
a OR b AND NOT c+d SHL e       a OR (b AND (NOT (c+(d SHL e))))
```

Balanced, parenthetical subexpressions can always be used to override the assumed parentheses; thus, the last expression above could be rewritten to force application of operators in a different order, as shown:

```
(a OR b) AND (NOT c)+ d SHL e
```

This results in these assumed parentheses:

```
(a OR b) AND ((NOT c) + (d SHL e))
```

A nonparenthetical expression is well-formed only if the expression that results from inserting the assumed parentheses is well-formed.

3.4 Assembler Directives

Assembler directives are used to set labels to specific values during the assembly, perform conditional assembly, define storage areas, and specify starting addresses in the program. Each assembler directive is denoted by a pseudo operation that appears in the operation field of the line. The acceptable pseudo operations are shown in Table 3-3.

Table 3-3. Assembler Directives

Directive	Meaning
ORG	set the program or data origin
END	end program, optional start address
EQU	numeric "equate"
SET	numeric "set"
IF	begin conditional assembly
ENDIF	end of conditional assembly
DB	define data bytes
DW	define data words
DS	define data storage area

3.4.1 The ORG Directive

Syntax:

label ORG expression

Label is an optional program identifier and expression is a 16-bit expression, consisting of operands that are defined before the ORG statement. The assembler begins machine code generation at the location specified in the expression. There can be any number of ORG statements within a particular program, and there are no checks to ensure that you are not defining overlapping memory areas. Most programs written for CP/M begin with an ORG statement of the form:

```
ORG   100H
```

which causes machine code generation to begin at the base of the CP/M transient program area (TPA). If a label is specified in the ORG statement, the label is given the value of the expression. This label can then be used in the operand field of other statements to represent this expression.

3.4.2 The END Directive

Syntax:

label END

The END statement is optional in an assembly-language program. If it is present it must be the last statement. All subsequent statements are ignored in the assembly.

The label is optional. When END is used, the assembly process stops, and the default starting address of the program is taken as 0000.

3.4.3 The EQU Directive

Syntax:

label EQU expression

The EQU (equate) statement is used to set up synonyms for particular numeric values.

The label must be present and must not be used for any other statement. The assembler evaluates the expression and assigns this value to the label field. The label is usually a name that describes the value with a meaningful name. This name is used throughout the program to place parameters on certain functions. Suppose data received from a teletype appears on a particular input port, and data is sent to the teletype through the next output port in sequence. For example this is a series of equate statements to define ports to correspond to BDOS calls.

```
BDOS      EQU 005H          ;CALL TO BDOS
TTYBASE   EQU 3H            ;BASE PORT NUMBER FOR TTY
TTYIN     EQU TTYBASE       ;TTY DATA IN
TTYOUT    EQU TTYBASE+1     ;TTY DATA OUT
```

At a later point in the program, the statements that access the teletype can appear as follows:

```
IN          TTYIN           ;READ TTY DATA TO REG-A
OUT         TTYOUT          ;WRITE DATA TO TTY FROM REG-A
```

making the program more readable than if the absolute I/O ports are used. Further, if the hardware environment is redefined to start the teletype communications ports at 7FH instead of 3H, the first statement need only be changed to

```
TTYBASE         EQU 7FH         ;BASE PORT NUMBER FOR TTY
```

and the program can be reassembled without changing any other statements.

3.4.4 The SET Directive

Syntax:

label SET expression

The label can be used in other SET statements within the program. The expression is evaluated and becomes the current value associated with the label. Thus, the EQU statement defines a label with a single value, while the SET statement defines a value that is valid from the current SET statement to the next occurrence of the in SET statement. The use of the SET is similar to the EQU statement, but is used most often in controlling conditional assembly.

3.4.5 The IF and ENDIF Directives

Syntax:

IF expression

statement#1

statement#2

...

statement#n

ENDIF

The IF and ENDIF statements define which assembly-language statements that are to be included or excluded during the assembly process.

The assembler evaluates the expression following an IF statement. All operands in the expression must be defined before the IF statement. If the expression evaluates to a nonzero value, then statement#1 through statement#n are assembled. If the expression evaluates to zero, the statements are listed but not assembled. Conditional assembly is often used to write a single generic program that includes a number of possible run-time environments, with only a few specific portions of the program selected for any particular assembly. The following program segments, for example, might be part of a program that communicates with either a teletype or a CRT console (but not both) by selecting a particular value for TTY before the assembly begins.

```
BDOS     EQU    005H
TRUE     EQU    OFFFFH      ;DEFINE VALUE OF TRUE
FALSE    EQU    NOT TRUE    ;DEFINE VALUE OF FALSE
;
TTY      EQU    TRUE        ;TRUE IF TTY, FALSE IF CRT
;
TTYBASE  EQU    3H          ;BASE OF TTY I/O PORTS
CRTBASE  EQU    1H          ;BASE OF CRT I/O PORTS
         IF     TTY         ;ASSEMBLE RELATIVE TO
                            ;TTYBASE
CONIN    EQU    TTYBASE     ;CONSOLE INPUT
CONOUT   EQU    TTYBASE+1   ;CONSOLE OUTPUT
         ENDIF
;        IF     NOT TTY     ;ASSEMBLE RELATIVE TO
                            ;CRTBASE
CONIN    EQU    CRTBASE     ;CONSOLE INPUT
CONOUT   EQU    CRTBASE+1   ;CONSOLE OUTPUT
         ENDIF
         ...
         MVI    C,CONIN     ;READ CONSOLE DATA
         CALL   BDOS
         MVI    C,CONOUT    ;WRITE CONSOLE DATA
         CALL   BDOS
```

In this case, the program assembles for an environment where a teletype is connected, based at port 10H.

The statement defining TTY can be changed to

```
TTY EQU FALSE
```

In this case, the program assembles for a CRT based at port 20H.

3.4.6 The DB Directive

Syntax:

label DB e#1,e#2,...,e#n

The DB directive lets you define initialized storage areas in single- precision byte format.

e#1 through e#n are either expressions that evaluate to 8-bit values (the high-order bit must be zero) or are ASCII strings of a length no greater than 64 characters. There is no restriction on the number of expressions included on a single source line. The expressions are evaluated and placed sequentially into the machine code file following the last program address generated by the assembler. String characters are similarly placed into memory starting with the first character and ending with the last character. Strings of length greater than two characters cannot be used as operands in more complicated expressions.

Note: ASCII characters are always placed in memory with the parity bit reset (0). There is no translation from lower- to upper-case within strings. The optional label can be used to reference the data area throughout the remainder of the program. The following are examples of valid DB statements:

```
data:       DB      0,1,2,3,4,5
            DB      data and 0ffh,5,377Q,1+2+3+4

sign-on:    DB      'please type your name',cr,lf,0
            DB      'AB' SHR 8, 'C', 'DE' AND 7FH
```

3.4.7 The DW Directive

Syntax:

label DW e#1,e#2,...,e#n

The DW statement is similar to the DB statement except double-precision two-byte words of storage are initialized.

e#1 through e#n are expressions that evaluate to 16-bit results. ASCII strings of one or two characters are allowed, but strings longer than two characters are disallowed. In all cases, the data storage is consistent with the 8080 processor; the least significant byte of the expression is stored first in memory, followed by the most significant byte. The following are examples of DW statements:

```
doub:       DW      0ffefh,doub+4,signon-$,255+255
            DW      'a', 5, 'ab', 'CD', 6 shl 8 or 11b.
```

3.4.8 The DS Directive

Syntax:

label DS expression

The DS statement is used to reserve an area of uninitialized memory.

The label is optional. The assembler begins subsequent code generation after the area reserved by the DS. Thus, the DS statement given above has exactly the same effect as the following statement:

```
label:   EQU $ ;LABEL VALUE IS CURRENT CODE LOCATION
         ORG $+expression ;MOVE PAST RESERVED AREA
```

3.5 Operation Codes

Assembly-language operation codes form the principal part of assembly-language programs and form the operation field of the instruction. In general, ASM accepts all the standard mnemonics for the Intel 8080 microcomputer. Labels are optional on each input line. The individual operators are listed briefly in the following sections. In Tables 3-4 through 3-8, bit values have the following meaning:

- e3 represents a 3-bit value in the range 0-7 that can be one of the predefined registers A, B, C, D, E, H, L, M, SP, or PSW.
- e8 represents an 8-bit value in the range 0-255.
- e16 represents a 16-bit value in the range 0-65535.

These expressions can be formed from an arbitrary combination of operands and operators. In some cases, the operands are restricted to particular values within the allowable range, such as the PUSH instruction. These cases are noted.

In the sections that follow, each operation code is listed in its most general poem, along with a specific example, a short explanation, and special restrictions.

3.5.1 Jumps, Calls, and Returns

The Jump, Call, and Return instructions allow several different forms that test the condition flags set in the 8080 microcomputer CPU. The forms are shown in Table 3-4.

Table 3-4. Jumps, Calls, and Returns

Form with Bit Values	Example	Meaning
JMP e16	JMP L1	Jump unconditionally to label
JNZ e16	JNZ L2	Jump on nonzero condition to label
JZ e16	JZ 100H	Jump on zero condition to label
JNC e16	JNC L1+4	Jump no carry to label
JC e16	JC L3	Jump on carry to label
JPO e16	JPO $+8	Jump on parity odd to label
JPE e16	JPE L4	Jump on even parity to label
JP e16	JP GAMMA	Jump on positive result to label
JM e16	JM a1	Jump on minus to label
CALL e16	CALL S1	Call subroutine unconditionally
CNZ e16	CNZ S2	Call subroutine on nonzero condition
CZ e16	CZ 100H	Call subroutine on zero condition
CNC e16	CNC S1+4	Call subroutine if no carry set
CC e16	CC S3	Call subroutine if carry set
CPO e16	CPO $+8	Call subroutine if parity odd
CPE e16	CPE $4	Call subroutine if parity even

(continued)

Table 3-4. Jumps, Calls, and Returns (continued)

Form with Bit Values	Example	Meaning
CP e16	CP GAMMA	Call subroutine if positive result
CM e16	CM bl$c2	Call subroutine if minus flag
RST e3	RST 0	Programmed restart, equivalent to CALL 8*e3, except one byte call
RET		Return from subroutine
RNZ		Return if nonzero flag set
RZ		Return if zero flag set
RNC		Return if no carry
RC		Return if carry flag set
RPO		Return if parity is odd
RPE		Return if parity is even
RP		Return if positive result
RM		Return if minus flag is set

3.5.2 Immediate Operand Instructions

Several instructions are available that load single- or double- precision registers or single-precision memory cells with constant values, along with instructions that perform immediate arithmetic or logical operations on the accumulator (register A). Table 3-5 describes the immediate operand instructions.

Table 3-5. Immediate Operand Instructions

Form with Bit Values	Example	Meaning
MVI e3,e8	`MVI B,255`	Move immediate data to register A, B, C, D, E, H, L, or M (memory)
ADI e8	`ADI 1`	Add immediate operand to A without carry
ACI e8	`ACI 0FFH`	Add immediate operand to A with carry
SUI e8	`SUI L + 3`	Subtract from A without borrow (carry)
SBI e8	`SBI L AND 11B`	Subtract from A with borrow (carry)
ANI e8	`ANI $ AND 7FH`	"Logical and" A with immediate data
XRI e8	`XRI 1111$0000B`	"Exclusive or" A with immediate data
ORI e8	`ORI L AND 1+1`	"Logical or" A with immediate data
CPI e8	`CPI 'a'`	Compare A with immediate data, same as SUI except register A not changed.
LXI e3,e16	`LXI B,100H`	Load extended immediate to register pair. e3 must be equivalent to B, D, H, or SP.

3.5.3 Increment and Decrement Instructions

The 8080 provides instructions for incrementing or decrementing single- and double-precision registers. The instructions are described in Table 3-6.

Table 3-6. Increment and Decrement Instructions

Form with Bit Value	Example	Meaning
INR e3	`INR E`	Single-precision increment register. e3 produces one of A, B, C, D, E, H, L, M.
DCR e3	`DCR A`	Single-precision decrement register. e3 produces one of A, B, C, D, E, H, L, M.
INX e3	`INX SP`	Double-precision increment register pair. e3 must be equivalent to B, D, H, or SP.
DCX e3	`DCX B`	Double-precision decrement register pair. e3 must be equivalent to B, D, H, or SP.

3.5.4 Data Movement Instructions

Instructions that move data from memory to the CPU and from CPU to memory are given in the following table.

Table 3-7. Data Movement Instructions

Form with Bit Value	Example	Meaning
MOV e3,e3	MOV A,B	Move data to leftmost element from rightmost element. e3 produces on of A, B, C, D, E, H, L, or M. MOV M,M is disallowed.
LDAX e3	LDAX B	Load register A from computed address. e3 must produce either B or D.
STAX e3	STAX D	Store register A to computed address. e3 must produce either B or D.
LHLD e16	LHLD L1	Load HL direct from location e16. Double-precision load to H and L.
SHLD e16	SHLD L5+x	Store HL direct to location e16. Double-precision store from H and L to memory.
LDA e16	LDA Gamma	Load register A from address e16.
STA e16	STA X3-5	Store register A into memory at e16.
POP e3	POP PSW	Load register pair from stack, set SP. e3 must produce one of B, D, H, or PSW.
PUSH e3	PUSH B	Store register pair into stack, set SP. e3 must produce on of B, D, H, or PSW.
IN e8	IN 0	Load register A with data from port e8.
OUT e8	OUT 255	Send data from register A to port e8.
XTHL		Exchange data from top of stack with HL.
PCHL		Fill program counter with data from HL.
SPHL		Fill stack pointer with data from HL.
XCHG		Exchange DE pair with HL pair.

3.5.5 Arithmetic Logic Unit Operations

Instructions that act upon the single-precision accumulator to perform arithmetic and logic operation are given in the following table.

Table 3-8. Arithmetic Logic Unit Operations

Form with Bit Value	Example	Meaning
ADD e3	ADD B	Add register given by e3 to accumulator without carry. e3 must produce one of A, B, C, D, E, H, or L.
ADC e3	ADC L	Add register to A with carry, e3 as above.
SUB e3	SUB H	Subtract reg e3 from A without carry, e3 is defined as above.
SBB e3	SBB 2	Subtract register e3 from A with carry, e3 defined as above.
ANA e3	ANA 1+1	"Logical and" reg with A, e3 as above.
XRA e3	XRA A	"Exclusive or" with A, e3 as above.
ORA e3	ORA B	"Logical or" with A, e3 defined as above.
CMP e3	CMP H	Compare register with A, e3 as above.
DAA		Decimal adjust register A based upon last arithmetic logic unit operation.
CMA		Complement the bits in register A.
STC		Set the carry flag to 1.
CMC		Complement the carry flag.
RLC		Rotate bits left, (re)set carry as a side effect. High-order A bit becomes carry.
RRC		Rotate bits right, (re)set carry as side effect. Low-order A bit becomes carry.

(continued)

Table 3-8. Arithmetic Logic Unit Operations (continued)

Form with Bit Value	Example	Meaning
RAL		Rotate carry/A register to left. Carry is involved in the rotate.
RAR		Rotate carry/A register to right. Carry is involved in the rotate.
DAD e3	DAD B	Double-precision add register pair e3 to HL. e3 must produce B, D, H, or SP.

3.5.6 Control Instructions

The control instructions are:

- HLT halts the 8080 processor.
- DI disables the interrupt system.
- EI enables the interrupt system.
- NOP means no operation.

3.6 Error Messages

When errors occur within the assembly-language program, they are listed as single-character flags in the leftmost position of the source listing. The line in error is also echoed at the console. The error codes are listed in the following table.

Table 3-9. Error Codes

Error Code	Meaning
D	Data error: element in data statement cannot be placed in the specified data area.
E	Expression error: expression is ill-formed and cannot be computed at assembly time.
L	Label error: label cannot appear in this context; might be duplicate label.
N	Not implemented: features that will appear in future ASM versions. For example, macros are recognized, but flagged in this version.
O	Overflow: expression is too complicated (too many pending operators) to be computed and should be simplified.
P	Phase error: label does not have the same value on two subsequent passes through the program.
R	Register error: the value specified as a register is not compatible with the operation code.
S	Syntax error: statement is not properly formed.
V	Value error: operand encountered in expression is improperly formed.

Table 3-10 lists the error messages that are due to terminal error conditions.

Table 3-10. Error Messages

Message	Meaning
NO SOURCE FILE PRESENT	The file specified in the ASM command does not exist on disk or data pack.
NO DIRECTORY SPACE	The directory is full; erase files that are not needed and retry.
SOURCE FILE NAME ERROR	Improperly formed ASM filename, for example, it is specified with ? fields.
SOURCE FILE READ ERROR	Source file cannot be read properly by the assembler; execute a TYPE to determine the point of error.
OUTPUT FILE WRITE ERROR	Output files cannot be written properly; most likely cause is a full disk or data pack, erase and retry.
CANNOT CLOSE FILE	Output file cannot be closed; check to see if disk is write protected.

Section 4
CP/M Dynamic Debugging Tool

4.1 Introduction

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. Invoke the debugger with a command in one of the following forms:

```
DDT
DDT filename.HEX
DDT filename.COM
```

where filename is the name of the program to be loaded and tested. The DDT program is brought into main memory in place of the Console Command Processor (CCP) and resides directly below the Basic Disk Operating System (BDOS) portion of CP/M. Refer to Section 5 for standard memory organization. The BDOS starting address, located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area (TPA) size.

The second and third forms of the DDT command perform the same actions as the first, except they also load the specified HEX or COM file. The action is identical to the following sequence of commands:

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

where the I and R commands set up and read the specified program to test.

Upon initiation, DDT prints a sign-on message in the form:

```
DDT VER 2.2
```

Following the sign-on message, DDT prompts you with the hyphen character, -, and waits for input commands from the console. You can type any of several single-character commands, followed by a carriage return to execute the command. Each line of input can be line-edited using the following standard CP/M controls:

Table 4-1. Line-editing Controls

backspace	Result
backspace	removes the last character typed, or the character before the cursor
CTRL-U	removes the entire line, ready for retyping
CTRL-C	reboots system

Commands can be up to 32 characters in length. An automatic carriage return is inserted as character 33. Commands having less than 32 characters must be followed by a manually inserted carriage return. The first character determines the command type. Table 4-2 describes DDT commands.

Table 4-2. DDT Commands

Command Character	Result
A	enters assembly-language mnemonics with operands.
D	displays memory in hexadecimal and ASCII.
F	fills memory with constant data.
G	begins execution with optional breakpoints.
I	sets up a standard input File Control Block.
L	lists memory using assembler mnemonics.
M	moves a memory segment from source to destination.
R	reads a program for subsequent testing.
S	substitutes memory values.
T	traces program execution.
U	untraced program monitoring.
X	examines and optionally alters the CPU state.

In some cases, the command character is followed by zero, one, two, or three hexadecimal values, which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. The commands are executed when the carriage return is typed.

At any point in the debug run, you can stop execution of DDT. Use either a CTRL-C or G0 (jump to location 0000H). Save the current memory image by using a SAVE command of the form:

```
SAVE n filename. COM
```

where n is the number of pages (256 byte blocks) to be saved. The number of blocks is determined by taking the high-order byte of the address in the TPA and converting this number to decimal. For example, if the highest address in the TPA is 1213H, the number of pages is 12H or 18 in decimal. You could type a CTRL-C during the debug run, returning to the CCP level, followed by

```
SAVE 18 X.COM
```

The memory image is saved as X.COM and can be directly executed by typing the name X. The memory image can be recalled with the command

```
DDT X.COM
```

which reloads the previously saved program from location 100H through page 18, 12FFH. The CPU state is not a part of the COM file; thus, the program must be re-started from the beginning to test it properly.

4.2 DDT Commands

The individual commands are described in this section. In each case, you must wait for the hyphen prompt character before entering the command. In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, the the numbers are represented by lower-case letters. These numbers are always assumed to be in a hexadecimal radix and from one to four digits in length. Longer numbers are automatically truncated on the right.

Many of the commands operate upon a CPU state that corresponds to the program under test. The CPU state holds the registers of the program being debugged and initially contains zeros for all registers and flags except for the program counter, P, and stack pointer, S, which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded, see the I and R commands.

4.2.1 The A (Assembly) Command

Syntax:

As

DDT allows in-line assembly language to be inserted into the current memory image using the A command where s is the hexadecimal starting address for the in-line assembly. DDT prompts the console with the address of the next instruction to fill and reads the console, looking for assembly-language mnemonics followed by register references and operands in absolute hexadecimal form. Each successive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, you can review the memory segment using the DDT disassembler. (See the L command).

The assembler/disassembler portion of DDT can be overlaid by the transient program being tested. In this case, the DDT program responds with an error condition if the A and L commands are used.

4.2.2 The D (Display) Command

Syntax:

D
Ds
Ds,f

The D command allows you to view the contents of memory in hexadecimal and ASCII formats.

In the first form, memory is displayed from the current display address, initially 100H, and continues for 16 display lines. Each display line takes the following form:

aaaa bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb cccccccccccccccc

aaaa is the display address in hexadecimal. bb represents data present in memory starting at aaaa. The ASCII characters are to the right starting at aaaa represented by the sequence of character C where nongraphic characters are printed as a period. Both upper- and lower-case alphabetics are displayed as upper-case on a console device that supports only upper-case. Each display line gives the values of 16 bytes of data, with the first line truncated so that the next line begins at an address that is a multiple of 16.

The second form of the D command is similar to the first, except that the display address is first set to address s.

The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pressing the RETURN key.

4.2.3 The F (Fill) Command

Syntax:

Fs,f,c

s is the starting address, f is the final address, and c is a hexadecimal byte constant. DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f, the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.

4.2.4 The G (Go) Command

Syntax:

G
Gs
Gs,b
Gs,b,c
G,b
G,b,c

A program is executed using the G command, with up to two optional breakpoint addresses.

The first form executes the program at the current value of the program counter in the current machine state, with no breakpoints set. The current program counter can be viewed by typing an X or XP command.

The second form is similar to the first, except that the program counter in the current machine state is set to address s before execution begins.

The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed.

The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and clears both breakpoints. The last two forms take the program counter from the current machine state and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. There is no intervention between the starting address and the break address by DDT. Upon encountering a breakpoint, DDT stops execution and displays

```
*d
```

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. Breakpoints must differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, the commands:

G,1234
G400,400

produce an immediate breakpoint without executing any instructions.

4.2.5 The I (Input) Command

Syntax:

Ifilename
Ifilename.typ

The I command inserts a filename into the default File Control Block (FCB) at 5CH. The FCB created by CP/M for transient programs is placed at this location (see Section 5). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. This filename is also used by DDT for reading additional HEX and COM files.

If the second form is used and the filetype is either HEX or COM, subsequent R commands can be used to read the pure binary or hex format machine code. Section 4.2.8 gives further details.

4.2.6 The L (List) Command

Syntax:

L
Ls
Ls,f

The L command is used to list assembly-language mnemonics in a particular program region.

L
Ls
Ls,f

The first form lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. If an execution breakpoint is encountered, the list address is set to the current value of the program counter (G and T commands). Long displays can be aborted by pressing the RETURN key during the list process.

4.2.7 The M (Move) Command

Syntax:

Ms,f,d

The M command moves a block of program or data areas from one location to another in memory.

s is the start address of the move, f is the final address, and d is the destination address. Data is first moved from s to d, and both addresses are incremented. If s exceeds f, the move operation stops; otherwise, the move operation is repeated.

4.2.8 The R (Read) Command

Syntax:

R
Rb

The R command is used in conjunction with the I command to read COM and HEX files from the disk into the transient program area (TPA) in preparation for debugging.

b is an optional address that is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H through 0FFH (the first page of memory). If b is omitted, then b=0000 is assumed. The R command requires a previous I command, specifying the name of a HEX or COM file. The load address for each record is obtained from each individual HEX record, while an assumed load address of 100H is used for COM files. Any number of R commands can be issued following the I command to reread the program under test, if the tested program does not destroy the default area at 5CH. Any file specified with the filetype COM is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced with the ASM command).

The command,

```
DDT filename.filetype
```

which initiates the DDT program, equals to the following commands:

```
DDT
-Ifilename.filetype
-R
```

Whenever the R command is issued, DDT responds with either the error indicator ? (file cannot be opened, or a checksum error occurred in a HEX file) or with a load message. The load message takes the form:

```
NEXT PC
nnnn pppp
```

where nnnn is the next address following the loaded program and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

4.2.9 The S (Set) Command

Syntax:

Ss

The S command allows memory locations to be examined and altered. s is the hexadecimal starting address for the alteration of memory to be examined and/or altered. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in memory. If a carriage return is typed, the data is not altered. If a byte value is typed, the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period or an invalid input value is detected.

4.2.10 The T (Trace) Command

Syntax:

T
Tn

The T command allows selective tracing of program execution for 1 to 65535 program steps.

In the first form, the CPU state is displayed and the next program step is executed. The program terminates immediately, with the termination address displayed as

```
*hhhh
```

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint occurs. A breakpoint can be forced in the trace mode by typing shift and delete. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

Program tracing is discontinued at the CP/M interface and resumes after return from CP/M to the program under test. Thus, CP/M functions that access I/O devices (such as the drives) run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real-time because DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but commands that use the breakpoint facility (G, T, and U) accomplish the break using an RST 7 instruction, which means that the tested program cannot use this interrupt location. The trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

To get control back to DDT during a trace, press the RETURN key. This ensures that the trace for the current instruction is completed before interruption.

4.2.11 The U (Untrace) Command

Syntax:

U
Un

The U command is identical to the T command, except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (0FFFFH) steps to be executed in monitored mode and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

4.2.12 The X (Examine) Command

Syntax:

X
Xr

The X command allows selective display and alteration of the current CPU state for the program under test. r is one of the 8080 CPU registers listed in the following table.

Table 4-3. CPU Registers

Register	Meaning	Value
C	Carry flag	(0/1)
Z	Zero flag	(0/1)
M	Minus flag	(0/1)
E	Even parity flag	(0/1)
I	Interdigit carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack pointer	(0-FFFF)
P	Program counter	(0-FFFF)

In the first case, the CPU register state is displayed in the format:

```
CfZfMfEflf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst
```

where f is a 0 or 1 flag value, bb is a byte value, and dddd is a double-byte quantity corresponding to the register pair. The inst field contains the disassembled instruction that occurs at the location addressed by the CPU state's program counter.

The second form allows display and alteration of register values, where r is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, the flag or register value is not altered. If a value in the proper range is typed, the flag or register value is altered. BC, DE, and HL are displayed as register pairs. The entire register pair must be typed when B, C, or the BC pair is altered.

4.3 Implementation Notes

The organization of DDT allows certain nonessential portions to be overlaid to gain a larger transient program area (TPA) for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the CCP. The assembler/disassembler is loaded with the DDT nucleus, but can be overlayed unless it is used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus, which, in turn, contains a JMP instruction to the BDOS. Thus, programs that use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than of the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area (TPA). If the A, L, T, or X commands are used during the debugging process, the DDT program again alters the address field at 6H to include this module, further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a ? in response) and the trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.

Section 5
CP/M 2.2 System Interface

5.1 Introduction

This chapter describes CP/M system organization including the structure of memory and system entry points. This section provides the information you need to write programs that operate under CP/M and use the peripheral and disk I/O facilities of the system.

CP/M is logically divided into four parts, called the Basic Input/Output System (BIOS), the Basic Disk Operating System (BDOS), the Console Command Processor (CCP), and the Transient Program Area (TPA). The BIOS is a hardware-dependent module that defines the exact low level interface with a particular computer system that is necessary for peripheral device I/O.

The BIOS and BDOS are logically combined into a single module with a common entry point and referred to as the FDOS. The CCP is a distinct program that uses the FDOS to provide a human-oriented interface with the information that is cataloged on the back-up storage device. The TPA is an area of memory, not used by the FDOS and CCP, where various nonresident operating system commands and user programs are executed. The lower portion of memory is reserved for system information and is detailed in later sections. Memory organization of the CP/M system is shown in Figure 5-1.

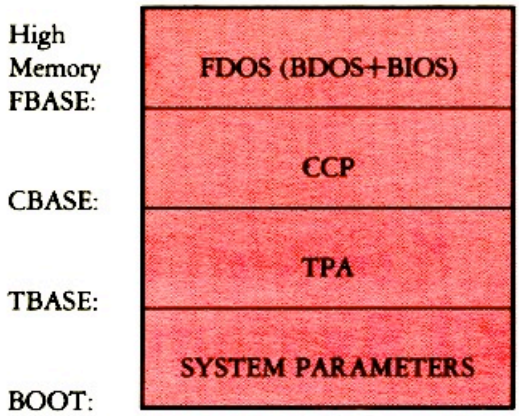


Figure 5-1. CP/M Memory Organization

BOOT is located at 0000H, which is the base of random access memory. The machine code found at location BOOT performs a system warm start, which loads and initializes the programs and variables necessary to return control to the CCP. Transient programs jump to location BOOT to return control to CP/M at the command level. TBASE=0100H, which is location 0100H. The principal entry point to the FDOS is at location 0005H where a jump to FBASE is found. The address field at 0006H contains the value of FBASE and can be used to determine the size of available memory, assuming that the CCP is being overlaid by a transient program.

Transient programs are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt. Each command line takes one of the following forms:

```
command
command file1
command file1 file2
```

where command is either a built-in function, such as DIR or TYPE, or the name of a transient command or program. If the command is a built-in function of CP/M, it is executed immediately. Otherwise, the CCP searches the currently addressed disk or digital data pack for a file by the name

```
command.COM
```

If the file is found, it is assumed to be a memory image of a program that executes in the TPA and originates at 100H in memory. The CCP loads the COM file from the disk into memory starting at 100H and extending up to CBASE, if necessary.

If the command is followed by one or two file specifications, the CCP prepares one or two File Control Block (FCB) names in the system parameter area. These optional FCBs are in the form necessary to access files through the FDOS and are described in Section 5.2.

The transient program receives control from the CCP and begins execution, using the I/O facilities of the FDOS. The transient program is called from the CCP. Thus, it can simply return to the CCP upon completion of its processing, or can jump to BOOT to pass control back to CP/M. In the first case, the transient program must not use memory above CBASE. In the second case, memory up through FBASE-1 can be used.

The transient program can use the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the disk subsystem. The I/O system is accessed by passing a function number and an information address to CP/M through the FDOS entry point at 0005H. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB to the CP/M FDOS. The FDOS performs the operation and returns with either a disk read completion indication or an error number indicating that the disk read was unsuccessful.

5.2 Operating System Call Conventions

This section provides detailed information for performing direct operating system calls from user programs.

CP/M facilities available for access by transient programs fall into two general categories: simple device I/O and disk file I/O. The simple device operations are

- read a console character
- write a console character
- read a sequential character
- write a sequential character
- get or set I/O status
- print console buffer
- interrogate console ready
- write list device character
- read console buffer.

The following BDOS operations perform disk or data pack I/O:

- disk/data pack system reset
- drive selection
- file creation
- file close
- directory search
- file delete
- file rename
- random or sequential read
- random or sequential write
- interrogate available disks/data pack
- interrogate selected disk/data pack
- set DMA address
- set/reset file indicators.

Access to the BDOS functions is accomplished by passing a function number and information address through the primary point at location 0005H. In general, the function number is passed in register C with the information address in the double byte pair DE. Single byte values are returned in register A, with double byte values returned in HL; a zero value is returned when the function number is out of range. For compatibility, register A = L and register B = H upon return in all cases. CP/M functions and their numbers are listed below.

0	System Reset	19	Delete File
1	Console Input	20	Read Sequential
2	Console Output	21	Write Sequential
3	Reader Input	22	Make File
4	Punch Output	23	Rename File
5	List Output	24	Return Login Vector
6	Direct Console I/O	25	Return Current Disk*
7	Get I/O Byte	26	Set DMA Address
8	Set I/O Byte	27	Get Addr(Alloc)
9	Print String	28	Write Protect Disk*
10	Read Console Buffer	29	Get R/0 Vector
11	Get Console Status	30	Set File Attributes
12	Return Version Number	31	Get Addr(Disk Parms)
13	Reset Disk System	32	Set/Get User Code
14	Select Disk	33	Read Random
15	Open File	34	Write Random
16	Close File	35	Compute File Size
17	Search for First	36	Set Random Record
18	Search for Next	37	Reset Drive

40 Write Random with Zero Fill

* or data pack

Upon entry to a transient program, the CCP leaves the stack pointer set to an eight-level stack area with the CCP return address pushed onto the stack, leaving seven levels before overflow occurs. Although this stack is usually not used by a transient program (most transients return to the CCP through a jump to location 0000H), it is large enough to make CP/M system calls because the FDOS switches to a local stack at system entry. For example, the assembly-language program segment below reads characters continuously until an asterisk is encountered. Then, control returns to the CCP.

```
BDOS    EQU     0005H       ;STANDARD CP/M ENTRY
CONIN   EQU     1           ;CONSOLE INPUT FUNCTION
;
        ORG     0100H       ;BASE OF TPA
NEXTC:  MVI     C,CONIN     ;READ NEXT CHARACTER
        CALL    BDOS        ;RETURN CHARACTER IN <A>
        CPI     '*'         ;END OF PROCESSING?
        JNZ     NEXTC       ;LOOP IF NOT
        RET                 ;RETURN TO CCP
        END
```

CP/M implements a named file structure on each disk, providing a logical organization that allows any particular file to contain any number of records from completely empty to the full capacity of the drive. Each drive is logically distinct with a disk directory and file data area. The disk filenames are in three parts: the drive select code, the filename (consisting of one to eight nonblank characters), and the filetype (consisting of zero to three nonblank characters). The filetype names the generic category of a particular file, while the filename distinguishes individual files in each category. The filetypes listed in Table 5-1 name a few generic categories that have been established.

Table 5-1. CP/M Filetypes

Filetype	Meaning
ASM	Assembler Source
PRN	Printer Listing
HEX	Hex Machine Code
BAS	Basic Source File
INT	Intermediate Code
COM	Command File
PLI	PL/I Source File
REL	Relocatable Module
TEX .	TEX Formatter Source
BAK	ED Source Backup
SYM	SID Symbol File
$$$	Temporary File

Source files are treated as a sequence of ASCII characters, where each line of the source file is followed by a carriage return, and line-feed sequence (0DH followed by 0AH). Thus, one 128-byte CP/M record can contain several lines of source text. The end of an ASCII file is denoted by a CTRL-Z character (1AH) or a real end-of-file returned by the CP/M read operation. CTRL-Z characters embedded within machine code files (for example, COM files) are ignored and the end-of-file condition returned by CP/M is used to terminate read operations.

Files in CP/M can be thought of as a sequence of up to 65536 records of 128 bytes each, numbered from 0 through 65535, thus allowing a maximum of 8 megabytes per file. Note, however, that although the records may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are divided into 16K byte segments called logical extents, so that counters are easily maintained as 8-bit values. The division into extents is discussed in the paragraphs that follow, however, they are not particularly significant because each extent is automatically accessed in both sequential and random access modes.

In the file operations starting with Function 15, DE usually addresses a FCB. Transient programs often use the default FCB area reserved by CP/M at 005CH for simple file operations. The basic unit of file information is a 128-byte record used for all file operations. Thus, a default location for disk I/O is provided by CP/M at location 0080H which is the initial default DMA address. See Function 26.

The FCB data area consists of a sequence of 33 bytes for sequential access and a series of 36 bytes in the case when the file is accessed randomly. The default FCB, normally located at 005CH, can be used for random access files, because the three bytes starting at 007DH are available for this purpose. Figure 5-2 shows the FCB format with the following fields:

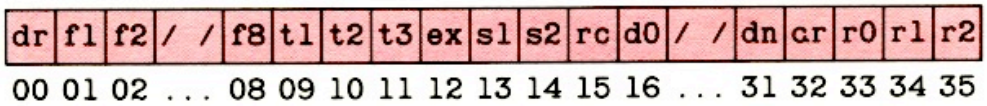


Figure 5-2. File Control Block Format

The following table lists and describes each of the fields in the File Control Block.

Table 5-2. File Control Block Fields

Field	Definition
dr	drive code (0-16) 0 = use default drive for file 1 = auto disk select drive A, 2 = auto disk select drive B, . . . 16 = auto disk select drive P. (ADAM BIOS recognizes only drives A,B,C,D and M.)
f1...f8	contain the filename in ASCII upper-case, with high bit = 0
t1, t2, t3	contain the filetype in ASCII upper-case, with high bit = 0 t1', t2', and t3' denote the bit of these positions, t1' = 1 = >Read-Only file, t2' = 1 = >SYS file, no DIR list
ex	contains the current extent number, normally set to 00 by the user, but in range 0-31 during file I/O
s1	reserved for internal system use
s2	reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH
rc	record count for extent ex; takes on values from 0-127
d0...dn	filled in by CP/M; reserved for system use
cr	current record to read or write in a sequential file operation; normally set to zero by user
r0, r1, r2	optional random record number in the range 0-65535, with overflow to r2, r0, r1 constitute a 16-bit value with low byte r0, and high byte r1

FCBs are stored in a directory area of the disk and are brought into central memory before you proceed with file operations (see the OPEN and MAKE functions). The memory copy of the FCB is updated as file operations take place and later recorded permanently on disk at the termination of the file operation, (see the CLOSE command).

The CCP constructs the first 16 bytes of two optional FCBs for a transient by scanning the remainder of the line following the transient name, denoted by file1 and file2 in the prototype command line described above, with unspecified fields set to ASCII blanks. The first FCB is constructed at location 005CH and can be used as is for subsequent file operations. The second FCB occupies the d0...dn portion of the first FCB and must be moved to another area of memory before use. If, for example, the following command line is typed:

```
PROGNAME B:X.ZOT Y.ZAP
```

the file PROGNAME.COM is loaded into the TPA, and the default FCB at 005CH is initialized to drive code 2, filename X, and filetype ZOT. The second drive code takes the default value 0, which is placed at 006CH, with the filename Y placed into location 006DH and filetype ZAP located 8 bytes later at 0075H. All remaining fields through cr are set to zero. It is your responsibility to move this second filename and filetype to another area, usually a separate file control block, before opening the file that begins at 005CH, because the open operation overwrites the second name and type.

If no filenames are specified in the original command, the fields beginning at 005DH and 006DH contain blanks. In all cases, the CCP translates lower-case alphabetics to upper-case to be consistent with the CP/M file naming conventions.

As an added convenience, the default buffer area at location 0080H is initialized to the command line tail typed by the operator following the program name. The first position contains the number of characters, with the characters themselves following the character count. Given the above command line, the area beginning at 0080H is initialized as follows:

BOOT + 0080H:

```
+00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +A +B +C +D +E
 E   ''  'B' ':' 'X' '.' 'Z' 'O' 'T'  ''  'Y' '.' 'Z' 'A' 'P'
```

where the characters are translated to upper-case ASCII with uninitialized memory following the last valid character. Again, it is the responsibility of the programmer to extract the information from this buffer before any file operations are performed, unless the default DMA address is explicitly changed.

5.2.1 BDOS Calling Conventions

Individual functions are described in detail in the pages that follow. These functions are summarized on table 5-3.

Table 5-3. BDOS System Function Summary

Function Number Decimal	Hex	Function Name	Input	Output
0	0	System Reset	none	none
1	1	Console Input	none	A = ASCII char
2	2	Console Output	E = char	none
3	3	Reader Input	none	A = ASCII char
4	4	Punch Output	E = char	none
5	5	List Output	E = char	none
6	6	Direct Console I/O*	E = 0FFH (input or 0FEH (status) or char (output)	A = char status
7	7	Get I/O Byte	none	A = I/O byte Value
8	8	Set I/O Byte	E = I/O Byte	none
9	9	Print String	DE = Buffer Address	none
10	A	Read Console Buffer	DE = Buffer Address	Console Characters in Buffer
11	B	Get Console Status	none	A = console status
12	C	Return Version Number	none	HL: Version Number
13	D	Reset Disk System	none	none
14	E	Select Disk	E = Disk Number	none
15	F	Open File	DE = FCB Address	A = Directory Code
16	10	Close File	DE = FCB Address	A = Directory Code
17	11	Search For First	DE = FCB Address	A = Directory Code
18	12	Search For Next	none	A = Directory Code
19	13	Delete File	DE = FCB Address	A = Directory Code
20	14	Read Sequential	DE = FCB Address	A = Directory Code
21	15	Write Sequential	DE = FCB Address	A = Directory Code
22	16	Make File	DE = FCB Address	A = Directory Code
23	17	Rename File	DE = FCB Address	A = Directory Code
24	18	Return Login Vector	none	HL = Login Vector**
25	19	Return Current Disk	none	A = Current Disk Number
26	1A	Set DMA Address	DE = DMA Address	none
27	1B	Get ADDR (ALLOC)	none	HL = ALLOC Address**
28	1C	Write Protect Disk	none	none
29	1D	Get Read/only Vector	none	HL = R/O Vector Value**
30	1E	Set File Attributes	DE = FCB Address	A = Directory Code
31	1F	Get ADDR (Disk Parms)	none	HL = DPB Address
32	20	Set/Get User Code	E = 0FFH for Get E = 00 to 0FH for Set	User Number
33	21	Read Random	DE = FCB Address	A = Return Code
34	22	Write Random	DE = FCB Address	A = Return Code
35	23	Compute File Size	DE = FCB Address	r0, r1, r2
36	24	Set Random Record	DE = FCB Address	r0, r1, r2
37	25	Reset Drive***	DE = Drive Vector	A = 0
38	26	Access Drive	not supported	
39	27	Free Drive	not supported	
40	28	Write Random with Fill	DE = FCB Address	A = Return Code

* Function 6 must be used with functions 1,2,9, or 11.

** A=L and B=H upon return *** Default drive cannot be reset.

These abbreviations are used in the BDOS function summary.

Addr = Address
Alloc = Allocation
Attrib = Attribute
Buf = Buffer
Char = ASCII Character
Con = Console
Cur = Current
Dir = Directory
Dsk = Disk
Err = Error
Parm = Parameter
Prot = Protect
Ran = Random
Rec = Record
Rtn = Return
Seq = Sequential
Stat = Status
Sys = System
Usr = User
Vect = Vector
Vers = Version

FUNCTION 0: SYSTEM RESET

Entry Parameters:
Register C: 00H

The System Reset function returns control to the CP/M operating system at the CCP level. The CCP reinitializes the disk subsystem by selecting and logging-in disk drive A.

FUNCTION 1: CONSOLE INPUT

Entry Parameters:
Register C: 01H

Returned Value:
Register A: ASCII Character

The Console Input function reads the next console character to register A. Graphic characters, along with carriage return, line-feed, and back space (CTRL-H) are echoed to the console. Tab characters, CTRL-I, move the cursor to the next tab stop. A check is made for start/stop scroll, CTRL-S, and start/stop printer echo, CTRL-P. The FDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

FUNCTION 2: CONSOLE OUTPUT

Entry Parameters:
Register C: 02H
Register E: ASCII Character

The ASCII character from register E is sent to the console device. As in Function 1, tabs are expanded and checks are made for start/stop scroll and printer echo.

FUNCTION 3: READER INPUT

Entry Parameters:
Register C: 03H

Returned Value:
Register A: ASCII Character

The Reader Input function reads the next character from the logical reader into register A. See the IOBYTE definition in Chapter 6. Control does not return until the character has been read.

FUNCTION 4: PUNCH OUTPUT

Entry Parameters:
Register C: 04H
Register E: ASCII Character

The Punch Output function sends the character from register E to the logical punch device.

FUNCTION 5: LIST OUTPUT

Entry Parameters:
Register C: 05H
Register E: ASCII Character

The List Output function sends the ASCII character in register E to the logical listing device.

FUNCTION 6: DIRECT CONSOLE I/O

Entry Parameters:
Register C: 06H
Register E: OFFH (input) or
char (output)

Returned Value:
Register A: char or status

Direct Console I/O is supported under CP/M for those specialized applications where basic console input and output are required. Use of this function should, in general, be avoided since it bypasses all of the CP/M normal control character functions (for example, CTRL-S and CTRL-P).

Upon entry to Function 6, register E either contains hexadecimal FF, denoting a console input request, or an ASCII character. If the input value is FF, Function 6 returns A = 00 if no character is ready; otherwise A contains the next console input character.

If the input value in E is not FF, Function 6 assumes that E contains a valid ASCII character that is sent to the console.

Function 6 must not be used in conjunction with other console I/O functions.

FUNCTION 7: GET I/O BYTE

Entry Parameters:
Register C: 07H

Returned Value:
Register A: I/O Byte Value

The Get I/O Byte function returns the current value of IOBYTE in register A. See Chapter 6 for IOBYTE definition.

FUNCTION 8: SET I/O BYTE

Entry Parameters:
Register C: 08H
Register E: I/O Byte Value

The SET I/O Byte function changes the IOBYTE value to that given in register E.

FUNCTION 9: PRINT STRING

Entry Parameters:
Register C: 09H
Registers DE: String Address

The Print String function sends the character string stored in memory at the location given by DE to the console device, until a $ is encountered in the string. Tabs are expanded as in Function 2, and checks are made for start/stop scroll and printer echo.

FUNCTION 10: READ CONSOLE BUFFER

Entry Parameters:
Register C: OAH
Registers DE: Buffer Address

Returned Value:
Console Characters in Buffer

The Read Buffer functions reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either input buffer overflows or a carriage return or line-feed is typed. The Read Buffer takes the form:

```
DE:   +0   +1   +2   +3   +4   +5   +6   +7   +8   ...   +n
      mx   nc   c1   c2   c3   c4   c5   c6   c7   ...   ??
```

where mx is the maximum number of characters that the buffer will hold, 1 to 255, and nc is the number of characters read (set by FDOS upon return) followed by the characters read from the console. If nc < mx, then uninitialized positions follow the last character, denoted by ?? in the above figure. A number of control functions, summarized in Table 5-4, are recognized during line editing.

Table 5-4. Edit Control Characters

Character	Edit Control Function
Backspace	backspaces and erases one character position
CTRL-C	reboots when at the beginning of line
CTRL-E	causes physical end of line
CTRL-H	backspaces one character position
CTRL-J	(line feed) terminates input line
CTRL-M	(return) terminates input line
CTRL-R	retypes the current line after new line
CTRL-U	clears current line
CTRL-X	same as CTRL-U
CTRL- <--	scroll screen left
CTRL- -->	scroll screen right

Certain functions that return the carriage to the leftmost position (for example, CTRL-X) do so only to the column position where the prompt ended. This convention makes operator data input and line correction more legible.

FUNCTION 11: GET CONSOLE STATUS

Entry Parameters:
Register C: 0BH

Returned Value:
Register A: Console Status

The Console Status function checks to see if a character has been typed at the console. If a character is ready, the value 0FFH is returned in register A. Otherwise a 00H value is returned.

FUNCTION 12: RETURN VERSION NUMBER

Entry Parameters:
Register C: 0CH

Returned Value:
Registers HL: Version Number

Function 12 returns a two-byte value with H = 00 designating the CP/M release.

FUNCTION 13: RESET DISK SYSTEM

Entry Parameters:
Register C: 0DH

The Reset Disk function is used to programmatically restore the file system to a reset state where all disks are set to Read-Write. See functions 28 and 29, only disk drive A is selected, and the default DMA address is reset to 0080H. This function can be used, for example, by an application program that requires a disk/data pack change without a system reboot.

FUNCTION 14: SELECT DISK

Entry Parameters:
Register C: 0EH
Register E: Selected Disk

The Select Disk function designates the drive named in register E as the default disk or digital data pack for subsequent file operations, with E = O for drive A, 1 for drive B, and so on. The drive is placed in an on-line status, which activates its directory until the next cold start, warm start, or disk system reset operation. If the disk or digital data pack is changed while it is on-line, the drive automatically goes to a Read-Only status in a standard CP/M environment, see Function 28. FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1, 2, 3, and 13 ignore the selected default drive and directly reference drives A, B, C, D, and M.

FUNCTION 15: OPEN FILE

Entry Parameters:
Register C: 0FH
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Open File operation is used to activate a file that currently exists in the disk or data pack directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte s1 is automatically zeroed) where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included, and bytes ex and s2 of the FCB are zero.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of FCB, thus allowing access to the files through subsequent read and write operations. An existing file must not be accessed until a successful open operation is completed. Upon return, the open function returns a directory code with the value 0 through 3 if the open was successful or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB, the first matching FCB is activated. The current record, (cr) must be zeroed by the program if the file is to be accessed sequentially from the first record.

FUNCTION 16: CLOSE FILE

Entry Parameters:
Register C: 10H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Close File function performs the inverse of the Open File function. Given that the FCB addressed by DE has been previously activated through an open or make function, the close function permanently records the new FCB in the reference disk or digital data pack directory see functions 15 and 22. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the filename cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, the close operation is necessary to record the new directory information permanently.

FUNCTION 17: SEARCH FOR FIRST

Entry Parameters:
Register C: 11H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found; otherwise, 0, 1, 2, or 3 is returned indicating the file is present. When the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A *32 (that is, rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through ex matches the corresponding field of any directory entry on the default or auto-selected drive. If the dr field contains an ASCII question mark, the auto drive select function is disabled and the default disk or digital data pack is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

FUNCTION 18: SEARCH FOR NEXT

Entry Parameters:
Register C: 12H

Returned Value:
Register A: Directory Code

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to Function 17, Function 18 returns the decimal value 255 in A when no more directory items match.

FUNCTION 19: DELETE FILE

Entry Parameters:
Register C: 13H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Delete File function removes files that match the FCB addressed by DE. The filename and type may contain ambiguous references (that is, question marks in various positions), but the drive select code cannot be ambiguous, as in the search and search next functions.

Function 19 returns a decimal 255 if the referenced file or files cannot be found; otherwise, a value in the range 0 to 3 returned.

FUNCTION 20: READ SEQUENTIAL

Entry Parameters:
Register C: 14H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

Given that the FCB addressed by DE has been activated through an Open or Make function, the Read Sequential function reads the next 128-byte record from the file into memory at the current DMA address. The record is read from position cr of the extent, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next read operation. The value 00H is returned in the A register if the read operation was successful, while a nonzero value is returned if no data exist at the next record position (for example, end-of-file occurs).

FUNCTION 21: WRITE SEQUENTIAL

Entry Parameters:
Register C: 15H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

Given that the FCB addressed by DE has been activated through an Open or Make function, the Write Sequential function writes the 128-byte data record at the current DMA address to the file named by the FCB. The record is placed at position cr of the file, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case, newly written records overlay those that already exist in the file. Register A = 00H upon return from a successful write operation, while a non-zero value indicates an unsuccessful write caused by a full disk or digital data pack.

FUNCTION 22: MAKE FILE

Entry Parameters:
Register C: 16H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Make File operation is similar to the Open File operation except that the FCB must name a file that does not exist in the currently referenced disk directory (that is, the one named explicitly by a nonzero dr code or the default disk if dr is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. You must ensure that no duplicate filenames occur. A preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The Make function has the side effect of activating the FCB and thus a subsequent open is not necessary.

FUNCTION 23: RENAME FILE

Entry Parameters:
Register C: 17H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Rename function uses the FCB addressed by DE to change all occurrences of the file named in the first 16 bytes to the file named in the second 16 bytes. The drive code dr at postion 0 is used to select the drive, while the drive code for the new filename at position 16 of the FCB is assumed to be zero. Upon return, register A is set to a value between 0 and 3 if the rename was successful and 0FFH (255 decimal) if the first filename could not be found in the directory scan.

FUNCTION 24: RETURN LOG-IN VECTOR

Entry Parameters:
Register C: 18H

Returned Value:
Registers HL: Log-in Vector

The log-in vector value returned by CP/M is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A. A 0 bit indicates that the drive is not on-line, while a 1 bit marks a drive that is actively on-line as a result of an explicit drive selection or an implicit drive select caused by a file operation that specified a non-zero dr field.

FUNCTION 25: RETURN CURRENT DISK

Entry Parameters:
Register C: 19H

Returned Value:
Register A: Current Disk

Function 25 returns the currently selected default drive number in register A. The drive numbers range from 0 through 4 corresponding to drives A, B, C, D, and M.

FUNCTION 26: SET DMA ADDRESS

Entry Parameters:
Register C: 1AH
Registers DE: DMA Address

DMA is an acronym for Direct Memory Address, which is often used in connection with disk controllers that directly access the memory of the mainframe computer to transfer data to and from the disk or digital data pack subsystem. The DMA address has come to mean the address at which the 128-byte data record resides before a disk or digital data pack write and after a disk or digital data pack read. Upon cold start, warm start, or disk system reset, the DMA address is automatically set to 0080H. The Set DMA function can be used to change this default value to address another area of memory where the data records reside. Thus, the DMA address becomes the value specified by DE until it is changed by a subsequent Set DMA function, cold start, warm start, or disk system reset.

FUNCTION 27: GET ADDR (ALLOC)

Entry Parameters:
Register C: 1BH

Returned Value:
Registers HL: ALLOC Address

An allocation vector is maintained in main memory for each on- line drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the base address of the allocation vector for the currently selected drive. However, the allocation information might be invalid if the selected disk or digital data pack has been marked Read-Only. Although this function is not normally used by application programs, additional details of the allocation vector are found in Chapter 6.

FUNCTION 28: WRITE PROTECT DISK

Entry Parameters:
Register C: 1CH

The Write Protect Disk function provides temporary write protection for the currently selected disk or digital data pack. Any attempt to write to the disk or digital data pack before the next cold or warm start operation produces the message:

```
Bdos Err On d: R/O
```

FUNCTION 29: GET READ-ONLY VECTOR

Entry Parameters:
Register C: 1DH

Returned Value:
Registers HL: R/O Vector Value

Function 29 returns a bit vector in register pair HL, which indicates drives that have the temporary Read-Only bit set. As in Function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to Function 28 or by the automatic software mechanisms within CP/M that detect changed media.

FUNCTION 30: SET FILE ATTRIBUTES

Entry Parameters:
Register C: 1EH
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2') can be set or reset. The DE pair addresses an unambiguous filename with the appropriate attributes set or reset. Function 30 searches for a match and changes the matched directory entry to contain the selected indicators. Indicators f1' through f4' are not currently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations.

FUNCTION 31: GET ADDR (DISK PARMS)

Entry Parameters:
Register C: 1FH

Returned Value:
Registers HL: DPB Address

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the environment changes, if required. Normally, application programs do not require this facility.

FUNCTION 32: SET/GET USER CODE

Entry Parameters:
Register C: 20H
Register E: OFFH (get) or
User Code (set)

Returned Value:
Register A: Current Code or
(no value)

An application program can change or interrogate the currently active user number by calling Function 32. If register E = 0FFH, the value of the current user number is returned in register A, where the value is in the range of 0 to 15. If register E is not 0FFH, the current user number is changed to the value of E, module 16.

FUNCTION 33: READ RANDOM

Entry Parameters:
Register C: 21H
Registers DE: FCB Address

Returned Value:
Register A: Return Code

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the 3-byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). The sequence of 24 bits is stored with the least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (Function 35). Byte r2 must be zero, however, since a nonzero value indicates overflow past the end of file.

Thus, the r0, r1 byte pair is treated as a double-byte, or word value, that contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8- megabyte file. To process a file using random access, the base extent (extent 0) must first be opened. Although the base extent might or might not contain any allocated data, this ensures that the file is properly recorded in the directory and is visible in DIR requests. The selected record number is then stored in the random record field (r0, r1), and the BDOS is called to read the record.

Upon return from the call, register A either contains an error code, as listed below, or the value 00, indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. The record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. In this case, the last randomly read record reread if you switch from random mode to sequential read and the last record is rewritten if you switch to a sequential write operation. You can advance the random record position after each random read or write to obtain the effect of sequential I/O operation.

Error codes returned in register A following a random read are listed below.

```
01   reading unwritten data
02   (not returned in random mode)
03   cannot close current extent
04   seek to unwritten extent
05   (not returned in read mode)
06   seek past physical end of disk
```

Error codes 01 and 04 occur when a random read operation accesses a data block that has not been previously written or an extent that has not been created, which are equivalent conditions. Error code 03 does not normally occur under proper system operation. If it does, it can be cleared by simply rereading or reopening extent zero as long as the disk is not physically write protected. Normally, nonzero return codes can be treated as missing data, with zero return codes indicating operation complete.

FUNCTION 34: WRITE RANDOM

Entry Parameters:
Register C: 22H
Registers DE: FCB Address

Returned Value:
Register A: Return Code

The Write Random operation is initiated similarly to the Read Random call, except that data is written to the disk or digital data pack from the current DMA address. Further, if the disk extent or data block that is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the FCB are set to correspond to the random record that is being written. Again, sequential read or write operations can begin following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. Reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created because of a directory overflow.

FUNCTION 35: COMPUTE FILE SIZE

Entry Parameters:
Register C: 23H
Registers DE: FCB Address

Returned Value: Random Record Field Set

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous filename that is used in the directory scan. Upon return, the random record bytes contain the virtual file size, which is, in effect, the record address of the record following the end of the file. Following a call to Function 35, if the high record byte r2 is 01, the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value as before (r0 is the least significant byte), which is the file size.

Data can be appended to the end of an existing file by simply calling Function 35 to set the random record position to the end of file and then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If the file was created in random mode and holes exist in the allocation, the file can contain fewer records than the size indicates. For example, if only the last record of a 256K file is written in random mode (that is, record number 2047), the virtual size is 2048 records, although only one block of data is actually allocated.

FUNCTION 36: SET RANDOM RECORD

Entry Parameters:
Register C: 24H
Registers DE: FCB Address

Returned Value: Random Record Field Set

The Set Random Record function causes the BDOS automatically to produce the random record position from a file that has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary initially to read and scan a sequential file to extract the positions of various key fields. As each key is encountered, Function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabulating the keys and their record numbers, the user can move instantly to a particular keyed record by performing a random read, using the corresponding random record number that was saved earlier. The scheme is easily generalized for variable record lengths, because the program need only store the buffer-relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of Function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, Function 36 is called, which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

FUNCTION 37: RESET DRIVE

Entry Parameters:
Register C: 25H
Registers DE: Drive Vector

Returned Value:
Register A: 00H

The Reset Drive function allows resetting of specified drives. The passed parameter is a 16-bit vector of drives to be reset; the least significant bit is drive A:.

FUNCTION 40: WRITE RANDOM WITH ZERO FILL

Entry Parameters:
Register C: 28H
Registers DE: FCB Address

Returned Value:
Register A: Return Code

The Write With Zero Fill operation is similar to Function 34, with the exception that a previously unallocated block is filled with zeros before the data is written.

Section 6
CP/M 2.2 Alteration

6.1 Introduction

To achieve device independence, CP/M is separated into three distinct modules:

- BIOS is the Basic I/O System, which is hardware dependent.
- BDOS is the Basic Disk Operating System, which is not dependent upon the hardware configuration.
- CCP is the Console Command Processor, which uses the BDOS.

Of these modules, only the BIOS is dependent upon the particular hardware. This section describes the BIOS operations and tables in brief. For more detailed information, see one of the additional reference books listed in the bibliography at the end of Part B.

All disk-dependent portions of CP/M 2.2 are placed into a BIOS, a disk parameter block.

6.2 Media Organization

ADAM's organization of Digital Data Packs and Disks is the same, with the exception that data packs hold 256 Kbytes of storage while disks hold 160 Kbytes. The media is organized in blocks. Each block is made up of eight 128 byte sectors (one Kbyte).

Data packs and disks have 64 directory entries (two blocks) reserved for them. The RAM disk (64K Memory Expander) has 32 directory entries. Since the RAM disk does not accept the system, it reserves 55K of space for user files.

Media space is divided as follows:

Block	CP/M Module Name
0	Boot
1 to 6	BIOS
7 to 12	CCP and BDOS
13 and 14	Directory
15 up	User Files

Upon cold boot ADAM examines Block 0 which gives instructions for loading BIOS, CCP and BDOS. CCP and BDOS may be overwritten by a program since both are reloaded upon warm boot.

The disk drives are arranged with one block per logical track (passes to Select Track). There are 4 logical tracks for each of the 40 physical tracks on the disk.

6.3 The BIOS Entry Points

The entry points into the BIOS from the cold start loader and BDOS are detailed below. Entry to the BIOS is through a jump vector located at high memory and is reached by the jump at location 0 entering the second position in the table (the jump to WarmBoot). The jump vector is a sequence of 31 jump instructions that send program control to the individual BIOS subroutines. The BIOS subroutines might be empty for certain functions (they might contain a single RET operation).

For example, if the jump vector is at DA00H it takes the form shown below, where the individual jump addresses are given to the left. At 0 there is a jump to DA03H.

```
DA00H              JP   Boot
DA03H              JP   WarmBoot
DA07H              JP   CONSOLEINSTATUS
DA09H              JP   CONSOLEINPUT
DA0CH              JP   CONSOLEOUTPUT
DA0FH              JP   PRINTEROUTPUT
DA12H              JP   PUNCHOUTPUT
DA15H              JP   READERINPUT
DA18H              JP   HomeDisk (no stack used)
DA1BH              JP   SELECTDISK
DA1EH              JP   SelectTrack
DA21H              JP   SelectSector
DA24H              JP   SelectDMA
DA27H              JP   READ
DA2AH              JP   WRITE
DA2DH              JP   LISTSTATUS
DA30H              JP   HomeDisk (sector translate)
DA33H to DA54H          Reserved for Future Use.
DA57H              JP   FlushConsoleInput
DA5AH              JP   VramFill
DA5DH              JP   VramWrite
DA60H              JP   VramRead
DA63H              JP   WriteVDPReg
DA66H              JP   InitVDP
DA69H              JP   AuxWrite
DA6CH              JP   AuxRead
DA6FH              JP   AuxOutStatus
DA72H              JP   AuxInStatus
DA75H              JP   InitAuxDevice
DA78H              JP   ReadlBlock
DA7BH              JP   WritelBlock
DA7EH to DA87H          Reserved for future use
DA8AH              JP   WildCardContinue
```

Listing 6-1. BIOS Entry Points

Each jump address corresponds to a particular subroutine that performs the specific function outlined below. There are three major divisions in the jump table: the system reinitialization, which results from calls on BOOT and WarmBoot; simple character I/O, performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST; and disk or digital data pack I/O, performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

All simple character I/O operations are assumed to be performed in ASCII, upper- and lower-case, with high-order (parity bit) cleared to zero. An end-of-file condition for an input device is given by an ASCII CTRL-Z (1AH). Peripheral devices are seen by CP/M as logical devices and are assigned to physical devices within the BIOS.

To operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines. LIST, PUNCH, and READER can be used by PIP, but not the BDOS.

The following list describes the characteristics of each device.

- CONSOLE is the principal interactive console that communicates with you. It is accessed through CONST, CONIN, and CONOUT. ADAM's console default is its keyboard for input and your television or monitor for output.
- LIST is the principal listing device. The SmartWRITER printer is ADAM's listing device.
- PUNCH is the auxiliary output device. On ADAM, this defaults to the RS232 interface.
- READER is the auxiliary input device. On ADAM, this defaults to the RS232 interface.

A single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. Alternately, the PUNCH and LIST routines can just return, and the READER routine can return with a 1AH (CTRL-Z) in register A to indicate immediate end-of-file.

For added flexibility, you can implement the IOBYTE function, which allows reassignment of physical devices. The IOBYTE function creates a mapping of logical-to-physical devices that can be altered during CP/M processing, see the STAT command in Section 1.6.1.

The IOBYTE function is defined as a single location in memory, location 0003H, is maintained, called IOBYTE, which defines the logical-to-physical device mapping that is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown in the following figure.

Table 6-1. IOBYTE Fields

	Most Significant		Least Significant	
IOBYTE AT 003H	LIST	PUNCH	READER	CONSOLE
	bits 6,7	bits 4,5	bits 2,3	bits 0,1

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. Table 6-2 gives the values that can be assigned to each field.

Table 6-2. IOBYTE Field Values

Value	Meaning
	CONSOLE field (bits 0,1)
0	console is assigned to the console printer device (TTY:)
1	console is assigned to the CRT device (CRT:)
2	batch mode: use the READER as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
3	user-defined console device (UC1:)
	READER field (bits 2,3)
0	READER is the teletype device (TTY:)
1	READER is the high speed reader device (PTR:)
2	user-defined reader #1 (UR1:)
3	user-defined reader #2 (UR2:)
	PUNCH field (bits 4,5)
0	PUNCH is the teletype device (TTY:)
1	PUNCH is the high speed punch device (PTP:)
2	user-defined punch #1 (UP1:)
3	user-defined punch #2 (UP2:)
	LIST field (bits 6,7)
0	LIST is the teletype device (TTY:)
1	LIST is the CRT device (CRT:)
2	LIST is the line printer device (LPT:)
3	user-defined list device (UL1:)

The IOBYTE is handled by the BIOS. No CP/M commands use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H) except for PIP, which allows access to the physical devices, and STAT, which allows logical-physical assignments to be made or displayed. For more information see Section 1.

Disk and data pack I/O is always performed through a sequence of calls on the various disk access subroutines that set up the disk or data pack number to access, the track and sector on a particular disk, and the Direct Memory Access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation.

There is often a single call to SELDSK to select a drive, followed by a number of read or write operations to the selected disk or digital data pack before selecting another drive for subsequent operations. Similarly, there might be a single call to set the DMA address, followed by several calls that read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE routines perform several retries before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it reports the error to you. The HOME subroutine selects track 00 for the next operation and is often treated in exactly the same manner as Select Track with a parameter of 00.

The exact responsibilities of each BIOS entry point subroutine are described on the following pages.

6.3.1 BIOS ENTRY POINT Subroutines

BOOT The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a sign-on message. The IOBYTE function is set at this point. The various system parameters that are set by the Warm Boot entry point must be initialized, and control is transferred to the CCP for inputting user console commands. Note that register C must be set to zero to select drive A.

WarmBoot The WarmBoot entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H. The CP/M system is reloaded from the beginning tracks of drive A. System parameters must be initialized as follows:

location 0,1,2 Set to JP WarmBoot for warm starts

location 3 Set to initial value of IOBYTE.

location 4 High nibble = current user no; low nibble = current drive

location 5,6,7 Set to JMP BDOS, which is the primary entry point to CP/M for programs.

Refer to Section 6.4 for complete details of page zero use. Upon completion of the initialization, the WarmBoot program must branch to the CCP to restart the system. Upon entry to the CCP, register C is set to the drive to select after system initialization. The WarmBoot routine should read location 4 in memory, verify that is a legal drive, and pass it to the CCP in register C.

CONSOLEINSTATUS The status of the currently assigned console device is sampled and returned as 0FFH in register A if a character is ready to read and 00H in register A if no console characters are ready.

CONSOLEINPUT The next console character is read into register A. If no console character is ready, the routine waits until a character is typed before returning.

CONSOLEOUTPUT The character is sent from register C to the console output device. The character is in ASCII.

PRINTEROUTPUT (LIST) The character is sent from register C to the currently assigned listing device. The character is in ASCII.

PUNCHOUTPUT The character is sent from register C to the currently assigned punch device. The character is in ASCII.

READERINPUT The next character is read from the currently assigned reader device into register A with zero parity. An end-of-file condition is reported by returning an ASCII CTRL-Z(1AH).

HOMEDISK The head of the currently selected drive (initially drive A) is moved to the track 00 position.

SELECTDISK The drive given by register C is selected for further operations, where register C contains 0 for drive A, 1 for drive B, and so on. On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header. The DPH contains pointers to the disk parameter block which contains the information about the disk and directory. The DPH also contains pointers to CP/M work areas.

If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator.

Select Track Register BC contains the track number for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point. You can choose to seek the selected track at this time or delay the seek until the next read or write actually occurs. Register BC can take on values in the range 0-159 corresponding to valid track numbers for the ADAM disk drives and 0-255 for digital data pack subsystems.

SELECTSECTOR Register BC contains the sector number, 1 through 8, for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point.

SELECTDMA Register BC contains the DMA (Disk Memory Access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, all subsequent read operations read their data into 80H through 0FFH and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H.

READ The READ subroutine attempts to read one sector based upon the drive that has been selected, the track that has been set, and the DMA address that has been specified. It returns the following error codes in register A:

0 no errors occurred

1 nonrecoverable error condition occurred

CP/M responds only to a zero or nonzero value as the return code. If the value in register A is 0, CP/M continues since the disk operation was completed properly. Any errors that occur are handled by the BIOS after sufficient retry attempts. Unless you enter an ignore command, the program terminates with a warm boot.

WRITE Data is written from the currently selected DMA address to the currently selected drive, track, and sector. The error codes given in the READ command are returned in register A, with error recovery attempts as described above. Normally, register C should contain zero on entry. For physical deblocking, register C contains the information. See section 6.6 for details.

LISTSTATUS This can be used to check the status of your list device. The value 00 is returned in A if the list device is not ready to accept a character and 0FFH if a character can be sent to the printer.

HomeDisk (SECTOR TRANSLATE) Logical-to-physical sector translation is performed to improve the overall response of CP/M. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In general, SECTRAN receives a logical sector number relative to zero in BC and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL.

FLUSHCONSOLEINPUT reads all input from the current console device until there is no more.

The following jumps are specific to ADAM. Set-up details for these jumps are given in Appendix C.

VRAMFILL Fills a section of VRAM with the same value.

VRAMWRITE Copies memory from the Z80 space into VRAM.

VRAMREAD Copies memory from VRAM into Z80 space.

WRITEVDPREG Writes data to one of the video registers.

INITVDP Sets up the video and VRAM for textual mode using graphics 2 mode.

AUXWRITE Writes directly to an auxiliary device. Check the stack before calling.

AUXREAD Reads directly from an auxiliary device.

AUXOUTSTATUS Checks to see if the auxiliary device is ready to send a character.

AUXINSTATUS Checks to see if the auxiliary device is ready to receive a character.

INITAUXDEVICE Initializes any auxiliary devices called at warm boot.

READ1BLOCK Direct access routine. Reads a block from tape or disk into user space.

WRITE1BLOCK Writes a block to tape or disk from user space.

WILDCARDCONTINUE When the Wild Card key is pressed, this jump is made. Can be used for program interrupts.

6.4 Reserved Locations in Page Zero

Main memory page zero, between locations 00H and 0FFH, contains several segments of code and data that are used during CP/M processing. The code and data areas are given in the following table.

Table 6-3. Reserved Locations in Page Zero

Locations	Contents
000H - 0002H	Contains a jump instruction to the warm start entry location D603H. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.
0003H - 0003H	Contains the Intel standard IOBYTE.(See Section 6.3).
0004H - 0004H	Currentdefault drive number (0=A,...,3=D).
0005H - 0007H	Contains a jump instruction to the BDOS and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in Chapter 5, and LHLD 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M, allowing the CCP to be overlaid. The DDT program changes the address field to reflect the reduced memory size in debug mode.
0008H - 0027H	Interrupt locations 1 through 5 not used.
0030H - 0037H	Interrupt location 6 is reserved.
0038H - 003AH	Restart 7; contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
003BH - 003FH	Reserved.
0040H - 0041H	Address of the base of the current menus labeled as Smart Keys.
0042H - 0043H	Address of the base of the current Smart Key return strings. Labeled as Smart Key Values.

Table 6-3. Reserved Locations in Page Zero (continued)

Locations	Contents
0044H - 0045H	Address of the base of the current Smart key description or display strings. Labeled as Smart Key Descrip.
0046H - 0047H	Address of the currently displayed or active menu. Labeled as CurrKeyMenu.
004EH - 004FH	RAM card installed byte. Labeled as IsRAMDisk-There. Contains 0 or FFH. If 0, then the ram disk is available. If FFH, then ram disk is not available.
004FH - 004FH	Boot device designator.
0050H - 005BH	Reserved.
005CH - 007CH	Default File Control Block produced for a transient program by the CCP.
007DH - 007FH	Optional default random record position.
0080H - 00FFH	Default 128-byte disk buffer, also filled with the command line when a transient is loaded under the CCP.

This information is set up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

If, for example, a particular program performs only simple I/O and must begin execution at location 0, it can first be loaded into the TPA, using normal CP/M facilities, with a small memory move program that gets control when loaded. The memory move program must get control from location 0100H, which is the assumed beginning of all transient programs. The move program can then proceed to move the entire memory image down to location 0 and pass control to the starting address of the memory load.

If the BIOS is overwritten or if location 0, containing the warm start entry point, is overwritten, you must bring the CP/M system back into memory with a cold start sequence.

6.5 Sector Blocking and Deblocking

Upon each call to BIOS WRITE entry point, the CP/M BDOS includes information that allows effective sector blocking and deblocking.

On each call to WRITE, the BDOS provides the following information in register C:

0 = (normal sector write)
1 = (write to directory sector)
2 = (write to the first sector of a new data block)

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update; when the write is to other than the first sector of an unallocated block; or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written.

PART D:
APPENDICES

Appendix A
Glossary

ADAM:
ADAM lets you convert ADAM data files (SmartBASIC or SmartWRITER) to CP/M files.

ADAM BIOS calls:
See BIOS calls, ADAM specific.

address:
Number representing the location of a byte in memory. Within CP/M there are two kinds of addresses: logical and physical. A physical address refers to an absolute and unique location within ADAM's memory space. A logical address refers to the offset or displacement of a byte in relation to a base location. A standard CP/M program is loaded at address 0100H, the base value; the first instruction of a program has a physical address of 0100H and a relative address or offset of 0H.

allocation vector (ALV):
An allocation vector is maintained in the BIOS for each logged-in disk drive. A vector consists of a string of bits, one for each block on the drive. The bit corresponding to a particular block is set to one when the block has been allocated and to zero otherwise. The first two bytes of this vector are initialized with the bytes AL0 and AL1 on, thus allocating the directory blocks. CP/M Function 27 returns the allocation vector address.

AL0, AL1:
Two bytes in the disk parameter block that reserve data blocks for the directory. These two bytes are copied into the first two bytes of the allocation vector when a drive is logged in. See allocation vector.

ALV:
See allocation vector.

ambiguous filename:

Filename that contains either a ? or an * in the primary filename, filetype, or both. When you replace characters in a filename with these characters, you create an ambiguous filename and can easily reference more than one CP/M file in a single command line.

American Standard Code for Information Interchange:

See ASCII.

applications program:

Program designed to solve a specific problem. Typical applications programs are business accounting packages, word processing (editing) programs and mailing list programs.

archive attribute:

File attribute controlled by the high-order bit of the t3 byte (FCB+11) in a directory element. This attribute is set if the file has been archived.

argument:

Symbol, usually a letter, indicating a place into which you can substitute a number, letter, or name to give an appropriate meaning to the formula in question.

ASCII:

American Standard Code for Information Interchange. ASCII is a standard set of seven-bit numeric character codes used to represent characters in memory. Each character requires one byte of memory with the high-order bit usually set to zero. Characters can be numbers, letters, and symbols. An ASCII file can be intelligibly displayed on the video screen or printed on paper.

assembler:

Program that translates assembly language into the binary machine code. Assembly language is simply a set of mnemonics used to designate the instruction set of the CPU.

assembly language:

Assembly language consists of a series of names that represent the various combinations of on's and off's (electrical pulses) that instruct the computer.

back-up:

Copy of a disk or file made for safekeeping or the creation of the duplicate disk or file.

BASIC:

Beginners All-purpose Symbolic Instruction Code is a programming language.

Basic Disk Operating System:

See BDOS.

baud rate:

Measurement of data flow in which the number of signal elements per second is based on the duration of the shortest element. When each element carries one bit, the baud rate is the same number as the bits per second.

BDOS:

Basic Disk Operating System. The BDOS module of the CP/M operating system provides an interface for a user program to the operating system. This interface is a set of function calls made to the BDOS through calls to location 0005H in page zero. The user program specifies the number of the desired function in register C. User programs running under CP/M should use BDOS functions for all I/O operations. The BDOS normally resides in high memory directly below the BIOS.

binary:

Base 2 numbering system. A binary digit can have one of two values: 0 or 1. Binary numbers are used in computers because the hardware can most easily exhibit two states: off and on. Generally, a bit in memory represents one binary digit.

Basic Input/Output System:

See BIOS.

BIOS:

Basic Input/Output System. The BIOS is the only hardware-dependent module of the CP/M system. It provides the BDOS with a set of primitive I/O operations. The BIOS is an assembly language module. The BIOS interfaces the CP/M system to ADAM through a standardized jump table at the front of the BIOS routine and through a set of disk parameter tables which define the disk environment. Thus, the BIOS provides CP/M with a completely table-driven I/O system.

BIOS base:

Lowest address of the BIOS module in memory. By definition, the first entry point in the BIOS jump table.

BIOS CALLS, ADAM specific:

The calls specific to ADAM are as follows:: VRAM FI11, VRAMWRITE, VRAMREAD and INITVDP, AUXWRITE, AUXREAD, AUXOUTSTATUS, INITAUXDEVICE, READ1BLOCK, WRITE 1BLOCK, and WILDCARDCONTINUE.

bit:

Switch in memory that can be set to on (1) or off (0). Bits are grouped into bytes, eight bits to a byte. A byte is the smallest directly addressable unit. By common convention, the bits in a byte are numbered from right, 0 for the low-order bit, to left, 7 for the high-order bit. Bit values are often represented in hexadecimal notation by grouping the bits from the low-order bit in groups of four. Each group of four bits can have a value from 0 to 15 and thus can easily be represented by one hexadecimal digit.

BLM:

See block mask.

block:

Basic unit of disk space allocation. Each disk drive has a fixed block size (BLS) defined in its disk parameter block in the BIOS. A block can consist of 1K, 2K, 4K, 8K, or 16K consecutive bytes. The ADAM block is 1K. Blocks are numbered relative to zero so that each block is unique and has a byte displacement in a file equal to the block number times the block size.

block mask (BLM):

Byte value in the disk parameter block at DPB + 3. The block mask is always one less than the number of 128 byte sectors that are in one block. BLM = (2 ** BSH) − 1.

block shift (BSH):

Byte parameter in the disk or data pack parameter block at DPB + 2. Block shift and block mask (BLM) values are determined by the block size (BLS). BLM = (2 ** BSH) − 1.

blocking and deblocking algorithm:

The ADAM disk sector size is 1024 bytes. The 128-byte CP/M sectors must be blocked and deblocked by using a blocking and deblocking algorithm between the BIOS disk I/O routines and the actual disk I/O. The blocking and deblocking algorithm allows the BDOS and BIOS to function exactly as if the entire disk consisted only of 128-byte sectors as CP/M expects.

BLS:

Block size in bytes. See block.

boot:

Process of loading an operating system into memory. A boot program is a small piece of code that is automatically executed when you power-up or reset ADAM. The boot program loads the rest of the operating system into memory. This process is sometimes called a cold boot or cold start.

The boot resides on the first block of your system disk or digital data pack. When executed, the boot loads the remaining sectors of the system tracks into high memory. Finally, the boot transfers execution to the boot entry point in the BIOS jump table so that the system can initialize itself.

boot, cold:

Booting the operating system from an inoperative start, where all the computer power has been previously off or pressing RESET. Cold boot may also refer to a jump in the Bios jump table.

boot, warm:

Booting the operating system from an operative state, when the ADAM CP/M is currently running. Press ^C for a cold boot. Warm boot may also refer to a jump in the BIOS jump table. See warm start.

bootstrap:

See boot.

BSH:

See block shift.

BTREE:

General purpose file access method that has become the standard organization for indexes in large data base systems. BTREE provides near optimum performance over the full range of file operations, such as insertion, deletion, search, and search next.

buffer:

Area of memory that temporarily stores data during the transfer of information.

built-in commands:

Commands that permanently reside in memory. They respond quickly because they are not accessed from a disk on data pack.

byte:

Unit of memory, data pack, or disk storage containing eight bits. A byte can represent a binary number between 0 and 255 and is the smallest unit of memory that can be addressed directly in 8-bit CPUs.

CCP:

Console Command Processor. The CCP is a module of the CP/M operating system. It is loaded directly below the BDOS module and interprets and executes commands you type. Usually these commands are programs that the CCP loads and calls. Upon completion, a command program may return control to the CCP if it has not overwritten CCP. If it has, the program can reload the CCP into memory by a warm boot operation initiated by either a jump to zero, BDOS system reset (Function 0), or a cold boot.

Except for its location in high memory, the CCP works like any other standard CP/M program; that is, it makes only BDOS function calls for its I/O operations.

CCP base:

Lowest address of the CCP module in memory. This term sometimes refers to the base of the CP/M system in memory, because the CCP is the lowest CP/M module in high memory.

checksum vector (CSV):

Contiguous data area in the BIOS, with one byte for each directory sector to be checked, that is, CKS bytes. See CKS. A checksum vector is initialized and maintained for each logged-in drive. Each directory access by the system results in a checksum calculation that is compared with the one in the checksum vector. If there is a discrepancy, the drive is set to Read-Only status. This feature prevents the user from inadvertently switching disks without logging in the new disk. If the new disk is not logged-in, it is treated the same as the old one. Data on the new disk will be destroyed by a write.

CKS:

Number of directory records to be checked summed on directory accesses. This is a parameter in the disk parameter block located in the BIOS. If the value of CKS is zero, then no directory records are checked. CKS is also a parameter in the diskdef macro library, where it is the actual number of directory elements to be checked rather than the number of directory records.

COM:
Filetype for a CP/M command file. See command file.

command:
CP/M command line. In general, a CP/M command line has three parts: the command keyword, command tail, and a carriage return. To execute a command, enter a CP/M command line after the CP/M prompt at the console and press the RETURN key.

command file:
Executable program file of filetype COM. A command file is a machine language object module ready to be loaded and executed at the absolute address of 0100H. To execute a command file, enter its primary filename as the command keyword in a CP/M command line.

command keyword:
Name that identifies a CP/M command, usually the primary filename of a COM file or a built-in command. The command keyword precedes the command tail and the carriage return in the command line.

command syntax:
Statement that defines the correct way to enter a command. The correct structure generally includes the command keyword, the command tail, and a carriage return. A syntax line usually contains symbols that you replace with actual values when you enter the command.

command tail:
Part of a command that follows the command keyword in the command line. The command tail can include a drive specification, a filename and filetype, and options or parameters. Some commands do not require a command tail.

CON:
Mnemonic that represents the CP/M console device. For example, the CP/M command PIP CON:=example.txt. displays the file example.txt on the console device. The explanation of the STAT command tells how to assign the logical device CON: to various physical devices.
See console.

concatenate:
Name of the PIP operation that copies two or more separate files into one new file in the the specified sequence.

concurrency:
Execution of two processes or operations simultaneously.

configure:
To configure a program means to change some program parameters to reflect the equipment used.

CONFIG:
CONFIG lets you change CP/M parameters: keyboard translations, Smart Key color, text and action; character colors; cursor shape and color, background color, serial card settings and the I/O default.

CONIN (CONSOLEINPUT):
BIOS entry point to a routine that reads a character from the console device.

CONOUT (CONSOLEOUTPUT):
BIOS entry point to a routine that sends a character to the console device.

console:
Primary input/output device. The console consists of a listing device, such as a screen and a keyboard through which the user communicates with the operating system or applications program.

Console Command Processor:
See CCP.

CONST (CONSOLEINSTATUS):
BIOS entry point to a routine that returns the status of the console device.

control character:
Nonprinting character combination. CP/M interprets some control characters as simple commands such as line editing functions. To enter a control character, hold down the CONTROL key and strike the specified character key.

Control Program for Microcomputers:
See CP/M.

COPY:
The ADAM-specific CP/M copy utility lets you copy files onto the same media, to another, or from drive to drive.

CP/M:
Control Program for Microcomputers. An operating system that manages computer resources.

CPMADAM:
CPMADAM lets you convert CP/M data files created in CP/M to files that can be used in your ADAM SmartWRITER or SmartBASIC programs.

CP/M prompt:
Characters that indicate that CP/M is ready to execute your next command. The CP/M prompt consists of an upper-case letter, A, B, C, D, M followed by a > character; for example, A>. The letter designates which drive is currently logged in as the default drive. CP/M will search this drive for the command file specified, unless the command is a built-in command or prefaced by a select drive command: for example, B:STAT.

CP/M 2.2
The Control Program for Microcomputers developed by Dr. Gary Kildall. CP/M 2.2 is the current version. It has been customized for ADAM.

CSV:
See checksum vector.

cursor:
One-character symbol that can appear anywhere on the console screen. The cursor indicates the position where the next keystroke at the console will have an effect.

data file:
File containing information to be processed by a program.

deblocking:
See blocking and deblocking algorithm.

default:
Currently selected disk drive and user number. Any command that does not specify a disk drive or a user number references the default disk drive and user number. When CP/M is first invoked, the default disk drive is drive A, and the default user number is 0.

default buffer:
Default 128-byte buffer maintained at 0080H in page zero. When the CCP loads a COM file, this buffer is initialized to the command tail; that is, any characters typed after the COM file name are loaded into the buffer. The first byte at 0080H contains the length of the command tail, while the command tail itself begins at 0081H. The command tail is terminated by a byte containing a binary zero value. The I command under DDT and SID initializes this buffer in the same way as the CCP.

default FCB:
Two default FCBs are maintained by the CCP at 005CH and 006CH in page zero. The first default FCB is initialized from the first delimited field in the command tail. The second default FCB is initialized from the next field in the command tail.

delimiter:
Special characters that separate different items in a command line; for example, a colon separates the drive specification from the filename. The CCP recognizes the following characters as delimiters: . : = ; < > _, blank, and carriage return. Several CP/M commands also treat the following as delimiter characters: , [ ] * ( ) $. Do not use delimiter characters and lower-case characters in CP/M filenames.

Digital Data Pack:
The digital cassette tape for ADAM.

DIR:
Parameter in the diskdef macro library that specifies the number of directory elements on the drive.

DIR attribute:
File attribute. A file with the DIR attribute can be displayed by a DIR command. The file can be accessed from the default user number and drive only.

DIRBUF:

128-byte scratchpad area for directory operations, usually located at the end of the BIOS. DIRBUF is used by the BDOS during its directory operations. DIRBUF also refers to the two-byte address of this scratchpad buffer in the disk or data pack parameter header at DPbase + 8 bytes.

directory:

Portion of a disk or data pack that contains entries for each file on the media. In response to the DIR command, CP/M displays the filenames stored in the directory. The directory also contains the locations of the blocks allocated to the files. Each file directory element is in the form of a 32-byte FCB, although one file can have several elements, depending on its size. The maximum number of directory elements supported is specified by the drive's disk or data pack parameter block value for DRM.

directory element:

Data structure. Each file on a disk or data pack has one or more 32-byte directory elements associated with it. There are four directory elements per logical directory sector. Directory elements can also be referred to as directory FCBs.

directory entry:

File entry displayed by the DIR command. Sometimes this term refers to a physical directory element.

disk:

Magnetic media used for mass storage. Programs and data are recorded on the disk in the same way music can be recorded on cassette tape. The CP/M operating system must be initially loaded from disk or digital data pack when the computer is turned on.

diskdef macro library:

Library of code that can be used with MAC (DRI's microassembler) to create disk definition tables such as the DPB and DPH automatically.

disk drive:

Peripheral device that reads and writes information on disk. CP/M assigns a letter to each drive under its control. For example, CP/M refers to the drives in ADAM's four-drive system as A, B, C, and D.

disk parameter block (DPB):

Data structure referenced by one or more disk parameter headers. The disk parameter block defines disk characteristics in the fields listed below:

SPT is the total number of sectors per track.
BSH is the data allocation block shift factor.
BLM is the data allocation block mask.
EXM is the extent mask determined by BLS and DSM.
DSM is the maximum data block number.
DRM is the maximum number of directory entries--1.
AL0 reserves directory blocks.
AL1 reserves directory blocks.
CKS is the number of directory sectors check summed.
OFF is the number of reserved system tracks.

The address of the disk parameter block is located in the disk or data pack parameter header at DPbase +0AH. CP/M Function 31 returns the DPB address. Drives with the same characteristics can use the same disk parameter header and thus the same DPB. However, drives with different characteristics must each have their own disk parameter header and disk parameter blocks. When the BDOS calls the SELDSK entry point in the BIOS, SELDSK must return the address of the drive's disk parameter header in register HL.

disk parameter header (DPH):

Data structure that contains information about the disk or data pack drive and provides a scratchpad area for certain BDOS operations. The disk parameter header contains six bytes of scratchpad area for the BDOS and the following five 2-byte parameters:

XLT is the sector translation table address.
DIRBUF is the directory buffer address.
DPB is the disk parameter block address.
CSV is the checksum vector address.
ALV is the allocation vector address.

Given n disk drives, the disk parameter headers are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1.

DKS:

Parameter in the diskdef macro library specifying the number of data blocks on the drive.

DMA:

Direct Memory Access. DMA is a method of transferring data from the disk into memory directly. In a CP/M system, the BDOS calls the BIOS entry point READ to read a sector from the disk into the currently selected DMA address. The DMA address must be the address of a 128-byte buffer in memory, either the default buffer at 0080H in page zero, or a user-assigned buffer in the TPA. Similarly, the BDOS calls the BIOS entry point WRITE to write the record at the current DMA address to the disk.

DN:
Parameter in the diskdef macro library specifying the logical drive number.

DPB:
See disk parameter block.

DPH:
See disk parameter header.

DRM:
2-byte parameter in the disk parameter block at DPB + 7. DRM is one less than the total number of directory entries allowed for the drive. This value is related to DPB bytes AL0 and AL1, which allocates up to 16 blocks for directory entries.

DSM:
2-byte parameter of the disk parameter block at DPB + 5. DSM is the maximum data block number supported by the drive. The product BLS times (DSM+1) is the total number of bytes held by the drive. This must not exceed the capacity of the physical disk less the reserved system tracks.

editor:
Utility program that creates and modifies text files. An editor can be used for creation of documents or creation of code for computer programs. The CP/M editor is invoked through the command ED.

EOS:
ADAM's 7K Extended Operating System is referred to as EOS.

EX:
Extent number field in an FCB. See extent.

execute:
Start the processing of executable code.

executable:
Ready to be run by the computer. Executable code is a series of instructions that can be carried out by the computer. For example, the computer cannot execute names and addresses, but it can execute a program that prints all those names and addresses on mailing labels.

execute a program:
Start the processing of executable code.

EXM:
See extent mask.

extent:

16K consecutive bytes in a file. Extents are numbered from 0 to 31. One extent can contain 1, 2, 4, 8, or 16 blocks. EX is the extent number field of an FCB and is a one-byte field at FCB + 12, where FCB labels the first byte in the FCB. Depending on the block size (BLS) and the maximum data block number (DSM), an FCB can contain 1, 2, 4, 8, or 16 extents. The EX field is set to 0, but contains the current extent number during file I/O. The term FCB folding describes FCBs containing more than one extent.

extent mask (EXM):

A byte parameter in the disk parameter block located at DPB + 3. The value of EXM is determined by the block size (BLS) and whether the maximum data block number (DSM) exceeds 255. There are EXM + 1 extents per directory FCB.

FCB:

See File Control Block.

file:

Collection of characters, instructions, or data that can be referenced by a unique identifier. Files are stored on various types of media, such as disk, or magnetic tape. A CP/M file is identified by a file specification and resides on disk as a collection of from zero to 65,536 records. Each record is 128 bytes and can contain either binary or ASCII data. Binary files contain bytes of data that can vary in value from 0H to 0FFH. ASCII files contain sequences of character codes delineated by a carriage return and line-feed combination; normally byte values range from 0H to 7FH. The directory maps the file as a series of physical blocks. Although files are defined as a sequence of consecutive logical records, these records can not reside in consecutive sectors on the disk. See also block, directory, extent, record, and sector.

File Control Block (FCB):

Structure used for accessing files on disk or data pack. Contains the drive, filename, filetype, and other information describing a file to be accessed or created on the disk. A file control block consists of 36 consecutive bytes specified by the user for file I/O functions. FCB can also refer to a directory element in the directory portion of the allocated disk space. These contain the same first 32 bytes of the FCB, but lack the current record and random record number bytes.

filename:

Name assigned to a file. A filename can include a primary filename of one to eight characters; a filetype of zero to three characters. A period separates the primary filename from the filetype.

file specification:

Unique file identifier. A complete CP/M file specification includes a disk or data pack drive specification followed by a colon, a primary filename of one to eight characters, a period, and a filetype of zero to three characters. For example, b:example.txt is a complete CP/M file specification.

filetype:
Extension to a filename. A filetype can be from zero to three characters and must be separated from the primary filename by a period. A filetype can tell something about the file. Some programs require that files to be processed have specific filetypes.

formatting:
Formatting writes the hex bit value E5 throughout the disk or data pack. This information lets the disk be written to, read, and used with your CP/M operating system. CP/M 2.2 formatted disks and data packs cannot be used interchangeably with DISK MANAGER formatted disks or data packs or those that have been initialized using SmartBASIC.

FSC:
Parameter in the diskdef macro library specifying the first physical sector number. This parameter is used to determine SPT and build XLT.

hardware:
Physical components of a computer.

hexadecimal notation:
Notation for base 16 values using the decimal digits and letters A, B, C, D, E, and F to represent the 16 digits (A=10, B=11, etc.). Hexadecimal notation is often used to refer to binary numbers. A binary number can be easily expressed as a hexadecimal value by taking the bits in groups of 4, starting with the least significant bit, and expressing each group as a hexadecimal digit, 0-F. Thus the bit value 1011 becomes 0BH and 10110101 becomes 0B5H.

hex file:
ASCII-printable representation of a command, machine language, file.

hex file format:
Absolute output of ASM is a hex format file, containing a sequence of absolute records that give a load address and byte values to be stored, starting at the load address.

HOME (HomeDisk):
BIOS entry point which sets the disk head of the currently selected drive to the track zero position (used as sector translate on ADAM to copy BC to HL).

input:
Data going into the computer, usually from an operator typing at the terminal or by a program reading a disk or digital data pack.

input/output:
See I/O.

interface:

Object that allows two independent systems to communicate with each other, as an interface between hardware and software in a microcomputer.

I/O:

Abbreviation for input/output. Usually refers to input/output operations or routines handling the input and output of data in the computer system.

IOBYTE:

A one-byte field in page zero, currently at location 0003H, that can support a logical-to-physical device mapping for I/O. The IOBYTE on ADAM is easily set using the command:

STAT <logical device> = <physical device>

Or using CONFIG. The CP/M logical devices are CON:, RDR:, PUN:, and LST:; each of these can be assigned to one of four physical devices. The IOBYTE can be initialized by the BOOT entry point of the BIOS and interpreted by the BIOS I/O entry points CONST, CONIN, CONOUT, LIST, PUNCH, and READER. Depending on the setting of the IOBYTE, different I/O drivers can be selected by the BIOS. For example, setting LST:=TTY: might cause LIST output to be directed to a serial port, while setting LST:=LPT:causes LIST output to be directed to the SmartWRITER.

K:

Abbreviation for kilobyte. See kilobyte.

keyword:

See command keyword.

kilobyte (K):

1024 bytes or 0400H bytes of memory. This is a standard unit of memory. 1024 kilobytes equal one megabyte, or over one million bytes.

LIST (PRINTEROUT):

A BIOS entry point to a routine that sends a character to the list device, usually the SmartWRITER printer.

list device:

Device such as a printer onto which data can be listed or printed.

LISTST (LISTSTATUS):

BIOS entry point to a routine that returns the ready status of the list device.

loader:

Utility program that brings an absolute program image into memory ready for execution under the operating system, or a utility used to make such an image. For example, LOAD prepares an absolute COM file from the assembler hex file output that is ready to be executed under CP/M.

logged in:

Made known to the operating system, in reference to drives. A drive is logged in when it is selected by the user or an executing process. It remains selected or logged in until you change disks or data packs enter CTRL-C at the command level, or until a BDOS Function 0 is executed.

logical:

Representation of something that might or might not be the same in its actual physical form.

logical sector:

See sector.

logical-to-physical sector translation table:

See XLT.

LSC:

Diskdef macro library parameter specifying the last physical sector number.

LST:

Logical CP/M list device, usually the SmartWRITER printer. The CP/M list device is an output-only device referenced through the LIST and LISTST entry points of the BIOS. The STAT command allows assignment of LST: to one of the physical devices: TTY:, CRT:, LPT:, or UL1:, provided these devices and the IOBYTE are implemented in the LIST and LISTST entry points of your CP/M BIOS module. For example, PIP LST:=Example.txt prints the file Example.txt on the list device.

megabyte:

Over one million bytes; 1024 kilobytes. See byte and kilobyte.

Microprocessor:

Silicon chip that is the central processing unit (CPU) of the microcomputer.

multi-programming:

The capability of initiating and executing more than one program at a time. These programs, usually called processes, are time-shared, each receiving a slice of CPU time on a round-robin basis. See concurrency.

nibble:

One half of a byte, usually the high-order or low-order 4 bits in a byte.

OFF:

Two-byte parameter in the disk parameter block at DPB + 13 bytes.
This value specifies the number of reserved system tracks. The disk directory begins in the first sector of track OFF.

OFS:

Diskdef macro library parameter specifying the number of reserved system tracks. See OFF.

operating system:

Collection of programs that supervises the execution of other programs and the management of computer resources. An operating system provides orderly input and output between the computer and its peripheral devices. It enables user-written programs to execute safely. An operating system standardizes the use of computer resources for the programs running under it.

option:

One of many parameters that can be part of a command tail. Options specify additional conditions for a command's execution.

output:

Data that is sent to the console, disk, data pack, or printer.

page:

256 consecutive bytes in memory beginning on a page boundary, whose base address is a multiple of 256 (100H) bytes. In hex notation, pages always begin at an address with a least significant byte of zero.

page relocatable program:

See PRL.

page zero:

Memory region between 0000H and 0100H used to hold critical system parameters. Page zero functions primarily as an interface region between user programs and the CP/M BDOS module.

parameter:

Value in the command tail that provides additional information for the command. Technically, a parameter is a required element of a command.

peripheral devices:

Devices external to the CPU. For example, terminals, printers, and disk drives are common peripheral devices that are not part of the processor but are used in conjunction with it.

physical:

Characteristic of computer components, generally hardware, that actually exist. In programs, physical components can be represented by logical components.

primary filename:

First 8 characters of a filename. The primary filename is a unique name that helps the user identify the file contents. A primary filename contains one to eight characters and can include any letter or number and some special characters. The primary filename follows the optional drive specification and precedes the optional filetype.

PRL:

Page relocatable program. A page relocatable program is stored on disk or data pack with a PRL filetype. Page relocatable programs are easily relocated to any page boundary.

program:

Series of coded instructions that performs specific tasks when executed by a computer.

prompt:

Any characters displayed on the video screen to help the user decide what the next appropriate action is. A system prompt is a special prompt displayed by the operating system. The alphabetic character indicates the default drive. Some applications programs have their own special prompts. See CP/M prompt.

PUN:

Logical CP/M punch device. The punch device is an output-only device accessed through the PUNCH entry point of the BIOS. In ADAM, PUN defaults to the RS232 interface.

PUNCH (PUNCHOUTPUT):

BIOS entry point to a routine that sends a character to the punch device.

random access:

Disk system storage method. The sequence in which the computer reads information is not predetermined. Information can be accessed from random locations.

RDR:

Logical CP/M reader device. The reader device is an input-only device accessed through the READER entry point in the BIOS. See PUN:.

READ:

Entry point in the BIOS to a routine that reads 128 bytes from the currently selected drive, track, and sector into the current DMA address.

READER (READERINPUT):

Entry point to a routine in the BIOS that reads the next character from the currently assigned reader device.

Read-Only (R/O):

Attribute that can be assigned to a disk file or a disk drive. When assigned to a file, the Read-Only attribute allows you to read from that file but not write to it. When assigned to a drive, the Read-Only attribute allows you to read any file on the disk, but prevents you from adding a new file, erasing or changing a file, renaming a file, or writing on the disk. The STAT command can set a file or a drive to Read-Only. Every file and drive is either Read-Only or Read-Write. The default setting for drives and files is Read-Write, but an error in resetting the disk or changing media automatically sets the drive to Read-Only until the error is corrected. See also ROM.

Read-Write (R/W):

Attribute that can be assigned to a disk file or a disk drive. The Read-Write attribute allows you to read from and write to a specific Read-Write file or to any file on a disk that is in a drive set to Read-Write. A file or drive can be set to either Read-Only or Read-Write.

record:

Group of bytes in a file. A physical record consists of 128 bytes and is the basic unit of data transfer between the operating system and the application program. A logical record might vary in length and is used to represent a unit of information. Two 64-byte logical records can be stored in one 128-byte physical record. Records are grouped together to form a file.

recursive procedure:

Code that can call itself during execution.

reentrant procedure:

Code that can be called by one process while another is already executing it. Thus, reentrant code can be shared between different users. Reentrant procedures must not be self-modifying; that is, they must be pure code and not contain data. The data for reentrant procedures can be kept in a separate data area or placed on the stack.

restart (RST):

One-byte call instruction usually used during interrupt sequences and for debugger break pointing. There are eight restart locations, RST 0 through RST 7, whose addresses are given by the product of 8 times the restart number.

R/O:

See Read-Only.

ROM:

Read-Only memory. This memory can be read but not written and so is suitable for code and preinitialized data areas only.

RST:

See restart.

R/W:

See Read-Write.

sector:

In a CP/M system, a sector is always 128 consecutive bytes. A sector is the basic unit of data read and written on the disk by the BIOS. A sector can be one 128-byte record in a file or a sector of the directory. The BDOS always requests a logical sector number between 0 and (SPT − 1). This is typically translated into a physical sector by the BIOS entry point SECTRAN.

sectors per track (SPT):

A two-byte parameter in the disk parameter block at DPB + 0. The BDOS makes calls to the BIOS entry point SECTRAN with logical sector numbers ranging between 0 and (SPT − 1) in register BC.

SECTRAN (HomeDisk, Select Transaction):
Entry point to a routine in the BIOS that performs logical-to-physical sector translation for the BDOS.

SELDSK (SelectDisk):
Entry point to a routine in the BIOS that sets the currently selected drive.

SETDMA (SelectDMA):
Entry point to a routine in the BIOS that sets the currently selected DMA address. The DMA address is the address of a 128-byte buffer region in memory that is used to transfer data to and from the disk in subsequent reads and writes.

SETSEC (SelectSector):
Entry point to a routine in the BIOS that sets the currently selected sector.

SETTRK (SelectTrack):
Entry point to a routine in the BIOS that sets the currently selected track.

software:
Programs that contain machine-readable instructions, as opposed to the hardware, which are the actual physical components of a computer.

source:
Disk, data pack, or device from which information is taken.

source file:
ASCII text file usually created with an editor that is an input file to a system program, such as a language translator or text formatter.

SP:
Stack pointer. See stack.

spooling:
Process of accumulating printer output in a file while the printer is busy. The file is printed when the printer becomes free; a program is not slowed waiting for the printing process.

SPT:
See sectors per track.

stack:
Reserved area of memory where the processor saves the return address when a call instruction is received. When a return instruction is encountered, the processor restores the current address on the stack to the program counter. Data such as the contents of the registers can also be saved on the stack. The push instruction places data on the stack and the pop instruction removes it. An item is pushed onto the stack by decrementing the stack pointer (SP) by 2 and writing the item at the SP address.
In other words, the stack grows downward in memory.

syntax:
Format for entering a given command.

SYS:
See system attribute.

system attribute (SYS):
File attribute. You can give a file the system attribute by using the SYS option in the STAT command or by using the set file attributes function, BDOS Function 12. A file with the SYS attribute is not displayed in response to a DIR command. If you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number on the same drive. This feature makes commonly used programs available under any user number.

system prompt:
Symbol displayed by the operating system indicating that the system is ready to receive input. See prompt and CP/M prompt.

system tracks:
Tracks reserved on the disk or data pack for the CP/M system. The number of system tracks is specified by the parameter OFF in the disk parameter block (DPB). The system tracks for a drive always precede its data tracks. The command SYSGEN copies the CP/M system from the system tracks to memory, and vice versa.

target:
The disk, data pack, or device to which information is sent.

terminal:
See console.

TPA:
Transient Program Area. Area in memory where user programs run and store data. This area is a region of memory beginning at 0100H and extending to the base of the CP/M system in high memory. The first module of the CP/M system is the CCP, which can be overwritten by a user program. If so, the TPA is extended to the base of the CP/M BDOS module. If the CCP is overwritten, the user program must terminate with either a system reset (Function 0) call or a jump to location zero in page zero. The address of the base of the CP/M BDOS is stored in location 0006H in page zero least significant byte first.

track:
Data on the disk media is accessed by combination of track and sector numbers. Tracks form concentric rings on the disk; standard double density disks have 40 tracks. Each track consists of a fixed number of numbered sectors. Tracks are numbered from zero to one less than the number of tracks on the disk.

transient commands:
Commands that are commonly used programs (such as PIP, DDT, and STAT) that execute out of the TPA.

Transient Program Area:
See TPA.

transparent:
Any operation or function that occurs without being visible to the user.

USER:
Term used in CP/M systems to distinguish distinct regions of the directory.

user number:
Number assigned to files in the disk or data pack directory so that different users need only deal with their own files and have their own directories, even though they are all working from the same disk. In CP/M, files can be divided into 16 user groups.

utility:
Tool. Program that enables the user to perform certain operations, such as copying files, erasing files, and editing files. The utilities are created for the convenience of programmers and users.

VOLUME:
EOS data packs and disks get a VOLUME designation when they are initialized.

VRAM:
ADAM's 16K video display is generated by the T199 chip.

vector:
Location in memory. An entry point into the operating system used for making system calls or interrupt handling.

warm start:
Program termination by a jump to the warm start vector at location 0000H, a system reset (BDOS Function 0), or a CTRL-C typed at the keyboard. A warm start reinitializes the disk subsystem and returns control to the CP/M operating system at the CCP level. The warm start vector is simply a jump to the WarmBoot entry point in the BIOS.

WarmBoot:
Entry point to a routine in the BIOS used when a warm start occurs. A warm start is performed when a user program branches to location 0000H, when the CPU is reset, or when you type CTRL-C. The CCP and BDOS are reloaded from the system tracks of drive A.

wildcard characters:
Special characters that match certain specified items. In CP/M there are two wildcard characters: ? and *. The ? can be substituted for any single character in a filename, and the * can be substituted for the primary filename, the filetype, or both. By placing wildcard characters in filenames, you create an ambiguous filename and can quickly reference one or more files.

word:
16-bit or two-byte value, such as an address value.

WRITE:

Entry point to a routine in the BIOS that writes the record at the currently selected DMA address to the currently selected drive, track, and sector.

write-protect:

Information cannot be written to a disk or data pack that is write-protected.

XLT:

Logical-to-physical sector translation table located in the BIOS.

SECTRAN uses XLT to perform logical-to-physical sector number translation. XLT also refers to the two-byte address in the disk parameter header at DPBASE + 0. If this parameter is zero, no sector translation takes place. Otherwise this parameter is the address of the translation table.

ZERO PAGE:

See page zero.

End of Appendix A

Appendix B
Keyboard and Screen Codes

Part I
Source Code Listing for the ADAM keyboard, Organized by Key.

Name of the Key	Standard Code	SHIFT Code	CONTROL Code	HOME Code
WILD CARD	090H	098H	090H	
Smart Key I	081H	089H	081H	
Smart Key II	082H	08AH	082H	
Smart Key III	083H	08BH	083H	
Smart Key IV	084H	08CH	084H	
Smart Key V	085H	08DH	085H	
Smart Key VI	086H	08EH	086H	
UNDO	091H	099H	091H	
! 1	031H	021H	031H	
@ 2	032H	040H	000H	
# 3	033H	023H	033H	
$ 4	034H	024H	034H	
% 5	035H	025H	035H	
_ 6	036H	05FH	01FH	
& 7	037H	026H	037H	
* 8	038H	02AH	038H	
TAB	0D9H	0B9H	009H	
Q	071H	051H	011H	
W	077H	057H	017H	
E	065H	045H	005H	
R	072H	052H	012H	
T	074H	054H	014H	
Y	079H	059H	019H	
U	075H	055H	015H	
A	061H	041H	001H	
S	073H	053H	013H	
D	064H	044H	004H	
F	066H	046H	006H	
G	067H	047H	007H	
H	068H	048H	008H	

J	06AH	04AH	00AH		
K	06BH	04BH	00BH		
Z	07AH	05AH	01AH		
X	078H	058H	018H		
C	063H	043H	003H		
V	076H	056H	016H		
B	062H	042H	002H		
N	06EH	04EH	00EH		
M	06DH	04DH	00DH		
< ,	02CH	03CH	02CH		
(9	039H	028H	039H		
) 0	030H	029H	030H		
` -	02DH	060H	02DH		
= +	02BH	03DH	02BH		
~ ^	05EH	07EH	01EH		
1	06CH	04CH	00CH		
: ;	03BH	03AH	03BH		
" '	027H	022H	027H		
I	069H	049H	009H		
O	06FH	04FH	00FH		
P	070H	050H	010H		
{ [	05BH	07BH	01BH		
}]	05DH	07DH	01DH		
> .	02EH	03EH	02EH		
? /	02FH	03FH	02FH		
RETURN	00DH	00DH	00DH		
| \	05CH	07CH	01CH		
WP/ESCAPE	01BH	01BH	01BH		
	0FFH	0FFH	0FFH		
	0FFH	0FFH	0FFH		
SPACE	020H	020H	020H		
MOVE/COPY	092H	09AH	092H		
STORE/GET	093H	09BH	093H		
CLEAR	096H	09EH	096H		
up arrow	0A0H	0A8H	0A4H	0ACH	
right arrow	0A1H	0A9H	0A5H	0ADH	
down arrow	0A2H	0AAH	0A6H	0AEH	
left arrow	0A3H	0ABH	0A7H	0AFH	
BACKSPACE	008H	0B8H	008H		
INSERT	094H	09CH	094H		
PRINT	095H	09DH	095H		
DELETE	097H	09FH	07FH		
HOME	080H	080H	080H		

Four arrow key combinations produce unique codes, allowing diagonal graphics movements:

Up arrow + right arrow:	0A8H
Down arrow + right arrow	0A9H
Down arrow + left arrow:	0AAH
Up arrow + left arrow:	0ABH

LOCK acts as a "Shift Lock Key." When it is depressed, all keys (including numbers and Smartkeys) will behave as if shifted.

Part II
Keycodes generated by the ADAM keyboard, Organized by Value.

DEC	HEX	ASCII	CV KEY(S)	REPEAT	COMMENTS
0	00	NUL	cntrl 2	N	Null (subs. for cntrl @)
1	01	SOH	cntrl A	N	Start of Heading
2	02	STX	cntrl B	N	Start of Text
3	03	ETX	cntrl C	N	End of Text
4	04	EOT	cntrl D	N	End of Transmission
5	05	ENQ	cntrl E	N	Enquiry
6	06	ACK	cntrl F	N	Acknowledge
7	07	BEL	cntrl G	N	Bell
8	08	BS	cntrl H or BACKSPACE	Y	Backspace (see NOTE 5)
9	09	HT	cntrl I or TAB	N	Horizontal Tab (see NOTE 6)
10	0A	LF	cntrl J	N	Line Feed
11	0B	VT	cntrl K	N	Vertical Tab
12	0C	FF	cntrl L	N	Form Feed
13	0D	CR	cntrl M or RETURN	N	Carriage Return
14	0E	SO	cntrl N	N	Shift Out
15	0F	SI	cntrl O	N	Shift In
16	10	DLE	cntrl P	N	Data Link Escape
17	11	DC1	cntrl Q	N	Device Control 1
18	12	DC2	cntrl R	N	Device Control 2
19	13	DC3	cntrl S	N	Device Control 3
20	14	DC4	cntrl T	N	Device Control 4
21	15	NAK	cntrl U	N	Negative Acknowledge
22	16	SYN	cntrl V	N	Synchronous Idle
23	17	ETB	cntrl W	N	End of Transmission Block
24	18	CAN	cntrl X	N	Cancel
25	19	EM	cntrl Y	N	End of Medium
26	1A	SUB	cntrl Z	N	Substitute
27	1B	ESC	cntrl [or WP/ESCAPE	N	Escape
28	1C	FS	cntrl \	N	File Separator
29	1D	GS	cntrl]	N	Group Separator
30	1E	RS	cntrl ^	N	Record Separator
31	1F	US	cntrl 6	N	Unit Separator- (subs. for cntrl _)
32	20	SP	space bar	Y	Space
33	21	!	shift 1	Y	Exclamation Point
34	22	"	shift '	Y	Quotation Mark (double quote)
35	23	#	shift 3	Y	Number Sign
36	24	$	shift 4	Y	Dollar Sign
37	25	%	shift 5	Y	Percent
38	26	&	shift 7	Y	Ampersand
39	27	'	'	Y	Apostrophe (single quote)
40	28	(	shift 9	Y	Opening Parenthesis
41	29	)	shift 0	Y	Closing Parenthesis
42	2A	*	shift 8	Y	Asterisk
43	2B	+	+	Y	Plus
44	2C	,	,	Y	Comma
45	2D	-	-	Y	Hyphen (Minus)
46	2E	.	.	Y	Period (Decimal Point)
47	2F	/	/	Y	Slant
48	30	0	0	Y	
49	31	1	1	Y	
50	32	2	2	Y	
51	33	3	3	Y	
52	34	4	4	Y	
53	35	5	5	Y	
54	36	6	6	Y	
55	37	7	7	Y	

DEC	HEX	ASCII	CV KEY(S)	REPEAT	COMMENTS
56	38	8	8	Y	
57	39	9	9	Y	
58	3A	:	shift ;	Y	Colon
59	3B	;	;	Y	Semicolon
60	3C	<	shift ,	Y	Less Than
61	3D	=	shift +	Y	Equals
62	3E	>	shift .	Y	Greater Than
63	3F	?	shift /	Y	Question Mark
64	40	@	shift 2	Y	Commercial At
65	41	A	shift A	Y	upper case
66	42	B	shift B	Y	" "
67	43	C	shift C	Y	" "
68	44	D	shift D	Y	" "
69	45	E	shift E	Y	" "
70	46	F	shift F	Y	" "
71	47	G	shift G	Y	" "
72	48	H	shift H	Y	" "
73	49	I	shift I	Y	" "
74	4A	J	shift J	Y	" "
75	4B	K	shift K	Y	" "
76	4C	L	shift L	Y	" "
77	4D	M	shift M	Y	" "
78	4E	N	shift N	Y	" "
79	4F	O	shift O	Y	" "
80	50	P	shift P	Y	" "
81	51	Q	shift Q	Y	" "
82	52	R	shift R	Y	" "
83	53	S	shift S	Y	" "
84	54	T	shift T	Y	" "
85	55	U	shift U	Y	" "
86	56	V	shift V	Y	" "
87	57	W	shift W	Y	" "
88	58	X	shift X	Y	" "
89	59	Y	shift Y	Y	" "
90	5A	Z	shift Z	Y	upper case
91	5B	[	[	Y	Opening Bracket
92	5C	\	\	Y	Reverse Slant
93	5D	]	]	Y	Closing Bracket
94	5E	^	^	Y	Circumflex
95	5F	_	shift 6	Y	Underline
96	60	`	shift -	Y	Grave Accent
97	61	a	A	Y	lower case
98	62	b	B	Y	" "
99	63	c	C	Y	" "
100	64	d	D	Y	" "
101	65	e	E	Y	" "
102	66	f	F	Y	" "
103	67	g	G	Y	" "
104	68	h	H	Y	" "
105	69	i	I	Y	" "
106	6A	j	J	Y	" "
107	6B	k	K	Y	" "
108	6C	l	L	Y	" "
109	6D	m	M	Y	" "
110	6E	n	N	Y	" "
111	6F	o	O	Y	" "
112	70	p	P	Y	" "
113	71	q	Q	Y	" "
114	72	r	R	Y	" "
115	73	s	S	Y	" "
116	74	t	T	Y	" "
117	75	u	U	Y	" "
118	76	v	V	Y	" "
119	77	w	W	Y	" "

DEC	HEX	ASCII	CV KEY(S)	REPEAT	COMMENTS
120	78	x	X	Y	" "
121	79	y	Y	Y	" "
122	7A	z	Z	Y	lower case
123	7B	{	shift [	Y	Opening Brace
124	7C	|	shift \	Y	Vertical Line
125	7D	}	shift]	Y	Closing Brace
126	7E	~	shift ^	Y	Tilde
127	7F	DEL	cntrl DELETE	Y	Delete (subs. for DEL)

** end of ASCII codes **

** start of COLECO special codes (i.e. non-ASCII) defined by group **

A) SMARTKEY GROUP

DEC	HEX	CV KEY(S)	REPEAT	COMMENTS
128	80	HOME	N	group exception
129	81	I	N	smartkey 1
130	82	II	N	smartkey 2
131	83	III	N	smartkey 3
132	84	IV	N	smartkey 4
133	85	V	N	smartkey 5
134	86	VI	N	smartkey 6
135	87			unused code
136	88			unused code
137	89	shift I	N	
138	8A	shift II	N	
139	8B	shift III	N	
140	8C	shift IV	N	
141	8D	shift V	N	
142	8E	shift VI	N	
143	8F			unused code

B) WORD PROCESSOR "hard key" GROUP

DEC	HEX	CV KEY(S)		COMMENTS
144	90	WILD CARD	N	
145	91	UNDO	N	
146	92	MOVE	N	
147	93	STORE	N	
148	94	INSERT	N	
149	95	PRINT	N	
150	96	CLEAR	N	
151	97	DELETE	N	independent of DEL (ASCII)
152	98	shift WILD CARD	N	
153	99	shift UNDO	N	
154	9A	shift MOVE	N	(COPY)
155	9B	shift STORE	N	(FETCH)
156	9C	shift INSERT	N	
157	9D	shift PRINT	N	
158	9E	shift CLEAR	N	
159	9F	shift DELETE	N	independent of DEL (ASCII)

C) CURSOR CONTROL GROUP

DEC	HEX	CV KEY(S)	REPEAT	COMMENTS
160	A0	up arrow	Y	north
161	A1	right arrow	Y	east
162	A2	down arrow	Y	south
163	A3	left arrow	Y	west
164	A4	cntrl up arrow	Y	
165	A5	cntrl right arrow	Y	
166	A6	cntrl down arrow	Y	
167	A7	cntrl left arrow	Y	

168	A8	up arrow + right arrow	Y	northeast - (see Note 7)
169	A9	right arrow + down arrow	Y	southeast - "
170	AA	down arrow + left arrow	Y	southwest - "
171	AB	left arrow + up arrow	Y	northwest - (see Note 7)
172	AC	HOME + up arrow	N	(see Note 7)
173	AD	HOME + right arrow	N	"
174	AE	HOME + down arrow	N	"
175	AF	HOME + left arrow	N	(see Note 7)

D) GENERAL KEY GROUP

DEC	HEX	CV KEY(S)	REPEAT	COMMENTS
176	B0			unused code
177	B1			" "
178	B2			" "
179	B3			" "
180	B4			" "
181	B5			" "
182	B6			unused code
183	B7			
184	B8	shift BACKSPACE	Y	(see NOTE 5)
185	B9	shift TAB	N	(see NOTE 6)
186	BA			unused code
187	BB			" "
188	BC			" "
189	BD			" "
190	BE			unused code
191	BF			

** end of COLECO special codes defined by group **

** NOTES **

NOTE 1: The lock key will act as a "shift lock function," i.e. when in the active state (on), all keys will behave as if they were in their shifted key versions. If the lock key is inactive (off), all key presses are treated as unshifted key presses. In other words, the ADAM lock key functions similar to a typwriter's shiftlock.

NOTE 2: The remaining codes 0C0H thru 0EFH are unused codes.

NOTE 3: Codes 0F0H thru 0FFH are reserved for internal use by the keyboard software.

NOTE 4: The following keys have no code assigned to them: CNTRL, SHIFT, LOCK. These keys are used internally by the keyboard software to calculate the CNTRL, SHIFT and LOCK values for all other keys. No serial transmission will occur when these keys are pressed.

NOTE 5: Codes 008H and 0B8H are provided for purposes of non-destructive and destructive BACKSPACE. The interpretation of these codes is application dependent. The following convention is recommended:

008H = BACKSPACE (as defined by ASCII)
0B8H = destructive BACKSPACE

NOTE 6: Codes 009H and 0B9H are provided for purposes of right TAB and left TAB. The interpretation of these codes is application dependent. The following convention is recommended:

009H = right TAB (as defined by ASCII)
0B9H = left TAB

NOTE 7: Sequence independent, time critical. To receive this value from the keyboard, the second key must be pressed within about 1/2 second.

Part III
Escape Sequences and Control Characters.

Escape Sequences

Set character set color (00-7F):	ESC b	TI9918Color Byte*
Set background color:	ESC c	TI9918 ColorByte*
Save Cursor Position:	ESC j	
Restore Cursor Position:	ESC k	
Turn Cursor Display Off:	ESC m	
Turn Cursor Display On:	ESC n	
Erase From Cursor To Beginning Of Line	ESC o	
Start hi-bit character set:	ESC p	
Start regular character set:	ESC q	
Move Cursor Up One Line:	ESC A	
Move Cursor Down One Line:	ESC B	
Move Cursor Right One Character:	ESC C	
Move Cursor Left One Character:	ESC D	
Erase Screen & Home Cursor:	ESC E	
Home Cursor:	ESC H	
Reverse Linefeed:	ESC I	
Erase From Cursor To End Of Screen:	ESC J	
Erase From Cursor To End Of Line:	ESC K	
Position Cursor (GOTOXY):	ESC Y	32+Y 32+X
Set Smart Key Strings:	ESC :	(return string) 00 (display string) 00 (display type)
Set hi-bit character set color (80-FF):	ESC i	TI9918ColorByte*
Force On System Smart Keys:	ESC l	

Only the 4 LSBits of the background ColorByte are significant.

Control Characters

Hex	Dec	
00	00	Nul action
01	01	Tab left to an 8 character boundary (non-destructive).
02	02	Erase line and return cursor.
03	03	Move cursor down.
04	04	Move cursor left (non-destructive).
05	05	Move cursor right (non-destructive).
06	06	Move cursor up.
07	07	Beep noise.
08	08	Destructive backspace.
09	09	Tab right to an 8 character boundary (non-destructive).
0A	10	Linefeed.
0B	11	Erase from cursor to end of screen (inclusive).
0C	12	Erase screen and Home cursor.
0D	13	Return cursor to start of line.
0E	14	Home cursor.
0F	15	Erase from cursor to end of line (inclusive).
10	16	Enable 80 column horizontal scroll.
11	17	Disable 80 column horizontal scroll
12	18	Trap Smart Keys on console input.

(continued)

Control Characters (continued)

Hex	Dec	
13	19	Do not trap cursor keys on console input.
14	20	Scroll virtual screen horizontally 1 character to the left.
15	21	Scroll virtual screen horizontally 1 character to the right.
16	22	Turn off Smart Key display.
17	23	Turn on Smart Key display.
18	24	Toggle Smart Key display on/off.
19	25	Turn off Smart Key display and do not trap Smart Keys on console input.
1A	26	Turn on Smart Key display and trap Smart Keys on console input.
1B	27	Escape: start escape sequence.
1C	28	Start using alternate color character set (flipping hi-bit).
1D	29	Stop using alternate color character set.
1E	30	Reverse linefeed.
1F	31	Erase from cursor to beginning of line (inclusive).

Appendix C
ADAM BIOS Details

ADAM BIOS Details

The extended BIOS jumps provided in ADAM CP/M give you greater access to ADAM's features. The Jump vectors are listed in Part C, Section 6.3. Further assigned addresses for these ADAM specific jumps are listed here.

`VramFill` fills a section of VRAM with the same value.
Entry Parameters:
DE = VRAM starting address.
BC = Number of bytes to be "filled."
A = Value to be written into VRAM.
Exit: None.

`VramWrite` copies memory from the Z80 space into VRAM.
Entry Parameters:
HL = Base address of bytes to be written.
DE = Starting address in VRAM.
BC = Number of bytes to be written.
Exit: None.

`VramRead` copies from VRAM into Z80 space.
Entry Parameters:
BC = Number of bytes to be read from VRAM.
DE = Destination Z80 Buffer address.
HL = Source VRAM address.
Exit: None.

`WriteVDPReg` writes data to one of the video registers.
Entry Parameters:
D = VDP Register number (0 to 7)
E = Data for that register
Exit: None.

InitVDP sets up the video and VRAM for text mode using Graphics 2 mode.

Entry Parameters: None.
Exit: None.

AuxWrite writes directly to an auxiliary device. Check the stack before calling.

Entry Parameters:
C = character to be sent.
Exit: None.

AuxRead reads directly from an auxiliary device.

Entry Parameters:
A = character returned.
Exit: None.

AuxOutStatus checks to see if the auxiliary device is ready to send a character.

Entry Parameters:
None.
Exit Parameter:
A = FFH
Returns: Ready to send.

AuxInStatus checks to see if the auxiliary device is ready to receive a character.

Entry Parameter:
None.
Exit Parameter:
A = FFH
Returns: Character has been received.

InitAuxDevice initializes any auxiliary devices called at warm boot.

ReadlBlock is a direct access routine. Reads a block from the tape or disk into user space.

Entry Parameters:
A = Device ID
DE = Buffer address location of the data after the read
BC = Block number
Exit: Carry Set if no error
Clear if error

WritelBlock writes a block to tape or disk from user space.

Entry Parameters:
A = Device ID
DE = Buffer address location of data to be written
BC = Block number.
Exit: Carry Set if no error
Clear if error

WildCardContinue is used for program interrupts. Press WildCard key to make this jump.

Appendix D
ADAM Filter Program

```
                ;
                ;   BY STEPHEN MUNNINGS
                ;      WATERLOO, ONTARIO
                ;      CANADA
                ;
                ;   THIS PROGRAM TAKES AN INPUT FILE, AND CREATES
                ;     AN OUTPUT FILE THAT IS A FIXED-UP VERSION.
                ;   IT ASSUMES THAT THE INPUT FILE IS A CONVERTED
                ;     "ADAM'ED" FILE FROM THE ADAM SYSTEM.
                ;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                ;   EQUATES
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                ;
                ;   BDOS EQUATES
                ;
0005 =          BDOS    EQU     5
0009 =          PRINTS  EQU     9       ;PRINT STRING ENTRY CODE
000A =          READST  EQU     10      ;READ CONSOLE STRING
000F =          OPEN    EQU     15      ;OPEN FILE CODE
0010 =          CLOSE   EQU     16      ;CLOSE FILE CODE
0014 =          READS   EQU     20      ;READ FILE SEQUENTIAL RECORD CODE
0015 =          WRITES  EQU     21      ;WRITE SEQUENTIAL RECORD CODE
0016 =          CREATE  EQU     22      ;CREATE A FILE (OUTPUT)
0017 =          RENAM   EQU     23      ;RENAME A FILE CODE
001A =          SETDMA  EQU     26      ;SET THE DMA ADDRESS CODE
0080 =          BUFSIZ  EQU     128     ;SIZE OF THE RECORD BUFFERS
                ;
                ;   CHARACTER EQUATES
                ;
000D =          CR      EQU     0DH     ;<CR>
000A =          LF      EQU     0AH     ;<LF>
0013 =          SOU     EQU     13H     ;WORD PROCESSING START OF UNDERLINE
0014 =          EOU     EQU     14H     ;WORD PROCESSOR END OF UNDERLINE
001A =          EOF     EQU     1AH     ;END OF FILE CHARACTER
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                ;   PROGRAM CODE
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
0100                    ORG     100H
                ;
0100 210000     START   LXI     H,0
0103 39                 DAD     SP
0104 226F04             SHLD    STORST  ;SAVE OLD STACK POINTER
0107 31E506     RESTART LXI     SP,STACKEND
010A 113E03             LXI     D,HDMSG ;PRINT OUT HEADING MESSAGE
010D CD1702             CALL    PSTRING
0110 AF                 XRA     A       ;RESET THE DISK BUFFER COUNTERS
0111 32E304             STA     GETBC
0114 32E404             STA     PUTBC
0117 118203     NEWMSG  LXI     D,INPROMPT      ;PROMPT FOR INPUT FILE NAME
011A CD1C02             CALL    PROMPTR
011D CA1701             JZ      NEWMSG  ;IF NO CHARS - REPROMPT
```

```
0120 0600               MVI     B,0
0122 E5                 PUSH    H       ;SAVE OLD MEMORY ADDRESS
0123 09                 DAD     B       ;SET A 00 STRING TERMINATOR
0124 3600               MVI     M,0
0126 E1                 POP     H       ;RESTORE OLD ADDRESSS
0127 119B04             LXI     D,FCB1
012A CD3F02             CALL    OPENF   ;OPEN THIS FILE FOR INPUT
012D D23901             JNC     GETYPE  ;IF IT OPENED O.K., THEN GET TYPE
0130 115E03             LXI     D,ERMSGI ;PUT OUT ERROR MESSAGE
0133 CD1702             CALL    PSTRING
0136 C31701             JMP     NEWMSG  ;AND PROMPT FOR INPUT FILE AGAIN
0139 11B503     GETYPE  LXI     D,TYPROMP       ;PROMPT FOR CONVERSION FILE
TYPE
013C CD1C02             CALL    PROMPTR
013F CA3901             JZ      GETYPE  ;IF NO RESPONSE - REPROMPT.
0142 E6DF               ANI     0DFH    ;UPPERCASE THE FIRST LETTER
0144 FE41               CPI     'A'
0146 CA4F01             JZ      ATYPE   ;IF A - THEN SET SWITCH
0149 FE48               CPI     'H'
014B C23901             JNZ     GETYPE  ;IF NOT H THEN REPROMPT
014E 3E00               MVI     A,$-$   ;SET NON-ZERO INTO A,
014F                    ORG     $-1     ;USE XRA CODE AS THE VALUE
014F AF         ATYPE   XRA     A       ;ZERO A FOR A TYPE CONVERSION
0150 B7                 ORA     A       ;SET Z OR NZ FLAG
0151 F5                 PUSH    PSW     ;AND SAVE IT FOR LATER
0152 119B03     GETOUT  LXI     D,OUTPROM
0155 CD1C02             CALL    PROMPTR ;PROMPT FOR OUTPUT FILE NAME
0158 CA5201             JZ      GETOUT  ;RE-PROMPT IF NO ANSWER
015B 11BF04             LXI     D,FCB2
015E CD3F02             CALL    OPENF   ;SEE IF FILE IS THERE
0161 DA8F01             JC      OCREATE ;NO - CREATE IT.  YES - RENAME IT TO
TYPE
0164 11CF04             LXI     D,FCB2+16
0167 21BF04             LXI     H,FCB2
016A 011000             LXI     B,16
016D CD0E02             CALL    MOVE    ;MOVE NAME TO SECOND PART OF FCB
0170 11D804             LXI     D,FCB2+16+9
0173 3E24               MVI     A,'$'
0175 0603               MVI     B,3
0177 CD5502             CALL    TFILL   ;SET THE NEW FIELD TO $$$
017A 0E17               MVI     C,RENAM
017C 11BF04             LXI     D,FCB2
017F CD0500             CALL    BDOS
0182 3C                 INR     A       ;SEE IF RENAME WORKED
0183 C28F01             JNZ     OCREATE ;YES - CREATE THE NEW FILENAME
0186 110304     OERR    LXI     D,OERRMS
0189 CD1702             CALL    PSTRING ;TYPE ERROR MESSAGE
018C C35201             JMP     GETOUT  ;TRY TO GET A GOOD OUTPUT FILE
018F 11CF04     OCREATE LXI     D,FCB2+16
0192 CD7E02             CALL    ZFILL
0195 11BF04             LXI     D,FCB2
0198 0E16               MVI     C,CREATE
019A CD0500             CALL    BDOS    ;CREATE THE OUTPUT FILE
019D 3C                 INR     A
019E CA8601             JZ      OERR    ;IF CREATE FAILED - PUT MESSAGE
01A1 F1                 POP     PSW     ;RESTORE A OR H TYPE STATUS
01A2 C43302             CNZ     DUMPAGE ;IF H TYPE - DISCARD FIRST 256 BYTES
01A5 AF                 XRA     A       ;SET THE <CR> FLAG TO NO <CR>
01A6 3D                 DCR     A
01A7 F5                 PUSH    PSW
01A8 CDB002     FILT    CALL    GETC    ;GET A CHARACTER
01AB D2B001             JNC     NPEND   ;SKIP NEXT IF NOT PHYSICAL EOF
01AE 3E1A               MVI     A,EOF   ;INDICATE E.O.F. IN CHARACTER
01B0 4F         NPEND   MOV     C,A     ;SAVE THE CHARACTER IN C
01B1 FE13               CPI     SOU
01B3 CAA801             JZ      FILT    ;IF CHARACTER IS SOU - IGNORE IT
01B6 FE14               CPI     EOU
01B8 CAA801             JZ      FILT    ;ALSO IGNORE EOU
01BB FE0A               CPI     LF
01BD C2C401             JNZ     DOTEST  ;IF <LF> IGNORE <CR> SWITCH
01C0 F1                 POP     PSW     ;FIX STACK
01C1 C3CA01             JMP     CRTEST  ;GO TO RIGHT RESUMPTION SPOT
01C4 F1         DOTEST  POP     PSW     ;RESTORE <CR> STATUS
01C5 3E0A               MVI     A,LF
01C7 CCE802             CZ      PUTC    ;IF <CR> SET - OUTPUT <LF>
01CA 79         CRTEST  MOV     A,C
```

```
01CB FE0D               CPI     CR      ;WAS THE CHARACTER A <CR>
01CD F5                 PUSH    PSW     ;SAVE THE STATUS OF THE TEST
01CE CDE802             CALL    PUTC    ;SEND OUT CHARACTER
01D1 79                 MOV     A,C     ;RESTORE THE CHARACTER AGAIN
01D2 FE1A               CPI     EOF     ;SEE IF END OF FILE CHARACTER
01D4 C2A801             JNZ     FILT    ;NO - CONTINUE MOVING CHARACTERS
01D7 11BF04             LXI     D,FCB2  ;NOW FILL UP THE BUFFER WITH EOF'S
AND WRITE
                ;                       THE LAST BUFFER
01DA 0601               MVI     B,1
01DC CDED02             CALL    RPUTC
01DF 119B04     CLOSEI  LXI     D,FCB1  ;TRY TO CLOSE THE INPUT FILE
01E2 0E10               MVI     C,CLOSE ;BUT DON'T WORRY ABOUT IT IF IT
01E4 CD0500             CALL    BDOS    ;DOES NOT CLOSE PROPERLY
01E7 11BF04             LXI     D,FCB2  ;NOW TRY TO CLOSE THE OUTPUT FILE
01EA 0E10               MVI     C,CLOSE
01EC CD0500             CALL    BDOS
01EF 3C                 INR     A
01F0 C2F901             JNZ     CLOSOK  ;IF IT WORKED - SKIP ERROR MESSAGE
01F3 113604             LXI     D,CLOSERO
01F6 CD1702             CALL    PSTRING ;PUT OUT THE ERROR MESSAGE
01F9 11D203     CLOSOK  LXI     D,REPEAT ;ASK IF USER WANTS TO COPY AGAIN
01FC CD1C02             CALL    PROMPTR
01FF CAF901             JZ      CLOSOK  ;ASK AGAIN IF NO RESPONSE
0202 E6DF               ANI     0DFH    ;UPPERCASE THE FIRST CHARACTER
0204 FE59               CPI     'Y'     ;SEE IF AFFIRMATIVE
0206 CA0701             JZ      RESTART ;YES - START OVER
0209 2A6F04             LHLD    STORST  ;RESTORE THE ORIGINAL STACK POINTER
020C F9                 SPHL
020D C9                 RET     ;RETURN CONTROL TO THE CCP
                ;
                ;   SUBROUTINE TO SIMULATE THE Z80 LDIR INSTRUCTION
                ;
020E 7E         MOVE    MOV     A,M     ;SIMULATE THE Z80 LDIR INSTRUCTION
020F 12                 STAX    D
0210 13                 INX     D
0211 23                 INX     H
0212 05                 DCR     B
0213 C20E02             JNZ     MOVE
0216 C9                 RET
                ;   SUBROUTINE TO CALL BDOS TO PRINT A STRING
                ;
0217 0E09       PSTRING MVI     C,PRINTS ;SUBROUTINE TO PRINT A STRING
0219 C30500             JMP     BDOS
                ;
                ;   SUBROUTINE TO PROMPT HTE USER
                ;
021C CD1702     PROMPTR CALL    PSTRING  ;PRINT THE PROMPT STRING
021F 3E26               MVI     A,BUFFS  ;LOAD STRING BUFFER SIZE
0221 117104             LXI     D,BUFFR  ;LOAD BUFFER ADDRESS
0224 12                 STAX    D        ;SET THE MAX BUFFER SIZE
0225 0E0A               MVI     C,READST
0227 CD0500             CALL    BDOS     ;LET BDOS READ THE CONSOLE
022A 217204             LXI     H,BUFFR+1
022D 4E                 MOV     C,M      ;GET CHARACTER COUNT
022E 23                 INX     H
022F 7E                 MOV     A,M      ;GET THE FIRST CHARACTER
0230 0D                 DCR     C
0231 0C                 INR     C        ;SET THE Z COND BASED ON C
0232 C9                 RET
                ;
                ;   SUBROUTINE TO IGNORE 256 CHARACTERS
                ;
0233 0600       DUMPAGE MVI     B,0
0235 C5         ROO     PUSH    B
0236 CDB002             CALL    GETC
0239 C1                 POP     B
023A 05                 DCR     B
023B C23502             JNZ     ROO
023E C9                 RET
```

```
                        ;
                        ;    SUBROUTINE TO OPEN A FILE
                        ;
023F D5         OPENF   PUSH    D           ;SAVE THE FCB ADDRESS
0240 CD5C02             CALL    BFCB        ;BUILD THE FILE NAME IN THE FCB
0243 D1                 POP     D           ;RESTORE FCB ADDRESS
0244 0E0F               MVI     C,OPEN
0246 CD0500             CALL    BDOS        ;TRY TO OPEN THE FILE
0249 17                 RAL                 ;SET THE CARRY FLAG IF ERROR IN OPEN
024A C9                 RET
                        ;
                        ;    SUBROUTINE TO EDIT THE FILENAME STRING INTO THE FCB
                        ;
024B 3E3F       QFTOK   MVI     A,'?'       ;QUESTION MARK FILL ENTRY
024D CD5502             CALL    TFILL
0250 C3A602             JMP     TSKIP
0253 3E20       SFTOK   MVI     A,' '       ;SPACE FILL A PART OF THE FCB
                        ;
0255 12         TFILL   STAX    D           ;FILL A PART OF THE FCB
0256 13                 INX     D
0257 05                 DCR     B
0258 C25502             JNZ     TFILL
025B C9                 RET
                        ;
025C 23         BFCB    INX     H
025D 7E                 MOV     A,M
025E 2B                 DCX     H           ;SEE IF A DISK ID IS HERE
025F FE3A               CPI     ':'         ;BY TESTING CHAR 2 FOR A :
0261 C26A02             JNZ     NODRIVE
0264 7E                 MOV     A,M
0265 E61F               ANI     1FH         ;(A->1 B->2 ETC.)
0267 23                 INX     H
0268 23                 INX     H
0269 0E00               MVI     C,$-$       ;USED TO SKIP NEXT INSTRUCTION
026A                    ORG     $-1
026A AF         NODRIVE XRA     A           ;ZERO THE DRIVE NUMBER (DEFAULT)
026B 12                 STAX    D           ;SAVE THE DRIVE NUMBER
026C 13                 INX     D
026D 0608               MVI     B,8         ;FILENAME IS 8 CHARACTERS LONG
026F CD8402             CALL    GETOKEN     ;GET THE TOKEN
0272 7E                 MOV     A,M
0273 FE2E               CPI     '.'         ;WAS IT TERMINATED BY .
0275 C27902             JNZ     NOTYPE      ;NO - ASSUME NO TYPE FIELD
0278 23                 INX     H           ;SKIP THE PERIOD
0279 0603       NOTYPE  MVI     B,3
027B CD8402             CALL    GETOKEN     ;GET THE TYPE IF ANY
                        ;
027E AF         ZFILL   XRA     A
027F 0618               MVI     B,24
0281 C35502             JMP     TFILL
                        ;
0284 7E         GETOKEN MOV     A,M
0285 B7                 ORA     A           ;IF CHR IS 00 - END OF STRING
0286 CA5302             JZ      SFTOK       ;SO SPACE FILL TOKEN
0289 FE2A               CPI     '*'
028B CA4B02             JZ      QFTOK       ;IF CHAR+8, THE QUESTION FILL
028E FE2E               CPI     '.'
0290 CA5302             JZ      SFTOK       ;IF PERIOD, SPACE FILL THE TOKEN
0293 FE5A               CPI     'Z'
0295 D29F02             JNC     NOFOLD
0298 FE40               CPI     'A'-1
029A DA9F02             JC      NOFOLD
029D E6DF               ANI     0DFH
029F 12         NOFOLD  STAX    D           ;STORE THE CHARACTER
02A0 13                 INX     D
02A1 23                 INX     H
02A2 05                 DCR     B
02A3 C28402             JNZ     GETOKEN     ;LOOP CONTROL OPS
                        ;
02A6 7E         TSKIP   MOV     A,M         ;SKIP THE REMAINDER OF THE INPUT
TOKEN
02A7 B7                 ORA     A
02A8 C8                 RZ      ;RETURN IF CHARACTER IS ZERO
02A9 FE2E               CPI     '.'
02AB C8                 RZ                  ;ALSO RETURN IF PERIOD
02AC 23                 INX     H
02AD C3A602             JMP     TSKIP
```

```
                        ;
                        ;   SUBROUTINE TO GET NEXT CHARACTER FROM A DISK FILE
                        ;
02B0 119B04     GETC    LXI     D,FCB1
02B3 3AE304             LDA     GETBC
02B6 B7                 ORA     A
02B7 CACB02             JZ      GETBUFR  ;IF NO CHARS IN BUFFER - GET NEXT
RECORD
                        ;
02BA 3D         GETCH   DCR     A
02BB 32E304             STA     GETBC    ;UPDATE THE BUFFER COUNTER
02BE 47                 MOV     B,A
02BF 3E7F               MVI     A,BUFSIZ-1
02C1 90                 SUB     B        ;CALCULATE CHARACTER OFFSET
02C2 5F                 MOV     E,A
02C3 1600               MVI     D,0
02C5 21E504             LXI     H,GETBUF
02C8 19                 DAD     D
02C9 7E                 MOV     A,M      ;GET THE CHARACTER
02CA C9                 RET
                        ;
02CB D5         GETBUFR PUSH    D        ;SAVE THE FCB ADDRESS
02CC 11E504             LXI     D,GETBUF
02CF 0E1A               MVI     C,SETDMA
02D1 CD0500             CALL    BDOS     ;SET THE BUFFER ADDRESS
02D4 D1                 POP     D
02D5 0E14               MVI     C,READS
02D7 CD0500             CALL    BDOS     ;READ THE NEXT RECORD
02DA B7                 ORA     A        ;SEE IF READ WORKED
02DB C2E602             JNZ     PENDG    ;ASSUME PHYSICAL EOF ON ERROR
02DE 3E80               MVI     A,BUFSIZ
02E0 32E304             STA     GETBC    ;SET NEW BUFFER SIZE
02E3 C3BA02             JMP     GETCH    ;NOW RETURN THE CHARACTER
                        ;
02E6 37         PENDG   STC     ;SET THE CARRY FLAG
02E7 C9                 RET
                        ;
                        ;   SUBROUTINE TO PUT A CHARACTER TO A DISK FILE
                        ;
02E8 0600       PUTC    MVI     B,0
02EA 11BF04             LXI     D,FCB2
02ED D5         RPUTC   PUSH    D
02EE F5                 PUSH    PSW
02EF 78                 MOV     A,B      ;SEE IF LAST CALL
02F0 B7                 ORA     A
02F1 C20503             JNZ     BFILL    ;YES - FILL LAST BUFFER
02F4 CD3303             CALL    GETAD
02F7 F1                 POP     PSW
02F8 77                 MOV     M,A
02F9 7B                 MOV     A,E
02FA 3C                 INR     A
02FB FE80               CPI     BUFSIZ
02FD CA1503             JZ      PUTBUFR
0300 32E404             STA     PUTBC    ;SAVE THE NEW OFFSET
0303 D1                 POP     D
0304 C9                 RET
                        ;
0305 F1         BFILL   POP     PSW
0306 CD3303             CALL    GETAD
0309 FE80       RPUT    CPI     BUFSIZ
030B CA1503             JZ      PUTBUFR
030E 361A               MVI     M,EOF
0310 3C                 INR     A
0311 23                 INX     H
0312 C30903             JMP     RPUT     ;PUT EOF'S TILL END OF BUFFER
                        ;
0315 AF         PUTBUFR XRA     A
0316 32E404             STA     PUTBC    ;RESET OUTPUT BUFFER COUNTER
0319 116505             LXI     D,PUTBUF
031C 0E1A               MVI     C,SETDMA
031E CD0500             CALL    BDOS     ;SET THE DMA ADDRESS
0321 D1                 POP     D
0322 0E15               MVI     C,WRITES
0324 CD0500             CALL    BDOS     ;WRITE OUT THE BUFFER
0327 B7                 ORA     A
0328 C8                 RZ               ;RETURN IF GOOD WRITE
```

```
0329 115704             LXI     D,FULLMSG       ;PUT OUT ERROR MESSAGE
032C CD0900             CALL    PRINTS
032F D1                 POP     D
0330 C3DF01             JMP     CLOSEI  ;CLOSE FILES AND EXIT
                ;
                ;
0333 3AE404     GETAD   LDA     PUTBC   ;GET THE CHARACTER ADDRESS
0336 5F                 MOV     E,A
0337 1600               MVI     D,0
0339 216505             LXI     H,PUTBUF
033C 19                 DAD     D
033D C9                 RET
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                ;   CONSTANTS AND WORK AREA DEFINITIONS
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
033E 0A2046494CHDMSG    DB      LF,' FILE CONVERTER  VERSION 1.0',CR,LF,'$'
035E 0A2A2A2046ERMSGI   DB      LF,'** FILE NOT FOUND. PLEASE
RETRY.',CR,LF,'$'
0382 0A454E5445INPROMPT DB      LF,'ENTER INPUT FILE NAME: $'
039B 0A454E5445OUTPROM  DB      LF,'ENTER OUTPUT FILE NAME: $'
03B5 0A434F4E56TYPROMP  DB      LF,'CONVERSION TYPE (A OR H)?: $'
03D2 0A444F2059REPEAT   DB      LF,'DO YOU WANT TO CONVERT ANOTHER FILE? (Y
OR N): $'
0403 0A2A2A20450ERRMS   DB      LF,'** ERROR CREATING OR RENAMING ',CR,LF
0424 2020204F55         DB      '   OUTPUT FILE.',CR,LF,'$'
0436 0A2A2A2045CLOSERO  DB      LF,'** ERROR CLOSING OUTPUT FILE.',CR,LF,'$'
0457 0A2A2A204FFULLMSG  DB      LF,'** OUTPUT FILE FULL.',CR,LF,'$'
046F            STORST  DS      2
0471            BUFFR   DS      40
0026 =          BUFFS   EQU     38
0499            PAD     DS      2
049B            FCB1    DS      36
04BF            FCB2    DS      36
04E3            GETBC   DS      1
04E4            PUTBC   DS      1
04E5            GETBUF  DS      128
0565            PUTBUF  DS      128
05E5                    DS      256
                STACK
06E5 =          STACKEND EQU    $
06E5                    END
```

Appendix E
CP/M Messages

Messages come from several different sources. CP/M displays error messages when there are errors in calls to the Basic Disk Operating System (BDOS). CP/M also displays messages when there are errors in command lines. Each utility supplied with CP/M has its own set of messages. The following lists CP/M messages and utility messages alphabetically. Messages other than those listed here might be displayed by an application program. Check the application program's documentation for explanations of those messages.

CP/M Messages

?

DDT. This message has four possible meanings: 1)DDT does not understand the assembly language instruction. 2) The file cannot be opened. 3) A checksum error occurred in a HEX file. 4) The assembler/disassembler was overlayed.

ABORTED

PIP. You stopped a PIP operation by pressing a key.

ADAM TAPE/DISK OR DIRECTORY FULL

There is no more room on the disk or data pack, or you have tried to put more than 64 filenames into the directory.

ASM ERROR MESSAGES

D	Data error: data statement element cannot be placed in specified data area.
E	Expression error: expression cannot be evaluated during assembly.
L	Label error: label cannot appear in this context (might be duplicate label).
N	Not implemented: unimplemented features such as macros are trapped.
O	Overflow: expression is too complex to evaluate.
P	Phase error: label value changes on two passes through assembly.

ASM ERROR MESSAGES (continued)

R Register error: the value specified as a register is incompatible with the code.

S Syntax Error: improperly formed expression.

U Undefined Label: label used does not exist.

V Value error: improperly formed operand encountered in an expression.

BAD DELIMITER

STAT. Check the command line for typing errors.

Bad Load

CCP error message.

BAD SYSTEM, RETRY ON A:

Drive A does not have the CP/M operating system on it. Replace the disk or data pack on drive A with one containing CP/M 2.2 and press ^C.

Bdos Err On d:

Basic Disk Operating System error on the designated drive: CP/M replaces d: with the drive specification of the drive where the error occurred. This message is followed by one of the four phrases in the situations described below.

Bdos Err On d: Bad Sector

This message appears when CP/M finds no disk in the drive, when the disk is improperly formatted, when the drive latch is open, or when power to the drive is off. Check for one of these situations and try again. This could also indicate a hardware problem or a worn or improperly formatted disk. Press ^C to terminate the program and return to CP/M, or press RETURN to ignore the error.

Bdos Err On d: File R/O

You tried to erase, rename, or set file attributes on a Read-Only file. The file should first be set to Read-Write (R/W) with the command: STAT filespec $R/W.

Bdos Err On d: R/O

Drive has been assigned Read-Only status with a STAT command, or the media in the drive has been changed without being initialized with a ^C. CP/M terminates the current program as soon as you press any key.

Bdos Err on d: Select

CP/M received a command line specifying a nonexistent drive. CP/M terminates the current program as soon as you press any key. Press RETURN or CTRL-C to recover.

Break "x" at c

ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred.

- **#** Search failure. ED cannot find the string specified in an F, S, or N command.
- **?** Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, Q, or O command is not alone on its command line.
- **0** The file specified in an R command cannot be found.
- **>** Buffer full. ED cannot put any more characters in the memory buffer, or the string specified in an F, N, or S command is too long.
- **E** Command aborted. A keystroke at the console aborted command execution.
- **F** Disk or directory full. This error is followed by either the disk or directory-full message. Refer to the recovery procedures listed under these messages.

CANNOT CLOSE CP/M FILE

The file conversion was not successful. Use another disk or data pack and begin again. The converted CP/M file cannot be closed.

CANNOT CLOSE DESTINATION FILE--{filespec}

PIP. An output file cannot be closed. You should take appropriate action after checking to see if the correct disk or data pack is in the drive and that the disk is not write-protected.

CANNOT CREATE TARGET

The copy operation cannot be completed. Restart the operation with a different disk or digital data pack.

Cannot close, R/O
CANNOT CLOSE FILES

CP/M cannot write to the file. This usually occurs because the disk is write-protected.

ASM. An output file cannot be closed. This is a fatal error that terminates ASM execution. Check to see that the disk or data pack is in the drive, and that the disk is not write-protected.

DDT. The disk or data pack file written by a W command cannot be closed. This is a fatal error that terminates DDT execution. Check if the correct disk is in the drive and that the disk is not write-protected.

SUBMIT. This error can occur during SUBMIT file processing. Check if the correct system disk is in the A drive and that the disk is not write-protected. The SUBMIT job can be restarted after rebooting CP/M.

**** CANNOT COPY FILE ONTO ITSELF**

COPY has found another file of the same name on the target disk/tape.

**** CANNOT FIND ADAM FILE**

ADAM is unable to find the file you named on the source tape. Check your file name for capital and lower-case letter matches in file names.

**** CANNOT FIND SOURCE**

The filename you specified cannot be found. Check for typing errors and make sure you specified the correct drive name.

CANNOT READ : filename

PIP. PIP cannot read the specified source. Reader cannot be implemented.

CANNOT WRITE

PIP. The destination specified in the PIP command is illegal.
You probably specified an input device as a destination.

Checksum error

PIP. A HEX record checksum error was encountered. The HEX record that produced the error must be corrected, probably by recreating the HEX file.

CHECKSUM ERROR
LOAD ADDRESS hhhh
ERROR ADDRESS hhhh
BYTES READ:
hhhh:

LOAD. File contains incorrect data. Regenerate HEX file from the source.

Command Buffer Overflow

SUBMIT. The SUBMIT buffer allows up to 2048 characters in the input file.

Command too long

SUBMIT. A command in the SUBMIT file cannot exceed 125 characters.

CORRECT ERROR, TYPE RETURN OR CTRL-Z

PIP. A HEX record checksum was encountered during the transfer of a HEX file. The HEX file with the checksum error should be corrected, probably by recreating the HEX file.

**** CP/M FILE NOT FOUND**

CPMADAM. The source file you requested is not on the disk or data pack. Check for typographical errors.

CRC ERROR IGNORE, ABORT, RETRY

Indicates an error pertaining to a data pack. Open the drive door and reseat the data pack before re-trying the operation.

DESTINATION IS R/O, DELETE (Y/N)?

PIP. The destination file specified in a PIP command already exists and it is Read-Only. If you type Y, the destination file is deleted before the file copy is done.

Directory full

SUBMIT. There is not enough directory space to write the $$$.SUB file used for processing SUBMITs. Erase some files or select a new disk and retry.

** DISK IS FULL

The copy operation cannot be complete due to lack of space on the disk or data pack.

DISK FULL

ED. There is not enough disk space for the output file. This error can occur on the W, E, H, or X commands. If it occurs with X command, you can repeat the command prefixing the filename with a different drive.

DISK READ ERROR--{filespec}

PIP. The input disk file specified in a PIP command cannot be read properly. This is usually the result of an unexpected end-of-file.
Correct the problem in your file.

DISK WRITE ERROR--{filespec}

DDT. A disk write operation cannot be successfully performed during a W command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space.

PIP. A disk write operation cannot be successfully performed during a PIP command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space and execute PIP again.

SUBMIT. The SUBMIT program cannot write the $$$.SUB file to the disk. Erase some files, or select a new disk and try again.

ERROR: BAD PARAMETER

PIP. You entered an illegal parameter in a PIP command. Retype the entry correctly.

ERROR: CANNOT OPEN SOURCE, LOAD ADDRESS hhhh

LOAD. Displayed if LOAD cannot find the specified file or if no filename is specified.

ERROR: CANNOT CLOSE FILE, LOAD ADDRESS hhhh

LOAD. Caused by an error code returned by a BDOS function call.
Drive might be write-protected.

ERROR: DISK READ, LOAD ADDRESS hhhh

LOAD. Caused by an error code returned by a BDOS function call.

ERROR: DISK WRITE, LOAD ADDRESS hhhh

LOAD. Destination drive is full.

```
ERROR: INVERTED LOAD ADDRESS, LOAD ADDRESS hhhh
```

LOAD. The address of a record was too far from the address of the previously-processed record. This is an internal limitation of LOAD, but it can be circumvented. Use DDT to read the HEX file into memory, then use a SAVE command to store the memory image file on disk.

```
** ERROR In Writing System
```

SYSGEN cannot write system to target drive. Try adjusting media.

```
ERROR: NO MORE DIRECTORY SPACE, LOAD ADDRESS hhhh
```

LOAD. Disk directory is full.

```
** ERROR IN READING DRIVE d:
ABORT OR RETRY
(A or R)?
```

BACKUP. Error occurred while reading from the source drive. ABORT and begin again.

```
Error on line nnn message
```

SUBMIT. The SUBMIT program displays its messages in the format shown above, where nnn represents the line number of the SUBMIT file. Refer to the message following the line number.

```
** ERROR READING DRIVE d:
ABORT OR RETRY
(A OR R)?
```

ADAM or CPMADAM. Cannot read source drive.

```
** ERROR READING DRIVE d:
ABORT, RETRY OR CONTINUE
(A, R OR C)?
```

BACKUP. Cannot write to target drive.

```
** ERROR WRITING TO DRIVE d:
ABORT OR RETRY
(A OR R)?
```

CPMADAM cannot write to target (ADAM) drive.

```
FILE ERROR
```

ED. Disk or data pack or directory is full, and ED cannot write anything more on the drive. This is a fatal error, so make sure there is enough space on the disk to hold a second copy of the file before invoking ED.

```
FILE EXISTS
```

You have asked CP/M to create or rename a file using a file specification that is already assigned to another file. Either delete the existing file or use another file specification.

REN. The new name specified is the name of a file that already exists. You cannot rename a file with the name of an existing file. If you want to replace an existing file with a newer version of the same file, either rename or erase the existing file, or use the PIP utility.

File exists, erase it

ED. The destination filename already exists when you are placing the destination file on a different disk than the source. It should be erased or another disk selected to receive the output file.

** FILE IS READ/ONLY **

ED. The file specified in the command to invoke ED has the Read-Only attribute. Ed can read the file so that the user can examine it, but ED cannot change a Read-Only file.

File Not Found

CP/M cannot find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive.

STAT. STAT cannot find the specified file. The message might appear if you omit the drive specification. Check to see if the correct disk is in the drive.

FILE NOT FOUND--{filespec}

PIP. An input file that you have specified does not exist.

filename required

ED. You typed the ED command without a filename. Reenter the ED command followed by the name of the file you want to edit or create.

hhhh??=dd

DDT. The ?? indicates DDT does not know how to represent the hexadecimal value dd encountered at address hhhh in 8080 assembly language. dd is not an 8080 machine instruction opcode.

INSUFFICIENT DRIVE OR DIRECTORY SPACE

The file conversion cannot be completed due to lack of available space on the disk or data pack. This message is also displayed when the number of filenames on a directory exceeds 64.

Insufficient memory

DDT. There is not enough memory to load the file specified in an R or E command.

Invalid Assignment

STAT. You specified an invalid drive or file assignment, or misspelled a device name. This error message might be followed by a list of the valid file assignments that can follow a filename. If an invalid drive assignment was attempted the message Use: d:=RO is displayed, showing the proper syntax for drive assignments.

Invalid control character

SUBMIT. The only valid control characters in the SUBMIT files of the type SUB are ^ A through ^ Z. Note that in a SUBMIT file the control character is represented by typing the circumflex, ^, not by pressing the control key.

INVALID DIGIT--{filespec}

PIP. An invalid HEX digit has been encountered while reading a HEX file. The HEX file with the invalid HEX digit should be corrected, probably by recreating the HEX file.

Invalid Disk Assignment

STAT. Might appear if you follow the drive specification with anything except =R/O.

INVALID DISK SELECT

CP/M received a command line specifying a nonexistent drive, or the disk in the drive is improperly formatted. CP/M terminates the current program as soon as you press any key.

Invalid File Indicator

STAT. Appears if you do not specify RO, RW, DIR, or SYS.

INVALID FORMAT

PIP. The format of your PIP command is illegal. See the description of the PIP command.

INVALID HEX DIGIT
LOAD ADDRESS hhhh
ERROR ADDRESS hhhh
BYTES READ:
hhhh

LOAD. File contains incorrect HEX digit.

INVALID SEPARATOR

PIP. You have placed an invalid character for a separator between two input filenames.

INVALID USER NUMBER

PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15.

Missing Block, Adjust Media
Abort, Retry

This message appears if there is a bad section of tape or disk. It also appears when a non-CP/M disk or data pack is used.

Missing Drive

This message appears when the system is directed to a drive that has been configured but is not present.

Missing Media
Abort, Retry

The system cannot find a disk or data pack in the drive specified.

n?

USER. You specified a number greater than fifteen for a user area number. For example, if you type USER 18<cr>, the screen displays 18?.

NO DIRECTORY SPACE

ASM. The disk directory is full. Erase some files to make room for PRN and HEX files. The directory can usually hold only 64 filenames.

NO DIRECTORY SPACE--{filespec}

PIP. There is not enough directory space for the output file. You should either erase some unnecessary files or get another disk with more directory space and execute PIP again.

NO FILE--{filespec}

DIR, ERA, REN, PIP. CP/M cannot find the specified file, or no files exist.

ASM. The indicated source file cannot be found on the indicated drive.

DDT. The file specified in an R or E command cannot be found on the disk or data pack.

NO FILE FOUND

The filename you specified cannot be found. Check for typographical errors and to be sure that you have specified the correct drive.

NO INPUT FILE PRESENT ON DISK

DUMP. The file you requested does not exist.

No memory

There is not enough (buffer?) memory available for loading the program specified.

**** NO RECORDS IN CP/M FILE**

The file specified exists, but it is empty.

NO SOURCE FILE PRESENT

ASM. The assembler cannot find the file you specified. Either you mistyped the file specification in your command line, or the filetype is not ASM.

NO SPACE

SAVE. Too many files are already on the disk, or no room is left on the disk to save the information.

No SUB file present

SUBMIT. For SUBMIT to operate properly, you must create a file with filetype of SUB. The SUB file contains usual CP/M commands. Use one command per line.

NOT A CHARACTER SOURCE

PIP. The source specified in your PIP command is illegal. You have probably specified an output device as a source.

NOT A LEGAL CP/M FORMAT

The disk or data pack you are using has not been formatted for CP/M.

**** NOT DELETED ****

PIP. PIP did not delete the file, which might have had the R/O attribute.

OUTPUT FILE WRITE ERROR

ASM. You specified a write-protected disk as the destination for the PRN and HEX files, or the disk has no space left. Correct the problem before assembling your program.

Parameter error

SUBMIT. Within the SUBMIT file of type sub, valid parameters are $0 through $9.

QUIT NOT FOUND : filename and parameter

PIP. The string argument to a Q parameter was not found in your input file.

Read error

TYPE. An error occurred when reading the file specified in the type command. Check the disk and try again. The STAT filespec command can diagnose trouble.

READER STOPPING

PIP. Reader operation interrupted.

Record Too Long

PIP. PIP cannot process a record longer than 128 bytes.

Requires CP/M 2.0 or later

XSUB. XSUB requires the facilities of CP/M 2.0 or newer version.

Requires CP/M 2.0 or new for operation

PIP. This version of PIP requires the facilities of CP/M 2.0 or newer version.

SOURCE FILE NAME ERROR

ASM. When you assemble a file, you cannot use the wildcard characters * and ? in the filename. Only one file can be assembled at a time.

SOURCE FILE READ ERROR

ASM. The assembler cannot understand the information in the file containing the assembly-language program. Portions of another file might have been written over your assembly-language file, or information was not properly saved on the disk. Use the TYPE command to locate the error. Assembly-language files contain the letters, symbols, and numbers that appear on your keyboard. If your screen displays unrecognizable output or behaves strangely, this probably indicates that computer instructions have been written over your file.

START NOT FOUND : filename and parameter

PIP. The string argument to an S parameter cannot be found in the source file.

"SYSTEM" FILE NOT ACCESSIBLE

You tried to access a file set to SYS with the STAT command.

** TARGET DRIVE SMALLER THAN SOURCE DRIVE

A disk holds less records than a data pack. You cannot backup from a datapack onto a disk.

** TOO MANY FILES **

STAT. There is not enough memory for STAT to sort the files specified, or more than 512 files were specified.

UNEXPECTED END OF HEX FILE--{filespec}

PIP. An end-of-file was encountered prior to a termination HEX record. The HEX file without a termination record should be corrected, probably by recreating the HEX file.

Unknown Drive
Ignore, Abort

The drive you specified does not exist in the current configuration.

UNKNOWN ERROR

The system is unable to descern the source of the error. Retry the operation.

Unrecognized Destination

PIP. Check command line for valid destination.

Use: STAT d:=R/O

STAT. An invalid STAT drive command was given. The only valid drive assignment in STAT is STAT d:=R/O.

VERIFY ERROR:--{filespec}

PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. Usually this indicates a failure of either the destination disk or drive.

```
Write Protected
Abort, Retry
```

PIP. The disk is WRITE PROTECTED. Remove the protection tab and press either Ignore or Retry.

```
WRONG CP/M VERSION (REQUIRES 2.0)
```

```
XSUB ACTIVE
```

SUBMIT. XSUB has been invoked.

```
XSUB ALREADY PRESENT
```

SUBMIT. XSUB is already active in memory.

```
Your input?
```

If CP/M cannot find the command you specified, it returns the command name you entered followed by a question mark. Check that you have typed the command line correctly, or that the command you requested exists as a .COM file on the default or specified disk.

Appendix F
Summary of Commands

How to Enter a CP/M Command

To enter a CP/M command, you must type a complete command line in response to a CP/M system prompt. A command line is composed of a keyword, an optional command tail, and a RETURN. The keyword identifies a command to be executed. The optional command tail consists of a file specification plus options or parameters. To complete the command, you must press the RETURN key.

CP/M File Specifications

CP/M identifies every file by its unique specification. A file specification is composed of three parts: a drive specification, a primary filename, and a filetype. The term filespec is an abbreviation for file specification and indicates any valid combination of drive specification, filename, and filetype. This summary uses the following symbols to represent the parts of a filespec.

`d:`	the optional drive specification. Must be a single alphabetical character in the range A through P, followed by a colon.
`filename`	the required primary filename. 1 to 8 alphanumeric characters.
`.typ`	the optional filetype. 0 to 3 alphabetic or numeric characters separated from the primary filename by a period.

Valid combinations of the elements of a filespec are shown below:

- filename
- d:filename
- filename.typ
- d:filename.typ

Certain CP/M commands can select and process several files when wildcard characters are included in the primary filename or filetype. The two wildcard characters are ?, which matches any single letter in the same position, and *, which matches any character at that position and any other characters remaining in the filename or filetype. The command summary indicates which commands can accept wildcard characters (* or ?) in a filename or filetype.

Command Summary Conventions

The following special symbols define command syntax in the command summary:

{}	indicate an optional item within DDT and ED.
|	separates choices.
n	indicates a number.
<cr>	indicates a RETURN keystroke.
^ or CTRL	indicates a CONTROL keystroke.
o	indicates an option or option list.
R/W	means Read-Write.
R/O	means Read-Only.
SYS	means System attribute.
DIR	means Directory attribute.
...	preceding element can be repeated as many times as desired.
*	wildcard - replaces all or part of a filename and/or filetype.
?	wildcard - replaces any single character in the same position.

Note: The required parts of the command line are shown in boldface. Optional parts are in normal type.

ADAM

Syntax:

```
ADAM
```

Purpose:

This utility converts EOS (SmartWRITER and SmartBASIC) files to CP/M compatible files. The conversion prompts you for the information required. The file type is indicated by the letter to the left of the file name in the directory. When the conversion is complete, ADAM asks you if you want to convert another file. If you type N, the current CP/M drive prompt re-appears.

Prompt 1

```
ADAM TAPE/DISK IS DRIVE
(A, B, C, or D)?
```

Prompt 2

```
VOLUME = [volume]
TRANSFER ADAM FILE?
```

Prompt 3

```
ADAM FILE TYPE
(H, h, A, or a)
```

Prompt 4

```
NAME CPM FILE?
```

Prompt 5

```
ANOTHER FILE TO TRANSFER
(Y or N)? __
```

ASM

Syntax:

```
ASM filename
ASM filename.ABC
```

Purpose:

ASM assembles assembly language statements, producing both an output file in Intel™ hexadecimal format and a print file. The source filetype must be ASM. Character A indicates source file drive, B indicates HEX file destination, and C indicates PRN file destination. If you put a Z in position B or C, the HEX file or PRN file is suppressed. An X in position C sends the print file to the console.

Examples:

```
A>ASM PROG
A>ASM PROG.BBX
A>ASM b:PROG.bzz
```

BACKUP

Syntax:

BACKUP

Purpose:

This utility lets you copy the entire contents of a disk or data pack. System files, if present, are copied. The utility copies a disk or data pack exactly. It does not reorganize the files to maximize space allocation. The utility prompts you for the information required. You can use either a single or multi-drive system to do backups.

Prompt 1

```
BACK UP TO DRIVE
(A, B, C, or D)?
```

Prompt 2

```
INSERT SOURCE MEDIA
PRESS RETURN WHEN READY
```

Prompt 3

```
INSERT TARGET MEDIA
PRESS RETURN WHEN READY
```

Prompt 4

```
Back Up Another
(Y or N)? —
```

CONFIG

Syntax:

```
CONFIG
```

Purpose:

This utility lets you change system parameters by modifying the values in a table. The parameters you can change are presented as entries on a menu. Before you can alter any of the items listed on the menu, select an option that reads the tables into the TPA. When you are finished making changes, you must write the new tables out to a drive if you want to record the changes permanently. Because of the number of options available to you, the utility provides you with a series of menus and prompts. Below is a list of the parameters you can change.

```
Keyboard Translations
System Smart Key Menu
Character Colors
Cursor Changes
Background Color
Serial Card Settings
Set Default I/O Byte
Initial Smart Key State
```

COPY

Syntax:

```
COPY afn destination:
COPY ufn destination:
COPY filename A filename B
```

Purpose:

COPY lets you duplicate selected files. Files can be copied from one disk or data pack to another, or they can be copied onto the same disk or data pack. Use the third format shown to copy a file onto the same disk. The operating system can only be copied using the SYSGEN command.

Examples:

```
A>COPY MYFILE.* B:
A>COPY MYFILE.BAK B:
A>COPY OLDFILE NEWFILE
```

CPMADAM

Syntax:

```
CPMADAM
```

Purpose:

This utility converts CP/M files to EOS compatible files so that they can be used with SmartWRITER and SmartBASIC. The utility prompts for the information required. The EOS file that is created has a filetype of A (that of a SmartBASIC file) unless otherwise specified. To specify a different EOS filetype, follow the ADAM filename with the filetype in brackets. For example, `ADAMFILE[h]`.

Prompt 1

```
ADAM TAPE/DISK IS DRIVE
(A,B,C or D)? __
```

Prompt 2

```
CPM FILE NAME?
```

Prompt 3

```
ADAM FILE NAME?
```

Prompt 4

```
Another file to transfer
(Y or N)?
```

DDT

Syntax:

```
DDT filespec
```

Purpose:

DDT is used in debugging machine language programs. If you specify DDT without a filespec, DDT loads into the TPA and prompts for a command. The command character can be followed by one or more arguments: hexadecimal values, file specifications, or other information, depending on the command. Arguments are separated from each other by commas or spaces. No spaces are allowed between the command character and the first argument. Optional parts of the command are enclosed in curly brackets {}.

Examples:

```
A> DDT
A> DDT PROGRAM.COM
```

DDT Command Summary

A{s} (Assemble)
Enter Assembly Language Statements

D{s{,f}} (Display)
Display Memory in Hex and ASCII

Fs,f,bc (Fill)
Fill Memory Block - Byte

G{s}{,b1{,b2}} (Go)
Begin Execution

Hc1,c2 (Hex)
Hexadecimal Sum and Difference

Ifilespec (Input)
Set Up Input Command Line for R or W commands

L{s{,f}} (List)
List Memory in Mnemonic Form

(continued)

DDT Command Summary (continued)

Ms,f,d (Move)
Move Memory Block

R{o} (Read)
Read Disk File to Memory

Ss (Set)
Set Memory Values

T{n} (Trace)
Trace Program Execution

U{n} (Untrace)
Monitor Execution without Trace

X{r} (Examine)
Examine and Modify CPU Registers

Symbol	Meaning
bc	byte constant
b1	breakpoint one
b2	breakpoint two
c1	byte or word constant one
c2	byte or word constant two
d	destination for data
f	final address
n	number of instructions to execute
o	offset to load address
r	8080 CPU register or flag name
s	starting address

DIR

Syntax:

```
DIR d:filename.typ
```

Purpose:

DIR displays names of files on the specified disk. DIR does not display files set to SYS.

Examples:

```
A> DIR
A> DIR B:
A> DIR B:MYFILE.TEX
A> DIR A*.ASM
A> DIR PROG???.PRN
A> DIR PROGRAM.*
```

DUMP

Syntax:

```
DUMP ufn
```

Purpose:

The DUMP program prints the contents of the unambiguous file specified in hexidecimal form at the console. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexidecimal.

Examples:

```
DUMP MYFILE
```

ED

Syntax:

```
ED d: filename.typ
```

Purpose:

ED is the CP/M line editor. Using the ED subcommands, you can create or alter files.

Example:

```
A> ED TEST.DAT
```

ED Control Characters

Character	Effect
	IN ED, the control key appears as an up arrow.
↑L	Logical <cr><lf> within strings
↑X	Line delete
↑Z	String terminator/separator, exit insert mode.
backspace	Delete character
break	Stop function

ED Command Summary

Command	Action
nA	Append n lines from original file to memory buffer.
0A	Append file until buffer is half full.
#A	Append file until buffer is full (or end of file).
B, −B	Move CP to the beginning (B) or bottom (−B) of buffer.
nC, −nC	Move CP n characters forward (C) or back (−C) through buffer.
nD, −nD	Delete n characters before (−D) or after (D) the CP.
E	Save new file and return to CP/M.
Fstring {↑Z}	Find character string.
H	Save the new file, then reedit, using the new file as the original file.
I	Enter insert mode; use ↑Z to exit insert mode.
Istring {↑Z}	Insert string at CP.
Jsearch_str↑Zinsert_ string↑Zdelete_to_str{↑Z}	Juxtapose strings.
nK, −nK	Delete (kill) n lines from the CP.
nL, −nL, 0L	Move CP n lines.
nMcommands	Execute commands n times.
n, −n	Move CP n lines and display that line.

(continued)

ED Command Summary (continued)

Command	Action
n:	Move to line n.
:ncommand	Execute command through line n.
Nstring {↑Z}	Extended find string.
O	Return to original file.
nP, −nP	Move CP 23 lines forward or backward and display 23 lines at console.
Q	Abandon new file; return to CP/M.
R	Read X$$$$$$$.LIB file into buffer.
Rfilespec	Read filename.LIB into buffer.
Sdelete str↑Zinsert str {↑Z}	Substitute string.
nT, −nT, 0T	Type n lines.
U, −U	Upper-case translation.
V, −V, 0V	Line numbering on/off; display free buffer space.
nW	Write n lines to new file.
nX	Write n lines to temporary LIB file.
0X	Delete file X$$$$$$$.LIB.
nZ	Wait n seconds.

ERA

Syntax:

```
ERA filename.typ
```

Purpose:

ERA erases a file. Groups of files can be erased using ambiguous filenames.

Examples:

```
A> ERA PROGRAM.BAS
A> ERA B:LETTER.DAT
A> ERA A:LETTER.*
A> ERA *.BAK
A> ERA B:*.*
```

FORMAT

Syntax:

FORMAT

Purpose:

The format utility initializes a disk or data pack for use with the CP/M operating system. If a disk or digital data pack is not formatted for CP/M, it defaults to the EOS operating system that is built in to ADAM.

Examples:

Prompt 1

```
FORMAT MEDIA ON DRIVE
(A, B, C OR D)?
```

Prompt 2

```
INSERT DISK TO BE FORMATTED
INTO DRIVE A

PRESS RETURN WHEN READY
TO FORMAT

FORMATTING BLOCK 000
```

LOAD

Syntax:

```
LOAD ufn
```

Purpose:

The LOAD command reads the unambiguous file specified. The file is assumed to contain HEX format machine code. LOAD produces a memory image file that can subsequently be executed.

Examples:

```
LOAD MYFILE
```

PIP

Syntax:

```
Destination=Source

PIP d:|filespec=filespec[option]
PIP filespec=d:[option]
PIP d:filename.typ=filespec1, filespec2, ...
PIP filespec|device:=filespec[options]
|device:[options]
PIP <cr>
*
```

Purpose:

PIP copies files, combines files, and transfers files between peripheral devices. The first filespec is the destination; the second filespec is the source. Source filespecs with options can be repeated, separated by commas, to combine two or more files into one concatenated file. The source or destination can be any CP/M logical device. PIP entered with no command tail displays a * prompt and awaits your series of commands, entered and executed one line at a time.

Examples:

COPY FROM ONE DISK TO ANOTHER:

```
A> PIP B:=A:DRAFT.TEX
A> PIP B:DRAFT.TEX=A:
```

COPY A FILE AND RENAME IT:

```
A> PIP B:NEWDRAFT.TEX=A:OLDDRAFT.TEX
A> PIP NEWDRAFT.TEX=OLDDRAFT.TEX
```

COPY MULTIPLE FILES:

```
A> PIP <cr>
A> PIP B:=*.TEX [AV]
A> PIP B:=*.COM [RW]
A> PIP B:=C:DRAFT.*
A> PIP B:=*.*
A> PIP B:=C:*.*
```

COMBINE MULTIPLE FILES:

```
A> PIP B:NEW.DAT = FILE1.DAT,FILE2.DAT
```

COPY, RENAME, AND GET FROM USER 1:

```
A> PIP NEWDRAFT.TEX = OLDDRAFT.TEX[G1]
```

COPY TO/FROM LOGICAL DEVICES:

```
A> PIP B:FUNFILE.SUE=CON:
A> PIP LST:=CON:
A> PIP LST:=B:DRAFT.TEX [T8]
A> PIP PRN:=B:DRAFT.TEX
```

PIP Options

Option	Description
A	Archive: copy only modified files.
Dn	Delete any characters past column n.
E	Echo transfer to console.
F	Filter form-feeds from source data.
Gn	Get file from user number n.
H	Test for valid Hex format.
I	Ignore :00 Hex data records, test for valid Hex format.
L	Translate upper-case to lower-case.
N	Number the output lines, incrementing by 1 starting at 1.
O	Object file transfer, ↑Z ignored.
Pn	Set page length to n (default n = 60).
Qs↑Z	Quit copying from source at string s.
R	Read files that have been set to SYS (system).
Ss↑Z	Start copying from the source at the string s.
Tn	Expand tabs to n spaces.
U	Translate lower-case to upper-case.
V	Verify that data has been written correctly.
W	Write over files that have been set to Read-Only.
Z	Zero the parity bit.

PIP Logical Devices

The following devices are mapped in the IOBYTE using STAT and must be defined in BIOS.

CON:	Console device
LST:	List device (printer)
PUN:	Punch device (reads serial port)
RDR:	Reader device (writes to serial port)
INP:	Patched character input
OUT:	Patched character output
PRN:	Like LST; tabs on 8th character, page breaks on 60th line; numbers lines.
EOF:	Generates CTRL-Z (end-of-file).
NUL:	Generates 40 nulls for PUN device.

PIP Physical Devices

TTY:	Console, terminal, reader, punch, teletypewriter
CRT:	Console, terminal, CRT device
PTR:	Paper tape or card reader
PTP:	Paper tape or card punch
LPT:	List device, line printer
UC1:	User-defined console
UR1:	User-defined reader
UR2:	User-defined reader
UP1:	User-defined punch
UP2:	User-defined punch
UL1:	User-defined listing device

REN

Syntax:

```
REN newfile.typ = oldfile.typ
```

Purpose:

REN changes the name of the existing file, called oldfile, to the name specified by newfile.

Examples:

```
A> REN NEWFILE.DAT=OLDFILE.INP
A> REN NEWFILE.DAT=OLDFILE.DAT
B> REN NEWLIST=OLDLIST
```

SAVE

Syntax:

```
SAVE n d:filename.typ
```

Purpose:

SAVE copies to a disk or data pack the contents of the Transient Program Area (TPA) starting at location 100 hex and continues to the page indicated by n in the command tail. SAVE assigns the filename and filetype specified in the command tail. Wildcard characters are not accepted. The number of pages to save can be calculated using DDT.

Examples:

```
A> SAVE 3 EXTRA.COM
A> SAVE 40 QUICK
A> SAVE 4 X.Y
A> SAVE 10 B:ZOT.COM
```

STAT

Syntax:

Display Status:

```
STAT
STAT d:
STAT filespec
STAT d:DSK:
STAT DEV:|VAL:|USR:
```

Change Status:

```
STAT d:=R/O
STAT filespec $R/O|R/W|SYS|DIR
STAT filespec $S
```

Purpose:

STAT returns information about files, drives, and other peripheral devices. STAT changes attributes of drives and files. STAT with no command tail returns free storage space in kilobytes for the current drive, and also tells if the drive is Read-Only (R/O) or Read-Write (R/W). Use STAT to set the drive to R/O and CTRL-C to reset it to R/W. STAT filespec returns the number of kilobytes used by a file, and the attributes of that file. STAT filespec can, with an option, set a file or files to R/O, R/W, SYS, or DIR. STAT displays files set to SYS in parentheses. STAT accepts wildcards in the command tail.

Examples:

```
A> STAT
A> STAT myfile.txt
A> STAT C:letter.bak
A> STAT genledgr.dat $R/O
A> STAT *.com $R/O
A> STAT *.bak
A> STAT B:*.*
```

STAT Options:

R/W	Read-Write
R/O	Read-Only
SYS	System attribute
DIR	No system attribute
S	Display the size of the file or files based on last record numbers.
USR:	Display USER numbers containing files.
DSK:	Display characteristics of the drive.
VAL:	Display possible STAT commands and devices.
DEV:	Displays current logical device assignments.

SUBMIT

Syntax:

```
SUBMIT filename parameters
```

Purpose:

SUBMIT starts execution of a file of CP/M commands, one command per line in the file. The SUBMIT file must have a filetype of .SUB. Any optional parameters following the file specification in the command line, such as a drive or file specification, are substituted for their corresponding formal parameters ($1,$2...) in the SUBMIT file.

Examples:

```
A> SUBMIT START
A> SUBMIT B:START
A> SUBMIT START C:LETTER
```

In the preceding examples, START.SUB is the SUBMIT file. The command lines in START.SUB are not displayed here. In the second example, START.SUB is on drive B. In the last example, the actual parameters C: and LETTER follow the SUBMIT filename. SUBMIT.COM creates a temporary file and substitutes the actual parameters in the command line for the formal parameters ($1, $2...) appearing in the SUBMIT file START.SUB. C: is substituted for any $1 parameters that appear in the SUBMIT file. START.SUB LETTER is substituted for any $2 parameters that appear in the START.SUB SUBMIT file.

SYSGEN

Syntax:

SYSGEN

Purpose:

The SYSGEN copies the CP/M operating system from the source data pack or disk onto the target data pack or disk. SYSGEN does not destroy user files. SYSGEN does not initialize (FORMAT) a data pack or disk. The utility prompts for the information required.

Prompt 1

```
ENTER SOURCE DRIVE NAME
(OR RETURN TO REBOOT)?
```

Prompt 2

```
ENTER DESTINATION DRIVE NAME
(OR RETURN TO REBOOT)?
```

Prompt 3

```
ENTER DESTINATION DRIVE NAME
(OR RETURN TO REBOOT)?
```

TYPE

Syntax:

```
TYPE d:filename.typ
```

Purpose:

TYPE displays the contents of an ASCII file on the screen. Press any key to stop the display. TYPE does not accept wildcard filespecs. Entering a CTRL-P before the type command causes the output to be echoed to the printer; another CTRL-P stops the printing.

Examples:

```
A> TYPE letter.dat
B> TYPE a:document.law
B> TYPE program.bas
A> TYPE B:reader.asm
```

USER

Syntax:

```
USER n
```

Purpose:

USER with a number from 0 to 15 changes the current user number to the number specified by n. CP/M assumes a default user number of 0. STAT USR: displays user numbers containing files.

Examples:

```
B> USER 2
A> USER 7
```

XSUB

Syntax:

```
XSUB
```

Purpose:

XSUB, when used within a SUBMIT file, allows most programs to accept input from command lines in the SUBMIT file rather than from the console. The XSUB command must appear before the command line that invokes the program requiring console input.

Example:

```
A> TYPE SAVER.SUB

XSUB
DDT
I$1.HEX
R
GO
SAVE 1 $2.COM
```

CP/M Control Character Summary

Keystroke	Action
CTRL-C	aborts the current program and performs a warm start.
CTRL-E	forces a physical carriage return, but does not send command to CP/M.
CTRL-H	same as BACKSPACE.
CTRL-J	line-feed; terminates input at the console.
CTRL-L<cr>	clears ADAM screen.
CTRL-M	same as carriage return.
CTRL-P	echoes all console activity at the printer; a second CTRL-P ends printer echo.
CTRL-R	retypes current command line; useful after using RUB or DEL key.
CTRL-S	stops console listing temporarily; CTRL-S resumes the listing.
CTRL-U	cancels line; displays #; cursor moves down one line and awaits a new command.
CTRL-X	deletes all characters in command line.
CTRL-Z	string or field separator.
BACKSPACE	moves cursor back one space; erases previous character.
RETURN	carriage return.
^left arrow	Scroll screen left
^right arrow	Scroll screen right

CP/M Filetypes

The filetype is an optional three character ending separated from the filename by a period. The filetype generally indicates a special kind of file. Common filetypes and their meanings follow.

Filetype	Meaning
.ASM	Assembly language source file; the CP/M Assembler, ASM, assembles or translates an .ASM file into machine language.
.BAK	Back-up file created by text editor; the editor renames the source file with this filetype to indicate that the original file has been processed. The original file stays on disk as the back-up file, so you can refer to it.
.COM	8080 executable file.
.HEX	Program file in Intel hexadecimal format.
.IRL	Indexed REL file produced by LIB.
.PRL	Page Relocatable file, a file that does not require an absolute segment. It can be relocated in any page boundary (256 Bytes).
.PRN	Printable file displayable on console or printer.
.SUB	Filetype required for submit file containing one or more CP/M commands. The SUBMIT program executes commands in files of type SUB, providing a batch execution mode for CP/M.
.XRF	Cross-reference file produced by XREF.
.$$$	Temporary file.

INDEX

INDEX