CP/M®
Simplified

Jeffrey R. Weber

WSI WEBER SYSTEMS INCORPORATED

CP/M SIMPLIFIED

1st EDITION

Jeffrey Weber

Weber Systems Inc.
Cleveland, Ohio

Published by
Weber Systems Inc.
8437 Mayfield Road
Cleveland, Ohio 44026

For information on translations and book distributors outside the USA, please contact WSI at the above address.

CP/M Simplified First Edition

Library of Congress Catalog Card Number 81-66910
ISBN 0-938862-05-7

Typesetting: Shelly Harpold

Contents

1. INTRODUCTION TO CP/M AND MP/M 9

Typical Computer System 9. Computer 10. Disk Drives 10. Terminal Display and Keyboards 13. Printer 13. Software 14. CP/M, MP/M Defined 14. CP/M Versions 15.

2. USING CP/M FOR BUSINESS COMPUTING 17

Custom vs. Package Programs 17. WSI Business Applications Packages 18. Hardware Requirements 19. CP/M Operating System 22. Accounts Receivable System 23. Accounts Payable System 24. General Ledger System 25. Payroll/Personnel System 25. Inventory System 27. Mailing List 27. Word Processing 28.

3. BASIC CP/M OPERATION 31

Copying the System Diskette 32. Cold Start 33. System Prompt 33. Warm Start 34. Disk Directory 34. DIR 35. Running a CBASIC Program 36. ED 37. REN 39. Loading a Diskette 39. PIP 40. Printing Files 43. Erasing Files 45.

4. CP/M AND MP/M COMMANDS 47

Command Format 48. Control Characters 49. File Handling Commands 52. Device Handling Commands 54. Program Handling Commands 56. Control Characters 57. Filenames 57. Filename Match 58. Filename Extension Types 59. Built-In and Transient Commands 61. DIR 61. TYPE 62. REN 63. ERA 63. SAVE 64. SYSGEN 64. STAT 66. SUBMIT 73. SUB with XSUB 75. ASM 77. LOAD 80.DUMP 82. DDT 83. SAVE 84. Calculating Pages 85.

5. MP/M AND CP/M 2.2 87

Program Scheduling 87. User Areas 92. File Protection 94. DISKRESET 96. SPOOL 97. SCHED 98. TOD 98. ABORT 99. ATTACH 99. CONSOLE 100. DIR 100. ERASE 100. TYPE 101. MPMSTAT 101. GENMOD 103. GENHEX 103. PRLCOM 103. GENSYS 104. MPMLDR 104.

6. USING THE PIP COMMAND TO HANDLE FILES 107

PIP Defined 108. Copying a Single File 108. Copying Several Files 110. Copying all Files 112. Copying With a Drive 113. With 2 Drives 114. Aborting PIP 116. Using PIP to Copy Other Peripherals 116. Device Keywords 119. Examples of PIP File Transfers 119. Physical Device Names in PIP 124. Special Device Names 124. Sending Text Files to Devices 125. ASC II Conversion Table 127. ASC II Character Codes 128. Concatenating Text Files 132. Concatenating Non-Text Files 134. Concatenating Hex Files 134. PIP Parameters 136. PIP Parameter Examples 139. Copying Part of a File 140. Using PIP in CP/M 2.2 and MP/M 141. File Attributes 142. Using PIP to Copy from User Areas 142. Using PIP with Read-Only Files 144. Using PIP with System Files 145.

7. THE CP/M EDITOR (ED) 147

Introduction 147. CP and Line Numbers 150. ED Mechanics 150. ED with Source Files 157. Accidentally Erasing the Edit Buffer 158. Creating a New File 158. Using T to Display Text 165. Moving the CP 166. Determining the CP Position 166. Ending the ED Session 169. Using the A Command 171. Moving Inside the Edit Buffer 172. Changing Text with ED 176. Substituting Text in the Edit Buffer 179. Writing Lines to the Edit Buffer 182. The N Command 184. R Command and Library Source File 185. X Command and Holding File 185. J Command 187. M Command 188. ED Error Conditions 188.

8. INTERNAL OPERATION OF CP/M 191

Overview 191. Allocation of Memory 193. CP/M File Structure 195. File Control Block 196. File Control Block Descriptions 199. CP/M System Operation Overview 201. FDOS and CCP 202. BIOS 204. BDOS 204. BIOS Function Numbers 205. BDOS Function Numbers 206. Altering CP/M 213. Altering Memory Size 213. Installing MP/M 214. Altering CP/M 218.

9. CP/M AND MP/M REFERENCE GUIDE 219

Format 219. ABORT 221. ASM 223. ATTACH 226. CONSOLE 227. DDT 228. DIR 230. DSKRESET 233. DUMP 234. ED 235. ERA 236. ERAQ 238. GENHEX 240. GENMOD 241. GENSYS 243. LOAD 245. MOVCPM 247. MPMLDR 249. MPMSTAT 250. PIP 251. PRLCOM 256. REN 257. SAVE 258. SCHED 261. SPOOL 262. STAT 263. STOPSPLR 266. SUBMIT 267. SYSGEN 269. TOD 271. TYPE 273. USER 274. XSUB 275.

10. CBASIC PROGRAMMING LANGUAGE 277

CBASIC Beginnings 277. CBASIC Structure 277. CBASIC Errors 278. Using ED to Edit a CBASIC Program 279. Entering New Text with I 280. Correcting Errors in ED 281. Saving the Program on Disk 282. Compiling the CBASIC Program 283. Running the Compiled CBASIC Program 284. Introduction to Programming Practices 284. Flowcharting Techniques 285. Module Structure 285. CBASIC Program Statements 288. Flowchart Symbol Description 289. Line Numbers 291. Remark Statements 291. CBASIC Keyword Summary 293.

11. PRACTICAL OPERATIONAL GUIDELINES 295

Introduction 295. Computer Room Organization 296. Environmental Problems 296. System Documentation 298. Computer Room Supplies 299. Copying Diskettes 300. Storing and Handling Diskettes 300. The Printer 302. How to Shut Down the System 303. CP/M and Diskette Space 303. System Troubleshooting 303.

APPENDICES

A. COMMON CP/M ERROR MESSAGES 305

B. PIP DEVICE NAMES 307

C. FILENAME EXTENSION TYPES 309

INDEX 313

CHAPTER 1. INTRODUCTION TO CP/M AND MP/M

INTRODUCTION

In this chapter, we want to teach you the basics of operating a computer system, as well as to introduce you to CP/M. We assume no prior knowledge of computers in this book.

THE TYPICAL COMPUTER SYSTEM

A computer system consists of hardware and software. The hardware refers to the system's physical devices, such as the central processor, terminal, disk drives, and printer. The software are the instructions as well as the data files that are manipulated by these programs.

ILLUSTRATION 1-1. TYPICAL COMPUTER SYSTEM

THE COMPUTER

The most well-known manufacturers of small computers; Tandy, Apple, and Commodore, generally house the computer itself in a plastic cabinet with a built-in keyboard. Sometimes, a video screen and disk drives are enclosed in the same housing with the computer and keyboard.

The computer's main function is to process data. The computer stores information or data in memory. Memory is measured in kilobytes or K. One K is equal to 1024 bytes of storage space. Typical memory sizes of small computer systems are 16K, 32K, 48K, 64K, 96K and 128K.

The computer has two types of memory; ROM and RAM. ROM (Read-Only Memory) is used to store permanent information such as the operating system and compiler. RAM (Random Access Memory) can be both read from and written onto. RAM is volatile. That is when the computer is turned off, the information being stored in RAM will be lost. If the information being stored in RAM is to be saved, it must be transferred to a storage device such as disk or tape.

All computer memory is based on the word. A word is a single logical unit of information on which the CPU will operate. A word does not always specify the same number of bits. A bit is the smallest means of representing data. A bit may be either 0 or 1--the two binary states.

In an 8 bit system, a word is 8 bits in length. Since there are 8 bits to a byte, a word is one byte in length in an 8 bit system. In a 16 bit system a word is two bytes or 16 bits in length. In a 4 bit system, a word is 4 bits in length. An example of one byte stored in the computer's memory would be: 00001111. All characters, data, and instructions are based upon these groups of 8 bits, or bytes.

THE DISK DRIVES

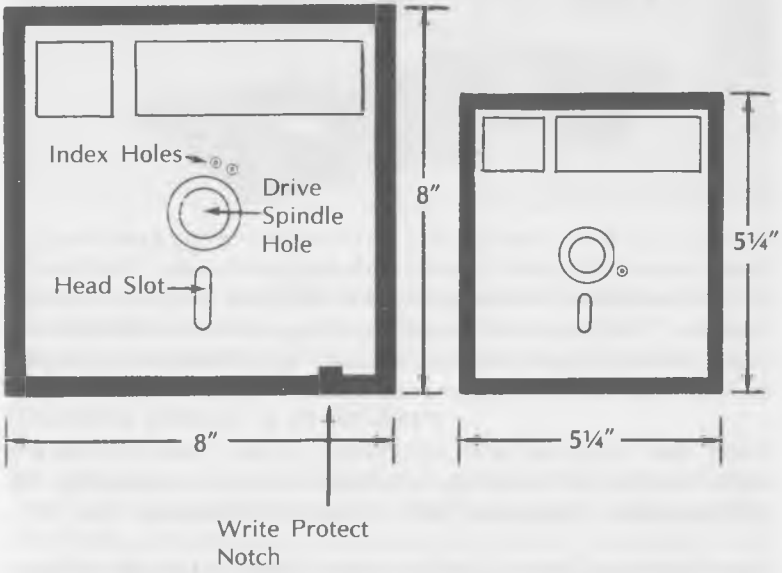
Since the computer's RAM memory is a temporary means of storage, the computer system must have a permanent storage device. Magnetic disk drives are the most commonly used storage devices on small computers, although magnetic tapes

can also be used. For our purposes in studying CP/M, we will assume that the system is equipped with disk drives.

There are two common methods of storing data on disk; hard disk storage and floppy disk storage. Hard disks give the user a large amount of storage and high speed of operation. The main disadvantage of hard disk storage is that hard disk drives are more expensive than floppy disk drives.

Floppy disks are generally used in small computer systems. Floppy disks are available in two standard sizes; 5¼ inch and 8 inch. The 5¼ inch floppy diskette is known as a mini-diskette, while the 8 inch diskette is simply called a diskette.

ILLUSTRATION 1-2. DISKETTE SPECIFICATIONS



Diskettes are fragile and should be handled with care. Do not touch, scratch, or allow dust to settle on the exposed areas of a diskette. Since diskettes are a magnetic storage device, they must be kept away from magnetic fields, as these may erase existing data. Be especially careful to keep diskettes away from telephones, as these often contain magnetic devices.

ILLUSTRATION 1-3. 8" DISKETTE

Each floppy disk consists of a cardboad covering containing a diskette made of mylar coated with magnetic oxide. The hole in the center of the diskette allows the disk drive motor to turn the diskette. The long, oval-shaped opening allows the disk drive's read/write head to come into contact with the diskette surface so that information may be read from or written onto the disk.

Every disk is divided into concentric circles. These circles are called tracks. Each track is in turn divided into sectors by the CP/M operating system. This is shown in Illustration 1-4.

Most diskettes have a write-protect notch. On the 8 inch diskette, this notch is covered with a piece of aluminized paper. When you remove this paper, the write-protect notch is exposed, which means that the disk cannot be written onto.

The operation of the write-protect notch is exactly the opposite with the 5¼ inch minifloppy diskette. The aluminized paper must be removed before one can write on the disk. As long as the paper is present on the diskette, it cannot be written onto.

ILLUSTRATION 1-4. TRACKS & SECTORS

The write-protect notch is used to prevent the operator from inadvertently writing over important information. Master diskettes are generally write-protected and stored in a safe area.

TERMINAL DISPLAY & KEYBOARDS

The terminal display and keyboard are generally built into the same unit within the computer. The keyboard is the means by which the operator can communicate with the computer. The terminal display or video display will display information to the operator via the video monitor.

PRINTER

The printer provides a print-out (hard-copy) of information specified by the operator. The printer is a separate unit connected to the computer itself via connector cables.

SOFTWARE

The hardware is only one component of a computerized system. The software is the other component. The software is the sequence of instructions that when placed in the computer's memory, will direct the actions of the computer.

There are two types of software: system software and applications software. The system software is the operating system of the computer. CP/M is an example of an operating system. The operating system includes the programs that direct the computer in transferring data to and from storage devices such as disk drives and cassette tape units.

The applications software include programs which are designed to perform a specific task such as word processing, accounts receivable, accounts payable, payroll, mailing list, etc.

CP/M & MP/M DEFINED

CP/M is an abbreviation for 'Control Program for Microprocessors', while MP/M stands for 'Multiprogramming Control Program for Microprocessors'.

CP/M is an operating system which allows the user to take full advantage of the hardware. CP/M will read and process data entered via the keyboard, display data on the video monitor, send data to the printer, manage disk space, and manage the allocation of the computer's memory.

The main difference between CP/M and MP/M is that CP/M is designed as an operating system for a single operator while MP/M is a multi-user operating system. In other words, MP/M allows several terminals to be used simultaneously in the computer system. The differences between CP/M and MP/M will be explained later in this book.

CP/M OVERVIEW

CP/M is a group of programs stored on a diskette known as the system diskette. The resident monitor or bootstrap loader which is present in every computer system) will load CP/M from the system diskette whenever the system is properly turned on.

Once the system is turned on, CP/M begins monitoring the keyboard for commands. The operator may then enter the proper command to activate the desired program.

Once the program has been loaded into the computer's RAM, the operator can begin using that program. When the program is ended, the CP/M prompt will again appear, and will await the entry of new commands via the keyboards.

DIFFERENT VERSIONS OF CP/M

Several different versions of CP/M have been released. In this book, we will discuss the features of CP/M up to CP/M 2.2. We will also discuss MP/M.

CHAPTER 2. USING CP/M FOR BUSINESS COMPUTING

Complete data processing systems are now available for just a few thousand dollars, which are just as powerful as systems costing hundreds of thousands of dollars just a few years ago. These dramatic cost reductions have made the computer affordable for almost any small business. In fact, a computer has almost become a necessity now for any business large or small.

There are many models of these new, low-cost business microcomputers. They include the Apple, Commodore CBM, Tandy's TRS-80 line, the IBM Personal Computer, the Xerox 820, and various other manufacturers such as Northstar, Intertec, Cromemco, Dynabyte, Altos, Zenith, Ohio Scientific, Wang, and Hewlitt-Packard.

When choosing a business system, you may wish to consider several or all of these hardware manufacturers to find the hardware configuration best suited to the needs of your business. However, the main consideration in choosing a business system is the applications software. By applications software, we mean the specific programs that will handle the various information processing tasks required by your business. Examples of applications programming functions include accounts receivable, accounts payable, general ledger, payroll, job costing, inventory, word processing, order entry, billing, mailing list, and a host of other tasks.

Since the most important consideration in building a business system is choosing the applications software, many managers make the selection of applications software their first step in building a business system. Then, they choose the hardware and operating system on which the applications software will best operate. This is the preferred method for selecting a computerized system for business applications.

CUSTOM VS PACKAGE BUSINESS PROGRAMS

Once you have identified the specific needs of your business, you must decide what applications software will best meet those needs. The first question you must ask yourself is whether to develop custom applications programs or purchase packaged applications software.

Custom applications software is that which you develop yourself--either by writing the programs or contracting with a programmer to write them for you. A package applications system is a standardized system sold by software houses and designed to fit the needs of a wide range of businesses.

The obvious advantage of custom applications software is that it will be specially designed to fit the needs of your business. The drawback of custom software is that creating a complete, debugged applications system is a long and expensive task that can take years and hundreds of thousands of dollars to develop. For all but the largest business users, this makes custom business programs impractical.

Although packaged programs may not fit the exact needs for your specific business, such programs are generally the only practical solution to a businesses software needs from a cost and time standpoint. It is important for the business user to examine a number of different applications software packages to find the one best suited for his or her business.

One way to gain some of the advantages of customized software with the low cost of a package system is to purchase the system that most closely fits the information processing needs of your business, and then modify or add programs that will adapt the package system so it more closely fits the exact needs of your business. This will increase the initial cost of the applications software, but in many cases, the increased efficiency justifies the additional expense. If a full or part-time programmer is available, additional special purpose programs can be added to most packaged systems that result in these systems fitting the needs of your business more closely.

WSI BUSINESS APPLICATION PACKAGES

Weber Systems Incorporated publishes a series of business applications packages priced from $29.95 to $499.95. These packages are equivalent to those offered by software houses for several times these prices. Because of mass distribution, we have been able to offer complete business applications packages at low prices. These quality packages are within the financial reach of any firm.

The following packages are either currently available or are in the development stage.

WSI Small Business Accounting System
*Accounts Receivables
*Accounts Payables
*General Ledger

WSI Micro Filing System
*Data Base Capacity
*Mailing List Capability

WSI Word Processing System

WSI CompuCalc
*Electronic Spread Sheet Capability

WSI applications packages will run on most microcomputers such as Apple, TRS-80, CP/M based systems, IBM PC's, the Osborne I, and more.

If you wish more information on WSI applications, send a self-addressed stamped enveloped to the address below, and our catalogue will be mailed to you.

Weber Systems Inc.
P.O. Box 413
Gates Mills, OH 44040

Support is available for WSI applications software packages.

BUSINESS SYSTEM HARDWARE REQUIREMENTS

Naturally, business system hardware requirements differ according to the size of the business and the complexity of that businesses applications programs. For our purposes, we will concentrate on the needs of the small to medium sized business rather than large businesses. These firms are more typical users of microcomputers.

The requirements of the central processor impose few limitations on the business user. The speed of the processor is rarely a major factor in choosing hardware for business applications. The size of the memory does limit the choice of hardware somewhat. Most business applications packages require 48 to 64K of RAM.

The most important consideration in choosing hardware for a business system is the disk storage. A disk system must be chosen that allows for sufficient storage of applications programs, data files, and possibly even operating systems and utilities. Both 5¼" and 8" diskettes are widely used for business. Two disk drives is the recommended minimum for most business applications. Generally, one drive is needed to hold the disk containing the applications programs and data files, while the second disk drive is used to make disk back-ups (copies) of the data and program files on the first drive.

Another disk storage system exists for the business user in the form of hard disks. Hard disks are desirable in that they offer much more storage capacity and are faster than floppy disks. Moreover, if several different applications systems are being used by a business, a hard disk system makes it possible to include several or even all of these application programs and their data files on a single disk. Usually, a maximum of one or two applications programs and data files may be stored on a floppy disk. A hard disk may store a large number of applications programs and data files. For most businesses, all applications programs and data files can be stored on one or two hard disks. Therefore, the hard disk system entails less disk handling than the floppy disk system, as there is no need to insert and remove the disk each time a different applications package is used.

Also, a hard disk system allows more data files to be on-line (available on demand to the user). Finally, because of its larger storage capacity, the hard disk system allows more efficient applications programs to be used.

The drawback to hard disk systems is their cost. A hard disk system is signicantly more expensive than a floppy disk system. However, if your business applications require a large amount of data storage or a great deal of file handling, the additional expense of a hard disk system is usually well justified.

A hard disk system requires back-up capabilities just as a floppy disk system. However, in some hard disk systems, the disk is not removable. Just as tape drives are often used as back-ups on large main-frame computers many hard disk systems contain both a fixed disk and a removable disk for making back-ups. On microcomputers with a floppy disk built into the mainframe, such as the Apple III® or TRS-80 Model II®, the built-in floppy disk may be used to make back-up copies of files on the hard disk drive.

One final note before we leave the subject of hard disks. If your system is to have multiple terminals, a hard disk is almost a necessity.

The final consideration when choosing hardware for your business system is the choice of a printer. Two main types of printers may be required depending upon your applications.

--A line printer for speed.
--A daisy wheel printer for quality.

A daisy wheel is preferred for word processing applications, where the nature of the output dictates typewriter quality. The drawback to the daisy wheel printer is that it is generally much slower than a line printer. Since a high speed printer is often needed to produce extensive reports from large data files, a daisy wheel printer is often impractical for printing voluminous reports.

If your business system will be required to produce lengthy reports, a line printer with its high speed may be necessary. If your reports are not overly lengthy, you may be able to use a daisy wheel printer to produce reports, giving you the flexibility of using your computer for word processing applications. Check with your computer dealer for his opinion before making your final decision.

Many firms require both types of printers--a daisy wheel for word processing and a line printer for report writing. However, some businesses find that the line printer outputs acceptable quality for many of their word processing applications. These firms use a line printer for both report writing and word processing applications.

CP/M OPERATING SYSTEM

Once you have decided upon your hardware configuration, you must choose an operating system. The operating system is a series of programs which are designed to provide the user with convenient commands for executing applications programs and handling files.

The minimal requirements for an operating system are that it allows the user to give a name to a program or data file, store that file on disk, display the file on the video screen, and print it on the printer.

Most computer manufacturers have their own individual operating systems. For example, Commodore offers various versions of DOS (Disk Operating System), while TRS-80 computers use various versions of TRSDOS. The drawback here is that an applications program written to run under TRSDOS may not run under DOS and vice versa. In other words, the systems are not compatible.

For this reason, many business users choose to operate under CP/M. CP/M is a standard operating system developed and trademarked by Digital Research, which is available for use on almost any model microcomputer which uses an Intel 8080, 8086, or 8088 compatible microprocessor.

Software houses find it more attractive to write applications systems that run under CP/M than systems that will run on only one model microcomputer. This makes the CP/M operating system an important consideration when planning your system. Obviously, you will find a much larger and more varied selection of programs written for CP/M that those written for an operating system used by only a single manufacturer, such as TRSDOS or DOS. For this reason, many business users of microcomputers choose to install CP/M so as to have a wider range of software from which to choose.

Another advantage of CP/M for the business user is that the applications software he develops or purchases for use on a CP/M system is hardware independent. In other words, the software will run on any hardware that will accept CP/M. Therefore, if the user wishes to switch hardware, he will not lose his investment in applications software.

BUSINESS APPLICATIONS--A MACRO VIEW

Every business must maintain a certain number of files. These files generally involve accounts payable, accounts receivable, personnel, inventory, and general ledger accounts. Additional files often include a customer list, vendor list, asset list, back-order list, as well as a multitude of others. These lists (or files) must be managed. Management can be effected either by hand--such as a bookkeeper--by an electronic device--such as a computer--or as is generally the case, by a combination of both.

These various files used by the business must be processed, maintained, and updated. New data (transactions) must be entered into the file. Changes must be made in existing data (file maintenance). Information must be processed and transferred to other files which are affected by changes in the original files (updates). Finally, reports must be output reflecting all of this activity. Applications programs are designed to automatically perform these file processing tasks.

INDIVIDUAL BUSINESS SYSTEMS

The various tasks of a business can be broken into sub-groups. The most commonly used names for these sub-groups are:

Accounts Receivable System
Accounts Payable System
Inventory System
Personnel/Payroll System
General Ledger System

ACCOUNTS RECEIVABLE SYSTEM

The accounts receivable system keeps track of what is owed to the firm. The major files in an accounts receivable system are the customer file and the invoice file. The customer file usually contains one record for each customer. The customer record generally contains separate fields for the customer's name, address, telephone, account identification number, and total purchases.

The invoice file contains one record for each sale made or credit allowed. Each invoice record contains a field for an invoice identification number, a billing date, an invoice amount, a shipping amount, a sales tax amount, and invoice payments.

Transactions reflecting invoices, debit and credit memos issued and payments received are entered into the accounts receivable system as transactions. These transactions then update the invoice and customer files. Finally, invoices and account statements are printed as well as reports of overdue accounts, summaries of transactions entered, summaries of changes to the invoice and customer files, and summaries of each customer's account activity.

ACCOUNTS PAYABLE SYSTEM

An accounts payable system keeps track of the invoices billed to your business by its vendors--the amounts that your business owes. The major files in an accounts payable system are the vendor file, the invoice file, and the check file. Each vendor record contains data on the vendors from whom your firm makes purchases. Fields in the vendor record generally include an identification number, name, address, phone, and total purchases.

The invoice file contains records for the various invoices, debit and credit memos billed to your firm. Invoice record fields generally include an invoice number, description, purchase order number, date, invoice amount, discount amount, freight amount, and tax amount. For accounts payable systems that are interactive with the general ledger system, the invoice record will contain fields for account numbers and amounts to be posted to the proper general ledger accounts.

The check file contains information on each check issued by your firm to pay its accounts. Fields generally included on a check record include the identification number of the vendor being paid, a check register number, a check amount, and the invoice number or numbers being paid by the check.

Transactions reflecting invoices, debit memos, and credit memos issued to your firm and payments made by your firm are entered into the accounts payable system as transactions. These transactions then update the vendor, invoice, and check files. Finally, checks are printed as well as reports summarizing transactions entered, file update activity, checks written, and vendor account activity.

GENERAL LEDGER SYSTEM

A general ledger system keeps track of the balances of the firm's asset, liability, income, and expense accounts. The primary files in a general ledger system are the account file and the posting files.

The account file contains a record for every account, whether it is an asset, liability, income, or expense account. Each account record generally contains fields for the account number, name, current month total, current quarter total, current year total, first previous quarter total, second previous quarter total, third previous quarter total, and previous year total. In addition, the account record generally contains fields which govern how it will appear when it is printed on the Balance Sheet or Income and Expense reports.

The posting file contains records for every separate posting to a general ledger account. Each posting record generally contains the general ledger account number to which it is to be posted, the posting date, the amount to be posted, and a description of the posting for future reference.

Postings can be entered either directly by the operator, or they may be automatically created by systems which interact with the general ledger system. For example, when an invoice is entered into the accounts receivable system, a posting will be created to credit the sales account (Income) and debit the accounts receivable account (Asset).

A general ledger system usually contains one program to allow the entry of posting transactions with a separate program to update these posting transactions to the permanent records on the account file. Another program is also necessary to add records to or drop records from the account file, as well as to change field data on existing records in the account file. A good general ledger system also has a cash journal program to allow direct posting of transactions affecting the cash accounts to the account file.

PAYROLL/PERSONNEL SYSTEM

The payroll/personnel system calculates employee pay, income

tax deductions, payroll tax deductions, and miscellaneous deductions. Moreover, a history must be kept for every employee of gross pay, all taxes withheld, and all deductions. Finally, a payroll system should have the ability to print payroll checks, a payroll journal, 941A forms, W-2 forms, Insurance Reports, and Employee Absentee Reports.

Generally, a payroll system revolves around an employee master file, an employee history file, tax files, a deduction/miscellaneous pay file, and a transaction file.

The employee master file contains one record for every employee who has been on payroll in a given year. The employee master record fields include an employee number, name, address, social security number, number of exemptions, marital status, pay rate, vacation, and sick hours remaining. The employee master record also contains fields that keep track of the following information for the current pay period, current quarter-to-date pay period, and current year-to-date pay period.

Hours worked
Overtime Hours Worked
Vacation Hours
Miscellaneous Pay
Total Pay
Federal Income Tax Withheld
State Income Tax Withheld
City Income Tax Withheld
FICA Withheld
Deductions Withheld

The employee history file contains one record for every paycheck that was issued during the year. Each record contains fields for storing that pay period's payroll and deduction data.

The deduction/miscellaneous file will contain records of information related to non-recurring deductions and payments to employees.

The transaction file may also be known as the entry file. This file contains records of information describing a single transaction that affects an individual employee's pay. These records are

eventually updated to the employee history and master files.

The tax files contain federal, state, and city tax rates, and salary cut-off points.

INVENTORY SYSTEM

An inventory system should perform the following inventory management functions.

Inventory Maintenance
Purchase Order Entry
Sales Order Entry
Back Ordering
Inventory File (includes quantity, cost, vendor, identification number, and date of sale)
Minimum Quantity Search
Inventory Activity Reports
Inventory Lists (differing reports based on a number of report criteria)

The inventory system generally revolves around the inventory file. The following information is held as a field in the record for every item in the inventory file.

Item Number
Description
Storage Location
Number on Hand
Vendor Number
Filling Price
Purchase Price
Last Sale Date
Minimum Reorder Quantity

MAILING LIST

The mailing list program is perhaps the most underused business system. Many businesses merely use the customer list from the accounts receivable file for their mailing list. In actuality, an efficient mailing list should contain different files, each of which contain prospects as well as actual customers. For example, one

list may contain the purchasers of $1000 worth of spare parts in the past 12 months. Another list might contain all coupon respondents to a trade journal advertisement. A third list might contain your competitor's customers. The possibilities for increasing your firm's profits through the use of a properly managed mailing list system are enormous.

WORD PROCESSING PROGRAM

If you own a microcomputer, you already have the makings of a word processor. You need not invest thousands of dollars in a computer dedicated solely to the word processing task such as those manufactured by Wang or Lanier. By purchasing word processing software, you can convert your microcomputer into a word processor.

A word processor is an applications program designed so that text can be typed on the keyboard, stored on disk,then edited and modified as desired, and finally printed in a specified format. Word processors are now widely used by businesses for typing correspondence, manuals, and even books. Word processing is becoming increasingly important in the publishing and printing industries.

With a word processor, a rough draft is first typed in on the terminal for storage on disk. Next, a printout of the text is output. The author of the text inspects the printout and indicates revisions where necessary. The text can then be edited by the operator on the screen until it has reached its final form. That final form of the text can then be printed out under a number of different formatting options. For example, the text can be left-justified, right-justified, or tabulated.

Word processors are valuable tools for contracts or form letters where the same basic document must be reproduced a number of times with just a few words changed. With a word processor, a perfect printout can be obtained with a minimum of effort.

If you use your business microcomputer for producing lengthy reports as well as word processing, you may need two printers--a faster line printer for report writing and a letter quality daisy wheel printer for documents produced with the word processing software.

A large number of word processors are available that will run on microcomputers equipped with the CP/M operating system. Many of these operating systems offer optional dictionary programs to allow for spelling checks in documents.

CHAPTER 3. BASIC CP/M OPERATION

INTRODUCTION

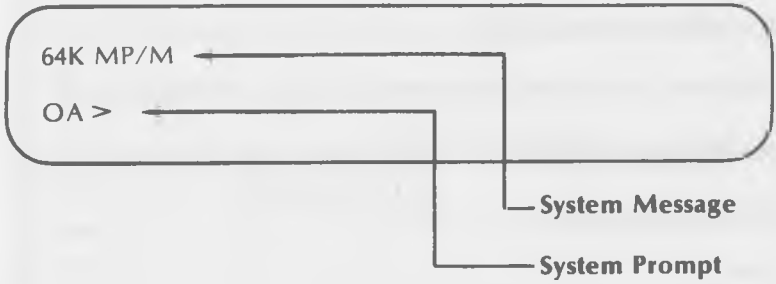
We will assume that you are working with a TRS-80 Model II computer with 64K of RAM, two 8 inch disk drives, and a line printer.

To turn on the Model II, press the On/Off switch so that the computer will be turned on. Then, turn on your printer and disk drives.

Next, insert a copy of your CP/M system diskette in Disk Drive A. Hit the Return key and the following system message and system prompt will appear.

```
64K CP/M

A>
```

With MP/M:



FORMATTING A BLANK DISKETTE

All diskettes used under CP/M must be **formatted.** Formatting is the process of initializing a blank diskette's track and sector information so that it is compatible with the disk controller being used. Different software houses have customized CP/M so that it can be used with different hardware systems. Each version will have a customized formatting utility program. Refer to your specific CP/M system's documentation for details on formatting blank diskettes.

COPYING THE SYSTEM DISKETTE

If you have only one original copy of your system diskette, you should order or make several copies of that original. Store your original in a safe place and use your copies for actual system operation.

To make copies of your system diskette, insert the system diskette in Drive A, and a new blank diskette in Drive B. Then, type in the characters as shown in the following.

```
A>SYSGEN ↵

SYSGEN VER 1.4

SOURCE DRIVE NAME# (OR RETURN TO REBOOT) A ↵

SOURCE ON A, THEN TYPE RETURN ↵

FUNCTION COMPLETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) B ↵

DESTINATION DRIVE B, THEN TYPE RETURN ↵

FUNCTION COMPLETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) ↵

A>PIP B:=A:*.* [V] ↵        } This will copy
(Copying Messages)          } CP/M utilities.

A>
```

When reading the screen displays in this book, the characters underlined are to be typed in by the operator. The symbol, ↵, stands for the Return key, which is a special key on the keyboard which functions much like the carriage return on a typewriter.

In the previous example of SYSGEN, when the system prompt, A>, appears, remove the system diskette copy from Drive B, as well as the original system diskette from Drive A. Label both diskettes. Store the original system diskette in a safe place. Use the copy to operate with--or to make additional copies.

COLD START

In the section titled, 'Turning On the Computer', you performed the bootstrap operation. The terms bootstrap and cold start are synonymous. They both refer to starting up the system. What actually happens is that the computer's resident monitor reads the CP/M system from the system diskette.

SYSTEM PROMPT

A system prompt is a message displayed by the system when it is ready for your next command. For CP/M version 1.4 and earlier versions of CP/M, the system prompt is A>. For CP/M version 2.2 and MP/M, the system prompt is OA>. The A stands for the disk drive in both cases. The zero (0) in MP/M and CP/M 2.2 stands for user area zero. The concept of user areas will be discussed later in this book.

The system prompt always tells you what your current disk drive is. If you wished to switch to Drive B, you could do so by keying in the following command.

A>B:↵

The system prompt would subsequently appear as the following.

B>

If you have more than 2 disk drives, they would be named Drive C, Drive D, etc.

To get back to Drive A from Drive B, you would type in the following:

B>A:↵

The new prompt would appear as follows.

A>

WARM START

A warm start interrupts whatever the computer has been doing and restarts the operating system, giving the system prompt. To perform a warm start, press the Control key (CTRL) and hold it down while you press the C key. This is known as Control C or (↑C).

Control C is most commonly used to stop a program that was chosen in error. After Control C has been entered, the program will be stopped, and control will revert to CP/M.

If the warm start does not bring up the system prompt (A>), see if the disk drive lights are on. If a light is on, and no diskette is in place, try inserting a diskette so that the computer has a disk to read from. If this fails, you will have to perform a cold start. However, be sure to remove all diskettes before turning off the computer's or the disk drive's power.

HARDWARE--CP/M INTERACTION

System diskettes are designed for a particular combination of hardware components. If you change any of the elements of your hardware configuration; the video screen, printer, disk drives, or memory size, chances are that the original system diskette will not work.

DISK DIRECTORY

The system diskette contains a number of different programs. To list these different programs by name, you would type in the Directory command (DIR) as shown below.

```
A>DIR ↵
A:PIP      COM
A:ASM      COM
A:LOAD     COM
A:PROGR    COM
A:STAT     COM
A:ED       COM
A:PROG1    INT
```

Since the prompt is in Drive A and the system diskette is in Drive A, the above screen display example lists the various filenames of the files stored on the disk in Drive A. This list is known as the Disk Directory. Each filename is preceded by 'A:', which indicates that the file being listed is contained in the disk on Drive A.

The letters following 'A:' form the filename. For example, the first file has the name 'PIP' followed by 'COM'.'PIP' is known as the primary name while 'COM' is called the extension. These are combined and separated by a period to form the complete filename, 'PIP.COM'.

The extension specifies the file type. All files with the extension 'COM' are command files (or transient commands). Files with the extension 'BAS' are BASIC source code program files. Files with the extension 'INT' are BASIC intermediate program files (compiled program files).

Data and text files do not require a specific extension, although you may make up your own extensions to help classify your data files.

USING DIR TO FIND A SPECIFIC FILE

You may use the DIR command to find a specific file. To do so, type the DIR command followed by the filename as follows:

```
A>DIR PIP.COM↵

A:PIP COM

A>
```

A space must separate the DIR command and the filename.

If you were in Drive B and wished to list a file in A, you would execute the DIR command as follows:

```
B>DIR A:PIP.COM↵
```

```
B > DIR A:PIP.COM
A:PIP COM
B>
```

If you switch to a disk drive that does not contain a diskette, the system will hang. That is, the disk drive light will come on, and the keyboard will go dead. You will have to perform a cold start to restart the system.

RUNNING A CBASIC APPLICATIONS PROGRAM

We will now discuss how to run a CBASIC applications program under CP/M. For our discussion, we will assume that the program being run was written in CBASIC and was named CHECKWRITER.

CBASIC is one of the most commonly used languages that run under CP/M. CBASIC is a compiled language. Program files written in CBASIC source code must be compiled into intermediate code before they can be executed. Program files written in source code should be specified with the filename extension .BAS. Intermediate program files should be specified with the filename extension .INT.

To implement CBASIC you will need at least the following two programs, CBAS2.COM and CRUN2.COM. CBAS2.COM is the CBASIC compiler. CBAS2.COM would be used to compile the source code program (CHECKWRITER.BAS in our example) into the intermediate code program (CHECKWRITER.INT). CRUN2.COM is the CBASIC run-time monitor, which must be used to actually execute the intermediate program file.

Returning to our example of running CHECKWRITER.BAS, you would first make certain that the diskette you were using contained a copy of CP/M, CBAS2.COM, CRUN2.COM, and CHECKWRITER.BAS. You can use PIP to copy the various programs needed onto a single diskette.

Once you have the necessary files copied onto your diskette, place that diskette in drive A of your computer. After the prompt appears enter the following command.

```
A>CBAS2 CHECKWRITER.BAS
```

The preceding command will compile CHECKWRITER.BAS, and will output the intermediate file CHECKWRITER.INT.

Once the program has been compiled, it is ready to be run via the run-time monitor CRUN2.COM. The following command will execute CHECKWRITER.INT.

```
A>CRUN2 CHECKWRITER
```

CBASIC is a compiled language. CP/M also supports other languages which are interpreted languages. In an interpreted language, the program executes directly from the source code. No compilation is necessary as with compiled language.

The advantage to an interpreted language is that the compilation step is eliminated. The disadvantage is that execution of an intepreted source code program is generally slower than a compiled intermediate file.

USING ED TO CREATE A FILE

CP/M includes an editor program, ED or ED.COM, that can be used to create a file. ED is contained in the System Diskette. You can enter the DIR command to check to be certain that ED is listed.

To execute the ED command, type 'ED' followed by a space, and then the name of your new file. If your new file was to be named ACCT.DAT, you would type in the following.

```
A>ED ACCT.DAT
```

The following messages would be displayed on the screen after the ED command was entered.

```
NEW FILE
*
```

The asterisk is the editor's prompt. It signifies that the ED program is up and ready for your next ED command. The various ED commands will be described in detail in Chapter 7. We will introduce them briefly here.

First of all, the I command is used to insert new characters into your file. Enter 'I' after the ED prompt (*). Then, press the Return key.

Next, type the new data that you wish inserted into the file. Again, end by pressing Return.

When you have finished inserting text, simultaneously press the Control and the Z keys (↑Z). This will end the I command giving you the ED prompt (*).

The following is an example of the use of the 'I' command.

```
*I
ADD THIS TO NEWFILE, ACCT.DAT.
↑Z
*
```

Now that you have entered data into the file, ACCT.DAT, you may wish to display what was entered. You will use the B, T, or #T commands to do so. The B command takes you to the beginning of a file. The T command displays a line of text. The #T command displays the entire text.

The following commands will display the contents of ACCT.DAT.

```
*B ——To go to beginning of file.
*#T —— To display the file contents.
*ADD THIS TO NEWFILE, ACCT.DAT
*
```

Now that you have entered data into the file and displayed that data, you can save the file by using the E command as follows:

```
*E
A>
```

The E command will save your file and conclude the ED program. The operating system will take over and the file will be saved in whatever disk the system was in, A or B. In our example, the system was in Drive A(A > ED ACCT.DAT) when the file was created, so the file will be stored on Drive A. Note that the system prompt appears after the file has been saved.

Remember, if you do not save the file on disk by executing the E command, the file will be lost from the computer's memory when you turn the system off or perform a warm start.

RENAMING A FILE

The REN (Rename) command is used to change a file's name. It takes the following form.

REN New name = Old name

For example, if you wished to rename ACCT.DAT as NAME.DAT, you would use the REN command as follows:

A>REN NAME.DAT = ACCT.DAT ↵

LOADING A NEW DISKETTE

When a diskette is inserted into a drive, CP/M automatically reads the directory into the computer's memory and logs the diskette. If you change diskettes and attempt to write on the new diskette, CP/M will not allow you to write on the new diskette until a warm start has been performed. This feature is provided to prevent you from acidentally erasing data on your diskettes.

Therefore, when you change diskettes, you must enable CP/M to write on the new diskette by performing a warm boot.

For example, suppose that your System Diskette was in Drive A and you had just inserted a new diskette in Drive B, which had been previously used to make a copy of the System Diskette. Press ↑C to perform a warm start.

Before the warm start, the map for the previous diskette was still in the computer's memory. After the warm start, the map is

changed so that the computer can write information to the new diskette.

If you are only reading from the new diskette, a warm start is not necessary. A warm start is only required when you will be writing to the new diskette.

If you are using MP/M, you may not change diskettes or insert a new diskette unless you perform a disk reset. We will describe the procedure for disk reset later.

After you have performed the warm boot, the system prompt will appear (A>), allowing you to access the files in either drive.

COPYING A FILE

Whenever you obtain or create a new program or data file, you should make a copy of it immediately. Store your master diskette in a safe place and use your copies for day to day operation.

Also, when you are updating a file, you should make a copy of that updated file before leaving the system. Be sure to label the copy and record its date.

The PIP command is used in CP/M and MP/M to copy files. PIP is an abbreviation for Peripheral Interchange Program. PIP is a program written in machine language that resides on the System Diskette.

Let's assume that the System Diskette is on Drive A and a data file, ACCT.DAT, is on Drive B. The following prompt will appear:

```
A>
```

You execute PIP by typing it in as a command as illustrated below.

```
A>PIP
*
```

The asterisk is PIP's program prompt and tells the user that PIP is executing.

Your next step is to type in the PIP expression that will perform the desired copying. Such PIP expressions take the following form.

d:copyname = d:originalname

In the above expression, d stands for the letter of the disk drive. Copyname and original name respectively stand for the names of the file to be created as the copy, and the actual existing file from which the copy is to be made.

For example, the following example:

*A:ACCT.BAK = B:ACCT.DAT ↵

tells PIP to make a copy of ACCT.DAT on Drive B and to name this copy ACCT.BAK, placing it on Drive A.

After the copy operation has finished, the PIP prompt will appear as follows.

*

If you press the Return key, you will exit PIP, giving control back to the operating system.

*↵

A>

Regardless of which drive was copied to or from, PIP will always return to the drive from which you originally executed the command. Since PIP was originally executed from Drive A in our example, that is the drive to which it will return.

The disk drive lights will go on and off as PIP is copying a file. That is due to the fact that PIP copies the files in segments called blocks. If the file is lengthy, PIP must keep going back to the original file to get more blocks. This is the reason why the disk drive lights will blink on and off.

Returning to our example, we have a copy of the file named ACCT.DAT which was renamed ACCT.BAK on Drive A. Suppose

we now wish to make a copy of ACCT.BAK on Drive B. We would use the following command to do this.

A>PIP B:ACCT.BAK = A:ACCT.BAK↵

Notice that we used the same name for the new file as was used for the original file. When using the same filename for both the copy and the original, it is not necessary to name the copy's filename in your command. The following command would then be the equivalent of the command given earlier.

A > PIP B: = A: ACCT.BAK↵

Remember, you can not have two files on the same diskette with the same name. If you attempted to copy ACCT.BAK without specifying a new drive or new name, the following statement would appear.

INVALID FORMAT (error)

In actual usage, the expression causing the error will replace the (error) in the above statement.

If an error does occur, press the Rubout or Delete key. This will clear the error specified by the Invalid Format statement.

COPYING AN ENTIRE DISKETTE

We just covered how to copy a file from one drive to another using the PIP command. Most copying operations involve copying an entire diskette. In these cases, the user wishes to make a back-up copy of every file on the diskette.

To copy an entire diskette with just one command, you need to use a device known as a filename match. The filename match is an expression which instructs the computer to perform the instruction specified on any file that matches it.

The most convenient expression to use for a filename match that will match all files is *.*. The symbol * will match any name in its field. Therefore, *.* will match any filename, because all filenames consist of two field names separated by a period.

The filename match, *.*, will match all of the following.

ACCT.BAK
FILE.TXT
NAMES.COM
PROGRAM.BAS

Of course, there is always an exception to the rule, and this is it. In CP/M 2.2 and later versions and MP/M, the filename match *.* will only match those files in the current user area. This will become clear when we discuss the concept of current user areas in Chapter 5.

When copying an entire diskette, you should use a blank formatted diskette to make your copy. If the diskette on which the copy is to be made already has files on it there may not be enough room for all the files to be written. Also, a filename may already exist on the diskette to be copied to with the same name as a file on the diskette being copied from. In such a case, the copy operation will be stopped because of the rule, "Two files cannot exist on the same diskette with the same name."

The following command would be used to copy all files on Drive A onto Drive B.

A>PIP B: = A:*.*

This command will search Drive A for any files that match *.*. It will then copy these files onto Drive B, giving them identical filenames.

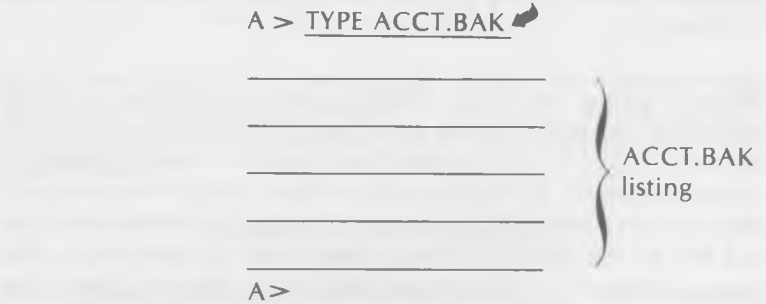
When using the PIP command, only files on other diskettes will be copied. The CP/M system itself is a special program not stored as a file. The CP/M system is stored on two reserved tracks on the diskette. If CP/M is to be copied, the SYSGEN command must be used to do so. We will discuss SYSGEN in detail in Chapter 4.

PRINTING FILES

You are now ready to learn to send a copy of a file to the printer. There are two different methods of doing so in CP/M.

One method is to use the CTRL and P keys along with the TYPE command. First of all, press the CTRL (Control) key and the P key (↑ P) simultaneously. Now, whatever you type at the keyboard will be displayed on the screen as well as printed by the printer. Before using Control P, be sure that your printer is turned on.

After having entered Control P (↑ P), you can now enter a TYPE command at the keyboard to list a file on the printer. This is shown below.



The file ACCT.BAK will be displayed on your screen and simultaneously printed by your line printer when the above command is entered while Control P is active.

If you press Control P again, you will turn off the printer. If you use the TYPE command when Control P is turned off, the file will only be displayed on the video screen. It will not be listed on the printer.

The second method of printing files uses the LPT: device name. This is generally more efficient than using Control P with the TYPE command, especially if you wish to print several files.

An example of the use of the LPT: is given below.

```
A > PIP LPT: = ACCT.BAK
```

This command sends the file ACCT.BAK to the listing device, in our case the printer.

You can also use a filename match to print several or all files as illustrated below.

```
A>PIP LPT: = *.*
```

ERASING FILES

Suppose that you had the file ACCT. BAK on both Drive A and Drive B, and you wished to erase the file on Drive A.

First of all, before erasing any files, check your directory to see which files you have on the diskette in Drive A.

```
A > DIR
ACCT BAK
ACCT DAT
A>
```

Now, use the DIR command to list the files on the diskette on Drive B.

```
A > B:
B > DIR
ACCT BAK
ACCT DAT
B>
```

You can now issue an ERA command that will erase ACCT.BAK in Drive A as illustrated below.

```
B > ERA A:ACCT.BAK
```

If you wish to erase several or all the files on a diskette, you can do so by using a filename match as illustrated below.

```
B > ERA ACCT.*
```

This will erase all files on Drive B with filename ACCT regardless of what that file's filename extension is.

TURNING OFF THE SYSTEM

Before turning off the power to your computer and/or disk drives, be sure to remove all diskettes from their drives. If you turn off the system with any diskettes still in their drives, they may be erased.

You should also make back-up copies of any files that were created or updated before turning off your system.

CONCLUSION

You have now learned the basic terminology of computer hardware and software, how to turn a system on and off, how to start CP/M, how to use the basic CP/M commands such as ERA, PIP, DIR, REN, and ED, and how to use special functions such as Control-C and Control-P. You have also learned how to list files on your display and/or printer, as well as how to make back-up copies of files.

Believe it or not, you now know enough about CP/M to run most applications programs running under CP/M. However, if you wish to learn more about CP/M and its capabilities, continue on.

CHAPTER 4. CP/M & MP/M COMMANDS

INTRODUCTION

This chapter will cover the CP/M built-in commands including; DIR, REN, ERA, SAVE, and TYPE, as well as the CP/M transient commands such as SYSGEN, PIP, ED, STAT, ASM, LOAD, DUMP, DDT, SUBMIT, and MOVCPM. Each command as well as its practical use will be discussed in detail. The control characters will also be covered.

This chapter will give you all the information you need to use the CP/M commands. After you have studied this chapter and have become familiar with these features of CP/M, you can use Chapter 9 as a quick reference guide for the various CP/M and MP/M commands.

The descriptions of commands in this chapter are based on their usage in CP/M 1.4. These same commands are used with modifications in CP/M 2.2 and MP/M 1.0. These modifications will be discussed in the next chapter. Also, CP/M 2.2 and MP/M contain additional commands not used in CP/M 1.4. These will also be discussed in the next chapter.

Even if you are a user of CP/M 2.2 or MP/M, you should read this chapter to gain a basic understanding of the CP/M commands before proceeding to Chapter 5.

COMMAND FORMAT

In the remainder of this book, we will use a pre-defined format for describing commands. Each command will be printed in uppercase letters. The arguments which must be supplied with each command are given in lower case letters. If two or more arguments are enclosed in parentheses, then you can choose between the two. The symbol ↵ will stand for the Return key. The symbol ↑ will stand for holding down the Control key while another key is being pressed.

OVERVIEW

CP/M contains a number of control characters which will be discussed in the next section. These are listed in Table 4-1. CP/M also contains at least 5 built-in commands including:

TYPE
DIR
REN
ERA
SAVE

Finally, the following transient commands may also be executed under CP/M when they are present on the System Diskette.

SYSGEN
PIP
ED
LOAD
ASM
DUMP
DDT
SUBMIT
MOVCPM
STAT

These commands are listed in Tables 4-2, 4-3, and 4-4.

TABLE 4-1. CONTROL CHARACTERS

CONTROL KEY	FUNCTION	RESULT
CONTROL-C	Restart system. Allows writing on a newly inserted diskette.	Restarts CP/M. Allows you to write on newly installed diskette.
CONTROL-C**	Stops an MP/M program while running.	Restarts MP/M and aborts the program running.
CONTROL-D**	Used to detach a running program from a terminal.	Returns control to MP/M.
CONTROL-E	Used to type a command longer than the video display's line length.	The cursor will move to the next line without executing the command.
CONTROL-M or RETURN	Ends PIP (copy) program.	Ends PIP returning control to CP/M or MP/M.

**MP/M only

TABLE 4-1 (CONT). CONTROL CHARACTERS

CONTROL KEY	FUNCTION	RESULT
CONTROL-P	When first pressed, sends everything to be typed and displayed at the terminal to the printer as well. When pressed a second time, it stops sending data to the printer.	On/Off switch for outputting on the printer what is input at the keyboard and displayed on the video screen.
CONTROL-R	The current line will be retyped.	Used to 'clean up' a line after corrections were made.
CONTROL-S	Stops and restarts display.	Stops display so it can be read when first pressed.
CONTROL-U**	Erases entire line.	Cursor moves to the next line. Displays # as a signal to begin a new command.

**MP/M only

TABLE 4-1 (CONT.). CONTROL CHARACTERS

CONTROL KEY	FUNCTION	RESULT
CONTROL-X*	Erases entire line.	Cursor moves back to beginning of the line and erases the previous line.
DELETE (RUB-OUT) CONTROL-H*	Erases the last character typed.	Cursor will repeat the character being deleted.
RETURN, CONTROL-M, CONTROL-J (line feed)	Executes the command line.	The current command is executed.
E followed by RETURN	Ends ED (editor).	Saves text in source file and edit buffer.
G0 followed by RETURN	Ends DDT (debugger).	Returns control to CP/M or MP/M.

*CP/M 2.2 and MP/M only

TABLE 4-2. CP/M FILE HANDLING COMMANDS

COMMAND	RESULT
DIR { filename / filename match } ↵	Lists all filenames in the directory. DIR alone lists all files on a diskette.
ED filename ↵	Editor program is used to create a file on disk or allows modification of the contents of an existing file.
ERA { filename / filename match } ↵	Erases file.
PIP new filename = old filename ↵	Copies a file and gives it a name.
PIP d:new filename = d:old filename ↵	Copies from one drive to another.

TABLE 4-2 (CONT.). FILE HANDLING COMMANDS

COMMAND	RESULT
REN new filename = old filename ↵	Renames a file.
STAT { d:filename / d:filename match }	Displays sizes of files and space taken up by them.
STAT d: ↵	Displays free disk space.
STAT d:filename ↵	Displays file attributes.
TYPE filename ↵	Displays contents of the file on the screen. If Control-P is activated, the file will also be sent to the printer.

TABLE 4-3. CP/M DEVICE HANDLING COMMANDS

COMMAND	RESULT
d: ↵	Switch from one drive to another. The d: is the symbol for the new drive (A, B, C, D, . . .).
CONSOLE ↵**	Displays console number.
DSKRESET ↵ **	Allows system operator to change disks.
MPMLDR or SYSGEN ↵ **	Used in MP/M only to copy or reconfigure MP/M.
MOVCPM ↵	Creates a different version of the CP/M system.

**MP/M only

TABLE 4-3. (CONT.). CP/M DEVICE HANDLING COMMANDS

COMMAND	RESULT
MPMSTAT ↵ **	Displays run time status.
PIP ↵	Copy from one disk to another.
SPOOL filename ↵	Spools file to the printer.
STAT DEV: ↵ STAT VAL: ↵	Reports current and possible device assignments respectively.
STAT log:=phy: ↵	Assigns the physical device phy: to the logical device log:.
STOPSPLR ↵ **	Stop and delete the queue for spool.
TOD ↵ ** TOD mm/dd/yy hh:mm:ss	Display or set time and date.
USER n ↵	Changes the user area. When used without n--displays current user area.

**MP/M only.

TABLE 4-4. CP/M PROGRAM HANDLING COMMANDS

COMMAND	RESULT
program name	Execute program or transient command.
ASM filename	Creates object file in machine language.
DDT filename RDT filename**	Used to debug a program. RDT is the MP/M debugger.
DUMP filename	Prints the object file.
LOAD filename	Creates a new command file with a .COM filename extension from an object file.
SCHED mm/dd/yy hh:mm	Schedules programs for execution. The mm/dd/yy is the date while the hh:mm is the time.
SUBMIT filename parameter 1,2, . . .	Submits a batch of commands to be executed.

**MP/M only

CONTROL CHARACTERS

The use of control characters is described in Table 4-1. It is important to have a working knowledge of control characters when using CP/M. The most commonly used control characters, Control-C and Control-P were discussed in Chapter 2.

Another commonly used control character is the Delete key (DEL) or Control-H (in CP/M version 2.2 or MP/M). Suppose you began typing as follows.

```
A > PIP B:ACCT:BSK =
```

In the previous example, suppose that you had actually wished to type ACCT:BAK =. You would hit the DEL key three times to erase the last three characters as follows:

```
A > PIP B: ACCT:BSK = = KS
```

You would then finish typing the command. However, do not press Return when the command is finished.

```
A > PIP B:BSK = = KSAK = A:ACCT.DAT
```

By using Control-R, the line will be retyped with corrections incorporated.

```
A > PIP B:ACCT.BAK = A:ACCT.DAT
```

Press the Return key after the command is redisplayed, and the command will be executed.

FILENAMES

Filenames take the following format.

```
NAME.EXT
```

The NAME is the primary filename and .EXT is the filename extension. The primary filename may be up to 8 characters long, while the filename extension must contain 3 characters. Primary filenames and filename extensions may contain letters,

numbers, or special characters. However, the following symbols may not be used in filenames.

?<>.,:; =[]

Filename extensions are used to identify different types of files. Some extensions are required, while others are optional. Optional filename extensions are used merely for the convenience of the operator. The different extension types are described in Table 4-5.

When using filenames as an argument to a command, the entire filename, including its extension must be used except where that file is a transient command. In these cases, the .COM extension need not be typed in.

FILENAME MATCH

When you wish a command to act on several files at once, instead of just a single file, use a filename match rather than the filename itself as the argument of the command. The format for using a filename match is as follows.

PIP (filename match)

A filename match is a group of characters used to specify several files at once. The filename match can consist of letters, digits, a period, or two special symbols, * and ?.

The ? symbol will match any character as long as it is in the same position as the ?. Therefore, the following:

PROG?.BAS

will match each of the following filenames.

PROG1.BAS PROG2.BAS PROG3.BAS

TABLE 4-5. FILENAME EXTENSION TYPES

EXTEN-SION	EXAMPLE	DESCRIPTION
ASM	LOAD.ASM	Required for assembly language source files to be used with the ASM command.
BAK	DATA.BAK	Used to identify a back-up copy of a file.
BAS	PROGRAM1.BAS	Required for BASIC program source files.
COM	PIP.COM	Required command file of a transient command.
HEX	PROGRAM1.HEX	Required for program file in hexadecimal format which is to be LOADed.
INT	PROGRAM1.INT	Required for BASIC intermediate (already compiled) program files.
PRL	PROGRAM1.PRL	Required for MP/M relocatable program files.
PRN	PROGRAM1.PRN	Required for the listing file of an assembly language program.

TABLE 4-5. FILENAME EXTENSION TYPES (CONT.)

EXTEN-SION	EXAMPLE	DESCRIPTION
RSP	SPOOL.RSP	Required for MP/M system program files.
SUB	CHANGE.SUB	A text file with CP/M commands or programs; is to be executed batch style by the SUBMIT command.
$$$	DATA.$$$	A temporary or scratch file to be created and erased by ED and other programs.

The * symbol will match with any number of characters as well as no characters at all. Therefore, *.BAS would match with SAMPLE.BAS, PROGRAM.BAK, or even just .BAS. It would not however, match with SAMPLE.BAK or PROGRAM1.BAK. The filename match *.* will match any filename since all filenames contain a period.

BLANKS

CP/M requires that all commands be followed by at least one blank space. In the following example, the PIP command must be followed by a blank space.

A > PIP B: = A:*.*

Other blanks may be inserted optionally, but at least one blank is required after a command.

BUILT-IN & TRANSIENT COMMANDS

All commands take the form of the following example.

A > ERA PROG1.BAS ↵

This command is telling the computer to erase the file named PROG1.BAS.

All commands are actually assembly language programs. Commands are either built-in or transient. The built-in commands (DIR, ERA, REN, SAVE, TYPE) are built into the CP/M operating system.

Transient commands are also assembly language programs that reside on disk. However, transient commands are not built into the CP/M operating system. They can be copied, moved, or deleted from the diskette.

The following transient commands are generally provided with CP/M.

ASM
ED
DUMP
LOAD
PIP
MOVCPM
STAT
SYSGEN
SUBMIT

BUILT-IN COMMANDS

We will now discuss each of the built-in commands (except SAVE) in detail and give examples of each.

DIR (Directory)

As mentioned earlier, the Directory command is used to list all of the files on a diskette. For example, to list all the files on Drive A, type the following command.

A > DIR ↵

If you wished to list the files on Drive B, you would type the following command.

```
A > DIR B:
```

If you wished to search for a specific file on a drive, you would name that file as the argument of the command. The following command would search for the file ACCT.DAT on Drive A.

```
A > DIR ACCT.DAT
```

You can also use a filename match with the DIR command to list files that match on a disk.

The following command would display all files with the extension .DAT.

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

In CP/M version 2.2, the directory is listed in a four column format:

```
A>DIR
A:PIP    COM:ASM      COM:DUMP  COM:LOAD  COM
A:ED     COM:MOVCPM   COM:DDT   COM:STAT  COM
A:XSUB   COM:COPY     COM:LIST  COM:SYSGEN COM
A:ACCT   DAT:PROG1    BAS:PROG2  BAS:PROG3 BAS
```

TYPE

The TYPE command can be used to display any ASC II file on the video screen. The format of the TYPE command is as follows.

```
TYPE d:filename.extension
```

The d:stands for the disk drive letter.

You can print the file on the printer as well as display it by pressing Control-P before giving the TYPE command. When the file is printed as well as displayed on the video screen, it will be displayed at a much slower pace than if it were printed alone.

The TYPE command provides a quick means of examining a file.

REN (Rename)

The REN command allows you to change the name of a file. The format of the REN command is as follows:

REN new filename = old filename

For example, if you wished to change the name of ACCT1.DAT to ACCT2.DAT, you would give the following command.

A > REN ACCT2.DAT = ACCT1.DAT ↵

ACCT1.DAT, which is on Drive A will be renamed ACCT2.DAT. The file contents will remain as they were.

ERA

The ERA command is used to erase a file. It has the following format.

ERA d:filename

The drive (d:) does not necessarily have to be specified. The following is an example of the ERA command.

A > ERA ACCT.DAT ↵

You can use a filename match to erase more than one file at a time. For example, you could erase all files on the diskette on Drive A with the extension .DAT with the following command.

A > ERA *.DAT ↵

Be very careful when you are using the ERA command. You do not want to inadvertently erase the wrong file.

SAVE

The SAVE command is a complex subject and will be discussed in detail later in this book. The SAVE command is used to store information from the Transient Program Area (TPA).

TRANSIENT COMMANDS

We will describe each of the ten standard CP/M transient commands; PIP, ED, SYSGEN, STAT, ASM, LOAD, DUMP, DDT, SUBMIT, and MOVCPM.

SYSGEN

As we described in Chapter 1, SYSGEN is used to copy the CP/M operating system from one disk onto another. When you first receive your system diskette, you should use it to make a copy on a blank diskette.

Use the PIP transient command to transfer all of the .COM and .SYS files to your new diskette. However, to copy the CP/M operating system stored on the reserved tracks of the diskette, the SYSGEN command must be used.

The SYSGEN program works by bringing the CP/M operating system into the memory of the computer, and then writes the system onto the new diskette.

The following dialogue occurs between the operator and the SYSGEN program.

```
A > SYSGEN ↵
SYSGEN VERSION x.x
1 SOURCE DRIVE NAME (OR RETURN TO SKIP) A ↵
2 SOURCE ON A THEN TYPE RETURN ↵
3 FUNCTION COMPLETE
4 DESTINATION DRIVE NAME (OR RETURN TO REBOOT) B
5 DESTINATION ON B THEN TYPE RETURN ↵
FUNCTION COMPLETE
6 DESTINATION DRIVE NAME (OR RETURN TO REBOOT) ↵
A>
```

*1--if CP/M had already been copied into memory using MOVCPM, you would type RETURN instead of the source drive name.

*2--Before typing RETURN, you should insert the diskette with the CP/M system into the correct drive.

*3--CP/M has been placed into memory and is ready to be written onto the new diskette.

*4--Place the blank diskette into Drive B.

*5--CP/M will be written on the diskette in Drive B after RETURN is pressed.

*6--You can make another copy by typing B. If you wish to finish, press RETURN.

After this series of steps have been finished, the new diskette will contain only CP/M (if it were originally a blank diskette). If you wish to copy the rest of the files on the diskette in Drive A on the diskette in Drive B, you can do so by issuing the following PIP command with a filename match as illustrated below.

```
A > PIP B = A:*.*
```

If you did not wish to copy all of the files using the filename match, you could copy only those files desired by issuing a separate PIP statement for each file to be copied.

After you have copied the CP/M operating system and the .COM files onto your new system diskette copy, you should test that copy before using it in everyday operation.

PIP

PIP has already been introduced and will be explained in detail in Chapter 6.

ED

ED is the editor. This has been introduced and will be explained in detail in Chapter 7.

STAT

STAT is used to display the system status or to change device assignments. The STAT command is also used to display available disk space.

The least complex STAT command is used to display the remaining disk space. The following is an example of a STAT command used in this fashion.

```
A > STAT↵

A:R/W,SPACE:132K
A>
```

R/W stands for read-write. This means that the files on the disk can be read from or written onto. In other words, the operator can create new files or delete old ones.

R/W is the opposite of R/O. R/O stands for read-only. A diskette designated R/O can only have its files read from. These files can not be written onto. In other words, the disk is write-protected.

Most 8 inch diskettes hold 224K of available file space. In our example above, the 132K indicates that 132K of space is left on the diskette in Drive A available to be written onto. The other 92K have already been written onto.

BYTES & HOW THEY RELATE TO RECORDS

At this point, we will digress from our discussion of the STAT command and explain the concept of bytes and how they relate to records. A byte is an 8 bit location in memory. A bit is the smallest unit of memory. A bit may either equal zero (0) or 1.

In CP/M, 128 bytes form a record, which is the size of one sector on most disk. Eight records in turn make up 1028 bytes or 1K.

Files exist on diskette as 16K sections known as extents. These extents do not necessarily lie directly next to each other on the diskette. Each extent contains the starting address of the next extent. This is know as 'chaining'.

You need not be concerned with locating files or allocating space for them. The CP/M system does this automatically through its Basic Disk Operating System (BDOS).

The allocation of space is dynamic. That is, the system will allocate additional space for your data or program file as you add records to it. You never need to specify a maximum program length.

STAT (CONT.)

STAT can also be used to display a list of possible device assignments for the logical names CON:, RDR:, PUN:, and LST:. You would type in the following command to list device assignments for these logical names.

A > STAT VAL:↵

The system would respond with a display similar to the one listed below.

```
A > STAT VAL:↵
CON:    =    CRT:    BAT:    TTY:    UC1:
RDR:    =    PTR:    TTY:    UR1:    UR2:
PUN:    =    PTP:    TTY:    UP1:    UP2:
LST:    =    TTY:    CRT:    LPT:    UL1:
```

Notice that a list of four different physical names is given for each device. These names may vary depending upon your system.

The actual device assignments may also be displaying by using the STAT command as follows:

A > STAT DEV:↵

The actual device assignments will be displayed on the video screen as follows.

```
A > STAT DEV:

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

According to the above display, the CON: device is the CRT (Cathode Ray Tube) or the video screen. The RDR: device is a user defined reader. The PUN: device is a paper tape punch device. The LST: device is a line printer.

You may use the STAT command as follows to modify the physical device names.

STAT logical name = physical name: ...

The logical name is the logical device name (CON:, RDR:, PUN: or LST:). The physical name is the physical device name (in our case LPT, CRT, UR1, etc.).

The following STAT command would be used to change the LST: device from the line printer to the CRT. All copy operations would now go to the CRT, rather than to the line printer.

The CON: device must be a device that can both send and receive data (such as the video display). The RDR: device must be able to send data to the computer (an input device). The PUN: and LST: devices must be able to receive data from the computer (an output device).

STAT IN CP/M VERSION 1.4

When the STAT command is used in CP/M version 1.4, the information displayed will be as follows.

```
A > STAT ↵

BYTES REMAINING ON A:144K

A:R/O

A>
```

As we mentioned earlier, R/O stands for read-only. In our example, the diskette in Drive A can not be written upon. If you do attempt to write upon this diskette, the following error message will appear.

BDOS ERR ON B:READ ONLY

If you get this message, you can reset the disk to R/W (read-write) by hitting any key on the keyboard.

To set a diskette in Drive A to R/O (read-only), type in the following command.

A > STAT A:R/O ↵

If you wish to display the size of a file or of several files on the video screen, you could do so by using the STAT command in the format outlined below.

STAT d: { filename or filename match }

Again d: stands for the letter of the disk drive. If the file is in the drive that is currently being specified by CP/M, the drive letter may be omitted.

If a filename match is used rather than a single filename, the STAT command will match all files against the filename match and display them alphabetically on the video screen. The

following is an example of how to use the filename match with the STAT command.

```
A >STAT B:*.DAT

 RECS    BYTES    EXT    D:FILENAME.TYP
   8       1K      1        B:ACCT1.DAT
   8       1K      1        B:ACCT2.DAT
  16       2K      1        B:ACCT.DAT
```

The RECS field informs the operator how many 128 byte records were assigned to the file. The BYTES field shows how many kilobytes were assigned to a particular file. BYTES can be calculated by multiplying 128 (bytes per record) times the number in RECS and dividing the total by 1024 (1K is equal to 1024 bytes).

The EXT field gives the number of 16K extents which are allocated to the file.

As shown in our example, the smallest amount of space that can be allocated by CP/M is 1K for each file. On standard diskettes, each record has 128 bytes. Therefore, CP/M always allows a minimum of 8 records when allocating disk space.

If you are using double density diskettes, each record is allocated 256 bytes. The smallest amount of space that can be allocated by CP/M for double density diskettes is 2K. Again, CP/M allows a minimum of 8 records for each file.

If hard disks are being used, CP/M will allow a minimum of 4K for each file. This is true even if the file uses only a small portion of that space.

STAT IN CP/M 2.0 & 2.2

The use of the STAT command in CP/M versions 2.0 and 2.2 includes several enhancements. If you use the STAT command with a filename or filename match, you will get the same video display as shown in CP/M version 1.4. This is shown in the following example.

```
B > STAT *.*
RECS    BYTES     EXT    ACC

  25      3K       1     R/W B:ACCT1.DAT
  16      2K       1     R/W B:ACCT2.DAT
   8      1K       1     R/W B:ACCT3.DAT
```

If the STAT VAL: command is used under CP/M 2.0 or 2.2, the following type of video display results.

```
A > STAT VAL:

TEMP R/O DISK: d = R/O
SET INDICATOR: d:filename.typ $R/O $R/W $SYS $DIR
DISK STATUS: DSK:d:DSK
USER STATUS: USR:
JOBYTE ASSIGN
CON:          = TTY:    CRT:    BAT:    UC1:
RDR:          = TTY:    PTR:    UR1:    UR2:
PUN:          = TTY:    PTP:    UP1:    UP2:
LST:          = TTY:    CRT:    LPT:    UL1:
```

The last four lines in the above layout are the same as the layout for the STAT VAL command used in CP/M version 1.4.

The first five lines are used to show what STAT commands can be used and what they will do. For instance, if you wished to set an entire disk as read-only, you would use the following STAT command.

```
A > STAT A: = R/O
```

This command would set the entire disk on Drive A as read-only.

To set the R/O$, R/W$, SYS$, or $DIR status for a file, use the following format:

```
STAT d: = R/O
```

To display the status of the current disk, you can use the STAT DSK command. To display the status of a disk on a non-current drive, you would use the drive identifier (d:) with the STAT DSK command as illustrated below.

```
STAT d:DSK:
```

In CP/M 2.2 and MP/M, the STAT USR: command can be used to display current user areas. The usage of this command will be explained in detail later, after we discuss the concept of current user areas.

USING $S ARGUMENT IN CP/M VERSION 2.2

CP/M version 2.2 allows you to use an optional field, $S, with STAT commands. This optional field causes the size of the file to be displayed. This command takes the following format.

```
STAT d: { filename       } $S
        { filename match }
```

For example, if you are using the STAT command, you might add the $S optional field as follows.

```
A > STAT ACCT1.DAT $S
```

```
SIZE    RECS    BYTES    EXT    ACC
60      60      12K      1      R/W A:ACCT.DAT
```

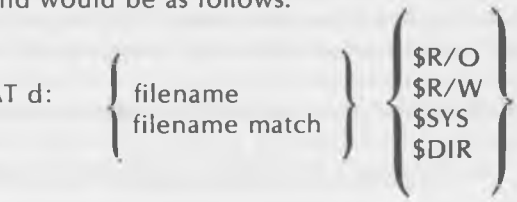
The Size field tells you how many records have been allocated to the file.

The Recs field gives the number of records in every 16K extent.

The Bytes field gives the actual number of bytes allocated for a file. If you are working with a random access file, this is the only figure that actually gives you the space allocated for that file. Since a randomly accessed file might have records counted in the Recs field that are not actually being used, the Recs field may not give an accurate picture of the size of the file. The Size field should be used to determine the logical number of records in a file.

The Ext field gives the number of extents (16K blocks) allocated to a file. The Acc field tells what type of access is allowed to a file, R/O or R/W.

A file's attributes can be changed by including the attribute desired as an argument to the STAT command. The format of such a command would be as follows.



An example of the use of the STAT command to change the status of the file ACCT1.DAT to R/W would be as follows.

```
A > STAT ACCT1.DAT $R/W
```

SUBMIT COMMAND

The SUBMIT command allows you to execute a series of CP/M commands as if they were instructions in a program.

Suppose a certain series of commands were used over and over by the operator. Rather than executing each individual command in this series over and over, it would be far easier to execute the entire series as one single command. The SUBMIT command allows you to do this.

The SUBMIT command searches for a file with an extension of 'SUB'. This file contains actual command lines. Each command line includes arguments that are to be replaced by actual values, when the command is executed.

This .SUB file is created like any other text file by using the ED program.

For example, suppose the file, EXAMPLE.SUB, contained the following lines.

```
DIR $1:$2
PIP A: = $1:$2
```

In our example, $1 and $2 are the arguments. They are used much like variables are used in a BASIC program. When the SUBMIT command is used, these arguments will be replaced by the actual values given in the SUBMIT command.

In our example, the argument $1 will be replaced by a letter which will indicate the disk drive. The argument $2 will be replaced by a filename (including its extension).

Suppose that the following SUBMIT command was executed.

A > SUBMIT EXAMPLE.SUB B ACCT1.DAT ↵

When the command is executed under CP/M, first of all the system will search for the EXAMPLE.SUB file. When the file is found, the commands will be executed. To execute the DIR command, the value 'B' will be substituted for the argument $1, and ACCT1.DAT will be substituted for $2. In other words, the filename ACCT1.DAT will be displayed in the directory for Drive B.

Next, the PIP command will be executed to copy the ACCT1.DAT file onto Drive A, giving the new file the same name.

The SUBMIT command takes the following general form.

SUBMIT filename a1 a2 a3 ...

The filename need not necessarily include the '.SUB' extension as this is automatically supplied by the program. To prevent confusion however, many operators make it a practice to include the '.SUB' extension.

In our general form for SUBMIT, a1, a2, a3, etc. all stand for the value specified to replace the argument. The first value, a1, will replace the argument $1 everywhere in the '.SUB' file; a2 will replace $2; and so on. An argument must always consist of a dollar sign ($) followed by an integer.

If you wish to include a Control key combination in a .SUB file, you must use the Up Arrow key to denote Control rather than the Control key. This is required as generally, the user may not use the Control key when creating .SUB files with the editor.

To perform a SUBMIT operation, a diskette must be inserted into Drive A. You must then reboot the system by pressing ↑ C.

To stop a SUBMIT operation, you must press the RUBOUT (or DELETE) key. The SUBMIT command automatically creates a temporary file $$$. SUB that will hold the commands from the .SUB file. When the SUBMIT operation has been competed, this file ($$$ .SUB) will be deleted. Also, if you press the DELETE key or if an error is detected during the SUBMIT operation, the $$$.SUB file will be erased.

SUB WITH XSUB

In CP/M 1.4, SUBMIT is used to create the temporary file $$$.SUB from the .SUB file supplied. The CCP (Console Command Processor) then executes each line of $$$.SUB as if the user were keying in separate commands. The CCP is the part of the system that reads and executes what is typed in as a command.

In CP/M version 2.2, another program, XSUB, is available as a transient command that allows the user wider use of .SUB files. XSUB allows the user to include commands to programs that use buffered input (other that the CCP). The ED, DDT, and PIP programs all use buffered input.

For example, XSUB allows the user to execute ED commands in a .SUB file as input to the ED program. By using this facility, one SUBMIT operation allows a complex ED operation to be performed separately.

Suppose that INSTRUCT.SUB contained the following lines.

```
XSUB
DIR$1.*
ED $1.$2
#A
B
I
"Note to typesetter: 10 point Oracle with 1.5 lines
leading."
E
A > SUBMIT INSTRUCT DRAFT.TXT
```

In the previous example, the first line of INSTRUCT.SUB is the XSUB command. When SUBMIT is executed, it first tells the CCP to execute XSUB. The XSUB program moves to the area directly below the CCP. It will remain there until the next system reset or cold boot.

As long as XSUB is active, the following message will be displayed above the system prompt.

XSUB ACTIVE

As long as there are commands in the file INSTRUCT.SUB, XSUB will execute them (unless it is interrupted by a warm start).

In our example, XSUB will execute the commands listed in INSTRUCT.SUB. First, DRAFT will replace $1, and the DIR program will display all files that match DRAFT.*.

In the second command, TXT is substituted for $2. XSUB executes the ED program on DRAFT.TXT. The next command (#A) moves DRAFT.TXT into the edit buffer. Then, the B command moves ED's character pointer to the beginning of the buffer. The I command is then used to insert text at the beginning of the buffer, and finally the E command is issued to save the edit buffer as a file and to end the ED program. These ED commands will be discussed later in detail.

When INSTRUCT.SUB has no more commands remaining, the CCP will take over and wait for more commands to be entered at the terminal. However, XSUB will remain active unless it is over written in memory or unless the operator does a cold start or system reset.

As long as XSUB is active, the SUBMIT command can be used to execute other .SUB files. These .SUB files need not include XSUB as a command line if XSUB is already active.

ASSEMBLING (ASM), LOADING (LOAD), & DUMPING (DUMP)

INTRODUCTION

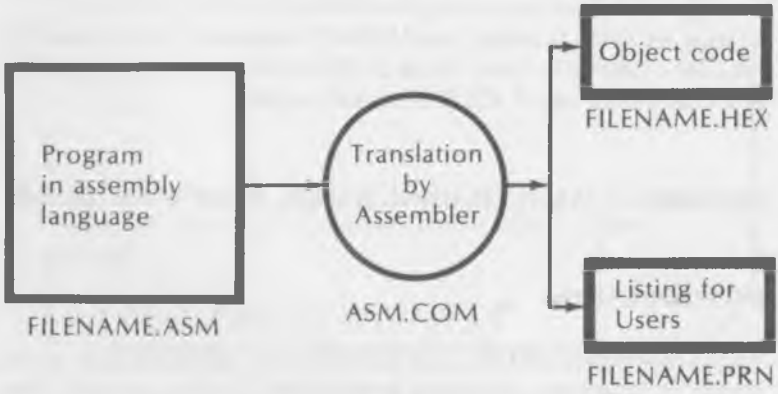
ASM (Assemble), LOAD, and DUMP are operations that allow assembly language programs to work just like a command. The ASM and LOAD commands can be used to turn an assembly language source program into a sort of homemade temporary command.

If you wish to write assembly language programs for a CP/M or MP/M system, you will need to know the assembly language of the computer that you are using. Standard CP/M runs on the Zilog Z80 or Z8000 microprocessors as well as the Intel 8080, 8086, and 8088 microprocessors.

ASSEMBLING (ASM)

The ASM command is used to execute the 8080 Assembler program which is stored in CP/M under the filename, ASM.COM. This assembler program is used to translate an assembly language source file (previously written in 8080 assembly language) into a machine language file. A machine language file is written in binary code, which is the language used by the microprocessor itself. However, when communicating with the operator, either via the video display or the printer, the file will be displayed in hexadecimal form.

ILLUSTRATION 4-1. THE ASSEMBLY PROCESS



You may have other assembly language programs in your CP/M system other than ASM.COM. Several more powerful assemblers are available for the 8080, and other asemblers are also available for the other microprocessors. These other assemblers also end with the extension .COM, and are executed just as ASM.COM is executed.

The source file written in assembly language contains the extension .ASM. When the source file is assembled into machine language, it is given the extension .HEX.

The assembler also creates a printable file with the extension .PRN. This is the file used to print out the program. This .PRN file contains the lines from the original .ASM source code along with error messages and the corresponding machine code in hexadecimal notation.

For example, if an assembly language file named EXAMPLE.ASM was the source file, the assembled machine language file would be named EXAMPLE.HEX and the printable file would be named EXAMPLE.PRN.

EXECUTING ASM

There are two ways to execute ASM. One is by specifying ASM followed by the filename as shown in the example below.

```
A > ASM EXAMPLE
```

The primary name of the source file need only be specified in the ASM command. The extension need not be included, because the file is assumed to have the extension, .ASM. If the file does not have the extension .ASM, the file will not be assembled.

When the above command is executed, the ASM program will assemble the file named EXAMPLE.ASM. The ASM program assumes that the file named is on the current drive. ASM will place the assembled machine language file, EXAMPLE.HEX, and the printable file, EXAMPLE.PRN, on the current drive.

In our example command, the ASM program will assume that EXAMPLE.ASM is on Drive A. It will also place EXAMPLE.HEX and EXAMPLE.PRN on that drive.

An example of the second form of an ASM command is shown below.

```
A > ASM EXAMPLE.ABX
```

Each of the three letters of the filename extension, .ABX, has a special significance.

The first letter indicates the letter of the drive containing the source file. In our example, the source file EXAMPLE.ASM is contained on Drive A.

The second letter indicates which drive is to receive the .HEX file (the machine language file). In our example, Drive B is to receive that file. If the letter Z is used for the second letter, the ASM program will not create a .HEX file--but it will create a .PRN file.

The third letter indicates which disk drive is to receive the .PRN file. If an X is used for the third letter, as in our example, the .PRN file will be sent only to the terminal. If a Z is used as the third letter, ASM will not create the .PRN file, but it will still create the .HEX file

ASM ERROR MESSAGES

Error messages under ASM are generally in assembly language source code. These codes are interpreted in the documentation for the assembler. Errors can be corrected by using the DDT (Debugger) program to modify the program file or by correcting the source file and reassembling.

Some of the errors you may encounter under ASM are listed in Table 4-6.

LOAD

Once you have produced a .HEX file with the ASM program, you can turn that file into a .COM file (transient command) by using the LOAD command to load it into the operating system. This is pictured in illustration 4-2.

ILLUSTRATION 4-2. THE LOADING PROCEDURE

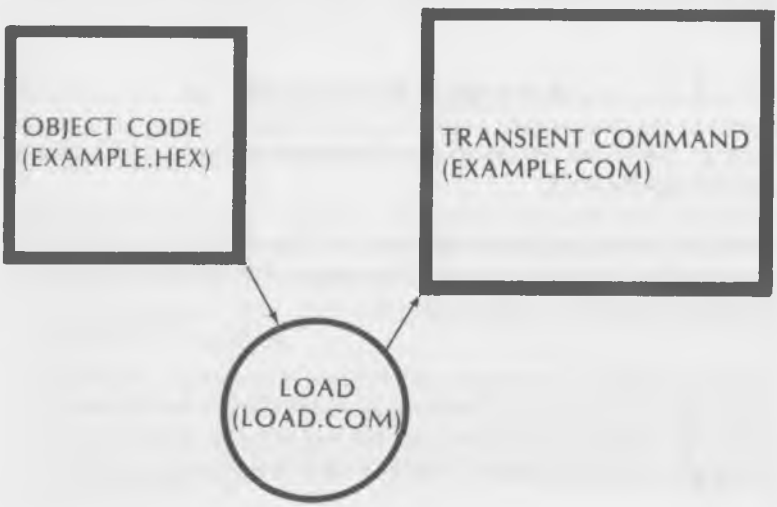


TABLE 4-6. ASM ERROR MESSAGES

MESSAGE	EXPLANATION
SOURCE FILE NAME ERROR	Filename in ASM is improper. Remember, filename matches are not allowed in ASM.
SOURCE FILE READ ERROR	Source file cannot be read by ASM. Use the TYPE command to find the incorrect line in the source program file.
NO SOURCE FILE PRESENT	Usually due to the omission of the .ASM filename extension.
OUTPUT FILE WRITE ERROR	Usually due to filled disk. The .HEX or .PRN output file can not be written.
NO DIRECTORY SPACE	Due to a filled disk directory. Erase files that are not needed.
CANNOT CLOSE FILE	Generally due to attempts to write on a write-protected disk. Output file (.HEX or .PRN) cannot be closed.

To execute the LOAD command, type in LOAD followed by the filename of the .HEX file, as shown below.

```
A > LOAD EXAMPLE↵
```

The .HEX extension need not be included in the filename.

When the above command is input, the LOAD program will search for the EXAMPLE.HEX program on Drive A. When that file is found, a duplicate will be created and named EXAMPLE.COM.

If you wish to load a file not on the current drive, specify the disk drive name as part of the command as shown below.

```
A > LOAD B:NEWEXAM↵
```

The above command will load the file NEWEXAM.HEX from Drive B and create NEWEXAM.COM on Drive A.

DUMP

The DUMP command is used to display the contents of a file on the terminal in hexadecimal form.

All data is actually stored in memory in binary form (base 2). This binary data can be represented in hexadecimal form (base 16). Hexadecimal data is much easier for humans to work with than binary data. In this book, when hexadecimal data is being given, we will end the hexadecimal number with an H. The H represents the fact that the number is in hexadecimal notation. For example, 100H would be the same as 256 in decimal notation.

When using the DUMP command, both the filename and its extension must be given.

```
A > DUMP ACCT1.DAT↵
```

In the above example, the contents of ACCT1.DAT will be displayed in hexadecimal notation at the video screen.

DEBUGGING

Bugs are errors in a computer program. These errors are corrected by using the debugger program, DDT. The DDT.COM program is supplied with a standard CP/M or MP/M operating system.

DDT can either be used to correct errors or to bring a file into main memory so that a copy of it can be saved.

The DDT command takes the following form.

DDT (d:) { filename.HEX / filename.COM }

If the file specified exists on a drive other than the current drive, include that drive's letter as part of the DDT command.

When the DDT command is used, DDT replaces the Console Command Processor (CCP) as the operating system, which will read the command line by line. The Console Command Processor reads and interprets console input and system errors. The CCP is discussed in more detail in Chapter 8.

When the DDT program replaces the CCP program, the DDT program will take over the function of reading the command line.

There are several different commands used with the DDT program. We will discuss the I (Input) command, the R (read) command, and the G0 command (used to stop DDT and return to the operating system).

DDT commands may be used only after DDT has been executed. Once DDT has been executed, a prompt message (a different one for each system) will appear on the terminal video screen followed by the DDT prompt (-). You may type in DDT commands after the DDT prompt has appeared.

The first DDT command we will discuss is the I (input) command. The I command is used to input a file into the Transient Program Area (TPA). This command accomplishes the same thing as if the filename were supplied with the original DDT command. The I command is useful in that you may add a second file to the first one specified in the DDT command.

The R command reads the file in the TPA and displays a load message. This load message takes the following form.

NEXT	PC
xxxH	xx

The number beneath the NEXT column is the address after the program that has been loaded. This number is in hexadecimal form. The PC is the program counter.

The value displayed under NEXT can be used to calculate the number of pages to be used with the SAVE command. This concept will be discussed in the next section.

The G0 command is used to end DDT. Control is returned to the operating system. However, the program is left in the Transient Program Area so that a copy can be saved by using the SAVE command.

SAVE

The SAVE command is used to save one or more pages of the Transient Program Area (TPA) as a file on disk with a filename specified by the operator. One page is the equivalent of 256 bytes of memory.

The SAVE command is used to create an exact image of a file currently in the TPA. An example of a situation where the SAVE command might be used is where the DDT was being used on a program and that program began working correctly. The operator would want to save the program as a file for future use.

The SAVE command takes the following form.

SAVE p filename

The p is a decimal number giving the number of pages to be saved.

CALCULATING THE NUMBER OF PAGES (P) UNDER THE SAVE COMMAND

Many computer programmers find calculating the number of pages to be SAVE'd somewhat confusing. However, the method is actually very simple.

First of all, the R command must be used under the DDT program to display the NEXT field. As we discussed earlier, the figure under the NEXT column is the next higher address following the program in the TPA. In other words, it is the last address of the TPA plus 1. Remember, the number supplied under NEXT is in hexadecimal form.

Since the number under NEXT is the last address plus 1 in the TPA, if you subtract 1 from that figure, you will have the last TPA address--or the end of the program. The trick is that you must subtract in hexadecimal arithmetic. After this subtraction has been completed, you must convert the hexadecimal address into decimal pages.

To make this conversion, follow these steps.

1. Convert the NEXT address from hexadecimal to decimal form.

2. Subtract 1--to convert the NEXT address to the end of program address.

3. Subtract 256--the beginning address of the Transient Program Area (256 to 100H).

4. Divide the remainder by 256.

5. If the result is an integer, use that figure as the number of pages (p). If the result is not an integer, round the result to the next highest integer and use that figure for p.

We will run through an example to make this clear.

Suppose that the value given under NEXT by the R command in DDT was 2900H. If you convert 2900H to decimal, the decimal number address is 10496. Subtracting 1 and then 256 yields 10239. Division by 256 yields 39.99. Rounding to the next highest integer results in 40; which is the figure that should be used for p--the number of pages to be saved.

If you wished to save this portion of memory via the SAVE command, you would use the following command.

A > SAVE 40 EXAMPLE.COM

The data from 100H through and including 28FFH (H signifies hexadecimal) would be saved on the disk on Drive A under the filename EXAMPLE.COM.

CHAPTER 5. MP/M & CP/M 2.2

INTRODUCTION TO MP/M

The MP/M operating system is designed for the execution of several programs simultaneously on the same system. This phenonema is know as timesharing.

Of course, the programs are not actually executed at the exact same time--although to the users, it seems as if they are. The computer executes one program and then immediately goes to another. This switching back and forth occurs so rapidly that each user gets the impression that he or she is the only one using the system.

PROGRAM SCHEDULING

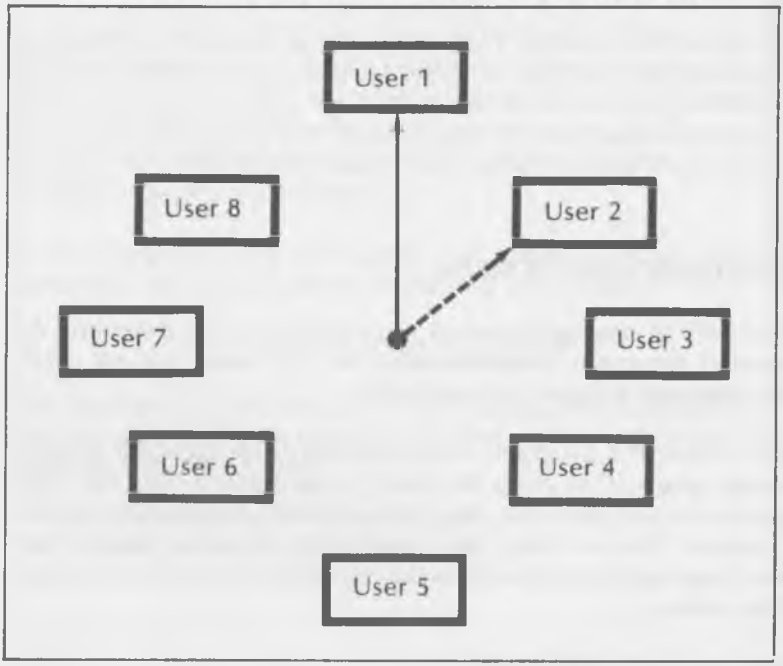
With a timesharing system, each program must be scheduled to run on the processor in a certain order. This is known as program scheduling.

The simplest method of scheduling is round-robin scheduling. This is where every program receives an equal amount of computer time in its turn. Round-robin scheduling is illustrated in Illustration 5-1.

A better method of scheduling than round-robin is a scheduling system where certain more important programs are run before other less important programs. In such a scheduling system, each program is assigned a priority level which determines when it will run.

In setting priorities, processes or programs are usually classified as either I/O functions or CPU functions. I/O functions are

ILLUSTRATION 5-1. ROUND-ROBIN SCHEDULING



programs that perform input/output functions, while CPU functions are programs that perform Central Processor Unit operations.

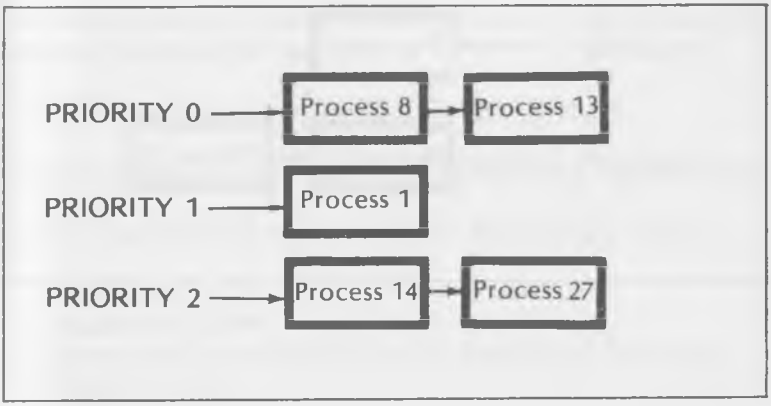
For example, if a program was being executed in the CPU, it would eventually request an input or output operation. An input/output operation takes several milliseconds and is slow compared to a CPU operation, which operates in microseconds.

When an I/O command is issued, the program being executed is usually temporarily blocked. The program is then known as being dormant.

While the program is dormant, the I/O operation will begin. This I/O operation will be undertaken concurrently with a CPU operation. When the I/O operation has been completed, the program will no longer be blocked. It will now become active rather than dormant, and will take its place in the scheduling list.

A multi-layered priority system with three priority levels is shown in illustration 5-2. Notice that the highest priority is zero, while the lowest is two.

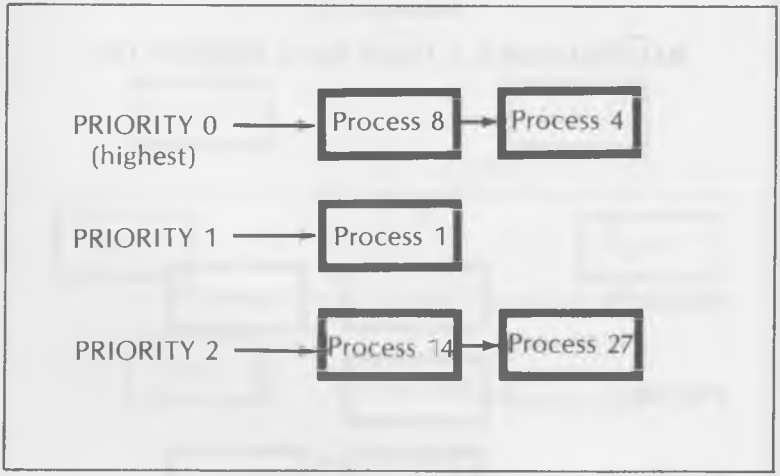
ILLUSTRATION 5-2. THREE LEVEL PRIORITY LIST



In Illustration 5-2, process 8 will be executed next, followed by process 13. When all of the processes at level 0 have been completely executed, the processes at level 1 will be executed, followed by the processes at level 2. In Illustration 5-2, process 1 will follow process 13. Processes 14 and 27 will then follow process 1.

If a new process were entered at a high priority level, that process would interrupt the scheduling. For example, suppose that process 8 had been executed. While process 13 was executing, a new process (4) was entered at priority level 0. The scheduling would now appear in the order shown in Illustration 5-3.

ILLUSTRATION 5-3. REVISED PRIORITY SCHEDULING



Generally, the scheduling systems we have described apply only to the central processor. However, scheduling may also be used for the disk and/or the printer.

Processes for the printer are scheduled on a First-In-First-Out basis. This is known as FIFO. The scheduling program for a FIFO scheduling system is called a spooler.

In the case of the disk, the scheduling program will attempt to make disk access as efficient as possible by reading or writing those processes that require the least movement of the read-write head. Therefore, processes waiting for disk service may be serviced in any order depending upon when the block they require physically comes under the read-write head.

In addition to a scheduling program, a time-sharing system requires the following.

1. Flags and interrupts to synchronize the processes.

2. Protective facilities so that if a single process malfunctions, the other processes will not be damaged.

3. An efficient allocation of memory so that the processes occupy memory space only when necessary.

MP/M provides all of these facilities. However, MP/M does have certain limitations including.

1. MP/M will handle a maximum of eight processes.

2. Each process must have its own fixed memory segment (48K).

3. The files may only be one of the following four types.

   Read/Only (R/O)
   Read/Write (R/W)
   System (SYS)--the file will not be listed in the directory.
   Directory (DIR)

SIMILARITIES BETWEEN CP/M 2.2 & MP/M

CP/M version 2.2 contains several features actually designed for MP/M. One of these features is the USER command and user areas, which are used to segregate your own files from those of other users in the system. Also, CP/M offers a feature whereby file attributes may be assigned by a STAT command to protect your files from accidentally being written over or deleted.

These additional features are necessary for MP/M, because it is a multi-user system. With a multi-user system, several operators may be using the resources of the system at the same time. Therefore, with several operators using the same device (CPU, disk drive, printer, etc.) at the same time, more protection is needed for each user's files.

Besides the two common features shared by both CP/M and MP/M, additional features are found only in MP/M that are useful in a multi-user environment. These include the following.

--SPOOL command which regulates traffic in the line printer.

--CONSOLE command which displays the current console number in a system with multiple consoles.

--GENMOD command which is used to produce a relocatable program.

--DSKRESET command which is used to schedule the changing of disks in a multi-user system.

--TOD command which is used to display or set the system time and date.

--SCHED command which is used to schedule programs to be executed at a later time.

--MPMSTAT command which is used to display system information.

USER AREAS AND THE USER COMMAND

When MP/M is accessed, the following message will appear upon each console.

```
MP/M
nA>
```

The n refers to the user number for that console. Consoles can be numbered from 0 (for the first console) to 15. The maximum number of consoles that can be supported by MP/M is 16. Any user number from 0 to 15 may be assigned to any console by using the USER command.

The A in the previous MP/M prompt refers to the disk drive, Drive A. Just as with CP/M, the drive letter may be changed by the operator according to his wishes.

In CP/M version 2.2 and MP/M, each disk is divided into 16 separate areas. These areas are referred to as user areas. The purpose of user areas is to keep all of each user's files separate. For example, user 1 would keep his files in user area 1, user 2 in user area 2, etc. User area 0 is reserved for the system files.

PUTTING A FILE INTO A USER AREA

For whatever user area you are currently in, the files that you create will be placed in that user area. By the same token, to access a file, you must be in that file's user area before you can gain access to it.

When you first activate CP/M version 2.2 or MP/M, the current user area will be zero. In MP/M, the following prompt appears when the system is activated.

```
0A>
```

The zero (0) stands for user area zero. The A stands for Drive A. Therefore, MP/M reminds you that your current user area is zero when it is activated.

In CP/M version 2.2, you must use the following version of the STAT command to determine the current user area.

```
STAT USR:
```

It is possible to copy a file that is in another user area. To do so, you need to use the G parameter of the PIP command. This is available in MP/M and CP/M 2.2. The G parameter will be explained in the next chapter.

Unless you use the USER command, you will stay in user area zero. If the program and data files that you create under CP/M 2.2 or MP/M are in user area zero, then they will be compatible with the earlier versions of CP/M without separate user areas.

To go to another user area in CP/M 2.2 or MP/M, you must use the USER command as given in the format below.

USER n

The n stands for the user area specified by the operator. If n is supplied between zero and fifteen, the USER command will move to that user area as illustrated below.

```
0A > USER 5
5A >
```

In the example above from MP/M, the initial user area upon start-up is zero. This is indicated by the system prompt 0A > . This system prompt indicates that the current user area is zero and the current disk drive is A.

When you use the USER command to move to another user area, you will stay in that area until another USER command is given or until the system is rebooted (which places the system back to user area zero).

FINAL POINTS ON USER AREAS

User areas are designed to be used with systems containing a large amount of disk storage, usually a hard disk.

For this reason, most sytems using MP/M are equipped with hard disk drives rather than floppy disk drives.

PROTECTING FILES IN MP/M & CP/M 2.2

In CP/M 2.2 and MP/M, when a file is created , it is done so with the $DIR and $R/W file attributes. The $DIR attribute signifies that the file will be listed when the DIR command is used. The R/W attribute signifies that the file can be read from or written onto. Of course, this means that the file can also be erased.

In CP/M version 2.2 and MP/M, you can prevent a program from being listed by a DIR command by changing the $DIR attribute to the $SYS, or system, attribute. The STAT command is used to change this file attribute as shown below.

```
A > STAT PROGRAMA.BAK $SYS
```

This command changes PROGRAMA.BAK into a system file--one that will not be listed by a DIR command.

You can also use the STAT command to change a file from $R/W (read-write) to $R/O (read-only). By changing a file to read-only, you will prevent it from being erased or written onto. The command below illustrates how to use the STAT command to change a file to read-only.

```
A > STAT PROGRAMA.BAK $R/O
```

This command changes PROGRAMA.BAK from read-write to a read-only file.

STAT can also be used to assign the $SYS file attribute. The $SYS file attribute is the opposite of $DIR. The $SYS file attribute can be useful in hiding a file from other users in a multi-user system. When a file has the $SYS file attribute, the DIR command will not display that file. However, complete protection is not afforded as the STAT command can be used to display $SYS as well as $DIR files. However, by using the $SYS file attribute along with the $R/O file attribute, your files will be afforded some protection from other users.

The effect of the DIR and STAT commands on files with the attributes $SYS, $DIR, $R/W, and $R/O is illustrated below.

```
A > DIR PROGRAMA.BAK

NOT FOUND

A > STAT PROGRAMA.*

Size    Recs    Bytes   Ext   Acc
  64      64      8K     1    R/O A:PROGRAMA.TXT
  64      64      8K     1    R/W A:PROGRAM.BAS
  64      64      8K     1    R/O (A:PROGRAMA.BAK)
```

In the above example, PROGRAMA.BAK is not listed by the DIR command, as it was previously assigned the $SYS file attribute by a STAT command.

The STAT command lists all files with the prefix PROGRAMA regardless of whether they have the $SYS or $DIR file attribute. The PROGRAMA.BAK file is listed in parentheses because it has both the $SYS and $R/O file attributes. The other files have the $DIR file attribute. PROGRAMA.TXT and PROGRAMA.BAK are $R/O files, while PROGRAMA.BAS is a R/W file.

Finally, when you are using the PIP command to copy a read-only or system file, the new file created will have the read-write and directory file attributes. If you wish the newly created file to have the $SYS and $R/O file attributes, you must use the STAT command to change these attributes.

MP/M SYSTEM OPERATIONS

An MP/M system with just one user operates just like CP/M version 2.2. Usually, an MP/M system involves several users--each with his own terminal. Each user can operate the system--which will operate just like CP/M version 2.2 from each individual operator's point of view.

However, multiple users place demands on the system which one user will not. These demands include scheduling, spooling files to the printer, setting the system time and date, changing diskettes, and displaying system information. MP/M provides for all of these demands with a set of MP/M commands.

DSKRESET (MP/M COMMAND)

In an MP/M system, it is not advisable for any one user to remove a disk from a drive without first warning the other users of what he is about to do. The DSKRESET command warns other users of an MP/M system that someone else wishes to change a diskette.

The DSKRESET command resides on the system diskette as either a .COM or .PRL (Page Relocatable) file. The DSKRESET command can be executed just by typing in its primary name as shown below.

```
0A > DSKRESET ↵
```

When the DSKRESET command is entered, the following message will appear on every terminal connected to the system.

Confirm reset disk system (Y/N)? Y

For a disk reset to be allowed, each user must key in Y (for yes) in answer to the above prompt.

SPOOL (MP/M COMMAND)

Many times, more than one user will want to send files to the printer, or the same user may wish to send several files at once. In these cases, the user or users will want to line up the file being sent to the printer in a queue or line. This is known as spooling. The queue is also known as the spool queue.

The SPOOL command is used to send files to the spool queue. This command takes the form outlined below.

SPOOL filename (filename, filename, . . .)

Only the first filename is a required entry. The others are optional. The filename must include the extension (if any) as well as the primary filename.

The following is an example of the use of the SPOOL command.

0A > SPOOL FILE1.TXT FILE2.TXT FILE3.TXT ↵

The SPOOL command as used above will send three files, FILE1.TXT, FILE2.TXT, and FILE3.TXT, to the LST device (usually the printer).

The STOPSPLR command can be used to stop a spool operation and empty the spool queue. An example of this command is shown below.

0A > STOPSPLR ↵

SCHED (MP/M COMMAND)

The MP/M operating system has the capability to keep the date and time. You can use this capability to schedule programs to be run at a preset time on a preset date. MP/M will keep track of the system time and date and will run these programs as scheduled.

The SCHED command is used to schedule execution of a program at a specified date and time. The format of the SCHED command is as follows.

SCHED mm/dd/yy hh:mm program name

The date is supplied in mm/dd/yy as the month for mm (00-12); day for dd (01-31); and year for yy. The time is supplied in hh:mm as hours (00-24) and minutes (00-60).

The program being supplied is assumed to have an extension of either .COM or .PRL. It is not necessary to supply this extension, the primary filename alone is sufficient. The following example illustrates the use of the SCHED command.

```
0A > SCHED 01/01/82 01:30 NEWYEAR↵
```

The program NEWYEAR.COM (or NEWYEAR.PRL) will be executed at 1:30 AM on New Years Day of 1982--as long as your system is running, and that date is encountered by the system.

TOD (MP/M COMMAND)

The TOD command is used to display or reset the time of day. To display the system time, use the TOD command as illustrated below.

```
0A > TOD ↵
Sat 04/12/81 02:33:14
```

The illustration below shows the procedure for using the TOD command to reset the system time and day.

```
0A > TOD 04/12/81 02:36:00 ↵
Strike any key to set time
 Sat 04/12/81 02:36:00
0A >
```

When the "Strike any key" message appears, you may set the system time whenever you wish by merely striking a key.

ABORT (MP/M COMMAND)

The Control C command (↑ C) will abort a program that is currently attached to the console. However, it has no effect on programs which are detached.

You can use the ABORT command to abort any program, both those that are attached and those that are detached from the console. The ABORT command can even be used to ABORT programs belonging to another console in the system. An example of the use of the ABORT command is illustrated below.

3A > ABORT PROGRAMA 3 ↵

The above command will abort PROGRAMA scheduled at the console where the command was entered, (assume console 3). The console number is not required in this case, as you are aborting a program on the same console where the command was entered, console 3.

However, if you are aborting a program scheduled on another console, you must specify a console after the program name in the ABORT command. If we suppose user number 3 is still console 3, the following command will abort PROGRAMA at console 4.

3A > ABORT PROGRAMA 4 ↵

ATTACH (MP/M COMMAND)

Control D (↑ D) may be used to detach a program from the console. The program will continue to operate without making use of the console until it is reattached to the console. This command generally is used to free the console for data entry, or the execution of another program. However, for Control D to detach a program, the program itself must check the console for that command.

You can use the ATTACH command to attach a program to a console. A program must always be reattached to the same

console from which it was detached. The use of the ATTACH command is illustrated below.

```
0A > ATTACH PROGA
```

CONSOLE (MP/M COMMAND)

The CONSOLE command allows the user to examine the number of the console being used. Remember, the console number does not always correspond to the user number. The following example illustrates the use of the CONSOLE command.

```
3A > CONSOLE
CONSOLE = 2
```

This example shows that user number 3 is using console number 2.

DIRECTORY (MP/M COMMAND)

The DIR command works in MP/M as it does in CP/M with one extra option, the S option. By including S after the filename, all $SYS files as well as $DIR files will be listed by the DIR command. The use of the S option is illustrated below.

```
3A > DIR PROGRAMA.* S
```

All files with the primary name, PROGRAMA, will be included in the directory listing including those with $SYS file attributes.

ERASE (MP/M COMMAND)

The ERASE command is used in MP/M as it is in CP/M with one additional option, the ERAQ command. ERAQ is a form of erase used in MP/M that allows the user to erase a set of files that match a specific pattern.

The ERAQ command will not erase files with read-only ($R/O) attributes, nor will it erase disks specified as read-only.

Unlike ERASE, ERAQ requests the user to verify each file

before erasing it. An example of the use of the ERAQ command is given below.

```
0A > ERAQ PROGRAMA.*
   A:PROGRAMA.INT ? Y
   A:PROGRAMA.TXT ? Y
```

A Y must be entered by the operator to verify the command, before the file will actually be erased.

TYPE (MP/M COMMAND)

The TYPE command is used in MP/M as in CP/M to type a file. MP/M contains an additional option known as a pause. The pause allows the operator to display a specified number of lines. After those lines are typed, the listing process will pause until the Return key has been pressed. The following command illustrates the use of the TYPE command with the pause option.

```
1B>TYPE PROGRAMA.TXT P15
```

The first fifteen lines of PROGRAMA.TXT will be typed followed by a pause, until the Return key has been pressed.

MPMSTAT (MP/M COMMAND)

MPMSTAT is a special form of the STAT command used in MP/M to display a complete run-time status of MP/M. The form of the command is illustrated below.

```
0A > MPMSTAT
```

An outline of the output of the MPMSTAT command is shown in Illustration 5-4.

A detailed explanation of Illustration 5-4 goes beyond the scope of this chapter. However, a brief description of the MP/M Status Display outline will follow.

Ready Process (es)--a list showing all of the ready processes in order of priority. The process with the highest priority is that which is running.

ILLUSTRATION 5-4. MPMSTAT OUTPUT

```
******MP/M Status Display******

Ready Process(es):
Process(es)    DQing
Process(es)    NQing
Delayed Process(es):
Polling Process(es):
Process(es) Flag Waiting:
Flag(s) Set:
Queue(s):
Process(es) Attached to Consoles:
Process(es) Waiting for Consoles:
Memory Allocation:
```

Process(es) DQing--each queue is shown along with those processes which have executed a read operation on the queue. The processes are listed in order of priority. They are waiting for a message to be written to the queue.

Process(es) NQing--same as DQing except that the processes must wait for a buffer to write a message to the queue.

Delayed Process(es)--lists those processes delayed for a specific length of time.

Polling Process(es)--lists those processes that poll an I/O device waiting for a ready status.

Process(es) Flag Waiting--lists the processes opposite their corresponding flag number.

Flag(s) Set--gives a list of the flags that are set.

Queue(s)--lists the queues in the system. An MX at the beginning of a queue name denotes mutual exclusion. Queues named with upper case letters may be accessed via a console command.

Process(es) Attached To Console--lists the processes with their corresponding console numbers.

Process(es) Waiting For Consoles--lists the processes by console and by priority. These processes have been detached and are waiting for their console in order to again resume execution.

Memory Allocation--displays a memory map which gives the base, size, bank, (if applicable), the resident process, and the console number.

GENMOD (MP/M COMMAND)

GENMOD, as well as GENHEX and PRLCOM, are commands used only by assembly language programmers. The format of GENMOD is as follows.

0A > GENMOD d:FILE1.HEX d:FILE2.PRL $DDDD

FILE1 contains two hexadecimal files which are concatenated and set off from each other by 0100H bytes. The GENMOD command transforms FILE1 into FILE2, which is page relocatable (PRL). PRL is a file extension which is required for MP/M programs which are relocatable.

DDDD$ is an optional parameter that specifies the additional amount of memory required by the program in hexadecimal form.

GENHEX (MP/M COMMAND)

GENHEX is used to transfer a COM file into a HEX file. GENHEX is often used before GENMOD. An example of the use of GENHEX is given below.

0A > GENHEX B:PROGRAMMA.COM

PRLCOM (MP/M COMMAND)

The PRLCOM command is used to transform a PRL file into a COM file. An example of the use of PRLCOM is given below.

0A > PRLCOM B:PROGRAMA.PRL A:PROGRAMA.COM

GENSYS (MP/M COMMAND)

The GENSYS command is used for MP/M system generation. The operator will be prompted for all required parameters and information. After this data has been entered, the MPM.SYS file will be created.

The MPMLDR command is used to load and execute the MPM.SYS file. The dialogue between the user and system is shown in Illustration 5-5. The underlined areas denote user responses.

MPMLDR (MP/M COMMAND)

The MPMLDR command loads the MPM.SYS file that contains the MP/M system. MPMLDR then executes the MPM.SYS file to bring up MP/M. MPMLDR also displays the system parameters. Use of MPMLDR is illustrated as follows.

```
A > MPMLDR ↵

MP/M 1.0 Loader

Number of consoles = 4
Breakpoint RST# = 5
Top of memory = COFFH
Memory Segment Table:

SYSTEM DAT C000H 0100
.
.
.
.
.
.
.
.
.
.
.
```

ILLUSTRATION 5-5. GENSYS DIALOGUE

```
A > GENSYS

        MP/M SYSTEM GENERATION


____ ____ _____ _____ _____ _____ ____ ____ ____

____ ____ _____ _____ _____ _____ ____ ____ ____

Top page of memory = ___ ___
Number of consoles = ___
Breakpoint RST# # = ___
Add system call user stacks (Y/N)? ___
Z80 CPU (Y/N)? ___
Bank switched memory (Y/N)? ___
Memory segment bases (ff terminates list)
___ ___
___ ___
___ ___
___ ___
Select Resident System Processes: (Y/N)
ABORT     ?____
SPOOL     ?____
MPMSTAT   ?____
SCHED     ?____
```

CHAPTER 6. USING THE PIP COMMAND TO HANDLE FILES

INTRODUCTION

We introduced the PIP command in Chapter 3, where we showed how PIP can be used to copy a file. Although PIP's primary purpose is to copy files, it has many more uses.

In this chapter, we will cover the different uses of the PIP command. It is likely that you will not require many of the different usages of PIP. You will probably only require a few of these different usages. Therefore, you may find it useful to briefly skim the areas of this chapter which you are not likely to make use of, and concentrate on those areas you will make use of.

Some of the different usages of PIP are given below.

--PIP can be used to concatenate files.

--PIP can be used to cut off lines that are too long for the screen using the D option.

--PIP can be used to automatically print text in formatted pages with the P option.

--PIP can be used to print formatted text using tabs.

--PIP can be used to print a group of files with just one command by giving a PRN specification.

WHAT IS PIP?

Basically, PIP is a program that transfers files. PIP is an abbreviation for Peripheral Interchange Command. As the name implies, PIP allows files to be transferred between any two peripheral devices.

In Chapter 3, we only discussed transfers from one disk to another. In this chapter, we will cover the additional uses of PIP. We will study in detail PIP's primary function--that of transferring and copying files--as well as the use of PIP for processing files as it transfers them.

COPYING A SINGLE FILE WITH PIP

PIP can be either executed as a single line command or as a program. The example below illustrates the use of PIP as a single line command.

```
A > PIP B:PROGRAM2.TXT = PROGRAM1.TXT
A >
```

PROGRAM1.TXT is assumed to be on the current system drive, Drive A. The above command will instruct PIP to make a copy of PROGRAM1.TXT, name that copy PROGRAM2.TXT, and place PROGRAM2.TXT on Drive B. When these operations have been completed, the program will respond with the system prompt.

When the PIP command is used as a program, it can perform a number of different copy operations. The example below illustrates the use of PIP as a program.

```
A > PIP

*B:PROG1.BAK = PROG1.TXT
*A: = B:PROG3.BAS
*
A >
```

Once PIP has been executed as a program, it will display the symbol, *, which is the PIP prompt. Once the PIP prompt has been displayed, PIP commands may then be executed. In our example above, the first command makes a copy of PROG1.TXT on Drive A, names that copy PROG1.BAK, and then places PROG1.BAK on Drive B.

The next line makes a copy of PROG3.BAS, which is on Drive B, uses the original filename, PROG3.BAS for the copy, and places that copy on Drive A.

By pressing Return, PIP will be ended as a program. Control will return to CP/M, and the system prompt will appear.

The format for copying files using PIP is as follows.

d:copy = d:original

The d: is the letter representing the disk drive. The copy is the new name for the file being copied, and the original is the original filename.

If the d: on the right of the equal sign is omitted, PIP will assume that the file to be copied is on the current drive. The d: on the left may also be omitted, as long as a filename is supplied. In this case, the copy with the new filename will be placed on the same diskette with the original.

If the name of the copy is to be the same as that of the original, the copy's filename need not be keyed in, although the new drive letter must be included. An example of this is shown below.

d: = original

The first d: must be a different drive than the second d:. The above format will only work if the drive that is to receive the copy is different from that containing the original file. This is because of the rule that a single diskette cannot have two files with the same filename.

The following examples will illustrate some uses of abbreviated

PIP commands. We will assume that PROGRAMA.BAS is on Drive B and FILEA.TXT is on Drive A.

```
A > PIP

*A: = B:PROGRAMA.BAS
*B: = FILEA.TXT
*A:NEWFILE.TXT = A:FILEA.TXT
*B:FILEA.TXT = FILEA.TXT
```

COPYING SEVERAL FILES WITH PIP

Of course, multiple individual PIP commands can be used to copy several different files. For example, the following three files:

PROGRAMA.BAS
TEXTA.BAS
FILEA.BAS

could be copied from Drive B to Drive A by using the following PIP commands.

```
A > PIP

*A: = B:PROGRAMA.BAS
*A: = B:TEXTA.BAS
*A: = B:FILEA.BAS
*
A>
```

However, there is an easier way to copy multiple files using PIP. There are two symbols used by PIP, ? and *, which can be used when copying multiple files.

The symbol ? may be used in a filename with a PIP command. The ? will match any character that appears in its position, as shown below.

PROGRAM?.BAS

This will match the following:

PROGRAMA.BAS
PROGRAMB.BAS
PROGRAMC.BAS

but will not match:

PROG.BAS
PROGRAA.BAS

Suppose the above 5 files were on Drive B. The following PIP command can be used to copy PROGRAMA.BAS, PROGRAMB.BAS, and PROGRAMC.BAS from Drive B to Drive A.

A > PIP A: = B:PROGRAM?.BAS↵

All three files will transfer with just one PIP command.

The second matching symbol, *, will match anything in its field when used with PIP. For example, let's assume the following files were on Drive B.

PROGRAMA.TXT
PROGRAMA.BAS
FILE1.TXT
FILE1.BAK
FILE2.TXT
DATA1.BAS

If the following command was issued using the symbol, *;

A > PIP A: = B:PROGRAMA.*↵

the files, PROGRAMA.TXT and PROGRAMA.BAS would be copied to Drive A.

If the following command was issued;

A > PIP A: = B:*.TXT↵

the files, PROGRAMA.TXT, FILE1.TXT, and FILE2.TXT would be copied to Drive A.

COPYING ALL FILES WITH PIP

First of all, take note that the title of this section is "COPYING ALL FILES WITH PIP", not "COPYING AN ENTIRE DISKETTE WITH PIP". That is because it is not possible to copy the CP/M system itself with the PIP command. This is due to the fact that the CP/M system is not a file.

If a diskette contains only files, then PIP can be used to copy an entire diskette. However, if a diskette contains files as well as the CP/M system, PIP can only copy the files. SYSGEN must be used to copy the CP/M system.

Another thing to remember about the CP/M system is that it is not listed in the diskette directory by the DIR command. The DIR command only lists files. Since CP/M is not a file, it is not listed as a file. To determine whether or not CP/M is on a diskette, try to bring up the system prompt by executing a warm start by pressing Control C.

With these warnings in mind, let's turn to the use of PIP to copy all files on a diskette.

To copy all files on a diskette, you would use the symbol, *, with the PIP command as illustrated below.

```
A > PIP B: = A: *.*
```

This command would copy all files on the diskette in Drive A onto the diskette on Drive B.

The format below is used to copy all files on a diskette.

```
PIP d: = d: *.*
```

The first d stands for the letter of the drive containing the diskette which is to accept the copies. The second d stands for the drive holding the diskette containing the files which are to be copied.

When you do use the * symbol with PIP to copy all files on a diskette, we strongly recommend that you use the V option as

illustrated below.

```
A > PIP B: = A:*.* [V]
```

The V (verify) options checks to see that the files copied are identical to the original. The drawback to using the V option is that the PIP command takes much longer when it is used with the V option.

COPYING WITH PIP

COPYING FILES WITH ONE DRIVE

Copying files using PIP is much more difficult using one drive than two. The following steps below show how to copy FILE1.DAT to another disk (B) using PIP.

Step 1. The user should first insert a diskette that has PIP on it. PIP should be typed in, followed by a Return. The diskette with the file to be copied is inserted and the copy command is typed in as shown below:

```
A > PIP
*B: = A:FILE.DAT

Please mount disk B on main drive (hit enter when ready)
```

Step 2. When the preceding message appears, the user should exchange diskettes and press Return. What PIP does is create a file called FILE1.$$$ for FILE1.DAT. If FILE1.DAT exists on diskette B, PIP will delete FILE1.DAT on diskette B, the following will be displayed.

```
A > PIP
*B: = A:FILE1.DAT

Please mount disk B on main drive (hit enter when ready)

Please mount disk A on main drive (hit enter when ready)
```

Step 3. Diskette B should now be removed from the drive and diskette A inserted into the drive. The Return key should then be pressed. PIP now will load FILE1.DAT from diskette A into memory. PIP will then display the following prompt.

```
A > PIP
*B: = A:FILE1.DAT

Please mount disk B on main drive (hit enter when ready)

Please mount disk A on main drive (hit enter when ready)

Please mount disk B on main drive (hit enter when ready)
```

Step 4. Again diskette A should be removed from the drive and diskette B inserted in its place. The user should then press Return. PIP will write FILE1.DAT from memory to diskette B. The PIP prompt should reappear.

When larger files are copied with PIP in a single drive system, additional prompts may appear prompting the user to mount the diskette on either drive A or drive B. Continue to follow the instructions given by PIP until the copying operation has been completed.

COPYING WITH TWO DISK DRIVES USING PIP

If your system is equipped with two or more disk drives, you're in luck. Using PIP with a two drive system is both easier and faster than using PIP with a one drive system.

PIP can be used in either its single line format as shown below, or PIP can be used in its multiple command format.

```
A > PIP B: = NEW.TXT
A >
```

In the preceding example, PIP is used to copy NEW.TXT from drive A to drive B. Note that after Return is pressed, the CP/M prompt appears.

PIP can also be used as a program to perform multiple copy operations. An example of PIP usage for multiple copy operations is shown below:

```
A > PIP
*B: = NEW.TXT
*B: = OLD.TXT
*
A >
```

In the first line of the preceding example, PIP is loaded into the computer's memory. Once PIP has been loaded, the user can enter one or more copy operations.

When the Return key is pressed with no preceding entry for a copy operation, the PIP program will end, and the CP/M prompt will be displayed.

Notice how much easier copying operations are when your system has two disk drives installed rather than just one. No complex disk swaps are required with a two drive system, unlike a one drive system.

USING [V] TO VERIFY DISK COPYING OPERATIONS

The V (verification) can be used with PIP commands to verify that the files being copied were copied accurately. When the V option is used with PIP, the copying operation is slowed somewhat.

It is recommended that the V option be used when copying important files, to be certain that these files were copied correctly.

The V option must be enclosed in brackets and must appear at the end of the PIP command. Generally, spaces may not separate the brackets used with V and the last character of the PIP command. This is also the case with the other PIP options.

The following are examples of the use of PIP commands with the V option.

```
A > PIP B: = A:FILE.TXT[V]
A >
```

```
A > PIP

*B: = A:FILE1.TXT[V]
*B: = A:FILE2.TXT[V]
*
A >
```

In the first example, FILE.TXT will be copied from the diskette in drive A to the diskette in drive B and will then be verified.

In the second example, FILE1.TXT and then FILE2.TXT will be copied from A to B and then verified.

ABORTING PIP

If you press any character key on the keyboard while PIP is transferring a file, the transfer process will be aborted. The message, ABORTED, will be displayed on the video screen as confirmation of this.

USING PIP TO COPY TO OTHER PERIPHERALS

PIP can transfer files between almost any periphal device, not just between two disks. In this section, we will describe how PIP may be used to transfer files between any two devices.

TRANSFERRING FROM DISK TO PRINTER WITH TYPE

The process of transferring a file from a disk drive to the printer is

commonly referred to as printing a file. Most applications software packages have specialized print routines that send data from the disk files to the printer. The major advantage of these specialized print routines is that your data is formatted according to the program specifications.

Also, CP/M's TYPE command can be used as illustrated below to send a file to the printer.

```
A > ↑P
A > TYPE FILE1.TXT↵
```

The Control P will turn the printer on. The TYPE command will result in the file being printed exactly as it is on the disk without any reformatting.

There is one formatting facility offered with the TYPE command, Control I. Control I will expand any tab characters contained in the file, and assume a tab position at every eighth column.

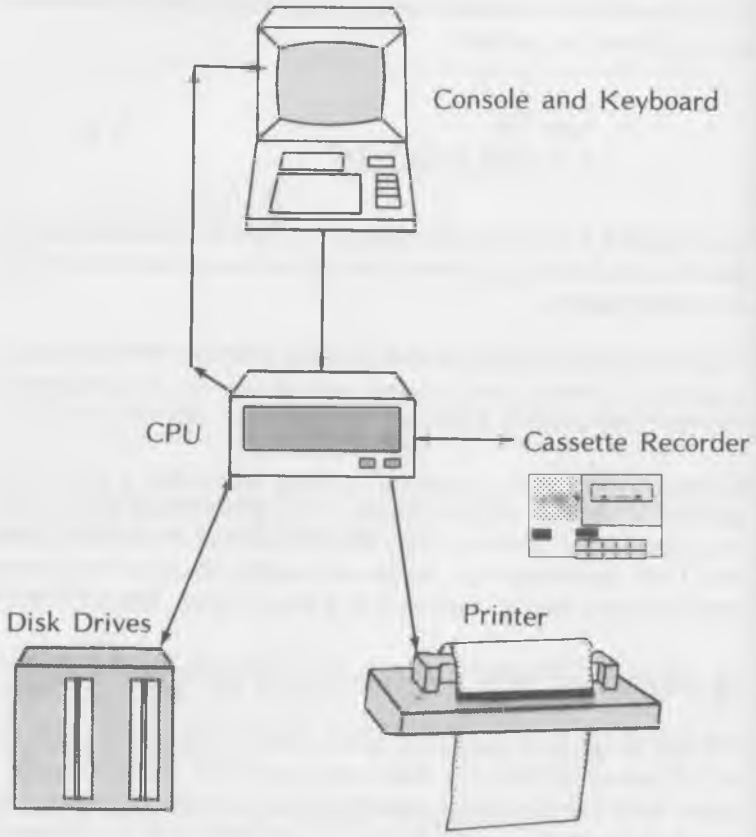
Generally, the TYPE command is used to display a file on the screen rather than on the printer. The CRT terminal displays text at a speed 15 to 30 times faster that the printer. Therefore, using the TYPE command to the screen rather than to the printer results in text being displayed at a much faster rate.

TRANSFERRING FILES TO PERIPHERALS WITH PIP

PIP can be used to send a file to any device capable of receiving it. PIP can send files to a disk drive, a printer, a video screen, a paper tape punch, and a cassette tape recorder. However, PIP may not be used to send files to a card reader or a keyboard.

Generally, the file is input from a disk and output to the screen or printer. However, files can also be input from the cassette tape unit or the keyboard. Files can also be output to a disk drive or to the cassette recorder. Illustration 6-1 displays some of the input and output combinations possible.

ILLUSTRATION 6-1. INPUT/OUTPUT TRANSFER POSSIBILITIES



Files can be transferred between different peripheral devices via file transfer routines in programs. However, PIP can also be used to transfer files between different peripheral devices. When PIP is used to make such transfers, special keywords must be used to represent the various peripheral devices. We will examine these in the next section.

DEVICE KEYWORDS

First of all, the difference between a physical device name and a logical device name must be understood. A physical device name is the actual name of a specific peripheral device. A logical file name is an overall name that can actually refer to any one of several peripheral devices.

The following are the logical device names allowed in PIP commands.

CON--or console; includes the keyboard and the screen display.

LST--or listing device; includes the printer, teletype, or modem.

PUN--or punch; includes tape or card punches.

RDR--or reader; includes card or paper tape reader.

In most cases, the CON device will be the terminal; the LST device will be the printer; and the PUN and RDR device names will not be used.

The STAT command as explained earlier can be used to display the specific device assignments for each logical device name. The STAT command can also be used to change these assignments.

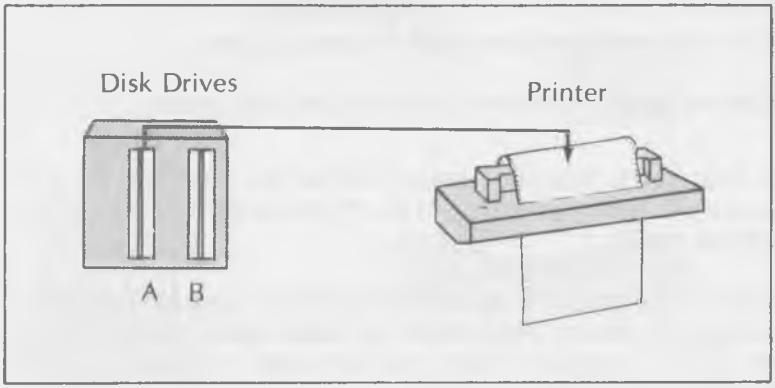
EXAMPLES OF PIP FILE TRANSFERS

The logical file names just desribed can be used with the PIP command to transfer files between peripheral devices. The following examples illustrate this usage of PIP.

```
A > PIP
*LST: = A:TEXT1.FILE
*CON: = B:PROGRAMA.BAK
*PROGRAM.BAS = RDR:
*PUN: = TEXT2.FILE
*
A >
```

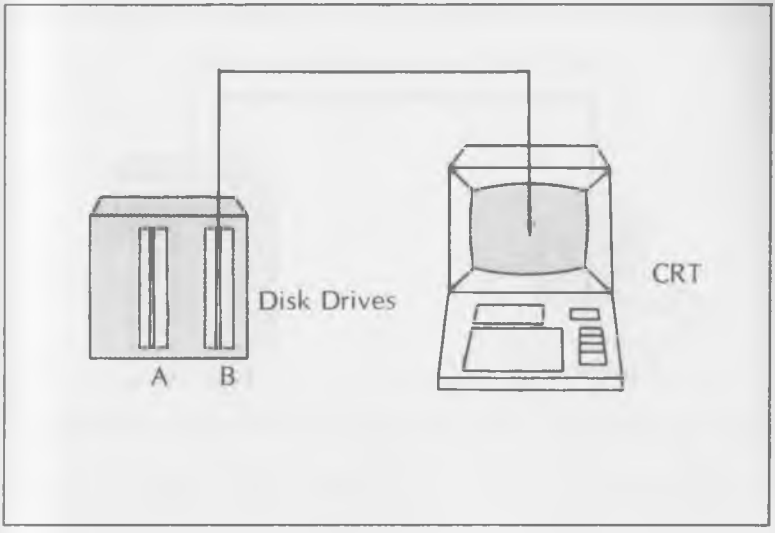
The first PIP command transfers a copy of TEXT1.FILE from Drive A to the LST: device. This is shown in Illustration 6-2.

ILLUSTRATION 6-2. LST: = A:TEXT1.FILE



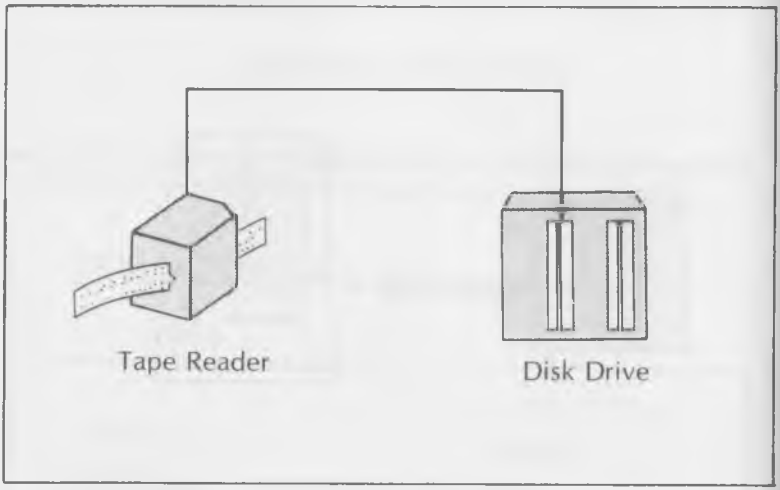
The second PIP command transfers PROGRAMA.BAK from Drive B to the CON: device. This is shown in Illustration 6-3.

ILLUSTRATION 6-3. CON: = B:PROGRAMA.BAK



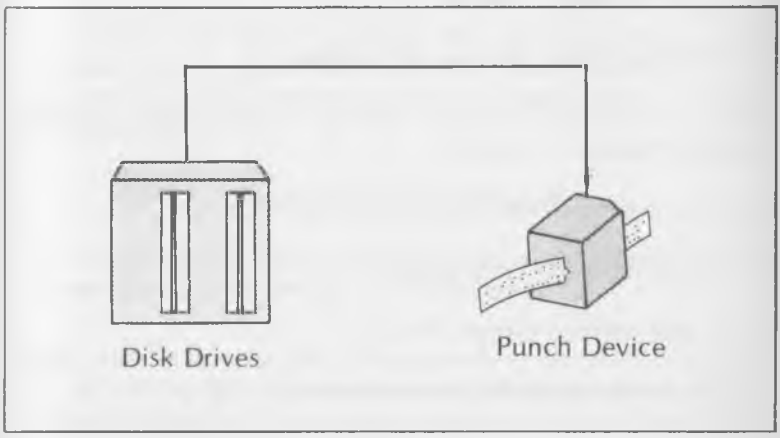
The third PIP command reads the information coming from the reader device and creates the file PROGRAM.BAS. This command allows a program on punched cards or paper tape to be read into the system and stored as a file on disk. This is illustrated in Illustration 6-4.

ILLUSTRATION 6-4. PROGRAM.BAS = RDR:



The fourth PIP command sends a copy of the file PROG.BAS from the disk to the punch device. This is shown in illustration 6-5.

ILLUSTRATION 6-5. PUN: = TEXT2.FILE



PHYSICAL DEVICE NAMES IN PIP

Physical as well as logical device names may be used with PIP commands.

CRT:--or Cathode Ray Tube for the console or terminal.

LPT:--for the line printer.

PTP:--for the paper tape or card punch.

PTR:--for the paper tape or card reader.

TTY:--or teletype. This can also be used for the console, terminal, reader, or punch.

UC1:--user defined console or terminal.

UL1:--user defined listing device.

UP1:--user defined output device.

UP2:--second user defined output device.

UR1:--user defined reader.

UR2:--second user defined reader.

COMPLEX PIP TRANSFERS

The preceding PIP operations all performed just file transfers. However, PIP can do more than simply copy files. PIP has the capability to also perform complex processing procedures on text and hexadecimal files while they are being transferred.

SPECIAL DEVICE NAMES

One of these additional PIP capabilities are special device names. The following special device names may be used with PIP.

EOF:--sends an end-of-file mark to the device named. The end-of-file mark is the ASC II code for ↑Z. The end-of file (EOF)

mark is automatically sent by PIP during ASC II text file transfers. The EOF special device name need only be used to send this code in special cases. The following command illustrates the use of EOF.

```
*PUN: = XYZ.ASM, EOF:↵
```

This command sends a copy of XYZ.ASM to the punch device followed by the EOF character (↑Z).

NUL--sends the ASC II code for the null character (0) to the physical device specified. The NUL special device name is generally used in connection with a punch (PUN) device, as shown below.

```
*PUN: = FILE1.HEX, NUL:↵
```

The above example sends FILE1.HEX to the PUN device followed by the ASC II code for the null character.

PRN:--works like the LST: device name in that the file is sent to the LST device (generally the printer). The difference in PRN lies in the fact that the tabs are expanded every eighth character, the lines are numbered, and form feeds (advance printer to new page) are given after every 60 lines of output. Also, a form feed is given before the first line is printed in that the output begins with a new page. The use of PRN is illustrated below.

```
*PRN: = FILE1.TXT ↵
```

SENDING TEXT FILES TO DEVICES

The files created by word processing applications and editor programs as well as most data files are text files. In text files, the binary codes represent text, while in program files (.COM, .BAS, .INT, .HEX), the binary codes represent actual instructions or numbers. Certain PIP special copy operations can only be performed on text files. These include copying only part of a file, dropping characters during the copy operation and translating upper case characters to lower case.

Also, only certain devices may receive or send text files. For example, text files may be only sent to printers or the video screen. The reader and punch devices (RDR: and PUN:) can send or receive text files.

A text file generally is coded in ASC II format. In ASC II, a byte is used to represent a character. A byte consists of 8 bits. Table 6-1 shows the ASC II code in byte form for all characters including special control characters. The ASC II codes are also listed with their binary and hexadecimal equivalents in Table 6-2.

TABLE 6-1. ASC II CONVERSION TABLE

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1	Column / Row	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

NUL	Null	DC1	Device control 1
SOH	Start of heading	DC2	Device control 2
STX	Start of text	DC3	Device control 3
ETX	End of text	DC4	Device control 4
EOT	End of transmission	NAK	Negative acknowledge
ENQ	Enquiry	SYN	Sychronous idle
ACK	Acknowledge	ETB	End of transmission block
BEL	Bell, or alarm	CAN	Cancel
BS	Backspace	EM	End of medium
HT	Horizontal tabulation	SUB	Substitute
LF	Line feed	ESC	Escape
VT	Vertical tabulation	FS	File separator
FF	Form feed	GS	Group separator
CR	Carriage return	RS	Record separator
SO	Shift out	US	Unit separator
SI	Shift in	SP	Space
DLE	Data link escape	DEL	Delete

TABLE 6-2. ASC II CHARACTER CODES IN ASCENDING ORDER

Hexa-decimal	Binary		ASCII
00	000	0 0 0 0	NUL
01	000	0 0 0 1	SOH
02	000	0 0 1 0	STX
03	000	0 0 1 1	ETX
04	000	0 1 0 0	EOT
05	000	0 1 0 1	ENQ
06	000	0 1 1 0	ACK
07	000	0 1 1 1	BEL
08	000	1 0 0 0	BS
09	000	1 0 0 1	HT
0A	000	1 0 1 0	LF
0B	000	1 0 1 1	VT
0C	000	1 1 0 0	FF
0D	000	1 1 0 1	CR
0E	000	1 1 1 0	SO
0F	000	1 1 1 1	SI
10	001	0 0 0 0	DLE
11	001	0 0 0 1	DC1
12	001	0 0 1 0	DC2
13	001	0 0 1 1	DC3
14	001	0 1 0 0	DC4
15	001	0 1 0 1	NAK
16	001	0 1 1 0	SYN
17	001	0 1 1 1	ETB
18	001	1 0 0 0	CAN
19	001	1 0 0 1	EM
1A	001	1 0 1 0	SUB
1B	001	1 0 1 1	ESC
1C	001	1 1 0 0	FS
1D	001	1 1 0 1	GS
1E	001	1 1 1 0	RS
1F	001	1 1 1 1	US

TABLE 6-2 (CONT). ASC II CHARACTER CODES IN ASCENDING ORDER

Hexa-decimal	Binary	ASCII
20	010 0000	SP
21	010 0001	!
22	010 0010	"
23	010 0011	#
24	010 0100	$
25	010 0101	%
26	010 0110	&
27	010 0111	'
28	010 1000	(
29	010 1001	)
2A	010 1010	*
2B	010 1011	+
2C	010 1100	,
2D	010 1101	-
2E	010 1110	.
2F	010 1111	/
30	011 0000	0
31	011 0001	1
32	011 0010	2
33	011 0011	3
34	011 0100	4
35	011 0101	5
36	011 0110	6
37	011 0111	7
38	011 1000	8
39	011 1001	9
3A	011 1010	:
3B	011 1011	;
3C	011 1100	<
3D	011 1101	=
3E	011 1110	>
3F	011 1111	?

TABLE 6-2 (CONT). ASC II CHARACTER CODES IN ASCENDING ORDER

Hexa-decimal	Binary	ASCII
40	100 0000	@
41	100 0001	A
42	100 0010	B
43	100 0011	C
44	100 0100	D
45	100 0101	E
46	100 0110	F
47	100 0111	G
48	100 1000	H
49	100 1001	I
4A	100 1010	J
4B	100 1011	K
4C	100 1100	L
4D	100 1101	M
4E	100 1110	N
4F	100 1111	O
50	101 0000	P
51	101 0001	Q
52	101 0010	R
53	101 0011	S
54	101 0100	T
55	101 0101	U
56	101 0110	V
57	101 0111	W
58	101 1000	X
59	101 1001	Y
5A	101 1010	Z
5B	101 1011	[
5C	101 1100	\
5D	101 1101	]
5E	101 1110	^
5F	101 1111	—

TABLE 6-2 (CONT). ASC II CHARACTER CODES IN ASCENDING ORDER

Hexa-decimal	Binary	ASCII
60	110 0000	`
61	110 0001	a
62	110 0010	b
63	110 0011	c
64	110 0100	d
65	110 0101	e
66	110 0110	f
67	110 0111	g
68	110 1000	h
69	110 1001	i
6A	110 1010	j
6B	110 1011	k
6C	110 1100	l
6D	110 1101	m
6E	110 1110	n
6F	110 1111	o
70	111 0000	p
71	111 0001	q
72	111 0010	r
73	111 0011	s
74	111 0100	t
75	111 0101	u
76	111 0110	v
77	111 0111	w
78	111 1000	x
79	111 1001	y
7A	111 1010	z
7B	111 1011	{
7C	111 1100	¦
7D	111 1101	}
7E	111 1110	~
7F	111 1111	DEL

Table 6-1 shows how the binary and hexadecimal equivalents of each ASC II character formed. To form the binary equivalent for a character, find the values for bits b7, b6, and b5 from the column position in Table 6-1. Then, find the values for bits b4, b3, b2, and b1 from the row position in Table 6-1. By combining these, the binary equivalent for any ASC II character can be determined.

For example, the bit values for T are 101 for b7, b6, and b5, and C100 for b4, b3, b2, and b1. The binary equivalent for T is 1010100.

Table 6-1 can also be used to find the hexadecimal equivalent for ASC II characters. The first portion of the hexadecimal value is the column hexadecimal value 0-7. The record portion is the row hexadecimal value 0-F. For example, the hexadecimal value for the ASC II character Z is 5A.

Normally, a text file is coded in ASC II where an 8 bit code is used to represent each character. However, programs that have been processed by a compiler are generally represented in the more compact hexadecimal form.

When you are transferring files to the printer or the screen display, it is important that you specify whether the data being transferred is hexadecimal or ASC II. With ASC II files, PIP will transfer a file until the end of file character (↑ Z) is reached in ASC II text files or until the end of a file is reached in other types of files.

CONCATENATING TEXT FILES

Concatenation is a process where two data items or groups are combined into one. Concatenation is most commonly applied to strings and text files. Here, we will cover the concatenation of two or more text files. The following example uses PIP to concatenate FILE1.TXT and FILE2.TXT into TOTALFILE.TXT.

```
A>PIP
*TOTALFILE.TXT = FILE1.TXT, FILE2.TXT
```

Concatenation can also be used with PIP to combine several files on different drives at once as illustrated in the following example.

```
A > PIP
*B:NEWFILE.TXT = A:TOTALFILE.TXT,B:LETTER.TXT,A:ADD.TXT
*A:FINAL.TXT = B:NEWFILE.TXT,A:FILE7.TXT,B:FILE9.TXT
*
A >
```

In the first command of the above example, TOTALFILE.TXT from Drive A, LETTER.TXT from Drive B, and ADD.TXT from Drive A, are concatenated in that order to form NEWFILE.TXT on Drive B.

In the second command, NEWFILE.TXT on Drive B, FILE7.TXT on Drive A, and FILE9.TXT on Drive B, are concatenated to form FINAL.TXT on Drive A.

Only text files can be concatenated in this manner. Non-text file concatenation follows different rules. Remember, each text file ends with the EOF character (↑ Z). When PIP concatenates text files, it removes the EOF character (↑ Z) from between the files being joined, and places one EOF mark at the end of the newly concatenated file. Naturally, if the files being concatenated were not text files, the procedures just outlined would not work.

Concatenation with text files can be used to add to an existing file as well as to create a new file. This is shown in the example below.

```
A > PIP
*ORIGINAL.TXT = ORIGINAL.TXT, ADD.TXT, NEW.TXT
*
A >
```

The above command will add ADD.TXT and NEW.TXT to the contents of ORGINAL.TXT. ADD.TXT and NEW.TXT will not be altered by this command. However, ORIGINAL.TXT will only exist in its newly concatenated form. The earlier version of ORIGINAL.TEXT will no longer exist.

Concatenation can also be used with physical and logical device names to send two or more files to a device at once. For example, the command given below will send FILE1.TXT and FILE2.TXT to the printer.

```
A >PIP↵

*LPT: = FILE1.TXT, FILE2.TXT↵
*↵
A>
```

CONCATENATING NON-TEXT FILES WITH PIP

As mentioned in the last section, concatenating non-text files with PIP requires a different set of commands than text file concatenation. Non-text files do not have an end-of-file character. PIP normally will concatenate by copying after the EOF character has been read. Therefore, to force PIP to continue copying the next file (if non-text) a parameter must be inserted where the EOF mark would occur if the file were a text file. That parameter must tell PIP to continue copying.

This transfer parameter is [X]. The X parameter must be enclosed in brackets, and positioned directly after the file it applies to.

The following command illustrates the use of the X parameter.

```
A > PIP
*FILEZ.HEX = FILEA[X], FILEB[X], FILEC↵
```

The above PIP command will concatenate FILEA, FILEB, FILEC into FILEZ. Notice that the [X] parameters are used to force PIP to continue concatenation past the ends of FILEA and FILEB.

CONCATENATING HEX FILES WITH PIP

Hexadecimal files are usually created by an assembler. An assembler translates a program written in assembly language into machine language. Machine language consists solely of binary codes which correspond to actual program instructions. This machine code is stored in a hexadecimal file. In other words, the CP/M assembler creates a HEX file when it translates an assembly language program into machine language.

HEX files have a special significance with regards to PIP. First of all, PIP will assume any HEX file to be in proper Intel format. PIP will check the HEX file for proper format, legal hexadecimal values, and check sums. Therefore, exercise caution when using HEX files with PIP.

You can use either the H-parameter or the I-parameter with a PIP command to copy a HEX file.

When the H-parameter is used, PIP will check all file data to be certain that it is in correct Intel hexadecimal format. If the data is not in proper hexadecimal format, PIP will prompt you via the terminal for corrections. The H-parameter will also remove non-essential characters from between hexadecimal records while the file copying procedures are taking place. The following is an example of the use of the H-parameter.

```
A > PIP
*NEWFILE.HEX = OLDFILE.HEX[H]
*
A >
```

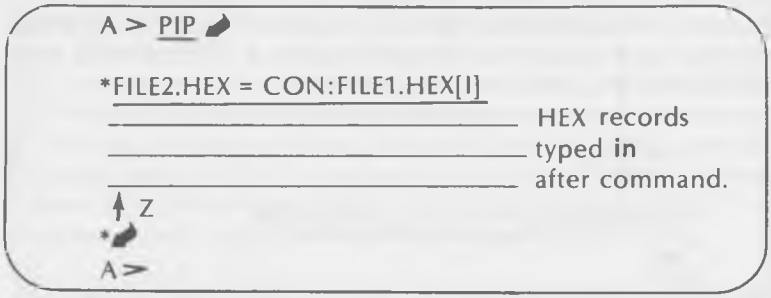
The previous command copies OLDFILE.HEX into NEWFILE.HEX while checking for invalid hexadecimal format.

The I parameter automatically sets the H-parameter. In other words, the I parameter does what the H-parameter does, plus much more. When the I-parameter is used, PIP will ignore any :00 records in the original hexadecimal format. The I-parameter also will check for improper hexadecimal format and remove non-essential records from between hexadecimal records as the H-parameter does. The following is an example of the use of the I-parameter.

```
A > PIP
*NEWFILE.HEX = OLDFILE.HEX[I]
*
A >
```

The above command transfers OLDFILE.HEX to NEWFILE.HEX, while removing :00 records and checking for valid hexadecimal format.

You can copy from a device such as the console, paper tape, or card reader to a HEX file. In these cases, you need not use the H-parameter to check for improper hexadecimal format or checksum error, although you may need to use the I-parameter to eliminate :00 records. If PIP discovers an invalid format or checksum error, it will report this error via the terminal and await correction. An example of the use of PIP for copying from a device to a HEX file is shown below.



In this example, PIP transfers the input from the CON: device (terminal) into FILE2.HEX until a ↑ Z is entered. Then, PIP continues by copying FILE1.HEX into FILE2.HEX, while checking for improper format and removing :00 records.

PIP PARMETERS

As we explained earlier, parameters are letters enclosed in brackets that follow a filename in a PIP expression. The parameters affect how that file is copied. More than one parameter may be specified in a PIP expression. Some parameters require additional letters and/or digits.

It is not necessary to memorize these various parameters. You will probably only ever use a few of them. However, it is important to know that they are available.

B--is the block mode transfer parameter. When the B parameter is specified, data from a continuous reading device such as a cassette tape unit or a paper tape reader will be read in blocks.

Blocks are sectors of a larger file. PIP will read data into a buffer until it reads the ASC II character (↑S) from the device from which the data is being read. PIP will then clear the buffer and begin reading in more data. The size of this buffer will depend upon your particular hardware. The buffer size will be documented in your system manual. An example of the B command is given below.

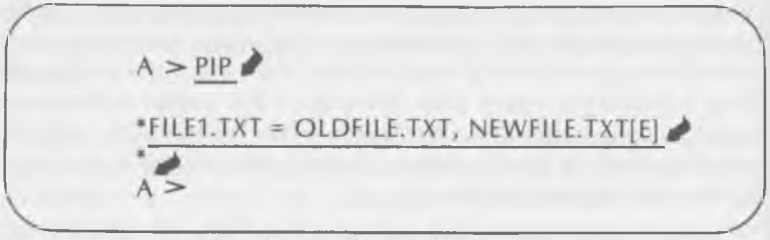
```
A > PIP
*FILE1.TXT = RDR:[B]
*
A >
```

D##--is the delete parameter. This command is used when copying text files. PIP will delete any characters past the column number given in ##. This command is valuable when sending data to a printer with a narrow column width or a terminal with a narrow screen. An example of the use of the B parameter is given below.

```
A > PIP
LPT: = FILE1.TXT[D40]
*
A >
```

The above command causes data to be printed up to column 40 on the printer.

E--is the echo or redisplay parameter. This parameter causes all copying operations to be displayed on the terminal screen as they are actually being performed. The following is an example of a command using the E parameter.



F--removes any form feeds from a file.

H--is used for a hexadecimal file transfer. PIP will check for proper Intel hexadecimal format.

I--is used with HEX files. I automatically sets the H-parameter so proper format is checked for, and also removes any :00 records during the file transfer.

L--transforms all upper case characters to lower case.

N--adds line numbers to each line being copied to a new file. Each line number is followed by a colon. Leading zeros are omitted from the line number (ex. 009, would be 9).

N2--adds line numbers to each line being copied, but leaves the leading zeros in place and inserts a tab space after each line number. N2 is much more frequently used than N.

O--is used for object (or non-ASC II file transfers).

P##--inserts a page eject at every line specified by #. For example, if ## is in actuality 40, a page eject will be placed at every 40th line. If P is used alone or P1 is specified, page ejects will occur at every 60th line.

The P parameter is often used in comination with F. The F parameter is used to remove form feeds, and then P is used to insert page ejects. This can be useful when a file must be printed to a pre-determined page format.

Q (string) ↑Z--PIP will cease copying from the file when it encounters the string of characters specified after the Q parameter. The final character of your string must be ended with ↑Z to specify the file end.

S (string) ↑Z--PIP will begin copying from the file when the string of characters following the S parameter is encountered. Again, that string must end with ↑Z.

T##--is used to expand the tab space to the number specified in place of ##. Tab space in a text file is created by Control-I (↑I). The T parameter expands this tab space.

U--is used to translate all lower case letters to upper case during the file transfer.

V--or verify is used to check to see that the data was copied correctly.

Z--turns the parity bit to zero in ASC II character inputs.

PIP PARAMETER EXAMPLES

The following examples will illustrate some of the possible usages of the PIP parameters.

```
A > PIP ↵

*PRN: = LETTER.TXT[NP] ↵
*↵
A >
```

The above PIP command sends the file LETTER.TXT to the PRN: device with line numbers (N) and with page ejects at every 60 lines (P defaults to the equivalent of P60).

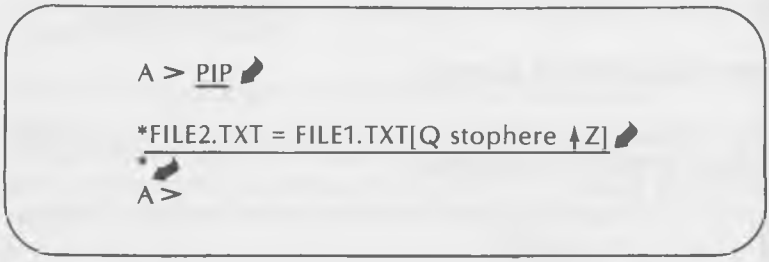
```
A > PIP ↵

*LPT: = FILE1.TXT[N2P50U] ↵
```

The above PIP command sends FILE1.TXT to the line printer with line numbers including leading zeros and followed by a tab space (N2), page ejects at every 50th line, and lower case characters translated into upper case (U).

COPYING PART OF A FILE

PIP can be used to copy only part of text file by using the S and the Q parameters. The starting and stopping places are specified by the strings named after the S and the Q parameters.

The string following the S parameter indicates the starting point, while the string following the Q parameter indicates a stopping point. PIP automatically searches for each of these strings as it performs its copy operations. Remember, to end each string after the S and Q parameter with Control Z (↑ Z). The following example illustrates the use of the S and Q parameters to copy only a portion of a file.



The PIP command will copy FILE1.TXT from its beginning point until the string "stophere" is encountered. The new file will be named FILE2.TXT. If the PIP command were entered as in the example below, a different set of circumstances would ensue.

```
A > PIP FILE2.TXT = FILE1.TXT[Q STOPHERE ↑Z]
```

In this example, PIP would copy until the string "STOPHERE" was encountered in FILE1.TXT. Notice that in the first example, PIP searches for "stophere" while in the second, PIP is seaching for "STOPHERE".

Whenever PIP is executed as a one line command (as in the second example), the string is always translated to the uppercase. This is not the case where the copy expression is typed in following the prompt, *.

The following is an example of the use of both the S and the Q parameters.

```
A > PIP↵

*NEWFILE.TXT = OLDFILE.TXT[X Beginhere ↑Z Q Stophere ↑Z]↵
*↵
A >
```

The PIP command will begin copying OLDFILE.TXT when it finds the string "Beginhere". PIP stops copying when it finds the string "Stophere".

NEWFILE.TXT will contain that portion of OLDFILE.TXT between those two strings.

Note that PIP was executed as a program, rather than as a one-line command. This means that the upper-lower case string "Beginhere" and "Stophere" will begin and end PIP when they are encountered. If PIP had been executed as a one-line command, the upper case string "BEGINHERE" and "STOPHERE" would have had to be encountered for PIP to start or end its copying operations.

USING PIP IN CP/M 2.2 & MP/M

CP/M 2.2 contains some restrictions regarding the use of PIP that do not apply to CP/M 1.4. Most of these differences deal with user areas. For example, if you are making use of user areas in CP/M 2.2, you can not create a file in another user area, nor can you copy from a file that resides in another user area.

Before we explain the differences between using PIP in CP/M 1.4, we will briefly discuss the topics of user areas and file attributes. This discussion will help clarify the use of PIP in CP/M and MP/M.

USER AREAS

Any diskette can be separated into separate user areas. You have the option to have only one user area, but CP/M allows you to separate your files into separate user areas and go from one user area to another by issuing the USER command.

CP/M 2.2 has been structured with the option of one user area or multiple user areas, so that it is compatible with MP/M. If you plan to eventually upgrade to a multi-user system, your CP/M diskettes will be compatible with an MP/M system. If you wish to use CP/M so that your disks are compatible with earlier versions of CP/M and MP/M, use only user area 0.

FILE ATTRIBUTES

In both CP/M and MP/M, you have the option to place a file attribute on a file. A file attribute indicates how a file is to be used by the system. Examples of file attributes are system, ($SYS), directory ($DIR), read-only (R/O), and read-write (R/W). File attributes are set with the STAT command.

The read-only file attribute (R/O) indicates that a file cannot be changed or erased. In other words, the system is not allowed to write to the file. Before you can write to a file with a R/O file attribute, that attribute must be changed to read-write (R/W) via the STAT command.

If a file has the system file attribute ($SYS), it will be displayed by the DIR command. You also cannot read from a $SYS file. This also means that the file cannot be copied. By giving a file the R/O and $SYS file attributes, that file will be write-protected.

PIP in CP/M 2.2 and MP/M contains several parameters which can be used to override the restrictions of file attributes and also to copy files from one user area to another. These parameters are as follows.

G##--stands for get. It allows you to get a file from the user area specified. This parameter also allows you to write to read-only (R/O) files.

R--allows you to read files with system ($SYS) file attributes. This parameter also allows you to write to read-only (R/O) files.

W--allows you to write to read-only (R/O) files.

USING PIP TO COPY FROM USER AREAS

As long as a file exists in the same user area as it is being copied

to, it can be transferred between diskettes without using the G parameter. For example, if FILE1.TXT exists in user area 3 of Drive A, you can copy it to user area 3 of Drive B without using the G parameter. However, you must be in user area 3 to make this transfer.

If you wish to make a copy of a file that exists in another user area in your current user area, you must use the G parameter as shown in the following example.

```
3A > PIP
*A: = B:FILE1.TXT[G4]
*
3A >
```

As shown by the system prompt, the current disk drive is A and the current user area is 3. The above PIP command will transfer FILE1.TXT residing in user area 4 of Drive B to user area 3 (the current user area) of Drive A.

If you want to use PIP to copy other user areas, you first will have to be sure that a copy of PIP.COM resides in the user area which is to receive the copy. If PIP does not exist in your current area of at least one of your disk drives, then you can not use the PIP command. The following sequence of commands illustrates how to copy PIP into user areas.

2A > USER 0	Change from user area 2 to user area 0.
0A > DDT PIP.COM	DDT--debugger--is executed on PIP.COM.
DDT VERS.xx.xx	This is the DDT sign-on message.
NEXT PC IC80 xxxxxx	IC80 is the hexadecimal address at the end of PIP.COM. This is used to calculate the number of pages to be used with the SAVE command.

-G0	G0 terminates DDT, but leaves PIP.COM in TPA, so that a copy can be saved with the SAVE command.
0A > USER 5	We are supposing that you wish to put a copy of PIP.COM in user area 5.
5A > SAVE 28 PIP.COM	SAVE will create a copy of PIP.COM in user area 5 of Drive A, with 28 pages of memory. The 28 is derived by converting the high order byte (first two characters) of the hexadecimal 1C80 into its decimal value. Hexadecimal 1C is the equivalent of 28 in base 10.

USING PIP WITH READ-ONLY FILES

When a file has the read-only system attribute, PIP will not overwrite it. That is, PIP will not delete the original file and create a new file with the same filename in its place. If you do attempt to overwrite an R/O file, PIP will respond with a prompt as illustrated below.

```
A > PIP B: OLDFILE.TXT = NEWFILE.TXT
DESTINATION FILE IS R/O, DELETE (Y/N)? Y
A >
```

In the above example, we attempted to make a copy of NEWFILE.TXT and place it on Drive B with the filename OLDFILE.TXT. However, OLDFILE.TXT already exists on Drive B with the R/O file attribute.

PIP responds with the message that the destination file, OLDFILE.TXT, is read-only, and asks if we wish to delete OLDFILE.TXT. By replying with a Y for yes, the original OLDFILE.TXT will be deleted and replaced with a copy of NEWFILE.TXT. This copy will reside on Drive B with the filename, OLDFILE.TXT. It will not have the R/O file attribute.

If the response to the above prompt was N for no rather than Y for yes, PIP would have responded with the following message.

NOT DELETED

You can use the W parameter to override the display of this prompt message. The W parameter tells PIP to ignore the R/O system attribute. When the W parameter is placed at the end of a PIP expression in which a number of files are being copied, the R/O attributes of all of those files will be ignored. This is illustrated below.

```
A > PIP NEWFILE.TXT = FILE1.TXT, FILE2.TXT, FILE3.TXT[W]

A>
```

USING PIP WITH SYSTEM FILES

If either the original file or the copy file has the system ($SYS) file attribute, PIP will not be able to locate these files, as they will not be listed in the disk directory. The R parameter can be used so that PIP ignores both the $SYS and the R/O file attributes. This allows PIP to find the original file or create the copy file.

If the R parameter is placed at the end of a number of different files, it will ignore the R/O and $SYS file attributes of all of the files being concatenated. This is illustrated below.

```
A > PIP NEW.TXT = COPY1.TXT, FILE1.TXT, FILE2.TXT[W]
```

CHAPTER 7. THE CP/M EDITOR (ED)

INTRODUCTION

In this chapter, we will describe in detail the use of a very important CP/M program, the editor (ED). In this chapter, we will describe both how to use the editor program as well as how the editor works, and what facilities it offers to the user. If you are using a word processing applications program with your computer, you will find our discussion of the editor very valuable in helping you understand your word processing system.

Any editor program, including CP/M's, allows you to create and edit text files. A text file is any file that consists of characters; a book, poem, letter, manual, etc. The editor program allows you to easily move from line to line and change, delete, or insert characters without extensive retyping. A good editor program should also allow you to locate any group of characters you indicate in the text file, and to substitute for them. Finally, an editor program should allow you to combine two text files and interweave separate lines of text from each of them.

You should be aware of the difference between a word processing applications program and an editor utility program. If you plan to use your microcomputer for extensive word processing tasks, you would be wise to consider the purchase of a word processing applications program.

A word processing program includes an editor program for inputting text, and also includes a program for outputting that text on the printer. A good word processing program allows you to output text in a number of different formats; right justified, left justified, both margins justified, columns of data in tab positions, boldface type, expanded type, and many more formats.

Choose your word processor carefully. Some word processors are more well suited to a particular task than others. Before choosing a word processing program, define your work tasks. Then, choose the word processing program that will best accomplish those tasks.

ED--THE CP/M EDITOR

The editor program, ED.COM, is included on the system diskette. If you wish to execute the editor program, you can do so by keying in ED followed by the name of the text file that you wish to edit. This is illustrated below.

```
A > ED FILE1.TXT↵
```

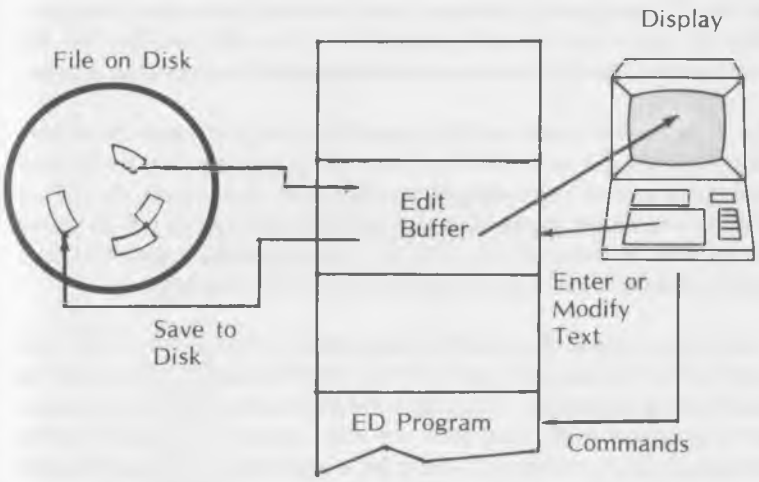
You may now use ED to create or modify FILE1.TXT. Remember, a filename must be supplied with the ED command. Otherwise, the editor program will not execute.

When you use the ED command with a filename as illustrated above, ED will first check the disk directory to see if the filename specified is that of an existing file. If the filename specified in the ED command cannot be found on the disk directory, ED will assume that a new file is to be created with the filename specified.

The file that is specified in your ED command becomes the source file. Any following ED commands will bring this source file into the edit buffer as pictured in Illustration 7-1.

The edit buffer is a block of memory within the computer that has been reserved for ED's use in processing text files. In CP/M's edit program, only a portion of large text files may be loaded into

ILLUSTRATION 7-1. EDIT BUFFER



the edit buffer at any one time. More powerful editor programs can handle larger text files.

After the text file has been moved to the edit buffer, you can change that text or type in new text to add to it. Before leaving the ED program, you must recopy the modified text file from the edit buffer back to the disk. If you end ED or turn off the system before the modified text file is copied from the buffer back to the disk, that modified text file will be lost. The original text file (without changes) will still exist on the disk. The modified text file in the edit buffer is saved by using the E command.

Once the ED program has been executed, the ED prompt will be displayed until program is terminated. The ED prompt is as follows, *.

The E command would be keyed in as follows.

*E ↵

We will discuss more of the commands used in ED in the following sections.

CP & LINE NUMBERS

CP stands for the character pointer. The character pointer is used by the ED program to always point at one particular character. The CP can be moved around in the edit buffer by ED commands. The CP is not actually displayed on the video screen.

The commands used in ED generally refer to those characters following the CP, or both the character pointed to by the CP and those characters following that character. You move the CP to the right to move forward in the text file, and to the left to move backwards in the text file. This may seem confusing at first, but will be made clear by the examples in this chapter.

In ED, each line in the text file is assigned a line number. The line number is not actually part of the text file, but is only used for positioning purposes. In newer CP/M versions, the line numbers are displayed with the text on the screen. In older CP/M versions, the V command must be executed to display the line numbers. You can move to any line by specifying its line number with an ED command. This will be illustrated later.

Both the CP and line numbers are actually imaginary devices, in that they exist only in the edit buffer, and are only used to move between lines and characters in that buffer. The CP and line numbers will not be output in a print run of the text file.

ED MECHANICS

To show how ED operates, let's assume that we wish to modify FILE1.TXT, which is shown in illustration 7-2.

First of all, we will execute ED as follows.

A > ED FILE1.TXT ↵

ED will set up space in the Transient Program Area (TPA) for the edit buffer. A temporary file, FILE1.$$$ will be created. This file will be used to store the edited file, so that the original file will not be erased. This initial ED also prepares FILE1.TXT to be copied into the edit buffer.

ILLUSTRATION 7-2. FILE1.TXT

```
From FILE1.TXT, this is line 1.
This is line 2.
This is line 3, but it could be
numbered differently. (This is line 4.)
And this is line 5
Line 6.
Line 7.
Line 8.
Line 9.
Line 10.
Line 11.
Line 12.
Line 13.
Line 14.
Line 15.
Line 16.
Line 17.
Line 18.
Line 19.
Line 20.
Line 21.
Line 22.
Line 23.
Line 24.
Line 25.
Line 26.
Line 27.
```

However, FILE1.TXT is not actually copied into the edit buffer until the A (append) command is given. An example of the use of the append command is given below.

```
*2A
```

The ED command 2A specifies that the first two lines of the text file are to be moved into the edit buffer. This is shown in Illustration 7-3.

ILLUSTRATION 7-3. EDIT BUFFER AFTER 2A COMMAND

```
1: From FILE1.TXT, this is line 1.
2: This is line 2.
```

If you wish to expand the edit buffer to include the first five lines of FILE1.TXT, you could do so with a second A command, as shown below.

```
*3A
```

The edit buffer would now contain the first five lines of FILE1.TXT as shown in Illustration 7-4.

ILLUSTRATION 7-4. EDIT BUFFER AFTER 2A & 3A COMMANDS

```
1: From FILE1.TXT, this is line 1.
2: This is line 2.
3: This is line 3, but it could be
4: numbered differently. (This is line 4.)
5: And this is line 5.
```

These lines now residing in the edit buffer can now be modified or added to. Illustration 7-5 shows how to add 3 new lines to the edit buffer by using the I command.

ILLUSTRATION 7-5. USING THE I COMMAND TO ADD NEW LINES TO THE EDIT BUFFER

```
1: From FILE1.TXT, this is line 1.
2: This is line 2.
3: This is line 3, but it could be
4: numbered differently. (This is line 4.)
5: And this is line 5.
: *I
6: Line 5A was just added from the keyboard.
7: Line 5B was also just added.
8: So was line 5C.
9: ^Z
: *
```

The W (Write) command is used to send text from the edit buffer into the temporary output file. As you will recall, the temporary output file is opened so as to store the edited file without writing over the original file. In our example, the temporary output file is FILE1.$$$.

Illustration 7-6 shows the effect of the W command on the edit buffer and the temporary output file.

ILLUSTRATION 7-6. EFFECT OF W COMMAND ON EDIT BUFFER & TEMPORARY OUTPUT FILE

ORIGINAL EDIT BUFFER

```
1: From FILE1.TXT, this is line 1.
2: This is line 2.
3: This is line 3, but it could be
4: numbered differently. (This is line 4.)
5: And this is Line 5.
6: Line 5A was just added from the keyboard.
7: Line 5B was also just added.
8: So was Line 5C.
```

*2W

NEW EDIT BUFFER

```
3: This is line 3, but it could be
4: numbered differently. (This is line 4.)
5: And this is line 5.
6: Line 5A was just added from the keyboard.
7: Line 5B was also just added.
8: So was line 5C.
```

TEMPORARY OUTPUT FILE--FILE1.$$$

```
From FILE1.TXT, this is line 1.
This is line 2.
```

Now that we have written two lines to the temporary output file, FILE1.$$$, with the W command, the remaining lines move up to the beginning of the edit buffer. This frees up space in the edit buffer.

If we use the A command to add 15 more lines to the edit buffer from FILE1.TXT on the disk, the results will be as shown in Illustration 7-7.

ILLUSTRATION 7-7. APPENDING 15 MORE LINES TO THE EDIT

FILE.TXT on Disk

```
From FILE1.TXT, this is line 1.
This is line 2.
This is line 3, but it could be
numbered differently. (This is line 4.)
And this is line 5.
Line 6.
Line 7.
Line 8.
Line 9.
Line 10.
Line 11.
Line 12.
Line 13.
Line 14.
Line 15.
Line 16.
Line 17.
Line 18.
Line 19.
Line 20.
Line 21.
Line 22.
Line 23.
Line 24.
Line 25.
Line 26.
Line 27.
```

ILLUSTRATION 7-7 (CONT.). APPENDING 15 MORE LINES TO THE EDIT BUFFER.

Edit Buffer in TPA (memory)

```
 3: This is line 3, but it could be
 4: numbered differently. (This is line 4.)
 5: And this is line 5.
 6: Line 5A was just added from the keyboard.
 7: Line 5B was also just added.
 8: So was line 5C.
 9: Line 6.
10: Line 7.
11: Line 8.
12: Line 9.
13: Line 10.
14: Line 11.
15: Line 12.
16: Line 13.
17: Line 14.
18: Line 15.
19: Line 16.
20: Line 17.
21: Line 18.
22: Line 19.
23: Line 20.
```

The E command can be used to end this editing session by copying the text in the edit buffer to the temporary output file, FILE1.$$$. The E command will also copy the remainder of the source file, FILE1.TXT, to the temporary output file, FILE1.$$$. By the remainder of FILE1.TXT, we mean those lines that were not appended to the edit buffer via the A command.

After the edit buffer and remainder of the source file have been copied to FILE1.$$$, ED renames the original FILE1.TXT to FILE1.BAK.

ED then renames the temporary output file, FILE1.$$$ to FILE.TXT. This is shown in Illustration 7-8.

ED is in effect creating a back-up copy of the original FILE1.TXT file called FILE1.BAK. This is the original text file before modifications in the editing session were made. ED then renames the edited file from FILE1.$$$ to FILE1.TXT.

When the ED command is executed for a text file, any BAK files associated with that file will be automatically deleted.

ED WITH SOURCE FILES

When you specify a filename with an ED command, ED first searches for that file. If the file is not found, ED assumes that it does not exist, and creates a new file with the filename given in the command. This file is called the source file. For this example, assume the source filename to be SOURCE.TXT.

Even though the source file is in fact empty, the A command is used to append lines from the source file to the edit buffer. The lines appended are each given a line number in the edit buffer.

You may then use the W command to write lines from the edit buffer to the temporary output file (SOURCE.$$$). When you write lines to the temporary output file, you will be freeing space in the edit buffer. This, of course, allows you to append more lines from the source file into the edit buffer.

When you have finished editing, use the E command to rename the original source file (SOURCE.TXT) with the BAK filename extension (SOURCE.BAK). The temporary output file (SOURCE.$$$) is then given the name of your original text file (SOURCE.TXT). The newly modified file is now stored on disk as SOURCE.TXT.

ACCIDENTALLY ERASING THE EDIT BUFFER

There are several ways that ED may be accidentally ended; a power loss, mistakenly hitting Control-C, or a system error. The E and Q commands can be used to end ED on purpose.

When ED is accidentally terminated, the data in the edit buffer will be lost. The temporary output file will be kept on the disk with its original name (ex. SOURCE.$$$), as will the original source file (SOURCE.TXT).

The temporary output file will contain only those lines written to it from the edit buffer. All data remaining in the edit buffer will have been lost.

CREATING A NEW TEXT FILE WITH ED

The first step in creating a new text file is to keyin the ED command with the desired filename. This is illustrated below.

```
A>ED SOURCE.TXT
```

Since we are in Drive A above, SOURCE.TXT will reside on that drive. We are in Drive A because the ED program itself (ED.COM) is in that drive. You can execute ED from Drive A and put SOURCE.TXT on Drive B by using the command illustrated below.

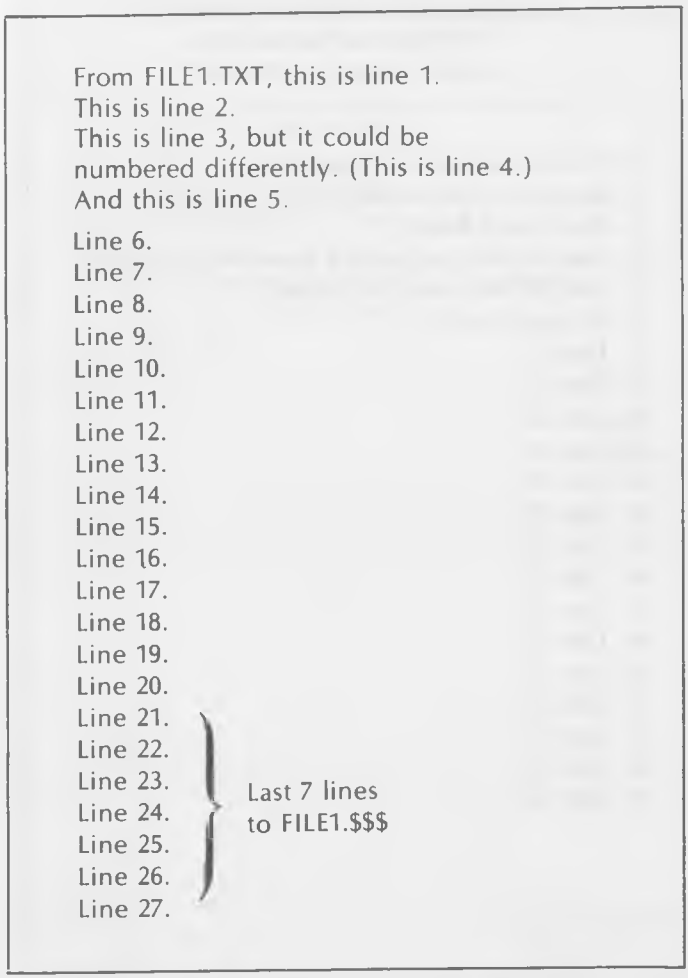
```
A>ED B:SOURCE.TXT
```

You can also execute ED from another drive by prefixing that drive's letter before the ED command as shown in the following.

```
B>A:ED SOURCE.TXT
```

ILLUSTRATION 7-8. USING THE E COMMAND TO END AN EDITING SESSION

FILE1.TXT on Disk



The last 7 lines from FILE1.TXT are output to the temporary output file when the E command is issued. FILE1.TXT is then renamed to FILE1.BAK.

ILLUSTRATION 7-8 (CONT). USING THE E COMMAND TO END AN EDITING SESSION

Edit Buffer in TPA Before E

```
 3: This is line 3, but it could be
 4: numbered differently. (This is line 4.)
 5: And this is line 5.
 6: Line 5A was just added from the keyboard.
 7: Line 5B was also just added.
 8: So was Line 5C.
 9: Line 6.
10: Line 7.
11: Line 8.
12: Line 9.
13: Line 10.
14: Line 11.
15: Line 12.
16: Line 13.
17: Line 14.
18: Line 15.
19: Line 16.
20: Line 17.
21: Line 18.
22: Line 19.
23: Line 20.
```

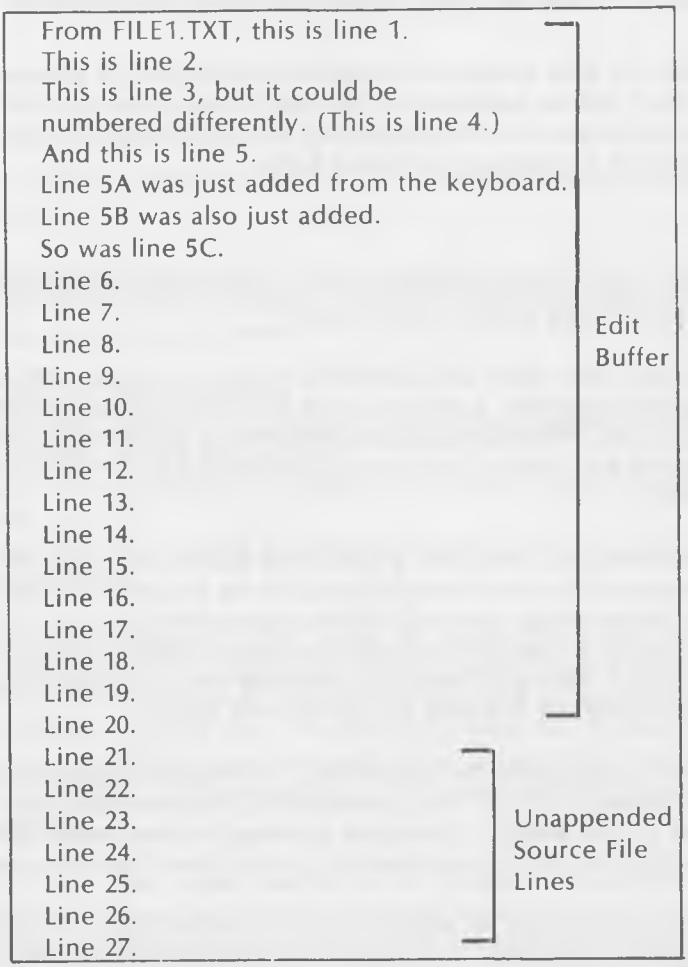
Entire Edit Buffer Output to FILE1.$$$

ILLUSTRATION 7-8 (CONT). USING THE E COMMAND TO END AN EDITING SESSION

After the following is issued;

*E

**FILE1.$$$ (Temporary Output File)
is Renamed to FILE1.TXT.
FILE1.$$$ as shown below.**



Here, ED will be executed from A, and SOURCE.TXT will be stored in A.

When a source file name is used with ED for a file that does not exist on the diskette, the following message will be displayed.

```
NEWFILE
*
```

A new file with the filename entered will be created. The ED prompt (*) indicates that the user can enter ED commands.

When the new source file has been created, the CP (character pointer) will be positioned at the beginning of the edit buffer. You can insert text at the beginning of the edit buffer by initially giving the I command, as shown below.

When the I command is given, the cursor will be moved to the next blank line on the video screen.

You may now insert text merely by typing--as you would on a regular typewriter. Each line must be ended by pressing the Return key. When Return is pressed, the line will be transmitted into the edit buffer. This line is then known as a "line in the buffer".

Sometimes, you may wish a line in the buffer to contain more characters than the number allowed by the length of one line in your video display (generally 80 characters). In these cases, when you come to the end of a line on your video screen, press Control E. This will move the cursor to the next line without transmitting the previous line to the edit buffer.

When you do press the Return key, the entire line (spread over two display lines) will be transmitted to the edit buffer as one "line in the buffer". One rule to keep in mind when using Control E is that the maximum length of a line is 128 characters.

Let's suppose that we wished to insert the first line of Hamlet's soliloquy using the I command. Our example from the initial ED command will appear as follows.

```
A > ED SOURCE.TXT

NEW FILE
*I
"To be or not to be, that is the question."
Reference: Hamlet by Wm. Shakespeare
^Z
*
```

Control Z is used to end the use of the I command as illustrated above.

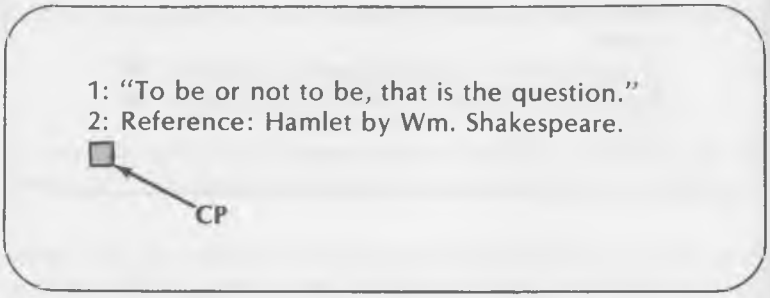
While inserting text with the I command, you can use several Control key combinations to manipulate the text being inserted. These are described below.

Control U--delete the line.
Control E--return the "carriage" without transmitting the line.
Control R--retype the last line.
Control H--backspace & delete a character (new ED version only).
Control X--backspaces to the beginning of the line (new ED version only).

Another command often used in conjunction with the I command is the U command. If you specify the U command before the I command, any text inserted will be automatically converted to upper case. The text being typed will appear to be in upper and lower case, but in actuality, it is all being converted to upper case. To terminate the U command, use the negative U command.

Once text has been placed in the buffer, it may then be displayed. However, before that display can take place, the CP (character pointer) must be moved back to the beginning of the buffer. When the I command was given and text inserted, the CP

was moved. Using our example, the CP would be positioned as illustrated below.



In the above example, the CP is positioned at the end of the buffer.

In this book, we are going to explain the position of the CP in a manner differing slightly from the way CP is described in the Digital Research reference manual. We find Digital Research's explanation to be somewhat confusing.

Digital Research defines the CP as being positioned between 2 characters as illustrated below.

We prefer to visualize the CP as an imaginary pointer as illustrated below.

In the next few sections, we will discuss methods for moving the CP.

USING T TO DISPLAY TEXT IN THE EDIT BUFFER

The T command is used to display text in the edit buffer. The configuration of the T command is as follows.

*±nT

The n can be any number, zero, or the # sign. For example, if the following command were keyed in:

*5T ↵

the next 5 lines from the CP (character pointer) will be displayed on the screen.

If a negative number is specified in the T command, that number of lines preceding the CP (but not including the CP) will be displayed on the screen.

For example, the following command:

*-5T

will display the 5 lines preceding the CP.

If a zero is specified, the current line in the edit buffer (the line with the CP in it) will be displayed on the screen from its beginning up to (but not including) the CP.

If no n is specified, a value of 1 is assigned to n by default. If a value of 1 is given n (either by assignment or by default), the current line is displayed from the CP to its end.

If a # sign is specified with T, the entire edit buffer following the CP will be displayed.

You can use the B command in conjunction with the #T to display the entire edit buffer. First, issue the B command as illustrated below.

*B ↵

This will move the CP to the beginning of the edit buffer.

Next, issue the #T command as illustrated below.

```
*#T
```

This command displays the entire edit buffer following the CP. Since the CP had previously been moved to the beginning of the edit buffer with the B command, the entire edit buffer will be displayed.

When the # sign is used with a negative sign as shown below, all lines preceding (but not including) the CP will be displayed on the screen.

```
*-#T
```

MOVING THE CP

We previously showed how the B command could be used to move the CP to the beginning of the edit buffer. Generally, the B command is used in conjunction with the T command to display the entire buffer.

The CP can be moved within the current line by using the C command. The configuration of the C command is given below.

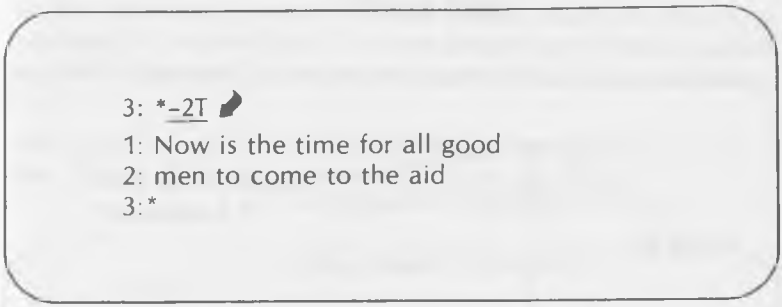
```
±nC
```

If the sign is positive, the CP will be moved forward by the number of characters given by n. If the sign is negative, the CP will be moved n characters backwards.

Let's evaluate the following example to make this discussion more concrete.

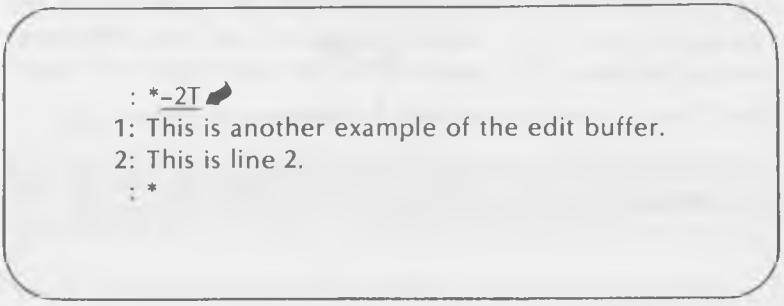
DETERMINING THE CP POSITION FROM THE VIDEO DISPLAY

The display of lines from the edit buffer on the screen will provide you with the position of the CP. Assume that the following T command was issued.



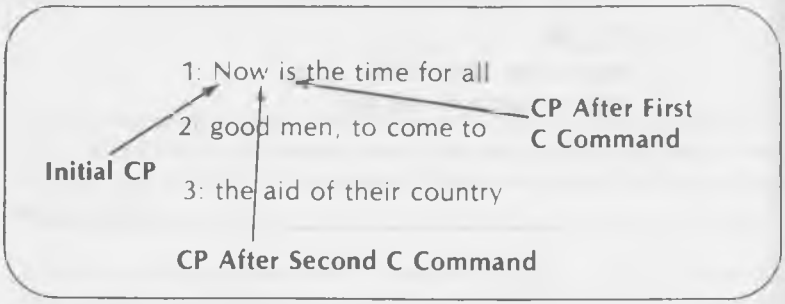
Notice that the ED prompt appears in line 3 and this line contains a line number. This tells the user that text is contained in line 3 and that the CP is positioned at the beginning of line 3.

In some cases, the prompt will be positioned at a line without a line number. This is illustrated below.



In this example, the colon (:) tells the user that the CP is positioned at the beginning of line 3, but that line 3 does not as yet have text.

EDIT BUFFER



We will assume that the CP is positioned within the edit buffer as illustrated above--pointing to the first character at the beginning of the edit buffer. Suppose the following command were issued.

*5C

The C would move five characters forward as shown above.

If the following C command were issued next:

*-3C

the CP would move backwards 3 characters as shown above.

In CP/M versions 1.4 or later, the T command can also be used to both move the CP and display a specified line number.

```
2: *3:T
3: The aid of their country.
3: *
```

In the previous command, the initial part of the first command,3:, moved the CP to the beginning of line 3, which was then displayed by the second part of the initial command, T.

This command can also be used without T to move the CP to a line. This is illustrated in the following example.

In the previous example, the initial command, 2:, moved the CP to line 2.

ENDING THE ED SESSION

Before ending the ED session, you must save the contents of the edit buffer. The E command allows you to save the contents of the edit buffer and exit the ED program simultaneously.

If you are in the edit buffer and you issue the E command, the edit buffer along with the remainder of the source file will be transferred into the temporary output file. The temporary output file is then renamed to the name of the original source file.

The original source file will be renamed with its original filename and the extension .BAK. The use of the E command is shown in Illustration 7-9.

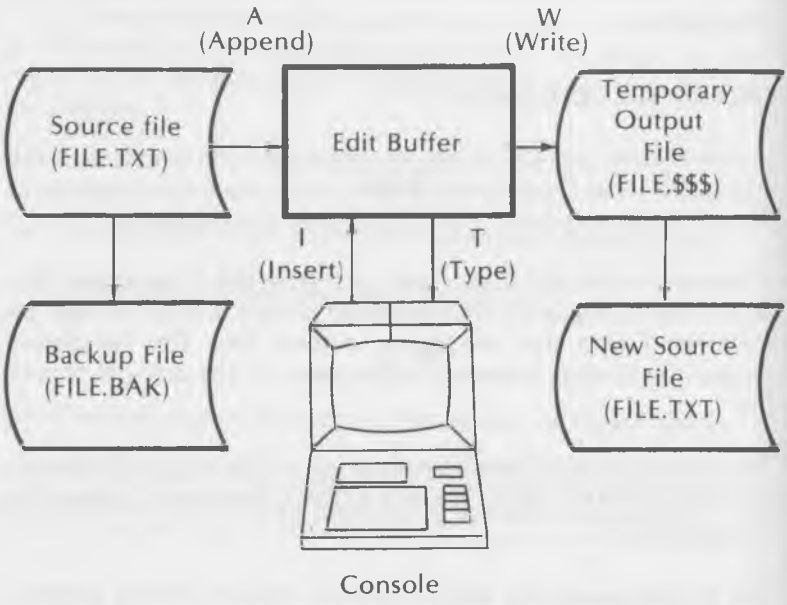
The Q command can also be used to end an editing session. When a Q command is issued, the editing session will be ended without any alterations to the source file. The original source file will remain unchanged on the diskette. The temporary output file will be erased. No backup file will be created. The CP/M prompt will be displayed after the Q command has been issued.

Before the Q command is actually effected, the user will be prompted as follows:

Q-(Y/N)?

This prompt is displayed to prevent the user from accidentally ending the edit session in error.

ILLUSTRATION 7-9. EFFECT OF THE E COMMAND ON THE EDITING SESSION



AN IMAGINARY EDITING SESSION

In the remainder of this chapter, we will illustrate the use of the ED commands with a practical example.

USING THE A COMMAND TO APPEND LINES TO THE BUFFER

When you are using ED to edit an existing text file, you should first execute the A command to bring text lines into the edit buffer. The format for the A command is as follows.

nA

The n specifies the number of lines to be appended from the source file. If no value is given for n, A will append one line. In other words, the default value for n is 1. If a # sign is used for n, the entire source file will be brought into the edit buffer (up to a maximum of 65535 lines).

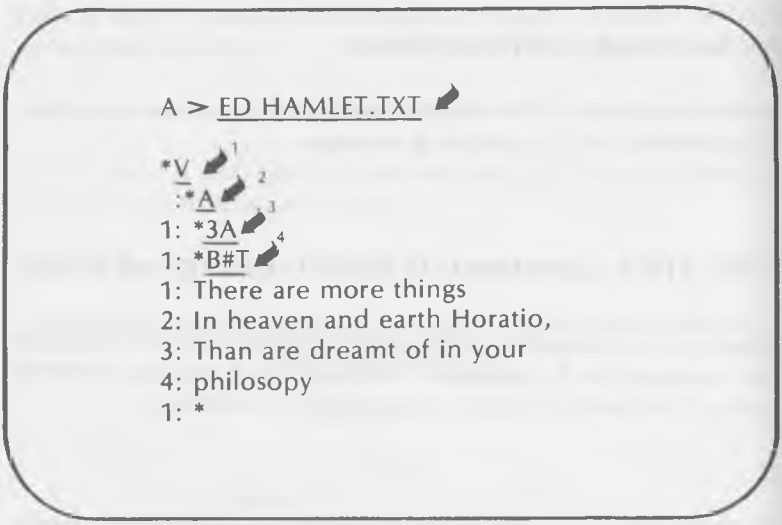
If 0 is used for n as shown below;

the A command will append lines so as to fill half of the buffer. This is especially useful when working with large files.

The 0A command is often used in conjunction with 0W to append and write out half of the edit buffer. This will be discussed in detail later in this chapter.

If you had used the A command to append only part of the source file, you can use the A command again to append more lines from the source file. The second A command will begin appending where the first A command left off.

The following example illustrates the use of the A command using HAMLET.TXT as the source file.



Notes:

1- Necessary only in Version 1.4; not needed in Version 2.2.

2- Appends one line to edit buffer from source file.

3- Appends next three lines--which is the remainder of the source file.

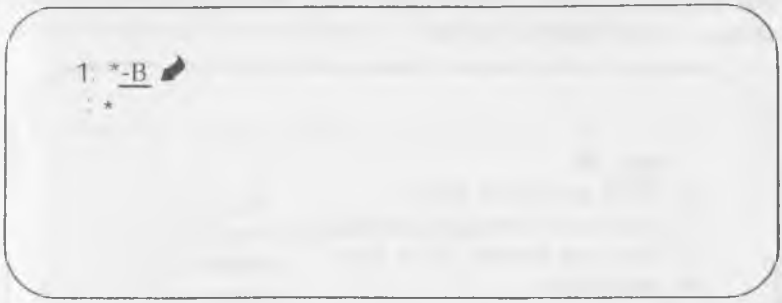
4--Goes to the beginning of the edit buffer and displays all lines.

Note that the #A command could have been used in place of the A and the 3A commands to bring the entire HAMLET.TXT into the edit buffer.

MOVING INSIDE THE EDIT BUFFER

Once you have appended text to the edit buffer, you can move around within the edit buffer, change any text within the edit buffer, and insert new text anywhere within the edit buffer. You can also append other lines of text from a special source file known as the library source file. We will discuss this concept later in this chapter.

Let's assume that we are still working with the lines of HAMLET.TXT in the edit buffer, and we wish to insert 2 more lines of text. First, the -B command must be issued to move to the end of the last line in the edit buffer. This is shown below.

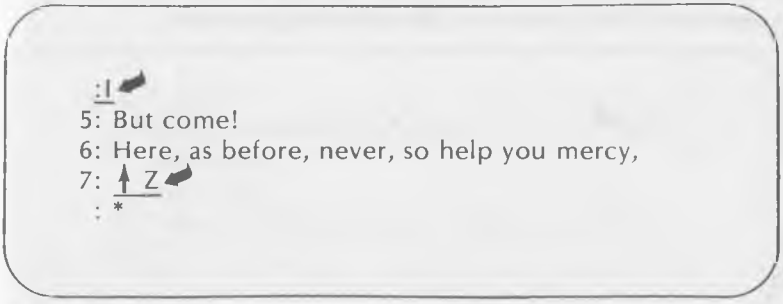


The end of the last line in the edit buffer actually is a Carriage Return (CR). In ASC II, the Carriage Return is a combination of the Return (the character which returns the carriage) and the line feed (the character which generates a new line).

The end of the edit buffer then is actually the beginning of the next new line. However, this new line does not have a line number until characters have been inserted into it via the I command.

Therefore, when you move to the end of the edit buffer with the -B command, only a colon(:) followed by the prompt will appear on the video display.

The example below shows how to insert text using the I command.



The ↑Z is used to end the insert mode when the operator has finished inserting text.

The B and T commands again can be used together to move to the beginning of the edit buffer, and then display it. This is shown in the example below.

```
: *B#T
1: There are more things
2: In heaven and earth Horatio
3: Than are dreamt of in your
4: philosophy.
5: But come!
6: Here, as before, never, so help you mercy,
1: *
```

You may also use line numbers to move to another line within the edit buffer. In CP/M 2.2 and later versions, line numbers are displayed automatically. In CP/M version 1.4, the V command is used to turn on the line number display, and –V turns off the line number display. Versions of CP/M earlier that 1.4 do not display line numbers.

In CP/M versions 1.4, 2.2, and later, you can move to a different line by typing the line number followed by a colon, as an ED command. This is shown in the following example.

```
1: *2:
2: *
```

A range of lines can also be specified. In such cases, the first line is typed in followed by two colons, and then the second line.

The following example illustrates the use of the T command with a range of line numbers to display those lines.

```
2: *3::6T
3: Than are dreamt of in your
4: philosophy.
5: But come!
6: Here, as before, never, so help you mercy,
3: *
```

Notice that the CP is moved to the beginning of the first line specified in the range (line 3) in the previous example. When a single line is specified, the CP is moved to the beginning of that line. When a range of lines is specified, the CP is moved to the beginning of that line in the range. The lines within the range specified are displayed by the T command. The CP will remain at the beginning of the first line in the range.

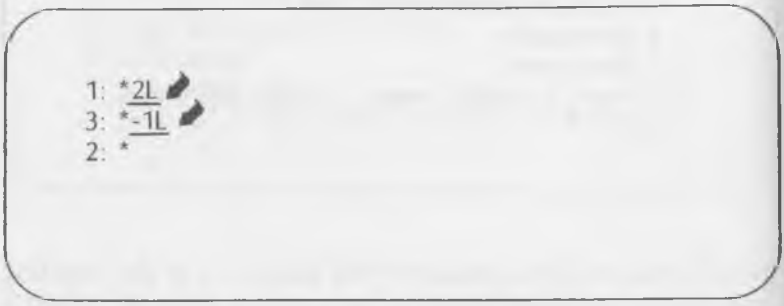
You can also select a range of lines from the current line to an ending line by specifying the ending line preceded by a colon. This is shown in the following example.

```
3: *:6T
3: Than are dreamt of in your
4: philosophy.
5: But come!
6: Here, as before, never, so help you mercy,
3: *
```

The L command can also be used to move inside the edit buffer. The format for the L command is as follows.

+nL

If the sign is +, the L command will move the CP forward. If the sign is negative (-), the L command will move the CP backwards. The n indicates the number of lines that the CP will be moved forwards or backwards. The L command moves the CP to the beginning of the line selected. The use of the L command is illustrated below.



If the command 0L is specified, the CP will be moved to the beginning of the current line.

CHANGING TEXT WITH ED

Text is changed in ED by moving the CP within the edit buffer, deleting the old text, and inserting the desired new text.

The K command is used to delete text. The format of the K command is shown below.

+nK

The K (kill) command will delete the current line if n is not specified. If n is specified, K will delete +n lines forward from (and including) the current line, or -n lines backwards from (and not including) the current line.

You can use the # sign with K to clear the edit buffer. The command;

```
-+#K
```

will clear out the current line and the next 65535 lines following it. The command;

```
-#K
```

will delete 65535 lines preceding, but not including the current line.

Remember, once lines have been deleted with the K command, they are gone forever. Unless these lines were previously saved in the source or back-up file, they will be unavailable. The example below illustrates the use of the K command with the lines of HAMLET.TXT in the edit buffer.

```
2: *1::6T
1: There are more things
2: In heaven and earth Horatio,
3: Than are dreamt of in your
4: philosophy.
5: But come!
6: Here, as before, never, so help you mercy,
1: *5:
5: *2K
 : *1::4T
1: There are more things
2: In heaven and earth Horatio,
3: Than are dreamt of in your
4: philosophy
1: *
```

In the preceding example, the T command causes lines 1 to 6 in the edit buffer to be displayed. The CP is then moved to line 5 where the 2K command is issued to delete the current line and the line following it.

The T command is then used to redisplay lines 1 to 4 in the edit buffer. These are now the only remaining lines in the edit buffer (assuming HAMLET.TXT in the edit buffer contained just the original 6 lines).

You can also delete individual characters in the edit buffer as opposed to entire lines, by using the D command. The format for the D command is given below.

+n D

If no n is specified, the D command will erase the current character (the character being pointed to by the CP). If +n is specified with D, the D command will erase +n characters following and including the CP. If -n is specified with the D command, the D command will erase -n characters preceding but not including the CP.

The D command is often used in conjunction with the I command to delete and then insert characters into the edit buffer. The I command takes one of two forms.

I (insertion) ↑Z
or
I ↵

When I is used with the carriage return (↵), a new line is automatically begun when the carriage return is inserted.

When I is used with ↑Z, no new line is started. When deleting and inserting individual characters or groups of characters, I with ↑Z is generally used to avoid the insertion of carriage returns.

The following example illustrates the use of the D and I commands on the lines of HAMLET.TXT (remember, lines 5 and 6 were erased with K) in the edit buffer.

```
1: *B#T
1: There are more things
2: In heaven and earth Horatio,
3: Than are dreamt of in your
4: philosophy.
1: *T
1: There are more things
1: *15C
1: *6DI ideas ↑Z 0LT
1: There are more ideas
1: *
```

In the previous example, we used the B#T command to display the lines in the edit buffer. We then used the T command to display the current line in the edit buffer. The command 15C was then issued to move the CP 15 characters to the right.

In the following line, the 6D command deletes the character pointed to by the CP as well as the following 5 characters. The D command moves the CP as it deletes characters.

We then use the I command to insert the new text, ''ideas''. The I command is terminated with Control Z. Note that the I command inserts characters before the CP. The I command also moves the CP as it inserts characters.

The 0L command was used to move the CP to the beginning of the current line and the T command was used to display that line.

SUBSTITUTING TEXT IN THE EDIT BUFFER

The F (or Find) command provides the CP/M user with an easy method for moving the CP. The format of the F command is given below.

F(text) (↑Z)

The text consists of those characters to be located by F. The F command will move the CP to the character that directly follows the last character found. The example below illustrates the use of the F command with the lines from HAMLET.TXT in the edit buffer.

```
1: F more ↵
1: *
```

The F command will move the CP to the position immediately after the last character found. The CP position after the preceding F command entry is shown below.

```
There are more ideas
               ↑
              CP
```

The F command is generally used in conjunction with the D and I commands to find and insert, find and delete, or find, delete, and insert characters. This is illustrated in the example below.

```
1: *Fideas ↑Z-5DIthings ↑Z0LT ↵

1: There are more things
1: *
```

The F command moves the CP to the end of 'ideas' in line 1. Notice that the F command is terminated with ↑Z rather than a carriage return, so that more commands may be issued.

The D command '-5D' is then issued to delete 'ideas' and move the cursor in position for the I command.

The I command is used to insert 'things'. The 0L command is then issued to move the CP to the beginning of the current line. The T command then displays the newly revised line.

The F command is ended with the carriage return (↵) if no other command is to be included on that line. If as in the example above, the F command if followed by additional commands, it is ended with ↑Z.

The F command can also be used in the format given below;

nF (text...) ↑Z

where n is a positive number which tells you to find the n th occurence of the character specified by (text...).

For example, in the following command,

3FHoratio↵

the CP will move when it finds the third occurence of 'Horatio' in the edit buffer.

The S command can also be used to find and substitute groups of characters. In fact, S (substitute) is much more widely used than the F and I command combination just discussed. In effect, the S command combines the actions of the F, D, and I commands into one.

The format for S is as follows;

nS (old text) ↑Z(new text) {↑Z / ↵}

The S command will search the edit buffer for the characters specified by 'old text' and will substitute this group of characters with the characters named in 'new text'.

You have the option of ending the new text string with either ↑Z or the carriage return (↵). The ↑Z is issued if another command is to be added to the S command. The carriage return ends the command.

The n indicates the number of times that 'newtext' is to be substituted for 'oldtext'. This substitution will be executed until the n th substitution has been made--or until the end of the edit buffer has been reached.

The default value for n is one. In other words, if no n is specified, the substitution will be made one time. If a value is used for n, the substitution will be made throughout the edit buffer. The substitution will be made throughout the edit buffer the number of times indicated.

The use of the S command is illustrated below.

```
1: *4T↵
1: There are more things
2: In heaven and earth Horatio,
3: Than are dreamt of in your
4: philosophy.
1: * S Horatio ↑Z Ophelia ↑Z B4T↵
1: There are more things
2: In heaven and earth Ophelia,
3: Than are dreamt of in your
4: philosophy
1: *
```

The above example will substitute, Ophelia for Horatio, when Horatio is encountered in the edit buffer. The B command moves the CP back to the beginning of the edit buffer, and the T command displays the first 4 lines of the buffer.

WRITING INDIVIDUAL LINES TO THE EDIT BUFFER

Generally, an edit session is ended with the E or H command. Both the E and the H write the entire edit buffer to the temporary output file, copy the remaining lines in the source

file to the output file, and rename the temporary output and source files. The E command terminates ED while the H command keeps ED active.

You can use the W command to write lines from the edit buffer to the output file without copying the entire edit buffer or the remaining source file to the output file.

The W command takes the following format.

nW

The n signifies the number of lines following and including the current line to be written to the output file. If a value is not specified for n, only the current line will be written to the output file.

The following example illustrates the use of the W command.

```
1: *B#T
1: There are more things
2: In heaven and earth Horatio,
3: Than are dreamt of in your
4: philosophy.
1: *2W
3: #T
3: Than are dreamt of in your
4: philosophy.
3: *#W
  : *
```

In the example above, the initial command, B#T, moves the CP to the beginning of the edit buffer and then displays the entire edit buffer.

The command, 2W, then writes the first two lines of the edit buffer to the temporary output file. After the 2W command is

issued, the remaining lines in the edit buffer are then displayed by the #T command.

The #W command outputs all remaining lines in the edit buffer to the temporary output file. Now that the edit buffer is empty, no line number is displayed as illustrated by the final line in our example, : *

Even though the entire edit buffer has been written to the temporary output file, the rest of the source file must also be written to the temporary output file. Also, the temporary output file (in our case HAMLET.$$$) must be renamed (HAMLET.TXT), and the source file (HAMLET.TXT) must also be renamed (HAMLET.BAK). You can use the E and the H commands to perform these tasks.

As you may remember, earlier we discussed the usage of the 0A command to append lines to the edit buffer from the source file so as to fill it halfway. The 0W command can also be issued to write half the lines in the edit buffer.

THE N COMMAND

The N command operates much like the F command, except that the N command does not stop at the end of the edit buffer. The N command will continue to append lines from the source file into the edit buffer until the specified group of characters has been found.

The format for the N command is as follows.

$$nN\ (\text{text}) \left\{ \begin{matrix} Z \\ \blacksquare \end{matrix} \right\}$$

The N command will start searching at the CP in the edit buffer for the n th occurence of 'text'. If N does not find the n th occurence of 'text' in the edit buffer, it will automatically append lines from the source file into the edit buffer until the n th occurence of 'text' is found (or until the source file is depleted).

If more commands are to follow, the N command should be ended with ↑Z. Otherwise, N should be ended with a carriage return.

Just as with the F command, the N command places the CP immediately after the last character of the n th occurence of the string specified by 'text'.

THE R COMMAND AND THE LIBRARY SOURCE FILE

A library source file is a file with the extension .LIB. For example, HAMLET2.LIB, would be considered a library source file.

A library source file can be used as a secondary source file--a file from which text can be inserted into the edit buffer. An obvious application of using a library source file is when you wish to merge two separate files into one final, combined file.

By specifying one file as the source file and the second file with the extension, LIB, you can merge the two files into the edit buffer to form a single file.

The R command will insert lines from the LIB file specified in that command. The format of the R command is as follows.

R filename

The filename in the above format is the name of the file with a LIB extension to be inserted into the edit buffer. When the R command is issued, that file will be inserted into the edit buffer beginning at the current CP until the end-of-file mark (↑Z) is reached in the LIB file.

THE X COMMAND AND THE HOLDING FILE

The X command is available in CP/M version 1.4 and later versions as well. The X command is used to transfer lines from the edit buffer to the holding file. The holding file is named, X$$$$$$$.LIB. This file only exists when ED is operational.

If ED is terminated, the holding file will be deleted with one exception. If ED is terminated with a warm boot (↑C), the

holding file will not be erased. However, once ED is again activated, the holding file will be deleted.

You may have noticed that the holding file is a LIB type file. Therefore, you can use the R command to transfer lines back into the edit buffer from the holding file.

The format of the X command is illustrated below.

nX

The X command copies the next n lines following the current line from the edit buffer to the temporary file, X$$$$$$$.LIB. The K command can be used to delete lines from the edit buffer after copying them.

The X command may be issued several times to accumulate lines in the X$$$$$$$.LIB file. The lines will be accumulated in the order in which they are transferred.

As mentioned previously, the R command can be used to retrieve lines from the X$$$$$$$.LIB file. In such cases, the R command is used alone without a filename as illustrated below.

: *R

When the R command is issued as illustrated above, all of the lines transferred previously into X$$$$$$$.LIB will be copied into the edit buffer following the CP. Note that the R command simply copies the lines from X$$$$$$$.LIB; it does not empty that file. In other words, you can use the R command to copy the X$$$$$$$.LIB file over and over. This is very useful where certain lines must be repeated over and over.

If you do wish to delete the contents of X$$$$$$$.LIB, you can do so by issuing the command 0X. This is a form of the X command where the prefix n has been set to zero.

In certain cases, you may wish to save the temporary file, X$$$$$$$.LIB, after ED has ended. In these cases, use ↑C to end the ED session. Then, rename the X$$$$$$$.LIB file so that it will not be erased the next time ED is executed.

THE J COMMAND

The J (juxtapose) command is used to rearrange groups of characters in the edit buffer. The format of the J command is as follows.

nJ string1 ↑Zstring 2 ↑Z string 3 { ↑Z }

The J command will begin searching after the CP in the edit buffer for string 1. If string 1 is located, the J command will insert string 2 immediately behind string 1, and then move the CP to the end of string 2.

The J command will then search for string 3. If string 3 is located the J command will delete all characters between string 2 and string 3. The CP will be left pointing to the first character of string 3. If string 3 is not located, the J command will not delete any characters.

The J command will follow this pattern for the number of times specified in n--or until it has run through the entire edit buffer.

The use of the J command is illustrated in the example below.

```
1: *B#T
1: There are more things
2: In heaven and earth Horatio,
3: Than are dreamt of in your
4: philosophy
1: *1J earth ↑Z, ↑Z Than ↑Z B#T
2: There are more things
3: In heaven and earth, Than are dreamt of in your
4: philosophy.
2: *
```

In the preceding example, the J command finds 'earth', and then inserts string 2 (,) immediately following string 1. When string 3 is located (than in line 3), all characters between string 2 and string 3 are deleted (Horatio).

THE M COMMAND

The M (macro) command can be used to repeat a set of ED commands over and over. M takes the following format.

nM command string {↑Z}

M will execute each of the ED commands in 'command string' for the number of times indicated by n, as long as n is greater than 1. If n is equal to zero or one, the string of commands will be executed over and over until the end of the edit buffer has been reached, or until an error condition occurs.

ED ERROR CONDITIONS

Table 7-1 lists the various ED error indicators. When an error condition occurs in ED, the last character read by ED before the error occurence will be displayed along with one of the error indicators listed in Table 7-1.

The newer versions of CP/M display a complete error message rather than just an error symbol. This error message takes the following form;

BREAK error symbol AT ed command

where 'error symbol' is the ED error symbol and 'ed command' was the ED command being executed when the error occurred.

TABLE 7-1. ED ERROR INDICATORS

Indication	Meaning
?	ED does not recognize this command.
>	The edit buffer is full. The D, K, S, W, E, or H commands must be used to delete characters. Another source of this error may be a string that is too long. This error may have been the result of the use of the F, N, or S command.
#	The indicated command cannot be executed the number of times specified. An example would be when the edit buffer's end is reached before the command can be executed the specified number of times.
0	This occurs when the LIB file cannot be opened by the R command. Usually, this is caused by an incorrect LIB file name or no existing LIB files.

CHAPTER 8. INTERNAL OPERATION OF CP/M

INTRODUCTION

In this chapter, we will present an overview of CP/M's internal operation. We will discuss the various components of CP/M, what each component does, and the way in which each of these components interact. The way that the various CP/M modules are spread over memory will also be discussed. The organization of the file system will be given in detail.

Each of CP/M's three modules will then be discussed in detail with the commands that are related to each module. The adaptation of CP/M to differing hardware configurations will then be discussed. The chapter will conclude with an overview of MP/M's operation.

If you are only interested in learning CP/M operation, this chapter is not required reading. However, if you wish to know how CP/M operates internally, you should read this chapter.

CP/M OVERVIEW

CP/M consists of the following three modules.

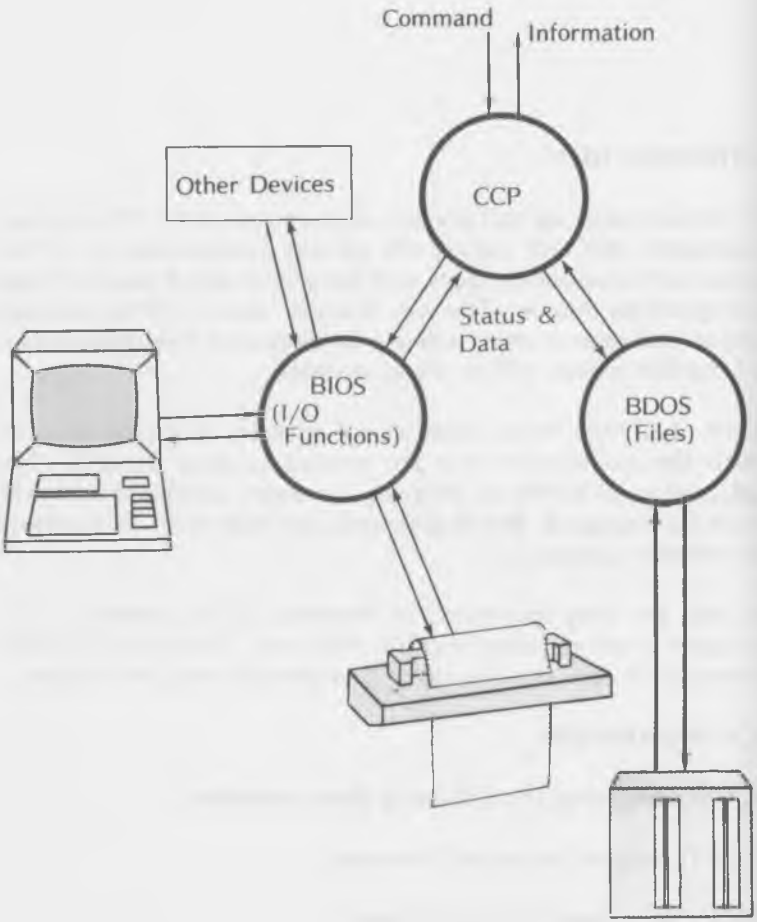
CCP (Console Command Processor)

BIOS (Basic Input/Output System)

BDOS (Basic Disk Operating System)

Illustration 8-1 shows the actions of these three modules on the system's physical devices and their interaction with each other.

ILLUSTRATION 8-1. CCP, BIOS, BDOS CONTROL



CCP's main function is to interpret the commands keyed in at the keyboard. CCP also performs some internal processing. As can be seen in Illustration 8-1, CCP draws upon BIOS and BDOS to perform its functions.

BIOS consists of routines which communicate with the various physical devices attached to the system. These routines are known as peripheral drivers. BIOS sends and receives data and status information between a physical device and CCP's debugger. BIOS is called by the CCP as pictured in Illustration 8-1.

BDOS, CP/M's disk operating system, consists of a large number of utility routines designed to manage the disk files. BDOS performs disk managing functions such as locating blocks of data, verifying that data, and allocating and releasing storage areas. BDOS performs virtually all of these tasks without the user being aware of its disk managing activities.

ALLOCATION OF MEMORY

As shown in Illustration 8-2, CP/M divides the available memory into separate zones for the system, Transient Program Area, CCP, BDOS, Bootstrap Leader, and BIOS.

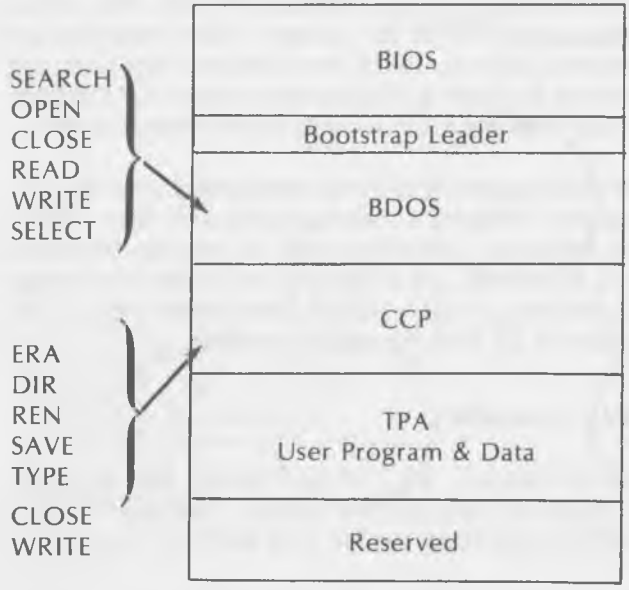
The first 256 memory locations are reserved for the system. These are known as page 0, and will be discussed later in detail.

The next memory area is known as the Transient Program Area or TPA. This is the area of memory available for program execution, and is the largest area of memory allocated.

The TPA begins at either address 0100H (hexadecimal) or 4300H (hexadecimal), depending upon the hardware configuration. If the computer being used has pre-stored programs in ROM being stored at the lower memory addresses, the TPA begins at 43000H. The TRS-80 is an example of such a hardware system.

If programs on ROM are not being stored at the lower addresses, the TPA begins at location 100H (or 256 decimal), Examples of computers where the TPA begins at the standard address of 100H are Cromemco, Altair, and Northstar.

ILLUSTRATION 8-2. CP/M MEMORY ALLOCATION



TPA extends upwards through memory to its upper limit--the base of cbase.

Cbase is the base of the CCP area of memory. The exact address of cbase differs according to the amount of memory available in the system.

CP/M assumes that the system has either 16, 32, 48, or 64K. If the system has 16K of memory, CP/M assumes that the memory address of cbase is 2900H. For every 16K of memory, CP/M adds 4000H to the address of cbase.

CP/M FILE STRUCTURE

Before you can fully comprehend the operation of BDOS and the CCP, you must understand the overall structure of the file system. To this end, we will examine the CP/M file structure.

The file can be defined as a logical unit containing information or programs. The function of any disk operating system, including BDOS, is to manage this logical unit on the physical storage device. In our case, the physical device is the diskette.

The diskette is organized into tracks and sectors as shown in Illustration 8-3. Information is recorded in concentric circles known as tracks. These tracks are in turn divided into sectors.

ILLUSTRATION 8-3. TRACKS — SECTORS

On an 8 inch diskette, each sector consists of 128 bytes. This is also known as a record. Each file on a diskette consists of a collection of records. It is impossible to keep a file's records in physical sequence on the diskette. These sectors containing a file's records are spread across the entire diskette's surface.

Logically, a list of some sort must be kept to keep track of the location of all sectors associated with a particular file. Different operating systems use different techniques for accomplishing this purpose. In CP/M, a file control block is used to keep track of all sectors belonging to a disk file. Each file control block has the capacity to describe up to 16K bytes of a file. Up to 16 total file control blocks can be used to keep track of a CP/M file.

In CP/M, a file is divided into units. A unit may contain anywhere from 0 to 128 records. Since each record (sector) contains 128 bytes, a unit may consist of from 0K to 16K (128 records x 128 bytes per record).

A CP/M file may contain up to 16 units. The largest CP/M file possible would contain 16 units of 16K per unit (or 256K). This capacity is slightly greater than the maximum capacity of a standard 8 inch diskette.

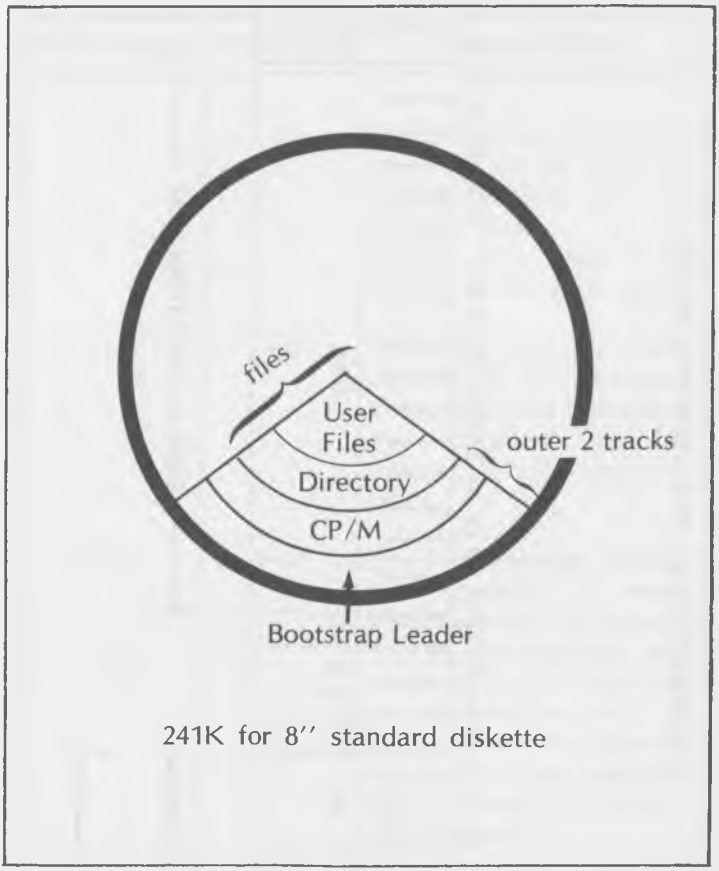
FILE CONTROL BLOCK

The file control block (FCB) consists of 33 bytes and is stored in the directory area. The FCB is actually a directory of the sector allocated to a particular file. When a file is accessed through CP/M, its FCB is brought from the diskette to the TPA so that it can be quickly accessed by the operating system.

The structure of the file control block is pictured in Illustration 8-5 and described in Table 8-1.

The relationship of file control blocks to FDOS and CCP will be discussed in more detail later in this chapter.

ILLUSTRATION 8-4. DISK SPACE UTILIZATION



The outer two* tracks of the diskette are used to store the CP/M system. The remainder of the diskette is used to store the files and file directory.

*5¼" diskettes have the outer 3 tracks reserved for the CP/M system.

ILLUSTRATION 8-5. FILE CONTROL BLOCK

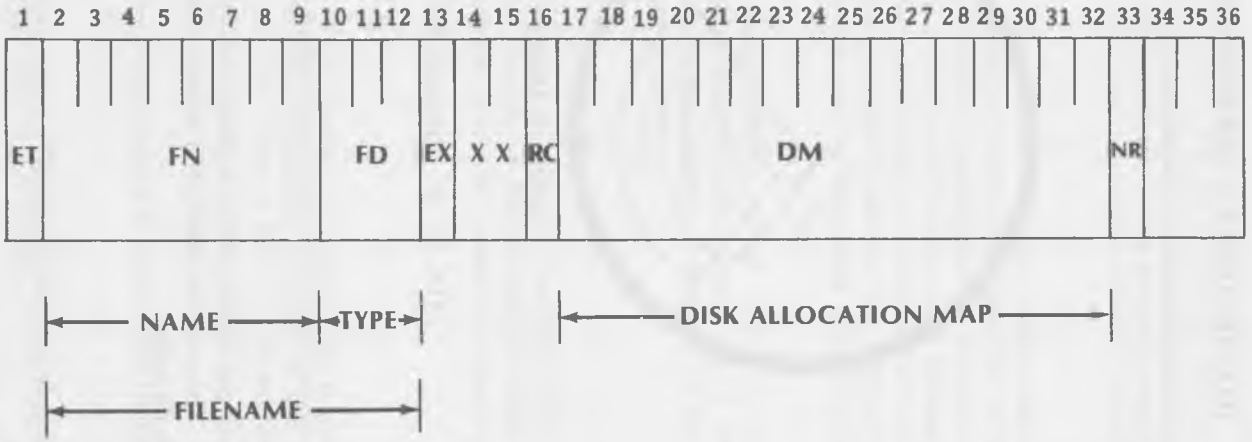


TABLE 8-1. FILE CONTROL BLOCK FIELD DESCRIPTIONS

FIELD NAME	POSITION	DESCRIPTION
ET	0	This is the entry type and is assumed to be zero in CP/M version 1.4. In CP/M version 2.2 or MP/M, this position is known as the drive code.
FN	1-8	This is the file name which may consist of up to eight characters. Any characters not supplied will be entered as the ASC II blank.
FD	9-11	This is the file type. The file type consists of three alphanumeric characters. Again, characters not supplied by the user will be entered as ASC II blanks. When the diskette directory is listed, both the file name and file type will be displayed.
EX	12	This is the file extent which is normally zero.
XX	13-14	This field is not used and is normally set to zero.

TABLE 8-1. FILE CONTROL BLOCK FIELD DESCRIPTIONS (CONT.)

FIELD NAME	POSITION	DESCRIPTION
RC	15	This field contains the record count. A file may contain from 0 to 128 records. The record count is also referred to as the current extent size.
DM	16-31	These positions are known as the disk allocation map. This section keeps track of which disk sectors are used by this file.
NR	32	This field is known as the next record, and contains the next record to read or write. This field is normally zero. In CP/M 2.2, this field is known as CR (Current Record).
R0-1-2	33-35	R 0-1-2 are used only in CP/M 2.2 for random accesses and contains the optimal random access number.

CP/M SYSTEM OPERATION OVERVIEW

As shown in Illustration 8-2, the CP/M operating system remains in the lower end of memory (addresses 000H to 100H). The gateway to the CP/M operating system is memory address 005H. The gateway is a single fixed point in memory where control is transferred back to CP/M.

For example, if a program is loaded into the TPA (beginning at 100H), the program can "jump" back to the operating system by executing a JMP instruction to address 005H.

From an operator's point of view, CP/M works as outlined below.

1. The CCP (Control Command Processor) displays the system prompt and then waits for a command.

2. The command line keyed in is transmitted via BIOS (Basic Input/Output System) to the CCP buffer.

3. When the command (with any associated filename) is received, CCP will execute that command, if it is a built-in command (a built-in command is one that is permanently in the system's memory).

4. If the command received by CCP is not built-in, CCP assumes the command to be a transient program with the .COM filename extension. An example would be STAT.COM. The CCP will request that BDOS find the file and read it onto the TPA. Finally, the CCP will create a file control block for the file or files named after the command.

For example, in the following command;

A > ERA SAMPLE.TXT ↵

a file control block would be constructed for SAMPLE.TXT.

5. The CCP now ends its control. This leaves the space in memory formerly used by the CCP free, allowing the TPA to temporarily expand and use this space.

6. BDOS now locates the specified file using the file control block created by the CCP in Step 4. At this point, the newly created file control block contains only the filename.

Each file on the diskette has its own file control block. All BDOS does is match the filename of the file control block created by CCP in Step 4 with the filenames from the file control blocks for the files on disk.

7. When BDOS finds the correct file on disk, it supplies data from that file's file control block on disk to the current file control block created by the CCP. Whenever the program accesses that file on the disk, BDOS updates the information on the current file control block. When the file is closed by the program, BDOS copies the current file control block from the computer's memory onto the diskette. Both the file and that file's file control block are simultaneously updated.

FDOS & CCP IN OPERATION

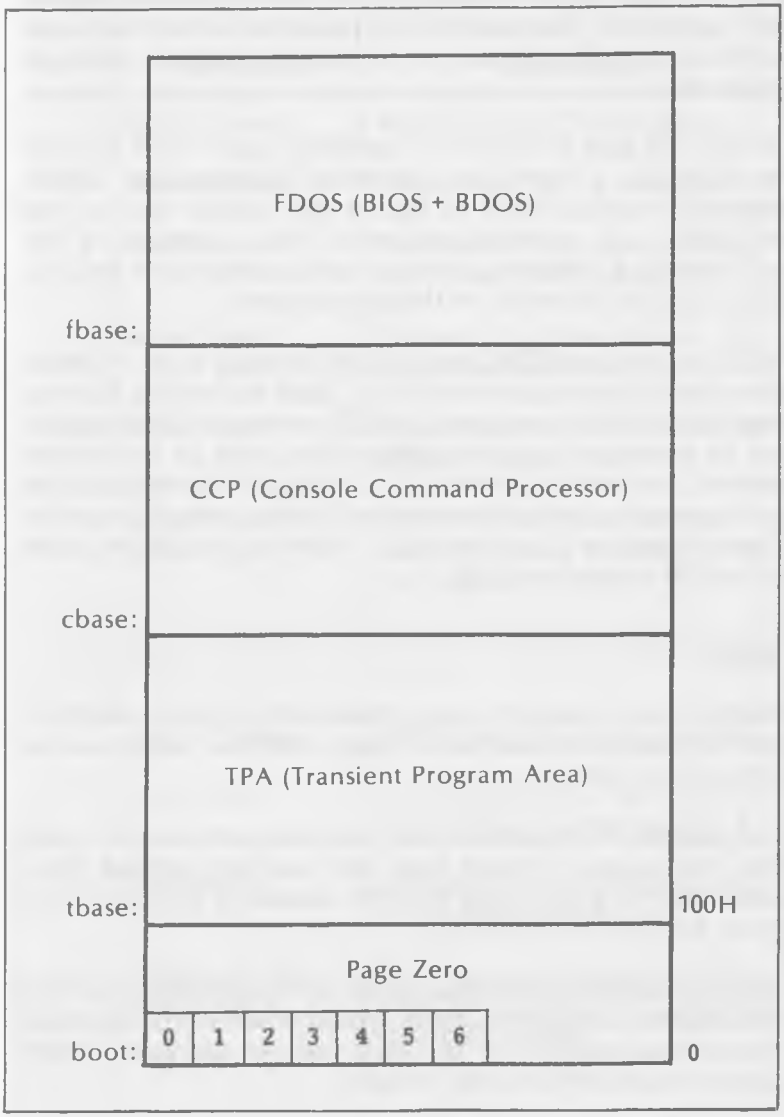
Illustration 8-6 portrays CP/M's memory map after CP/M has been loaded into the computer's main memory. Note the differences between Illustration 8-6 and Illustration 8-2. First of all, BIOS and BDOS have been combined as FDOS. Secondly, each of these four memory areas has a specified base; boot, tbase, cbase, and fbase.

The exact memory address locations for boot, tbase, cbase, and fbase depend upon the version of CP/M being used. However, these addresses are present in any CP/M system.

The boot is the location where machine code instructions are begun which perform a system restart. A program in TPA can transfer control back to CP/M by jumping to the boot.

The user program or transient command fills memory from tbase to cbase, and can extend into fbase if it will overwrite the CCP.

ILLUSTRATION 8-6. CP/M MEMORY MAP AFTER LOADING



BIOS

BIOS is essentially a message carrier allowing communication with peripherals. Examples of BIOS operations would be to send a character to the printer or to read a character from the keyboard.

BIOS is the part of the CP/M operating system that must be modified for a particular hardware configuration. Digital Research supplies BIOS so that it will operate on an Intel MDS-800 with standard peripherals that connect to the MDS-800. If a different hardware configuration is to be used, BIOS must be altered to fit this environment.

BIOS (as well as BDOS) are accessed through boot + 0005H (the principle entry point for FDOS). Table 8-2 lists the function numbers for BIOS operations. A BIOS command is communicated by sending a function number along with an information address. For instance, if the ASC II character A were to be sent to the console, the function number 2 (write console) would be placed in register C, and the ASC II value for A would be placed in the CPI register pair D,E.

BDOS

BDOS is also accessed by entry through FDOS (boot + 0005H). A BDOS operation is specified by using a function number and an information address.

For example, if a sequential read was to be performed on a disk file, the program should send the function number for a sequential read (20) along with the address of the file control block for the file to be read.

BDOS would then attempt to complete the specified function. If the read was completed, BDOS would return with a successful completion indicator. If the read was not successful, BDOS would return with an error indicator.

The BDOS function numbers along with their description are given in Table 8-3.

TABLE 8-2. BIOS FUNCTION NUMBERS

Function	Operation	Description
1	Read Console	Returns an ASC II character.
2	Write Console	Outputs an ASC II character.
3	Read Reader	Returns ASC II character from the reader device (RDR:).
4	Write Punch	Outputs a character to the punch device (PUN:).
5	Write List	Outputs an ASC II character to the list device (LST:).
*6	Direct Consol Input Output	Send 'FF' to receive character or status. Or send a character to the console.
7	Get I/O Status	Return byte with status of device.
8	Set I/O Status	Send byte with status to device.
9	Print Buffer	Send entire string--starting with the address and ending with $.
10	Read Buffer	Send address of read buffer and return with filled buffer.
11	Interrogate Console Ready	If the least significant bit of the byte is 1, then the console character is ready.

*CP/M 2.2 and MP/M only

TABLE 8-3. BDOS FUNCTION NUMBERS

Function	Operation	Description
12	Lift Disk Head (CP/M 1.4)	This function lifts the head from the current disk.
12	Return Version Number (CP/M 2.2 & MP/M)	This function returns the version number of your CP/M system to provide version independent programming.
13	Reset Disk System	This function initializes BDOS, resets the read/write state for all disk, selects Drive A, and sets the default DMA address to boot + 0080H. This address is used by programs to allow disk changes without a ↑C or system restart.
14	Select Disk	A disk drive is selected as the current drive for succeeding file operations (1 = A; 2 = B; 3 = C; etc.).

TABLE 8-3. BDOS FUNCTION NUMBERS (CONT.)

Function	Operation	Description
15	Open File	If a file control block address is sent with this function number, BDOS will find a matching file control block in the directory area of the disk. BDOS will return with the correct directory code, which will indicate that the proper information had been copied to the file control block. This process permits later file access.
16	Close File	If a file control block address is sent with this function code, BDOS will write the updated directory information in the file control block on the disk.
17	File Search	If this function is sent with the address of the file control block that contains a filename, BDOS will search for the first match of that filename. BDOS will return the address of the file control block that matches the file control block set up by CCP.
18	Search For Next Occurence	This function is used to search for the next occurrence of a filename. This function is used after 17. The address of the next file control block on disk will be returned.

TABLE 8-3. BDOS FUNCTION NUMBERS (CONT.)

Function	Operation	Description
19	Delete File	If this function is sent with the address of a file control block containing a filename, BDOS will delete that file from the disk.
20	Sequential Read	If the file had previously been opened or created by the Make File function, the Sequential Read function will read the next 128 bytes (record) into memory at the current DMA address. The function returns an indicator for a successful read, an end of file, or unwritten data during random access.
21	Sequential Write	If the file had been previously opened or created by the Make File function, the Sequential Write function will write the next 128 bytes (beginning at the current DMA address) to the file named by the file control block. The Sequential Write will overwrite any existing data.

TABLE 8-3. BDOS FUNCTION NUMBERS (CONT.)

Function	Operation	Description
22	Make File	The Make File function creates a new file as well as opening it. If you send the address of a file control block with a new filename, the Make File function will create the file and initialize its file control block in main memory as well as on disk.
23	Rename File	If the address of the file control block is sent with this function, BDOS will rename the filename area of the block and record it on disk.
24	Return Log-In Vector.	This function is used to determine which disk drives are on line.
25	Return Current Disk*	This function returns a number that corresponds to the letter of the disk drive currently selected.

*CP/M 2.2 and MP/M only

TABLE 8-3. BDOS FUNCTION NUMBERS (CONT.)

Function	Operation	Description
26	Set DMA Address	The DMA address is the Direct Memory Address. This is the address where the file pointer stopped following a read or write operation. This function sets the DMA to a different value so that data records can be found elsewhere in memory. After a cold start, a warm start, or a disk reset, the DMA is set to boot + 0080H.
27	Get Allocation Vector Address	An allocation vector is maintained in the main memory for each on-line drive. This function is used to return the address of the vector for the current drive. This vector can be used by a program (example-STAT) to determine the remaining disk storage space.
28	Write-Protect Disk	This function will temporarily protect a disk from being written over. An error message will be generated on any attempted write to the disk.
29	Get Read-Only Vector	The Get Read-Only Vector function returns a function which indicates which drives are read-only.

TABLE 8-3. BDOS FUNCTION NUMBERS (CONT.)

Function	Operation	Description
30	Set File Attributes	The Set File Attributes function makes it possible to either set or clear read-only and system attributes attached to files.
31	Get Disk Parameter Block Address	This function returns the BIOS disk parameter block address. This address can be used to compute disk space and to change disk parameters.
32	Get or Set User Code*	This function can be used to determine which user code is currently active or to change the user code currently active.
33	Random Read	The Random Read uses the Random Record Number field in the file control block to select a record number and read that record. After the record indicated has been read, this function will set the DMA at the record read. The NR (Next Record) field in the file control block will not be incremented as in a Sequential Read.

*CP/M version 2.2 & MP/M

TABLE 8-3. BDOS FUNCTION NUMBERS

Function	Operation	Description
34	Random Write	As with the Random Read, the Random Write uses the Random Record Number field to select a record. This record then writes the data from the current DMA to the disk. Again, the Next Record field is not advanced.
35	Compute File Size	When sent with a file control block address, this function returns the record address of the end of that file. This gives the virtual size of the file. If a file is sequential, the virtual size equals the file's physical size. If the file is random, the virtual size includes blank spaces which are the result of random write operations.
36	Set Random Record Position	This function gives the random record position after a number of sequential read or write operations. This function can be used to make an initial sequential search of a file before random read or write operations. If can also be used to switch from sequential to random operations.

ALTERING CP/M

As we mentioned before, Digital Research supplies a form of CP/M that operates on the Intel MDS-800 microcomputer system. Other forms of CP/M are available that run on other hardware systems. Generally, these versions of CP/M require no alterations.

However, you may someday find yourself in a situation where you already have CP/M installed on your system, but wish to change your input or output devices. In such cases, you will have to alter the BIOS module of CP/M. If you have MP/M, you will have to alter both the BIOS and XIOS modules.

This alteration is known as patching. Patching simply means inserting the new input/output routines required by your new hardware configuration.

Patching is a simple task. However, the specific routines to be changed depend upon the hardware changes involved.

Because of the numerous possibilities, we cannot present patching instructions in this book. Refer to your CP/M verson's CP/M Alteration Guide for specific patching instructions.

ALTERING MEMORY SIZE

Another alteration of a hardware configuration is an increase or decrease in memory size. When memory size is changed (by installing or removing memory boards), CP/M must also be altered. The MOVCPM program can be used to reconfigure CP/M for any size memory. In such cases, MOVCPM takes the format outlined below;

$$\text{MOVCPM}\left\{\begin{matrix}*\\nn\end{matrix}\right\}\ (*)$$

The argument (nn or *) informs CP/M of the amount of memory that the new CP/M system should manage. If nn is set to 32, the CP/M will be configured for 32K of RAM. If nn is set to 48, CP/M will be configured for 48K of RAM, ect. If the argument is

left blank, or if an asterisk (*) is specified as the argument, MOVCPM will configure the CP/M system to use all of the available RAM in the host computer. For example, if the host computer has 64K of RAM, and the following command is issued;

MOVCPM**

CP/M will be configured to manage 64K of RAM.

The second asterisk is optional. If a second asterisk is supplied, MOVCPM will keep the newly-configured CP/M system in memory. The CP/M system can then be saved with a SAVE command, or written out to a diskette with the SYSGEN command.

If only a single asterisk is used as illustrated below;

MOVCPM*

MOVCPM will boot the new system without writing it on disk.

Some examples of the use of MOVCPM are listed in Table 8-4.

When the command, MOVCPM** ↵, is executed, the following prompt will be displayed.

READY FOR SYSGEN

When a command of the form, MOVCPM nn* ↵, is executed, the following prompt will be displayed.

SAVE nn CPM nn.COM

The nn's stand for the number of kilobytes of RAM.

INSTALLING MP/M

An MP/M system must be installed from an existing CP/M system. MP/M is brought up from CP/M by executing

MPMLDR.COM. This command loads the MP/M system into memory from the diskette. The MP/M system is held as MPM.SYS on the diskette.

Before the MP/M system can be loaded into memory with MPMLDR.COM, it must be generated with the program GENSYS.COM. The GENSYS.COM program operates under CP/M and is provided with MP/M.

The GENSYS program asks the operator questions and uses his answers to build MPM.SYS. The following illustrates an example run of GENSYS.

```
A > GENSYS

MP/M 1.0 SYSTEM GENERATION
TOP PAGE OF MEMORY = 0
NUMBER OF CONSOLES = 3
BREAKPOINT RST# = 5
ALLOCATE USER STACKS FOR SYSTEM CALLS? (Y/N) Y
MEMORY SEGMENT BASES, (FF TERMINATES LIST)
  :00,0
  :00,1
  :00,2
  :FF
SELECT RESIDENT SYSTEM PROCESSES: (Y/N) Y

TIME      ? Y
SCHED     ? Y
ATTACH    ? Y
SPOOL     ? Y
MPMSTAT   ? Y
A >
```

TABLE 8-4. MOVCPM

COMMAND	DESCRIPTION
A > MOVCPM ↵	CP/M is reconfigured so as to manage all available RAM in the host computer. The system will then be executed without being written to a diskette.
A > MOVCPM** ↵	CP/M is reconfigured to manage all of the host computer's RAM. The system is left in memory in preparation for a SYSGEN or SAVE command.
A > MOVCPM 48 ↵	This command reconfigures CP/M to manage 48K of RAM. CP/M is executed without being written to disk.
A > MOVCPM 48* ↵	CP/M is reconfigured to manage 48K of RAM. The system is left in memory in preparation for a SAVE or SYSGEN command.

The second and fourth examples use the optional second asterisk to leave the newly configured CP/M in memory, so that a SAVE command can be used to write the contents of memory out to disk or a SYSGEN command can be used to create a system diskette.

In the first line of this prompt sequence, the operator activates the GENSYS program. The second line is the system response.

The third line is the prompt for the top page of memory. In answer to this prompt, the operator should enter the top page of his system's RAM memory. If the operator enters a zero (as he did in our example), MPMLDR will determine the top page of RAM memory when it loads MP/M.

The fourth line prompts the operator to enter the number of consoles to be attached to the MP/M system. Up to 16 consoles may be used in an MP/M 1.0 system. Each console requires one page or 256 bytes of memory.

The fifth line of dialogue prompts the operator to enter the Breakpoint RST# (of breakpoint restart number). This sets the number of break points for the DDT debugger programs.

The sixth line allocates user stacks for system calls. If the answer is yes, stack space will be allocated so that CP/M .COM files can be used as commands in MP/M. This prompt is necessary because an MP/M command requires more stack space than is required in CP/M.

The seventh line prompts you to form one to eight user memory segments. Each user segment area has the same address space but different bank numbers. The first memory location that you specify should be your actual initial RAM location. This will be 0000H (unless you have ROM memory beginning at 000H). A comma will follow this location, followed by the bank number. Key in FF when you wish to end this list.

The twelfth line in our example, prompts the operator to select resident system processes. If the operator does not desire to select resident system processes, he should answer with a Y to this prompt. He should also respond with a Y for each specific process he wishes as resident rather than transient.

A resident process is a program that lies within the operating system much like a built-in CP/M command. Resident Processes are not displayed in the directory.

Once MPM.SYS has been generated, the system prompt (A>) will appear. The operator can then use MPMLDR.COM to load MPM.SYS into memory and execute it. MPMLDR.COM requires no dialogue. It is executed as a command.

ALTERING MP/M

Like CP/M, MP/M is designed to run on the Intel MDS-800 microcomputer. Modified versions of MP/M are available for other hardware environments from software vendors.

As with CP/M, when MP/M is used for a hardware environment other than the Intel MDS-800, XIOS (MP/M's BIOS--known as extended BIOS) must be altered. Also, MPMLDR.COM must be altered to load and execute MP/M. Specific instructions on altering XIOS and MPMLDR.COM for differing hardware configurations are available in Digital Research's documentation for MP/M.

CHAPTER 9. CP/M AND MP/M REFERENCE GUIDE

INTRODUCTION

This chapter has been designed as a reference guide to the various CP/M and MP/M commands and utility programs. Each command keyword is used to reference that command. These keywords are presented in alphabetical order.

FORMAT

In this chapter, we have used a specialized format to present each command. The command's keyword is illustrated first. To the right of that illustration is the Version Checklist, which lists the three versions of CP/M and MP/M as follows.

■ CP/M version 1.4
□ CP/M version 2.2
□ MP/M version 1.0

If square is blackened in the checklist, the command being described applies to that version of CP/M or MP/M. If a square is left white, the command does not apply to that version of CP/M or MP/M.

In the preceding example, the command being described would apply to CP/M 1.4, but would not apply to CP/M 2.2 or MP/M 1.0.

The first line underneath the illustration of the command's keyword is a short description of the command's purpose.

The second line describes the nature of the command. For example, the ATTACH command may either be a resident process or may exist as a .PRL file.

The third line illustrates the following formats that may be used when executing the command or program. If an argument is not enclosed in parentheses, that argument is required. If an argument is enclosed in parentheses, it is optional.

When arguments are enclosed in brackets, the user may choose between the items enclosed by the brackets. At least one of the arguments in the brackets is required, unless the argument is enclosed in parentheses. In such cases, the argument is optional.

After the format, a description of the arguments is given followed by a description of what the command does. This is followed by a narrative description of how the command is used and some practical examples. In these examples, everything to be keyed in by the operator is underlined. The symbol ↵ stands for Return or Carriage Return. The symbol ↑ stands for the Control key.

ABORT

☐ CP/M version 1.4
☐ CP/M version 2.2
☒ MP/M version 1.0

Purpose: Aborts a program that has been scheduled with SCHED.

Command Nature: ABORT.COM or ABORT.PRL

Formats: 1. ABORT program name console number.

Arguments: program name: The name of the program which is to be aborted.

console number: The console number is the number of the console from which the program was initiated.

Command Description: The ABORT command stops the execution of the program specified in its argument. This command should be used with care as it may be used by any user to stop any program which was started at any console.

Command Usage: To stop a program, you must specify the program's filename and the console where it was initiated.

Examples: 3A > ABORT PROGRAMA 1 ↵

In the above case, PROGRAMA was started at console 1. PROGRAMA will be halted by the above command.

ASM

- ☒ CP/M version 1.4
- ☒ CP/M version 2.2
- ☐ MP/M version 1.0

Purpose: Assemble a file.

Command Nature: ASM.COM (supplied with CP/M or MP/M)

Formats:
1. ASM filename
2. ASM filename (.shp)

Arguments: filename: refers to the name of a source file (with the extension .ASM. The .ASM extension need not be specified in the command, as ASM automatically searches for a filename with the extension .ASM.

(.shp): are three optional letters that may be specified after the filename. The s is the letter for the disk drive that contains the .ASM source file referred to in the command. The h is the letter of the drive that is to receive the .HEX file created by ASM. If the letter Z is specified for h, the .HEX file will not be created. The p is the letter of the drive that is to receive the .PRN file created by ASM. If X is used in this position, the .PRN file will be sent to the video display. If a Z is specified, ASM will not create the .PRN file.

Command Description:

The assembler program (ASM.COM) is used to turn an assembly language source file written in 8080 or Z-80 code into a machine code file of type .HEX. The .HEX file can then be loaded into the system as a transient command.

ASM will also create a listing file with a .PRN extension that contains the assembly language source lines with error flags and hexadecimal notation.

Command Usage:

If the .ASM source file is on your current disk and you wish to create .HEX and .PRN files on your current disk, use format 1.

Otherwise, you must use format 2. The letter of the drive containing the source file must be specified in s. The drive to receive the .HEX file must be specified in h. The drive to receive the .PRN file must be specified in p.

If you specify the letter Z for h, the ASM command will skip creating the .HEX file and will only create the .PRN file.

If you specify a Z for p, then ASM will only create the .HEX file. ASM will skip creating the .PRN file.

If you wish ASM to only send the .PRN file to the terminal and not save it on disk, then you should specify the letter X for p.

In both of the formats for ASM, the command translates the assembly language source code into Intel hexadecimal notation, which is machine or binary code. When ASM discovers an error in the source file, it displays the line which is in error, along with an error code.

Examples:

A > ASM PROGRAMA ↵

The ASM command will be executed on PROGRAMA which is in the current line.

A > ASM PROGRAMB.ABX ↵

The above command executes ASM on PROGRAMB in Drive A. The newly created PROGRAMB.HEX is placed on Drive B, and PROGRAMB.PRN is sent to the terminal for display only and is not saved on disk.

ATTACH

☐ CP/M version 1.4
☐ CP/M version 2.2
▨ MP/M version 1.0

Purpose: Attach a console to a detached program.

Command Nature ATTACH.PRL or Resident Process.

Format: ATTACH program name.

Argument: Program name: refers to the filename for the detached program that was attached to the console when the ATTACH command was executed.

Command Description: The ATTACH program attaches a detached program to the console. That detached program must have been previously detached from that same console. A program can be detached from a console while running by pressing ↑D. When ↑D is pressed, the console will automatically attach the next waiting process.

Command Usage: If the ATTACH command is a resident MP/M system process or if ATTACH.PRL is accessible on a disk, the ATTACH program can be executed as a command when used with a program name as an argument.

Example:

```
3A > ATTACH PROGRAMA.PRL
```

In the example above PROGRAMA.PRL takes over the console.

CONSOLE

☐ CP/M version 1.4
☐ CP/M version 2.2
■ MP/M version 1.0

Purpose: To display the number of the terminal currently executing the CONSOLE command.

Command Nature: CONSOLE.COM or CONSOLE.PRL

Format: CONSOLE

Command Description: After the Console command is typed at the terminal, the command returns with the console number of the terminal being used. The Console command is often used in determining which console has detached programs waiting for it.

Example:

```
0A > CONSOLE↵
Console = 1
0A >
```

In the above example, the console being used is console 1.

DDT

- CP/M version 1.4
- CP/M version 2.2
- MP/M version 1.0

Purpose: DDT (debug) is used to load, alter, and test programs.

Command Nature: DDT.COM or DDT.PRL

Format: DDT (filename)

Argument: Filename: This is an optional argument. The file specified must be a .COM or .HEX file. The filename extension must be specified in the command.

Command Description: DDT first replaces the CCP in memory and then loads the file specified into TPA (Transient Program Area). If a filename is not specified as the argument, DDT occupies the TPA and waits for a file to be input into memory.

DDT has its own set of commands for inserting values, displaying memory locations, saving comments, setting breakpoints, and various other debugging functions.

DDT also displays the address following the final address in the program being debugged. This is known as the NEXT address. Finally, DDT displays the PC (program counter).

Command Usage: DDT must exist as a program on an accessible disk for it to be executed. DDT is ended with the command, G0.

Example:

A > DDT PROGRAM.HEX ↵

In the above example, PROGRAM.HEX will be loaded into TPA.

DIR

- ☐ CP/M version 1.4
- ▩ CP/M version 2.2
- ▩ MP/M version 1.0

Purpose: Displays a list of the filenames in the current disk drive's directory.

Command Nature: In CP/M, DIR is a built-in command. In MP/M, it is referred to as DIR.COM or DIR.PRL.

Format: DIR (d:) [filename / filename match]

Arguments: Filename: This is an optional argument that tells DIR to find just that file specified. A drive can be specified.

Filename match: This is also an optional argument for DIR. The files which match will be located and listed by DIR. A drive can be specified.

Command Description: If a filename or a filename match is specified as the argument, DIR will display only the filename specified or those filenames which match with the filename match specified.

If a drive is specified with the filename or filename match, DIR will search that drive (A:, B:, C:, ...) for the specified filenames.

If a filename or filename match is not specified, DIR will list all filenames in the current file. Only files with the $DIR system attribute will be listed. Those files with the $SYS attribute will not be listed.

In MP/M 1.0 and CP/M, only those files in the current user area will be listed.

Command Usage:

DIR is a built-in command in CP/M version 1.4 and version 2.2. This means that DIR is part of the operating system of these versions and can be executed from any disk drive and user area.

DIR is supplied as DIR.PRL or DIR.COM in MP/M. DIR.PRL or DIR.COM must exist in the current drive unless another drive is specified as a prefix to DIR in the command. DIR must also be in the current user area for execution in MP/M.

Examples:

A > DIR

ED.COM	PIP.COM	DUMP.COM
LOAD.COM	BASIC.COM	STAT.COM
FILE2.TXT	CLIENT.TXT	FILE1.TXT

.
.
.
.
.
.
.

The above command displays all files on the current drive A, under CP/M version 1.4. If the above command was executed under CP/M version 2.2, all files with the $DIR attribute in Drive A, user area 0 would be displayed.

```
A > DIR *.TXT↵

FILE1.TXT
FILE2.TXT
CLIENT.TXT
INVOICES.TXT
.
.
.
.
.
.
```

The above command displays all files on the current Drive A with the file extension .TXT in CP/M version 1.4. In CP/M version 2.2, all files with the same filename extension .TXT and the $DIR attribute will be displayed, if said file is in user area 0 of Drive A.

DSKRESET

☐ CP/M version 1.4
☐ CP/M version 2.2
▨ MP/M version 1.0

Purpose: This command is used to notify other users in a multi-user system of a disk change.

Command Nature: DSKRESET.COM or DSKRESET.PRL

Format: DSKRESET

Command The following message is sent to the other terminals in a multi-user system when DSKRESET is executed.

Confirm reset disk system (Y/N) ?Y

If any terminal user responds with a N to the preceding prompt, the request to change the disk will not be allowed.

A user in a multi-user system should always use DSKRESET to communicate the fact that he or she intends to change a disk. If another user is accessing or updating files on the disk being changed, problems could be entailed.

Example:

```
3A > DSKRESET
Confirm reset disk system (Y/N) ?Y
```

The preceding message will appear at every terminal which is hooked up to the system.

DUMP

- CP/M version 1.4
- CP/M version 2.2
- MP/M version 1.0

Purpose: The Dump command dumps the file specified as its argument to a terminal. The contents of the file are displayed in hexadecimal form.

Command Nature: DUMP.COM

Format: DUMP filename

Arguments: filename: The name and extension of any disk file.

Command Description The DUMP command displays the contents of the disk file specified in hexadecimal form on the terminal. Sixteen bytes are listed at a time. Each line's absolute byte address is listed to the left.

Command Usage: The DUMP program is executed as a command when entered with a filename and its extension. If DUMP.COM is not on your current drive, precede the command with the letter of the drive containing the file.

Example:

A>DUMP FILE1.HEX ↵

The example above dumps FILE1.HEX onto the terminal.

ED

- CP/M version 1.4
- CP/M version 2.2
- MP/M version 1.0

Purpose: The purpose of ED is to edit a text file.

Command Nature: ED.COM or ED.PRL

Format: ED filename

Argument: filename: This is the name of the file that is to be edited. The file must be a .TXT file

Description: ED automatically creates an edit buffer in which the text file can be modified. The first step undertaken by ED is to erase any .BAK file for the filename specified. The user may then append the text file to the edit buffer where it may be modified. Text from LIB (Library) files may be inserted into the edit buffer. Text may be output from the edit buffer into a temporary output file. When ED is ended, ED will update the original (source) file and create a back-up file of that original source file.

Command Usage: ED.COM or ED.PRL must be on an accessible disk. Details of ED usage are described in Chapter 7.

Examples: A>ED TEXT.TXT ↵

The file (TEXT.TXT) specified can now be created or modified by the editor.

ERA

- CP/M version 1.4
- CP/M version 2.2
- MP/M version 1.0

Purpose: The purpose of ERA is to erase one or more files from a disk or diskette.

Format: ERA { filename | filename match }

Arguments: filename: The filename argument is provided with the ERA command to specify which file is to be erased. The filename's extension must be included. A drive letter may also be specified.

filename match: When a filename match is used as an argument for ERA, several files may be erased simultaneously. If the filename match, *.*, is used with ERA, all files on the current drive will be erased in version 1.4, or all files in the current drive and current user area will be erased in version 2.2 or MP/M.

Command Description: The ERA command erases any filenames specified by the argument. The only exceptions are drives that are specified read-only or files with the $R/O file attribute. In these cases, the file will not be erased.

If ERA does not find the filename specified, the following message will be displayed.

No File

You can specify a drive letter as part of a filename to erase a file on a non-current drive. For example, the following command;

```
A > ERA B:FILE1.TXT
```

will erase FILE1.TXT on Drive B.

Command Usage:

In CP/M versions 1.4 and 2.2, ERA is a built in command which may be executed from any drive. In MP/M, ERA exists as ERA.COM or ERA.PRL. ERA.COM or ERA.PRL must exist in the current drive or be referred to using a drive specifier as shown below.

```
0A > B:ERA FILE1.TXT
```

In CP/M 1.4, you can erase an entire diskette by specifying the filename match, *.*. To erase an entire disk or diskette in CP/M versions 2.2 or MP/M, you must use the ERA command with the filename match *.* for each user area. Be sure that no read-only files have been left which have not been deleted.

Examples:

```
A > ERA FILE1.TXT
```

FILE1.TXT is erased from Drive A.

```
A > ERA B:FILE2.TXT
```

FILE2.TXT is erased from Drive B.

```
2A > ERA *.*
```

All files (except those with $R/O file attributes) are erased from user area 2 of Drive A.

ERAQ

☐ CP/M version 1.4
☐ CP/M version 2.2
▩ MP/M version 1.0

Purpose ERAQ is used to erase one or more files from a disk or diskette.

Command Nature: ERAQ.COM or ERAQ.PRL

Format: ERAQ filename match

Argument: filename match: The filename match allows ERAQ to erase a set of files.

Command Description: The ERAQ command erases all those files that match its arguments, unless the disk has been specified as read-only or a file has the $R/O file attribute.

Command Usage: ERAQ differs from ERA in that the user is requested to confirm an erasure before the files are actually erased.

In MP/M, ERAQ is supplied as a command file (ERAQ.COM) for absolute memory or a relocatable file (ERAQ.PRL) for relocatable memory. ERAQ.COM or ERAQ.PRL must either exist in the current drive, or be referenced by their drive letter as illustrated below.

```
A > B:ERAQ*.*
```

If the filename match *.* is used as in the above example, all files (except those with the $R/O file attributes) will be erased.

Example:

```
0A>ERAQ FILE.* ↵
A: FILE.TXT? Y
A: FILE.INT? Y
```

The system will prompt the user to confirm the erasure of all files with the filename, FILE.*. If confirmed, each file will be erased.

GENHEX

- ☐ CP/M version 1.4
- ☐ CP/M version 2.2
- ▨ MP/M version 1.0

Purpose: GENHEX transforms a .COM file to a .HEX file.

Command Nature: GENHEX.COM or GENHEX.PRL

Format: GENHEX program name .COM offset

Arguments: program name: This refers to the name of the program. It must be of type .COM. The name of the program may be preceded by a disk identifier.

offset: This is the offset for the .HEX file that is to be generated.

Command Description: The GENHEX command outputs a file of type .HEX from a file of type .COM. The file is offset by the amount specified.

Command Usage: The GENHEX is generally used to generate a page-relocatable (.PRL) file using a GEN-MOD command. In these cases, the offset will either be 000H or 100H.

Example:

```
0A>GENHEX PROGA.COM 100 ↵
```

PROGA.HEX is generated from PROGA.COM and is offset by 100H bytes.

GENMOD

☐ CP/M version 1.4
☐ CP/M version 2.2
▨ MP/M version 1.0

Purpose: GENMOD creates a relocatable file from two concatenated .HEX files.

Command Nature: GENMOD.COM or GENMOD.PRL

Format: GENMOD filename.HEX filename.PRL ($bbbb)

Arguments: Filename.HEX: This .HEX file must contain two concatenated .HEX files, each offset from the other by 100H bytes.

Filename.PRL: This is the name of the .PRL file to be created from the two conatenated .HEX files. An .RSP extension could be substituted for .PRL if you wanted to create a resident system process.

$bbbb: This argument is optional. It specifies the number of bytes of additional memory for the program.

Command Description: GENMOD creates a .PRL (page relocatable) or .RSP (resident system process) from two concatenated .HEX files offset by 100H. When the optional argument $bbbb is given, GENMOD allocates the amount of additional memory specified in the argument for the program.

Command Usage: GENMOD is executed as GENMOD.COM in CP/M version 2.2. In MP/M, GENMOD must be executed as a .PRL file.

Example: 0A>GENMOD TEXT3.HEX TEXT4.PRL $1000

In the preceding example, GENMOD creates TEXT4.PRL from TEXT3.HEX with 1000H additional memory.

GENSYS

☐ CP/M version 1.4
☐ CP/M version 2.2
▩ MP/M version 1.0

Purpose: GENSYS generates an MP/M system from CP/M.

Command Nature: GENSYS.COM

Format: GENSYS

Command Description: The GENSYS program prompts the system operator with questions about the new MP/M system being generated. GENSYS then builds the file MPM.SYS to hold the system.

The MPMLDR.COM program is then used to load the newly generated MPM.SYS into memory.

GENSYS is also used to incorporate resident processes (.RSP files) into the system. GENSYS searches for files with the extension .RSP and then asks the user to select resident processes from a list of .RSP files.

Command Usage: GENSYS is executed from CP/M or MP/M. The operator must answer the various prompts with a response and by pressing the Return key.

Example:

```
2A>GENSYS

MP/M 1.0 System Generation

Top page of memory          =C0
Number of consoles          =3
Breakpoint RST#             =5
Allocate user stacks for system calls (Y/N) Y

Memory segment bases, (ff terminates list)
:00
:40
:60
:ff

Select Resident System Process: (Y/N)
TIME    ?Y
SPOOL   ?Y
SCHED   ?Y
ATTACH  ?Y
```

LOAD

- CP/M version 1.4
- CP/M version 2.2
- MP/M version 1.0

Purpose: The LOAD command loads a file into an area of memory where it can be executed. The LOAD command will also convert a .HEX file into an executable command with the extension .COM.

Command Nature: LOAD.COM

Format: LOAD filename

Argument: filename: The name of a file with a .HEX extension. The .HEX extension need not be specified in the command.

Command Description The LOAD command is used to convert a program in Intel hexadecimal format into an executable command file with an extension of type .COM.

Command Usage: The LOAD.COM program must be on an accessible disk for it to be executed. However, if you prefix the LOAD command with the letter of an alternative disk where it resides, LOAD.COM can be executed.

The file of type .COM which was created by the LOAD command can be loaded into TPA by simply typing that filename with the extension .COM.

Example: A > LOAD PROGRAM ↵

PROGRAM.HEX is transformed into PROGRAM.COM.

A > PROGRAM.COM ↵

PROGRAM.COM can now be executed as a command.

MOVCPM

- CP/M version 1.4
- CP/M version 2.2
- MP/M version 1.0

Purpose: The MOVCPM command is used to reconfigure a version of CP/M, so that it fits hardware with a different amount of RAM.

Command Nature: MOVCPM.COM

Format: MOVCPM $\left\{ \begin{matrix} * \\ bb \end{matrix} \right\}$ (*)

Arguments: bb: This is an optional argument which specifies the amount of memory for which CP/M is to be configured. For example, if 32 were specified for bb, CP/M would be configured for a system with 32K of RAM.

*: If only one asterisk is included in the command, MOVCPM will determine the total available RAM for the host computer, configuring CP/M for that size. If a second asterisk is specified, either after the first asterisk or after bb, MOVCPM will leave the new system in memory in preparation for a SYSGEN or a SAVE operation. If a second asterisk is not specified, MOVCPM will execute the newly configured system without recording it on disk.

Command Description: The MOVCPM program creates a copy of the system and then alters that system for operation in the new RAM size given in the argument bb or for the maximum RAM available in the host computer. If the second asterisk is supplied as in the following;

```
MOVCPM**
or
MOVCPM 64*
```

MOVCPM will leave the newly generated system in TPA. The SAVE or SYSGEN commands can later be used to record the new version on disk. If the second asterisk is not specified as in the examples below:

```
MOVCPM*
MOVCPM 64
```

the newly configured system will be executed but not recorded.

Command Usage: MOVCPM is generally used to prepare a revised CP/M or MP/M system for a new hardware configuration.

Examples:

A > MOVCPM** ↵

The preceding example constructs a version of CP/M for the maximum available memory. This system is left in memory, so that it can be saved on disk via the SYSGEN or SAVE commands.

A > MOVCPM 64 ↵

The preceding example constructs a 64K version of CP/M. This version is executed without being stored on disk.

MPMLDR

☐ CP/M version 1.4
☐ CP/M version 2.2
▩ MP/M version 1.0

Purpose: This MPMLDR program is used to load and execute the MP/M system.

Command Nature: MPMLDR.COM

Format: MPMLDR

Description: MPMLDR loads, relocates in memory, and executes the MPM.SYS file previously generated with the GENSYS command. MPMLDR issues a display of the system parameters including the number of consoles, the breakpoint, the top of memory, and a memory segment table.

Command Usage: MPMLDR.COM can be executed from either MP/M or CP/M.

Examples:

```
A>MPMLDR ↵

MP/M 1.0 Loader
Number of Consoles =3
Breakpoint RST =5
Top of Memory =C0FFH
Memory Segment Table:

SYSTEM DAT  C000H  0100
.
.
.
.
.
.
```

MPMSTAT

☐ CP/M version 1.4
☐ CP/M version 2.2
▩ MP/M version 1.0

Purpose: MPMSTAT displays the MP/M system status.

Command Nature: Resident Process or MPMSTAT.PRL

Format: MPMSTAT

Command Description MPMSTAT displays the names of those processes waiting for CPU time, those processes waiting for messages from queues, and those processes waiting to send messages.

The following are also displayed.

* delay and polling processes
* flags waiting
* flags set
* queues in operation
* processes waiting for consoles
* processes attached to consoles
* memory allocation

Example:

```
0A>MPMSTAT
```

Refer to Chapter 5 for a detailed description of MPMSTAT output.

PIP

- ■ CP/M version 1.4
- ■ CP/M version 2.2
- ■ MP/M version 1.0

Purpose: Used to perform copying operations.

Command Nature: PIP.COM

Formats:

1. PIP { d: / (d:) new file } = (d:) old file (p)

2. PIP ↵

* (d:) (new file) = (d:) old file (p)
*...
*...
*...
* ↵

3. PIP { device: / (d:) filename } =

{ device: / (d:) filename } (p), { device: / (d:) filename }, ...

4. PIP d:=(d:) filename match (p)

Arguments: { d: / (d:) new file }

In Format 1, this argument can be used in different variations to specify that the file is to have the same or a new name. The argument can also specify whether the file is to be placed on the same or a different drive.

(d:) old file

In Format 2, the name of the old file being copied is required. The name of the new file (the copy) is not required in the command, as long as it is placed on a disk different from the original.

(p)
In all PIP commands, various PIP parameters may optionally be included.

{ device / (d:) filename }

In Format 3, the operator must choose between a device (a device name such as CON:) or a filename with an optional drive specifier.

This form of PIP has several different uses. It can be used to send a file to a device, receive data from a file or a device, or send special device codes to a device.

d:=(d:)filename match

The drive specifier is required for the destination file--it is optional on the source file. This format generally is used to copy several files onto another diskette using the same filenames for the copies as for the originals.

Command Description:

In format 1, if only d: is supplied on the left side of the command (and new file is not supplied), the new file will have the same name as the original source file. Remember, however, the d: specified on the left side of the command must be different from the optional d: supplied on the right side of the command. In other words, the file being copied must be copied onto a different drive from the original--if they are both to have the same filename.

When the optional d: is omitted from the right side of the command, the original source file (old file) is assumed to be on the current drive.

If a new file name (new file) is supplied, the new file being copied will have that file name. In cases where a different file name is specified for the file being copied, both files (old file and new file) can be on the same disk.

In format 2, PIP follows the same rules as in format 1. The difference is that in format 2, the user can perform several PIP operations in a row. That is, when PIP is entered alone followed by a Return, PIP is left in memory. An asterisk (*) is displayed as a prompt for the user to enter PIP expressions. When a Return is entered immediately following the prompt (*), PIP will be terminated.

Format 3 illustrates the use of device names with PIP. The user is not allowed to copy from a receive-only device--nor may the user copy to a send-only device.

The left side of the command is the destination (the file or read device which is to receive the data). The right side of the command is the source (the sending device for files being copied). Several sources can be joined into one destination file or receiving device. The special device names are listed in Appendix B at the end of this book.

In format 4, the user can use a filename match on the right side of the PIP command to copy several files onto another diskette. The drive specifier (d:) on the left side of the command is required and must be different from the drive on which the source files are contained. The reason for this is the rule that two files with the same name can not exist on the same diskette.

PIP may be used as a one line command as illustrated in format 1. You may also execute PIP and leave it in memory, execute several PIP commands, and then terminate PIP by pressing Return.

Examples:

```
A > PIP ↵

*B: = PROGA.TXT ↵
*LST: = PROGA.TXT ↵
*
```

In the above example, the PIP command is executed in the first line. In the second line, PROGA.TXT is copied with the same filename on Drive B. In the third line, a copy of PROGA.TXT is sent to the LST: device.

A>PIP FILE2 = FILE1 [N] ↵

In the above example, FILE1 is given line numbers with the N parameter as it is copied as FILE2. FILE2 also resides on Drive A.

PRLCOM

☐ CP/M version 1.4
☐ CP/M version 2.2
■ MP/M version 1.0

Purpose: Transform a PRL file into a COM file.

Command Nature: PRLCOM.COM or PRLCOM.PRL

Format: PRLCOM (d:) source program .PRL (d:) destination program .COM

Arguments: Source program .PRL: This is the name of the source program. A drive unit may be optionally specified.

Destination program .COM: This is the name of the destination program. A disk unit may be specified optionally.

Command Description: The PRLCOM command is used to transform a program of type PRL into a program of type COM. If the COM filename is already being used, a message will be displayed by the system, and the user will be given the option of cancelling the command.

Command Usage: A user generally uses PRLCOM to convert a PRL file to a COM file so that it may reside in absolute TPA rather than a relocatable memory segment.

Example:

```
1A > PRLCOM FILE.PRL FILE.COM
```

The example above converts FILE from PRL to a COM type file.

REN

- ☐ CP/M version 1.4
- ☒ CP/M version 2.2
- ☐ MP/M version 1.0

Purpose: Used to rename a file.

Command Nature: Built in CP/M command. REN.COM or REN.PRL in MP/M.

Format: REN old filename = new filename.

Arguments: Old filename: This is a required argument for the REN command giving the name of the file whose name is to be changed.

New filename: This argument is required for REN. It gives the new filename to which the old filename is to be changed to.

In both arguments, drive letter prefixes are not allowed, and filename extensions are required.

Command Description: The REN command is used to change a file's name. The filename extension must be included in the command.

Command Usage: In CP/M, REN is a built in command that can be executed at any time. In MP/M, REN.COM or REN.PRL must be on an accessible disk to be executed.

Example: A > REN NEW FILE.TXT = OLD FILE.TXT ↵

The above example changes the name of the filename OLD FILE.TXT to NEW FILE.TXT.

SAVE

- ☐ CP/M version 1.4
- ☐ CP/M version 2.2
- ☐ MP/M version 1.0

Purpose: Used to save the contents of memory on a disk file.

Command Nature: Built-in command.

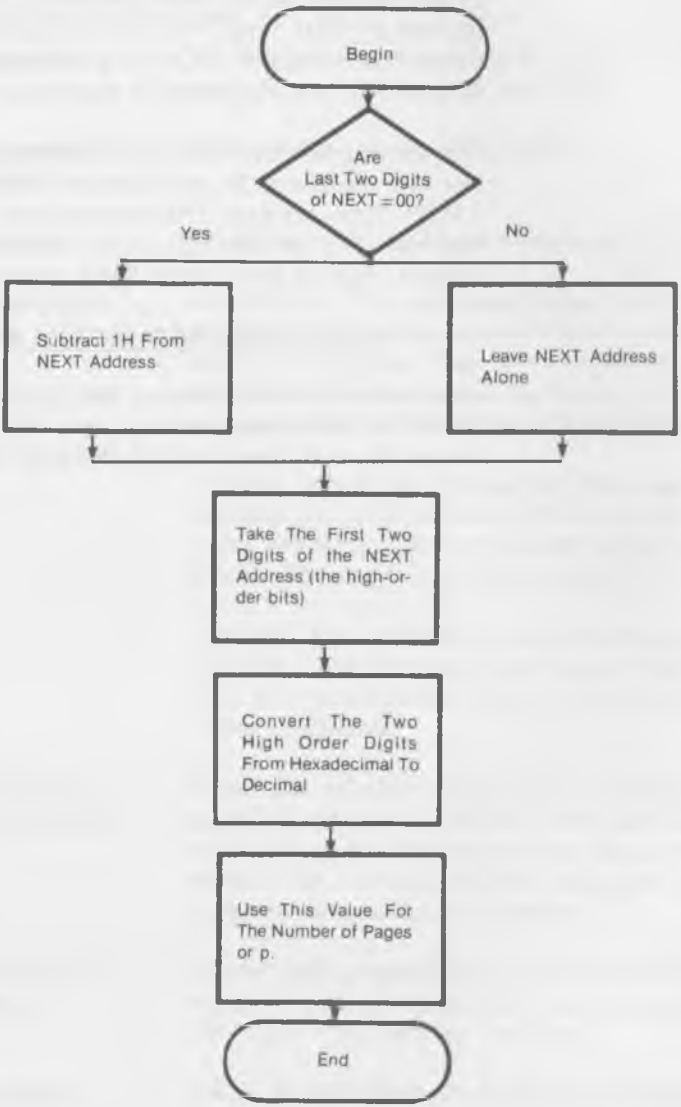
Format: SAVE p filename

Command Arguments: p: This is the required part of a SAVE command. The p refers to the number of pages to be saved. A page is a 256 byte segment.

Command Description: The SAVE command saves the contents of the TPA (Transient Program Area) beginning at memory location 100H and upwards in memory to p pages (the number of pages specified in the command).

Command Usage: To execute the SAVE command, you must first calculate p, the number of pages. To calculate p, use DDT to load the original program into memory. If you are using an early version of CP/M, use the R command to display the NEXT address. The NEXT address is the address following the loaded program in hexadecimal form. If you are using a later version of CP/M, the NEXT address is displayed automatically. You have no need to use the R command.

The address given by NEXT is one greater than the actual last address of the program. You can use the following algorithm to determine the number of pages (p) to specify in the SAVE command.



Examples:

```
A > DDT PROGRAMA.COM
NEXT PC
1B00 00
-G0
```

In the example above, PROGRAMA.COM is loaded by DDT into TPA. We are assuming that a later version of CP/M is being used. Therefore, no R command is necessary.

The value given by NEXT is 1B00H. Since the last two digits are 00, we subtract 1H to get 1AFFH. Now, we take 1AH and convert it to decimal. We get 26. This is the number of pages to specify for p in the SAVE command.

```
A > SAVE 26 PROGRAMB.COM
```

The above command saves PROGRAMA.-COM in TPA from memory into disk and names the resulting file PROGRAMB.COM.

SCHED

☐ CP/M version 1.4
☐ CP/M version 2.2
☒ MP/M version 1.0

Purpose: SCHED is used to schedule a program for execution at a later date and time.

Command Nature: Resident Process or SCHED.PRL

Format: SCHED mm/dd/yy hh:mm filename (.COM / .PRL)

Arguments: mm/dd/yy: This argument gives the date when the program scheduled is to be executed. It is required. The mm represents the month (1-12), dd represents the day (01-31), and yy represents the last two digits of the year.

hh:mm: This is the time when the program scheduled is to be executed. The hh stands for hours (00-24) and the mm stands for minutes (00-59). This argument is required.

filename: This is the filename for the file to be executed. The file must be of type .COM or .PRL, but the extension need not be specified in the command.

Command Description: When the SCHED command is executed, it waits in memory until the time and date match that of the respective arguments. When this occurs, SCHED executes the program named as its argument.

Command Usage: The SCHED program must be used with the required arguments. SCHED exists either as a .PRL file or as a resident process.

Example: 0A > SCHED 01/01/82 00:01 NEW YEAR ↵

SPOOL

☐ CP/M version 1.4
☐ CP/M version 2.2
☒ MP/M version 1.0

Purpose: Used to send one or more files to the spool queue, which is usually for the line printer.

Command Nature: Resident Process or SPOOL.PRL

Format: SPOOL filename, (filename,...)

Arguments: filename: The first filename is required, while the others are optional. Filename extensions must be specified.

Command Description: The SPOOL command sends files to the spool queue one by one. There, they wait until they are processed by the LST: device (generally the line printer). These files must be ASC II text files.

Command Usage: The SPOOL program will either be a resident system process or a .PRL file in the current disk drive. The command, STOPSPLR, can be used to cancel the spool queue operation.

Example: 0A > SPOOL FLA.TXT,FLB.TXT,FLC.TXT↵

The above example sends FLA.TXT, FLB.TXT, and FLC.TXT to the spool queue where they wait to be processed by the LST: device (usually the line printer).

STAT

- ☐ CP/M version 1.4
- ☐ CP/M version 2.2
- ☐ MP/M version 1.0

Purpose: The STAT command is used to display status information or assign devices.

Command Nature: STAT.COM or STAT.PRL

Formats:

1. STAT { DEV: / VAL: }

2. STAT gendevice: = phydevice:,...

3. STAT (d:) = (R/O)

4. STAT (d:) { filename / filename match }

5. STAT (d:) { filename / filename match } { $S / $R/O / $R/W / $SYS / $DIR }

6. STAT (d:) { DSK: / USR: }

Arguments:

{DEV:
VAL:}

This argument is used in the first format. When DEV: is used, STAT produces a display of actual device assignments. When VAL: is used, STAT displays potential device assignments. In CP/M 2.2, VAL: also lists possible STAT commands.

gendevice:phydevice

This argument is used in format 2. The gendevice: stands for the generic device name (CON:, PUN:, RDR:, or LST:). The dev: stands for any physical device that can be used with that generic device name.

(d:)=(R/O)

This argument is used in format 3. When d:=R/O is specified (d: is a drive letter), the drive specified by d: will be read only. If only d: is displayed as the argument, STAT will display that drive's current status. When no argument is specified, the STAT command will display the current drive's status read-only or read-write).

d: {filename
filename match}

This argument is used in format 4. If filename is used as the argument, the status (i.e. size in records and bytes, number of extents, filename, and type) will be displayed. If a filename match is specified, the status of several files will be displayed at once. In either case, the drive specifier (d:) is optional.

{ $S
$R/O
$SYS
$R/W
$DIR }

This argument is also used in format 4. $S is an optional field which causes the file's size to be displayed. The other parameters ($R/O, $R/W, $SYS and $DIR) are used to set file attributes. $R/O (read/only) protects a file from being overwritten or deleted. $R/W cancels out the $R/O attribute. The $SYS command prevents the file from being displayed by a $DIR command. $DIR cancels the $SYS file attribute.

(d:) DSK:
USR:

Format 6 is only used in CPM 2.2 and MP/M. The argument DSK: is used to display the disk characteristics. Unless a drive is specified by the optional d:, the current drive's characteristics will be displayed.

If the argument USR: is used, information on user areas will be displayed.

Command Description:

STAT is used to display information about files and diskettes, assign physical devices to generic names, give a file $R/O or $SYS file attributes, display current user areas, or display active user areas.

STAT is executed as either STAT.COM or STAT.PRL in one of the formats illustrated previously.

Examples:

A>STAT PIP.COM $S

```
Size   Recs   Bytes   Ext   Acc
 55     55     12K     1    R/O A:PIP.COM
```

The above example displays the size of PIP.COM. See Chapter 4 for examples of the various uses of STAT.

STOPSPLR

☐ CP/M version 1.4
☐ CP/M version 2.2
☐ MP/M version 1.0

Purpose: The STOPSPLR command is used to cancel out a SPOOL command and empty the spool queue.

Command Nature: Resident process or STOPSPLR.PRL

Format: STOPSPRL

Description: The STOPSPRL command is used to cancel a SPOOL operation already in progress and also to empty the spool queue.

Examples:

```
0A > STOPSPRL
```

In the above example, the STOPSPRL command will cancel the SPOOL operation and empty the SPOOL queue.

SUBMIT

- ☐ CP/M version 1.4
- ☐ CP/M version 2.2
- ☐ MP/M version 1.0

Purpose: SUBMIT is used to execute a batch of commands.

Command Nature: SUBMIT.COM or SUBMIT.PRL

Format: SUBMIT filename (value 1 value 2 value 3 ...)

Arguments: filename: This argument is required. The filename must be for a text file containing command lines. The filename is assumed to have the extension .SUB which is not required in the command.

Value 1 value 2 value 3...: These are optional values to be given to variables in the .SUB file. The arguments specified by value 1, value 2, value 3 etc. will replace the variables $1, $2, $3, etc. in the .SUB file.

The SUBMIT program accepts the file named with the .SUB extension and builds the file $$$.SUB, which is executed after a warm start (after the SUBMIT program terminates).

Command Description: The $$$.SUB file's command lines are then executed until the file is exhausted. When the $$$.SUB file is built, SUBMIT will substitute value 1 for $1, value 2 for $2, value 3 for $3 etc.

Submitted files can only be built when they are in Drive A.

Command Usage: By executing SUBMIT.COM or SUBMIT.PRL on a .SUB file, a batch of commands may be executed.

Example: Let's assume the file SAMPLE.SUB contained the following lines of text.

```
ERA $1.BAK
DIR $2.*
PIP $1.BAK = $2.BAK
ERA $2.BAK
```

If the SUBMIT program was used to submit this file as follows;

A>SUBMIT SAMPLE.SUB PROGA PROGB

the following command lines would be contained in $$$.SUB;

```
ERA PROGA.BAK
DIR PROGB.*
PIP PROGA.BAK = PROGB.BAK
ERA PROGB.BAK
```

When the SUBMIT command has finished substituting to build the file $$$.SUB, the system will execute the contents of the file $$$.SUB.

SYSGEN

- CP/M version 1.4
- CP/M version 2.2
- MP/M version 1.0

Purpose: SYSGEN is used to generate a copy of CP/M, bring the system into memory, and/or produce a copy of the system diskette.

Command Nature: SYSGEN.COM

Format: SYSGEN

Command Description: SYSGEN can be used to bring the system into memory and execute it. SYSTEM can also be used to initialize a system diskette; that is writing the first two tracks of the diskette with the system information.

Command Usage: After SYSGEN has been executed as shown below, a series of prompts will appear.

Examples: A > SYSGEN ↵

SYSGEN VERSION xx.xx
SOURCE DRIVE NAME (OR RETURN TO SKIP) A↵

Your response to the above prompt should be the letter of the drive where the system is located. However, respond with Return, if you want to skip the system read operation. This is done when the system is already in memory due to a MOVCPM operation.

SOURCE ON A, THEN TYPE RETURN ↵
FUNCTION COMPLETE

The system read operation is now complete, and the system is in main memory.

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) B↵

The response to this prompt should be the letter of the drive holding the system diskette to be initialized. If Return is pressed instead, the system will be executed in memory.

DESTINATION ON B, THEN TYPE RETURN↵

Here, the system is written on the system diskette. The system diskette can now be used.

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)↵
A>

In the above prompt, by pressing Return, the user has terminated SYSGEN.

TOD

☐ CP/M version 1.4
☐ CP/M version 2.2
☒ MP/M version 1.0

Purpose: This is the time of day command. It is used to display or set the time and date.

Command Nature: Resident Process or TOD.PRL

Format: TOD (mm/dd/yy) (hh:mm:ss)

Arguments: (mm/dd/yy): This argument is optional. It is only required when setting the date. The mm stands for the month (01-12). The dd stands for the day (01-31). The yy stands for the year.

(hh:mm:ss): This argument is required when setting the time or the date. The hh is the hour (00-24). The mm is the minute (00-59). The ss is the second (00-59). The argument is not required if TOD is used alone to display the system time and date.

Command Descriptin: In an MP/M system, TOD can be used to display the system time and date if it is used alone, without arguments.

If TOD is used with a time and/or date argument, TOD will prompt the user with the following message:

Strike a key to set a time.

When the user wishes to set the system time, he or she can do so by pressing any key.

TOD can exist as a .PRL file on your current disk or as a resident system process (a command).

Example:

```
0A > TOD ↵

FRI  10/22/81  20:01:37
```

In the above example, TOD displays the system time.

```
0A > TOD 10/22/81  20:01:37 ↵

Strike a key to set time

10/22/81  20:01:37
```

In the above example, when the operator strikes a key, the system time is set to that time and date supplied in the arguments.

TYPE

- ☐ CP/M version 1.4
- ☐ CP/M version 2.2
- ▩ MP/M version 1.0

Purpose: Displays the contents of a file on the console screen.

Command Nature: Built-in command in CP/M.
TYPE.COM or TYPE.PRL in MP/M.

Format: TYPE

Arguments: filename or filename match: The user must include either a filename (with extension) or a filename match. The filename displays the specific file named, while the filename match displays any files that match.

Command Description: The TYPE command can be used to display the contents of any file. However, the user will only be able to read the contents of an ASC II text file such as a listing file, source file, or .PRN file.

Command Usage: In CP/M, TYPE is a built-in command that can be executed at any time. In MP/M, TYPE may exist as either TYPE.COM or TYPE.PRL.

Examples: A > TYPE FILE1.TXT ↵

In the example above, FILE1.TXT will be displayed on the console screen.

USER

☐ CP/M version 1.4
☐ CP/M version 2.2
☐ MP/M version 1.0

Purpose: Used to change current user area or display user area in MP/M systems.

Command Nature: USER.COM or USER.PRL

Format: 1. USER (x) in MP/M

2. USER x in CP/M 2.2 and later

Argument: x: The argument x stands for the user area number (0-15). This argument is required in CP/M, but is optional in MP/M.

Command Description: If no argument is supplied, the USER command is executed alone in MP/M to display the current user number. If an argument (x) is supplied with USER in MP/M, the user are will be changed to that specified by the argument. In CP/M 2.2, the argument is required, meaning that the current user area may only be changed--not displayed.

Command Usage: The USER program is supplied as either USER.COM or USER.PRL.

Example:

```
A > USER 4
A >
```

In the example above, the current user area is changed to 4.

XSUB

☐ CP/M version 1.4
▩ CP/M version 2.2
▩ MP/M version 1.0

Purpose: This program is the extended SUBMIT, which allows input to programs executed in the submit file.

Command Nature: XSUB.COM

Format: XSUB

Command Description: XSUB is available in CP/M 2.2 and MP/M and offers the user expanded use of .SUB files. XSUB allows the user to include commands in the .SUB file to programs that use buffered input (other than the CCP). The ED, DDT, and PIP programs all use buffered input.

Command Usage: XSUB is inserted as the first command line of the .SUB file. The .SUB file is then executed line by line until all commands have been executed. XSUB will remain active until the next cold start.

```
XSUB
PIP 1$.TXT = 2$.BAK
```

Example:

```
A > SUBMIT COPY NEW ORIGINAL
```

The .SUB file (COPY.SUB) is executed by the SUBMIT command. Since XSUB is the first line in COPY.SUB, the XSUB program is relocated to the area of memory just below the CCP, where it remains active. Next XSUB executes the PIP command with NEW substituted for $1 and ORIGINAL substituted for $2.

CHAPTER 10. CBASIC PROGRAMMING LANGUAGE

CBASIC BEGINNINGS

CBASIC is a **programming language.** A programming language can be defined as a specific set of words and symbols which represent calculations, operations, decisions, and procedures which can be undertaken by the computer.

Binary symbols are used by computers to perform all of their operations. Since binary logic consists of long sequences of 1's and 0's, programming a computer using binary logic would be both difficult and inefficient.

Programming languages consist of instructions which are automatically reduced to binary instructions which are then performed by the computer. The advantage to programming languages are that unlike binary instructions, they can readily be understood by humans as well as computers.

CBASIC was first released in 1977. However, CBASIC did not become widely used in the microcomputer field until 1978, when CBASIC version 2 was released.

CBASIC version 2 had many features that software development firms were looking for. These software development firms began writing applications software in the CBASIC version 2 language.

The main advantage of CBASIC is that it works with the CP/M operating system. CP/M is one of the most widely used operating systems for microcomputers. Therefore, programs written in CBASIC will operate on the thousands of microcomputers on which CP/M is installed.

CBASIC STRUCTURE

CBASIC programs are written as source programs or source code. Source code consists of CBASIC reserved words such as

PRINT, READ, OPEN, DATA, variable names, arithmetic operators, and functions such as MID$, CHR$, and LEN. The source code is the form of CBASIC generally dealt with by programmers.

CBASIC converts the source program into a condensed series of symbols which can be executed by the computer. This conversion process is known as compiling. The series of symbols that can be executed by the computer is known as the object code.

CBASIC consists of two parts; the **compiler** and the **monitor**. As we just mentioned, the compiler translates the source code into object code. The monitor interprets the symbols used in object code into a language that can be read by the computer and executes the program.

The fact that CBASIC contains both a compiler and a monitor makes it unique among the differing versions of the BASIC programming languages. Many versions of BASIC are known as **real-time interpreters**. Real-time interpreters allow the programmer to enter a program and run it in one step. The compilation step is not necessary.

The advantage to real-time interpreters is that the programmer can trace a program step-by-step to find errors. Also, the compilation step is eliminated.

The advantage of a machine language compiler is that it executes a program much faster than a real-time interpreter. Once the source code has been translated into a form similar to the computer's native language, interpretation is not needed, and the program will execute much faster.

To the novice, a compiled language such as CBASIC may seem to be a nuisance. However, as we have discussed, compiled languages offer a real advantage in fast execution.

CBASIC ERRORS

Every programmer encounters errors. They occur in both simple and complex programs.

CBASIC distinguishes errors as two different types; **compiler errors** and **run-time errors**. Compiler errors are defined as errors

in the usage of the programming language. For example, the misspelling of the reserved work INPUT as INRUT would be an example of a compiler error. Compiler errors fortunately are easily discovered and corrected. As your skills as a CBASIC programmer increase, the frequency of compiler errors will decrease.

Run-time errors are errors that result in the program not doing what it is supposed to be doing. Run-time errors are much more difficult to detect than compiler errors. The process of solving run-time errors is generally a detailed and involved process.

USING ED TO EDIT A CBASIC PROGRAM

As you already know, CP/M includes a text editor program known as ED. ED is not as sophisticated as many of the text editor programs available. If you own a more sophisticated text editor than ED, you may wish to use it to enter your CBASIC programs.

Since everyone who has a CP/M installation has an ED program, we will use ED to illustrate how to edit a CBASIC program.

Your first step is to enter ED followed by the name of the file that you wish to enter and finally the carriage return.

A>ED PROGRAM1.BAS ↵

If PROGRAM1.BAS already existed on the diskette, then the preceding command would make that program file available for ED.

If PROGRAM1.BAS did not already exist on the diskette, then CP/M would assume that you wanted to create a new file with the filename PROGRAM1.BAS.

↵ signifies pressing the carriage return key.

—— signifies operator input.

If this was the case, ED would respond to your initial entry with the following:

```
NEW FILE
*
```

This message means that CP/M will create the new file on its directory. The directory is a portion of the diskette which contains the name, size, and location of every file residing on that diskette.

ENTERING NEW TEXT WITH I

After the ED command has been entered with an appropriate filename, an asterisk (*) will appear at the beginning of the next line. This asterisk lets you know that ED is ready for use, and is known as the ED prompt. The asterisk is just one of several prompts encountered in CP/M.

Once the ED prompt has appeared, the ED program will have been loaded into memory and will be waiting for further instructions. If you are working with an existing file which already contains text, you should first enter the Append command. The Append command copies lines from the file being edited into the **edit buffer**--an area in memory where the actual text processing takes place.

```
*500A ↵
```

The number preceding the Append command specifies the number of lines from the file being edited that are to be appended to the edit buffer.

If you are working with a new file, naturally, you will not need to append lines to the edit buffer.

To begin entering text into the edit buffer, the Insert command must be entered.

```
*I ↵
```

After the Insert command has been entered, the ED prompt character will disappear, and you will begin entering text into the edit buffer by typing on the keyboard.

Once you have finished inserting text into the edit buffer, you must notify the ED program that you wish to terminate the Insert command. This is done by pressing the Z key while holding down the Control key (Control-Z).

```
*↑Z
```

After Control-Z has been pressed, the Edit prompt will reappear.

At this point, you may wish to review the text that you have just entered so that you can check for errors. You can display the entire edit buffer by entering the following command.

```
*B#T
```

The B command moves back to the beginning of the edit buffer while the #T types out the lines in the edit buffer until the end of the file has been reached.

CORRECTING ERRORS IN ED

There are several ways to correct errors in the edit buffer. Generally, errors are corrected either by re-entering the entire line where the error occured or by substituting for the erroneous letters.

To re-enter a line, move to the line in the edit buffer where the correction is to be made. Use the Insert command to enter the new line. Then, enter Control-Z to end the Insert command. This is shown in the example below.

```
6: Z$ = A$ + B$
6: *
7: PRINTT Z$
7: *I
7: PRINT Z$
8: ↑Z
8: *K
8: *
```

After the new line has been entered, you must delete the erroneous line with the Kill command (K), as has been illustrated above. After the erroneous line has been deleted, you will have corrected the error.

Another method of correcting a line in the edit buffer besides reentering the entire line is to correct any erroneous letters by substituting the correct letters. This is done by using the Substitute or S command.

```
101: PRITT Z$
101: *STT↑Z NT↵
101: PRINT Z$
*
```

The preceding example corrects the misspelled reserved word PRITT to PRINT by using the Substitute command. The first command S stands for Substitute. The S command searches for text in the edit buffer until it matches the characters that are specified immediately following S. In our example, these are the letters TT.

After the letters to be substituted for have been entered, Control-Z is entered (↑Z). Control-Z indicates the end of the characters to be searched for.

After Control Z, the characters to replace the original characters are entered (NT).

The command line in our example:

STT↑ZNT

can be interpreted as "substitute the first two characters found in the text--TT--with NT.".

By including a number as a prefix with the S command, the Substitution will repeat that number of times. For example, in the following:

10STT↑ZNT

NT would be substituted for TT the next ten times TT is encountered in the edit buffer.

SAVING THE PROGRAM ON DISK

When you are satisfied with the program that has been entered into the edit buffer, you will want to save that program on disk.

The program can have any primary filename that you wish. However, the program must have a filename extension of .BAS.

To save the program on disk, you should enter the END command (E) as shown:

```
*E
A>
```

The E command saves that file on the diskette, ends the ED program and returns control to CP/M.

COMPILING THE CBASIC PROGRAM

Once you have entered your CBASIC program file, your next step is to compile that file. A CBASIC program is compiled by entering the name of the CBASIC compiler (CBAS2) followed by the name of the .BAS file to be compiled. This is illustrated in the following command.

```
A > CBAS2 PROGRAM.BAS
```

As your program is compiled, it will be displayed line by line as shown below.

```
CBASIC COMPILER VER 2.07
1: REM BEGIN PROGRAMA.BAS
2: A = 1
3: B = 2
4: C = 4*A + B
5: PRINT "THE ANSWER IS";C
6: END

NO ERRORS DETECTED
CONSTANT AREA:      9
CODE SIZE:          47
DATA STMT AREA:     0
VARIABLE AREA:      04
```

These preceding lines are known as the **compiler listing**. If the program contained any compiler errors, these would have been displayed with the program.

If the program does contain errors, these will be listed along with an error code. The total number of errors in the program is listed at the end of the compiler listing.

The lines printed at the end of the compiler listing are known as compiler statistics. The compiler listing gives the size of certain memory areas used by your CBASIC program when it is run. These compiler statistics are useful in more advanced programs.

RUNNING THE COMPILED CBASIC PROGRAM

Once the CBASIC program has been compiled, its compiled version will be saved on the diskette with the filename extension .INT. This file will contain the original BAS file in its compiled form, which can then be executed by the CBASIC interpreter.

You can run your compiled program by typing in the following command:

```
A> CRUN PROGRAMA ↵

CRUN VER 2.07
THE ANSWER IS 6

A>
```

INTRODUCTION TO PROGRAMMING PRACTICES

If you are just beginning to program, the first programming concept that you must understand is that programs must have structure and organization. A good program does not consist of an arbitrary series of BASIC statements. A good program is well-structured, well-organized, and easily modified.

When you are writing lengthy programs, your best strategy is to divide that long program into a series of smaller programs or **modules**, and then write each module separately. This practice is known as **modular programming**.

Modular programming offers several advantages to programmers. First of all, smaller programs are more easily written, coded, and debugged. Secondly, modular programs are more easily understood by others. This is especially important when

other people will be reading or using your programs. Finally, you can reuse individual modules over and over again in other programs.

FLOWCHARTING TECHNIQUES

An excellent method of organizing a program before you actually write it is to draw a flowchart of it. Flowcharts are a set of symbols and connectors which provide a map of the logical operation of the program.

There are two different types of flowcharts; **system flowcharts** and **program flowcharts**. A system flowchart is an overall map which depicts how the individual modules of the system interact as a whole. An example of a system flowchart for an Accounts Payable System is given in Illustration 10-1.

System flowcharts give an overall view of how the program works. Each of the boxes in the system flowchart describes a module in the overall system.

A program flowchart is shown for the Accounts Payable Checkwriter module in Illustration 10-2. The program flowchart differs from a system flowchart in that it is much more detailed. Every operation and calculation is described in a program flowchart.

MODULE STRUCTURE

Before you actually begin writing a program or a program module, it is a good idea to divide that module into a structure. Generally, you will find that structuring a program or module involves the following steps.

Define the data. The variables to be used by the program are set to their initial value. Arrays and tables are defined via DIM statements.

Input data. The data that the program needs to operate is retrieved--either from operator input or from a peripheral device such as a disk drive.

Process the data. The necessary operations are performed on the data.

ILLUSTRATION 10-1. SYSTEM FLOWCHART FOR ACCOUNTS PAYABLE SYSTEM

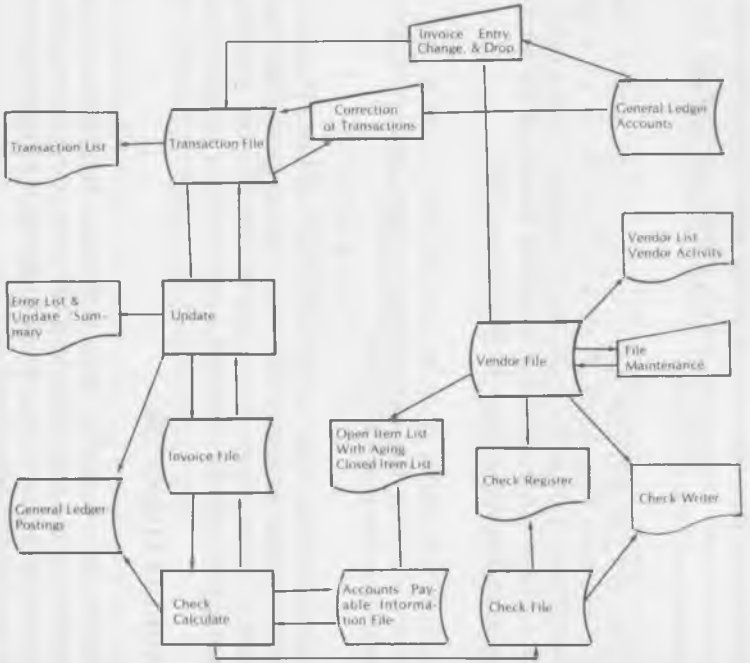
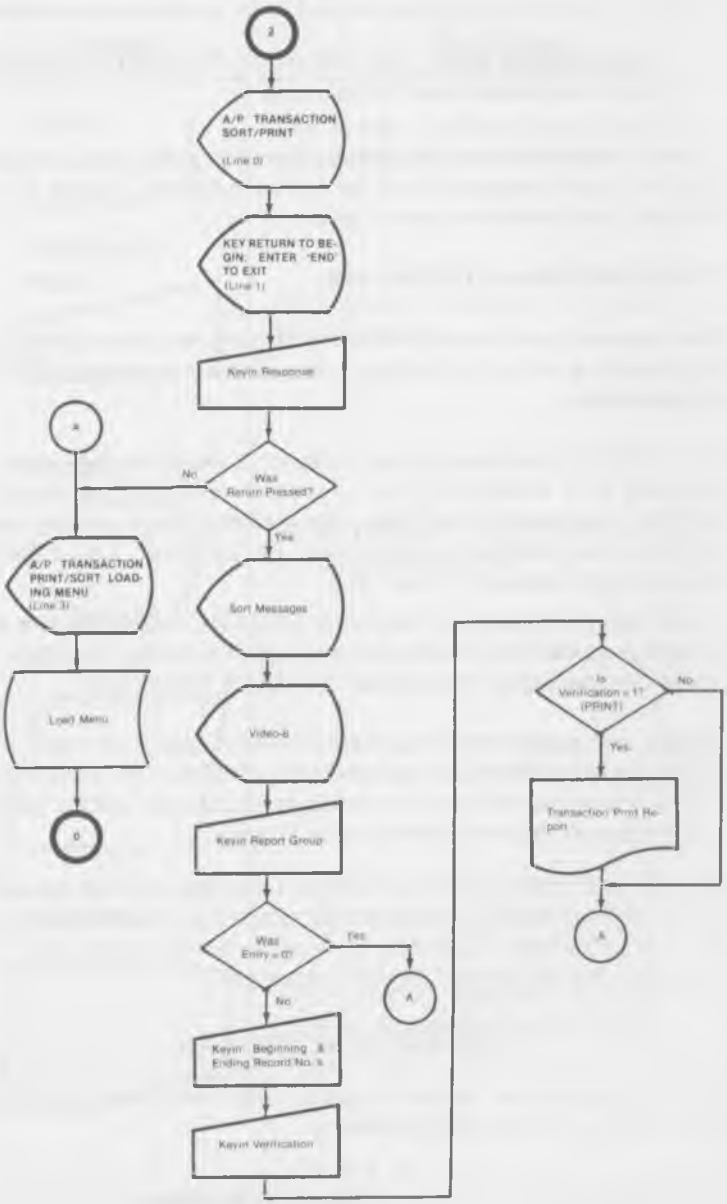


ILLUSTRATION 10-2. PROGRAM FLOWCHART FOR AP TRANSACTION PRINT



Output the data. The results of the data processing are either displayed on the console or printed by the line printer.

Store updated data. The disk which has been processed is stored on disk or tape for later use.

If you remember these five steps when designing your program modules, your programs will be well-structured, easier to understand, and more resistant to errors.

CBASIC PROGRAM STATEMENTS

Now that we have introduced some of the principles of modular programming, we will introduce some of the concepts of CBASIC programming.

Every CBASIC statement must contain at least one **keyword**. A keyword is a word that has a special reserved meaning in CBASIC. A keyword describes a pre-defined input process, output process, decision, calculation, or function. The CBASIC keywords are listed in Table 10-1.

In addition to at least one keyword, a CBASIC statement can also contain **parameters**. Parameters are variable names, numbers, or special symbols that may appear in CBASIC statements.

CBASIC programs may be written in either upper or lower case letters. When CBASIC programs are compiled, the lower case letters are automatically converted to upper case letters (unless this feature is suppressed with the $D toggle).

CBASIC statements may be divided among several separate lines. The backslash (\) character (Control-9 on most terminals is used to continue a CBASIC statement to the next line. For example, the following CBASIC statement:

```
IF A > 0 THEN GOTO 5999
```

could be expanded to two separate lines with the use of the backslash as shown in the following:

```
IF A > 0 \
   THEN GOTO 5999
```

TABLE 10-1. FLOWCHART SYMBOL DESCRIPTIONS

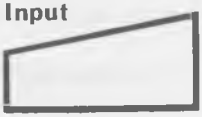
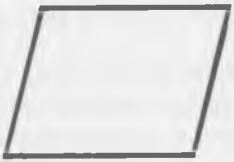
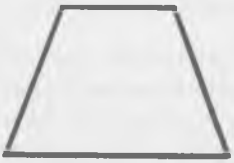
Symbol	Operation
Arrow 	This symbol indicates the flow of the program logic.
Operator Input 	This symbol indicates an entry from the keyboard.
Display 	This symbol describes what is displayed on the CRT.
Disk 	This symbol describes an action which takes place on the disk drive.
Printout 	This symbol indicates a document printed on standard paper by the printer.

TABLE 10-1 (CONT.). FLOWCHART SYMBOL DESCRIPTIONS

Symbol	Operation
Decision 	This block indicates a change in the logic flow in the program depending upon the answer to the question in the block.
Process or Note 	A program action is described, or a note to the reader is enclosed in this symbol.
Connector 	The program flow continues at the matching connector.
User Instructions 	The operator must perform some function outside the program.
Terminal 	This symbol indicates the program start or end.

Blank spaces as well as the backslash can be used to make CBASIC programs more readable. In CBASIC, wherever you are allowed to insert one blank space, you may insert as many blank spaces as you desire. Blank spaces are especially useful in indenting program lines so that the program takes on a certain outline and is easily read.

You may also place more than one CBASIC statement on a single line through the use of the colon (:). By separating CBASIC statements with a colon, they may appear on the same line, yet still be interpreted as separate statements. This is shown in the following example:

```
A = 0:GOTO 5999
```

LINE NUMBERS

Many versions of BASIC require the inclusion of a line number with every program statement. In CBASIC, line numbers need not be included with program statements. They are optional.

However, line numbers are required if that line number is referenced by a statement in the CBASIC program. This is often the case in GOSUB or GOTO statements.

In most versions of BASIC, line numbers must be integers and must occur in an ascending order. In CBASIC, line numbers do not have to be in any particular order, nor need they be integers.

CBASIC line numbers may either be an integer or a real number. Line numbers may contain fractions or decimals. They may even be exponential in nature. The following are examples of CBASIC line numbers.

```
1000
4.375
40000
17.5
```

CBASIC's only rule pertaining to line numbers is that the line number must contain 31 characters or less.

REMARK STATEMENTS

You will find it valuable to include comments in your programs

that explain what is being done in particular portions of it. Not only does this help others understand your programs, it also helps you remember the reasoning behind your program statements.

There are two ways to add comments to a CBASIC program.

1. Write comments beyond the backslash (\) character on any line.

2. Use REMARK statements.

Any characters appearing after the keyword REMARK or its abbreviation REM, are ignored by CBASIC. The following segment from a program uses REM statements to explain the program logic.

A REM statement can either be the only statement on a line or it may appear on the same line with other statements. If a REM statement appears on the same line with other statements, then the REM statement must be the last statement on the line. The following lines illustrate the use of the REM statements.

```
REM FUTURE VALUE OF AN INVESTMENT
INPUT I   REM ENTER INITIAL INVESTMENT
INPUT R   REM ENTER INTEREST RATE
INPUT C   REM ENTER # OF COMPOUNDING PERIODS
INPUT Y   REM ENTER # OF YEARS INVESTED
REM CONVERT INTEREST RATE TO DECIMAL
R = R/100
```

CONCLUSION

This chapter is meant only as an introduction to CBASIC. WSI does publish a complete guide to CBASIC titled "CBASIC SIMPLIFIED". This book is available for $13.95 from your local computer dealer or bookstore. It is also available by mail for $13.95 plus shipping and handling from:

Weber Systems Inc.
Box 413
Gates Mills, OH 44040

TABLE 10-2. CBASIC KEYWORDS SUMMARY

ABS	DELETE	INT	OUT	SQR
AND	DIM	INT%	PEEK	STEP
AS	ELSE	LE	POKE	STOP
ASC	END	LEFT$	POS	STR$
ATN	EQ	LEN	PRINT	SUB
BUFF	EXP	LET	RANDOMIZE	TAB
CALL	FEND	LINE	READ	TAN
CHAIN	FILE	LOCAL	RECL	THEN
CHR$	FOR	LOG	RECS	TO
CLOSE	FRE	LPRINTER	REM	UCASE$
COMMAND$	GE	LT	REMARK	USING
COMMON	GO	MATCH	RENAME	VAL
CONCHAR%	GOSUB	MID$	RESTORE	WEND
CONSOLE	GOTO	NE	RETURN	WHILE
CONSTAT%	GT	NEXT	RIGHT$	WIDTH
COS	IF	NOT	RND	XOR
CREATE	INITIALIZE	ON	SGN	
DATA	INP	OPEN	SIN	
DEF	INPUT	OR	SIZE	

CHAPTER 11. PRACTICAL OPERATIONAL GUIDELINES

INTRODUCTION

You should now have become proficient in the use of CP/M. Through this book, you have learned how to use CP/M as well as how CP/M operates.

CP/M is probably the most widely used operating system for microcomputers. Although CP/M will continue to evolve and improve, later CP/M versions will probably be compatible with earlier versions. Therefore, the CP/M foundations that you have mastered in this book will continue to be of use for future releases of CP/M. You need only learn the additional features of new versions of CP/M and you will have mastered that new version.

To become truly proficient at using CP/M, you must actually practice it with a microcomputer. Test all of the resources of CP/M. Even if you have no immediate use for the editor, try it out. It will prepare you for using word processing programs, as well as helping you become a more proficient CP/M user.

Mastering CP/M has another advantage that we have not yet mentioned. The basic concepts of most operating systems are very comparable. By mastering CP/M, you will be more easily able to adapt to and master other operating systems.

Now that you have an overall working knowledge of CP/M, we would like to outline some practical operating guidelines to using CP/M that will allow you to become as proficient as possible at using CP/M.

DISCIPLINED USE

Discipline is the key to successful usage of computers. Minor errors can cause serious problems in a program. By faithfully following correct operating procedures, almost all of these problems can be avoided. In the following sections, we will outline correct computer system operating procedures which will allow you to avoid almost all serious problems.

ORGANIZE THE COMPUTER ROOM

You should keep your computer room well organized. You must control the physical environment for the computer so that it will not cause errors. A list of environmental problems that can occur in a computer room is shown in Table 11-1.

You should also be sure that the necessary manuals, maintenance logs, diskettes, and documentation are readily available. These items are listed under the heading 'System Documentation' in Table 11-2.

Finally, you should be certain that adequate supplies are available in the computer room as listed in Table 11-3.

TABLE 11-1. ENVIRONMENTAL PROBLEMS

Item	Description
Ventilation	Be certain that the computer room has adequate ventilation.
Computer Outlet Vents	Be certain that the computer ventilation outlets are free from any obstruction. A common problem is an object being placed on top of a ventilation outlet.

TABLE 11-1. (CONT.) ENVIRONMENTAL PROBLEMS

Item	Description
Non-Metallic Diskette File Holders	If you use a file holder for your diskettes that can be magnetized, you risk having your disks erased.
Telephone	Keep telephones out of your computer work area. A ringing telephone on top of a diskette or a disk drive can erase a diskette.
Screwdrivers	Keep screwdrivers away from your computer work area. Screwdrivers can become magnetized and cause an erasure of diskettes.
Liquids	Keep all liquids away from your computer work area.
Smoking	No smoking should be allowed in the areas near your disk drives.
Disk Drives	Do not unneccessarily move your disk drives or allow them to be vibrated.
Carpeting	Guard against using carpeting that is prone to static electricity in your computer work area.

TABLE 11-2. SYSTEM DOCUMENTATION

- ☐ 1. All Required Manuals
- ☐ a.) Computer Manual
- ☐ b.) CRT Manual
- ☐ c.) Printer Manual
- ☐ d.) Disk Drive Manual
- ☐ 2. CP/M Diskette
- ☐ 3. CP/M Documentation
- ☐ 4. Applications Program Diskettes
- ☐ 5. Applications Program Documentation
- ☐ 6. Maintenance Record
- ☐ 7. Maintenance Phone Numbers
- ☐ 8. Record of Correct Printer Settings
- ☐ 9. Record of Correct CRT Terminal Settings
- ☐ 10. System Start-Up Procedures
- ☐ 11. Precautionary Procedures

TABLE 11-3. COMPUTER ROOM SUPPLIES

- ☐ Blank Diskettes
- ☐ Print Wheel
- ☐ Spare Ribbons
- ☐ Printer Paper
- ☐ Blank Forms Required for Applications Programs (Invoice Forms, Blank Checks, etc.)

COPYING DISKETTTES

Whenever a new program or system diskette is acquired, that disk should be copied first, before it is used. The original should then be stored in a safe area and the copy used for actual operations. Therefore, if your copy is ever accidentally damaged, the original is still available so you can make more copies.

When a lengthy file has been worked with extensively, the user should copy this file onto a fresh diskette. By doing this, the file's sectors will be copied sequentially onto the fresh diskette. This results in much faster loading of the file from the new diskette than from the original.

STORING & HANDLING DISKETTES

Rule 1. The first rule to remember when handling diskettes is to keep them away from any magnetic objects -- or any metallic objects that could have become magnetized. Examples include magnets, telephones, and magnetized screwdrivers.

Rule 2. The second rule to remember is to always store diskettes in their dust covers.

Rule 3. The third rule is to keep your diskette free from smoke or dust contamination.

Rule 4. The fourth rule is never to touch, scratch, or attempt to clean a diskette's surface.

Rule 5. The fifth rule is to always properly label your diskettes. The label should include the date and contents at the least. A good procedure is to generate a directory for each disk, then list

that directly on the printer, cut out that listing, and paste or tape it to the disk cover. This gives a complete guide to a diskette's contents.

Rule 6. Keep separate copies of your important diskettes in several different locations. In the case of a disaster (fire, theft, accidental erasure), you can copy your back-up diskettes, and avoid losing data files.

Rule 7. Do not expose diskettes to excessive heat or direct sunlight.

Rule 8. Never fold, bend, or mutilate a diskette.

Rule 9. Never write on a diskette with a ball point pen. Always use a felt tip pen. A ball point pen can damage the diskette inside the cover.

Rule 10. Whenever the disk drive's power is turned on or off, be certain that the diskette is not completely inserted in the drive.

Rule 11. Always make back-up copies of disks containing important files.

Rule 12. Never allow diskettes to be exposed to dust. Dust accumulates a static charge which allows it to stick to the surface of the diskette. This is especially important to remember when storing diskettes in industrial areas, schools (chalk dust), warehouses, stock rooms, and dental offices (grinding of plaster).

Rule 13. Keep your computer work area relatively dust free. If possible, install an electrostatic air cleaner for the room.

Rule 14. Try to keep the build-up of static electricity to a minimum in your computer room. Anti-static sprays are available for use on carpeting . Install a dehumifier if possible.

THE PRINTER

The most important rule to remember when using a printer is to understand its operation. Read your owner's manual thoroughly. Be certain that you understand the functions of all adjustment levers.

Before printing, be certain that these adjustment levers are set properly. For example, if you are using thin paper in your printer, but have the paper width lever adjusted for heavy paper, the printer may malfunction. Be sure that you understand the printer adjustments and set them properly.

Another potential error exists in environments where two different printers are used. This is common in a business environment where a daisy wheel printer is used for word processing applications and a line printer for reports. In these cases, separate CP/M versions may be necessary for each hardware configuration. Be certain that the CP/M diskette you are using corresponds to your hardware configuration.

Once your printer is operating properly and is listing data, more problems may be encountered. The paper may become jammed in the printer. If you are printing adhesive backed labels, this can especially be a problem.

If the printer does become jammed, it is important that the operator be present to shut off the operation. Otherwise, physical damage could be incurred by the printer. Since printing is a relatively slow operation, many operators do not attend the printer while it is operating. This is an acceptable practice as long as the printer is operating properly, and the operator frequently checks its operation.

However, at the beginning of a print run, it is recommended that the operator attend to the printer, as this is the period where it is most likely to jam. Also, when printing labels, it is recommended that the operator be present at all times, as jamming often occurs during label print runs.

Whenever a problem such as a jam occurs in a print run, that print run must be restarted. If the file being printed is of a short or medium length, it is usually easiest just to restart the print run.

However, if the file is a long one, you may wish to restart the listing near the point where the error occured. In such cases, you can use PIP or ED to select the portion of the file that requires reprinting.

HOW TO SHUT DOWN THE SYSTEM

If something is going wrong with your system, you may wish to shut it down. Do not turn the power off or pull the electrical cord out of the wall socket.

First of all, try a warm start by pressing CTRL-C. If this does not work, try pressing RESET. Although you will lose information held in RAM, you will not damage your file.

If your printer malfunctions, you can shut it down by merely turning off the power.

CP/M & DISKETTE SPACE

It is a good practice to install CP/M on your diskettes when they are blank. By doing this you are able to boot from any diskette. Also, your chances of damaging files when swapping diskettes when using PIP are reduced considerably.

SYSTEM TROUBLESHOOTING

Step 1. Check For Operator Errors

1. Are the cables properly connected?

2. Are any fuses blown?

3. Are all switches in the correct position?

4. Was the correct command given?
   a.) Turn the system off
   b.) Turn the system on. Repeat the command

Step 2. Check The Diskette

1. Use your back-up diskette to generate a second back-up.

2. Use your back-up in the system in place of the original which was causing the erratic system behavior.

Step 3. Check The Software

1. Make sure that you are using the correct version of CP/M for your system.

2. Make sure that you are using the correct compiler for your applications software (ex. CBASIC2).

3. Make sure that you are using the correct applications program diskette.

4. Make sure that the applications programs have been written so that they will work on your terminal and printer.

Step 4. Check The Hardware

1. Try to identify the defective device (printer, disk drive, memory board) using any troubleshooting advice in your owners manuals.

2. Try switching the malfunctioning component for a component that you know is good. For example, if you believe that your printer is malfunctioning, try using another printer (the same model).

APPENDIX A

COMMON CP/M ERROR MESSAGES

The following error message reports errors which are related to the CP/M system in general rather than to one particular command (i.e. ED, PIP, ASM, etc.).

BDOS ERR ON d:error

In the preceding error message, d: is a letter identifying the disk drive where the error took place. The error can be any one of the following error messages.

BAD SECTOR
SELECT
READ ONLY

The following message is another error condition found only in CP/M 2.2 and later versions.

FILE R/O

Each of these error messages will be discussed in the following four sections.

BAD SECTOR

A bad sector error occurs when the disk controller is unable to read information from a diskette. One major cause of bad sector errors is a bad diskette. A second possible cause of this error is defective disk drive controller.

A third possible cause of a bad sector error is if the diskette you are trying to read from was written onto by a different controller. Even though disk controllers are suppose to be standardized, there may be small differences in record formats, which might cause bad sector errors.

There are two possible ways to recover from a bad sector error. First of all, you can try pressing the Return Key. This tells the system to ignore the bad sector error and continue execution.

However, pressing the Return Key may not enable execution to continue. If this is the case, you can reboot the system by pressing Control-C. This will abort the program and return control to the CP/M system.

SELECT:

The SELECT: error is encountered when the user selects a disk drive which is not present. The drive named in the SELECT error message is the one selected in error. The system may be rebooted by pressing any key on the terminal.

READ ONLY

The READ ONLY error occurs when the user attempts to write to a diskette that has been specified as read only via the STAT command.

The READ ONLY error is also encountered when the user inserts a new diskette without first rebooting the system by pressing Control-C. When a new diskette is inserted, the user must press Control-C to change the diskette's memory map before that diskette can be written to.

If the READ ONLY error is encountered, error recovery can be made by pressing any key on the terminal and then rebooting the system by pressing Control-C.

FILE R/O:

The FILE R/O error is encountered on CP/M 2.2 and later versions. The FILE R/O error occurs when the user attempts to write to a file that has been specified as read-only ($R/O).

The FILE R/O error condition can be recovered by pressing any key on the terminal. When a key is pressed, the operation attempting to write to the read/only file will be aborted. The user should then use the STAT command to change the file attribute from read/only ($R/O) to read/write ($R/W).

APPENDIX B

PIP DEVICE NAMES

	LOGICAL DEVICE NAMES
CON:	console or terminal. Includes both keyboard and video display. (input/output)
RDR:	paper tape or card reader. (input)
PUN:	paper tape or card punch. (output)
LST:	listing device such as line printer. (output)

	PHYSICAL DEVICE NAMES
TTY:	or teletype. Used for console, terminal, reader, punch or listing device.
CRT:	or Cathode Ray Tube. Used for a console, terminal or listing device.
PTR:	or paper tape reader. Used for paper tape or card reader.

PTP:	or paper tape punch. Used for paper tape or card punch device.
LPT:	or line printer. Used for listing device (usually line printer).
UC1:	or user-defined console.
UR1:	or user-defined reader.
UR2:	or second user-defined reader.
UP1:	or user-defined punch device.
UP2:	or second user-defined punch device.
UL1:	or user-defined listing device.

APPENDIX C

FILENAME EXTENSION TYPES

Extension Name	Description	Example
ASM	Filename extension required for assembly language source files to be used with the ASM command.	PROG.ASM NEW.ASM
BAK	Filename extension for back-up of a file before it was edited by ED (text editor).	TEXT.BAK
BAS	Filename extension required for BASIC program source code files.	PROG.BAS
COM	Filename extension of transient command.	PIP.COM ED.COM
HEX	Filename extension required for files in hex format (machine language).	PROG.HEX NEW.HEX
INT	Filename extension required for BASIC program intermediate files (previously compiled).	PROG.INT
PRL	Filename extension required for MP/M relocatable programs.	TRAN.PRL
PRN	Filename extension required for the listing file of an assembly language file.	PROG.PRN

RSP	Filename extension required for MP/M resident system programs.	SPOOL.RSP
SUB	Filename extension required for text files containing CP/M commands which are to executed as a group via the SUBMIT program.	OPER1.SUB
$$$	Filename extension for temporary or scratch files which are created and then erased.	

INDEX

A Command (ED), 152, 171, 172, 280
ABORT, 99, 221, 222
Accounts Payable System, 24
Accounts Receivable System, 23
ASC II, 126, 127, 128, 129, 130, 131, 132
ASM Command, 48, 56, 77, 78, 223, 224, 225, 309
ASM, error messages, 80, 81
ASM, execution, 79
ASM, extension, 59
ATTACH, 99, 100, 226

B Command (ED), 38, 173, 174, 281
B Parameter (PIP), 136
Bad Sector Error, 305
BAK extension, 59, 309
BAS extension, 35, 59, 309
BASIC Files, 35
BDOS, 67, 191, 192, 193, 202, 204, 206, 207, 208, 209, 210, 211, 212
Binary Logic, 277
Bit, 10, 66
BIOS, 191, 192, 193, 194, 202, 204, 205, 213
Blanks, 60
Blocks, 137
Bootstrap Leader, 193, 194
Bugs, 83
Built-in Commands, 47, 61
Business Applications, 23
Byte, 10, 66
Bytes Field, 70, 73

CBAS, 283
CBAS2.COM, 36
cbase, 194, 203
CBASIC, 36, 277
CBASIC Keywords, 293
C Command, 166, 168
CCP, 83, 191, 192, 194, 201, 202, 203
Chaining, 66
Character Pointer, 150, 164
Cold Start, 33
COM extension, 35, 59, 96, 98, 309
Command, format, 48
Command, files, 35
Commands, built-in, 47, 61
Commands, transient, 61
Compiled Files, 35
Compiled Language, 36
Compiler, 278
Compiler Errors,
Compiler Listing, 283
Computer, 10
Computer Room, environment, 296
Computer Room, organization, 296
Computer Room, supplies, 299
Computer, system, 9
CON: 119, 121, 307
Concatenation, 132, 133, 134, 135
Console, 54, 92, 100, 227
Control-C, 49, 99
Control Characters, 49, 57
Control Command Processor, 83
Control-D, 49
Control-E, 49, 163
Control-H, 51, 57, 163
Control-J, 51
Control-M, 49, 51
Control-P, 50
Control-R, 50, 57, 163
Control-S, 50
Control-U, 50, 163
Control-X, 51, 163
Copying Diskettes, 42, 300
Copying, Files, 40
Copying System Diskette, 32
CP/M, business systems, 22
CP/M, defined, 14

CP/M, error messages, 305
CP/M, file structure, 195
CP/M, internal operation, 191
CP/M, operating guidelines, 295
CP/M, versions, 15
CRT: 124, 307
CRUN, 36, 284

D Command (ED), 178, 179
Daisy Wheel Printer, 21
DDT, 48, 56, 83, 84, 228, 229
Debugger, 48, 56, 83, 84
Delete Key, 51, 57
Device Assignments, 67, 68
Device Keywords, 119
Device Names, logical, 307
Device Names, physical, 307
Device Names, PIP, 307
Device Names, special, 124
DIR, 35, 48, 52, 61, 62, 100, 230, 231, 232
Disk Directory, 34
Disk Storage, 20
Disk Swapping, 114, 115, 116
Disks, Floppy, 11
Disks, Hard, 11
Diskette, Loading, 39
Diskettes, 11
Diskettes, copying, 300, 301
Diskettes, storing, 300, 301
$DIR, 72, 95, 96
$$$ Extension, 60, 310
D Parameter (PIP), 137
DSKRESET, 54, 92, 96, 97, 223
DUMP, 49, 56, 77, 82, 234
Dynamic Allocation, 67

E Command (ED), 38, 39, 51, 153, 157, 159, 160, 161, 169, 171, 182, 183, 283
ED, 37, 48, 52, 147, 148, 235
ED, error indicators, 189
ED, prompt, 149
Edit Buffer, 148, 149, 150, 280
Editor Prompt, 37
EOF, 124
EOF Character, 133
E Parameter (PIP), 137
ERA, 45, 48, 52, 63, 236, 237
Erasing Files, 45
ERAQ, 100, 238, 239
Error, bad selector 306
Error, file R/O, 306
Error Messages, CP/M, 305
Error, Read-Only, 306
EXT Field, 70, 73
Extents, 66

F Command, 179, 180, 181
File Attributes, 73, 95, 142
File Control Block, 196, 198, 199, 200
File Protection, 94
File R/O Error, 306
File Structure, CP/M, 195
Filename Extension, 57, 58
Filename Match, 42, 58
Filenames, 57
Files Erasing, 45
Files, Printing, 43
Files, Text, 147
Floppy Diskettes, 11
Flowchart Symbols, 289, 290
Format, 32
General Ledger System, 25
GENHEX, 103, 240, 243, 244
GENMOD, 92, 103, 241, 242
GENSYS, 104, 215, 217
G Parameter (PIP), 93, 142, 143
G0, 51, 144

Hard Disk, 11
Hard Disk, Storage, 20
Hardware, 19
H Command (ED), 153, 182, 183
HEX extension, 59, 309
HEX file, 79, 134, 135
H Command (ED), 153, 182, 183

I Command (ED), 38, 153, 162, 173, 280
I Parameter (PIP), 135, 138
Input Command (DDT), 84
INT extension, 35, 59, 309
Intermediate Program Files, 35
Interpreted Languages, 37
Interpreter, 278
Inventory System, 27

J Command (ED), 187, 188

K Command (ED), 176, 177, 281
Keyboard, 13
Keyword, 288
Keywords, CBASIC, 293

Language, Compiled, 36
Language, Interpreted, 37
L Command, 176
Line Numbers, 291
Line Printer, 21
Load, 48, 56, 77, 80, 245, 246
Loading, Diskette, 39
Logical Device Name, 68, 119, 307
L Parameter (PIP), 138
LPT: 44, 124, 308
LST: 119, 120, 307

Mailing List System, 27
M Command, 188
Modular Programming, 284
Modules, 284
Monitor, 278
MOVCPM, 48, 54, 213, 214, 216, 247, 248
MPM, 87
MPM, defined
MPM, installation, 214
MPMLDR, 54, 104, 215, 218, 249
MPMSTAT, 55, 92, 101, 218, 250

N Command, 184, 185
N Parameter (PIP), 138
N2 Parameter (PIP), 138
NUL, 125

Object Code, 278
O Parameter (PIP), 138

Pages, 85
Parameters, CBASIC, 288
Parameters, PIP, 136
Payroll/Personnel System, 25, 26
Physical Device Name, 68, 119, 307
PIP, 40, 48, 52, 55, 107, 108, 251, 252, 253, 254, 255
PIP, aborting, 116
PIP, copying with one drive 112, 113
PIP, copying several files, 110, 111
PIP, copying single files, 108, 109, 110
PIP, copying to other peripherals, 116, 117
PIP, device names, 307
PIP, prompt, 41
P Key, 44
P Parameter (PIP), 138
Primary Filename, 57
Printer, 13, 302
Printer, Daisy Wheel, 21
Printer, Line, 21
Printing Files, 43, 44
Priority Scheduling, 87, 88, 89
PRLCOM, 103, 256
PRL extension, 59, 96, 98, 309
PRN: 125
PRN extension, 59
PRN file, 79, 309
Program Flowcharts, 285
Programming Language, 277
Protection, files, 94
PTP: 124, 308
PTR: 124, 307
PUN: 119, 121, 126, 307

Q Parameter (PIP), 138, 140

RAM, 10
R Command (DDT), 84, 85, 86, 185, 186
RDR: 119, 121, 126, 307
RDT, 56
Read-Only Error, 306
Read-Only Files, 144
Read/Write Head, 12
Real-Time Interpreter, 278
Record, 66
Recs Field, 3, 70
Remark Statements, 292
REN 39, 48, 53, 63, 257
R/O, 66, 69
R/O $, 72, 95, 96
ROM, 10
R Parameter, 142, 145
RSP extension, 60, 310
Run-Time Errors, 278
Run-Time Monitor, 37
R/W, 66
R/W $, 72, 95, 96

$S, 72
SAVE, 48, 64, 84, 85, 86, 258, 259, 260
SCHED, 56, 92, 98, 261
Scheduling, priority, 87, 88

Scheduling, round-robin, 87, 88
S Command (ED), 181, 182, 282
Sector, 66, 195, 196
Select Error, 306
Size Field, 73
Software, applications, 14
Software, custom, 17, 18
Software, system, 17, 18
Source Code, 277
Source Code Files, 35
Source File, 157
S Parameter (PIP), 139, 140
Special Device Names, 124
Spool, 55, 92, 97, 262
Spooler, 90
STAT, 48, 53, 55, 66, 67, 68, 69, 70, 71, 72, 93, 94, 95, 96, 119, 263, 264, 265
STOPSPLR, 55, 92, 98, 261
SUB Extension, 60, 74, 75, 76, 77
SUBMIT, 48, 56, 73, 74, 75, 76, 77, 267, 268
$SYS, 72, 95, 96, 145
SYSGEN, 32, 43, 48, 54, 64, 269, 270
System Diskette, 32
System Flowcharts, 285
System Message, 31
System Prompt, 31, 33
System Shutdown, 303
System Troubleshooting, 303

T Command (ED), 38, 165, 166, 167, 174, 175
Temporary Output file, 153
Timesharing, 87
TOD, 55, 92, 98, 271, 272
TPA, 84
T Parameter (PIP), 139
Tracks, 12, 195
Transient Command, 58, 61, 64
Transient Program Area, 84, 150, 193, 194
Troubleshooting, system, 303, 304
TTY: 124, 307
Turning Off System, 46
TYPE, 44, 48, 53, 62, 101, 117, 273

U Command (ED), 163
UC1: 124, 308
UL1: 124, 308
UP1: 124, 308
UP2: 124, 308
Units, 196
U Parameter (PIP), 139
UR1: 124, 308
UR2: 124, 308
USER, 55, 91, 92, 93, 94, 274
User Areas, 91, 92, 93, 94, 141, 142

V Command (ED), 150, 174
Verify Option (PIP), 113, 139
Video Display, 13

Warm Start, 34
W Command (ED), 153, 154, 183, 184
Word Parameter (PIP), 145
Word, 10
Word Processing System, 28, 29, 147, 148
Write Protect Notch, 12, 13

X Command, 185, 186
XIOS, 213
XSUB, 75, 76, 77, 275

Z Option (PIP), 139

CP/M SIMPLIFIED

CP/M® Simplified is a clear, practical guide to the CP/M microcomputer operating system. CP/M is probably the most widely used operating systems for microcomputers. The following microcomputers can be operated under CP/M.

TRS-80 Model II	Intertec Superbrain
IBM Personal Computer	Zenith Z-89 & Z-90
Apple II	Northstar
Vector Graphics	Osborne I
Xerox 820	& Many More

For beginners, CP/M Simplified offers step-by-step instructions in the use of CP/M. No prior knowledge of computers is assumed. Everything from turning on the computer to correcting error situations is explained in simple, clear terms.

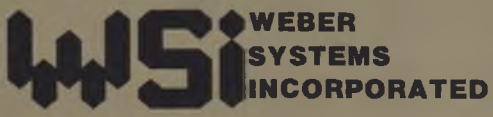
For experienced programmers, CP/M Simplified offers a complete description of all of CP/M's many advanced features. Subjects covered include:

CP/M Business Usage	CP/M Internal Operation
MP/M Operation	Introduction to CBASIC
CP/M Commands	& Many More

A handy reference guide to CP/M and MP/M commands is included along with several invaluable appendices.

No user or potential user of CP/M should be without CP/M Simplified.

*CP/M® is a registered trademark of Digital Research Inc.



ISBN: 0-938862-04-9 LC: 81-66910