Mitchell Waite • John Angermeyer

22015

cp/m® bible



The Authoritative Reference Guide to CP/M®

- Covers all CP/M built-in and transient commands
- Compares CP/M 1.3, 1.4, 2.0, 2.2 and 3.0
- Includes optional utilities MAC, SID, LIB, LINK and DESPOOL
- Describes BASIC, PASCAL, MAC, CBASIC, and CB-80
- Compares MP/M-II, CP/M-86, MP/M-86, CP/Net, and Concurrent CP/M
- Includes detailed command summary card and extensive bibliography

Harold & Martha Brown

CP/M® BIBLE

The Authoritative Reference Guide To CP/M®

BY

Mitchell Waite and John Angermeyer

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

Copyright © 1983 by Mitchell Waite

FIRST EDITION
FIRST PRINTING—1983

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-22015-6
Library of Congress Catalog Card Number: 83-61058

Edited by: *Welborn Associates*
Illustrated by: *T. R. Emrick*

Printed in the United States of America.

PREFACE

CP/M®* (Control Program for Microprocessors) is a special kind of program called a "disk operating system" that is extremely popular in personal computing and microcomputing. Due to the early introduction of CP/M to the market, almost every one of today's microcomputer and personal computer manufacturers utilizing the floppy disk offers a computer model that utilizes CP/M. Although CP/M is very popular (there are nearly 300,000 installations world wide and more than 500 suppliers of CP/M compatible programs), its strangeness to noncomputer people makes it more difficult to learn. In addition, there are hundreds of application programs, languages, and utilities that "run" under CP/M and have their own operating rules. This being the case, there is an undeniable need for a complete, clearly written and easy to use "reference guide" to the CP/M operating system and the most popular of these programs.

The CP/M® Bible is such a reference guide. It is geared to the current 2.2 version of CP/M-80 as well as the older Versions, 1.3 and 1.4. An additional overview is provided for CP/M Version 3.0. This book provides you with all the details of the various CP/M utilities, built-in commands, and transient commands. The CP/M commands covered in detail include: DIR, TYPE, ERA, REN, SAVE, and USER. Standard CP/M utilities covered include: FORMAT, SYSGEN, MOVCPM, STAT, PIP, ED, SUBMIT, XSUB, ASM, DDT, DUMP, and LOAD. In addition, these other popular CP/M utilities are cov-

ered in an appendix: DESPOOL®*, SID®*, ZSID®*, LINK-80®†, LIB-80®†, XREF®*, CREF-80®†, and XLT86®*. A second appendix overviews the start up and default procedures for the following CP/M-compatible languages: MAC®*, RMAC®*, MACRO-80®†, PL/I-80®*, BASIC-80®†, BASCOM®†, CBASIC®*, CB-80®*, Pascal/MT+®*, FORTRAN-80®†, COBOL-80®†. Another appendix is devoted to other operating systems related to CP/M, such as: MP/M-II®*, CP/M-86®*, MP/M-86®*, and CP/NET®*. Finally, four additional appendices provide command summaries, a list of CP/M prompts, ASCII crossreference information, and a bibliography.

The entire book is designed for quick and easy "look up" of any CP/M feature. A highly visual format is used to help you readily locate information. For example, each page is characterized by an icon (hieroglyphic) image in the upper corner, depicting the function being used. In addition to a standard table of contents and index, a special "Jump Table" (a sort of fancy index) at the beginning provides you with a way to find information about a particular command, and also to guide you quickly to the proper page for a given application. The page tops in *The CP/M® Bible* present the name of the CP/M facility (STAT, PIP, ED, DDT, etc.) in large letters. Numerous screens which show the different uses of a particular command and the corresponding "syntax" (wording) of the command follows. Brief, easy to follow explanations of the screens follow.

This book was specifically designed to be a reference for using CP/M-80. Although it may be all you need to master and use this most popular operating system, for first-time users of computers and CP/M, we recommend the tutorial book *CP/M® Primer* by Stephen M. Murtha and Mitchell Waite‡. For those interested in the deeper workings of CP/M, especially those programmers who want to modify their own BIOS, install a new printer, or use the CP/M system calls in their programs, we recommend the newly published companion book to this bible, titled *Soul Of CP/M®* by Mitchell Waite and Robert Lafore‡.

The authors have tried to make this book "functionally" helpful in your daily work with CP/M. We have designed it so that locating and digesting sought after information will be a quick and painless process. It is our sincere hope that this book will help you to experience CP/M as a very powerful tool that is, at the same time, fun to use.

MITCHELL WAITE AND JOHN ANGERMEYER

---

*CP/M, DESPOOL, SID, ZSID, XREF, XLT86, MAC, RMAC, PL/I-80, CBASIC, CB-80, PASCAL/MT+, MP/M-II, CP/M-86, MP/M-86, and CP/NET are registered trademarks of Digital Research, Inc.

†LINK-80, LIB-80, CREF-80, MACRO-80, BASIC-80, BASCOM, FORTRAN-80, and COBOL-80 are registered trademarks of Microsoft.

‡*CP/M Primer* (21791) and *Soul of CP/M* (22030) published by Howard W. Sams & Co., Inc.

INFORMATION JUMP TABLE

(Primary entries are in alphabetical order)
(Secondary entries are in logical order)

BACKING UP FILES AND DISKS

–Procedure	CHAPTER 2, Page 41
–Formatting The Disk (FORMAT)	CHAPTER 18, Page 277
–Copying System Image (SYSGEN)	CHAPTER 19, Page 285
–Copying Files (PIP)	CHAPTER 9, Page 117

COMMAND ACCESS

–Built-In Commands, Summary	CHAPTER 1, Page 31
–Built-In Commands, Access	CHAPTER 2, Page 45
–Transient Commands, Summary	CHAPTER 1, Page 32
–Transient Commands, Access	CHAPTER 2, Page 46
–File Access	CHAPTER 2, Page 47
–Using Wildcards	CHAPTER 2, Page 50
–Control Character Commands	CHAPTER 2, Page 52
–Optional CP/M Transient Utilities	APPENDIX A, Page 323
–CP/M-Compatible Languages	APPENDIX B, Page 347

COMMANDS

–Command Summary	APPENDIX I, Page 409

–Line Editing (Control Character) Commands	CHAPTER 2, Page 54
–Disk Directories (DIR)	CHAPTER 3, Page 68
–Renaming Files (REN)	CHAPTER 4, Page 71
–Erasing Files (ERA)	CHAPTER 5, Page 75
–Displaying File Contents (TYPE)	CHAPTER 6, Page 81
–Changing Disk User Areas (USER)	CHAPTER 7, Page 85
–Displaying and Changing Disk and File Status (STAT)	CHAPTER 8, Page 91
–Copying Files (PIP)	CHAPTER 9, Page 117
–Editing Text (ED)	CHAPTER 10, Page 157
–Submitting CP/M Commands (Automatically) (SUBMIT)	CHAPTER 11, Page 203
–Submitting commands and entries to programs being submitted (XSUB)	CHAPTER 12, Page 213
–Assembling Programs (ASM)	CHAPTER 13, Page 219
–Converting HEX files to COM files (LOAD)	CHAPTER 14, Page 237
–Modifying Object Code Files (DDT)	CHAPTER 15, Page 241
–Saving Data Into Files After DDT, SYSGEN, MOVCPM, and other commands (SAVE)	CHAPTER 16, Page 269
–Displaying Contents of Object Code Files (DUMP)	CHAPTER 17, Page 273
–Formatting New Disks (FORMAT)	CHAPTER 18, Page 277
–Copying The System Image (SYSGEN)	CHAPTER 19, Page 285
–Changing System Image To Recognize More or Less Memory (MOVCPM)	CHAPTER 20, Page 295
–CP/M Version 3.0 Commands	CHAPTER 21, Page 305

CP/M (GENERAL)

–Using CP/M	CHAPTER 1, Page 28
–Command Summaries	CHAPTER 1, Page 31
–CP/M Prompts	APPENDIX G, Page 403
–Overview Of CP/M Version 3.0	CHAPTER 21, Page 305
–Optional CP/M Transient Utilities	APPENDIX A, Page 323
–CP/M-Compatible Languages	APPENDIX B, Page 347
–CP/M-Related Operating Systems	APPENDIX C, Page 369

DEVICES

–CP/M Devices Overview	CHAPTER 1, Page 23
–Logical And Physical Device Definitions (STAT)	CHAPTER 8, Page 93
–Verifying And Making Device Assignments (STAT)	CHAPTER 8, Page 110
–Special Input Devices (PIP)	CHAPTER 9, Page 133
–Special Output Devices (PIP)	CHAPTER 9, Page 131
–Copying From Devices (PIP)	CHAPTER 9, Page 135
–Copying To Devices (PIP)	CHAPTER 9, Page 131

FILE ACCESS CHAPTER 2, Page 47

LANGUAGES, PROGRAMMING

–CP/M Assembler (ASM)	CHAPTER 13, Page 219
–Digital Research Macroassembler (MAC)	APPENDIX B, Page 348
–Digital Research Relocatable Macroassembler (RMAC)	APPENDIX B, Page 350
–Microsoft Macroassember (MACRO-80)	APPENDIX B, Page 352
–Digital Research PL/I-80	APPENDIX B, Page 354
–Microsoft BASIC Interpreter (BASIC-80)	APPENDIX B, Page 355
–Microsoft BASIC Compiler (BASCOM)	APPENDIX B, Page 357
–Digital Research BASIC Compiler (CBASIC)	APPENDIX B, Page 359
–Digital Research BASIC Native Code Compiler (CB-80)	APPENDIX B, Page 360
–Digital Research Pascal (Pascal/MT+)	APPENDIX B, Page 362
–Microsoft FORTRAN (FORTRAN-80)	APPENDIX B, Page 365
–Microsoft COBOL (COBOL-80)	APPENDIX B, Page 366

GENERAL REFERENCE

–CP/M Command Summary	APPENDIX I, Page 409
–A Comparison Of CP/M Versions	CHAPTER 2, Page 57
–Standard File Types	CHAPTER 2, Page 54
–Glossary Of Commonly Used Terms	APPENDIX E, Page 391
–CP/M-Related Operating Systems	APPENDIX C, Page 369
–Annotated Bibliography	APPENDIX D, Page 375
–ASCII Crossreference	APPENDIX F, Page 395

MICROCOMPUTER OPERATING SYSTEMS

–General	CHAPTER 1, Page 30
–CP/M (History and Overview)	CHAPTER 1, Page 23
–Other CP/M-Based Operating Systems	CHAPTER 1, Page 30
–CP/M Version 3.0	CHAPTER 21, Page 305
–A Comparison Of CP/M Versions	CHAPTER 2, Page 57
–Evolution Of CP/M-Related Operating Systems	APPENDIX C, Page 369
–CP/M-86	APPENDIX C, Page 371
–Concurrent CP/M-86	APPENDIX C, Page 372
–MP/M and MP/M-II	APPENDIX C, Page 370
–MP/M-86	APPENDIX C, Page 372
–CP/NET	APPENDIX C, Page 373

PROCEDURES

–How To Use This Book	CHAPTER 1, Page 23
–Using CP/M	CHAPTER 1, Page 24
–CP/M Prompts	APPENDIX G, Page 403
–Booting The CP/M System	CHAPTER 2, Page 38
–The Bootstrap Loader	CHAPTER 2, Page 38
–The Monitor Program	CHAPTER 2, Page 39
–Booting CP/M (Cold Boot)	CHAPTER 2, Page 39
–The CP/M Sign-On Message	CHAPTER 2, Page 40
–Backing Up The CP/M System Disk And Other Disks	CHAPTER 2, Page 41
–Warm Boot	CHAPTER 2, Page 42
–Using Disk Drives	CHAPTER 2, Page 43
–Accessing Files	CHAPTER 2, Page 47
–Using Wildcards	CHAPTER 2, Page 50
–Control Character Commands	CHAPTER 2, Page 52

STANDARD FILE TYPES CHAPTER 2, Page 54

UTILITIES, OPTIONAL

–Digital Research Despooler (DESPOOL)	APPENDIX A, Page 325
–Digital Research 8080/8085 Symbolic Instruction Debugger (SID)	APPENDIX A, Page 326
–Digital Research Z80 Symbolic Instruction Debugger (ZSID)	APPENDIX A, Page 328

–Digital Research Linker/Loader (LINK-80) APPENDIX A, Page 330
–Microsoft Linker/Loader (LINK-80) APPENDIX A, Page 333
–Digital Research Librarian (LIB-80) APPENDIX A, Page 336
–Microsoft Librarian (LIB-80) APPENDIX A, Page 338
–Digital Research Crossreference Utility (XREF) APPENDIX A, Page 340
–Microsoft Crossreference Utility (CREF-80) APPENDIX A, Page 341
–Digital Research 8080 to 8086/8088 Assembly Language Translator (XLT86) APPENDIX A, Page 343

WILDCARDS
–Substituting Groups Of Characters CHAPTER 2, Page 50
–Substituting Individual Characters CHAPTER 2, Page 51

CONTENTS

CHAPTER 1: INTRODUCTION

PURPOSE OF THIS BOOK 23
HOW TO USE THIS BOOK 23
MICROCOMPUTER OPERATING SYSTEMS 26
THE CP/M OPERATING SYSTEM 28
OTHER CP/M-BASED OPERATING SYSTEMS 30
SUMMARY OF CP/M COMMANDS 31
Built-In (Direct) Commands 31
Transient Commands 32
Special Control-Character Commands 35

CHAPTER 2: USING CP/M

GENERAL 37
BOOTING THE CP/M SYSTEM 38
The Bootstrap Loader 38
The Monitor Program 39
Booting The CP/M Operating System 39
The CP/M Sign-On Message 40
Backing Up The System Disk 41
Memory Management 42
Warm Boot 42

USING DISK DRIVES 43
ACCESSING COMMANDS AND FILES 45
Built-In Commands 45
Transient Commands 46
Accessing Files 47
USING WILDCARDS 50
Using The "*" Star Wildcard 50
Using The "?" Wildcard 52
CONTROL CHARACTER COMMANDS 52
Warm Boot Command 53
Line Termination Commands 53
Character Deletion Commands 53
Line Deletion Commands 54
Display Control Commands 54
Printer Control Command 55
STANDARD FILE TYPES 55
A COMPARISON OF CP/M VERSIONS 57
Built-In Commands 58
Transient Commands 59
Control Character Commands 61
Disk Specifications 62
New Or Changed BDOS Functions 63

CHAPTER 3: THE DIR COMMAND

SUMMARY 67
ACCESSING DIR 68
DISPLAYING THE COMPLETE DIRECTORY OF A DISK . . . 68
DISPLAYING THE DIRECTORY OF A SPECIFIC FILE OR GROUP OF FILES 69
PAUSING THE DISPLAY OF THE DIRECTORY 70
STOPPING THE DIR COMMAND 70

CHAPTER 4: THE REN COMMAND

SUMMARY 71
ACCESSING REN 71
RENAMING A FILE ON THE LOGGED DRIVE 71

RENAMING A FILE ON ANOTHER DRIVE 73

CHAPTER 5: THE ERA COMMAND

SUMMARY 75
ACCESSING ERA 75
ERASING A SINGLE FILE 75
ERASING GROUPS OF FILES 77
ERASING ALL FILES 78

CHAPTER 6: THE TYPE COMMAND

SUMMARY 81
ACCESSING TYPE 81
DISPLAYING THE CONTENTS OF A FILE USING TYPE ... 81
PAUSING THE DISPLAY OF A TEXT FILE 82
STOPPING THE TYPE COMMAND 83

CHAPTER 7: THE USER COMMAND

SUMMARY 85
ACCESSING USER 85
CREATING OR CHANGING USER AREAS 86
CREATING A USER AREA FOR THE FIRST TIME 87
CHANGING USER AREAS 88
DISPLAYING THE ACTIVE USER NUMBER AND THE NUMBER OF USER AREAS THAT CONTAIN FILES ... 88

CHAPTER 8: THE STAT COMMAND

SUMMARY 91
ACCESSING STAT 93
DESCRIPTION OF STAT COMMAND VARIATIONS 94
Displaying Unused Space And Protection Status On All Disks .. 94
Displaying Unused Space And Protection Status On A Particular Disk 95

Displaying The Status Of Specific Files 96
Displaying The Status Of All Files 98
Protecting Files 100
Protecting Complete Disks Temporarily 101
Hiding Files 103
Verifying Number Of Disk Users 105
Displaying Disk Drive Characteristics 106
Displaying Device Assignments 110
Displaying Allowed Device Names And Other Allowed STAT Commands 112
Assigning Physical Devices To Logical Names 115

CHAPTER 9: THE PIP COMMAND

SUMMARY 117
ACCESSING AND USING PIP 119
COPYING FILES BETWEEN DISK DRIVES 120
Loading PIP And Performing Copy Operations 120
Copying Nonexistent Files 125
Over-Writing Existing Files During A Copy Operation 125
Copying Protected Files 126
Copying System Files 127
Initiating Single-Line PIP Commands 127
Copying Several Files Into One File 128
USING STANDARD AND SPECIAL DESTINATION DEVICES WITH PIP 130
Copying To The Console Destination Devices (Video Terminal Or Teletypewriter) 131
Copying To The Output-Only Devices (Paper Tape Punch Or Other Device) 131
Copying To The List Devices (Line Printer Or Other Device) . . . 132
USING STANDARD AND SPECIAL SOURCE DEVICES 133
Copying Input On The Console Device Into A File On A Disk 134
Copying From An Input-Only Device Into A File On A Disk . . 135
Specifying Leaders And Trailers While Copying Source Files To The Output-Only Device 136
Specifying An End-Of-File Marker Special Source Device While Copying A File To The Output-Only Device 136

OPTIONAL PIP PARAMETERS 137
Verify Accuracy Of Copied Data By Comparing The Destination File With The Source File 137
Display Or Echo All Text On The Console Device As It Is Being Copied 138
Begin Copying A File After A Specified String Of Text 140
Quit Copying A File After A Specified String Of Text 141
Using Both The "Start" And "Quit" Parameters To Copy Specified Text 142
Convert All Upper-Case Characters To Lower Case During A Copy Operation 143
Convert All Lower-Case Characters To Upper Case During A Copy Operation 145
Delete All Characters On Each Line Of Text After A Specified Column 146
Convert ASCII Tab Characters To A Specified Number Of Spaces 147
Delete All Form-Feed Characters During A Copy Operation . . . 148
Insert Form-Feed Characters After Each Specified Number Of Lines During A Copy Operation 149
Add Sequential Line Numbers To Text During A Copy Operation 150
Copy From A Source "System" File 152
Erase A Read-Only File Having The Same Name As The Destination File Name 152
Copy Files From Other Disk User Areas 153
Interpret Files Being Copied As Non-ASCII Object/Machine Code Files 153
Specify "Block Mode" Format During A Copy Operation 155
Specify Intel Hexadecimal Format During A Data Transfer Operation 155
Ignore Any Null Records In Intel Hexadecimal Format During A Copy Operation 156
Set The Parity Bit Of All ASCII Bytes To Zero During A Copy Operation 156

CHAPTER 10: THE ED COMMAND

SUMMARY 157

GENERAL DESCRIPTION AND PROGRAM ACCESS 158
Initiating ED 160
Editing A File 160
Terminating ED And Saving The File 161
THE ED CHARACTER POINTER (CP) 162
DESCRIPTION OF ED COMMANDS 164
Appending Specific Lines Of Text From Source File Into The Editing Buffer 164
Appending Portions Or All Of Text From Source File Until The Editing Buffer Is Half Or Completely Full 166
Inserting Text Into The Editing Buffer 167
Engage Or Disengage Character Translation To Upper Case ... 172
Move CP Backwards Or Forwards One Or Several Characters On A Line 173
Move CP Backwards Or Forwards One Or Several Lines 174
Display Or Suppress Displaying Line Numbers 175
Move CP To A Specific Line Number 176
Move CP To Beginning Or End Of Editing Buffer 177
Display Text Before Or After CP 177
Move CP One Or Several Pages And Display Text 180
Delete One Or Several Characters On A Line Before Or After CP 181
Delete (Kill) One Or Several Lines Before Or After CP 182
Find Text String In Editing Buffer (Once Or Several Times) ... 183
Find Text String In Editing Buffer (Once Or Several Times), And Perform Append And Write Operations As Necessary 184
Find Text String In Editing Buffer And Substitute It With Specified Text (Once Or Several Times) 185
Find Text String In Editing Buffer And Juxtaposition It With Specified Text (Once Or Several Times) 186
Combining CP Movement Commands 187
Macro Repetition Of Command Groupings 189
Verify Total Editing Buffer Usage And Free Space 191
Write Several Lines From Editing Buffer To Temporary Output File 192
Write Several Lines From Editing Buffer To Temporary Library File 193
Read Contents Of Temporary File Created By The nX Command Into Editing Buffer 195
Read Contents Of Library File Into Editing Buffer 196

Save All Editing In Final File Without Exiting ED 197
Exit ED (Return To CP/M) And Save All Editing In Final File 198
Erase All Editing In Editing Buffer And Final File Without Exiting ED 199
Quit ED (Return To CP/M) And Erase All Editing In Editing Buffer And Final File 199
ED ERROR MESSAGES 200

CHAPTER 11: THE SUBMIT COMMAND

SUMMARY 203
ACCESSING AND USING SUBMIT 204
USING VARIABLES IN .SUB FILES 207
INSERTING COMMENTS IN .SUB FILES 209
SPECIAL SUBMIT FEATURES 211

CHAPTER 12: THE XSUB UTILITY

SUMMARY 213
PROGRAM ACCESS AND OPERATION 215
USING XSUB 215

CHAPTER 13: ASM COMMAND

SUMMARY 219
PROGRAM ACCESS 221
USING ASM 221
ASSEMBLING A FILE ON THE DEFAULT DRIVE 222
ASM AND CP/M CONSOLE ERROR MESSAGES 226
NO SOURCE FILE PRESENT 226
SOURCE FILE NAME ERROR 226
SOURCE FILE READ ERROR 226
OUTPUT FILE WRITE ERROR 227
CANNOT CLOSE FILE 227
NO DIRECTORY SPACE 227
SOURCE CODE PROGRAM (.ASM) ERROR MESSAGES 227

ASM SOURCE PROGRAM FORMAT 229
ASM SOURCE PROGRAM CONSTANTS AND EXPRESSIONS 230
ASM DIRECTIVES 233

CHAPTER 14: THE LOAD COMMAND

SUMMARY 237
PROGRAM ACCESS 237
FIRST ADDRESS 238
LAST ADDRESS 238
BYTES READ 238
RECORDS WRITTEN 238

CHAPTER 15: THE DDT COMMAND

SUMMARY 241
PROGRAM ACCESS AND DESCRIPTION 242
Loading DDT Into Memory Without Specifying A File For Examination And Modification 243
Loading DDT Into Memory By Specifying A File For Examination And Modification 244
COMMAND DESCRIPTIONS 245
Displaying The Contents Of Portions Of Memory 245
Filling The Memory With Constant Data 248
Inserting And Establishing A File Control Block For Referencing A File To Be Loaded 249
Reading A File On A Disk And Loading It Into Memory 251
Disassembling And Displaying Portions Of Memory With Assembly Code 253
Display And Optionally Change The Contents Of A Location In Memory 255
Insert Assembly Language Statements At Specific Locations In Memory 256
Copy The Contents Of A Specific Memory Block To Another Block Starting At A Specific Address 258
Execute Portions Of A Program In Memory And Set Breakpoints 259

Execute A Number Of Program Instructions In Memory And Display The Contents Of The CPU Registers 261
Trace The Execution Of A Number Of Program Instructions ... 263
Display The Contents Of The CPU Registers Or Specific Register, And Optionally Make Changes 264
Use The Built-In Calculator For Adding And Subtracting Hexadecimal Numbers 266

CHAPTER 16: THE SAVE COMMAND

SUMMARY 269
PROGRAM ACCESS 269

CHAPTER 17: THE DUMP COMMAND

SUMMARY 273
PROGRAM ACCESS 273

CHAPTER 18: THE FORMAT COMMAND

SUMMARY 277
ACCESSING THE DISK FORMATTING PROGRAM 278
DISK DENSITY 279
FORMATTING A SOFT-SECTORED FLOPPY DISK 280
OTHER FORMATTING PROGRAMS 283

CHAPTER 19: THE SYSGEN COMMAND

SUMMARY 285
ACCESSING SYSGEN 286
DESCRIPTION OF SYSGEN COMMAND VARIATIONS 287
Copying The CP/M System Image 287
Saving The CP/M System Image In A File 289
Copying A CP/M System Image From A File 292

CHAPTER 20: THE MOVCPM COMMAND

SUMMARY 295
ACCESSING MOVCPM 296
DESCRIPTION OF MOVCPM COMMAND VARIATIONS 297
Move CP/M System Image Using All Memory And Transfer Control 297
Move CP/M System Image Using Specified Memory And Transfer Control 298
Move CP/M System Image Using All Of Memory And Prepare System Image For Saving In A File 300
Move CP/M System Image Using Specified Amount Of Memory And Prepare System Image For Saving In A File 302

CHAPTER 21: AN OVERVIEW OF CP/M VERSION 3.0

INTRODUCTION 305
OPERATING ENVIRONMENT 305
USING CP/M PLUS 306
CP/M PLUS COMMAND OVERVIEWS 310

APPENDICES

APPENDIX A: OPTIONAL TRANSIENT UTILITIES

INTRODUCTION 323
DESPOOL (BACKGROUND PRINT) UTILITY 325
SID (8080/8085 SYMBOLIC INSTRUCTION DEBUGGER) UTILITY 326
ZSID (Z80 SYMBOLIC INSTRUCTION DEBUGGER) UTILITY 328
LINK-80 (LINKING/LOADER) UTILITY 330
Digital Research LINK-80 Summary 330
Microsoft LINK-80 Summary 333
LIB-80 (LIBRARIAN) UTILITY 336

Digital Research LIB-80 Summary 336
Microsoft LIB-80 Summary 338
XREF (CROSSREFERENCE) UTILITY 340
CREF-80 (CROSSREFERENCE) UTILITY 341
XLT86 (8080 TO 8086 ASSEMBLY LANGUAGE TRANSLATOR) UTILITY 343

APPENDIX B: CP/M-COMPATIBLE LANGUAGES

INTRODUCTION 347
MAC (MACROASSEMBLER) 348
RMAC (RELOCATABLE MACROASSEMBLER) 350
MACRO-80 (MACROASSEMBLER) 352
PL/I-80 (8080 PROGRAMMING LANGUAGE/ONE) 354
BASIC-80 (BASIC LANGUAGE INTERPRETER) 355
BASCOM (BASIC-80 COMPILER) 357
CBASIC2 (COMPILER BASIC LANGUAGE) 359
CB-80 (NATIVE CODE COMPILER FOR CBASIC) 360
PASCAL/MT+ (PASCAL PROGRAMMING LANGUAGE) . . . 362
FORTRAN-80 (8080 FORMULA TRANSLATOR LANGUAGE) 365
COBOL-80 (8080 COMMON BUSINESS ORIENTATED LANGUAGE) 366

APPENDIX C: CP/M-RELATED OPERATING SYSTEMS

INTRODUCTION 369
THE MP/M-II OPERATING SYSTEM 370
THE CP/M-86 OPERATING SYSTEM 371
THE MP/M-86 OPERATING SYSTEM 372
THE CP/NET NETWORK OPERATING SYSTEM 373

APPENDIX D: ANNOTATED BIBLIOGRAPHY

BOOKS ON CP/M 375

MAGAZINE ARTICLES ON CP/M 376
ASSEMBLY LANGUAGE PROGRAMMING BOOKS 389

APPENDIX E: GLOSSARY OF COMMONLY USED TERMS

APPENDIX F: ASCII CROSSREFERENCE

APPENDIX G: CP/M PROMPTS

APPENDIX H: CP/M ERROR MESSAGES

APPENDIX I: COMMAND SUMMARY

CHAPTER 1

Introduction

PURPOSE OF THIS BOOK

The purpose of this book is to provide a complete and easy-to-use reference source to the CP/M® operating system. If you are unfamiliar with CP/M, or microcomputer operating systems in general, you will find that this book provides detailed information about the use of typical microcomputer operating system commands and facilities. More specifically, you will learn how to use the commands provided by CP/M. This book is not, however, a tutorial on CP/M or about operating systems, in general. Its primary purpose is to offer the CP/M user easy-to-find reference information. (For a tutorial on CP/M, see *CP/M® Primer* by Stephen Murtha and Mitchell Waite, Howard W. Sams & Co., Inc.).

Throughout this book, each command described is presented in a clear and concise manner, and simple examples of command usage are provided where appropriate.

HOW TO USE THIS BOOK

Each CP/M command described in this book is presented in large, boldface letters at the top of the page where it first appears. This will facilitate thumbing through the book to find the command you need to reference, rather than always needing to use the index. Variations in the way each com-

mand can be used are also described in their own mini section. The standard format in which the command variation should be typed on the keyboard is presented at the beginning of each section in boldface letters and enclosed in a box. This further facilitates the referencing of commands, since not only will you easily be able to find the entire command description in the book, but specific variations of the command as well.



Command usage variation examples are presented with a top left corner of a screen as shown above. Next to these screen corners, the text is presented in a form virtually identical to the way you yourself would type the commands on any CP/M console device, be it a video terminal, a teleprinter, or a teletypewriter. Any text displayed by CP/M in response to a command entered by a user is replicated as it would be seen on a console device.

Throughout these descriptions of commands, appropriate informational references are made to other chapters, sections, and appendices of this book.

The chapter organization in this book has been fundamentally designed to reflect the way in which the CP/M user would most want to see the commands ordered. That is, all commands follow a logical sequence. No attempt has been made to group together built-in commands in a single group, as well as transient commands. The most commonly used commands are presented in the first chapters. The following are chapter summaries:

CHAPTER 2–USING CP/M: This chapter deals with general information about the CP/M operating system, how it is installed and how commands are accessed. It also includes various command form conventions that should be kept in mind while referencing the rest of the book. Even though this book is designed to be picked up and used as a ready reference book, it is a good idea to read this chapter first.

CHAPTERS 3 THROUGH 20: These chapters describe all of the CP/M built-in and transient commands. The command chapters are presented in a logical order so that not only can they be referenced easily, but they can also

be read in sequence comfortably. Overviews of each command are presented at the end of this chapter.

CHAPTER 21–OVERVIEW OF CP/M VERSION 3.0: This chapter provides an overview of Version 3.0 of CP/M-Plus, describing basic operating characteristics of the system. Short descriptions of each of the commands are also provided.

In addition to the four chapters, this book also contains several appendices which provide supplemental reference information. All of this information is not directly related to CP/M commands, but is often useful to have handy while working in the CP/M environment. These appendices consist of the following:

APPENDIX A – OPTIONAL TRANSIENT UTILITIES: This appendix contains concise and basic start-up reference information on transient command programs that may be optionally added to CP/M. Most of these programs consist of utilities that enhance the capabilities of the CP/M operating system as a program development tool.

APPENDIX B – CP/M-COMPATIBLE LANGUAGES: This appendix consists of concise and basic start-up reference information on several popular high-level programming languages compatible with the CP/M operating system.

APPENDIX C – CP/M-RELATED OPERATING SYSTEMS: This appendix briefly describes the characteristics of the MP/M-II, CP/M-86, MP/M-86, and CP/NET operating systems, and how they relate with CP/M.

APPENDIX D – ANNOTATED BIBLIOGRAPHY: This appendix provides a list of books and journal articles that directly relate to CP/M and its various uses.

APPENDIX E – GLOSSARY OF COMMONLY USED TERMS: This appendix provides a short list of commonly used terms, abbreviations, acronyms and mnemonics relating directly to CP/M, operating systems and computer systems, in general.

APPENDIX F – ASCII CROSS REFERENCE: This appendix describes the American Standard Code For Information Interchange (ASCII), and defines its various codes. A valuable table of hexadecimal, decimal, and octal values are given along with the codes.

APPENDIX G – CP/M PROMPTS: This appendix lists the various prompts that CP/M will display or print on the console device. Prompts are output indicators that CP/M displays to you depending on the program being executed. These prompts consist of those generated by CP/M itself, and others that are generated by various commands.

APPENDIX H – ERROR MESSAGES: This appendix consists of error messages generated by CP/M. Basic information is provided as to possible causes of a given error message, and possible remedies to alleviate the error condition.

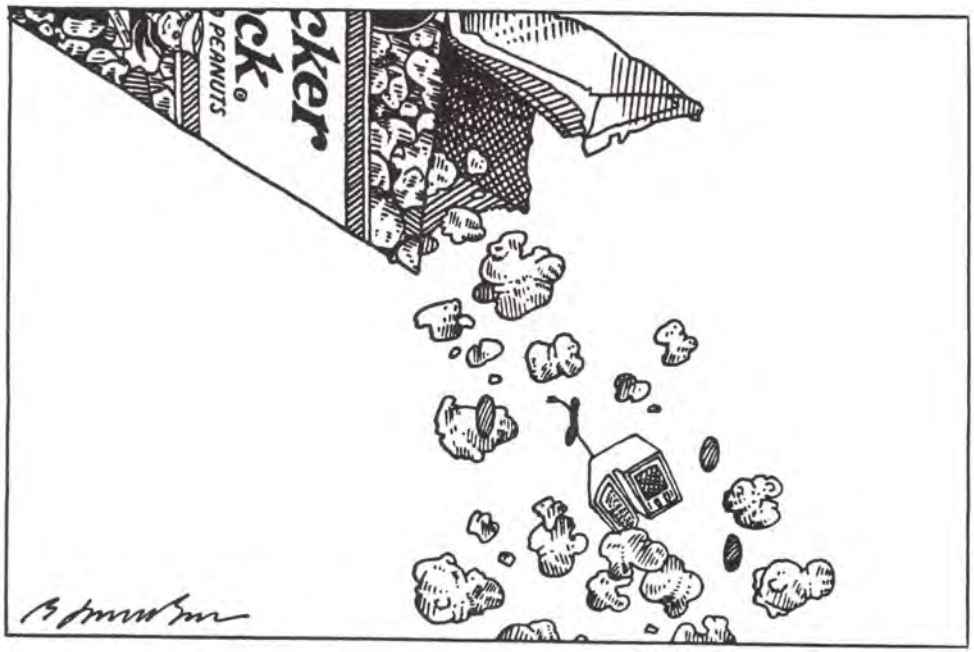
APPENDIX I – COMMAND SUMMARY: This appendix provides a complete summary of all of the CP/M commands, and command variations described in this book.

MICROCOMPUTER OPERATING SYSTEMS

From the early days of computer development to the present, there has always existed the need for supervisory programs to fulfill many requirements of computer system control and management. To fulfill this need, control programs were designed to function as "overseers" in the computer environment. These "control programs" provided guidelines and procedure conventions for using the various facilities of the computer system. They set the rules for doing those things on the computer system that so frequently needed to be done, such as determining exactly what information is on the disk media (i.e., what disk has my program), creating files, removing files, running programs, measuring the space on the disk media, and so on. Throughout the years, the term "operating system" has been commonly used to refer to this type of control program. In general, you can think of the operating system as a program that all other programs need before they can run. The OS (or DOS if a disk is being used) is always in the computer while it is turned on. Other programs come and go.

The operating system is almost always the first and most essential program to be installed in a computer, whether the computer be a large main-frame system, a minicomputer, or a microcomputer. Since the mid-1970s, the operating system has, for the most part, become essential to the operation of microcomputers. Although some computers do not use an operating system per se, the lack of such a control program often restricts considerably the use of these computers for more serious applications. In the case of CP/M, it was designed around a specific media, namely floppy disk. Therefore its structure

is somewhat mandated by the way the floppy is organized, its maximum storage capacity, and so on. The CP/M operating system will not work with a tape-cassette system, but will work with a hard disk.



Three primary functions are performed by most operating systems, as follows.

1. Provide orderly and consistent input/output (I/O) routines for the various peripheral devices of the computer (i.e., terminals, printers, hard or floppy disk drives, magnetic tape drives, etc.). The term "I/O" generally refers to the method by which the computer communicates with the various peripheral devices connected to it.
2. Provide "file" management and status reporting for the data stored in the computer. The file management system provides the user with a method for obtaining information on the files stored on the disk media. Such information includes file names, directories of files, file sizes, unused storage space, etc. These are things that are necessary information for manipulating the files on the disk. This file management system also interacts with the disk operating function which is used to read from and write to storage media devices like floppy or hard disks, etc.
3. Provide a method for the loading and executing of standard system commands and user-specified programs.

Many operating systems have sophisticated features. These include providing a method by which several programs or tasks can be executed simultaneously, a method to keep track of the amount of time each user spends on the system, reporting percentage of processor time used by a program, password system for computer security, etc. In spite of the fancy variations, however, all operating systems (including CP/M, which was originally designed for use with microcomputer systems), perform the three basic functions just mentioned in one form or another.

THE CP/M OPERATING SYSTEM

In 1973, the Intel 8080 microprocessor became the first device powerful enough to be used in a microcomputer. Several companies started manufacturing small computer systems based on this integrated circuit. At the beginning, these systems were rather primitive in that no compatible operating systems were immediately available. Since then, however, several operating systems have been developed for specific use with microcomputers based on the 8080, as well as systems based on other microprocessors which have been developed since then. Among these operating systems is CP/M, introduced in 1975 by Gary Kildall, who, in 1976, founded Digital Research, Inc. CP/M was originally designed for use with 8080 microprocessor-based computers, although later compatible microprocessors, the Intel 8085 and the Zilog Z80, also support CP/M. Since then, CP/M has become one of the most popular and widely used operating systems, and is regarded by many as an "industry standard" since it has been used by many microcomputer manufacturers and supports literally hundreds of different professional application programs developed by microcomputer software manufacturers.

The unprecedented popularity of CP/M is due to the ease of its operation, its simplicity and brief set of commands, and its built-in flexibility for adaptation to various microcomputer system environments. Several standard commands and utility programs are provided with the CP/M operating system. Some of these are file directory and file manipulation commands, file, device and free-space status commands, a text editor, file copying, etc. Most of them serve the same purposes as similar commands in other operating systems. Additional programs, such as a dynamic debugging utility and program loading utility (mainly for programmers), are provided which make CP/M a useful and powerful programmer's tool.

CP/M was originally developed for microcomputers using floppy disk drives, and thus is, in reality, a "DOS" (Disk Operating System). The IBM

3740 standard was the standard floppy disk format adopted for use by CP/M. This standard was established by the International Business Machines (IBM) corporation, and consists of a single-density, soft-sectored, 128-byte per sector format, for use by single-sided 8-inch floppy disks, commonly referred to in the industry as the "single density 8 inch standard." Since it has been adopted by many microcomputer, minicomputer, and mainframe computer manufacturers as "the" standard for 8-inch floppy disks, most manufacturers today set up their computers to provide compatibility with this hardware disk standard. By using this standard, floppy disks can be interchanged between almost all CP/M microcomputers. Since there are large numbers of floppy disk CP/M users, there is a large incentive to make your basic software product run under this DOS. This greatly facilitates the distribution of software stored on floppy disk media throughout the microcomputer industry and the CP/M-based microcomputer user network.

Since the mid-1970s, many manufacturers have developed other types of floppy disk drives utilizing various formats. (5¼-inch (mini) soft-sectored, single- and double-sided floppy disk drives are examples.) Developments of the 8-inch soft-sectored floppy disk have also advanced to support single-sided, double-density, as well as double-sided formats. Some manufacturers have adopted the hard-sectored format in both the 5¼-inch and 8-inch sizes. Although the IBM 3740 standard still remains the standard for maintaining single density compatibility between CP/M-based systems, the CP/M operating system can be modified to add compatiblity with various other floppy disk drive hardware. In addition to floppy disks, CP/M can also be made compatible for use with several types of hard disks, including Winchester disk drives. (Systems with at least one 8-inch single-density drive have access to the largest number of CP/M programs. Systems with special formats usually require the user either to buy the CP/M software from the computer manufacturer or wait till the software manufacturer comes out with a version for your computer. A link can be established between CP/M computer systems with different disk formats and files can be transferred. This link would be through a serial port. See *CP/M Primer* for details.)

The ability of CP/M to be modified for supporting various types of floppy and hard disk drive hardware and formats certainly makes it a versatile system. In addition, CP/M also supports a variety of peripheral devices (i.e., teletypewriters, teleprinters, video terminals, color CRT's, memory mapped devices, paper tape reader and punch machines, modems). Apart from the disk drives, the standard CP/M configuration supports four main peripheral devices:

CONSOLE DEVICE: The console device is the standard peripheral designed for user interaction with CP/M and the microcomputer. It may consist of a teletypewriter (TTY device) or a video terminal (CRT device).

INPUT-ONLY DEVICE: The input-only device was originally provided for use by a high-speed paper-tape reader machine. CP/M, however, can be modified so that other user-defined input-only devices can be used, such as a modem, or some other kind of link to another computer, etc.

OUTPUT-ONLY DEVICE: The output-only device was also originally provided for use by a high-speed paper-tape punch machine. Again, CP/M can be modified so that other user-defined output-only devices can be used.

LISTING DEVICE: The listing device is a general name for any device that generates hard copy output. This might include a send/receive teleprinter or a receive-only printing device (high-speed dot matrix or letter-quality impact printing devices).

OTHER CP/M-BASED OPERATING SYSTEMS

The burgeoning popularity of microcomputers and operating systems over the last several years, has precipitated several significant technological advancements in microcomputer design and use by operators. These advancements have led to more and more people using these computers at the same time, and this has necessitated the further development of the power of operating systems. Since CP/M is only a "single-user" operating system, and can only handle the execution of one program or task at a time, it has been useless for many applications that have required low cost (microprocessor-based) multiuser (more than one user at a time) and multitasking (the ability of a user to run more than one program at the same time) capabilities.

Since 1977, Digital Research has responded to this need with the MP/M operating system, which is based on CP/M and provides multiuser and multitasking capabilities. Also in recent years, progress has been made in the concepts of distributed processing and microcomputer networks. For this purpose, Digital Research developed CP/NET: A networking control program for CP/M and MP/M-based microcomputers connected together in a network.

Although the Intel 8080 microprocessor and its close cousins, the Intel 8085 and Zilog Z80, are still very popular today, many technological

advancements have been made to increase the power of integrated circuit microprocessors. Since 1977, several 16-bit, and 32-bit microprocessors have been developed, including the Intel 8088 and 8086, the Motorola 68000, the Zilog Z8000, and the National Semiconductor 16000. (Intel is currently developing a full 32-bit microprocessor called the iAPX-432.)

The Intel 8086 was one of the first successful 16-bit microprocessors. Because it is similar in some ways to the 8080, Digital Research developed CP/M-86 and MP/M-86, the 16-bit counterparts of CP/M and MP/M. Also, recently developed is Concurrent CP/M-86, a multitasking operating system providing concurrency (real-time) processing for microcomputer development applications. These systems were specifically designed to operate with the 8086 and 8088 microprocessors.

The new generation of 16-bit and 32-bit microprocessors consists of such a wide variety of microprocessor architectures, that it is virtually impossible to create a universal operating system for them all. Digital Research recently announced the support of the Motorola 68000 with CP/M-68K, and plans to support National 16000 with a special version of CP/M. (Such a version of CP/M may be so powerful as to rival the large mainframe operating systems). In spite of these exciting new announcements, there is more to come in the whole field of microprocessor, microcomputer, and operating system development.

This book was written specifically for the CP/M operating system for 8080, 8085, and Z80 microprocessor-based microcomputers. Concise comparative information is provided in Appendix C at the back of the book for the MP/M, CP/M-86, MP/M-86, and CP/NET operating systems.

SUMMARY OF CP/M COMMANDS

There are two types of commands provided by the CP/M operating system: Built-In (Direct) Commands and Transient Commands. Built-in commands are part of the CP/M operating system and are always stored in with it in the same area of memory. Transient commands are those that are always stored as files on a disk, and a particular transient command file must be present in order for the command to be used. Each of the built-in commands and the standard CP/M transient commands are described in their respective chapters, Chapters 3 through 20.

BUILT-IN (DIRECT) COMMANDS: The CP/M operating system contains a section called the Console Command Processor (CCP). The CCP is a

"program" in CP/M that is always checking the keyboard to see if certain letters are typed that are commands to CP/M. The CCP contains a set of built-in or "resident" commands (ones that can be accessed without being on the disk media). These built-in commands are used by typing the command name in response to the CP/M prompt (A>, B>, etc.). Because the commands are not contained in individual files, they are available for use regardless of the logged drive, that is, they do not need to be on the disk media to function, as the utility programs do. Each of the built-in commands is described in its own chapter, contained within Chapters 3 through 20. The following is a summary of each built-in or resident command.

DIR (Chapter 3): The DIR command is probably the most frequently used of all the commands provided by CP/M. DIR is an abbreviation for "directory" and is used to display or print the directory of all the files contained on a specified disk. See page 67.

REN (Chapter 4): The REN command stands for "rename" and is used for renaming specified files stored on a given disk. See page 71.

ERA (Chapter 5): The ERA command is short for "erase." It is used to erase or delete specified files contained on a given disk. See page 75.

TYPE (Chapter 6): The TYPE command is used to display or print the contents of a specified text file on the screen or paper of the console device. See page 81.

USER (Chapter 7): The USER command provides a means by which different users of CP/M can have access to their own separate files contained on the same disk. CP/M can have up to 15 users and each will not interfere with another's area. The USER command is only available in CP/M Version 2.0 and newer. See page 85.

SAVE (Chapter 16): The SAVE command is used to take the contents of a specified number of pages from the system's RAM memory and save (store) it in a specified file on a given disk. This is usually used only by programmers or system modifiers. See page 267.

TRANSIENT COMMANDS: Transient commands (utility programs) differ from built-in or direct commands, because they are temporarily stored (i.e., transient) in the user area of memory when they are running. Therefore,

it is necessary that transient commands be contained in files stored on the disk media. The standard CP/M transient commands are written in 8080 assembly language and, therefore, contain 8080 machine code. Consequently, they all have an extension or file type of ".COM" meaning a "command" type file. The ".COM" extension indicates that this is a directly executable file, i.e., you can type just the filename and the command will immediately begin running. In other words, when a transient command is used, only the filename need be typed, as CP/M will automatically look for the extension ".COM". A transient command file must be stored on a diskette in one of the drives for it to be accessible. For example, to execute the transient command "STAT", the file "STAT.COM" must be stored on a diskette inserted in one of the system's drives.

Only those commands which are ALWAYS provided with a new CP/M system disk are covered in the transient command chapters. Note that there are other nonstandard utility programs (not distributed by Digital Research) that may be provided on your CP/M system disk. Most of these programs have been added by the company that sold you the CP/M (not Digital Research) in the first place. In general, the types of these extra nonstandard utilities, and the way in which they operate vary depending on the manufacturer or distributor from whom you originally obtained the CP/M operating system. Some popular nonstandard transient program commands are described in Appendix A.

In each transient command chapter, we will be covering all the variations in which a utility progam can be used. The order in which the commands are presented in the chapters reflects what is probably the most common sequence in which a CP/M newcomer would use them. For example, the chapters describing the transient commands "STAT" and "PIP" are the first described among the transients, since these commands will enable you to manipulate files on the storage media. The "FORMAT" and "SYSGEN" commands are among the last presented, since even though they may be the first commands used, they are not used as frequently as the other commands.

Next is a summary of the standard CP/M transient commands. These commands are provided in all versions of CP/M (1.3, 1.4, 2.0, and 2.2), unless otherwise noted.

STAT (Chapter 8): The STAT command, stands for **Stat**us or **Stat**istics. It provides the CP/M user with various ways to determine the status of and assign characteristics to files, disk storage media, and devices and to perform general file "housekeeping" procedures. See page 91.

PIP (Chapter 9): An abbreviation for **P**eripheral **I**nterchange **P**rogram. The PIP command is used for the copying of files between the same or different devices, such as disk to disk, disk to printer, console to disk, disk to modem, etc. See page 117.

ED (Chapter 10): The ED command, short for Text **Ed**itor, provides the CP/M user with a text editor with which text files can be created and edited. See page 157.

SUBMIT (Chapter 11): The SUBMIT command allows user-chosen CP/M commands to be batched into a single file so that they may be automatically executed in a specific sequence. See page 203.

XSUB (Chapter 12) (Only in CP/M Ver. 2.0 and newer): XSUB (EXtended **SUB**MIT utility), is a subset utility of the SUBMIT command. It is used to allow the submitting of commands in response to prompts given by a program already being submitted by the SUBMIT command. See page 213.

ASM (Chapter 13): The ASM command, short for **Assem**bler. It "assembles" (converts) assembly language written with ED or similar text editor into what are called .HEX files, which may in turn be LOADed and converted into pure machine or object code programs. ASM is used by Assembly Language programmers who want to create their own 8080 .COM programs. See page 219.

LOAD (Chapter 14): The LOAD command is used to create a machine or object code file from a hexadecimal (.HEX) file. See page 237.

DDT (Chapter 15): DDT stands for **D**ynamic **D**ebugging **T**ool. It is a utility which offers the CP/M user a method for detecting and removing bugs in machine/object code or hexadecimal files, examining the BIOS and BDOS, etc. See page 241.

DUMP (Chapter 17): The DUMP command is used to display the contents of a machine or object code *file* on the screen or printer in hexadecimal format. See page 273.

FORMAT (Chapter 18): The FORMAT command is provided by many distributors of the CP/M operating system for the purpose of setting up

the disk for accepting information. The formatting program makes the diskette able to accept standard CP/M compatible programs. A formatting command may also be included if your system contains a hard disk, such as a Winchester disk drive or removable hard disk. The FORMAT command is nonstandard and may differ slightly in operation, depending on the distributor of your CP/M operating system. While on different computers, FORMAT's purpose and operation details remain about the same, while details of the actual physical organization of the tracks and sectors vary between systems. Thus most formats are computer hardware dependent. See page 277.

SYSGEN (Chapter 19): The SYSGEN command, short for **Sys**tem **Gen**eration, provides the means by which the CP/M user can generate the CP/M operating system image or copy it from one disk to another. See page 285.

MOVCPM (Chapter 20): MOVCPM means **Move CP/M**. It is used to configure the size of the CP/M operating system so that full use can be made of the computer's available memory, or so memory can be set aside "above" CP/M for holding special user programs. See page 295.

SPECIAL CONTROL-CHARACTER COMMANDS: There are several special commands provided by CP/M, called "Control-Character" commands. These commands provide a way for the user to control and modify the way in which command strings are entered, and the way CP/M responds to them. They are called Control-Character commands, because they are executed by simultaneously pressing the CONTROL key on the terminal along with a specified character. These commands are described in full in Chapter 2.

The next chapter provides an overview of the CP/M operating system. It contains the necessary information to understand the basic characteristics of CP/M which will help the use of commands described in subsequent chapters.

CHAPTER 2

Using CP/M

GENERAL

Every person who first uses CP/M must be familiar with the techniques for getting the system started, and for entering information into CP/M from the terminal keyboard. They must also be aware of the rules for inserting and removing the disk media in the drives. This chapter covers the operation of CP/M from the point of view of the user first turning on the system, inserting a CP/M system disk in the main disk drive, booting or loading CP/M, and using the console device to type commands.

The more complex use of CP/M involves the modifications to the actual CP/M program. This is the case when microcomputers are purchased and first set up. When such a computer is first set up, CP/M is usually modified to allow it to work with the specialized peripherals connected to the computer. Usually this is done by the distributor of your CP/M or your computer, or the manufacturer of your computer before the system is used by any operators. However, in cases where this installation hasn't been done, changes in the CP/M's internal input output driver section (its BIOS) must be made. The method in which these modifications are made is beyond the scope of this book, since they involve interesting, but detailed, technical descriptions on how the BIOS, BDOS and the other areas of CP/M are structured. This information, however, can be found in the book *Soul of CP/M®* by Mitchell Waite and Robert Lafore, Howard W. Sams & Co., Inc.

When the CP/M operating system is first installed on a microcomputer, the user needs to learn how to initialize the system and verify that the operating system will perform as intended. This chapter assumes that you have a CP/M system disk already modified (installed) for the particular hardware that makes up your microcomputer. If this *is* the case, then the following text will help you "bring up" the CP/M operating system on your computer so that you can immediately sit down and use the commands provided by CP/M.

As we mentioned earlier, if your CP/M system disk has not been modified for the peripherals in your system, then you should contact the manufacturer or distributor for advice on how the modification can be made. If it is your intention to modify CP/M yourself, then the user manuals accompanying the CP/M system disk will provide some but not all the information about how this can be done. There are also other books and journal articles on the subject of CP/M which provide information on certain techniques for CP/M system alteration. (Refer to Appendix D for an annotated bibliography.)

BOOTING THE CP/M SYSTEM

To make CP/M operational in your microcomputer, you must first "boot" or "load" the operating system from the CP/M system disk so that it is placed in the computer's Random Access Memory (RAM). The RAM memory is used for "scratch pad" purposes, so that it can be stored and refreshed over and over again with different programs. As long as power (electricity) for the microcomputer is on, the information contained in the RAM memory will remain intact. Once power is removed or the system reset switch is pressed, the information in the memory will be eliminated. For this reason, each time the power is removed and reapplied again, or the computer system is reset, the computer's operating system program, CP/M, is lost. Therefore, at times, the CP/M operating system must be loaded back into memory before work can begin. The function of bringing CP/M back to life is called "booting" CP/M. It causes the CP/M program that is stored on the disk media to be copied from there to the system's RAM. Booting is necessary because without the operating system information in memory, the central processor of the computer will not have anything with which to work.

THE BOOTSTRAP LOADER: All microcomputers capable of operating with CP/M have a "Bootstrap Loader." The Bootstrap Loader is a program stored in Programmable Read-Only Memory (PROM), (located on the central processing hardware of the computer) that will not be erased when the

power is removed. This special program is used to instruct the central processor of the procedure required for *booting* CP/M (or other operating system) from the system disk and *loading* the data into the RAM memory. Since the complete CP/M operating system (also known as the system "image") is stored on the first two tracks (Tracks 0 and 1) of the system disk, the Bootstrap Loader instructs the central processor to read only the first two tracks of the disk. When you reset or first turn on the computer, you can hear the results of the bootstrap loader as CP/M is copied into the system's memory.

THE MONITOR PROGRAM: The method by which the Bootstrap Loader causes CP/M to be booted varies among different types of microcomputers. Most systems simply boot CP/M when first turned on or reset. Some systems contain a "Monitor Program" or "Monitor Control Routine" (also stored in PROM) which is programmed to respond to specific commands typed on the console device. One of these commands may be defined for the purpose of booting the operating system off of a disk. Other commands may access memory locations, change default conditions in the computer, and so on. Some will display a prompt or a message as soon as power is applied to the system, while others will display the message only after a key, usually the space bar, is pressed on the console device. A typical monitor program message is shown in the following example:

```
CPU MONITOR PROGRAM
4800 BAUD SERIAL
?
```

BOOTING THE CP/M OPERATING SYSTEM (COLD BOOT): In the preceding example, the "?" character is the monitor program's prompt. It indicates that the program is ready to receive a command. The specific character used may vary among monitor control programs. In the system shown in the example, the command for booting the operating system is the character "B" (for "Boot"). (Since "B" is not defined by every monitor program as the boot command, you should refer to the Operation Manual accompanying your system if this is the boot approach.) To boot the CP/M operating system, first insert the system disk in the disk drive defined as the "Booting Drive" (usually drive "0" or "A"). As soon as the disk is properly inserted in the disk drive, the boot command is typed. In some systems, a typed command is not necessary since pressing the system reset button is sufficient to cause CP/M to boot. The initial booting of CP/M is called a "Cold Boot." It

refers to booting the operating system for the first time after the system has been powered up (and thus is still "cold.") If CP/M has been successfully booted, a message similar to the following will appear on the console device:

```
XXk CP/M vers Y.Y

A>
```

THE CP/M SIGN-ON MESSAGE: The CP/M message just shown is typical for most CP/M versions. The "XXk" is a value that represents the amount of RAM memory (in kilobytes) with which CP/M has been installed, or configured, for use in your system. In CP/M Version 1.4, for example, this amount "XX" may range from a minimum of 16K, up to 64K. In Version 2.0 and higher, the memory may range from 20K to 64K, or more. The exact number depends on the amount of RAM memory contained in your system, and the amount specified when CP/M was initialized by the manufacturer/distributor for correct operation in your system. See page 37.

The "vers Y.Y" portion of the message indicates the revision level of the CP/M, and may be 1.3, 1.4, 2.0, 2.2, etc. Depending on the version and the distributor from whom it was obtained, other information may also be contained in the message, such as the manufacturer's name and address, date of installation, peripheral equipment, and so on. This additional information may refer to the revision of the BIOS, and any modifications made for correct operation with specific hardware contained in your system, such as disk drive controllers.

The final item on the screen is the CP/M "prompt". This prompt is always located where the console device screen cursor or print head is initially positioned. Its location indicates the starting point where you may enter a CP/M command. The prompt is first represented by "A>" indicating that you are now using disk drive "A" (the "boot drive"), and that CP/M is waiting for you to type in a command. This prompt may be changed to represent other disk drives in your system. We will discuss this later.

It is important to remember that the disk in Drive A should always contain the CP/M operating system image on the first two tracks. (Normally, all disks used with the system will contain it.) If, after the system is up and running, a disk *without* the CP/M operating system in the system tracks is inserted in Drive A, the system will fail the first time a "warm" boot (after it is no longer "cold") is required. Therefore, it's always a good practice to copy, or back up, the operating system using the SYSGEN command

(described in Chapter 19) onto any newly initialized disk intended for use with CP/M.

BACKING UP THE CP/M SYSTEM DISK AND OTHER DISKS: When you first start using CP/M immediately after the CP/M operating system has been booted or loaded in the system, you must create a complete backup copy of the system disk. This is a standard procedure that is performed as soon as possible as a precautionary measure. It is essential because the CP/M system disk is the only *permanent* storage place for the CP/M operating system, its commands and application programs. By doing this, you will always have at least one (preferably several) usable copies of CP/M and its programs, should the main CP/M system disk be damaged or destroyed. (Actually, it is an excellent idea to make backup copies of *ALL* of your disks, not just the CP/M boot disks.)

The backup procedure can be accomplished as soon as CP/M has been successfully booted and is operating correctly. The following steps should be taken each time that you want to back up a disk, whether you are backing up the main CP/M system disk or any other CP/M-compatible disk.

1. Use the disk formatting program provided with your CP/M package to format a new disk. Normally, this program will be called "FORMAT", although its name may vary. For information on the use of this command, refer to the "FORMAT" description in Chapter 18, page 277.
2. After the new disk has been formatted, it is now ready to receive the information you want copied. If you want the new disk to be a bootable disk, then use the SYSGEN command, which will copy the CP/M system program on the first two tracks of the disk. Refer to the SYSGEN command description in Chapter 19. If the disk is not to be a bootable disk, then skip this step.
3. Once the disk has been formatted, and optionally conditioned to be a bootable disk, all the data and programs you want backed up can now be copied. Normally, this is accomplished with the PIP program. PIP allows you to copy individual files, groups of files, or all files from the original disk to the backup disk. To use this command, refer to the PIP description in Chapter 9. Some CP/M packages contain a special copying program (sometimes called "COPY"). This program will not only copy all the files on a disk, but will also optionally copy the system image, thus eliminating the need for the SYSGEN command (Step 2).

It is a good idea to get into the habit of backing up a disk every time you use it and change the data stored on it. You will be very grateful that you did

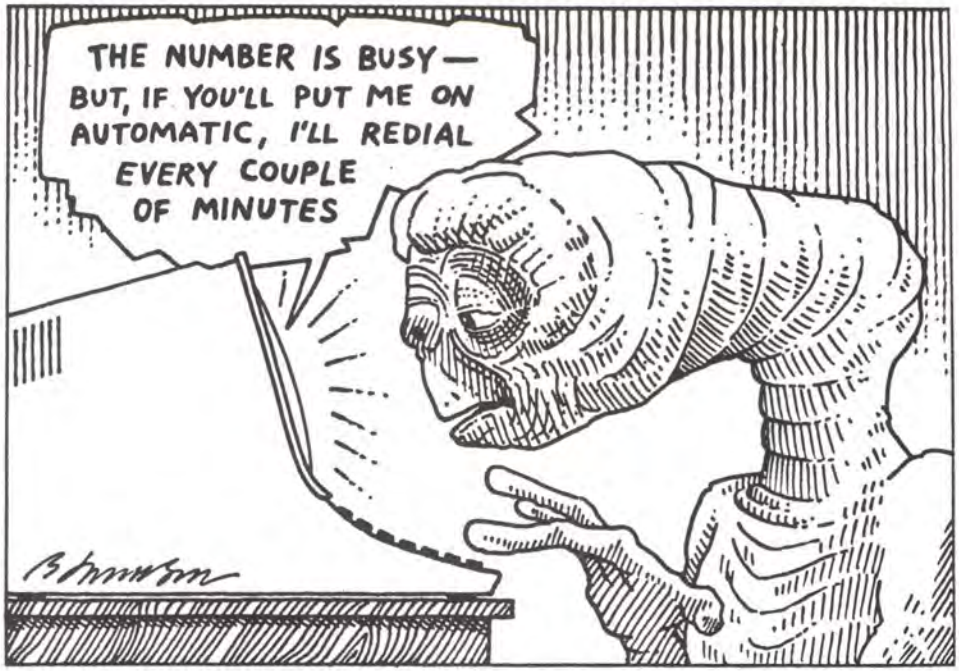
when a very important disk is rendered unusable, and the backup disk is there for you. We can't stress this enough, because the unthinkable happens to everyone now and then. Backup disks will save many hours work, not to mention the safeguarding of valuable software.

MEMORY MANAGEMENT: Once CP/M has been booted, only a small portion of RAM will be occupied with the operating system. The remainder of RAM may freely be used for application programs. The method by which the remaining memory is used is controlled by CP/M so that all of it or portions of it can be reused each time a new applications program is specified.

WARM BOOT: Now that first-time booting of CP/M (Cold Boot) has been explained, we will describe another type of booting, or loading, called "Warm Boot." A warm boot is used upon exiting from a program or utility that has been running on the system. This causes CP/M to re-initialize itself by reading in only a portion of the operating system off the system disk, leaving everything else intact. This procedure is automatically performed by CP/M, and does not require the user actually to issue the warm boot command. There are, however, other circumstances (see the following) that do require the user to input the warm boot command. When it is necessary to do this, the user may issue a warm boot by simultaneously pressing the CTRL (Control) and C keys (CTRL/C) on the console device.

A CTRL/C warm boot command issued by the user is required in circumstances when CP/M cannot initialize itself properly due to changing a disk in the boot disk drive, or to certain types of errors or malfunctions. These errors are usually accompanied by error messages so that the user knows that a warm boot is needed. Other types of errors that require a warm boot may not be accompanied by messages. In these cases, the only indication of an error that the user will have is that the system is inoperable.

If a warm boot fails to re-initialize CP/M, then a cold boot (reset) must be issued to reload the complete CP/M operating system. When this happens, you should try to determine the cause of the error or malfunction. The error could be caused by malfunctioning hardware, or due to a fault in the way the CP/M operating system or other programs are stored on the disk. If the error is due to the disk media (such as a damaged surface or a soft error), then the backup disk should be used as a substitute. Such an error may also be caused by trying to run a program that contains internal errors or one which is not compatible with CP/M.



USING DISK DRIVES

After the CP/M operating system has been successfully booted, the "logged" drive automatically becomes Drive A. The logged drive, or "default drive," is the drive which the computer assumes you are referring to in all subsequent commands (unless you specify otherwise).

You can tell which is currently the logged drive by the prompt on the console device. For example, "A>" indicates that Drive A is the logged drive. Other drives in the system can also be accessed through CP/M by specifying their names. Version 1.3 of CP/M, for example, will recognize a maximum of two drives (A and B). Version 1.4, on the other hand, will recognize a maximum of four (A, B, C, and D). Finally, Version 2.0 and higher support a maximum of sixteen drives (A through P). Making another drive the logged drive is accomplished by typing the name of the new drive, followed by a colon (:) and a press of the RETURN key. The following example shows the changing of Drive A to Drive B as the logged drive.

```
A>B:
B>
```

As shown in the preceding example, the name of the new logged drive must be followed by the ":" character so that CP/M can recognize it as a logical drive name. Omit the colon, and CP/M would consider "B" to indicate a command. For example, if you had a command file called "B.COM" stored on the disk inserted in Drive A, then the command would execute. If the file did not exist on the disk, then a short error message would appear as follows:

```
A>B
B?

A>
```

In either case, the computer would not give you the desired results. Note that the system leaves the current drive as the logged-in drive. It is therefore important to remember that the ":" character must always be specified when the intention is to change logged disk drives, (and when a disk drive is referred to in a CP/M command). As another example, to change the logged in drive from the A drive to the M drive, you would type M: at the A> prompt.

Before you indicate a new logged drive, you must first insert a diskette (hereafter called disk for short) in that drive. If a reference is made to Drive B and Drive B does not contain a disk, an error message similar to the one shown below will be presented. (The precise error message will depend on what has been defined by the distributor of your CP/M package. Some CP/M configurations will not display any error message.)

```
A>B:

DRIVE NOT READY
```

When this type of error condition occurs, the console device keyboard will be locked and a warm boot (CTRL/C) will *not* be adequate to re-initialize the system. In most systems, the situation can be remedied by simply inserting a disk in the drive (Drive B: here) and shutting the drive door (if one is equipped), whereupon CP/M automatically re-initializes itself and the "B>" prompt appears. If this doesn't work, then a cold boot (system reset) should be performed, whereupon the disk should be inserted in Drive B: *before* the change in logged drives is made.

If, while attempting to change the logged drive, a disk with an incorrect density or a format incompatibility (more on this soon) is inserted in a drive other than A, an error message will appear describing the problem. (The

actual message will vary according to the CP/M version and type of problem.) When this happens, a warm boot (CTRL/C) can be issued. If the warm boot does not rectify the problem, then the disk should be removed from the disk drive and a cold boot (system reset) should be performed. If it is determined that the type of disk is compatible with CP/M but has an incorrect format, then the disk should be initialized with the formatting program provided with the system disk before it is used again. Note that most formatting programs used for this purpose will *erase* the entire disk.

It is not always necessary to change logged disk drives when you wish to access a file that is stored on another drive. For example, a file on Drive B can be accessed while Drive A is still the logged drive. This can be done by preceding the accessing command with the drive name. (See the following for further description of this process.)

ACCESSING COMMANDS AND FILES

There are two basic methods in which commands are stored on a CP/M disk. According to the way they are stored and used, they fall into two categories:

1. Built-In or Direct Commands (also called *resident* commands), and
2. Transient Commands.

BUILT-IN COMMANDS: Built-in commands are those that are permanently stored in the memory as part of the CP/M operating system. They can be accessed and used, regardless of the drive you are using, by simply typing their name. Built-in commands are also called "Direct Commands" as they are not stored in individual files, but are automatically booted into the computer along with the CP/M operating system. The built-in commands provided by CP/M are

DIR (Directory)
REN (Rename)
ERA (Erase)
TYPE
USER (CP/M Ver. 2.0 or above only)
SAVE

The following example shows how the DIR command is accessed so that a display of the directory on disk drive A: can be made:

```
A>DIR

A: STAT    COM : PIP    COM : SYSGEN  COM : MOVCPM  COM
A: FORMAT  COM : LOAD   COM : TEXT1   COM : DDT     COM

A>
```

Note all that was done was the letters DIR were typed followed by pressing the return key. Optionally DIR can be entered followed by a space and a "B:". This indicates that the directory of the disk in Drive B is to be displayed, while Drive A remains the logged drive. When using built-in commands, it doesn't matter which drive is the logged drive, since these commands are contained in the main operating system and thus stored in RAM. We could also type DIR C: but, if there is no C drive on the computer system, an error message will occur.

TRANSIENT COMMANDS: Transient commands are those commands contained in files stored on disks, and have file names that end in ".COM". Transient commands are actually small utility programs. Transient refers to the fact that these commands are actually files on the disk that can be loaded into memory and then removed (erased), thereby giving them a temporary nature. The ".COM" portion of the name is the "file type" and means "command." Transient commands are accessed from the disks on which they are stored by simply entering the file name *without* the ".COM" filetype. Like the above described built-in commands, CP/M will figure out if the command is in the directory and execute it if it is. The standard transient command files (programs) provided with the CP/M operating system are:

STAT.COM (Status or Statistics)
PIP.COM (Peripheral Interchange Program)
ED.COM (Text Editor)
SUBMIT.COM
XSUB.COM (Extended Submit (CP/M Ver. 2.0 or above))
ASM.COM (Assembler)
LOAD.COM
DDT.COM (Dynamic Debugging Tool)
DUMP.COM
SYSGEN.COM (System Generation)
MOVCPM.COM (Move CP/M)

There may be many other transient command files that CP/M offers that are not a part of the set just shown.

Descriptions of each of the standard transient command files are provided in their individual chapters following this chapter. The following example shows how the transient command STAT is accessed:

```
A>STAT

A: R/W, Space: 140k

A>
```

As can be seen, only the file name portion of .COM command files is typed. If ".COM" is typed in, and thus included as part of the command, the following error message will appear:

```
A>STAT.COM

STAT.COM?

A>
```

Whenever a response such as this appears, it means that CP/M could not find the command you specified. The command may be nonexistent or incorrectly identified.

ACCESSING FILES: To verify that a transient command file or any other file is stored on a disk, the DIR command can be used to display the entire directory of that disk. The DIR command can also be used to see if a specific file is on the disk. This is accomplished by entering DIR followed by the "filespec" (file specification). The word filespec will be used henceforth in this book to represent the necessary sequence of characters representing a file and optionally the disk drive name on which the file is stored. The structure of a filespec is as follows:

filespec = x:filename.typ

A filespec always contains the file name (shown as "filename") and the file's extension or file type (shown as "typ"). A file name may be one through up

to a maximum of eight characters in length. A file type may consist of up to a maximum of three characters. Almost all files contain file types. The file type is normally used to identify the type of file, although for some files their use is optional. When the file type is required as part of the filespec, a period (".") is always used as separator between the file name and the file type. A separating period is not required when the file type is not used. A list of commonly used file types is provided later in this chapter. A filespec may also optionally contain a disk drive name (shown as "x:") to define the location at which "filename.typ" is stored. If the filespec does not contain a disk drive name, CP/M assumes that the file specified is on the logged drive. When a disk drive name is included in the filespec, a colon (":") must be placed as a separator between the drive name character and the file name. Both transient command files and files accessed by built-in and transient commands are always referred to as filespecs. The following shows some examples of valid filespecs:

x:filename.typ
FILE-1
STAT.COM
A:LETTER1
B:PROGRAM.BAS

The next example shows how to see if the transient command program filespec "STAT.COM" is stored on the disk in the logged disk drive.

```
A>DIR STAT.COM
```

The example following illustrates how to see if STAT.COM is stored on the disk in Drive B. while keeping Drive A as the logged drive. Note that this time, the filespec includes the disk drive name.

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

If the file is stored on the disk in the specified drive, the following will be displayed:

```
A: STAT    COM
```

or

```
B: STAT   COM
```

. . . depending on which drive (A or B) was specified. If the file is not stored on the disk in the specified drive, the following message will be displayed:

```
NO SUCH FILE
```

Once it has been determined that STAT.COM is stored on the disk in the specified drive, the transient command can be used by simply typing in "STAT". This will work since CP/M will automatically look for the file type ".COM" when the file type is absent from the command. Of course, it may be just as useful to type STAT and if it's not on the disk you will know it when the NO SUCH FILE message appears on the screen.

If you are logged in on drive B, and wish to use STAT which is on drive A, it is not necessary to switch the logged status from Drive B to Drive A to use it. Instead, the following command can be typed.

```
B>A:STAT

A: R/W, Space: 140k
B: R/W, Space: 358k

B>
```

The filespec may contain various combinations of disk drive names and command file names specified to attain the desired result. The next example shows that the logged working drive is B, the transient command ED.COM we wish to use is on Drive A, and the text file to be modified is on drive B.

```
B>A:ED B:TEXT1.TXT

*
*Q

B>
```

The ability to use programs and access files on disk media in drives we are not logged into, gives the CP/M system important flexibility. For example, if we are logged on drive A and it becomes "full" and can't hold any more files, we could place extra files on another drive. Or, if we need to run a program not on the logged-in drive, we could use DIR to examine other drives for the program and run it without changing to the disk it was on.

USING WILDCARDS

It is not always necessary to type in the full file name within the filespec, when the filespec follows a command. This degree of leniency comes in handy when the full name of the file is unknown, or when you wish to access a group of files that have similar names. Groups of characters in a file name or a file type can be substituted with "wildcards" to enable CP/M to find all files with names that approximate what you are looking for. There are two types of wildcards defined in the CP/M operating system:

* (used to represent a group of characters)
? (used to represent a single character)

Wildcards can be used with the following standard CP/M commands:

Built-In Commands: DIR and ERA
Transient Commands: STAT.COM and PIP.COM

All other commands require that the associated filespecs contain specific file names. If wildcards are specified, an error will occur.

USING THE "*" STAR WILDCARD



The following example demonstrates the use of the "*" wildcard (pronounced either "asterisk" or "star"). In the above command, a specific file name is included in the filespec, but a wildcard is substituted for a specific

file type. Thus, all files with the same file name are displayed, regardless of the file type. For example:

```
A>DIR TEST.*

A: TEST    DOC : TEST    BAS : TEST    INT : TEST    TXT
```

Here all the files with the file type TEST are displayed.



Conversely, the next example demonstrates the substitution of the file name with the "*" wildcard, while displaying a specific extension or file type. In this case, all files with the same extension are displayed, regardless of file name.

```
A>DIR *.COM

A: SYSGEN  COM : MOVCPM  COM : STAT  COM : PIP     COM
A: SUBMIT  COM : XSUB    COM : ED    COM : ASM     COM
A: DDT     COM : DUMP    COM : LOAD  COM : FORMAT  COM
```



The "*" wildcard can also be used like a "fill in with anything" symbol along with the file name, so as to represent a single character or a group of characters within a filespec. In the next example, all files with names *beginning* with "TEXT" and with the file type ".TXT" are displayed:

```
A>DIR TEXT*.TXT

A: TEXT10  TXT : TEXT50  TXT : TEXT100  TXT : TEXT200  TXT

A>
```

USING THE "?" WILDCARD



The "?" (question mark) wildcard is used in a similar manner as "*" with one important difference. The "?" wildcard can be employed to substitute characters in specific positions in the filespec. In other words, if you are uncertain about the first and last characters in a file type, but are sure that the middle character is an "O", then the "?" wildcard will be extremely valuable in locating the file you want.

The following example demonstrates the use of the "?" character as a variable when using the DIR command. The example shows "DIR *.?O?," which translates to: "Display all file names with extensions that contain the middle letter of 'O'." By entering that command, here's what will be displayed on the console device:

```
A>DIR *.?O?

A: SYSGEN  COM : MOVCPM  COM : DISKINFO  DOC : STAT    COM
A: COS     FOR : FLOW    DOC : PIP       COM : SAMPLE  DOC
A: SUBMIT  COM : DISPLAY FOR : XSUB      COM : ED      COM
A: ASM     COM : ACCOUNT FOR : DDT       COM : LOAD    COM

A>
```

Any combination of specified characters and "?" wildcards may be used in this manner. Likewise, any combination of specific characters and wildcards can be used in a command string, as long as the first character in the file name field is not a "?" wildcard. There can only be a maximum of eight question marks in a filename and three in a file type.

CONTROL CHARACTER COMMANDS

The CP/M operating system provides several special "control character" commands which are used for various functions, mostly editing. Control

character commands are different from all the other commands in the operating system in that they use the control key on the console device. (The control key may be called CTRL, CNTL, CNTRL, or CONTROL, depending on the type of terminal. This book uses CTRL throughout, as it is the name most commonly used for the key.) This key is pressed simultaneously with one of several other keys defined for the actual function. Since both keys are pressed simultaneously, a control character command is expressed, for example, as CTRL/X. When control characters are displayed on the screen or printed on the paper of a terminal, it will usually be displayed as "^X" where the "^" character represents CTRL.

WHERE'S THAT CONTROL KEY–USING YOUR PINKY

The CONTROL key is usually located on the far left of the console keyboard. One good way to use control character commands is place the little finger of your left hand on the CTRL key, and then use the left index finger to press those keys on the left side of the keyboard, and your right index finger for those keys on the right side.

The following paragraphs describe the functions of each of the control character commands.

WARM BOOT COMMAND: The command to issue a Warm-Boot, and restart CP/M, as described earlier in this chapter, is CTRL/C.

LINE TERMINATION COMMANDS: These consist of two commands or functions that are used to terminate a command or text line in lieu of the normal return key. They are as follows:

1. **CTRL/M:** This command will cause a carriage return. It is equivalent to pressing the RETURN key.

2. **CTRL/J:** This command will cause a line feed. It is equivalent to pressing the LINE FEED key. This command is only available in CP/M Version 2.0 and above.

CHARACTER DELETION COMMANDS: These two commands are used to delete individual characters in a command line. They are as follows:

1. **CTRL/H and BACKSPACE:** The CTRL/H command will cause the cursor to move one space to the left and thus erase the character it

encounters there. It is equivalent to pressing the BACKSPACE key. The command is only available in CP/M Version 2.0 and above.

2. **DELETE or RUBOUT:** By pressing this key, the character to the left of the cursor will be deleted, causing the character to be echoed on the screen. The key is called DEL, DELETE, or RUBOUT, depending on the terminal. This command is sometimes used instead of CTRL/H or BACKSPACE when the console device consists of a teletypewriter or teleprinter and it is desired to print the characters as they are deleted. In some CP/M configurations, this key command is modified to function the same as CTRL/H or BACKSPACE when a video terminal is used as the console device.

LINE DELETION COMMANDS: These two commands are used to cancel a command line that has not yet been terminated by the RETURN key. They are as follows:

1. **CTRL/U:** This causes a command line that has already been typed to be cancelled. Upon cancellation, the cursor moves to the beginning of the next line. This command is available in all versions of CP/M. Version 2.0 and above causes the "#" character to be displayed on the next line. This character is used to end lines frequently when two individuals are communicating over a modem link. By pressing ^U we can cancel the effect of the typed information so it won't be accepted by CP/M, but still retain a picture of it on the screen.

2. **CTRL/X:** This command performs the same function as CTRL/U in CP/M Version 1.4 or above. In Version 2.0 or above, the cursor will backspace to the beginning of the command line and erase the text.

DISPLAY CONTROL COMMANDS: These consist of three commands that are used to control the way in which text is displayed on the console device. They are

1. **CTRL/E:** This command is used to cause a command line to overflow onto the next line before the user has pressed the RETURN key. When the RETURN key is finally pressed, all lines caused by CTRL/E will be considered by CP/M as being a single command line. This command is only available in CP/M Version 1.4 and above.

2. **CTRL/R:** This command is used to reshape a command line that has echoed characters, deleted by the DEL, DELETE, or RUBOUT key. The command will eliminate all echoed characters and display the command in a more legible form on the following line. This command is only available in CP/M Version 1.4 and above. Version 2.0 places the "#" character on the next line preceding the retyped command line.

3. **CTRL/S:** This command is used to pause the scrolling of text on the console device. When the command is first issued, the scrolling stops. When it is issued a second time, the scrolling resumes. This procedure can be repeated as many times as necessary. This command is available in all versions of CP/M.

PRINTER CONTROL COMMAND: The printer control command is used to echo any text that is displayed on the console device. It enables text to be simultaneously displayed and sent to the printing device. The command is CTRL/P. When the command is issued the first time, echoing to the printing device commences. When it is issued a second time, the echoing stops. This can be repeated as many times as necessary. This command is available in all versions of CP/M.

STANDARD FILE TYPES

The following is a list of file types or extensions that define types of files. Some of the types are required by various CP/M commands or programs because a search will automatically be made for the right file type before the command executes. This will occur, in many cases, regardless of whether the file type was typed on the console device. Other CP/M commands or programs will simply verify that a particular specified file has the right file type. There are other file types that are not required by CP/M or CP/M commands, but their usage is recommended only for consistency so that files can be recognized by their type.

EXTENSION	TYPE	EXAMPLE
.ASM	Required for assembly language source code (text) files when used with the ASM command.	DUMP.ASM PRINTER.ASM
.BAK	Indicates a backup file. It is automatically created by ED and other text editors and word processors immediately after opening a text file.	TEXT.BAK LETTER1.BAK

EXTENSION	TYPE	EXAMPLE
.BAS	Used to represent a BASIC program source (text) file. It is optional for some types of BASIC languages, and required for others.	ACCOUNT.BAS GAME1.BAS
.CRF	Indicates a relocatable assembler crossreference file.	COMP.CRF MAP.CRF
.COB	Indicates a COBOL program source (text) file.	ACCOUNT.COB SEC.COB
.COM	Indicates a CP/M transient program command file. This extension is required by CP/M for such files so that the file can be recognized as a command. This enables the user to simply type the name of file without the file type.	STAT.COM PIP.COM
.DAT	Indicates that the contents of the file consists of data of some kind, normally ASCII data. This is a default extension for some FORTRAN languages.	PROG1.DAT TEXT.DAT
.DOC	Optional extension to indicate a document or text file.	INFO.DOC HELP.DOC
.FOR	Indicates a FORTRAN program source (text) file. It is required by some types of FORTRAN languages, and is optional for others.	DISPLAY.FOR MATH1.FOR
.HEX	Indicates a Hexadecimal file created by the ASM command. It is required by the LOAD command when loading the file to create a .COM file.	DUMP.HEX PRINTER.HEX
.INT	Indicates an intermediate program file consisting of object code, created by a BASIC compiler, and is required for most types of BASIC languages with compilers.	ACCOUNT.INT GAME1.INT
.LIB	Indicates a library source file normally containing text. It may be used for many things, but when used with ED for locating, substituting, and appending text in another text file, it must have the .LIB extention.	BUF1.LIB TEXT1.LIB
.LST	Indicates a list file created by some compiler BASIC languages, source program text with error messages to facilitate debugging.	SPACE.LST ERRS.LST
.MAC	Indicates a relocatable assembler source code file.	BIOS.MAC PRINTER.MAC
.MSG	Normally used the same way as .DOC.	ERR.MSG READ-ME.MSG
.OVR	Indicates an overlay file. It is used by other programs that require storage of data for overlays.	MSG1.OVR OVLY1.OVR

EXTENSION	TYPE	EXAMPLE
.PAS	Indicates a Pascal program source (text) file, required for use with some Pascal language compilers.	LEDGERS.PAS SCREENS.PAS
.PRN	Indicates a printable text file created by some text formatters, language compilers, ASM, and other assemblers and macroassemblers.	LOAD.PRN PRINTER.PRN
.REL	Indicates a relocatable object code file created by relocatable assemblers, FORTRAN, and other programming languages.	CIRCLE.REL SCREEN.REL
.SUB	Indicates a CP/M SUBMIT command file. When using SUBMIT, the file specified must have .SUB.	INIT.SUB LOAD.SUB
.SYM	Indicates a text file containing a symbols table, created by some macro assemblers.	BOOT.SYM BIOS.SYM
.TEX	Indicates a source text file for some text formatters.	MANUAL.TEX CHAPT1.TEX
.TXT	Used the same as .DOC.	LETTER.TXT INFO.TXT
.$$$	Indicates a temporary file created by PIP and ED or other programs.	MANUAL.$$$ INFO.$$$

As you can see, there are many possible file types in the CP/M system. These names are not always clear enough for the user to determine what type of file it is. Furthermore, manufacturers often make up their own file type codes, adding more confusion. For example, one manufacturer will call a file WSMSGS.OVR, which means "wordstar messages" (WSMSGS) overlay (.OVR) file. When in doubt you should consult this table and, if it's not here, consult the manuals that come with the product for the description of the file type.

A COMPARISON OF CP/M VERSIONS

Since 1975, when CP/M was first developed, many changes have been made to improve the operating system. Over the years, these changes have been incorporated in various versions of CP/M. The following tables reflect the changes made from Version 1.3 through the most recent Version 3.0.

BUILT-IN COMMANDS:

COMMAND	1.3	1.4	2.0	2.2	3.0	3.0 BANKED
DIR	YES		NEW: Displays 4 filespecs per line. Displays only "DIR" filespecs. Displays the filespecs for current user only.		NEW: Includes options which replace the function of STAT, plus new options. Displays filespec, size, records, attributes, protection status, and update and access date and time. Accesses DIR.COM when necessary.	
DIRSYS	NO				NEW: Same as DIR, but displays "SYS" (system) files.	
ERA (ERA or ERASE in Ver. 3.0 only)	YES	NEW: Asks ALL? after ERA *.*	NEW: Erases files of current user only.		NEW: "CONFIRM" option.	
REN (REN or RENAME in Ver. 3.0 only)	YES				NEW: Use wildcards; enter REN alone & use program dialog "NEW NAME" & "OLD NAME"; option to delete existing "NEW NAME" file; references RENAME.COM when necessary.	
SAVE	YES		NEW: Does not alter user memory (TPA).		NEW: Invoked before data is loaded into memory via SID; upon exit, prompts for filespec, optionally deletes old filespec, and asks for start and end locations.	
TYPE	YES				NEW: "PAGE" option to pause at each 14th line.	
USER	NO		YES		NEW: Enter USER alone and use program dialog "ENTER USER #:"; displays current user number with disk drive in prompt if greater than 0 (1A>, 5C>, etc.).	

TRANSIENT COMMANDS:

COMMAND	1.3	1.4	2.0	2.2	3.0	3.0 BANKED
ASM	YES				NO (Replaced by MAC and RMAC)	
COPSYS	NO (Similar to SYSGEN)				YES	
DATE	NO					
DDT	YES				NO (Replaced by SID)	
DEVICE	NO				YES	
DUMP	YES				NEW: Displays data in hexadecimal and ASCII formats.	
ED	YES	NEW: +/-V, 0V, nX, 0X, nnnn:, :mmmm, E, H, Q, O. Tabs always echo as spaces.	NEW: +V is default. Cannot alter R/O file. Cannot access a SYS file. n2 added.		NEW: Specify input & outfile; enter ED alone for program dialog of input and output files. Execute a command through line n. nXfilespec^Z, 0X^Z, 0Xfilespec^Z.	
FORMAT	Depends on the CP/M package and the distributor. The formatting program may have another name, such as: INIT, FMT, INITDSK, IN, etc. There is no standard formatting program available.					
GENCPM	NO (Similar to MOVCPM)				YES	
GENCOM	NO					
GET						
HELP	NO (A special version may be available from distributors, optionally purchased, or user created.)					
HEXCOM	NO (Same as LOAD)					
INITDIR	NO					
LIB	OPTIONAL					
LINK						
LOAD	YES				NO (Replaced by HEXCOM)	
MAC	OPTIONAL				YES	

Continued on next page

TRANSIENT COMMANDS–CONT.

COMMAND	1.3	1.4	2.0	2.2	3.0	3.0 BANKED
MOVCPM	YES				NO (Replaced by GENCPM)	
PATCH	NO				YES	
PIP	YES	NEW: Option parameters. Physical Device names. Ambiguous file names.	NEW: Parameters: [Gn], [R], [W].		NEW: Physical devices AXI and AXO. Drive names optional when default drive. [C] option for copy confirmation. May copy to and from User areas.	
PUT	NO				YES	
RMAC	OPTIONAL					
SET	NO				YES	NEW: Allows the assignment of passwords.
SETDEF	NO				YES	
SHOW	NO				YES	NEW: Shows if password is required.
SID	OPTIONAL				YES	
STAT	YES Displays free disk space.	NEW: Displays size of each file; log:=phy:; X:=R/O; Detects switched disks. Displays free space on all active disks.	NEW: VAL: menu. DSK:, X:DSK:, USR:, $S. File atr. R/W, R/O, DIR, SYS.		NO (The functions of STAT are performed by DIR, DIRSYS, SHOW, SET, and DEVICE.)	
SUBMIT	YES		NEW: Operates with XSUB		NEW: The XSUB function is built into SUBMIT. SUBMIT and GET files can be nested in any combination. SUBMIT files can be searched for in chained disk drives.	
SYSGEN	YES				NO (Replaced by COPSYS)	

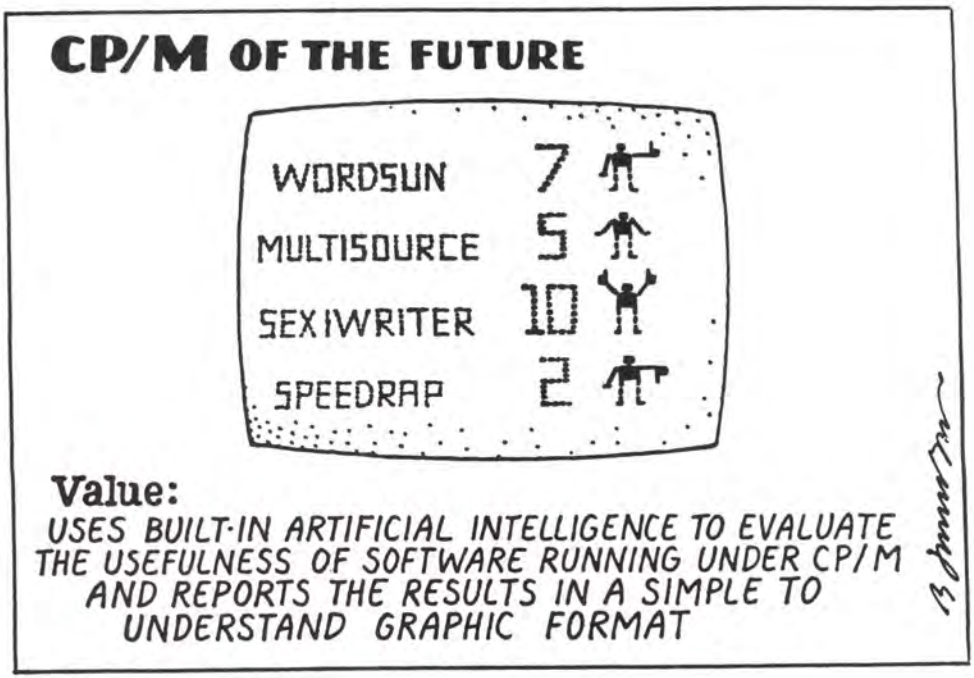
COMMAND	1.3	1.4	2.0	2.2	3.0	3.0 BANKED
XREF	OPTIONAL				YES	
XSUB	NO		YES		NO (Function included within SUBMIT; see GET.)	

CONTROL CHARACTER COMMANDS:

COMMAND	1.3	1.4	2.0	2.2	3.0	3.0 BANKED
CTRL/A	NO					YES
CTRL/B						
CTRL/C	YES					
CTRL/E	NO	YES				
CTRL/F	NO					YES
CTRL/G						
CTRL/H or BACKSPACE	NO		YES			
CTRL/I or TAB						
CTRL/J or LINEFEED						
CTRL/K	NO					YES
CTRL/M or RETURN	YES					
CTRL/P						
CTRL/R	NO	YES	Places a "#" at current cursor position before moving to next line to retype command entered thus far.			
CTRL/S	YES					
CTRL/U	YES		Places a "#" at current cursor position and moves the cursor to the next command line.			
CTRL/W	NO					YES
CTRL/X	NO	Same as CTRL/U	Same as CTRL/U, but backspaces and erases entire line.			
DEL, DELETE, RUBOUT	YES					

DISK SPECIFICATIONS:

SPECIFICATION	1.3	1.4	2.0	2.2	3.0
Maximum num. of drives	2	4	16		
Maximum Storage per Drive	1 Megabyte		16 Megabytes (Expandable)		512 Megabytes
Maximum File Size	128 Kilobytes		8 Megabytes		32 Megabytes
Maximum number of files	64		Expandable according to storage density.		
Access Method	Sequential		Sequential or random		
Location of disk characteristics	BDOS	Disk parameter	BIOS-Resident Disk Parameter Block (DPB)		



NEW OR CHANGED BDOS FUNCTIONS:

FUNCTION	1.3	1.4	2.0	2.2	3.0
0 - System Reset	YES				
1 - Console Input					
2 - Console Output					
3 - Reader Input	YES				NO
Auxiliary Input	NO				YES
4 - Punch Output	YES				NO
Auxiliary Output	NO				YES
5 - List Output	YES				
6 - Direct Console I/O	NO		YES		
7 - Get I/O Byte	YES				NO
Auxiliary Input Status	NO				YES
8 - Set I/O Byte	YES				NO
Auxiliary Output Status	NO				YES
9 - Print String	YES				
10 - Read Console Buffer	YES		Improved line editing		
11 - Get Console Status	YES				
12 - Lift Disk Head					
Return Version Number	NO		YES		
13 - Reset Disk System	YES				
14 - Select Disk					
15 - Open File	YES		Improved		
16 - Close File	YES				
17 - Search for First	YES		Improved		
18 - Search for Next					
19 - Delete file					
20 - Read Sequential	YES				
21 - Write Sequential	YES				
22 - Make file	YES		Improved		
23 - Rename file					
24 - Return Login Vector					

NEW OR CHANGED BDOS FUNCTIONS–CONT.

FUNCTION	1.3	1.4	2.0	2.2	3.0
25 - Return Current Disk	YES				
26 - Set DMA Address					
27 - Get Address (Allocation)					
28 - Write Protect Disk	NO		YES		
29 - Get Address (R/O Vector)					
30 - Set File Attributes					
31 - Get Address (Disk Param.)					
32 - Set/Get User Code					
33 - Read Random					
34 - Write Random					
35 - Compute File Size					
36 - Set Random Record					
37 - Reset Drive	NO				YES
38 and 39	NO				Undefined
40 - Write Random With Zero Fill	NO				YES
41 - Test and Write Record					
42 - Lock Record					
43 - Unlock Record					
44 - Set Multi-sector Cnt					
45 - Set BDOS Error Mode					
46 - Get Disk Free Space					
47 - Chain To Program					
48 - Flush Buffers					
49 - Set/Get System Control Block					
50 - Direct BIOS Calls					
51 through 58	NO				Undefined
59 - Load Overlay or Resident System Extension	NO				YES
60 - Call Resident System Extension					

FUNCTION	1.3	1.4	2.0	2.2	3.0
61 through 97	NO				Undefined
98 - Free Blocks	NO				YES
99 - Truncate File					
100 - Set Directory Label					
101 - Return Directory Label Data					
102 - Read File Date Stamps and Password Mode					
103 - Write File XFCB					
104 - Set Date and Time					
105 - Get Date and Time					
106 - Set Default Password					
107 - Return Serial Number					
108 - Get/Set Program Return Code					
109 - Get/Set Console Mode					
110 - Get/Set Output Delimiter					
111 - Print Block					
112 - List Block					
113 through 151	NO				Undefined
152 - Parse Filename	NO				YES

The Direct Console I/O call was added to Version 2.0 of CP/M, and 1.3 and 1.4 lack it. This call allows the user program to avoid the normal CP/M control character editing and filtering characters (CTRL/S, CTRL/Q, CTRL/X, etc.). See *Soul of CP/M.* Call 10 (Read Console Buffer) improved the editing capability of CP/M for the correcting of user-typed strings. The calls to open files, search for next, delete files, and to make and rename files have been improved. Most importantly, calls to allow the random access of files, (calls 33 through 36; Read Random, Write Random, Computer File Size, and Set Random Record) have been added to Version 2.0. Also, calls to write protect disks, set file attributes, and get user codes have been added. Calls 37 through 152 have been added to CP/M Version 3.0 (some of these calls in the range are undefined and reserved for future use). Most of these calls are used with the new features provided by Version 3.0, such as Load

Overlay or Resident System Extension, Read File Date Stamps, Return Directory Label Data, etc. Some of the calls provide an improvement over the overall operation of the operating system as compared to the previous versions. For an overview of CP/M Version 3.0, please refer to Chapter 21.

This reference information concludes Chapter 2, and the portion of the book which deals with general aspects of the CP/M operating system. The following chapters each deal with the individual commands provided by CP/M Versions 1.3 through 2.2.

Chapter 3



CP/M Versions:

1.3 **1.4** **2.0** **2.2**

Command Type:

Built-In

SUMMARY

The DIR command allows the CP/M user to display or print a list of file names that are stored on a disk. This list represents a "directory" of programs, data files, text modules, and other information stored on the disk, and provides important information regarding the "names and types" of files stored on the disk media inserted in a given drive. (See Chapter 2 for discussion of legal file names and types). Using wildcards, the DIR command may be used to list the complete directory of ALL the files stored on a given disk, only a specified group of files, or a single specific file.

To use the command, one simply types "DIR" in response to the CP/M prompt (A>, B>, etc.). The command may also be followed by a qualifying filespec.

ACCESSING DIR

DISPLAYING THE COMPLETE DIRECTORY OF A DISK



When DIR is used in its simplest form, it displays the complete directory of the disk inserted in the logged drive. The following example shows how the directory of the disk in Drive A is displayed:

```
A>DIR

A: STAT    COM : PIP   COM : SYSGEN  COM : MOVCPM COM
A: FORMAT  COM : LOAD  COM : ASM     COM : DDT    COM
A: SUBMIT  COM : XSUB  COM : ED      COM

A>
```

As shown, the DIR command will display the complete directory of the disk inserted in the logged drive, when the command is stand-alone and is not followed by a qualifying filespec. Note that as soon as the displaying of the directory is completed, the original CP/M prompt ("A>") reappears to indicate that CP/M is ready for another command.

The DIR command can also be used to display the complete directory of a disk inserted a drive other than the logged drive. This is done by typing in "DIR" followed by the drive name, such as "B:" or "C:". A space must be typed to separate "DIR" and the drive name. The next example shows how the directory of the disk in Drive B is obtained when Drive A is the logged drive.

```
A>DIR B:

B: STAT     COM : PIP      COM : TEXT01  DOC : YBASIC COM
B: ACCOUNTS BAS : ACCOUNTS TXT : LIBRARY LIB : TEXT01 BAK
B: SUBMIT   COM : XSUB     COM : ED      COM

A>
```

Even though the directory of the disk in Drive B was specified, the CP/M prompt "A>" will reappear when the display of the directory has ended, because Drive A still is the logged drive.

If you were to insert a properly formatted, CP/M compatible, *blank* disk in a drive, the following message would appear in response to the DIR command.


```
A>DIR B:

NO FILE

A>
```

This message would also appear if you used the DIR command to locate a specific file or group of files, that could not be found. The same would occur if you were to specify a filespec containing a nonexistent file.

Using DIR to locate specific files is covered next.

DISPLAYING THE DIRECTORY OF A SPECIFIC FILE OR GROUP OF FILES

```
DIR TEXT01.DOC
DIR *.DOC
```

The next example shows the use of the DIR command followed by a filespec as a qualifier, here TEST01.DOC. Only that file will be displayed if it is on the disk media.

```
A>DIR TEXT01.DOC

A: TEXT01 DOC

A>
```

The DIR command is often used to display the directory of a group of files. This is done by using the CP/M defined wildcard characters, as discussed in Chapter 2. Wildcards are also useful when the complete name of a specific file is unknown, but you think you know what some of the characters

may be. The following example shows the use of a wildcard in the filespec as a substitution for the file name, with the extension "DOC". Thus only files with the last three characters (the extension) "DOC" will be displayed.

```
A>DIR *.DOC

A: TEXT01    DOC : TEXT05 DOC : MANUAL1 DOC : INFO DOC
A: ACCOUNTS  DOC : HELP   DOC

A>
```

PAUSING THE DISPLAY OF THE DIRECTORY: In many cases, the directories of disks are quite large. As a result, the displayed information will scroll off the top of the screen (if you are using a video terminal as the console device) before the directory ends. When this occurs, the CTRL key and "S" key can be simultaneously pressed (CTRL/S) to cause the scrolling to "freeze." This will allow you to leisurely read segments of the directory. By pressing CTRL/S again, the directory will resume scrolling. This procedure can be repeated as many times as desired. When used with high speed terminals (4800 baud and up) the DIR command can respond very quickly after it is typed, and the information can scroll by before it can be read, it is recommended that you first position your figures over the console keyboard in such a way that CTRL/S can be typed before the directory disappears off the screen.

STOPPING THE DIR COMMAND: By pressing any key, other than CTRL/S, the scrolling of the directory ceases and the execution of the DIR command stops. When this occurs, the original CP/M prompt (A>, B>, etc.) reappears.

Chapter 4

REN

CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

Built-In

SUMMARY

The REN command is used to rename files contained in the directory of a disk, and can be used at any time, regardless of the disk one is using. Once a file has been renamed, the new name will be stored on the disk until it is changed or the file is deleted. Only one file can be renamed at a time, as the REN command always requires a filespec with a specific file name. Wild-cards cannot be used in conjunction with REN.

ACCESSING REN

RENAMING A FILE ON THE LOGGED DRIVE

```
REN NEW.TXT=OLD.TXT
```

To rename a file on a disk, a specific command format must be used. You might think that to rename something one would say the following: "Rename

the old name to the new name." The CP/M operating system, as well as many other operating systems, performs these types of functions in a different manner. The REN command renames files in a "logical" way, as is typical with most computers. That is to say, the format for renaming a file is the reverse of what one would normally assume. Thus, the renaming procedure used by REN would be rephrased as, "Rename: *new* name equals *old* name." This is an important distinction to remember as all other CP/M functions that require specifying two or more files in a command string will be done in the same way. You can easily remember this with the rule "new on the left, old on the right" (left wing people are generally younger than right wing people?). The following example demonstrates how the file "OLD.TXT" is renamed to "NEW.TXT".

```
A>DIR *.TXT

A:OLD  TXT

A>REN NEW.TXT=OLD.TXT

A>DIR *.TXT

A:NEW  TXT
```

As just demonstrated, you can use the DIR command first to verify the existence of the file to be renamed. When this is done, DIR can be employed again to verify that the file is properly renamed after the REN command. When renaming files, the full file name including the extension (if it has one) must be specified for the file that is to be renamed. The new file name need not have an extension, but is recommended that one be used as it will identify the type of file for future reference. When an attempt is made to rename a file that is nonexistent (it hasn't been properly specified or it's not on that disk), the following message appears.

```
A>REN NEW.TXT=OLDFILE.TXT

NO FILE

A>DIR OLDFILE.TXT

NO FILE

A>
```

In CP/M Version 2.0 or above, files are often set to a particular "protection status." This means that the file name and the contents of the file can only be modified in a certain way. A file may be set to Read/Write (R/W) or Read-Only (R/O). Files that are R/O are fully protected and cannot be modified (as the REN function must do) for changing their information. When an attempt is made to rename a R/O file, the following message is displayed.

```
A>REN NEW.TXT=OLD.TXT

File is set R/O                <---- The Return key is typed to
                                     return to the prompt.
A>
```


In the preceding example, the file "OLD.TXT" was previously set to R/O and the renaming of the file was subsequently prevented. For information on how to verify the protection status of a file prior to renaming it, and how to change the protection status, refer to the description of the STAT command in Chapter 8.

RENAMING A FILE ON ANOTHER DRIVE



The REN command may be used to rename a file on a specified disk drive, other than the logged drive. This is done by including the drive name in the first filespec specified (new file name) after "REN". The drive name should not be included in the second filespec (old file name). The following example shows the renaming of a file on Drive B while Drive A remains the logged drive.

```
A>REN B:NEW.TXT=OLD.TXT

A>
```

Only one instance of the disk drive name need be present in the command string, as long as it follows "REN" and precedes the new and old file names. The following command sequence is also allowed, but is unnecessary.

```
A>REN B:NEW.TXT=B:OLD.TXT

A>
```

If the drive name is not properly placed in the command string, or if two different drive names are specified, an error will occur. The following example shows improperly structured command strings.

```
A>REN NEW.TXT=B:OLD.TXT
B:OLD.TXT?

A>
A>REN B:NEW.TXT=A:OLD.TXT
A:OLD.TXT?

A>
```

The same error messages for renaming files on the logged drive apply when renaming files on another drive.

Chapter 5

ERA

CP/M Versions:

1.3 **1.4** **2.0** **2.2**

Command Type:

Built-In

SUMMARY

The ERA (usually pronounced *erase* but sometimes just *eee rrr aye*), is used to erase or delete specified files on a disk. ERA can erase files on the logged disk or it can be directed to erase files on another disk. A specific file, a group of files, or all files can be specified for deletion.

ACCESSING ERA

ERASING A SINGLE FILE

```
ERA TEXT01.TXT
```

To erase a file on a disk, simply type "ERA" followed by a filespec containing the full file name to be erased, in response to the CP/M prompt. If the file exists, and it is not protected in any way, it will be deleted from the disk's directory. Once a file has been erased, it will not be normally accessible

to CP/M.* The following example shows how file "TEXT01.TXT" is displayed in the directory, is erased, and then verified that it is no longer accessible.

```
A>DIR TEXT01.TXT

A:TEXT01  TXT

A>ERA TEXT01.TXT

A>DIR TEXT01.TXT

NO FILE

A>
```

Great care should be taken when using the ERA command; as once a file is deleted, it will no longer be available on the disk. You should, therefore, always double check to make sure that you really want to delete the file before the ERA command is used. It is good practice to make copies of all your important files, so that if a file is accidentally deleted, it can be recovered from a back-up disk. For information on how to back up a disk, refer to Chapter 2, page 41.

Files stored on disks other than the logged drive can also be specified for deletion with ERA. This is done by including the disk drive name within the filespec. For example:

```
A>ERA B:TEXT05.TXT

A>
```

*The ERA command actually does not erase anything. It just places a code in front of the file name in the directory saying that "This space is available." A commercially marketed restore program may be used to recover erased files. The book *Soul of CP/M* (ref. [2]) describes how to restore a file just using CP/M's DDT program.

ERASING GROUPS OF FILES

ERA *.TXT

A group of files can be specified for deletion with the ERA command by using wildcards, as described in Chapter 2. By using any combination of wildcards and text within the filespec, all files that fall under that category will be deleted from the specified disk. Caution should be taken not to delete a file contained in that group that you did not intend to delete.

HOW TO USE ERA AND WILDCARDS SAFELY

To avoid accidents, first use the DIR command followed by the exact same sequence of wildcards and text to verify that all the files contained in that group are to be deleted. If one or more files are contained in the group that should not be deleted, then the file or files should be renamed so that they do not fall within the group. Another safeguard would involve deleting individual files one at a time.

The following example shows that all files contained in the group "*.TXT" are to be deleted. The DIR command is used, and subsequently it is determined that the file "SAVE-ME.TXT" should not be deleted. The file is then renamed so that the intended ERA command string can be used.

```
A>DIR *.TXT

A:TEXT01   TXT : TEXT02   TXT : SAVE-ME TXT

A>REN SAVE-ME.DOC=SAVE-ME.TXT

A>ERA *.TXT

A>DIR *.TXT

NO FILE

A>
```

ERASING ALL FILES



The ERA command can be used to delete ALL files contained on a given disk. When this is done, the filespec specified must contain the "*.*" sequence to indicate that all file names and all extentions or file types are to be considered in the deletion process. In CP/M Versions 1.4 and above, the ERA command will ask, in response to this command string, if you really want to delete all files. This feature is provided as a double check. In earlier versions of CP/M, this double check is not made. If you type in "Y" for "YES" in response, then all files will be deleted. If you type in "N" for "NO" then the ERA command will be stopped before the files can be erased. Note that a lower case "y" will not work and ERA will assume you mean N. This is what it looks like:

```
A>ERA *.*

ALL (Y/N)?N                    <—— No files were deleted

A>
A>ERA *.*

ALL (Y/N)?Y                    <—— All files were deleted

A>
```

When a file or a group of files are not found on a disk while using the ERA command, then a message is displayed to indicate this. This is illustrated in the next example:

```
A>ERA PROG1.BAS

NO FILE

A>
A>DIR PROG1.BAS

NO FILE

A>
```


As we mentioned earlier, in CP/M Versions 2.0 and above, files are often set to a particular protection status. A file may be set to Read/Write (R/W) or Read-Only (R/O). Files with R/O are fully protected, which means that the file name and the contents of the file cannot be modified in any way. For information about R/W and R/O file attributes, refer to the transient command STAT description in Chapter 8. When an attempt is made to erase or delete a R/O file, the following message is displayed:

```
A>ERA PROG15.BAS

File is set R/O          <—— The RETURN key is typed
                             to return to the
A>                           prompt.
```

In this example, the file "PROG15.BAS" was set to R/O and the erasing of the file was subsequently prevented. For information on how to verify the protection status of a file prior to using the ERA command, and how to change the protection status, refer to the description of the STAT command in Chapter 8.

Chapter 6

TYPE

CP/M Versions:

1.3 **1.4** **2.0** **2.2**

Command Type:

Built-In

SUMMARY

The TYPE command lets the CP/M user display or print out the contents of a text file on screen or paper. Since this is a resident command, it can be accessed at any time, regardless of the disk drive currently in use. TYPE can be used to display the contents of *any* files which contain displayable or printable text (such as files with file types .TXT, .DOC, .HEX, .PRN, etc.). Nonprintable text files will produce strange effects when TYPEd out on the screen.

ACCESSING TYPE

DISPLAYING THE CONTENTS OF A FILE USING TYPE

```
TYPE TEXT01.TXT
```

To display the contents of a text file on the screen or paper, "TYPE" is entered in response to the CP/M prompt (A>, B>, etc.), followed by the

filespec. Wildcards are not allowed within the filespec when used with TYPE.

```
A>TYPE TEXT01.TXT

Quotes:

A ship on the beach is a lighthouse to the sea.
- Dutch proverb

Never go to sea with two chronometers; take one or three.
- Anonymous

The important thing is not to stop questioning. Curiosity
has its own reason for existing. - Albert Einstein

A>
```

As soon as the end of the file has been reached, the display will end and the original CP/M prompt will reappear. The TYPE command may be used to specify a file located on disk in a drive other than the logged drive, by including the drive name in the filespec. When using this command, only specific files may be indicated, i.e., wildcards cannot be used. If the file name within the filespec does not exist on the disk specified, the following message is displayed.

```
A>TYPE TEST20.ASM

TEST20.ASM?

A>DIR TEST20.ASM

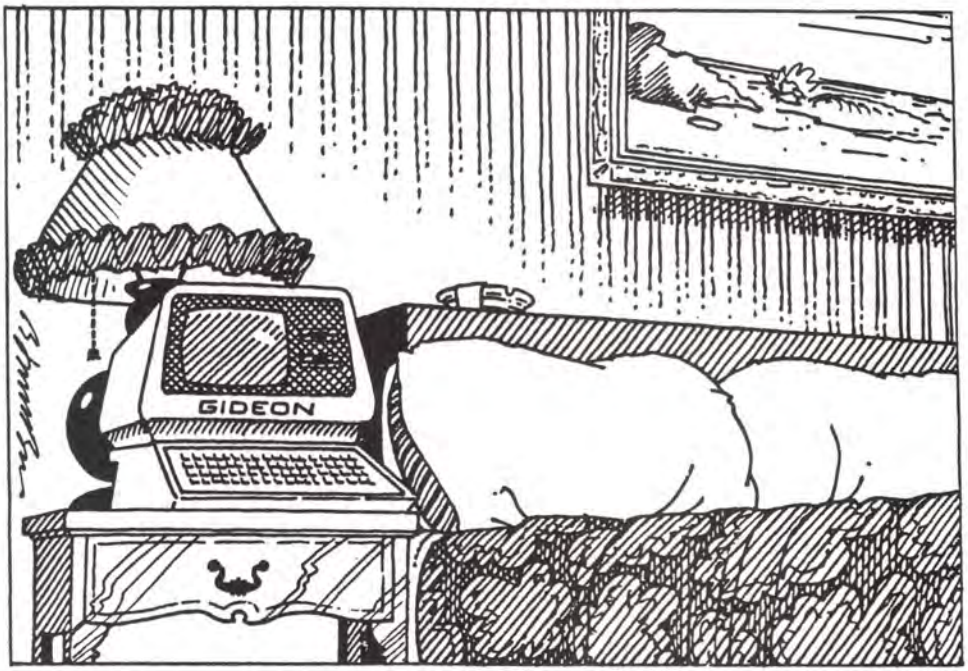
NO FILE

A>
```

The same type of message would appear if you were to specify a file name with illegal characters, such as wildcards ("*" or "?").

PAUSING THE DISPLAY OF A TEXT FILE: In most cases, the length of a file specified with the TYPE command will exceed the length of the screen

(if you are using a video terminal as the console device). The results are that the displayed information scrolls off the top of the screen before the end of the file is reached. When this is the case, the CTRL key and "S" key can be simultaneously pressed (CTRL/S) to cause the scrolling to "freeze." This allows you to read parts of the file at a time. By pressing CTRL/S again, the text resumes scrolling. This procedure can be repeated as many times as desired.



STOPPING THE TYPE COMMAND: By pressing any key, other than CTRL/S, the scrolling of the contents of a file ceases and the execution of the TYPE command stops, whereupon the original CP/M prompt (A>, B>, etc.) reappears.

If the file specified with the TYPE command was a nondisplayable file, i.e., a nontext file, the information displayed on the screen may consist of a series of strange and unrecognizable characters. When this occurs, sometimes the CP/M prompt does not reappear, and the system is temporarily locked. Normally a Warm-Boot (CTRL/C) is sufficient to restore the CP/M prompt. If this doesn't work, then the system should be reset and Cold-Booted. The displaying of such types of files should be avoided, since the information displayed will be, for the most part, unintelligible and useless.

Chapter 7

USER

CP/M Versions:
2.0 2.2

Command Type:
Built-In

SUMMARY

The USER command is only available in CP/M Version 2.0 and above. The command permits several different users of CP/M to have access to their own partitioned areas on the same disk. It allows the defining of up to 16 different user areas (0 through 15). This does not mean several users can use the same disk at the same time. CP/M is designed as a single user system, whereas MP/M is for more than one user. The CP/M USER command allows you to store programs in different areas of the same disk, and then each area can be dedicated to a specific user.

ACCESSING USER

When CP/M is first booted or loaded into the system's memory, the default user area on the system disk will be "0." This means that until you tell the computer otherwise, your designated user area will be number 0. Your directory is confined to user 0 and your files are all part of user 0. (Of course, you still can use up the space on the entire media.) When a new user area is first defined, such as User 1, it will not contain any files and its directory will be empty. The files in the User 0 area, or any other area for that matter, are not accessible by the person using the User 1 area, and vice versa. When a

particular user number is selected, the user area is active for all disk drives in the system, regardless of the logged disk drive*.

CREATING OR CHANGING USER AREAS



To create a new user area on all disks inserted in the system, or to simply change from one particular user area to another, "USER", followed by the desired number, is typed in response to the CP/M prompt. The user number selected must be within the range of 0 through 15.

```
A>USER 1

A>
```

When this command is typed, nothing of any consequence appears on the console device, apart from the reappearance of the original CP/M prompt of the logged drive (A>, B>, C>, etc.). Something has happened, however, as a result of what you've typed. The directory of the files in user area 0 will not be available to you in the new user area. If you type DIR, you will get NO FILES message if this is a new user area. When creating a user area for the first time, only the built-in or direct commands (DIR, REN, ERA, TYPE, etc.) are available for use. All transient command files are not available, because they exist in another user area.

Typing an illegal user number following the USER command causes the following error message to be displayed on the console device.

```
A>USER 16

16?

A>
```

*You can transfer programs between different user areas with the CP/M PIP utility or (more inconveniently) with the SAVE command.

The above error message was displayed because even though there are "16" user areas, there is no user area *number 16.* The USER command refers to the first area with a "0" and the last area with a "15."

CREATING A USER AREA FOR THE FIRST TIME: When creating a user area for the first time, you will notice that the area does not contain any files. This is demonstrated in the following example.

```
A>USER 1

A>DIR

NO FILE

A>
```

To create a directory of files in a new user area, it is necessary to have a copying program file, like "PIP.COM", in that area. PIP.COM, a transient command file, makes it possible for you to copy files from other user areas to the new user area (see the description of the PIP command). The following steps describe the correct sequence of commands used to move PIP.COM into the new user area, thus making it usable.

1. While in User 0, verify with the transient STAT command that the file "PIP.COM" is stored on the disk in this user area. Note the size of the file. The size will be given in kilobytes (normally, this value will be "8K"). More importantly, however, note the value in the "Recs" column. This value will normally be "58" and represents the number of 128-byte records the file uses. Write the exact number of records on paper for later use.
2. Load the file PIP.COM into memory with the transient DDT command (enter "DDT PIP.COM"). Once the DDT prompt (shown by the "-" character) is displayed, exit the program by performing a Warm Boot (CTRL/C). When you do this, the logged disk drive prompt reappears. The file PIP.COM is still stored in the system memory, i.e., RAM.
3. Enter the User command specifying the desired user area number, e.g., type USER 3.
4. Once the new user area has been selected, the built-in SAVE command is used to save the file PIP.COM, which is still in memory because of

the DDT load, from memory to the new user area of the disk. To use the SAVE command, however, you must first calculate the number of "memory pages" (a memory page equals 256 bytes) that are required to the complete file. To do this, retrieve the number of records obtained when the STAT command was used in Step 1, multiply the number by 128 (1 record equals 128 bytes) and obtain the total number of bytes used by the original file. Then divide the result by 256 to obtain the total number of memory pages. The final result is the number used with the SAVE command to save and store the file PIP.COM in the new user area. For example, if the number of records PIP.COM uses in 58, the following calculation is made to obtain an adequate number of memory pages for the SAVE command.

58 records × 128 bytes = 7,424 bytes (almost 8K)

7,424 bytes ÷ 256 bytes = 29 memory pages.

5. Once the SAVE command has been utilized to save PIP.COM, use the DIR command to verify that it has been stored on the disk. With this verification, you can then use the PIP command to copy files from another user area. A special parameter is added to the command string to specify the user area number from which the copying is to be done. This is described in the description of PIP in Chapter 9.

Note that CP/M only provides for the copying of files *from* other user areas, with PIP always in the user area you want to copy *to*. You cannot copy files *to* other user areas. Refer to the examples shown in the various sections of this book mentioned in the preceding steps, and read the descriptions of each before you attempt this procedure. Once you understand how to use these commands, return to this section and follow the instructions.

CHANGING USER AREAS: The maximum number of user areas on a disk is sixteen, 0 through 15. You can move around user areas at will by simply using the USER command followed by the number of the user area you wish to go to.

DISPLAYING THE ACTIVE USER NUMBER AND THE NUMBER OF USER AREAS THAT CONTAIN FILES: This is accomplished by using the transient command STAT. In order to use the command, you must have the file "STAT.COM" in the directory of the user area that you are in. The command is used as follows:

```
A>STAT USR:

ACTIVE USER:  0
ACTIVE FILES: 0 1 2

A>
```

This example shows that the user area number currently in use is 0, and that there are files contained in user areas 0, 1 and 2. The remaining user areas are empty.

Chapter 8

STAT

CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

Transient

SUMMARY

The STAT command, along with DIR, is one of the most commonly used CP/M utility programs. The command STAT (*Stat*us or *Stat*istics) is a transient program found on the disk under the name STAT.COM. The command is used extensively by the CP/M user for such things as listing files to determine file size details, the size in kilobytes of unused space on a disk and the attributes of the various files (read only, read/write, etc.). The STAT program has many other functions not frequently used. All of them provide the CP/M user with added flexibility in file and disk management. This section covers all uses of STAT, explaining the command sequences required for each function. The functions of STAT are briefly discussed in the following summary.

DISPLAYING UNUSED DISK SPACE (CP/M 1.3 or above) AND PROTECTION STATUS (CP/M 2.0 or above only): By using STAT in its simplest form, you can determine the amount of unused space on a disk. In CP/M Version 2.0 or above, verifying free disk space includes verifying the protection status of the whole disk*.

**Protection Status* A disk is considered protected when set to "Read-Only," and is unprotected when it is set to "Read/Write." A protected disk can only be read, which means that no modifications, deletions, or additions can be made to the disk. A protected disk is necessary

LISTING FILES: The STAT command can be used in a similar way to the DIR command to display or print a directory of files on a disk. But, in addition, STAT provides information, such as the size of each file and remaining unused space on the disk. In CP/M Version 2.0 or above, the protection status of each individual file (Read/Write or Read-Only) is displayed in the directory listing. A directory listing can be obtained using STAT from any disk inserted in a drive which is considered present by CP/M. As it is with the DIR command, STAT permits full usage wildcards ("*") and variable characters ("?") to list single files, groups of files, or complete directories.

PROTECTING FILES AND DISKS (CP/M 2.0 or above only): Individual files or complete directories can be "protected" so that they cannot be erased until the protection is removed. This is done by assigning the status of "Read-Only" to the file(s). Read-Only, or R/O, means that you can "read" the file(s) on the console device or print them, but you cannot modify or delete any portion of them. Conversely, files with the "Read/Write" status enable you to modify the contents, delete portions thereof, and erase the file from disk, as well as being able to read the file. Although seemingly restrictive, the Read-Only feature is very useful because it is often desirable (or even required) that certain files (but not the entire disk) be conditioned so that they cannot be accidentally erased. Thus we can either assign the disk to read only status or individual files on the disk. When a protected file needs to be modified or erased, however, the protection status can easily be removed using STAT by assigning a Read/Write status to files or to the entire disk.

HIDING FILES (CP/M 2.0 or above only): Individual files or complete directories can have assigned "attributes" which prevent them from being displayed when the DIR command is given. This is done when you want to hide a file when the DIR command is used. Hiding a file may be done for protecting the user from seeing a cluttered directory of files, particularly when a program contains many small modules. Hidden files can only be displayed when STAT is used to list the directory. Apart from its lack of appearance in the DIR directory listing, all other characteristics of any files possessing this "hide" attribute stay the same. The hide attribute can be reversed any time by assigning the opposite attribute using STAT.

---

when programs or files on the disk should never be written over or changed. An unprotected disk on the other hand can be read from or written to, i.e., additions, modifications, and deletions can be made to it. The command can be directed to any disk that is inserted in a drive connected to your computer, providing that the drives are properly recognized by the CP/M operating system.

REVEALING CURRENT DISK USERS (CP/M 2.0 or above only): There are situations where the CP/M USER command (Version 2.0 or greater) has been given and a user other than the default USER 0 is "logged" in. In case the user is not clear on what user number he has typed, the STAT command can be used to reveal who the current user is (for example, user 1, user 2, . . . user 15), and how many other user areas of the disk contain files. Logging on to a disk with a user number other than the default allows you to create and manipulate files on the same disk as other users. You will not be able to "see" the files of the other users, nor will they be able to see yours (unless people log on with someone else's user number). This means that your files are controlled (read, write, delete, rename) only by you. Refer to the description of the USER command.

REVEALING DISK DRIVE CHARACTERISTICS: The STAT command can be used to display information about how data is stored on a particular disk inserted in any or all drives recognized as present by CP/M. The information consists of the physical drive name (A, B, C, etc.), and includes specific technical details about how the disk is formatted. The information will depend on the type of disk (soft-sectored, hard-sectored, or hard disk) and the type of format used (single or double density). The maximum capacity of the disk (in kilobytes) will be given, the number of directory entries, plus details of how the tracks and sectors of the disk are utilized for the storage of data.

ASSIGNING AND VIEWING LOGICAL AND PHYSICAL DEVICES: The STAT command is used for assigning devices (peripherals) such as video terminals, teleprinters, and modems, to logical device names (such as CON:, PTR:, LPT:). This is necessary so that the devices can be recognized properly by CP/M. The devices can be assigned or re-assigned at any time using the proper STAT command sequence, as long as the assignments are within the parameters allowed by CP/M. STAT can display the existing assignments as well as a list of all possible device assignments. For example, you can assign the CP/M logical list device to the "physical" printer device or the "physical" video terminal device.

ACCESSING STAT

The STAT command is a transient program that exists in its own file, separate from the direct (built-in) CP/M commands. Its file name is normally

"STAT.COM", although you could call it anything you wish, such as STATUS.COM, DISK.COM, FILES.COM, S.COM, etc. This file must exist on a disk inserted in a connected drive before attempting to use the command. Use the DIR command to verify that the file does exist on one of the logged disks.

Once it has been determined that STAT.COM is stored on the specified disk, just type in "STAT", since CP/M will automatically look for the file type ".COM" when an extension is absent from the command string. STAT can be accessed from any drive by preceding the command with the disk name or identifier.

DESCRIPTION OF STAT COMMAND VARIATIONS

DISPLAYING UNUSED SPACE AND PROTECTION STATUS ON ALL DISKS

STAT

By typing in STAT followed by RETURN, basic information regarding the status of the current logged disk is displayed. For example:

```
A>STAT
A: R/W, Space: 140k
```

If the CP/M Version is 2.0 or above, the information starts with the protection status of the disk, indicating whether it's Read/Write (R/W) or Read/Only (R/O). The example above shows "R/W", which means the contents of the disk may be read, modified and/or deleted. If "R/O" were displayed, it would mean that the disk could only be read, thus preventing the user from deleting or modifying any of the contents. The last piece of information (here 140K) indicates the amount of remaining space (expressed in kilobytes) available on the disk. (In CP/M Versions 1.3 and 1.4, the protection status will not be presented in this report. For these versions, only the remaining space on the disk is shown.) Our example shows "140K" for remaining space, which means there are approximately 143,360 bytes (140 ×

1,024) or 140 kilobytes of free space that is still available on the disk for storage.

When disks have been inserted in any drives other than A, and the disks have been recognized as "present" by CP/M (a CTRL/C was issued while the disks where inserted, or commands were directed to the disks), the STAT command displays the available free disk space and the status of ALL the drives containing disks. This is shown in the following example:

```
A>STAT
A: R/W, Space: 140k
B: R/W, Space: 358k
C: R/W, Space: 402k
D: R/W, Space: 210k
A>
```

This information will be displayed regardless of whether STAT was typed in at the A>, B>, C> or D> prompt. (The included status information in this example is the same as the previous one.) For Disk B to be recognized as "present" by CP/M when using STAT, either a "Warm Boot" (CTRL/C) must be issued, or *a command must be directed to that disk after the disk has been inserted in the disk drive.* If this has not been done, or if no specific "reading" or "writing" has been directed to the disk, the STAT command will sometimes only display the free disk space and status of drive A. However, simply issuing the STAT command will be sufficient to make the disks "present" and recognizable CP/M.

DISPLAYING UNUSED SPACE AND PROTECTION STATUS ON A PARTICULAR DISK

```
STAT A:
STAT B:
STAT C:
STAT D:
```

When a disk drive (A, B, C, etc.) is specified after STAT, a simple report of the remaining storage space available on the particular disk specified is given. For example:

```
A>STAT B:
Bytes Remaining on B: 358k

A>
```

This example shows that the report consists only of the remaining unused space on disk B. The protection status is not presented.

DISPLAYING THE STATUS OF SPECIFIC FILES



By specifying a filespec *after* STAT, the "statistics" of that file are displayed in columns. The statistics indicate the file's size, Read/Write or Read-Only status, and number of records. At the top of each column is a heading which identifies the statistic located under it.

```
A>STAT TEST.DOC

Recs  Bytes  Ext  Acc
 110    14k    1  R/W A:TEST.DOC
Bytes Remaining On A: 476k

A>
```



Optionally, a "size" attribute may be typed with "STAT TEST.DOC". This will provide another column of information. This attribute, however, need never be used because it only provides a duplication of the value displayed in the "Recs" column. Here's how it would look if you used it:

```
A>STAT TEST.DOC $S

Size  Recs  Bytes  Ext  Acc
 110   110    14k    1  R/W A:TEST.DOC
Bytes Remaining On A: 476k

A>
```

The meanings of these statistics are as follows:

Size: This column indicates the number of records in the file. The number is the same as that shown in the *Recs* column. The column is optional, and is only displayed when the "$S" attribute is typed following STAT command sequence.

Recs: A record is a unit of measurement in disk data storage and is used by CP/M so that the bytes of a file can be grouped together for more efficient file manipulation. In CP/M, a record contains 128 bytes. Note that there are 8 records per 1,024 bytes, or 8 recs per 1 kilobyte. The number in this Recs column in the previous example indicates that there are one hundred ten 128-byte records in the file.

Bytes: The number in this column is an approximation of the actual size of the file in kilobytes. The number shown in the example is "14K" which normally would translate to 14 kilobytes or 14,336 bytes (14 × 1K, where 1K = 1,024 bytes). Functionally, however, 14K actually represents less than 14,336 bytes because, first off, the number of records times 128 is less then 14K. To determine more exactly the number of bytes in the file, the following calculation can be made:

$$\text{Number of } \textit{Records} \times 128 \text{ bytes} = \text{Actual Bytes}$$
$$110 \times 128 = 14{,}080$$

Thus in our file there are 14,080 bytes, not the 14,336 bytes. As a result, there would seem to be 256 bytes, or two unused sectors. Actually, this is not the whole story either. Further complicating the accuracy of the *Bytes* size figure is the fact that CP/M stores its files on the disk in large chunks called "allocation units." An allocation unit in most double density versions of CP/M 2.2 is 2,048 bytes, or 2K bytes long. Thus the number under Bytes represents the size of the file double density systems in allocation

units, and not its true size in records. As a result, all files have a minimum size of 2K. A file that needs more then 2K will show 4K under *Bytes*, as allocation is in 2K increments. When in doubt, the number in the *Recs* column when multiplied by 128 will give the exact disk space consumed by the file when using STAT. Note that an allocation unit in single density disks is 2K, making the value under *Bytes* more accurate.

Ext: An Extent is another CP/M unit of measure. It consists of 16K bytes of disk storage space. Thus, if a file size is 16K or less, only one Extent will be shown. If a file exceeds 16K, then the number of Extents is incremented by 1 for every additional 16K bytes added to the size of the file. The value in this *Ext* column indicates that the file consists of 1 "logical *Ext*ent." Normally you can ignore the *Ext* figure.

Acc: *Acc* is short for "Access." The characters in this column indicate whether the file has the Read/Write (R/W) or Read/Only (R/O) attribute for file protection assigned. This column is included in the report in CP/M Version 2.0 or above only.

The filespec is the last item listed in our example. In normal use, the most important item in this report is the *Bytes* heading, because it displays the approximate size of the file. File size can be compared with remaining space on the disk. For instance, you could use this information to determine that an 82K file can fit in 83K of unused disk space.

The next examples provide some variations on the preceding command sequence.

DISPLAYING THE STATUS OF ALL FILES

```
STAT *.*
```

This command sequence displays *all* the files stored on disk A. As can be seen in the next example, the STAT command lists the files (in alphabetical order according to file name), followed by their file types.

```
A>STAT *.*
Recs  Bytes  Ext  Acc
  35     6k    1  R/W A:COS.FOR
 258    34k    3  R/W A:DISKINFO.DOC
 151    20k    2  R/W A:FLOW.DOC
   1     2k    1  R/W A:INIT.SUB
  58     8k    1  R/W A:SAMPLE.DOC
  10     2k    1  R/W A:SINE.FOR
  41     6k    1  R/W A:STAT.COM
  71    10k    1  R/W A:TAN.FOR
 110    14k    1  R/W A:TEST.DOC
 149    20k    2  R/W A:TEST.BAS
 149    20k    2  R/W A:TEST.INT
Bytes Remaining On A: 476k
```

In addition to displaying complete directories, STAT can be used to display partial directories of specific groups of files. This can be accomplished in the same manner as with the DIR command. The same combinations of wildcards ("*" and "?") we discussed in Chapter 2 can be used. These are some command line examples:

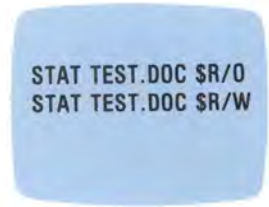
```
STAT FILE.*
STAT *.TYP
STAT FOR*.*
STAT FOR??.T?T
STAT FOR??.*

. . . etc.
```


Scrolling Control of STAT

Many times, disk directories contain so many files that the report displayed by STAT *.* scrolls off the screen before it can be read. To prevent this from happening, keep your fingers near the CTRL (Control) key and the "S" key while issuing the command. As soon as the screen starts displaying the directory, press the CTRL and the "S" keys simultaneously. This will stop (freeze) the scrolling of the directory, until you press CTRL/S again. This starting and stopping can be repeated as many times as necessary.

PROTECTING FILES



In CP/M Version 2.0 or above, the STAT command can also protect and unprotect individual files on a disk or complete directories. It is often desired to protect files so that they cannot be accidentally erased. The user must purposely *remove* the protection status of a file before it can be deleted.

Protection was discussed previously in this chapter. The following example illustrates the use of the protection attribute.

```
A>STAT TEST.DOC

Recs  Bytes  Ext  Acc
 110    14k    1  R/W A:TEST.DOC
Bytes Remaining On A: 476k
```

This example shows that the file has the R/W attribute. Assuming that you want to protect the file so that it can't be erased, the following command string can be entered:

```
A>STAT TEST.DOC $R/O

TEST.DOC set to R/O
A>STAT TEST.DOC

Recs  Bytes  Ext  Acc
 110    14k    1  R/O A:TEST.DOC
Bytes Remaining On A: 476k
```

After CP/M has assigned the R/O attribute to the file, STAT can be used to list the file and verify the protection status. Note that the *Acc* heading now says R/O. If you try to erase a file that has an R/O status assigned to it, the following error message will occur:

```
A>ERA TEST.DOC

BDOS ERR ON A: FILE R/O
```

Upon receiving the error message shown above, the cursor remains on the same line as the message. Pressing the RETURN key or performing a Warm Boot (CTRL/C) brings the cursor to the next line where the CP/M prompt reappears.

The protection attribute of the entire disk in the logged drive can be changed at any time from R/O to R/W, or vice versa, by using STAT. As we mentioned previously, the purpose of this R/O protection attribute is to prevent accidental erasure of files. Files stored on disks other than the logged drive can also have these attributes altered. This is done by including the disk drive within the filespec. Wildcards (*) and variable characters (?) can be used in the same combinations within the filespec as demonstrated previously for listing files. This is often useful when complete directories, or specific groups of files, must have a certain protection attribute assigned to them.

<(/)> Transfer of Files Between Different Versions of CP/M

When transferring files from CP/M Version 2.0 or above, to earlier versions, all files should have the assigned protection attribute of "R/W". Otherwise, errors may occur when manipulating the files using the earlier version of CP/M. Note that there is no way of set up and change the protection status of a file in earlier versions of CP/M. When transferring files from earlier to later versions of CP/M, the files will default to R/W status. Steps should therefore be taken to change the attribute, if files are to be protected.

PROTECTING COMPLETE DISKS TEMPORARILY

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

CP/M provides the capability of temporarily assigning disk protection, so that the disk is either R/W or R/O. When the protection is set to Read-Only (R/O), the complete contents of the disk will be protected, regardless of any

files set to R/W. This feature is provided to allow users to set a disk to R/O so that others cannot inadvertently write to it. This protection status will be effective only during the time that the disk is inserted in the drive. Once the disk is removed, the protection status reverts to Read/Write. As always, to remove the read-only status on the disk, a Warm Boot (CTRL/C) must be issued, setting the disk to Read/Write (R/W). This also shows that the so-called temporary protection status of CP/M is not viable if the user enters CTRL/C.

The next example shows how to use STAT when temporarily setting a disk to Read-Only.

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

A>
```

To verify that the disk is set to Read-Only, the basic STAT command is given.

```
A>STAT

A: R/O, Space: 149k
```

If an attempt is made to write to a write-protected (R/O) disk, the following error message will occur. By typing any key in response, a Warm Boot will occur and the disk will be set back to Read/Write. The example illustrates what happens when using the PIP command sequence to copy all files from disk B to disk A, when disk A is R/O. (The PIP command is described in detail later in this book.)

```
A>STAT

A: R/O, Space: 140k

A>PIP A:=B:*.*

BDOS ERR ON A: READ ONLY_      <—— RETURN was typed
                                   causing a Warm Boot

A>
A>STAT

A: R/W, Space: 140k
```

To set the disk back to Read/Write, either a Warm Boot (CTRL/C) or a cold boot (system re-boot) can be issued.

```
A>^C

A>STAT

A: R/W, Space: 140k
```

HIDING FILES

STAT TEST.DOC $SYS
STAT TEST.DOC $DIR

In CP/M Version 2.0 or above, the STAT command permits the user to "hide" files so that they are not listed when the DIR command is given. The "SYS" attribute is used to avoid the monotony of displaying "system" files every time you use DIR. Such files might be STAT.COM, PIP.COM, or SYSGEN.COM, which commonly exist on all disks. This function may also be used to provide a certain degree of security (such as preventing naïve CP/M users from accessing the files). Hiding a file is accomplished by assigning the $SYS attribute to the file or group of files to be hidden.

The example below shows the "hiding" of the STAT.COM file.

```
A>STAT STAT.COM $SYS

STAT.COM set to SYS
A>
```

Once the file is set to "SYS," the DIR command will not display it, as shown next. However, if the STAT command is used to list the file (also shown), the file *will be* displayed. The difference is that now the file name will be enclosed in parentheses, to indicate that it is a hidden "system" file.

```
A>DIR STAT.COM

FILE NOT FOUND

A>STAT STAT.COM

Recs  Bytes  Ext  Acc
  41     6k    1  R/W A:(STAT.COM)
Bytes Remaining On A: 476k

A>
```

These parentheses distinguish a hidden file from other files on the disk that are not "hidden." All or any number of files can be set in this way. Wildcards (*) and variable characters (?) can be used to specify files to be assigned this "$SYS" attribute.

Another attribute is used in the same way as "$SYS" to reverse the "hidden" status of a file or groups of files. This is the "$DIR" attribute, and, as its name implies, the files specified will be displayed when the DIR command is given. For example, assuming that STAT.COM is now hidden, watch what happens when the STAT.COM $DIR command is executed:

```
A>STAT STAT.COM $DIR

STAT.COM set to DIR
A>
A>DIR STAT.COM

A:STAT.COM

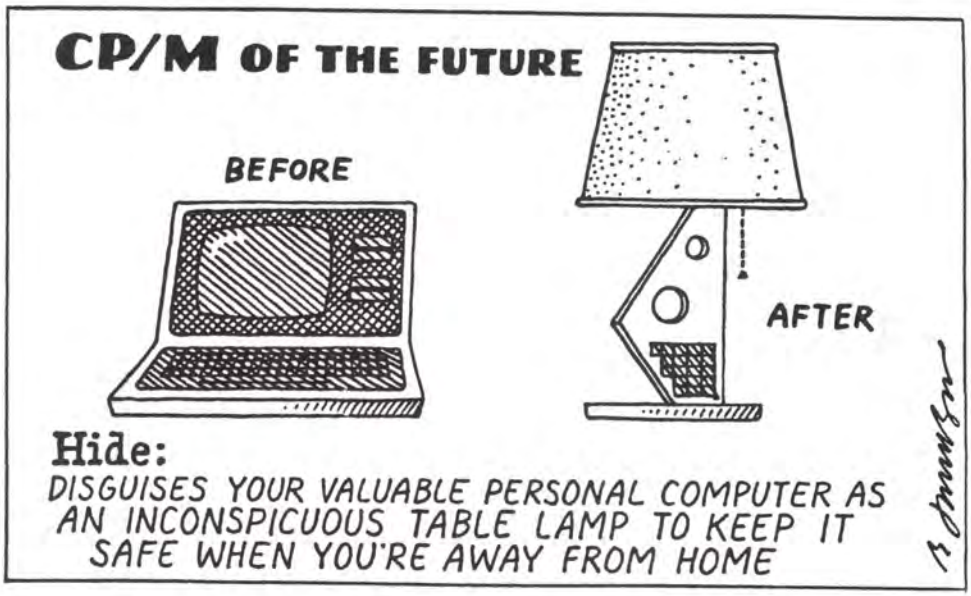
A>
A>STAT STAT.COM

Recs  Bytes  Ext  Acc
  41     6k    1  R/W A:STAT.COM
Bytes Remaining On A: 476k

A>
```

<(/)> Double Checking Files

When the DIR command fails to list the file(s) specified, you may wish to verify the same file sequence using STAT. This is done in case the file(s) was set with the $SYS attribute. This second verification method can ensure the existence or nonexistence of the file(s). Remember that STAT.COM $DIR can reverse the $SYS attribute.



VERIFYING NUMBER OF DISK USERS

STAT USR:

In CP/M Version 2.0 or above, the STAT command provides the ability to display the current "user number" on the disk. This number refers to which user is logged on and displays the numbers of all users with files on that disk. Refer to the USER command description for information on how user numbers are assigned.

```
A>STAT USR:

ACTIVE USER:  0
ACTIVE FILES: 0 1 2

A>
```

The above example shows that the active user (the one presently being used on the disk) has the number 0. There is a maximum of 16 users on a system. The "active files" line indicates which users actually have files stored on the disk. In this case users 0, 1, and 2 have active files. (If no other users than USER 0 were on the disk, then only both entries would show the number 0). STAT only has control over those files in the ACTIVE USER, no files in the inactive users are affected. In this example, only the files associated to "User 0" are affected. All other files will be ignored.

DISPLAYING DISK DRIVE CHARACTERISTICS

STAT DSK:

The STAT command provides the capability of displaying detailed disk information regarding capacity, number of directory entries, etc. This information may only be important to the advanced CP/M user who is interested in the inner details of diskette format. When the DSK: argument follows the STAT command, ALL drives that CP/M is aware of (A:, B:, etc.) will have detailed disk drive media information displayed. A single disk drive name can also be specified with the DSK: argument, and then only the characteristics for the specified drive will be displayed. In this last case, the argument would consist of "STAT X:DSK:" where "X:" is the specified drive.

The following five examples show the information that is displayed for different density formats of 8-inch standard soft-sectored floppy disks.

Examples 1 through 4 show the characteristics of single-sided disks, in single-density (128-bytes per sector), double-density with 256-bytes per sector, double-density with 1,024-bytes per sector, and a different double-density disk with 256 bytes per sector, respectively. Example 5 shows the characteristics of a double-sided disk in double-density with 1,024 bytes per sector. Explanations for each characteristic follow the examples.

The examples show the display for Drive A. It assumes that CP/M does not consider any other drives to be present.

Example 1
(Single-Sided, Single-Density Floppy Disk, with 128 bytes per sector)

```
A>STAT DSK:

   A:  DRIVE CHARACTERISTICS
1944:  128 Byte Record Capacity
 243:  Kilobyte Drive Capacity
  64:  32 Byte Directory Entries
  64:  Checked Directory Entries
 128:  Records/ Extent
   8:  Records/ Block
  26:  Sectors/ Track
   2:  Reserved Tracks
```

Example 2
(Single-Sided, Double-Density Floppy Disk, with 256 bytes per sector)

```
A>STAT DSK:

   A:  DRIVE CHARACTERISTICS
3856:  128 Byte Record Capacity
 482:  Kilobyte Drive Capacity
 128:  32 Byte Directory Entries
 128:  Checked Directory Entries
 128:  Records/ Extent
  16:  Records/ Block
  52:  Sectors/ Track
   2:  Reserved Tracks
```

Example 3
(Single-Sided, Double-Density Floppy Disk, with 1,024 bytes per sector)

```
A>STAT DSK:

   A:  DRIVE CHARACTERISTICS
4784:  128 Byte Record Capacity
 598:  Kilobyte Drive Capacity
 128:  32 Byte Directory Entries
 128:  Checked Directory Entries
 128:  Records/ Extent
  16:  Records/ Block
  64:  Sectors/ Track
   2:  Reserved Tracks
```

Example 4
(Single-Sided, Double-Density Floppy Disk, with 256 bytes per sector)

```
A>STAT DSK:

   A:  DRIVE CHARACTERISTICS
3888:  128 Byte Record Capacity
 486:  Kilobyte Drive Capacity
 128:  32 Byte Directory Entries
 128:  Checked Directory Entries
 128:  Records/ Extent
  16:  Records/ Block
  52:  Sectors/ Track
   2:  Reserved Tracks
```

Example 5
(Double-Sided, Double-Density Floppy Disk, with 1,024 bytes per sector)

```
A>STAT DSK:

   A:  DRIVE CHARACTERISTICS
9600:  128 Byte Record Capacity
1200:  Kilobyte Drive Capacity
 256:  32 Byte Directory Entries
```

```
256: Checked Directory Entries
128: Records/ Extent
 16: Records/ Block
 64: Sectors/ Track
  4: Reserved Tracks
```

The values shown in Example 1 (single-density, 128-bytes per sector) apply to all versions of CP/M (1.3 through 2.2). The values shown in Examples 2 and 3 (double-density, 256 and 1,024-bytes per sector, respectively) apply only to CP/M version 2.0 or above. Other types of disks, such as double-sided and hard disks, will have different values for each line. Examples 4 and 5 illustrate the characteristics for a different computer than in examples 1 through 3. Example 4 is a 256 bytes per sector single-sided, double-density disk, while Example 5 is for a double-sided, 1,024 bytes per sector disk. The following explains the meaning of the labels in the display.

128 BYTE RECORD CAPACITY: Indicates the maximum number of 128-byte records that may be stored on the disk. A record is a unit of measurement in data storage. It is used by CP/M to group bytes of a file so that file manipulation is more efficient. A record is a subdivision of a kilobyte or K-byte. Whereas 1K bytes equals 1,024 bytes (*not* 1,000 as one would normally assume); 128 bytes can divide 1,024 (1K) bytes equally, as follows:

1,024 divided by 128 = 8 records

As can be seen in our five examples using STAT DSK:, the value given in the Record Capacity line of the report increases from single-density to double-density, and from 256-bytes per sector double-density to 1,024-bytes per sector double-density.

KILOBYTE DRIVE CAPACITY: Indicates the maximum number of kilobytes that may be stored on the disk. This is usually the most useful line of information in the report. It indicates the amount of free space on an empty disk. As can be seen in the preceding examples, the kilobyte capacity increases significantly from single-density to double-density, and from 256-bytes per sector to 1,024-bytes per sector.

32 BYTE DIRECTORY ENTRIES: Indicates the maximum number of files you may store on a disk. A directory entry is 32-bytes long and never changes. For single-density disks (Example 1) 64 entries are usually adequate

(64 files). For double-density disks, with higher storage capacity, the number increases to 128 entries.

The maximum number of files will be LESS if any files are larger than one "extent" (i.e., 16K bytes). Thus as files get very large (greater than 16K) the maximum number of different files you can store on the disk decreases. For example, if ten different 48K-byte files are on the disk the following will happen. Each takes 3 extents (48 divided by 16 equals 3). Thus there are $10 \times 3 = 30$ directory entries consumed and only 34 ($64 - 30 = 34$) are left.

CHECKED DIRECTORY ENTRIES: When the storage medium is floppy disk, this value is usually the same as the value on the "32 BYTE DIRECTORY ENTRIES" line. For hard disk drives with nonremovable media (sealed hard disks), the entry will usually be "0." This value is listed because CP/M must check for changed media (i.e., when changing floppy disks without rebooting the system). The value defaults to "0" for nonremovable sealed hard disks, since they are never removed.

RECORDS/ EXTENT: This value determines the maximum number of records per extent in the directory entry. (An Extent equals 16K-bytes (128 records × 128 bytes/record = 16,382)). Note that a file may be made up of several extents. In our examples, we see that there are 128 maximum records per extent.

RECORDS/ BLOCK: This value indicates the MINIMUM amount of disk space which may be allocated to a file. Files are stored by putting them in what are called "allocation blocks". Such a block will be either 1K or 2K bytes long. The number of records (under the "Recs" column, when using STAT) must be used to calculate the actual number of bytes in the file. Note that in Example 1, there are 8 records per allocation block (8 records × 128 bytes = 1,024 bytes or 1K). In Examples 2 through 5, there are 16 records per allocation block (16 records × 128 bytes = 2,048 bytes or 2K). This allocation block size is standard throughout the disk. When the contents of a file equals less than 1K for single-density, or 2K for double-density, the remaining space of the block will be unused but still allocated to the file. This is why, for example, that on single-density files of less than 1K still display as 1K files when using STAT.

SECTORS/ TRACK: This value indicates the number of sectors contained in each track and this, in turn, depends on how the disk was formatted. For single-density disks, there are 26 sectors per track. For double-density disks, with 256-bytes per sector, there are 52 sectors per track. For double-density disks, with 1,024-bytes per sector, there are 64 sectors per track. You can see that 1,024 byte/sector formatting allows more capacity.

RESERVED TRACKS: This value indicates the number of tracks on the disk (usually Tracks 0 and 1) that are not available for the storage of files. This reserved area is used for storing the CP/M operating system (BIOS, BDOS, and CCP).

DISPLAYING DEVICE ASSIGNMENTS



The CP/M operating system provides the capability of assigning physical (actual) device names to logical (symbolic) device names. The logical device names are symbolic in the sense that actual physical devices are available to each one for assignment. This means each symbolic device can represent any of four *real* devices. There are four main logical devices in CP/M: Console, Reader, Transmitter, and List. Each of them may be assigned one of four physical devices. Physical devices may be printers, video terminals, keyboards, modems, graphics boards, plotters, other computer input or output ports, etc.

The STAT command also provides the capability of displaying the current assignments. The command string "STAT DEV:" is typed to produce the following report.


```
A>STAT DEV:

CON: is CRT:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:

A>
```

The logical (symbolic) devices are listed in the left-hand column. The actual physical devices to which each is assigned are listed in the right-hand column. The definitions for the logical devices are:

CON:—Console Device. This logical name is usually assigned for any physical device used for BOTH entering commands and displaying information. It is a bidirection logical device, meaning that it contains the ability for the input and output of data. This device usually provides the link between the CCP (Console Command Processor) and the operator terminal (video terminal or teleprinter). CON: is usually a video terminal or teletypewriter, however, it may also be a built-in video display and a separate keyboard. It may be used in a PIP command, such as "A>PIP CON:=B:file.typ" to display or print the specified file.

RDR:—Receive-Information/Paper Tape Reader Device. This logical name is used for any physical device that is used only to receive information. A typical physical device would be a paper-tape reader. It may be an input port for another keyboard, a digitizing tablet, an input only RS-232-C link to another computer (for transferring files back and forth), etc.

PUN:—Send-Information/Paper Tape Punch Device. This logical device name is used for any physical device only used for the sending of information. Typically, this physical device would be a paper-tape punch. It could also be a digital plotter, printer, output to another computer, etc.

LST:—Output List Device. This logical device is used for physical devices such as printers and other list (printing) devices.

The physical devices permitted by CP/M are described in the next few paragraphs.

DISPLAYING ALLOWED DEVICE NAMES AND OTHER ALLOWED STAT COMMANDS



In CP/M Versions 1.3 and above, the STAT command can be used to display a menu of all allowed physical device names that may be assigned to the four logical names. In CP/M 2.0 and above only, this menu includes

information about the allowed STAT commands and attributes described earlier in this section.

The following example shows the menu provided by STAT in CP/M Version 2.0 or above.

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

A>
```

The first part of the menu is self-explanatory since it lists features that have already been described in this section. The first line shows the sequence in which a disk is assigned the Read-Only (R/O) status. The "D" indicates the disk drive name. The second line shows the sequence for assigning attributes to files. The four attributes listed consist of "$R/O" (Read-Only), "$R/W" (Read/Write), "$SYS" (System (hiding a file)), and "$DIR" (Directory (the reverse of "$SYS")). The third line shows the sequence required to display disk characteristics ("DSK:" for all drives and "D:DSK:" for the specific drive name, "D:"). The fourth line shows the sequence for displaying the current "USER" number and the active files for all users.

The "IOBYTE ASSIGN:" portion of the STAT VAL: menu lists all the physical device names that may be assigned to each of the logical devices for input/output (I/O) purposes. An IOBYTE is a status located in the operating system memory that controls the assignment of the physical-to-logical devices. It is "programmed" to some default value when CP/M is first booted or when a CTRL/C is issued. It is otherwise reprogrammed by the STAT command.

Understand that the physical devices are numerous, but usually very few of them are implemented with actual hardware and software support. Often only the TTY:, CRT:, and LPT: devices are available. The distributor of your CP/M or the modifier of it would be the only ones who might change it.

These physical device names are explained below:

TTY:—Slow-Speed Console (Teletype Device). This physical device usually represents a slow-speed terminal, such as a teletypewriter or other terminal operating at 300 bits per second (bps) or less. TTY: can be assigned to any logical device, although it is usually assigned to CON: (Console) if it is being used as the operator's console.

CRT:—High-Speed Console (Cathode Ray Tube) Device. This device represents a high-speed console device that can only be assigned to CON: (Console). Cathode Ray Tube (CRT) video terminals usually operate at higher speeds than teletypewriters (often up to 9600 bits per second (bps), or higher).

BAT:—Batch Processing Device. Batch processing devices can only be assigned to CON: (Console). Console operator input to CP/M is made via RDR: (Receive-Information). This can be assigned for example, with TTY:. Also output from CP/M is made via LST: (Listing), assigned with TTY:, CRT:, or LPT: (Printer).

UC1:—User-Defined Console Device. This physical device name is assigned to CON: when special console equipment is used. The BIOS portion of CP/M is usually modified for this special device. As a result, this assignment will inform CP/M that the special equipment is present, and that the modified portion of the BIOS should be used.

PTR:—Paper Tape Reader Device. This physical device name is assigned to RDR: (High-Speed Reader) logical device when paper-tape reader equipment is installed.

UR1:—User-Defined Reader #1 Device. This device name is used when special equipment is installed for high-speed reading by CP/M. UR1: is assigned to RDR: (High-Speed Reader). Like all physical devices, CP/M's BIOS must be modified to use this device.

UR2:—User-Defined Reader #2 Device. This device name is used in the same way as UR1:, except it usually represents different special equipment. By this we are referring to equipment often requiring different modifications to CP/M's BIOS.

PTP:—Paper-Tape Punch Device. This physical device name is assigned to PUN: (High-Speed Punch) logical device when paper-tape punch equipment is installed.

UP1:—User-Defined Punch #1 Device. This device name is used when special equipment is installed for high-speed output by CP/M. The device is assigned to PUN: (High-Speed Punch) logical device.

UP2:—User-Defined Punch #2 Device. This device name is used in the same way as UP1:, except it usually represents different special equipment, often requiring different modifications to CP/M's BIOS.

LPT:—Line Printer Device. This physical device name is assigned to the LST: (Listing) logical device when a line printer is installed.

UL1:—User-Defined List Device #1. This physical device name is used with special output equipment. This includes printers that are different from the LPT: device, or special devices such as plotters. UL1: is assigned to LST: (Listing) logical device. These kinds of special equipment usually require modifications in the BIOS portion of CP/M. By assigning UL1: rather than LPT:, CP/M will be instructed to use the special code for it in the BIOS.

ASSIGNING PHYSICAL DEVICES TO LOGICAL NAMES

STAT log:=phy:

In addition to displaying logical to physical assignments, STAT can be used to make these assignments. This is done by first specifying the logical device name (CON:, RDR:, PUN:, or LST:), an equals sign (=), followed by the physical device name (TTY:, CRT:, LPT:, etc.). The following example shows the computer displaying the current device assignments, reassigning one of the devices, and then redisplaying the assignments.

```
A>STAT DEV:

CON: is TTY:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:

A>STAT CON:=CRT:
A>

CON: is CRT:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:

A>
```

If a device name is misspelled, or if the device is not allowed to have the assignment you've given it, the following message will appear.

```
A>STAT CON:=LPT:

INVALID ASSIGNMENT

A>
```

When several assignments are to be made, they can be grouped together on the same command line. Each physical-to-logical device assignment is separated from the other by a comma, and the command line is always terminated by a RETURN. An example of multiple assignments is provided below.

```
A>STAT CON:=CRT:,RDR:=UR1:,PUN:=UP1:,LST:=UL1:
A>
```

<(/)> CAUTION:

Caution should be taken when assigning devices. Do not assign a physical device for which there is no software support. You could find yourself in trouble with user-defined devices, for example. If these devices are not defined in the BIOS portion of CP/M, the system may "die" (hang up) when these device names are used in a CP/M command sequence. Your CP/M will have to be rebooted (Cold Boot) when this occurs.

Chapter 9



CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

Transient

SUMMARY

PIP stands for "**P**eripheral **I**nterchange **P**rogram." It is a transient program which allows the CP/M user to copy files between peripherals, i.e., from one disk to another, from a disk to a printer, etc. It also allows you to create a copy of a file on the same disk with a different name. The use of the words "peripheral interchange" in the program's name means that PIP transfers data back and forth between ALL types of devices, not just between disks. PIP can handle all types of files, including executable .COM files, text files, etc. In addition, the PIP command can be used to copy groups of files or individual files between any combination of devices. (Of course, these devices must be among those recognized by the PIP program and the CP/M operating system).

The following is a summary of the capabilities of PIP.

COPYING A FILE ON THE SAME DISK DRIVE: PIP can be used to copy a file on a disk. It does this by placing the copy in another file on the same disk. The file containing the copy can be named anything the user wishes.

COPYING A FILE OR FILES BETWEEN DIFFERENT DISK DRIVES: The program can be used to copy any individual file or almost any specified group of files between disk drives or other devices. The names of the files containing the copies can have the same or different names.

COPYING (CONCATENATING) SEVERAL FILES INTO ONE FILE: PIP can be used to specify a group of files to be copied and merged (concatenated) into one file. This can either be done on the same disk drive or between several drives.

COPYING A FILE OR FILES BETWEEN A DISK DRIVE AND A SPECIAL DEVICE: Built within PIP are certain parameters. They permit the user to copy files between a disk drive and a special peripheral, like the logical CP/M device representing a line printer, a modem, or whatever. PIP permits the copying of data in either direction, i.e., from a disk to a special device, or from a special device to a file on a disk.

COPYING DATA BETWEEN STANDARD CP/M DEVICES AND SPECIAL PIP DEVICES: With PIP, data transfers can be made between most standard CP/M logical and physical devices, as well as several special PIP devices. PIP can be used to copy files between two computers with different disk formats.

OPTIONAL PIP PARAMETERS: With almost any form of a PIP command sequence, the user can optionally specify single or multiple parameters. These parameters which instruct the program to perform certain operations while PIPing the file, are entered as attributes or "switches" to the normal PIP command string. The following is a list of the permissible PIP parameters:

- Display (echo) all text on the console device as it is being copied.
- Verify each copy operation by comparing the copy to the source file.
- Begin copying a file after a specified string of characters is reached, and/or terminate copying a file after a specified string of characters is reached.
- Convert upper-case characters to lower case during a data transfer, or all lower case to upper case.
- Delete all characters on each line of text after a specified column.

- Set tab stops to every column of a specified number.
- Remove all form-feed characters during data transfer, or insert a form-feed character after every specified number of lines.
- Add sequential numbers to each line as they are being copied.
- Specify "block mode" data transfer.
- Copy a file that has been set as a "system" file by the STAT command (CP/M Version 2.0 and newer only).
- Erase a file set to Read-Only (R/O) if it has the same name as the file to which PIP is copying (CP/M Version 2.0 and newer only).
- Copy from other disk user areas (CP/M Version 2.0 and newer only).
- Ignore end-of-file markers when copying machine/object code files.
- Perform checks for hexadecimal file format. As a result, any ":00" records during a hexadecimal file transfer can be ignored.
- Set the parity bit to zero for all ASCII bytes being copied.

ACCESSING AND USING PIP

To access the PIP transient command, you must have the PIP.COM file stored on a disk inserted in one of the computer's drives. There are basically two ways to access and use PIP. They are:

1. When you want to perform several consecutive operations using PIP, the program can be accessed by typing "PIP" in response to the logged drive prompt (A>, B>, etc.). This loads the program into memory, and a new prompt, called the "PIP prompt" (indicated by the "*" (asterisk) character) will appear. You can remain in this mode while performing all the operations you need. Press the RETURN key or initiate a Warm Boot command (CTRL/C) to exit from this mode.
2. PIP can be used when only a single operation is required and you want to return to the original CP/M prompt (A>, B>, etc.) immediately after the operation has been completed. This is accomplished by typing "PIP" followed by the command string representing the operation you want performed. Terminate by pressing the RETURN key.

The following paragraphs describe the various ways PIP can be used. We have provided details of correct syntax and format when specifying various combinations of parameters.

COPYING FILES BETWEEN DISK DRIVES

LOADING PIP AND PERFORMING COPY OPERATIONS



When using PIP for the first time, it's a good idea to access the command in its simplest form. This is accomplished by loading the program into memory without specifying any operation parameters. When "PIP" is typed in response to the logged drive prompt, the program is loaded into memory and the PIP prompt will appear on the console device. In response to this prompt, enter the operation you want PIP to perform. This process is demonstrated in the example shown next.

```
A>PIP
*
```

Once the PIP prompt is displayed, the program is ready to receive a command string.

The general format and construction of the command string for PIP is as follows:

*d:filespec=s:filespec1,filespec2, . . . etc.[option flags]

where

* = PIP Prompt
d: = destination device (to)
s: = source device (from)
filespec = optional filename and extension (at least one required)
option flag = one or several single-letter parameters

When the command string is entered, it is not necessary to accompany it with "PIP" because the program has already been loaded. The following example shows the command to copy a single file from Drive A to Drive B. After the command string is entered, press the RETURN key. The PIP prompt will reappear on the next line when the specified operation has been completed.

```
A>PIP
*B:=A:LETTER1.TXT
*
```

Note in this example, that Drive B and Drive A are entered in a peculiar sequence. The command sequence is saying, "I want disk B to contain file LETTER1.TXT which is presently on disk A." rather than "Copy LETTER1.TXT from disk A to disk B." The PIP command always uses this sequence. It is also used with most similar operations relating to other CP/M commands, such as the built-in REN (Rename) command. In the preceding example, Drive A is the "source" device and Drive B is the "destination" device. Thus, when speaking of the above command sequence, you could rephrase "B:=A:LETTER1.TXT" to "Destination Drive B equals Source file LETTER1.TXT on Drive A." In PIP, the destination is always on the left and the source on the right.

There are two other ways to copy a file from disk A to disk B. Remember, however, that the syntax or sequence "destination device equals source device" ("B:=A:" in the previous example) does not change. It doesn't change for any other PIP command sequences. The next example shows two variations in the way the file "LETTER1.TXT" can be copied from disk A on to disk B.

```
A>PIP
*B:LETTER1.TXT=A:LETTER1.TXT
*<CR>

A>
```

OR

```
A>PIP
*B:LETTER1.TXT=A:
*<CR>

A>
```

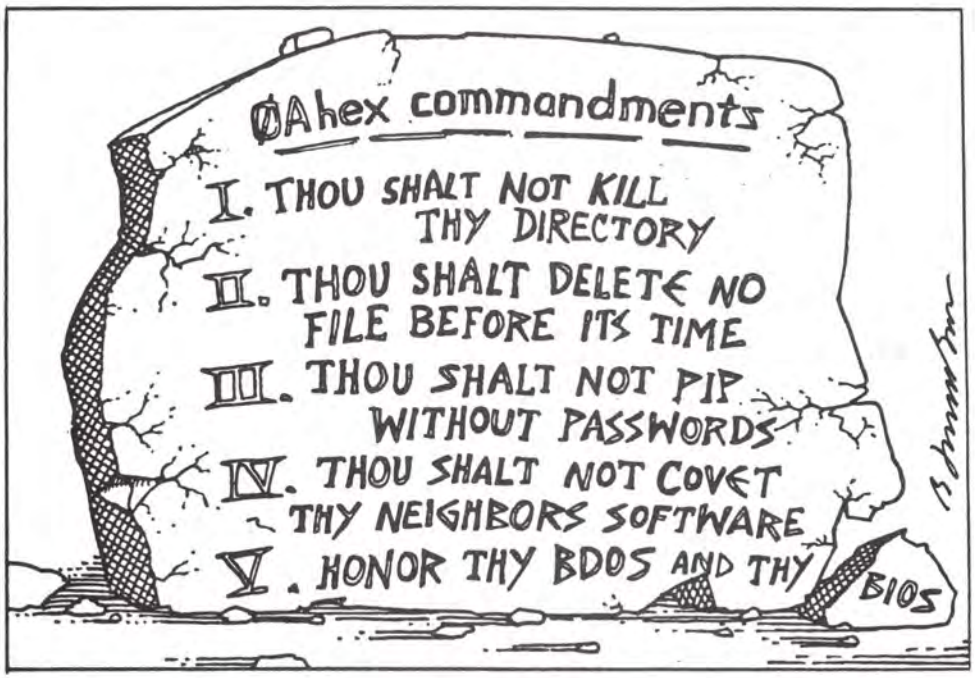
Of the three ways to specify a file copy command sequence, the one shown in the first example (i.e., destination device equals source device, followed by file name on source device) is the most commonly used and generally preferred.

Once the operation of copying the specified file between the two devices is complete, the PIP prompt will reappear. At this point, PIP is ready for another operation. While in this mode, you can specify as many operations as you want by repeatedly entering the command strings in response to the PIP prompts. You can return to the CP/M prompt at any time, by pressing the RETURN key in response to the PIP prompt.

The following example shows the copying of several different files between disk drives A: and B:, and then, finally, returning back to the CP/M prompt.

```
A>PIP
*B:=A:LETTER1.TXT
*B:=A:LETTER2.TXT
*B:=A:STAT.COM
*B:=A:ACCOUNTS.BAS
*B:=A:MAPS.FOR
*B:=A:NUMBERS.PAS
*<CR>

A>
```



While PIP is loaded into memory, you can specify groups of files to be copied with wildcard ("*" and "?") characters. The following example shows how PIP responds when this is done.

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

COPYING -
LETTER1.TXT
LETTER2.TXT

*B:=A:*.BAS

COPYING -
ACCOUNTS.BAS
PLOT5.BAS

*<CR>

A>
```

Any of the normal combinations of wildcard characters can be used with PIP operations. Refer to Chapter 2 (Using CP/M) for more information about using wildcard characters.

The PIP program also permits the specifying of the destination file name to be different from the source file name. By doing this, you effectively rename the "copy" of the file, not the source file itself. The next example demonstrates how source file "ACCOUNTS.TXT" on Drive A is copied into destination file "ACCOUNTS.DOC" on Drive B.


```
A>PIP
*B:ACCOUNTS.DOC=A:ACCOUNTS.TXT
*<CR>

A>
```

When copying a file onto the SAME disk that contains the original, you will need to specify a different name for the destination file. This is required because CP/M does not permit two files on the same disk to have the same name. The following example demonstrates how source file "LET-

TER1.TXT" on disk A is copied, thus creating "LETTER1B.TXT" on the same disk A.

```
A>PIP
*A:LETTER1B.TXT=A:LETTER1.TXT
*<CR>

A>DIR LETTER*.*

A : LETTER1 TXT : LETTER1B TXT

A>
```

During a copy operation, you cannot use wildcards when specifying a new file name for the destination file. This is because CP/M needs to know exactly what the file name is going to be when creating a new file. Wildcards can only be used with PIP when they are specified in the source field, if no file name was specified in the destination field. An error would occur if wildcards were used in the destination file names while using PIP. This is demonstrated in the next example.

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

IMPROPER FORMAT: B:SECOND.*

*<CR>

A>
```

The error message shown in this example is typical for most command specification errors when using PIP. The error message usually tells you where the fault lies in the command line, shown here as "B:SECOND.*".

The error message "IMPROPER FORMAT" will always display when wildcards are specified in destination fields. Only the source drive specification can utilize wildcards.

COPYING NONEXISTENT FILES: When a nonexistent file is specified as the source file in a PIP command string, an error message indicating the file name in question will be displayed. This is demonstrated in the following example.

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

NO FILE: =A:ZEROTEXT.DOC

*<CR>

A>DIR ZEROTEXT.DOC

NO FILE

A>
```

As can be seen in the above example, the nonexistence of the file was verified using the DIR command. In these situations, PIP still attempts to create the destination file even though no file was copied. If the directory on the destination drive is displayed, this "partial" file will be listed. The file name will be seen with an extension of ".$$$" which means the file was opened by CP/M but was not properly closed.* The STAT transient command can be used to display the directory of the destination drive. When "ZEROTEXT.$$$" is listed, it will have zero kilobytes (0K). Use the built-in ERA (Erase) command to delete this file.

OVER-WRITING EXISTING FILES DURING A COPY OPERATION: With CP/M Version 2.0 or newer, you may frequently encounter conflicting situations with a Read-Only (R/O) file protection attribute. (R/O is described in the STAT transient program description.) If a destination file name specified in a PIP command string already exists on the disk in the destination drive, the PIP program will over-write (write over) the file on the disk with the new copy only if the file is not write protected, i.e., it is set to Read/Write (R/W). (In CP/M Versions 1.3 and 1.4, files are

*The reason there is a filename.$$$ on the destination disk is because PIP trys to completely copy the file before erasing any file on the disk with a similar name. Only AFTER the copy is successful does PIP finally erase the old version and rename the filename.$$$ to filename.typ.

always R/W.) In CP/M Version 2.0 and newer, the PIP program will inform you if the destination file already in existence is set to R/O. It will ask you if the file should be over-written with the new copy, or terminate the copy process. If you answer "Y" the copy process will resume and the original destination file will be erased. If you answer "N", the message "NOT DELETED" will be displayed and that specific copy operation will terminate. To determine whether a file is R/W or R/O, use the STAT command to obtain the directory of the files on the destination drive. The next example demonstrates how PIP responds in this situation:

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

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

***NOT DELETED***

*B:=A:SAMPLE.TXT

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

*<CR>

A>
```

In the first part of this example, "N" was entered in response to the question given by PIP. The message "NOT DELETED" was promptly displayed, and the copy operation specified was terminated. In the second part of the example, "Y" was entered, which instructed PIP to proceed with the copy operation and erase the file on the destination disk.

COPYING PROTECTED FILES (CP/M Version 2.0 and newer only): When source files specified in a command string have been set to R/O, the file protection status is copied over only under certain circumstances. When copying an individual file, for example, the destination file, is set to R/W even if the source file was set to R/O. When a group of files are copied (using wildcards), however, the file protection status of the source files is also copied.

COPYING SYSTEM FILES (CP/M Version 2.0 and newer only): When a source file that has been set to the System (SYS) status, is specified in a PIP command string, the "NO FILE" message is displayed. This is the way PIP pretends that the file does not exist. To remedy the situation, the STAT command should be used to set the file to the Directory (DIR) status. Once this is done, the file can be copied as normal. Alternately, a PIP parameter flag can be specified to instruct PIP to go ahead and copy the file. This parameter flag is described later in this chapter.

INITIATING SINGLE-LINE PIP COMMANDS



There is a second way in which PIP operations can be specified. It differs from the previously described method in that the PIP program is temporarily loaded into memory, and only one command sequence may be specified at a time. This second method permits you to initiate a PIP command sequence from the logged drive prompt (A>, B>, etc.) and then return to the same prompt once the command function has been completed. (This is similar to the way other CP/M commands operate.) All the functions that may be performed by PIP while in the PIP mode, i.e., with the PIP prompt ("*"), can also be performed this way. Enter "PIP" in response to the logged drive prompt, follow with a space, and then enter the actual command sequence. The same command sequence syntax applies here as it does to all PIP operations.

The following example shows how a file is copied from disk A to disk B to return to the logged disk drive prompt once the copying is completed.

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

A>
```

The next example shows the response on the console device when a single-line PIP command sequence is entered to copy several files from disk A to disk B. Once all the files have been copied, the original logged drive prompt reappears.

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

COPYING -
SAMPLE.TXT
ACCOUNTS.TXT
LETTER1.TXT
LETTER2.TXT
MESSAGE.TXT

A>
```

COPYING SEVERAL FILES INTO ONE FILE



The PIP program provides a feature which allows you to copy several files into a single file. This process is called concatenating (linking) because the files copied are merged, or grouped together, in the order specified in the source file field. When the final copied file is viewed, it will be preceded by the text of the first original file. This is followed by the text of the second next file, and so on.

The correct syntax for this command sequence is as follows: A single destination file name must be specified, followed by the files you want copied in the order that you want them concatenated. Each file name is interspersed with a comma. The following example demonstrates how the source files "1.TXT" and "2.TXT" on Drive A are displayed to verify their contents, and

then how they are copied and concatenated into the destination file "ALL.TXT" on Drive B.

```
A>TYPE 1.TXT

DOCUMENT #1: This is the first in a series
of text files to be created on this system.

A>TYPE 2.TXT

DOCUMENT #2: This is the second in a series
of text files to be created on this system.

A>PIP B:ALL.TXT=A:1.TXT,2.TXT

A>TYPE B:ALL.TXT

DOCUMENT #1: This is the first in a series
of text files to be created on this system.
DOCUMENT #2: This is the second in a series
of text files to be created on this system.

A>
```

Many files can be specified for concatenation with PIP, although the length of the command string should not exceed the maximum allowed by the width of the console device screen (video terminal) or paper platen/roller (teletypewriter). Many kinds of files can be concatenated together, though files containing displayable text are the most commonly specified. You can concatenate object/machine code files, such as ".COM" transient command files, although the purpose in doing this would be for economical storage. In this case, the final concatenated file could not operate as a program, unless it was subsequently modified appropriately or designed for such concatenation beforehand.

All source files specified for concatenation need not be stored on the same device. By preceding each appropriate source file with a drive name, the PIP program can reference several source drives while the program is executing. The following example shows that the source files to be copied and concatenated into destination Drive B are located in Drives A, B, C, and D.

```
A>PIP B:ALL.TXT=A:1.TXT,B:2.TXT,C:3.TXT,D:4.TXT

A>
```

The concatenation command sequence can consist of any combination of source files and source devices. This is providing that the order entered represents the order in which you want the files concatenated. The command will only be executed correctly if all device and file names in the command string are valid.

USING STANDARD AND SPECIAL DESTINATION DEVICES WITH PIP

The PIP program provides the CP/M user with the ability to copy between other devices besides disk drives. In a PIP command sequence, almost any CP/M logical or physical device can be specified, as well as several special PIP devices (there are two special PIP devices which can be used for general output). A description of the standard logical and physical devices is provided in the STAT transient command description.

Almost all of the standard CP/M logical and physical output devices can be specified as destination devices in PIP command sequences. The only device not defined is the BAT: (Batch Processing) device. The following table provides a list of all the allowed PIP destination devices.

Logical:	CON: (Console)	PUN: (Output-Only)	LST: (List)
Physical:	TTY:	PTP:	LPT:
	CRT:	UP1:	UL1:
	UC1:	UP2:	
Special:		OUT:	PRN:

For more information on the characteristics of the standard logical and physical devices, refer to the STAT transient command description. The following are descriptions of the destination devices.

COPYING TO THE CONSOLE DESTINATION DEVICES (VIDEO TERMINAL OR TELETYPEWRITER)

```
PIP CON:=A:SAMPLE.DOC
PIP TTY:=A:SAMPLE.DOC
PIP CRT:=A:SAMPLE.DOC
PIP UC1:=A:SAMPLE.DOC
```

The console device may be specified as the destination device for a file. When it is, text can be displayed on the screen (video terminal) or printed on paper (teletypewriter). (This PIP command sequence has essentially the same effect as the built-in or direct TYPE command.) Either the logical CON: device or the physical devices TTY:, CRT:, or UC1: may be specified here. Remember that a physical device can only be specified if the STAT transient command has already been used to assign it logically to CON. This is true except for TTY: and CRT:, since PIP automatically assumes that the output device is to be CON:. As for UC1:, since it is a user-definable console device, the CP/M input/output routines in its BIOS must be modified accordingly. If UC1: is specified as a destination device, and CP/M has not been modified to support it, the system will probably crash, requiring a reboot (Cold Boot). Thus, the safest form of this command is always to specify CON: as the destination console device.

COPYING TO THE OUTPUT-ONLY DEVICES (PAPER TAPE PUNCH OR OTHER DEVICE)

```
PIP PUN:=A:PROG-1.HEX
PIP PTP:=A:PROG-1.HEX
PIP UP1:=A:PROG-1.HEX
PIP UP2:=A:PROG-1.HEX
PIP OUT:=A:PROG-1.HEX
```


One of the output-only devices may be specified as a destination device. The logical PUN: ("Punch") device is usually specified for this purpose. This device occasionally is used for purposes other than a paper-tape punch machine. To meet this requirement, any physical device can also be specified,

providing that in most cases, it is logically assigned to the PUN:. The default physical device for a paper-tape punch machine is PTP:. The UP1: and UP2: physical devices are user-definable. Unless CP/M has been modified for their use, they should never be used in a PIP command sequence.

There is also a special PIP device called "OUT:" that may be specified in a PIP command sequence. This is another special device that can be used to send information to a user-defined output destination. Normally this is assigned to the logical PUN: device. To use OUT:, the PIP program must be modified using the transient program DDT (Dynamic Debugging Tool). If you don't want to make any modifications, or if you're unfamiliar with the processes involved, please skip the following note. If you are familiar with the DDT command, and you have already determined what are the necessary modifications, the following information will be helpful.

<(/)> NOTE TO CP/M PROGRAMMERS:

In the PIP program, hexadecimal locations 103H through 105H hold a Jump instruction to your special input routine. The PIP program CALLs location 103H to input a character and, upon a RETURN, finds the character in location 109H. Locations 106H through 108H are reserved for the Jump to your special output routine. The PIP program loads Register C with the character to be output, and then CALLs location 106H. The locations 10AH through 1FFH are normally not used, and can be patched with your special routines.

For more detailed modification instructions, refer to the CP/M technical manuals.

COPYING TO THE LIST DEVICES (LINE PRINTER OR OTHER DEVICE)

```
PIP LST:=A:SAMPLE.DOC
PIP LPT:=A:SAMPLE.DOC
PIP UL1:=A:SAMPLE.DOC
PIP PRN:=A:SAMPLE.DOC
```

PIP allows the listing device to be specified as a destination device. The logical LST: device would be most commonly used, although the physical LPT: device

could also be used. The user-defined physical UL1: device can also be specified. In all cases, these will work only if CP/M has been modified for their use.

There also exists a special PIP device, called "PRN:". Using PRN: is equivalent to specifying LST: or LPT:, except that PIP automatically performs certain output formatting functions. The following functions are performed when PRN: is specified:

1. For every tab encountered, the output device will advance to the eighth character position.
2. All lines of text are sequentially numbered on output.
3. A form-feed command results in an initial page break and additional page breaks follow after every 60 lines of text.

USING STANDARD AND SPECIAL SOURCE DEVICES WITH PIP

In addition to the output or destination devices, there are several *input* or *source* devices which may be specified when using PIP. Once again, the physical BAT: (batch processor) device cannot be specified in a PIP command sequence. There are also three special PIP devices that may be used as source devices. The next table shows all of the permitted source devices.

Logical:	CON: (Console)	RDR: (Input-Only)	
Physical:	TTY:	PTR:	
	CRT:	UR1:	
	UC1:	UR2:	
Special:		INP:	NUL:
			EOF:

For more information on the characteristics of the standard logical and physical devices, refer to the STAT transient command description. The following are descriptions of the three special PIP source devices:

COPYING INPUT ON THE CONSOLE DEVICE INTO A FILE ON A DISK

```
PIP TEST.TXT=CON:
PIP TEST.TXT=TTY:
PIP TEST.TXT=CRT:
PIP TEST.TXT=UC1:
```

The PIP program can be used to direct the input on the console device to a file on a disk. This is accomplished by specifying the console device as the source device, and the file name and disk as the destination. The logical CON: device is usually used for this purpose, though the physical TTY: and CRT: devices can also be specified. The user-defined physical UC1: device should only be used if CP/M has been modified for its use.

To initiate this command sequence, the destination filespec is entered after PIP, followed by the source console device name. Once the RETURN key is pressed, PIP is placed in the console-to-disk file transmission mode. The console terminal will not operate normally. Instead, individual lines of text can be entered. When console input is completed, a CTRL/Z is entered (this is achieved by simultaneously pressing the CTRL and Z keys) which instructs PIP to insert an end-of-file marker in the file, close the file, and exit to CP/M. The file can be displayed later to verify its contents. This next example demonstrates how this is done.

```
A>PIP A:TEST.TXT=CON:

(text is entered at this point)
Now is the time for all good programmers to
save their work in files.
^Z    (CTRL/Z terminates input)

A>TYPE TEST.TXT

Now is the time for all good programmers to
save their work in files.

A>
```

With this command sequence, PIP will only terminate the data transfer if a CTRL/Z sequence is entered. All other command sequences will be entered into the destination file. They will not affect the way in which PIP operates while in this mode.

COPYING FROM AN INPUT-ONLY DEVICE INTO A FILE ON A DISK

```
PIP PGM1.HEX=RDR:
PIP PGM1.HEX=PTR:
PIP PGM1.HEX=UR1:
PIP PGM1.HEX=UR2:
PIP PGM1.HEX=INP:
```

The PIP program can be used to copy information received from the input-only source device into a file on a disk. The input-only device consists of the logical RDR: device. Usually a paper-tape reader machine, as represented by the physical device PTR: is assigned to this logical device. The PIP command sequence can contain as a source device either the logical RDR: device, or the physical PTR:, UR1:, or UR2: device. The user-definable UR1: and UR2: devices can only be used if CP/M has been modified for their use. When these devices are specified, PIP normally monitors the console device during data transfer. When a CTRL/Z is entered on the console device, data transfer terminates and the destination file is saved on the disk.

There is also a special PIP device called "INP:" which may be specified. This special device is used to retrieve information from a user-defined section of the PIP program. To use INP:, modify the PIP program via the transient program DDT (Dynamic Debugging Tool). If you do not wish to make any modifications, or if you are unfamiliar with the processes involved, please skip the following note. If you are familiar with the DDT command, and you know which modifications need to be made, the information will be helpful.

NOTE TO CP/M PROGRAMMERS:

The PIP program obtains input data on a character-by-character basis by CALLing hexadecimal location 103H. Data is returned in location 109H. The parity bit must be set to zero.

SPECIFYING LEADERS AND TRAILERS WHILE COPYING SOURCE FILES TO THE OUTPUT-ONLY DEVICE

```
PIP PUN:=NUL:
PIP PUN:=NUL:,A:PGM1.HEX
PIP PUN:=NUL:,A:PGM1.HEX,NUL:
PIP PUN:=A:PGM1.HEX,NUL:
```

NUL: is a special source device. It is used only when the destination is a paper-tape punch machine, assigned to the standard destination device PUN:. When the source device NUL: is entered in the command line accompanying a source file, the device must be separated from the source disk device and file name by a comma. For each occurrence of NUL: in the source field of a PIP command line, the program will send 40 ASCII null (NUL) characters to the output device. When the nulls are output, a 4-inch length of blank paper tape results. This is done to create "headers" and "trailers" to the actual data, which are useful for threading purposes. Normally, this special source device is not used for any other purpose. Some manufacturers or designers may modify CP/M so another type of device uses this source device and its null actions.

SPECIFYING AN END-OF-FILE MARKER SPECIAL SOURCE DEVICE WHILE COPYING A FILE TO THE OUTPUT-ONLY DEVICE

```
PIP PUN:=A:PGM1.HEX,EOF:
```

This EOF: (End-of-file marker) special source device is used to send a CP/M end-of-file marker (ASCII SUB character (hexadecimal 1A) or CTRL/Z) to the destination device. The destination device, in this case, is usually the logical PUN: device, assigned with a paper-tape punch machine. Its primary purpose is to terminate a file transfer and, therefore, can also be used to terminate a disk file. It is used when the files being transferred do not contain proper end-of-file markers, or when the destination device does not recognize the marker in a file. In normal operation, this special source device need never be specified since all correctly formatted CP/M files contain a CTRL/Z as an end-of-file marker. This source device may be used together with the

special NUL: device, as long as the devices and the file name are separated by a comma. The EOF: marker is useful when transferring files from non-standard CP/M devices and files created by other computer systems.

OPTIONAL PIP PARAMETERS

Special parameters can be added to PIP command lines to define certain operational characteristics during a data transfer. These parameters are usually entered at the end of a command string and are enclosed within square brackets ("[" and "]"). Individual or multiple parameters can be entered in the same command string. They must all be typed "in-line" and enclosed within a single pair of square brackets. All these parameters are optional, although you may find that you will use one or two almost all the time. The descriptions of the parameters are listed with those parameters most commonly used first, followed by those used less frequently.

VERIFY ACCURACY OF COPIED DATA BY COMPARING THE DESTINATION FILE WITH THE SOURCE FILE



The optional PIP parameter [V] (short for "Verify") is probably the most commonly used. When specified, it checks the copied data with the original data to verify accuracy of the transfer (it uses the normally hidden filename.$$$ file for this). If no error is found, the program will exit normally or resume with other commands. If an error is found, an error message is displayed on the console device. This [V] parameter will only work if the specified source and destination files are both on disks. When performing file-to-file copy functions, the [V] parameter is entered without a space immediately after the destination and source files. When copying several files and concatenating them into a destination file, the [V] parameter must follow the first source file. If this is not done, it will be ignored. It's a good idea to use this parameter at all times, since it will detect copying errors due to program malfunctions, faulty disks, disk drive hardware, or computer memory.

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

A>PIP A:ALL.TXT=B:1.TXT[V],2.TXT

A>
```

DISPLAY OR ECHO ALL TEXT ON THE CONSOLE DEVICE AS IT IS BEING COPIED

[E]

The "[E]" parameter (short for "Echo") can be used with any PIP command sequence that copies text from one device or file to another. The [E] parameter can be used in conjunction with any other parameters related to the transfer of ASCII text. It is normally used only with data consisting of displayable ASCII text. Using E with object/machine code files such as ".COM" files can cause weird results. This is because the values in the .COM files cause various control sequences (such as inverse, scroll, flash, tab, etc.) to be enabled on the display device. This parameter may be specified in a single file copy command sequence, although it is most often used when several files are being copied for concatenation into a single file. When the [E] parameter is used in this manner, the text of each file will be displayed (or printed) on the console device, as it is being copied. This tells the CP/M user which stage of copying has been reached (middle of file, end of first file, beginning of second file, third file, etc.).

It is advisable to use the [V] ("Verify") parameter with [E]. The next example shows the echoing of a single file as it is being copied.

```
A>;Contents of file TEST.TXT is as follows.
A>TYPE TEST.TXT
This is a test ASCII text file.
A>PIP B:=A:TEST.TXT[VE]
This is a test ASCII text file.

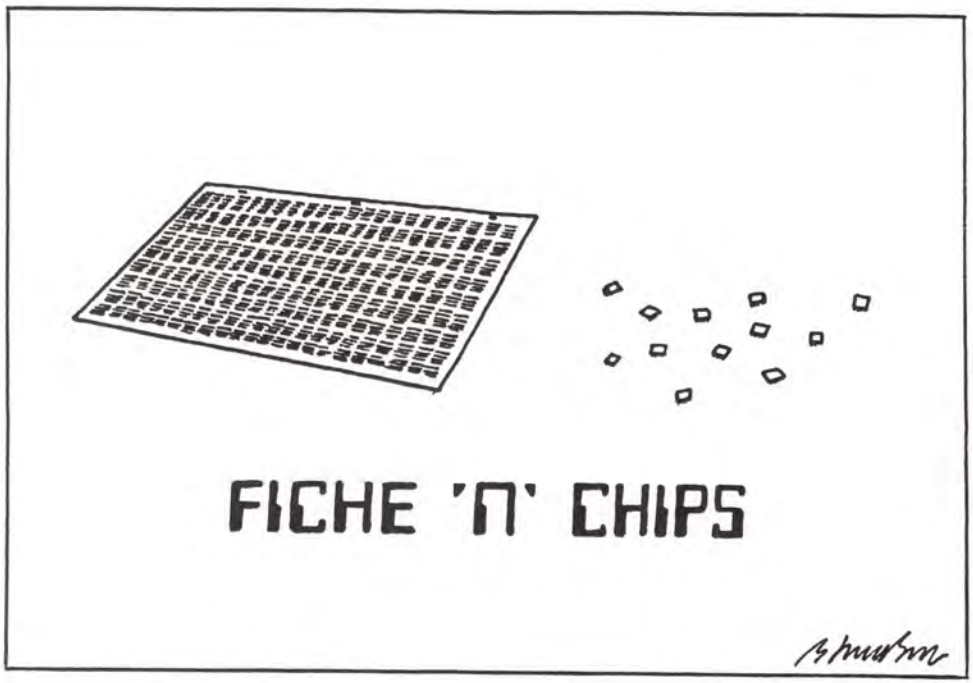
A>;The text was displayed as it was copied
A>;The copy operation is now complete
A>
```

The following example demonstrates the usefulness of the [E] parameter when copying several files to be concatenated into one file.

```
A>;Display contents of files in sequence
A>PIP B:FINAL.TXT=A:TEST1.TXT,TEST2.TXT[VE]
This is the text of the first file.
This is the text of the second file.
A>
```

This parameter may not prove to be too useful in some systems that utilize a video terminal as the console device. This is the case because the data being copied is often displayed too quickly for the console user to see, especially if the files being copied are small. This would not be a problem when a teletypewriter, or another type of printing device is used as the console device.

Of course, you would only need to [E] echo to the screen when you weren't already confident that the contents of the files to be concatenated were ready.



BEGIN COPYING A FILE AFTER A SPECIFIED STRING OF TEXT

[Sstring^Z]

The [S] parameter (short for "Start") instructs PIP to start a specified copy operation after it has encountered a specified string of text in the file being copied. The text existing *after* the specified string will be copied, but all preceding text will be ignored. This parameter is useful when you only want to copy a certain portion of a file.

This would be used for example when you want to copy only a later section of a document, a later piece of program code, etc.

The [S] parameter is entered at the end of the command string, by typing "[S" followed by the string of text which marks the beginning of the file section to be copied. The string parameter you enter is terminated by a CTRL/Z (achieved by simultaneously pressing the CTRL and the Z key) and, of course, followed by "]". For example, if you wanted to start the copying a file after the text string "CHAPTER 2", then the parameter sequence would look like "[SCHAPTER 2^Z]". [S] can be used with other parameters as long as all the specified parameters are related to text files.

The following example shows the results of using the [Sstring^Z] parameter.

```
A>TYPE MANUAL1.DOC
CHAPTER 1 (insert text at a later date)
CHAPTER 2 (insert text at a later date)
CHAPTER 3 (insert text at a later date)
CHAPTER 4 (insert text at a later date)

A>PIP B:CHAP-4.TXT=A:MANUAL1.DOC[VSCHAPTER 4^Z]

A>TYPE B:CHAP-4.TXT
CHAPTER 4 (insert text at a later date)

A>
```

The [S] parameter can also be used when copying several files to be concatenated into one file. This is accomplished by placing the parameter with the desired "start" string immediately after each source file, and right before the comma used to separate each source file. The following example shows how Chapter 3 of "1.DOC" and Chapter 4 of "2.DOC" are copied and concatenated into "3.DOC".

```
A>PIP B:3.DOC=A:1.DOC[SCHAPTER 3^Z],A:2.DOC[SCHAPTER 4^Z]

A>
```

QUIT COPYING A FILE AFTER A SPECIFIED STRING OF TEXT

[Qstring^Z]

The [Q] parameter (short for "Quit") is used to instruct PIP to terminate a specified copy operation AFTER it has encountered a specified string of text in the file. All the text existing *before* the specified string will be copied. All the remaining text will be ignored. This [Q] parameter is useful when a certain portion of a file is to be copied.

The parameter is entered at the end of the command string, by typing "[Q" followed by the desired string of text to mark the end of the file section to be copied. The parameter is terminated by a CTRL/Z (achieved by simultaneously pressing the CTRL and the Z key) and "]". For example, if you wanted to quit the copying a file after the text string "CHAPTER 2", then the parameter sequence would look like "[QCHAPTER 2^Z]". This parameter can be used with other parameters as long as all specified parameters are related to text files.

The following example shows the results of using the [Qstring^Z] parameter.

```
A>TYPE MANUAL1.DOC
CHAPTER 1 (insert text at a later date)
CHAPTER 2 (insert text at a later date)
CHAPTER 3 (insert text at a later date)
CHAPTER 4 (insert text at a later date)

A>PIP B:TEMP.TXT=A:MANUAL1.DOC[QCHAPTER 2^Z]

A>TYPE B:TEMP.TXT
CHAPTER 1 (insert text at a later date)
CHAPTER 2

A>
```

The [Q] parameter can also be used when copying several files to be concatenated into one file. This is accomplished by placing the parameter with the desired "quit" string immediately after each source file, and right before the comma used to separate each source file. The next example shows how Chapter 1 of "1.DOC" and Chapter 2 of "2.DOC" are copied and concatenated into "3.DOC".

```
A>PIP B:3.DOC=A:1.DOC[QCHAPTER 2^Z],A:2.DOC[QCHAPTER 3^Z]

A>
```

USING BOTH THE "START" AND THE "QUIT" PARAMETERS TO COPY SPECIFIED TEXT

[Sstring1^ZQstring2^Z]

The "start" [S] parameter can be used together with the "quit" [Q] parameter to instruct PIP to begin the copy operation at a specified point, and end it

at another specified point. To do this, both parameters are entered within a single pair of square brackets ("[ ]"). Type the "S" parameter first, followed by the "Q" parameter.

The following example shows how the copying of file "MANUAL1.DOC" starts with Chapter 2, and quits after Chapter 4.

```
A>TYPE MANUAL1.DOC
CHAPTER 1 (insert text at a later date)
CHAPTER 2 (insert text at a later date)
CHAPTER 3 (insert text at a later date)
CHAPTER 4 (insert text at a later date)
CHAPTER 5 (insert text at a later date)

A>PIP B:=A:MANUAL1.DOC[SCHAPTER 2^ZQCHAPTER 4^Z]

A>TYPE B:MANUAL1.DOC
CHAPTER 2 (insert text at a later date)
CHAPTER 3 (insert text at a later date)
CHAPTER 4 (insert text at a later date)

A>
```

The combination of "start" and "quit" parameters is often useful when you want to copy several files for concatenation into one file. For example, you may want to copy the last part of the first source file, and first part of the second source file, and concatenate them into one file. The "start" and "quit" parameters can be used to copy almost any combination of parts of various files.

CONVERT ALL UPPER-CASE CHARACTERS TO LOWER-CASE DURING A COPY OPERATION

[L]

The [L] parameter (short for "Lower") is used when all upper-case characters in a source file are to be coverted to lower case when copied into the

destination file. This parameter can be used only with text files, and can accompany other PIP parameters that are text-file-related. This [L] parameter is often useful when all characters in the source file are in upper case, and you want to create a copy of the file with standard upper and lower cases. It will not create upper and lower cases, but a text editor can later be used to provide initial capitals to beginnings of sentences, etc. The [L] parameter can thus be used in this manner, since it is easier to use a text editor to create initial caps throughout a file than to manually convert all characters to lower case. The [L] parameter is entered following the source file in the command line, as follows:

```
A>TYPE MANUAL1.DOC
CHAPTER 1 (insert text at a later date)
CHAPTER 2 (insert text at a later date)
CHAPTER 3 (insert text at a later date)
CHAPTER 4 (insert text at a later date)
CHAPTER 5 (insert text at a later date)

A>PIP B:LOWER.TXT=A:MANUAL1.DOC[L]

A>TYPE B:LOWER.TXT
chapter 1 (insert text at a later date)
chapter 2 (insert text at a later date)
chapter 3 (insert text at a later date)
chapter 4 (insert text at a later date)
chapter 5 (insert text at a later date)

A>
```

Note that the use of the parameter is not restricted to the copying of files. It can also be used when copying between standard CP/M or special PIP devices. For example, you could convert text obtained from the source device RDR (paper-tape reader) while it's being transferred to the destination device LST. The [L] parameter can be used with almost any combination of source and destination devices or files which allow the transfer of text.

```
A>PIP LST:=RDR:[L]

A>PIP LST:=A:UPPER.TXT[L]
```

CONVERT ALL LOWER-CASE CHARACTERS TO UPPER-CASE DURING A COPY OPERATION

[U]

The [U] parameter (short for "Upper") is used only with text files and can accompany other PIP parameters that text-file-related. It is employed when all lower-case characters in a source file need to be coverted to upper case when copied into the destination file. This parameter is often useful when the characters in the source file are lower case or mixed (upper and lower) cases, and you want to create a copy of the file with only upper-case characters, such as text files containing source code for programs, like ".ASM" (Assembly program) files. These can contain either all lower-case or mixed-case characters, and [U] can be used when a copy with "official" upper case only characters must be made. Whether a file is to be all lower case, upper case, or mixed cases, is usually a matter of personal preference or convenience.

The [U] parameter is entered following the source file in the command line, as follows:

```
A>TYPE LOWER.TXT
chapter 1 (insert text at a later date)
chapter 2 (insert text at a later date)
chapter 3 (insert text at a later date)
chapter 4 (insert text at a later date)
chapter 5 (insert text at a later date)

A>PIP B:UPPER.TXT=A:LOWER.TXT[U]

A>TYPE B:UPPER.TXT
CHAPTER 1 (INSERT TEXT AT A LATER DATE)
CHAPTER 2 (INSERT TEXT AT A LATER DATE)
CHAPTER 3 (INSERT TEXT AT A LATER DATE)
CHAPTER 4 (INSERT TEXT AT A LATER DATE)
CHAPTER 5 (INSERT TEXT AT A LATER DATE)

A>
```

Note that the use of the [U] parameter is not restricted to the copying of files. It can be used when copying between standard CP/M or special PIP devices, as well. For example, you could convert text obtained from the source device RDR (paper-tape reader) while being transferred to the destination device LST. This parameter can be used with almost any combination of source and destination devices or files which allow the transferring of text.

```
A>PIP LST:=RDR:[U]

A>PIP LST:=A:LOWER.TXT[U]
```

DELETE ALL CHARACTERS ON EACH LINE OF TEXT AFTER A SPECIFIED COLUMN

[Dn]

The [D] parameter (short for "**D**elete") is used to delete or remove all characters in every line of text that exist past a specified vertical column. This ability is sometimes useful when copying text data to a destination device with fewer columns than the width of the average line of text. (For example, when using a 40-column video terminal or printer, or when copying 132-column text to an 80-column device.) If, for example, it is specified to delete all characters beyond 80 columns from 132-column lines, all characters from the 1st column to the 80th column, inclusive, will be output. The characters from the 81st column to the 132nd column, however, will be deleted. The [D] parameter is rarely used when the specified destination is a file.

The [D] parameter is entered by typing "[D" followed immediately by a value ("n"). The value "n" may consist of any number from 1 through 255. If the source text lines are longer than 255 columns, character deletion will occur properly. The parameter is terminated by typing "]".

The next example shows how the text lines being transferred from a 60-column text file are prepared for a 40-column terminal console device.

```
A>TYPE 60-COL.TXT
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890

A>PIP CON:=A:60-COL.TXT[D40]
         1         2         3         4
1234567890123456789012345678901234567890

A>
```

CONVERT ASCII TAB CHARACTERS TO A SPECIFIED NUMBER OF SPACES

[Tn]

The [T] parameter (short for "Tab") is used when it is required to equalize or expand ASCII Horizontal Tab (HT) or ^I (CTRL/I) characters with the use of spaces. When text files are created using some text editors or word processors, ASCII horizontal tab characters are not always saved as spaces. Since some terminals and printers cannot properly recognize ASCII HT or ^I characters, the [T] parameter can be used to convert all occurrences of ^I (which is the tab character) to a specified number of spaces or vertical columns. (In this way, the computer continues printing text at the intended tab stop.) This conversion process will only work with proper ASCII ^I characters. If spaces in the text were created by the Tab Command of a text editor or word processor, they will be ignored by the [T] parameter.

The [T] parameter is used by first entering "[T" followed immediately by the desired value "n". The "n" value can actually be anything, though a value in excess of 20 would be unusual. The parameter is terminated by typing "]".

```
A>TYPE PROG-1.BAS
0010^IREM HOME-BREW BASIC PROGRAM 1
0020^IREM THIS IS A TEST PROGRAM LISTING
0030^IREM TO SEE HOW TABS DISPLAY
```

```
A>PIP PROG-1B.BAS=PROG-1.BAS[T8]

A>TYPE PROG-1B.BAS
0010    REM HOME-BREW BASIC PROGRAM 1
0020    REM THIS IS A TEST PROGRAM LISTING
0030    REM TO SEE HOW TABS DISPLAY

A>
```

Note in the preceding example, that even though ASCII tabs had been specified, the text following each tab was not "tabbed" (indented) as it should have been. Assuming that the tab stops in this example had a default value of 8 spaces, all the text beginning with "REM" should start at the 8th column. If the terminal to which this file was displayed, however, was not capable of correctly interpreting the tabs, then the [T] parameter could be used to alleviate the problem. You would create a new copy of the file with PIP and the T parameter and then send the new expanded tab file to the printer or terminal.

DELETE ALL FORM-FEED CHARACTERS DURING A COPY OPERATION

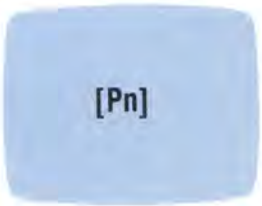


The [F] parameter (short for "**F**orm-Feed") is used during a copy operation when you want to remove all ASCII Form-Feed (FF or ^L) characters. It is often undesirable to have these characters in a file that will rarely be printed and will only be used for storage. Form-feed characters are used in text files as page breaks or "page ejects" on formatted output from line printing devices. The [F] parameter may also be used when the file is to be rearranged with a text editor, text formatter, or word processor that uses special page break commands (such as WordStar® Micropro, Inc.). In this case, the presence of form-feed characters would create problems. [F] will delete all valid ASCII form-feed characters encountered during a text transfer operation.

The [F] parameter is entered following the source file or device in the PIP command sequence. It may accompany other PIP parameters that pertain to the transfer of text.

Note that this parameter can be used in conjunction with the "[Pn]" parameter (insert form-feeds) as described next. In this way, you can delete existing form-feeds and insert new ones at specified locations. When both parameters are specified, the [F] parameter is entered before the [Pn] parameter, and both are enclosed within a single pair of square brackets, i.e., [FPn].

INSERT FORM-FEED CHARACTERS AFTER EACH SPECIFIED NUMBER OF LINES DURING A COPY OPERATION



The [P] parameter (short for "**P**agination") is used during a copy operation to insert form-feed characters (page breaks), after each specified number of text lines. This parameter is most often used when a line printer is a destination device. The parameter is entered at the end of the source file name, starting with "[P" followed a value (n). The value "n" may be any number ranging from 1 to 255, though the number specified should be within the limits of the total length of each piece of paper on the printing device. The specified value will be effective for all the text output to the destination device. If you specify a value greater than the total number of lines capable of being printed on a page, the results may be rather undesirable. If no number is specified, or if the specified number is a 1, PIP will use a default value of 60. For example, if the printing device is capable of printing 66 lines of text on a page, then a value of 60 could be specified with the [P] parameter. In that case, only 60 lines would be printed before a page break was made. The [Pn] parameter is terminated with a "]".

The [P] parameter should not be used if the destination device consists of a printer which automatically initiates blank top and bottom margins. The total of these two margins is usually 6 lines. These types of devices will print text only in the space between the two margins. Therefore, n = 60 would result in only 54 lines of printed text per page.

Note that this parameter can be used in conjunction with the "[F]" parameter (delete existing form-feeds), as described previously. In this way, they work together to delete existing form-feeds and insert new ones at specified locations. When both parameters are specified, the [F] parameter is entered before the [Pn] parameter, and both are enclosed within a single pair of square brackets, i.e., [FPn].

ADD SEQUENTIAL LINE NUMBERS TO TEXT DURING A COPY OPERATION

[N]
[N2]

The [N] and [N2] parameters (abbreviations for "Number") are used to add sequential line numbers to text during a copy operation. The [N] parameter, causes each line of text to be moved eight spaces to the right. These spaces are used to contain the line numbers, with the single digits placed at the seventh column. As the numbers are incremented, the tens, hundreds, and thousands digits are placed in order to the left. Colons (":") are placed in the eighth column for each line, and the text will begin at the ninth column. Note that the eight spaces used to move the text to the right will increase the total width of the text by eight characters. This may be undesirable, especially if the total width of a destination printer device is less than that of the output text. The following example shows the results of using the [N] parameter.

```
A>TYPE MANUAL1.TXT
First Line
    *
Tenth Line
    *
Hundredth Line
    *
Thousandth Line

A>PIP MANUAL2.TXT=MANUAL1.TXT[N]

A>TYPE MANUAL2.TXT
      1:First Line
                *
     10:Tenth Line
                *
    100:Hundredth Line
                *
   1000:Thousandth Line

A>
```

These line numbers may be useful when you are editing a version of a text document and wish to refer to it by line number. For example, you could be editing a document and you could refer to lines 100-105, 2078, etc.

The [N2] parameter can be used instead of the [N] parameter when all line numbers must have leading zeros, i.e., zeros in the spaces preceding the real numbers. When this [N2] option is used, the colon (":") is replaced by a space. The results of using [N2] are shown in the following example.

```
A>TYPE MANUAL1.TXT
First Line
    *
Tenth Line
    *
Hundredth Line
    *
Thousandth Line

A>PIP MANUAL2.TXT=MANUAL1.TXT[N2]

A>TYPE MANUAL2.TXT
0000001 First Line
          *
0000010 Tenth Line
          *
0000100 Hundredth Line
          *
0001000 Thousandth Line

A>
```

COPY FROM A SOURCE "SYSTEM" FILE (CP/M Version 2.0 and newer only)



Files are set to either "system" (SYS) or "directory" (DIR (nonsystem)) with the STAT transient command. Normally, PIP will not reference and copy a system file that has been specified as the source file. The [R] parameter (short for "**R**ead") is provided only in CP/M Versions 2.0 and above, to allow PIP to copy from a source file that has been set to the "system" status. The [R] parameter is thus used to ignore this file attribute when searching for a file on the source device.

This option might be used when you wish to copy a group of protected system files, such as all the CP/M utilities, overlay files, special modules, etc.

ERASE A READ-ONLY FILE HAVING THE SAME NAME AS THE DESTINATION FILE NAME (CP/M Version 2.0 and newer only)



The [W] parameter is short for "**W**rite". You can use it when you know that a read-only (R/O) file exists on the destination device by the same name as the file name specified in the destination field of a PIP command line. Read-only files are those that have been write-protected (i.e., nonerasable and nonmodifiable). They have been assigned their R/O attribute with the STAT transient command. Normally, when PIP is used in such a circumstance, the program will ask whether you want the existing read-only file on the destination device to be erased and replaced with the file specified in the destination field of the command. To avoid these program queries, the [W] parameter can be specified so that these files are automatically over-written.

You should, however, use this [W] parameter only when you are *absolutely certain* that this is what you want done. The read-only file protection feature and the normal PIP prompt for determining whether files should be erased are

valuable characteristics of the CP/M operating system. They can help prevent the inadvertent loss of valuable files and should not be discarded thoughtlessly.

COPY FILES FROM OTHER DISK USER AREAS (CP/M Version 2.0 and newer only)



The [G] parameter is used in situations which require the copying of files from one User area to another. When the [G] parameter is used, it applies only to User areas corresponding to the source file(s). Essentially this means that PIP can be used only for copying *from* other User areas. It cannot be used to copy *to* a User area.

The parameter is entered after the source file name, starting with "[G", followed by the User number ("n") and "]". Any User number from 0 through 15 can be specified. The next example shows various typical PIP command lines that may be used for copying from other User areas.

```
A>;Copy from A: User 0 to current A: User
A>PIP A:=A:STAT.COM[G0]

A>;Copy from B: User 7 to current A: User
A>PIP B:=A:SYSGEN.COM[G7]

A>
```

This option will allow you to borrow files from other users on your disk, provided you know their filenames.

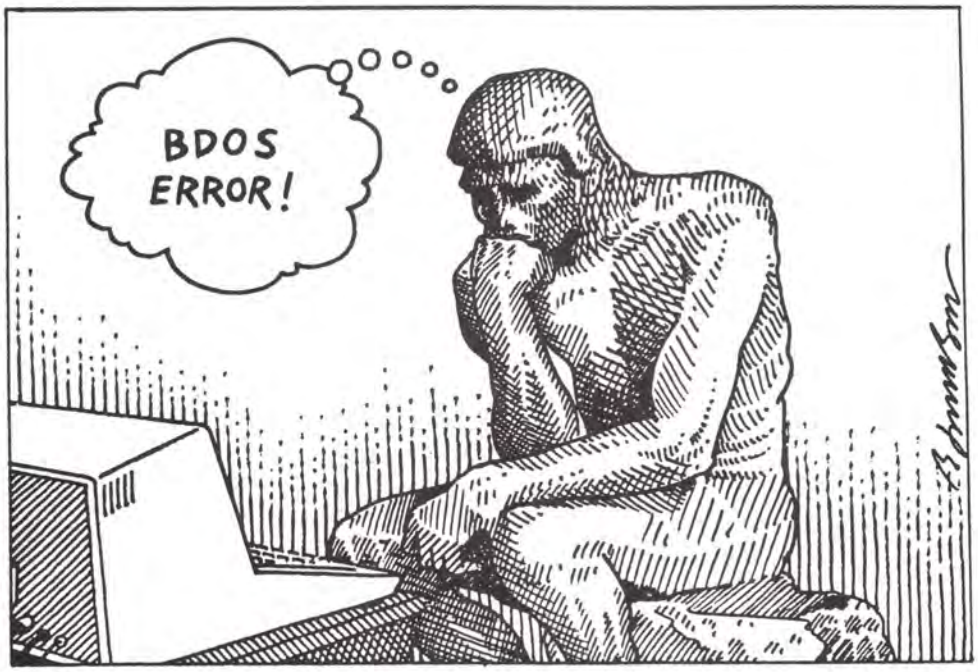
INTERPRET FILES BEING COPIED AS NON-ASCII OBJECT/MACHINE CODE FILES



The [O] (short for "**O**bject") parameter is used only with source files which contain object or machine code (i.e., binary data) rather than ASCII text. The [O] parameter instructs the PIP program to ignore all occurrences of CTRL/Z (ASCII SUB) control characters that might be contained in a binary file being copied. Sometimes these types of files contain one or several CTRL/Z control characters as valid control sequences. Because CTRL/Z is used by CP/M as the end-of-file marker in all files, the PIP program is designed to terminate a copy operation when it encounters a CTRL/Z. The PIP program must, therefore, be instructed to ignore any CTRL/Z control sequences encountered during a copy operation. Otherwise, PIP might detect a CTRL/Z control sequence in the middle of a file, and terminate the file copying procedure prematurely.

NOTE: This parameter need not be entered with ".COM" files (transient commands), since PIP automatically performs the function with these files. Non-.COM object or machine code files should be accompanied by the [O] option.

The following example shows how this parameter is added to a typical PIP copy command sequence. The first command string shows the use of the [O] parameter by itself. On the next line, the same command sequence is entered



with both the [O] and the [V] (Verify) parameter. Both parameters can be entered in any order.

```
A>PIP A:=B:PGM-01.BIN[O]
A>PIP A:=B:PGM-01.BIN[VO]
A>PIP A:ALL.BIN=B:1.BIN[VO],2.BIN

A>
```

SPECIFY "BLOCK MODE" FORMAT DURING A COPY OPERATION

[B]

The [B] parameter (short for "**B**lock Mode") specifies that data from the source be copied into a buffer (temporary storage area) until the ASCII character DC3 (^S, sometimes called "X-OFF") is found in the file. Once the character is received, data transfer from the source stops, and the data in the buffer is transferred to the destination device. Once the contents of the buffer has been copied to the destination device, PIP resumes with the retrieval of the next "block" of data from the source. The process then repeats itself. The [B] parameter is used when copying from devices, such as paper-tape reader machines, which continuously send data. [B] is needed because PIP is not capable of filling and emptying the buffer simultaneously during a continuous flow of data from the source device. However, the data from the source must have the ASCII DC3 (X-OFF) characters correctly placed throughout so as to correspond to the size of the buffer provided by PIP. This parameter need never be used for file-to-file copying.

SPECIFY INTEL HEXADECIMAL FORMAT DURING A DATA TRANSFER OPERATION

[H]

The [H] parameter (short for "**H**exadecimal") specifies that the data to be retrieved from the source device (or file) is in the special Intel Hexadecimal Format. This format is usually used only with paper tape punch and reader machines. When the [H] parameter is specified, checks are made to verify accuracy in the Hexadecimal format. If errors are detected, PIP will display messages for corrective action on the console device.

IGNORE ANY NULL RECORDS IN INTEL HEXADECIMAL FORMAT DURING A COPY OPERATION



The [I] parameter (short for "**I**gnore") instructs PIP not to copy any null records (":00") during an Intel Hexadecimal Format data transfer operation. This parameter should only be used if the [H] parameter (described previously) is also specified.

SET THE PARITY BIT OF ALL ASCII BYTES TO ZERO DURING A COPY OPERATION



The [Z] parameter (short for "**Z**ero") is used during a copy operation. It specifies that the parity bit (8th bit) in all ASCII bytes must be set to zero ("0"). In essence, the use of the [Z] parameter results in converting 8-bit ASCII bytes with parity to 7-bit ASCII bytes without parity. [Z] is often useful for this conversion procedure, as files with 8-bit ASCII bytes sometimes demonstrate some peculiarities when viewed or operated under the CP/M operating system.

Chapter 10



CP/M Versions:

1.3 **1.4** **2.0** **2.2**

Command Type:

Transient

SUMMARY

ED, (short for **ED**itor) is a CP/M transient command program. It provides a simple way for the CP/M user to create text, document or program files, view them and make any necessary modifications. Unlike some editing programs which are full-screen editors (i.e., full use is made of screen cursor position and attributes), ED is a line-orientated text editor. This means that it references and manipulates the contents of text files on a line-by-line basis. ED differs, however, from many other line-orientated text editors. Individual characters or words can be inserted, deleted, or modified, rather than requiring that a complete line of text be re-typed to make one or several changes.

ED has features which allow you to:

- Create a new file by entering ED and specifying the file name.
- Enter or open an existing file by appending one, several, or all lines of text to the main editing memory area.
- Temporarily open an auxiliary file for editing, and return to original file when finished.
- Move the character pointer (place where new text can be inserted, and existing text can be viewed, copied, moved or deleted). The character pointer can be moved to the beginning (first line) or the end (last line) of a file. In addition, it can be moved to any specified line, or to any word or character on a specific line.

- View specific lines (or all lines of text in the file being edited) that have been appended to the main editing memory area.
- Insert text starting either at the beginning of a specified line, or before a specified word or character on the line.
- Specify that all inserted text be in upper case, or permit all inserted text to be in lower and upper case as it's entered on the console device.
- Delete a specified character or characters, word or words on a specific line. It also enables the deletion of a specified line or number of lines.
- Find a specific character string in the file, and position the cursor at the beginning of the string.
- Find the next occurrence of a previously defined character string.
- Substitute one or several occurrences of a specific string of text with another specified string of text.
- Include one or several lines of text from another text file into the file being edited.
- Copy one or several lines to a new location within the file being edited.
- Copy a specified number of lines from the main text file to a temporary auxiliary file. This temporary file can be appended to another auxiliary file.
- Move one or several lines to another location within the file being edited (block move).
- Define a simple "macro" that causes the repetition of a user entered command string a specified number of times.
- Execute the "Sleep" function which delays the execution of a command for a specified amount of time.
- Determine the total size and the number of free bytes in the main editing memory area.
- Exit from ED to return to CP/M and save edited text from the text buffer into an output file. Or exit without saving the edited text (the original back-up file will not be erased).

The ED program provides many variations in the way the preceding features can be used.

The following section deals with the basic concepts of the operation of ED. The subsequent section will describe each ED function in depth.

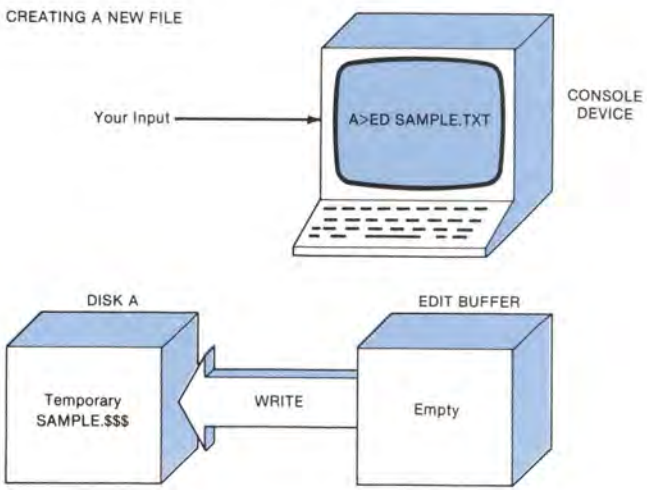
GENERAL DESCRIPTION AND PROGRAM ACCESS

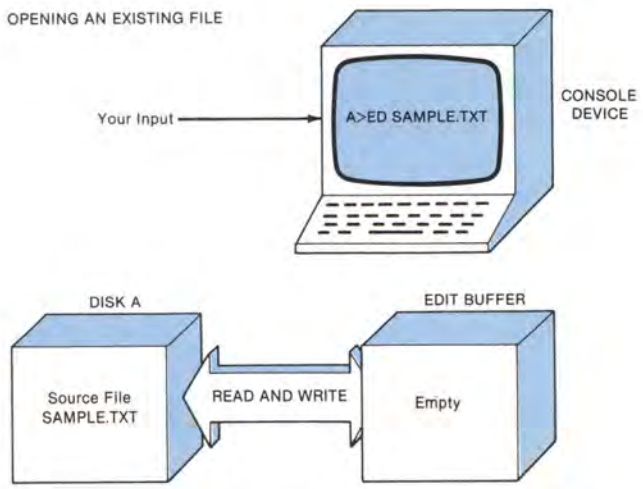
When using ED to create and manipulate text files, it is important to understand the basic concepts behind its operation in the CP/M operating

system environment. To begin, the transient command file, "ED.COM" must be on a disk inserted in one of the computer's drives. Preferably, ED.COM should be on the disk in the system drive (Drive A). The text file you wish to create or manipulate can be located either on the same or on a different disk. To access ED, type in "ED", and a space followed by the file name, in response to the logged drive prompt (A>, B>, etc.). If the specified file is new (not already stored on the disk) then ED will create that file. If the file already exists, then it will be opened, i.e., it will be placed in a state which permits access to its contents. You can only open files which contain displayable text. In most cases, text files will have an extension or file type of ".TXT", ".DOC", or something descriptive of a text file. (An error will occur if you try to open a machine or object code file, such as a ".COM" file.)

When ED has been accessed and loaded into the computer's memory, a certain portion of the memory is allocated for editing purposes. This area is called the "Editing Buffer." It's the place where ED will follow your instructions and perform all desired editing functions. If, for example, you are creating a new file, all entered text will be inserted into the text buffer. If you are editing an existing file, ED can be instructed to load several or all lines of text into the editing buffer. When all the text insertion and manipulation has been completed, you might want to create the latest version of the file. At this point, ED will save the contents of the editing buffer into the specified file on the disk media and return you to CP/M.

The following illustrations will help you to understand how the environment in which ED will operate is arranged.





INITIATING ED: To create a new file or edit an existing file, specify the file name on the same command line as "ED", in response to the logged CP/M prompt (A>, B>, etc.). If you want to create a file and store it on a disk other than the one containing ED.COM, then the drive name (B, C, D, etc.) can precede the file name. This can also be done if the specified file already exists on another disk.

The next illustration shows the correct syntax for the command line. It also provides a basic outline of the status of the file specified and the editing buffer.

EDITING A FILE: Once the desired file name has been specified in the command line, and the RETURN key has been pressed, the ED program will be loaded into RAM and the editing buffer will be established. ED will scan the disk (the default or specified disk) to determine whether the file already exists. If it doesn't, the message "NEW FILE" followed by the ED command prompt, ":*", will be displayed on the console. If the file does exist, then the ED command prompt will immediately be displayed.

The following examples show these two possibilities:

```
;When the file does not exist

A>ED SAMPLE.TXT
NEW FILE
    : *
```

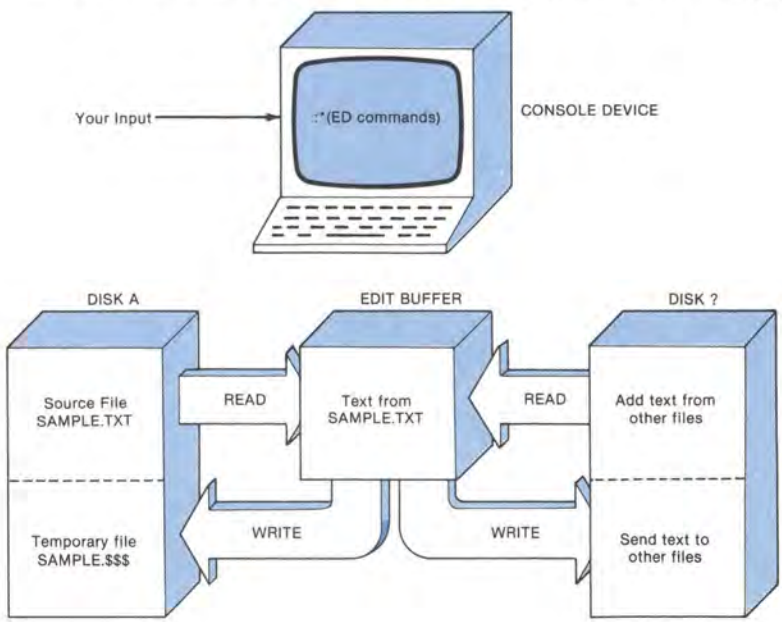
```
;if the file already exists

A>ED SAMPLE.TXT
    : *
```

Once the ED command prompt is displayed (as just shown), the program has been loaded into memory and the editing buffer is ready to accept text. If a new file was specified, then issue the insert command (insert is covered in detail later). This enables you to place text into the editing buffer, which can later be saved in the specified file. If the specified file already exists, and you wish to edit it now, issue the append command. This will copy a specified number of lines from the file into the editing buffer. While text is in the editing buffer, it can be manipulated with various commands. In this command mode, certain text can be appended or included into the editing buffer from the file of your choice and added to the original file. A number of lines copied from the editing buffer into a separate, temporary file can also be specified. During this process, the integrity of the original file remains intact. ED will create a temporary output file with extension .$$$ which will be used to contain the final edited text.

The next illustration shows the basic concepts that we've just discussed.

TERMINATING ED AND SAVING THE FILE: Once all the required text manipulation has been completed in the editing buffer, you normally give the

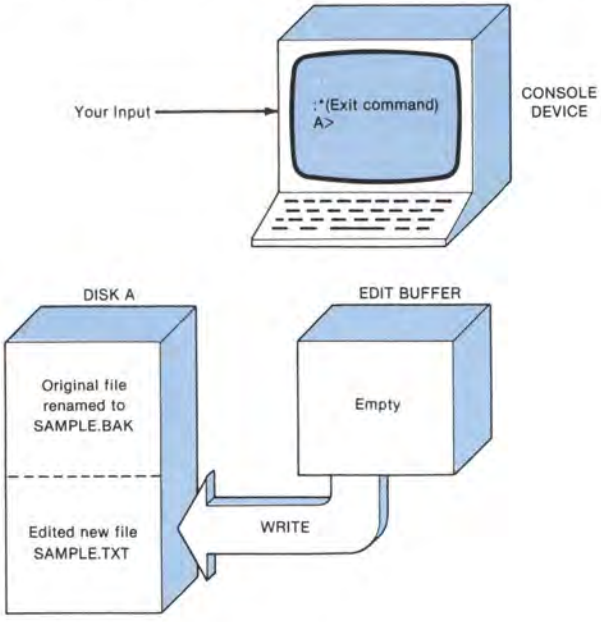


exit command. This will instruct ED to save the contents of the buffer into the specified file. When this is done, the original file (if it existed previously) will be renamed with the extension or file type of ".BAK" to indicate that it is the *backup* file.

If you decide *not* to save the contents of the editing buffer, then issue the quit command. This command causes the temporary output file to be erased, leaving the original file untouched.

After terminating ED, either by the exit or quit command, the original logged CP/M prompt (A>, B>, etc.) will reappear. At this point, you can use other CP/M commands, such as DIR or TYPE, to verify the existence of the edited file.

As you become more familiar with the ED commands, the layout in the illustrations will make more sense. If you are using ED for the first time, it's a good idea to create a sample file for experimenting with all the commands presented in this section. Those commands most frequently used are described first. They are followed by those commands used less frequently.



THE ED CHARACTER POINTER (CP)

The ED program provides a unique feature called the "Character Pointer". It allows the user to move to a certain position in any column location on any

line of text. The Character Pointer is abbreviated as "CP" in most CP/M manuals and CP/M-related documentation and books (including this one).

The CP is not visible to the user. It is a point of reference used by the ED program to determine from "where" in a line of text the effects of a subsequent command will be initiated. The CP can be positioned by the careful use of ED commands. Subsequent commands cause ED to reference the current CP position, until another command, which changes the position, is issued. The CP permits specific characters in a line of text to be inserted, modified, or deleted. Other line-oriented text editors, on the other hand, require the retyping of a complete line of text to make a change.

When the ED program is first initiated, the CP is at the first character position of the first line in the editing buffer. As subsequent lines of text are either inserted or displayed in the editing buffer, the CP always moves to the first character position of the current line. By this we mean the most extreme left column in the line. The CP will always be moved like this unless a specific command is issued instructing ED to position it elsewhere. The following example displays the next line, then moves the CP ahead 16 characters. The "^" up arrow or carat character indicates the CP location. This would not appear on your screen.

^ = CP

```
1: * First line of text.
1: * ^

2: * Second line of text.
2: * ^

2: * Second line of text.
2: * +16C              ^
(Move CP 16 character positions forward)
```

```
2: * Second line of text.
2: * Inew ^Z           ^
(Insert "new " at CP)

2: * Second line of new text.
2: * -3C            ^
(Move CP 3 character positions back)
```

There are several ED commands which are used to position the CP. These commands cause the CP to move back and forth along a line of text, to other

lines of text, and to the beginning or end of the editing buffer. The remaining part of this section provides descriptions of each individual ED command. Commands that affect, or are affected by, the position of the CP contain more detailed information than those on CP manipulation.

DESCRIPTION OF ED COMMANDS

APPENDING SPECIFIC LINES OF TEXT FROM SOURCE FILE INTO THE EDITING BUFFER



When an ED command sequence specifying an existing file has been initiated, the contents of the file cannot be seen on the console device until they have been "appended" or copied into the editing buffer. Once in the editing buffer, text can be displayed and manipulated. The A (Append) command is used when ED is referencing an existing file on a disk. It is entered in response to the ED command prompt, and consists of a number and the character "A", followed by pressing the RETURN key. The number preceding "A" specifies the number of lines of text to be appended or copied. Any number from 1 through 65,535 can be specified.

The next example shows how the first line of text in the source file is appended to the editing buffer. The command can consist of "1A" or simply "A", as the number "1" is always the default value when a number or symbol is absent.

```
A>ED SAMPLE.TXT
     : *1A
    1: * This is the first line of text
    1: *[]
```

OR

```
A>ED SAMPLE.TXT
   : *A
  1: * This is the first line of text
  1: *[]
```

Note: [] = cursor

The above example shows only one line specified for appending to the editing buffer.

When the append command is first used during an edit session, the appending process will always start with the first line of text in the source file. Once the line has been appended, the line number in the editing buffer is added to the ED prompt. When the RETURN key is pressed, the text associated with that line number is displayed following the ED prompt. In addition, the cursor rests on the next line which consists of the same prompt and line number. At this point, another ED command associated with line number 1 can be entered. Similarly, more text can be appended from the source file following the first line. When successive append commands are entered, the RETURN key can be pressed to display or print the appended lines of text. In the case of successive commands, the ED program will remember which lines have already been appended. For example, if the command "2A" was entered after "1A", then ED will append the next two lines after the first one, i.e., the second and third lines.

```
 : * 1A
1: * This is the first line of text.
1: * <CR>
 : * 2A
2: * This is the second line of text.
2: * <CR>
3: * This is the third line of text.
3: * <CR>
 : *
```


Any combination of append commands can be entered during an edit session. For example, you could append several lines of text from the source file, and then insert text entered on the console device after the last appended line. More A commands would cause the text to be appended *after* the inserted text. ED will automatically number all lines of text, whether they be

appended or inserted, according to the order in which they occur in the editing buffer.

APPENDING PORTIONS OR ALL OF TEXT FROM SOURCE FILE UNTIL THE EDITING BUFFER IS HALF OR COMPLETELY FULL



Generally, when appending lines of text from the source file, it is desired either to append most, all, or as many lines of text as can be supported by the editing buffer. The maximum number of lines supported by ED is 65,535. This is frequently the case when you plan to make modifications to sections throughout the text. By bringing all the lines of the text file into the buffer, you can access them all.

The "0" (zero) character can be used with the append command (i.e., "0A") instead of a specific number (i.e., "nA"). The 0A causes lines of text from the source file to be appended continuously until either all lines in the file have been appended or the editing buffer is half full. Alternately, the "#" character can be specified to cause *all* lines of text to be appended. In this case, the process continues until all lines have been copied, 65,535 lines have been reached, or the editing buffer is completely full. The size of the editing buffer depends on the size of the computer's RAM. Its total capacity determines the number of inserted or appended characters (bytes) allowed.

The following example shows the results of the "#A" command when the RETURN key is repeatedly pressed to verify that the entire text from the source file has been copied into the editing buffer.

```
   * : #A
  1* : This is the first line of text.
  1* : <cr>
     :
     V
500* : This is the last line of text.
500* : <cr>
   * :
```

In this example, the user types the return key <cr> consecutively for viewing every line of text. Once all the desired text has been appended to the editing buffer, you can use other ED commands to delete unwanted text, modify existing text, etc. The append command qualified by specific line numbers can be issued in combination with almost all other ED commands, in any order. ED will always keep track of which lines have already been appended. In this way, when further append commands are given, only those lines which have not yet been appended will be copied into the editing buffer.

INSERTING TEXT INTO THE EDITING BUFFER



When a file is first created with ED, it is desirable to initiate a command which will enable you to insert text into the editing buffer in between lines and columns. The "I" (Insert) command can be used at any time while the cursor is positioned at the ED command prompt. Normally, the insert command is entered in the form of the single character "I" followed by the RETURN key. (It may also be entered as a single command, or in conjunction with a short string of text that is to be inserted.) When the command is entered in this solo manner without the command string to insert, the ED prompt on the next line changes to single asterisk character "*" instead of the normal colon ":", indicating that you are in insert mode. Any text that you type from the keyboard at that point will be inserted in the editing buffer, until a CTRL/Z (achieved by simultaneously pressing the CTRL and the Z keys) command is typed. The CTRL/Z command will instruct ED to consider all text entered on the console as inserted text, whereupon the ED command prompt will reappear.

The following example shows how the insert command is used to insert text into the editing buffer to create a new file.

(^Z) = CTRL/Z Command

```
A>ED SAMPLE.TXT
NEW FILE
  : *I
 1:  This is the point from which new text is entered,
 2:  when creating a new file. As new lines of text are
 3:  inserted, ED will automatically assign line numbers
 4:  for you. A "line" with a "line number" is a line of
 5:  text which is terminated by pressing the RETURN key.
 6:  The RETURN key will cause the ASCII Carriage Return
 7:  (CR) and Line Feed (LF) characters to be inserted in
 8:  the editing buffer as the final characters of the line.
 9:  The CR/LF characters will be invisible when a line of
10:  text is displayed on the screen.
11:  ^Z
  : *
```

As just seen, the ED program automatically assigns a number to each line of inserted text.

ED interprets the pressing of the RETURN key as being the ASCII CR (Carriage Return) and the LF (Line Feed) characters. These two characters (and other ED commands) can be used to move the cursor and the CP (Character Pointer) to the first character position of the buffer's next line. This is where a new line number will be assigned. Text can be inserted in this manner until the CTRL/Z command is given. CTRL/Z terminates the insert mode and causes the ED command prompt (:) to reappear. When this point is reached, the CP is positioned on the line *immediately following the last line of inserted text.*

In insert mode, most of the CP/M control character editing commands (including ^X, ^U, ^Y, etc., as described in Chapter 2) will operate as normal. These commands can be used to modify text as it is being entered (typed in). (The insert command, and its various forms, can be used in almost any combination with other ED commands, as well.) The following descriptions provide information about each of the commands you can use *while in insert mode.*

CTRL/H or BACKSPACE: This command will cause the character to the left of the cursor to be erased or deleted.

RUBOUT or DELETE: This command will cause the character to the left of the cursor to be erased. When this is employed, the deleted character

will be redisplayed or echoed to the console device. (This will only be the case in a standard CP/M configuration. Some distributors of CP/M have modified the RUBOUT or DELETE key so that it performs like a CTRL/H command or the BACKSPACE key.)

CTRL/M or RETURN: This command instructs ED to insert the ASCII characters CR (Carriage Return) and LF (Line Feed). These characters cause the current line to terminate, and the next line to commence.

CTRL/R: Redisplays or reprints the current line in the same manner as the CP/M prompt. This command is available in CP/M Versions 1.4 and newer only.

CTRL/U and CTRL/X: Either of these commands can be used to delete a line of text entered on the console device. Both these commands operate in basically the same manner as when used in the main CP/M command mode.

CTRL/L: This command can be used to insert an ASCII FF (Form Feed) character in the text. A form-feed character will cause a page break when the file is printed.

Most other ASCII control sequences can be entered while you are in the insert mode. The use of control sequences other than those described above is rare. It is useful, however, to know which control sequences can be entered, should they be required.

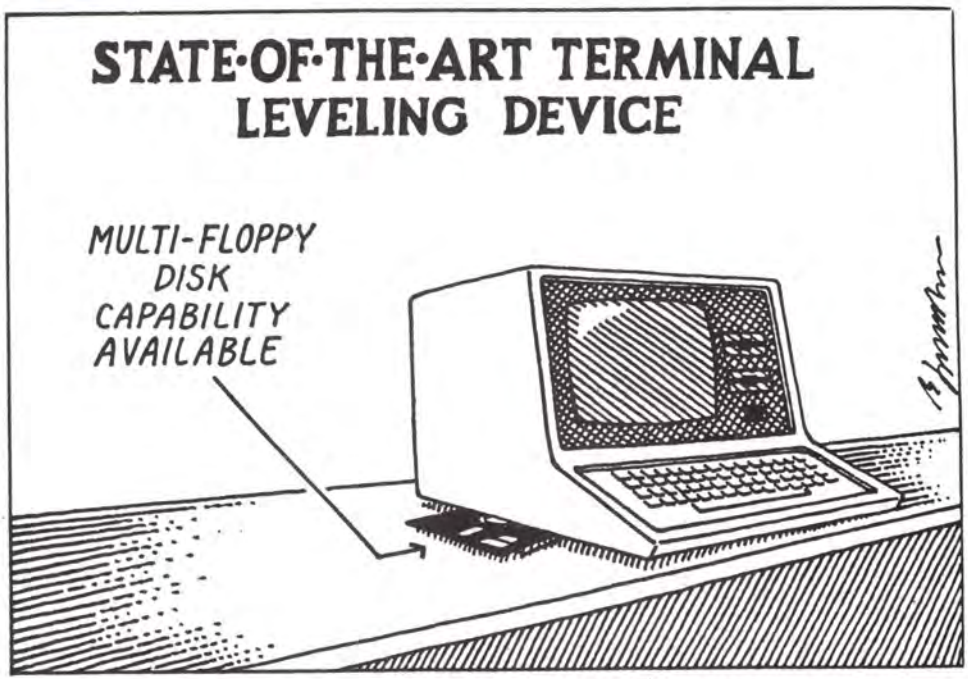
The insert command can also be used to insert a specific line of text. This is accomplished by typing in the text on the same command line as the "I" command. When this is done, ED interprets all text between the "I" command and the point where the RETURN key is pressed as a single new line of text. This form of the command is a quick way to insert text. All text typed in the command line will be placed on the same line, beginning at the next position of the CP. It is a good idea, therefore, to make sure that the text you type in does not exceed the length of the screen or paper of the console device when it is listed later.

The following example compares this form of the command (i.e., text following "I"), and normal use of the insert command (text terminated with a CTRL/Z, as described previously.

```
(^Z) = CTRL/Z Command
            : *I
            1:  This is the first line of text.
            2:  ^Z
            2: *IThis is the second line of text.
            2: *
```

Once the text has been inserted, you can return to the beginning of the editing buffer and, by repeatedly pressing the RETURN key, display the newly entered lines of text for verification.

```
1: *<cr>
1: * This is the first line of text.
1: *<cr>
2: * This is the second line of text.
2: *
```



The point in the editing buffer at which text is inserted depends on the position of the CP. When a file is first created, or when an existing file is opened, the CP position defaults to the first character position of the first line.

As the RETURN key is pressed to move down to the next line, the CP will move with it, and will be placed at the first character position of the current line. Because the insert command references the position of the CP, text can be inserted starting at almost any point along a line of existing text. By using certain ED commands which move the CP along a line or to other lines of text, one or several characters or words can be inserted at any designated point.

Another form of the I command consists of having the text typed in immediately following "I" and followed by a CTRL/Z on the same command line. This form of the command is useful for inserting text when the CP is positioned at a designated place along a line. For example, you may have missed a word in a given line of text. A command could be given to position the CP at the point where you want the new word inserted. The "Itext^Z" command string would then be entered to insert the new text at the place desired. The text to the right of the invisible CP would move to the right to make room for the new characters. The command would then immediately exit back to the command mode. This is demonstrated in the next example.

^ = CP

```
1: * This is the first line of text.
1: * ^
```

Use the "C" command to move the CP 26 character positions to the right.

```
1: * This is the first line of text.
1: *                          ^
```

The CP is now positioned at "t".

```
1: * This is the first line of text.
1: * Inew ^Z
```

Insert "new " at CP position and return to command mode. Note the space after the "w" in "new".

```
1: * This is the first line of new text.
1: *                              ^
```

The CP is still positioned at the 26th character position.

As new text is inserted into the editing buffer, it is good practice always to return to the inserted or modified line of text and display it to verify that it was inserted as intended. We recommend this since the CP position cannot be seen on the console device and it is very easy to miscalculate its position.

ENGAGE OR DISENGAGE CHARACTER TRANSLATION TO UPPER CASE

```
*U
*-U
```

In some instances, ED may have to translate all lower-case characters entered on the console device to upper case. This is sometimes required for assembly language files that will be used with assemblers requiring upper-case-only characters. Normally, ED will not translate characters. It will, however, insert characters in the editing buffer in lower and upper case as they are entered on the console device. ED will perform in this manner unless the "U" (upper case) command is given. To terminate the upper-case translation, you may give the "-U" command. Once characters have been translated into upper case, they cannot be converted back to lower case without retyping the characters.

The following example demonstrates that effect of this command.

```
  : *I
1:  This is the first line of text.
2:  ^Z
  : *U
  : *I
2:  This is the second line of text.
3:  ^Z
3: *-U
3: *I
  :  This is the third line of text.
  :  ^Z
  : *B
  : *<cr>
1: * This is the first line of text.
1: *<cr>
```

```
2: * THIS IS THE SECOND LINE OF TEXT.
2: *<cr>
3: * This is the third line of text.
3: *
```

This example shows that since the U command was not issued, the first line of text inserted in the editing buffer does not have any character translation. The second line, entered after the U command had been given, had all of its characters translated to upper case even though the characters were entered in upper and lower cases. By the use of the -U command, the third line was inserted into the editing buffer with no character translation. The use of -U is relatively rare, since most people don't want to have all text in a file in upper case. It is useful to be aware of -U because character translation to upper case can sometimes be turned on inadvertently.

MOVE CP BACKWARD OR FORWARD ONE OR SEVERAL CHARACTERS ON A LINE

***nC**
*** – nC**

It is useful to be able to insert text starting at a determined point on a line instead of retyping an existing line. The "character pointer" (CP, which is described earlier in this chapter) indicates where any inserted text will begin. To insert text beginning at a specific location on a line, the CP must be moved. The "nC" command moves the CP. The CP is normally positioned at the first character position of the current line before the nC or insert command is issued. To move the CP a certain number of character positions to the right along a line of text, the "nC" command is given, where "n" is the number of character positions. To move the CP to the left, the "-nC" command is given.

The next example demonstrates the effects of this command.

^ = CP

```
1: * First line of text.
1: * ^

2: * Second line of text.
2: * ^
```

```
2: * Second line of text.
2: * 16C                ^
```
(Move CP 16 character positions forward)

```
2: * Second line of text.
2: * -3C           ^
```
(Move CP 3 character positions back)

```
2: * Second line of text.
2: * <cr>          ^
3: * Third line of text.
3: * ^
```
(The CP is now at first position.)

The nC or -nC commands are only effective for the current line of text. When the RETURN key is pressed to advance to the next line of text, the CP will move with the cursor. When this happens, the CP will be positioned at the first character position of the new line. A subsequent use of the nC command will move the CP along the new line of text.

MOVE CP BACKWARD OR FORWARD ONE OR SEVERAL LINES

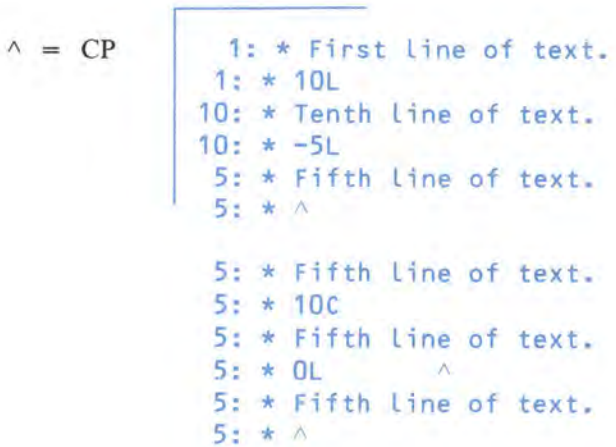
*nL
*-nL

The ED program allows the CP (character pointer, as described previously) to be moved backward or forward a specified number of lines. The "nL" command is used for this, where "n" is the number of lines the CP is moved from the current line. To move the CP forwards (towards the end of the editing buffer), use "nL". To move it backwards (towards the beginning of the editing buffer), use "-nL". After nL or -nL has been issued, the CP will be positioned at the first character position of the new line. At that point, the new line will be displayed with the cursor or print head right below it.

When the value of "n" in nL (or -nL) is greater than 0, the CP will be moved "n" number of lines forwards or backwards. If the value of "n" is 0 (0L), then the CP will be moved to the beginning of the current line. The 0L command is useful when the CP is positioned somewhere along the line, as

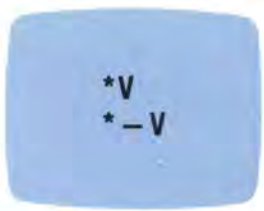
controlled by the "nC" command described previously, and you want to move it back to the beginning of the line.

The following example shows the various uses of this command.



This command is frequently used, since it is often necessary to move the CP back and forth several lines to review text for possible editing. Any number of lines can be specified for moving the CP, providing the specified number does not exceed the total number of lines presently in the editing buffer. If the number does go beyond the last line in the editing buffer, the CP stops at the end of the buffer. This is indicated by the ":*" prompt (i.e., no line number). Reaching the end of the editing buffer, however, does not necessarily mean that the end of the original source file has been reached.

DISPLAY OR SUPPRESS DISPLAYING LINE NUMBERS



Normally, each time ED is accessed, the program will assign numbers to each line of text in the editing buffer. The "-V" command can be used to suppress the displaying of line numbers. One does not often use this command, but for some applications it may be useful. While the displaying of

line numbers is suppressed, all the ED prompts appear as normal, without the line numbers. The ED program always keeps a track of the line numbers, whether they are displayed or not. It is suggested that line numbers always be displayed. ED is a line-orientated text editor, and line numbers make it easier for the user to keep a track of text in the editing buffer. If -V has been used to suppress the display of line numbers, "V" can be used to re-engage their display.

MOVE CP TO A SPECIFIC LINE NUMBER



The "n:" command can be used to move the CP (character pointer) to a specific line number in the editing buffer. This command is similar to the "nL" command, as described previously, except that an actual line number is specified as the target line. To use this command, simply type the line number followed by a colon (":"). The following example shows how this is done.

```
  1: * This is the first line.
  1: * 10:
 10: * This is the tenth line.
 10: * 100:
100: * This is the one hundredth line.
100: * 50:
 50: * This is the fiftieth line.
 50: * 20:
 20: * This is the twentieth line.
 20: * 1:
  1: * This is the first line.
  1: *
```

Using "n:" you can move the CP to *any* line in the editing buffer provided that you know the line exists.

MOVE CP TO BEGINNING OR END OF EDITING BUFFER

```
*B
*-B
```

ED provides a command to move the CP (regardless of its current position) to the absolute beginning or end of the editing buffer. To move the CP to the beginning of the buffer (i.e., the first line), the "B" command is used. To move the CP to the end of the buffer (i.e., the last line), the "-B" command is used.

The following example shows the results of using both these commands.

```
  1: * This is the first line.
  1: * 50:
 50: * This is the fiftieth line.
 50: * B
  1: * This is the first line.
  1: * -B
100: * This is the hundredth (last) line.
100: * <cr>
   : * B
  1: * This is the first line.
  1: *
```

DISPLAY TEXT BEFORE OR AFTER CP

```
*T
*0T
*0TT
*nT
*-nT
*#T
*-#T
```

The "T" (type) command can be used to view or display text that is in the editing buffer. There are various forms of the command, depending on the position of the CP (character pointer) and the amount of text to be displayed. The following paragraphs describe each form of the command.

T (DISPLAY TEXT ON LINE AFTER CP): The basic "T" command is used to display *all the current line's text after* the CP position. If the CP isn't at the first character position, this command is used to verify its actual position on the line. See the next example:

```
 1: * This is the first line.
 1: * 12C
 1: * T
*first line.
 1: *
```

This example shows that the CP was moved 12 character positions to the right of the beginning of the line. When the "T" command was issued, all characters on that line after the new CP position were displayed on the next line. The "*" character indicates the new CP position.

0T (DISPLAY TEXT ON LINE BEFORE CP): The "0T" command will display all characters on the current line *before* the CP position. This is demonstrated in the following example.

```
1: * This is the first line.
1: * 12C
1: * 0T
1: * This is the *<cr>
1: *
```

If the CP's current position is at the beginning of the line, then no text will be displayed. If the CP has been moved along the line to the right, the "0T" command will display all the characters up to the CP position, that is, all characters to left of the CP. When it does, the cursor or print head remains on that line ready for another command. The CP position is indicated by the "*" character. The preceding example showed the pressing of the RETURN key after the "*" character. You can, however, type any ED command at this point. The insert command, for example, or the "T" command, as described previously, could be used.

0TT (DISPLAY ALL TEXT ON THE LINE): The "0TT" command displays all the text on the current line, regardless of the CP position. "0TT" is useful when the current line is the first line in the editing buffer. This is a

line of text that ED will not normally display. The following example demonstrates this.

```
1: * OTT
1: * This is the first line.
1: *
```

nT (DISPLAY A SPECIFIED NUMBER OF LINES AFTER THE CP): The "nT" command is used to display a specified number of lines in the editing buffer *after* the current line and CP position. The specified number of lines ("n") is entered before the "T" command. Any number of lines may be displayed, providing the number does not exceed the remaining number of lines in the editing buffer. Once all the lines have been displayed, the cursor or print head returns to the original CP position (original current line). The next example demonstrates the use of this command.

```
1: * 5T
1: * This is the first line.
2: * This is the second line.
3: * This is the third line.
4: * This is the fourth line.
5: * This is the fifth line.
1: *
```

-nT (DISPLAY A SPECIFIED NUMBER OF LINES BEFORE THE CP): This command is similar to the one described previously, except that the lines specified for display are those prior to the current line and CP position. This is accomplished by preceding the "nT" command with the "-" character (indicating reverse). Once all the specified lines have been displayed, the cursor or print head returns to the original line and CP position. The following example demonstrates this.

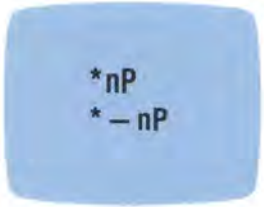
```
6: * -5T
1: * This is the first line.
2: * This is the second line.
```

```
3: * This is the third line.
4: * This is the fourth line.
5: * This is the fifth line.
6: *
```

#T (DISPLAY ALL LINES OF TEXT AFTER THE CP): This command will display ALL lines of text in the editing buffer after the CP position. The command consists of the "#" character, which precedes "T". If the CP is at the first line, this command will display the entire editing buffer. Once all lines have been displayed, the cursor or print head returns to the original line and CP position.

-#T (DISPLAY ALL LINES OF TEXT BEFORE THE CP): This command is the same as the previous one, except that all lines prior to the CP position will be displayed. Once all lines have been displayed, the cursor or print head returns to the original line and CP position.

MOVE CP ONE OR SEVERAL PAGES AND DISPLAY TEXT



Another method of displaying the text in the editing buffer is to use the "nP" command. This command will display what is called a "page" of text, which is a fixed number of lines. The default number of lines is 23, although in some CP/M packages the ED program has been modified so that the number of lines is 15 for use with video terminals that have fewer than 24 lines on the screen. This ED command is useful to display a screen-full of text according to whether your video terminal supports up to 16 or 24 horizontal lines. The command is entered by typing either "P" to display a single page after the CP position, or "P" preceded by a number ("n") to specify the number of pages to be displayed. The command can also specify that pages of text prior to the CP position be displayed, by preceding "P" or "nP" by the "-" character. When this command is used, the CP will move to the end of the last page displayed. You can, however, display a page of text without moving the CP position by using the "0P" command (after the CP position)

or the "-0P" command (before the CP position). The next example shows the effect of using the "P" command with the standard ED program (supporting 23 lines), using a video terminal with 24 horizontal lines.

```
^ = CP      1: * P
           2: * This is the second line.
           3: * This is the third line.
           4: * This is the fourth line.
           5: * This is the fifth line.
                          |
                          |
                          |
                          V
          22: * This is the twenty second line.
          23: * This is the twenty third line.
          23: * ^
```

You may not find this command very useful if the video terminal you are using as the console device does not support at least 24 horizontal lines. You may, however, find the command useful if the console device is a teletypewriter.

DELETE ONE OR SEVERAL CHARACTERS ON A LINE BEFORE OR AFTER CP

*nD
*-nD

The character deletion command is useful for making modifications to specific areas on a line of text. The "nD" command is used to delete a character, or a number of characters before or after the current CP (character pointer) position. To make character deletions after the CP, use the "D" command to delete one character, or the "nD" command to delete several characters, where "n" is the number of characters. When this command is used, ED will count the number ("n") of characters after the CP position and delete them.

To delete characters before the CP position, precede the "D" or "nD" command with the "-" character. The following example demonstrates how the CP is first positioned, and then a certain number of characters are deleted.

^ = CP

```
 5: * This is the the ffifth line.
 5: * 16C
 5: * T
*ffifth line.
 5: * D
 5: * OTT
 5: * This is the the fifth line.
 5: * OT
 5: * This is the the * -4D
 5: * OTT
 5: * This is the fifth line.
 5: *             ^
```

When using the "nP" command, note that the carriage return line terminator consists of the ASCII CR (carriage return) and LF (line feed) characters, and should be counted as two separate characters, even though they are not displayed on the console device.

DELETE (KILL) ONE OR SEVERAL LINES BEFORE OR AFTER CP

*nK
*-nK

This command is used to "kill" or delete lines of text from the editing buffer. This "K" command will delete all characters on the current line of text, after the CP (character pointer). If the CP is positioned at the first character position on the line, all characters on that line will be deleted. If the CP is positioned after the first character pointer, all the characters after the CP will be deleted and the characters before the CP will remain. To delete all characters on a line before the CP position, the "-K" command can be used. To delete several lines of text from the editing buffer, a number can precede the "K" command ("nK"). The number of lines specified for deletion will be

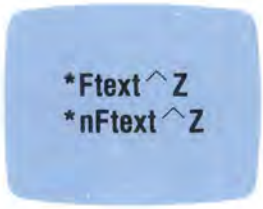
counted from the current line and CP position. The "nK" command will delete "n" number of lines after the CP position, and the "-nK" command will delete before the CP position. The following examples demonstrate the use of this command.

```
50: * Original line 50.
50: * K
50: * <cr>
50: * Original line 51.
50: *
```

```
 1: * Original line 1.
 1: * <cr>
 2: * Original line 2.
 2: * -K
 1: * OTT
 1: * Original line 2.
 1: *
```

This second example shows the results of using the "-K" command to delete a line of text previous to the line at which the CP and cursor are.

FIND TEXT STRING IN EDITING BUFFER (ONCE OR SEVERAL TIMES)



To find or search for a particular string of text in the editing buffer, the "F" (Find) command is used. The command is entered by typing in "Ftext^Z", where "text" is the particular string of text you want found, and ^Z (CTRL/Z) is the terminator. Once the RETURN key is pressed after this command string, ED will search the editing buffer starting from the current CP (character pointer) position until the text string is found. Once the string has been found, the CP will be positioned after the found text string.

Strings of text can be searched for either before or after the CP position. By preceding the aforementioned command string with the "-" character ("-Ftext^Z"), ED will begin searching towards the beginning of the editing buffer from the current CP position. If you know that there are several occurrences of the text string to be found, you can specify which occurrence of the string you want found, by preceding the command with a number. The number ("n") instructs ED to find the n'th occurrence of the searched-for text string, and the CP will be placed at that point. If you want to include the line terminator CR LF (ASCII carriage return and line feed) in the searched-for text string, type CTRL/L, and "^L" will appear in the command line. The following example demonstrates the use of this command.

^ = CP

```
  1: * Freference document number 30XX40.^Z
263: * for information, reference document number 30XX40. The
263: *                                                      ^
```

```
  1: * 4F, ^Z
  5: * set N, X, Y, 0, and then Z. Use this method for future
  5: *                ^
```

Note that this command will search only in the editing buffer for the text string. If the possibility exists that the searched-for text string may be in a part of the source file not yet appended to the editing buffer, use the "N" command as described next.

FIND TEXT STRING IN EDITING BUFFER (ONCE OR SEVERAL TIMES) AND PERFORM APPEND AND WRITE OPERATIONS AS NECESSARY



This command is used in the same manner as the "F" command described previously. The "N" command will, however, automatically perform append

operations from the source file to the editing buffer for more text, if the searched-for text string cannot be found among the information already in the editing buffer. While text is being appended to the editing buffer, ED will also write necessary text to the temporary output file to ensure that space is available in the editing buffer. The command string variations for this command are the same as those for the "F" command.

FIND TEXT STRING IN EDITING BUFFER AND SUBSTITUTE IT WITH SPECIFIED TEXT (ONCE OR SEVERAL TIMES)



There is a command that is similar to ED's Find ("F") command. It is a substitution command that finds and automatically replaces the found text string with another specified text string. The command is entered by typing "S" (Substitute) followed by the text string to be found, and the replacement text string. The CTRL/Z ("^Z") sequence is used twice in this command string. Once to separate the "find" string and "replace" string, and again to terminate the command. A number ("n") may also precede "S". It instructs ED to make the same search and substitution a specified number of times.

Text substitutions can be made on the text in the editing buffer before or after the CP (character pointer) position. Once the substitution is made, the CP will be positioned immediately after the substitution string. To specify the line terminator characters CR LF (ASCII carriage return and line feed) in either text string, enter CTRL/L ("^L").

The following example demonstrates the effects of the substitute command.

^ = CP

```
30: * This is the thirty first line.
30: * Sthirty first^Zthirtieth^Z
30: * OTT
30: * This is the thirtieth line.
30: *                          ^
```

```
50: * The locations are 300, 400, 500, and 600.
50: * 4S0^ZH^Z
50: * 0TT
50: * The locations are 30H, 40H, 50H, and 60H.
50: *                                          ^
```

If searching forward in the buffer, always make sure that the CP is positioned *before* the text to be searched for and replaced, before using the substitution command. If searching backwards, the CP should be positioned *after* the text being searched.

FIND TEXT STRING IN EDITING BUFFER AND JUXTAPOSITION IT WITH SPECIFIED TEXT (ONCE OR SEVERAL TIMES)



The "J" (Juxtaposition) command is a variation of the Substitution command (see preceding example). It allows you to search for a text string (starting from the current line and CP position) and add text to that found string. In addition, you can then delete text starting from the found string until a specified end text string is found. A "J" can be used with all or some of the command string features.

The command is entered by typing "J", the "find" text string, a CTRL/Z ("^Z"), the text string that is to be inserted next (juxtaposed) to the "find" string, another CTRL/Z, the "end" text string up to which text is to be deleted, and a final CTRL/Z. In other words, Jfind^Zinsert^Zend^Z. If there is no text to be deleted after the juxtaposed text, then simply enter two CTRL/Z's (with no text in between), after the "insert" text string. Precede "J" with the number ("n") if the "insert" text is to be inserted or juxtaposed after the "find" string a specified number of times. Note, however, that this repeated count only applies to the first occurrence of the "find" text string in the editing buffer. The "J" command string must be repeated manually for all other occurrences of the "find" string in the editing buffer.

The following examples demonstrate how this command is used.

^ = CP

```
40: * When unplugging the cable, be extremely careful.
40: * Jcable, ^Zdo so with caution^Z.^Z
40: * OTT
40: * When unplugging the cable, do so with caution.
40: *                                                ^
```

```
50: * 2T
50: *  ------------------------------------------------
51: * ¦ STEP ¦       ACTION       ¦     VERIFICATION     ¦
52: * ¦======¦======+++++=======¦======================¦
52: * 8J¦======^Z=^Z=====¦^Z
52: * -3T
50: *  ------------------------------------------------
51: * ¦ STEP ¦       ACTION       ¦     VERIFICATION     ¦
52: * ¦======¦====================¦======================¦
52: *                      ^
```

COMBINING CP MOVEMENT COMMANDS

***#AB#T**

The ED program permits the combination of certain commands, so that they can be typed on a single line, rather than on separate lines. For example, if you had just opened a file using ED, wanted to append the contents of the source file to the editing buffer, back up to the beginning, and then display the contents, you could enter: #A (Append), B (move CP to beginning of buffer), and #T (display ("type") buffer contents) all on one command line.

The next examples demonstrate the two ways this command sequence can be entered.

```
  : * #A
 1: * B
 1: * #T
 1: * This is the first line.
 2: * This is the second line.
                ¦
                ¦
                V
72: * This the seventy second (last) line.
72: *
```

```
  : #AB#T
 1: * This is the first line.
 2: * This is the second line.
                ¦
                ¦
                V
72: * This the seventy second (last) line.
72: *
```

As shown in the second example, the commands in a sequence must be entered in the order in which you want them executed. Not all of the ED commands can be entered in combination. Some commands may cause undesirable effects when grouped together with other commands. Another cautionary note involves certain rules of syntax which must be followed. Take careful note of the following rules, since the structure of command groups can often become lengthy and rather complex.

1. In CP/M Version 1.4 and newer, all ED commands can be grouped together, except for the following:

   E (Exit ED and save buffer contents in output file).
   H (Save buffer contents in output file without exiting ED).
   Q (Exit ED without saving buffer contents in output file).
   O (Delete buffer and output file contents without exiting ED).

   Any of the preceding commands must be entered as the only command on the command line. This restriction prevents the inadvertent typing of any of these commands in combination with others. These com-

mands grossly affect the operation of ED. They would, consequently, seriously affect any file being worked on.

2. When a group contains commands which use strings of text, CTRL/Z ("^Z") must be entered to indicate the end of the string(s). The following commands use text strings.

   I (Insert text in editing buffer).
   F (Find text in editing buffer).
   N (Find text in editing buffer and append if necessary).
   S (Substitute found text in editing buffer with specified text).
   J (Juxtaposition text to the found text, and delete text if necessary).

The next example demonstrates how groups of commands that contain text strings are structured.

```
40: * The disctionary should be referenced if in dout.
40: * Fdis^Z-1DFdout^Z-1DIbt^Z0TT
40: * The dictionary should be referenced if in doubt.
40: *
```

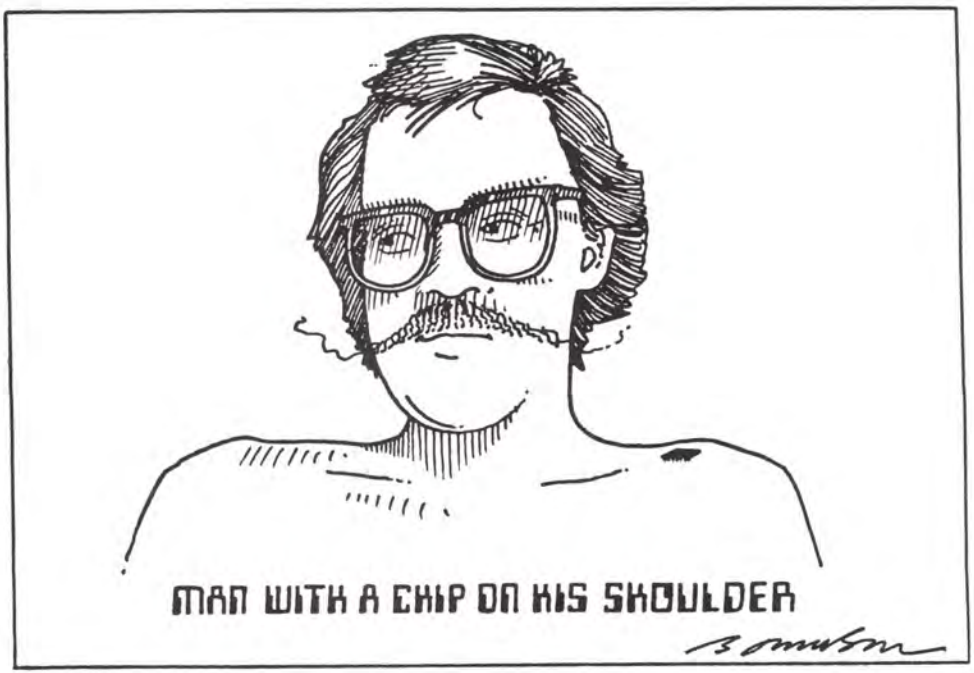
The preceding example shows how all of the misspellings were corrected throughout the complete line of text. First "dis" was searched for. This search positioned the CP right after the "s". Next the "-1D" command was given to delete the "s" to the left of the CP. Then "dout" was found, positioning the CP after the "t". The "t" was deleted with "-1D", and "bt" was inserted in its place. Finally, the "0TT" command was entered which caused the newly modified line to be displayed in its entirety. Remember that "^Z" must always be entered to end text strings. Only use the RETURN key to terminate the group of commands.

MACRO REPETITION OF COMMAND GROUPINGS

*nM

When groups of commands are used (as just described), it may be required that a particular command sequence be repeated several times

throughout the editing buffer. To establish this repetition, the "nM" command is used, where "n" is the number of times you want the command sequence repeated. When "nM" is used, ED will execute the command on every occurrence of the specified text string (after the CP position), in the editing buffer. The command will be repeated until the specified number of times has been reached. The "n" value must be a number greater than 1, or the command will continuously repeat itself through the editing buffer until an error condition is reached. The same thing will happen if a value of 0 is given.



The following example demonstrates the correct syntax when applying the repetition macro command to a group of commands.

```
40: * The disctionary should be referenced if in dout. The
40: * 5MFdis^Z-1DFdout^Z-1DIbt^ZOTT
40: * end. The dictionary should be referenced if in doubt.
67: * The dictionary should be referenced if in doubt. If the
```

```
 95: * chance. The dictionary should be referenced if in doubt.
178: * The dictionary should be referenced if in doubt.
332: * last. The dictionary should be referenced if in doubt.
332: *
```

This example shows that the command grouping was repeated 5 times. Each time effected the next occurrence of the specified text string.

VERIFY TOTAL EDITING BUFFER USAGE AND FREE SPACE



In CP/M Version 1.4 and newer, the "V" command (as described previously), has a special function. When "0V" is entered, information concerning the usage of the editing buffer is displayed. This information includes how much of the editing buffer is in use, and what is the total space available. The numbers presented represent bytes. This is demonstrated in the following example.

```
: * 0V
: * 21506/32183
: *
```

In the example, the number of bytes already used is presented first. This number is followed a "/" character and the total number of bytes available in the buffer. By subtracting the amount used (21506 in this example) from the total amount (32183), the amount of free space (10677 bytes), can be determined. The amount of RAM in your computer determines the total number bytes allowed in the editing buffer. The amount shown in the example is based on a system with a total of 64K ("K" = kilobytes, or 1,024 bytes) of memory. Systems with less memory (such as 48K, 32K, 16K, etc.) will have a lower amount, although systems with more memory (such as 128K) will probably have more, depending on how the CP/M system image was generated for memory utilization.

WRITE SEVERAL LINES FROM EDITING BUFFER TO TEMPORARY OUTPUT FILE



The ED "W" (Write) command allows the transfer of lines of text from the editing buffer to the temporary output file. This command will "write" the specified lines from editing buffer and place them after the most recently "written" line in the temporary output file. When this happens, the newly written line will be deleted from the editing buffer.

One, several, or all lines can be written with this command. When typing "W" by itself, only the first line in the editing buffer will be written. The "W" command can be preceded by a number (i.e., "nW"). When entering "0W", all lines will be written until the first half of the editing buffer is reached. When "n" is greater than 0, the first "n" number of lines in the editing buffer will be written. When entering "#W", all lines in the editing will be written. Writing will always start from the beginning of the editing buffer.

The "W" command is useful when the "A" (Append) command is employed to append small portions of text at a time from the source file. When ED is being used in this manner, the "#W" command can be used to write each small portion of text to the temporary output file. After the text has been edited, (i.e., the lines will be erased from the editing buffer), the original space will be made available for the next "Append" operation. It is advised that the "W" command only be used under these or similar circumstances. Otherwise, you may find peculiar results when examining the final output file.

The following example demonstrates the various uses of this command.

```
1: * 2T
1: * This is the first line.
2: * This is the second line.
1: * W
```

```
1: * 2T
1: * This is the second line.
2: * This is the third line.
1: *
```

In this example, the first line was written to the temporary output file. After the line was written, it was deleted from the editing buffer, and all lines thereafter were brought back to the beginning of the edit buffer. Each line number was decremented by one (line 2 became line 1, etc.).

The next example demonstrates the results of writing all lines from the editing buffer.

```
1: * 2T
1: * This is the first line.
2: * This is the second line.
1: * #W
 : * #T
 : *
```

The example demonstrates the results of writing all lines of text from the editing buffer with the "#W" command. Once the lines were written, they were deleted from the editing buffer. At this point, the buffer is empty, as indicated by the ":*" prompt without a line number. Now you can use the "A" (Append) command to append more text to the editing buffer, or include text from other temporary files. Be careful not to repeat the same "Append" and "Write" operation more than once (unless required) with the same set of text. If you do, you will have an output file containing two sets of identical text!

WRITE SEVERAL LINES FROM EDITING BUFFER TO TEMPORARY LIBRARY FILE

***nX**

You can write lines of text (beginning from the current line and CP position) from the editing buffer into a special temporary file, called a

"Library" file. This is done in a similar way to the operation of the "W" (Write) command (described previously). The difference between the two commands is that "X" requires that the certain number of text lines to be written must be specified. To do this, enter "nX", where "n" is the number of lines.

Once the "nX" command is given, the specified lines will be written to a special temporary file called "X$$$$$$$.LIB". Note that lines written into the temporary file will not be erased from the editing buffer. Once this file is created, the "R" (Read) command (described next), will "read" the contents of the file back into the editing buffer at a location immediately *after* the current line and CP position. The temporary X$$$$$$$.LIB file will exist until you exit ED, at which time it will automatically be deleted. This file will not, however, be deleted if there was a system failure which required a system reboot (Cold Boot) for restart during an ED edit session. If this happens, you can re-enter ED, find the place where you had been working, and use X$$$$$$$.LIB again.

A subsequent use of the nX command during the same ED edit session will erase the original contents X$$$$$$$.LIB. Any new text will replace the old text in X$$$$$$$.LIB.

The following example demonstrates how this command is used.

```
70: * 6T
70: * shouldn't be taken for granted.
71: * ------------------------------------------------
72: * This portion of the document will be updated when the
73: * document is reissued to a later revision.
74: * ------------------------------------------------
75: * Several items should be taken into consideration when
70: * 71:
71: * 4X
71: *
```

Note that in this example, lines 71 through 74 (4 lines) were written to X$$$$$$$.LIB with the "4X" command. These four lines were not deleted from the editing buffer, and consequently all line numbering remains intact. The text in X$$$$$$$.LIB can now be copied anywhere into the editing buffer by using the "R" Read command (next).

READ CONTENTS OF TEMPORARY FILE CREATED BY THE nX COMMAND INTO EDITING BUFFER

*R

The "R" (Read) command is used to copy text back into the editing buffer from the file "X$$$$$$$.LIB" which was created by the "nX" command (see preceding command). By positioning the CP at the editing buffer location where you want the text, you can "move" text from one location to another. This is done with the combined use of the "nX" and "R" commands. When the "R" command is used, it does not delete the contents of X$$$$$$$.LIB. Because of this, you can use the "R" command as many times as desired during the same ED edit session. The X$$$$$$.LIB file will not be deleted until you exit ED in the normal fashion.

The next example demonstrates the use of the "R" command, and is a continuation of the example in the "nX" description.

```
71: * 85:
85: * -2T
84: * Take careful note of how this procedure is followed.
85: * In future revisions, this procedure will be modified.
85: * R
85: * 84:
84: * 6T
84: * Take careful note of how this procedure is followed.
85: * ------------------------------------------------
86: * This portion of the document will be updated when the
87: * document is reissued to a later revision.
88: * ------------------------------------------------
89: * In future revisions, this procedure will be modified.
84: *
```

In this example, the contents of X$$$$$$$.LIB were read into the editing buffer *after* the current CP position, at line 85. Since the CP was positioned at the beginning of line 85, the text was added *before* the text on line 85. As a result, the pre-existing text on that line was moved down 4 lines to line 89.

All subsequent line numbers in the editing buffer were incremented by 4. The "R" command can be used as many times as you like at various places in the editing buffer, without deleting the contents of X$$$$$$$.LIB.

READ CONTENTS OF LIBRARY FILE INTO EDITING BUFFER

***Rfilename**

The contents of a specified text file can be read into the editing buffer. When commanding the ED program to do this, the file name must have a file type or extension of ".LIB". Once this has been done, you can use this command by simply typing in "R" followed by the file name, without the extension. ED will automatically search for the file name specified, and if it has an extension of the current CP position. Be sure to position the CP where you want the file to be placed *before* using this command.

The following example shows how this command is used.

```
  1: * -B
   : * -1L
 67: * OTT
 67: * This concludes Chapter 2.
 67: * <cr>
   : * Rchapter3
   : * 67:
 67: * #T
 67: * This concludes Chapter 2.
 68: *              ** CHAPTER 3: APPLICATIONS **
 69: *
 70: * This chapter describes the various applications of the

                    ¦
                    ¦
                    V

231: * This concludes Chapter 3.
 67: *
```

In this example, the text in the editing buffer terminates with the end of "Chapter 2". By using the "-B" command, the CP was positioned at the last line in the editing buffer. The file "CHAPTER3.LIB" was then specified with the "R" command. This caused the contents of the file to be copied into the editing buffer *after* the CP position. Finally, the CP was moved to end of the editing buffer. The CHAPTER3.LIB file was not affected by this command.

SAVE ALL EDITING IN FINAL FILE WITHOUT EXITING ED



During an ED editing session, it is often useful to be able to save all the text in the editing buffer in the final output file. As described earlier in this chapter, ED renames the original file specified for editing to "filename.BAK". By doing this, it creates a temporary output file which will contain text for the final output. When the "E" (Exit) command (described next), is used to exit ED, this temporary output file is saved on the disk. It is renamed to the original name of the file that you specified when first accessing ED. The "H" command allows you to perform this function without exiting ED. When this command is used, the following occurs:

1. All text in the editing buffer is moved to the temporary output file ("filename.$$$"). The text is then erased from the editing buffer.
2. Any text remaining in the original specified file ("filename.ext") that had not been appended to the editing buffer is copied to the appropriate place in the temporary output file ("filename.$$$").
3. The original, unmodified file ("filename.ext") is then renamed to "filename.BAK", and becomes the backup file.
4. The temporary output file ("filename.$$$") then becomes the new source file and is renamed with the name of the original file "filename.ext" (for example, "DOC1.TXT").
5. A new temporary output file ("filename.$$$") is then created.

Once the preceding occurs, the ED prompt ":*" appears (without a line number). This indicates that the editing buffer is empty and ready to receive

text. Any text that you append (using the "A" command) will be retrieved from the new source file (filename.ext). Any text that is output from the editing buffer will go to the new temporary output file (filename.$$$).

If you are editing a large document, or simply performing much text insertion and editing, it's a good idea to use this command every 15-30 minutes. In this way, if, for any reason, there is a system failure, most of the work accomplished will have been saved on the disk. Remember, if the system failure requires a reboot (Cold Boot), the contents of the editing buffer will be erased. It is, therefore, a good idea to use this command frequently if your system is prone to such failures. (No computer system is completely fail-safe!).

EXIT ED (RETURN TO CP/M) AND SAVE ALL EDITING IN FINAL FILE



Use the "E" (Exit) command to exit the ED program and save all the work you have done in the final output file. This command will instruct ED to return you to the CP/M command mode, and in the process, perform all the disk file actions as described previously under the "H" command description. The use of "E" is normal when exiting ED, in that it will save all of the editing you have done on the disk. When it is entered (it can be entered at line in the editing buffer), the console cursor or print head will move down one line. There will follow a short pause while ED performs all of the buffer-read, disk-read, and disk-write operations. Once these operations are completed, the original CP/M logged drive name prompt will reappear, and you will be back in the normal CP/M command mode.

The following example demonstrates what happens when this command is invoked.

```
45: * E

A>
```

Once you have exited from ED, it's a good idea to use the DIR or STAT command to verify that the files you were working with are as they should be. The new output file ("filename.ext") should have the same name as the file

you originally specified when you first accessed ED. In addition, a backup file should exist with the same file name as the original, but with an extension of ".BAK".

ERASE ALL EDITING IN EDITING BUFFER AND FINAL FILE WITHOUT EXITING ED



The "O" command is similar to the "H" command described previously, except that none of the contents of the editing buffer or the temporary output file are saved on the disk. This command is used to eliminate the work done thus far and create a "clean slate". By this we mean that you will have an empty editing buffer and an empty temporary output file. In other words, you will be returned to square one (i.e., when first entering ED). Because this command can be rather destructive, use it sparingly and only if you are absolutely certain that you want the described results.

When the "O" command is used, the ":*" prompt will appear (without a line number). This indicates that the buffer has been emptied. This command will not exit you from ED, but will return you to the original point where you first entered ED. The original source file will be unchanged, so any append operations initiated now (using the Append (A) command) will retrieve text from the source file as if you had just entered ED.

QUIT ED (RETURN TO CP/M) AND ERASE ALL EDITING IN EDITING BUFFER AND FINAL FILE



The "Q" (Quit) command is used to exit ED and return to the main CP/M command mode. ED will, however, perform all of the functions described for the "O" command (see preceding command). This command is used if you want to exit ED, *without saving* any of the work done (i.e., the editing buffer and the temporary output file will be emptied, and the original source file

will be left unchanged). The "Q" command should be used with caution. Do not invoke this command unless you are absolutely certain that you want the described effects.

As a safeguard against the inadvertent use of this command, the ED program will ask (once and only once), if you indeed wish to execute or abort the command. If you answer with "N" for no, then ED will *abort* the command, and you will be returned to point in the editor where you left off. If you answer "Y" for yes, then the Quit command will *execute*, and the logged CP/M drive name prompt will reappear.

Take heed of the last question that ED asks, since it is easy to accidentally type the "Q" without meaning to. This is shown in the next example.

WHEN ANSWERING "NO" TO QUIT:

```
 20: * Q
Q-(Y/N)?N
 20: *
```

WHEN ANSWERING "YES" TO QUIT:

```
 20: * Q
Q-(Y/N)?Y
 A>
```

When typing in "N" or "Y" in response to "Q-(Y/N)?" do not terminate the response with the RETURN key. ED will respond immediately upon typing "N" or "Y".

ED ERROR MESSAGES

There are several conditions for which ED will respond with an error message. An error message is usually related to a command that was entered improperly, or at the wrong place in the editing buffer. Each error message, however, contains a symbol which describes the nature of the error. The for-

mat of the error message is as follows: "BREAK-X-C". The word "BREAK" means that ED had to interrupt what you were doing and present the error message. "X" is a specific character which describes the type of error. "C" is the last character in the command given that was read by ED. The types of characters that can occur in the "X" field of the error message are:

? UNRECOGNIZED COMMAND: When the "?" character is presented, it means that the command you typed was not recognizable by ED. This means that either the command itself is not recognizable by ED, or the syntax of the command string was typed in incorrectly. Immediately following "?", the error message will present the last character of the incorrect command or command string that was read by ED.

>EDITING BUFFER FULL: When the ">" character is presented, it means that the editing buffer has been filled to capacity, and that it cannot accept any more input. To make room in the editing buffer you will have to use one of the following commands: nD, nK, nW, H, E, O, Q. Read the descriptions of these commands before any action is taken after this error message. Being fully familiar with the commands will help you to decide which of them is appropriate.

REPEAT COUNT TOO HIGH: When the "#" character is displayed, it means that the value specified for repeat count in the last command typed is too high. For example, if you type "#Ftextstring^Z", ED will determine that the count "#" (all occurrences in the editing buffer) is not allowed when using the "F" (Find) command. In response to this error message, specify a lower repeat count value.

O CANNOT OPEN LIBRARY FILE AFTER THE "R" COMMAND: When the "O" character is presented, it means that the library file name specified with the "Rfilename" command (Read file) could not be opened by ED. This error could either mean that the library file does not exist, or the file name specified had an extension or file type other than ".LIB". To resolve this problem, you should exit ED using the "E" command (Exit), and then search for the file originally specified with the "R" command. If the file is there, and it has an extension of ".LIB", then verify that it isn't a "system" file (see STAT command), and that its capacity is greater than 0K (0 kilobytes). If the file you wanted does not have the ".LIB" extension, then temporarily rename it, or make a copy of it so that the copy has the correct extension.

Chapter 11



CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

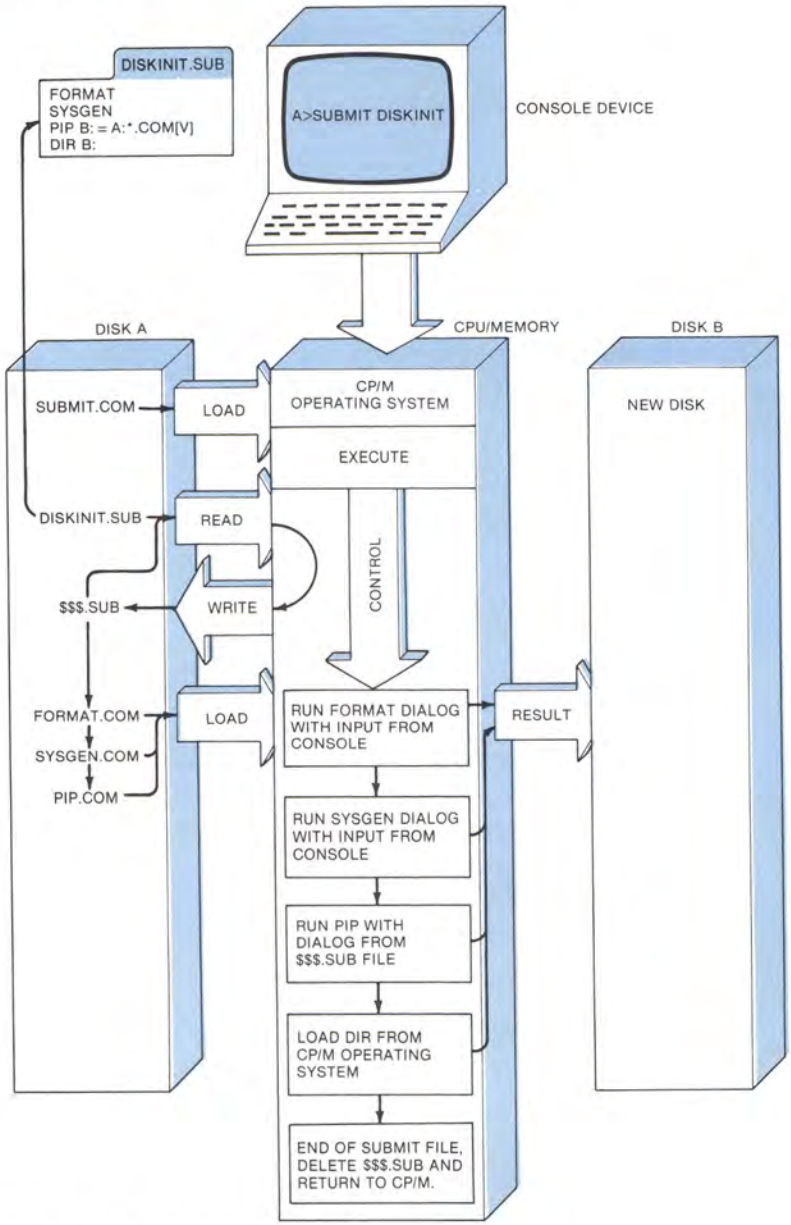
Transient

SUMMARY

SUBMIT is a utility that allows the CP/M user to type a short command which, in turn, causes a longer series of CP/M commands to execute automatically. To accomplish this, a special file containing the desired series of valid CP/M commands (in text form) is created. The SUBMIT program is used to "submit" the commands in the special file to the CP/M command line, as if they were individually entered at the console device by a user. Since commands are issued in an indirect manner, this "submitting" process is similar to the "indirect command processing" used in many types of operating systems.

The SUBMIT utility is used often by programmers who are developing programs that run on CP/M. A submit file can contain a set of repeated commands to the operating system that are frequently required, such as the ones for assembling a file and loading it. A submit file could also contain a series of simple disk-copy commands so as to automate the copy procedure, or commands to run certain programs in succession.

The next figure demonstrates the concept of "submitting" commands to the CP/M command line automatically without having to type them in individually each time.



ACCESSING AND USING SUBMIT

To utilize SUBMIT, you must first create a text "SUBMIT" file using the ED text editor, or some other CP/M-compatible text editor or word proces-

sor. All "SUBMIT" files must have an extension or file type of ".SUB". In a .SUB file, you can type a number of CP/M commands (built-in/direct or transient). Enter the commands in the sequence in which you want them executed. Press the RETURN key after each command or command string. Use the same syntax for these commands as you would when entering them on the CP/M command line.

The following example shows the contents of a ".SUB" file containing a sequence of commands that could be used to initialize a new floppy disk for the first time.

```
A>TYPE DISKINIT.SUB

FORMAT
SYSGEN
PIP B:=A:*.COM[V]
DIR B:

A>
```

The commands in this example could also have been entered individually (in the same sequence) in the main CP/M command line by the user, but the user could not walk away from the machine. The file, "DISKINIT.SUB", contains commands that are used to initialize a new floppy disk in Drive B.

To execute the commands in DISKINIT.SUB, the file name is entered after "SUBMIT" in the main CP/M command mode. (When using SUBMIT, the SUBMIT.COM file must be on the disk you are using.) The ".SUB" extension should not be typed, since the SUBMIT program will automatically find the specified file name, if the correct extension exists. When the SUBMIT program is executed, it is as if the SUBMIT program takes over the operation of your computer for you. The result is that each of the commands in the .SUB file will be executed in the order in which they were originally stored.

For those of you who are interested in the inner workings of SUBMIT, here's what actually happens inside the computer when the SUBMIT program is executed. To begin, the contents of the .SUB file specified for submission will be copied to a temporary file called "$$$.SUB". It is from this temporary file that SUBMIT takes all the commands that were in the

original file. Once all the text of the file have been sent, "$$$.SUB" will be erased and the originally specified .SUB file will be left unchanged.

Continuing the previous example, the following is a demonstration of what happens when the contents of file "DISKINIT.SUB" are "submitted" to the CP/M command line. (The text enclosed in parentheses represent the authors' comments.)

```
A>SUBMIT DISKINIT (your input)

A>FORMAT (typed by SUBMIT)
(the FORMAT program executes as normal,
accepting your input to format disk B)

A>SYSGEN (typed by SUBMIT)
(the SYSGEN program executes as normal,
accepting your input to copy the system
image from A to B

A>PIP B:=A:*.COM[V] (typed by SUBMIT)
(the PIP program executes as normal, and
all ".COM" files on A are copied to B)

A>DIR B: (typed by SUBMIT)
(the directory of all files on B is
displayed)

A>□
(Once finished, the cursor or print head "[]" is returned
to the original CP/M prompt. CP/M is now ready to accept
new commands)
```

As shown in the example, each CP/M command in the "DISKINIT.SUB" files is submitted to the CP/M command line in the order in which the commands occur in the file itself. When the execution of a command is complete, SUBMIT will then submit the command on the next line of the .SUB file. The SUBMIT program continues in this manner until the command on the last line of the file has been executed. At this point, the original CP/M logged drive prompt reappears (A>, B>, etc.).

Any CP/M command and virtually any transient command program can be submitted using the SUBMIT command. Commands that require user

input, such as FORMAT or SYSGEN, will execute as usual, i.e., as if you had entered them individually.

USING VARIABLES IN .SUB FILES

In addition to submitting CP/M commands, SUBMIT also allows special "variable symbols" to be inserted in .SUB files. These variable symbols allow the creation of .SUB files without specifying file names, disk drive names, or any other strings of characters in command lines. Instead, when executing SUBMIT with one of these files (i.e., one with variables), you can enter specific values for the variable symbols immediately following the filename of the submit file. Then the values in the command line are inserted into the variables in the submit file, giving them actual values. The variable symbols are inserted in the file in the form of "$1", "$2", "$3", etc., where "$1" is the variable symbol for the first substitution, "$2" for the second, and so on. To substitute variable symbols when executing SUBMIT, the following syntax is used:

A>SUBMIT FILENAME X1 X2 X3 . . . . .

. . . where "X1" will substitute all occurrences of "$1" in the file, "X2" will substitute all occurrences of "$2", "X3" all occurences of "$3", etc.

The next example shows how "DISKINIT.SUB" (the file in the first example in this section) has been converted to contain variables. With the inclusion of variables, the .SUB file is no longer restricted for use only with disks A and B.

```
A>TYPE DISKINIT.SUB

FORMAT
SYSGEN
PIP $2:=$1:*.COM[V]
DIR $2:

A>
```

By having "$2" where "B" used to be, and "$1" for "A", this SUB file can be used with any disk drive combination. If, for example, you wanted to use it with drives B and C instead of A and B, you would enter the SUBMIT command string as follows.

```
B>SUBMIT DISKINIT B C

B>FORMAT
(program executes)
B>SYSGEN
(program executes)
B>PIP C:=B:*.COM[V]
(program executes)
B>DIR C:
(program executes)
B>
```

This example shows that the submit operation of the file DISKINIT.SUB functioned as before, except that the disk drive names were specified. All occurrences of the variable symbol "$1" were substituted with "B" and all occurrences of "$2" with "C". The variable substitutions can only take place correctly (i.e., as you want them) if the substitution characters or text strings are entered in the desired sequence in the "SUBMIT FILENAME VARIABLE1 VARIABLE2" command string. If you had entered "SUBMIT DISKINIT C B", then "PIP C:=B:*.COM[V]" would become "PIP B:=C:*.COM[V]", and "DIR C:" would become "DIR B:". As a result, the outcome could be rather undesirable. It is always good practice, before submitting a file, to examine the .SUB file (using the TYPE command). In this way, you can verify how many and which variables are used, in addition to the sequence in which they have been entered.

SUBMIT variable characters can be used to substitute virtually any text string in a command. For example, a variable can be used to represent a file name position in a command string. A variable can even represent a CP/M command. A particular variable symbol can be repeated throughout a .SUB file as many times as necessary. SUBMIT will continue to make the appropriate substitutions until the end of the file has been reached. Use these variables symbols at your discretion and with caution. Always verify the .SUB file for the status of the variable symbols before submitting it.

The SUBMIT program will always recognize solitary "$" characters in .SUB files as variable symbols. If the "$" character needs to be placed as a (nonvariable) character in a command string, then you must precede it with another "$" character. This subtle trickery will work because when SUBMIT encounters "$$" in the .SUB file, it will ignore the first "$" and treat the second one as a valid, fixed character in the command string. If, for example, you wanted to use the STAT command string "STAT *.* $R/O" to set all

files on the disk specified to the Read-Only protection status, the .SUB file must contain the command string "STAT *.* $$R/O". In this way, the "$" is treated on an "as is" basis, and not as a variable. This is demonstrated as follows:

```
A>TYPE ALL-R/O.SUB

STAT *.* $$R/O
STAT *.*

A>
A>SUBMIT ALL-R/O

A>STAT *.* $R/O
(program executes as normal after SUBMIT
typed the command string correctly)
A>STAT *.*
(program executes as normal)

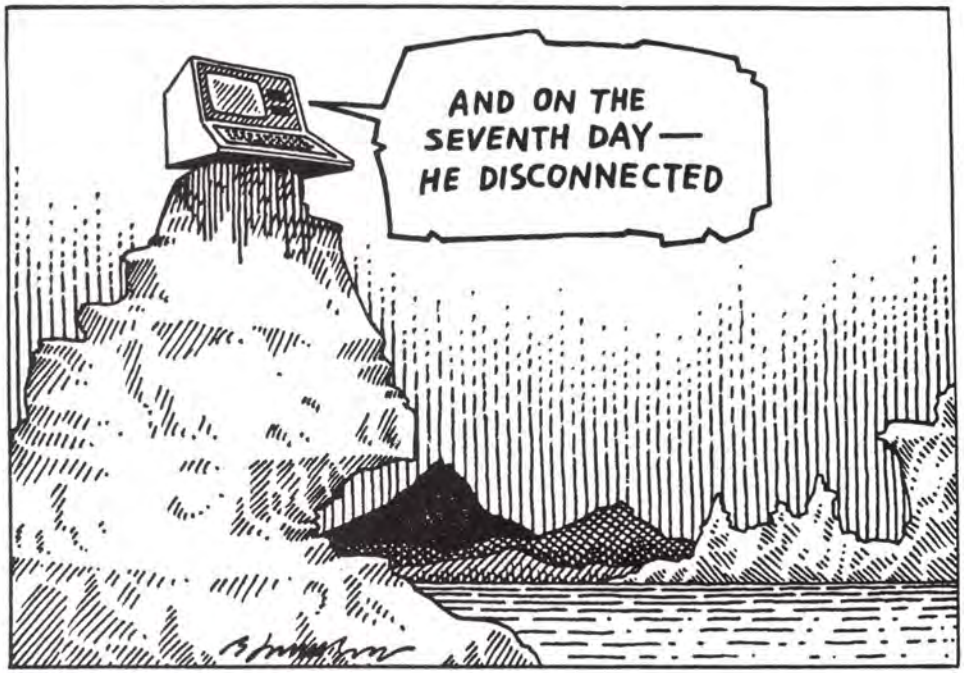
A>
```

The example shows that if the command string "STAT *.* $$R/O" had been inserted in the .SUB file, an error would have occurred.

INSERTING COMMENTS IN .SUB FILES

If several people are going to be using a complex or lengthy SUB file, it is useful to be able to insert comments in the file to document its purpose and parameters. You can insert an unlimited number of comments in a .SUB file as long as each comment line begins with a semicolon. Each comment line must also be separate from the command strings to be submitted. When the SUBMIT command string is entered, the comments will be submitted to the command line and will be recognized by CP/M as comments as long as the comment lines begin with a semicolon in the first character position.

The following example shows that the file DISKINIT.SUB (from the previous example) has now been modified to contain comments.




```
A>TYPE ALL-R/O.SUB

;The following sequence has two $$'s, so
;that SUBMIT does not interpret it as a
;variable symbol.
STAT *.* $$R/O
;After files are set R/O, they are displayed
STAT *.*

A>
```

After a .SUB file with comments is submitted, the comments will also be displayed. Be sure to put two "$" symbols in the comments for every literal "$" that is to be a part of the comment. You can also place variables in comment lines if you wish—the variable substitutions will be displayed when the file is submitted. These comments are useful for documenting the contents of the file when it is being examined with the TYPE command, or modified with a text editor or word processor.

SPECIAL SUBMIT FEATURES

In CP/M Version 2.0 and newer, a special program called "XSUB.COM" is provided. "XSUB.COM" is not a command but, rather, a subset program to SUBMIT.COM. This special utility allows text in a .SUB file to be passed on as input information to another program being submitted. In this way, the information is transmitted as if it had been entered manually on the console device. With this utility, you can create a .SUB file containing the command string to submit a particular program aid then insert the text representing the responses that you would ordinarily input on the console device in response to the program's prompts. This utility greatly increases the power of SUBMIT by incorporating more automation. The XSUB utility is described in the next chapter.

Chapter 12

XSUB

CP/M Versions:
2.0 2.2

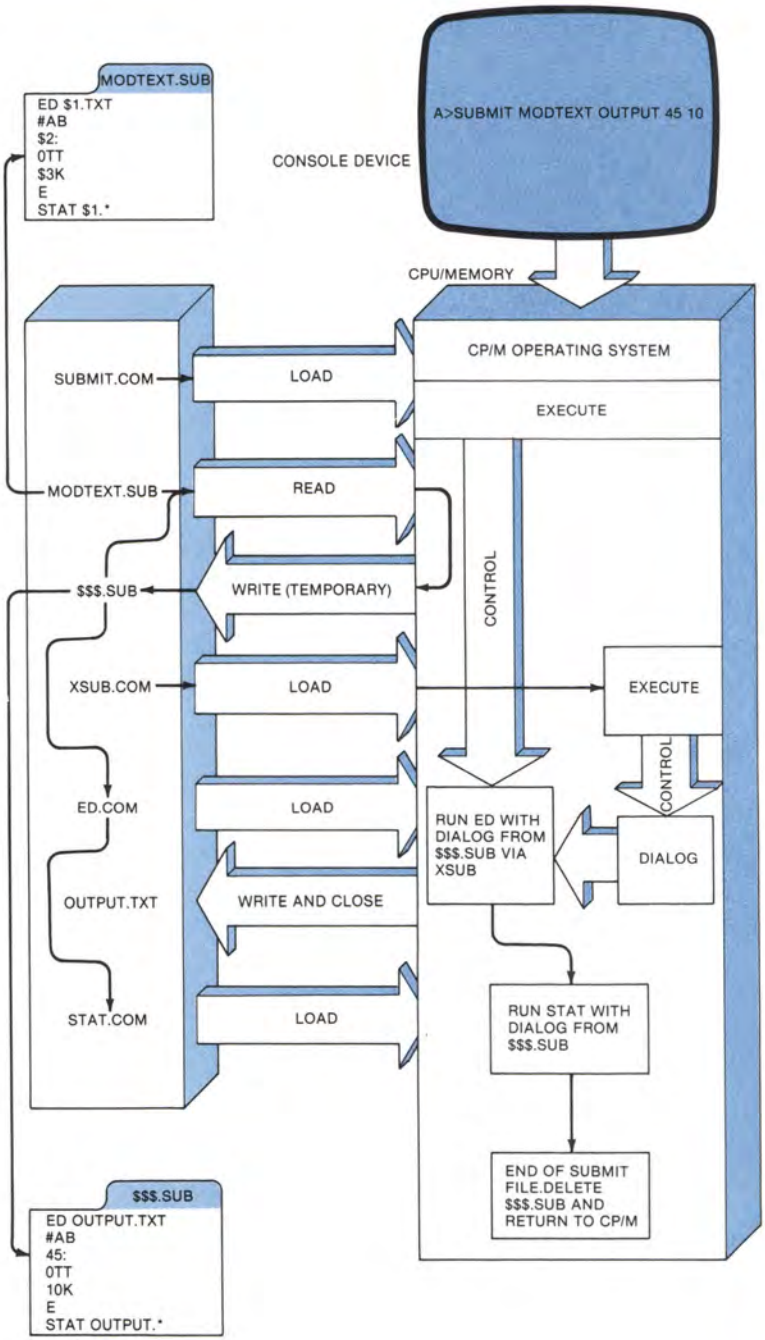
Command Type:
Transient

SUMMARY

The XSUB (**Ex**tended **SUB**MIT) utility is a special subset program used in conjunction with the SUBMIT program. The XSUB allows the user of SUBMIT to increase automation greatly when submitting commands and programs to the CP/M command line. A good understanding of the SUBMIT program is desirable before attempting to use XSUB.

The XSUB utility allows the creation of SUBMIT files (files with an extension of .SUB) so that they contain the command strings that will deliver certain CP/M commands and programs. SUBMIT files created with XSUB also contain text information ordinarily entered on the console device in response to program prompts. XSUB can be used with SUBMIT for a variety of applications, such as modifying an existing BASIC program, running it, submitting input to it automatically while it is running, modifying the program again in a different manner for subsequent execution, etc.

The next figure demonstrates the concept behind SUBMIT with XSUB. It shows how to use the command for sending commands or entries to application programs currently being submitted.


MODTEXT.SUB
ED $1.TXT
#AB
$2:
0TT
$3K
E
STAT $1.*
A>SUBMIT MODTEXT OUTPUT 45 10
CONSOLE DEVICE
CPU/MEMORY
CP/M OPERATING SYSTEM
EXECUTE
SUBMIT.COM
LOAD
MODTEXT.SUB
READ
$$$.SUB
WRITE (TEMPORARY)
CONTROL
XSUB.COM
LOAD
EXECUTE
CONTROL
ED.COM
LOAD
RUN ED WITH DIALOG FROM $$$.SUB VIA XSUB
DIALOG
OUTPUT.TXT
WRITE AND CLOSE
STAT.COM
LOAD
RUN STAT WITH DIALOG FROM $$$.SUB
END OF SUBMIT FILE.DELETE $$$.SUB AND RETURN TO CP/M
$$$.SUB
ED OUTPUT.TXT
#AB
45:
0TT
10K
E
STAT OUTPUT.*

PROGRAM ACCESS AND OPERATION

To use XSUB, both the SUBMIT.COM and the XSUB.COM files must exist on one of the disks currently inserted in a drive. Both programs are used together, but XSUB is not entered as a command by itself. Instead, the string "XSUB" is inserted in the .SUB file's first line of text. When the .SUB file is submitted, the XSUB.COM utility will also be "submitted" to a special area in the computer's memory (RAM). When this is done, special features will be added to SUBMIT, and a special buffer area will be established in RAM to permit the proper transfer of the submitted text from the .SUB file.

Once XSUB has been loaded into memory, it need not be executed again, unless the system requires a Cold Boot. Entering the CTRL/C (Warm Boot) sequence does not erase XSUB from the system's memory.

USING XSUB

When using XSUB in a .SUB file, only provide input text to the programs to be submitted *if* normally there is a way to exit these programs by means of a specific command string. In other words, those programs (such as PIP) that require the pressing of RETURN or CTRL/C (Warm Boot) to exit should *not* be used with the XSUB special features. This is because SUBMIT does not recognize blank lines (those containing only the ASCII Carriage Return and Line Feed characters) in the .SUB file. SUBMIT will treat these lines as errors. Lines containing the ^C character (ASCII ETX, i.e., 03 hexadecimal) used in an attempt to represent the CTRL/C (Warm Boot) sequence, will also be treated as errors. Therefore, if you want SUBMIT to exit from the program automatically, only use programs which require specific exit commands (such as "E" when using ED), or programs that exit immediately after typing the character input to the last prompt. Of course, you always have the option to enter the exit sequence manually on the console device.

The following example shows the contents of a file that uses the XSUB utility.

```
A>TYPE MODTEXT.SUB

XSUB
ED $1.TXT
#AB
```

```
$2:
OTT
$3K
E
STAT $1.*

A>
```

This example shows that the XSUB utility is to be used with the ED program. When the "MODTEXT.SUB" file is submitted, first XSUB will be loaded. The "ED $1.TXT" string will be submitted then with "$1" being substituted for the first variable text string. ED will open the specified file, and as soon as the ":*" prompt is displayed, features of XSUB will be set in motion.

The next line of the file contains a group of ED commands, and these will be submitted to ED as if you had typed them yourself on the console device. The subsequent lines also consist of ED commands, except the last one ("STAT $1.*"), which is a command that must be executed in the CP/M command mode. This is no problem since the previous line ("E") instructs ED to terminate (exit) the editing session and return to the CP/M command mode. In this way, "STAT $1.*" can execute properly.

The next example demonstrates what will be seen on the console device when this file is submitted.

```
A>SUBMIT MODTEXT OUTPUT 45 10

A>XSUB

A>ED OUTPUT.TXT

     : * #AB
    1: * 45:
   45: * OTT
   45: * From this line, kill the next 10_
   45: * 10K
   45: * E

(xsub active)

A>STAT OUTPUT.*
```

```
Recs  Bytes   Ext Acc
 110    14K     1 R/W A:OUTPUT.BAK
 109    13K     1 R/W A:OUTPUT.TXT
Bytes Remaining On A:  476K

A>
```


The example shows that when the file was submitted, the ED program was accessed normally. In addition, all of the ED commands in the file were submitted and subsequently executed successfully by ED. Once the "E" (Exit) command was submitted and executed, the ED program terminated. The message "(xsub active)" was displayed to indicate that SUBMIT had returned to the CP/M command mode and was ready to submit another CP/M command. Since XSUB is still active at this point, the XSUB features could be utilized for further input for other programs being submitted.

There are many different applications in which XSUB can be used with SUBMIT. Don't, however, expect your .SUB files to work as intended the first time you try them. These kinds of files often have to be modified and remodified to get them to work properly. For example, despite our information about which types of programs can and cannot be used with XSUB when automating the exit routine, you may encounter a program that ought to work, but won't. It's a good idea to experiment with SUBMIT and XSUB on a scratch disk (one that doesn't contain any significant files) to find out what you can and cannot do.

Chapter 13



CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

Transient

SUMMARY

ASM, the CP/M assembler program, allows source code program files in text format to be assembled into a 8080-microprocessor hexadecimal code program files. These hex program files are nonexecutable and must first be converted to a second executable .COM file with a utility called LOAD.COM*. Here is how the assembler works. The source code text files are created using the CP/M ED text editor or another CP/M compatible text editor or word processor. The text in the source file contains 8080-microprocessor assembly language statements. These statements are arranged to form a desired 8080 program. The origin of the program, the amount of space set aside for storage of variables, the names of special zero page locations, and more are specified in the text file. The ASM utility is then used to scan the text file and convert all the mnemonics into their respective hex codes and to assemble the results into an output file in hexadecimal format. This hexadecimal file can later be loaded into executable machine language format using the LOAD utility.

---

*The reason the Digital Research ASM assembler doesn't output pure object code suitable for immediate execution is that early computers used paper tape to load in programs to the computer. The paper tape was formatted in a special block manner so mistakes in reading it could be detected. This is the hexadecimal format that ASM still produces. Note, too, that .HEX files can be read into memory with DDT.

The ASM command performs the following three basic functions:

1. ASM reads the contents of the 8080 source-code file (FILENAME.ASM), assembles the code, and places the results in a hexadecimal output file (FILENAME.HEX). This hexadecimal file is then ready to be loaded into an executable machine-code file using the LOAD utility. The original source-code file is left unchanged.
2. In addition to creating the hexadecimal output file, ASM also creates another output file, a special version of the original source file. This special output file is called the "print" or "listing" file (FILENAME.PRN). It contains the same text as the source file in addition to the specific assembled hexadecimal code for each line of text code. This hexadecimal code is placed in the left-hand margin of the file.
3. During the assembly process, ASM lists any errors it encounters. These errors are listed on the console device (while ASM is executing), and in the output print file (FILENAME.PRN). The errors are presented in the form of symbols, and are placed on the offending lines of text code in the print file.

The CP/M ASM assembler will only assemble programs that are written in 8080-microprocessor assembly language code. Even though there are many 8085 and Z80 microprocessor-based computers using CP/M, you should not attempt to assemble programs with ASM that were written especially for operation on the Z80. The 8085 contains only one instruction that is different from the 8080, namely serial input and output, so usually all 8080 and 8085 programs are compatible. In all cases, 8080 machine-code programs will also run on an 8085 or Z80, but programs written **specifically** for the 8085 or Z80 will not run on an 8080. The ASM program permits the programmer to transport 8080 programs to 8085 and Z80-based computers without having to worry about code compatibility.

The description of ASM in this book only deals with the operation of the ASM program itself. It does not contain information about how to write 8080 assembly language programs. All assembly language programming statements, commands, and functions specifically related to using ASM are covered. Consequently, assembly language programmers can use this information to make sure that their programs will be assembled correctly using ASM.

PROGRAM ACCESS

To use ASM, the file ASM.COM must be stored on one of the disks inserted in a drive. The syntax of the ASM command string is different from other CP/M programs. The source-code file to be assembled can be on a disk in any disk drive, but it *must* have an extension or file type of ".ASM". If the source-code file is on the same disk drive as ASM.COM, simply type the following:

A>ASM FILENAME

. . .where "FILENAME" is the name of the source file (FILENAME.ASM). Do *not* type the extension ".ASM", as the ASM program will automatically search the disk for the file specified, assuming it must have an .ASM extension. If the file is found and contains the correct extension, ASM starts reading it and begins assembling the source code into hex code. When ASM is invoked in this manner, the hexadecimal output file (FILENAME.HEX), and the output print file (FILENAME.PRN) will automatically be output and stored on the logged disk. Both the .HEX and .PRN files will have the same file name as the original specified source-code file.

USING ASM

The ASM command string can also be entered with special optional parameters that determine the drives where files are to be retrieved and stored. These parameters consist of three individual characters entered in the ASM command string after a period (".") Because of this placement, they look like they form the extension of the source file, even though they don't.

The command string is entered as follows:

A>ASM FILENAME.123

. . . where the "1" character position is the place you would enter the name of the drive name (A, B, C, etc.) containing the disk with the source-code file (FILENAME.ASM). The "2" character position is the position you would enter the drive name where you wanted the output hexadecimal file to be stored (FILENAME.HEX). The "3" character position is the drive name to be stored with the output print file (FILENAME.PRN). Enter single letters only, without the colon (":") character. Any combination of drive names can

be specified in the optional parameters field. These specified drives must be installed in the computer and contain disks. Their names cannot exceed what's permitted by the version of CP/M you are using (A through B in CP/M Version 1.3, A through D in Version 1.4, and A through P in Version 2.0 and newer). For example:

A>ASM PROG-1.ABB

. . . instructs ASM to retrieve the source file PROG-1.ASM from drive A, and output both the .HEX and .PRN files to drive B.

There are two special optional symbols, "Z" and "X", that can be entered in the optional parameters field. If you want to suppress the output of either the .HEX or the .PRN files, or both, place the "Z" character in either the second or third character position. The effect will be the same if you place "Z" in both positions. Place the "X" character in the third character position of the parameters field if you want the contents of the print .PRN file to be displayed on the console device without actually creating the output .PRN file.

ASSEMBLING A FILE ON THE DEFAULT DRIVE



Compared to the writing of source-code programs, the operation of the ASM program itself is relatively simple. When the command string "ASM FILENAME" is entered, (where "FILENAME" is the .ASM file you want assembled), ASM will assume that the specified file is on the default drive. Once the command specifying a particular source-code file has been entered, ASM displays the name and version of the program on the console device. Output to the console pauses for a short time while the code is being assembled. If any errors are encountered during the assembly process, information to the effect will be displayed as the code is being assembled. When the assembly process is complete, the output hexadecimal file (FILENAME.HEX) and the output print file (FILENAME.PRN) is stored on the default drive.

The next example shows displayed information regarding the first free (unused) address following the assembled program. It also indicates the total percentage of the ASM symbol table used to assemble the program.

```
A>ASM TEST01

CP/M ASSEMBLER - VER X.X
020F
006H USE FACTOR
END OF ASSEMBLY

A>
```


This example shows that the assembly operation of source file "TEST01.ASM" was successful. We can deduce this from the fact that no assembly errors were displayed. The "X.X" on the first line is where the version of ASM will be displayed. This version will usually be the same as the version of CP/M (1.4, 2.0, etc.). On the next line, the first free (unused) address is shown to be "020F". This is a hexadecimal number, and represents the address immediately following the last address used by the assembled program. Subtracting 100 hex from this tells you how long the program is. On the next line, the message "006H USE FACTOR" indicates that 6 percent (hexadecimal) of the ASM symbols table was used. The ASM symbols table is an area used to store the values of labels from the source program. (The labels may contain addresses of special locations in the program, or data that is used by the program). This area is allocated by ASM during the assembly process. The percentage value given here will depend on the size of the program being assembled and the number of label values that need to be temporarily stored. This symbols table area has a maximum size, and should not be filled to capacity, if the assembly process is to be successful.

Once the assembly process is complete and ASM has terminated, verify that all relevant files are stored on the disk. There should be three main files pertaining to your program. This is shown as follows.

```
A>DIR TEST01.*

A: TEST01  BAK : TEST01  ASM : TEST01  HEX
A: TEST01  PRN

A>
```

The .BAK (backup) file will be there if it was not deleted after the last editing session with the .ASM file. If you examine the .ASM file, you will note that the contents are unchanged after the assembly process.

The .HEX file is the file you will use later for loading to machine code form, and debugging purposes, if necessary.

The .PRN is essentially a copy of the .ASM file. It includes the assembled code to left of the text. Each assembled code shares a line with its corresponding text code.

At this point, you can examine the three main files to verify the contents.

The next three examples show what the contents would look like for the source .ASM, assembled .HEX, and print .PRN files of the example program "TEST01".

THE .ASM FILE:

```
A>TYPE TEST01.ASM
;       FIRST TEST PROGRAM IN THE CP/M (8080) ASSEMBLY
;       LANGUAGE. START AT THE BEGINNING OF THE TRANSIENT
;       PROGRAM AREA.
        ORG 100H
SORT:   LXI  H, SW      ;ADDRESS SWITCH TOGGLE
        MVI  M, 1       ;SET TO 1 FOR FIRST ITERATION
        LXI  H, 1       ;ADDRESS INDEX
        MVI  H, 0       ;I = 0

        (etc.)

        END
A>
```

THE .HEX FILE:

```
A>TYPE TEST01.HEX
: 10010000214601360121470136007EFE09D2190140
: 100110002146017EB7C20001FF5F16002148011983
: 10012000194E79234623965778239EDA3F01B2CAA7
: 100130003F0156702B5E712B722B7321460134210C7
```

```
: 0701400047701340A01006E
: 04015800640001B0BE
: 0000000000
A>
```

THE .PRN FILE:


```
A>TYPE TEST01.PRN
                    ;       FIRST TEST PROGRAM IN THE CP/M
                    ;       (80 LANGUAGE. START AT THE
                    ;       BEGINNING O PROGRAM AREA.
0100                        ORG  100H
0100 214601         SORT:   LXI  H, SW      ;ADDRESS SWITCH
                                             TOG
0103 3601                   MVI  M, 1       ;SET TO 1 FOR
                                             FIRST
0105 214701                 LXI  H, 1       ;ADDRESS INDEX
0100 3600                   MVI  H, 0       ;I = 0
        (etc.)              END
020F                  D
A>
```

At first glance, the contents of the second example .HEX file (a series of hexadecimal numbers), looks pretty much unintelligible. It can, of course, be deciphered but it is much easier to look at the third example .PRN file to determine how the hexadecimal code corresponds to the source text code. Normally, the .PRN file will be used for verification and for the correlation of any debugging routines with the original source code. Note that all source-code text strings beginning with a semicolon (";") are treated as comment lines by ASM, rather than actual code to be assembled. (See the .PRN file.)

Note that the original text of the .ASM file in the .PRN file is shifted to the right sixteen spaces, to make room for the assembled code. These spaces translate to two ASCII horizontal tabs (HT) (an ASCII tab usually has a default of 8 spaces). If you are going to output the .PRN file on the computer's printing device, you may find that the right-hand margin will exceed the width of the paper in the printer. (It most definitely will, if the printer handles only 80 characters across.) To alleviate this, set the printer to 132-column mode, if it has this capability. Otherwise, edit the original .ASM file to shorten the width of the text by 16 spaces so that the .PRN file is output

correctly next time you use ASM. Or be willing to accept that part of the .PRN file will be printed on the perforated right edge of the paper, unless the printer can wrap text to the next line on every line.

ASM AND CP/M CONSOLE ERROR MESSAGES

When using the ASM program to assemble a program, there are various error messages that can be displayed both by the ASM program and the CP/M operating system. These messages are defined as follows:

NO SOURCE FILE PRESENT This message will be displayed after you type in an ASM command string specifying a source code .ASM file that could not be found on the specified disk. The problem could be caused by the file not existing on the default drive (if specified as such), or on the specified drive (when using the optional parameters). The error could also be caused by the specified file lacking the extension ".ASM". This message will also be presented if the command "ASM" is entered alone, without specifying a source file.

SOURCE FILE NAME ERROR This error message will display if the source file name was specified incorrectly. It will occur, for example, if the CP/M wildcard characters "*" or "?" were specified, as they are not allowed in an ASM command string.

SOURCE FILE READ ERROR This message will appear if ASM encounters illegal characters in the specified source file during the reading and assembly process. A character is considered illegal and uninterpretable by ASM if it is a nonprintable character. Nonprintable characters include control and escape sequences that were typed using the CTRL or the ESC keys. These characters could have been inserted in the file either intentionally, or accidentally when copying, concatenating, or improperly saving files. You can't always use the TYPE command to find out where these characters occur in the text. When text with these characters is displayed, the screen or the print head of the console device will probably do unusual things. Once this happens, you will know approximately where some of these characters are located. Use the text editor to eliminate them. In other cases, these characters will not show up on the CRT when TYPE is used. Only the editor or DDT could be used to find them.

OUTPUT FILE WRITE ERROR This error message will occur if ASM attempts to write the output .HEX and .PRN files to a write-protected disk or disks. It will also occur if there is no free disk space available. You should either un-write-protect the disk(s) or make sufficient room (by copying or deleting some of the existing files) before the ASM command string is entered again.

CANNOT CLOSE FILE This error message will display if CP/M locates the file specified by ASM, but cannot write any data to it because of a write protection status. Un-write-protect the disk before the ASM command string is entered again.

NO DIRECTORY SPACE This message will display if ASM did not encounter a sufficient number of free directory entries on the specified disk to store the output .HEX and .PRN files. This error is not the same as having insufficient free space on the disk, but refers to actual directory entries. (Normally, the CP/M operating system can contain up to 64 directory entries on a particular disk.) To alleviate this problem, replace the specified disk or disks with another that contains sufficient free directory entries. You could also copy and delete some of the existing files on the present disks to allocate room in the directory area.

SOURCE CODE PROGRAM (.ASM) ERROR MESSAGES

If, during the assembling process, the ASM program detects errors in the source-code program file (FILENAME.ASM), they will be listed as they occur on the console device before the process is completed. These errors consist of alpha character error codes. These codes will also be listed in the output print .PRN file, on the appropriate lines of text code.

The format in which ASM displays these error codes is as follows:

```
C HEX CODE  LABEL MNEMONIC OPERAND ;COMMENT
L 0100 214601 SORT: LXI       H, SW    ;ADDRESS
                                       SWITCH TOG
```

. . . where "C" is the error code. The code is presented at the first character position of the line containing the error. It is followed by a space before the 4-digit hexadecimal address of the assembled code is presented. Following the address is the actual assembled hexadecimal code, which will usually con-

sist of all zeros depending on the seriousness of the error. Do not be deceived if this code contains a value greater than "0000". ASM will often attempt to assemble the code until it reaches the error condition. This code will usually indicate that the line of code was only partially assembled. These error codes provided by ASM are defined as follows:

D–DATA ERROR: This error will occur if the expression in the line of code will not fit into the indicated data area. This usually tells us that the expression is too long, and should, if possible, be shortened.

E–EXPRESSION ERROR: This error message means that the expression in the line of code was improperly formed, or the value is too long to fit into the specified data area.

L–LABEL ERROR: This error will be presented if the label was used incorrectly in the line of code. A common cause of this error is a label specified on more than one statement line.

N–FEATURE NOT YET IMPLEMENTED: This error code will be presented when ASM encounters a directive in the program being assembled that it knows about, but cannot do anything with. The most probable cause for this error is the source program containing macroassembler directives for use with Digital Research's MAC program. An indication is thus made that the line of code containing the directive can only be processed using MAC, as ASM does not have adequate capabilities.

O–OVERFLOW ERROR: This error will be presented when ASM has encountered a line of code containing an expression too complicated to be processed. To alleviate this problem, break down the expression into smaller parts. Alternatively, the number of operators could be reduced.

P–PHASE ERROR: This error will be presented if a label in the source code has its value changed in an incorrect manner. If the value of a label is to be changed, the SET directive should be used. This error message can also be presented if ASM encounters duplicate labels during the assembly process.

R–REGISTER ERROR: This error indicates that the specified register is not compatible with the mnemonic in the line of code. For example, if the expression POP A was given, a register error would result since this expression is not valid when using ASM with 8080 assembly language programs.

S–SYMBOL ERROR: This error indicates that an illegal character is present in the comments field of the statement line (for example, the "!" character).

U–UNDEFINED SYMBOL: This error message will occur if a label for which no value was assigned is found in an expression. If, for example, the expression consisted of "LXI CONT" and the label "CONT:" had not been defined, this error code will be presented on the line containing the expression.

V–VALUE ERROR: This error code is presented when ASM encounters an expression or operand that has not been correctly specified. This will usually be due to typographical errors, such as the absence of commas or other necessary specific characters.

ASM SOURCE PROGRAM FORMAT

Certain formatting considerations exist when writing 8080 source code programs that are to be assembled using ASM. The following is a summary of the formats required by ASM. These are not necessarily the same as the formats specified by other variations of the 8080 Assembly Language.

STATEMENTS: Each assembly language statement or line of code can consist of one to five fields. These fields are groups of characters separated from each other by spaces or tabs. The format of these statement fields is as follows:

```
LINE NUMBER  LABEL  MNEMONIC  OPERAND  ;COMMENT
0010         SORT:  LXI       H, SW    ;ADDRESS
                                         SWITCH TOG
```

The ASM program sets tab stops at every 8th column position along a statement line. This results in tab stops at columns 1, 9, 17, 25, 33, and so on.

Each of the five fields just shown are discussed in the following paragraphs.

1. **LINE NUMBERS:** Line numbers, consisting of integer decimal numbers, can be optionally inserted in statement lines. When inserted, they are positioned at the beginning of the line. These numbers only serve the purpose of reference for the programmer. They are ignored by ASM when assembling the program.

2. **LABELS:** The label is an identifier used to represent an address or a certain value. A label can be 1 to 16 characters long, and should be terminated by a colon (":"). A dollar sign ("$") can also be inserted in the label to improve source-code readability. ASM will ignore any $ characters it encounters in the label field. It will not consider it as part of the sixteen or less character positions. Labels are optional for all statement lines except for the EQU (Equate) and SET directives. These two must always be preceded by a label. There are several reserved words which cannot be placed in the label field, since they have predetermined meanings. They are

   a. All 8080 instruction mnemonics.
   b. All 8080 register names: A, B, C, D, E, H, L, M, SP, and PSW.
   c. All ASM directive names: ORG, EQU, SET, DB, DS, DW, IF, ENDIF, and END.

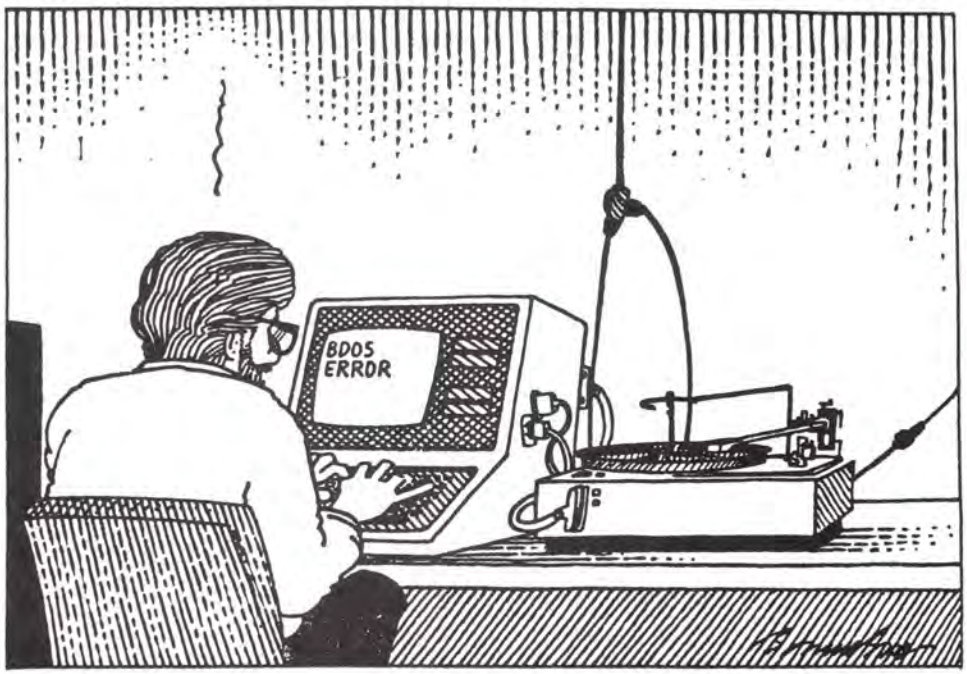
3. **MNEMONICS:** The mnemonic field must be included in every statement line. It is not optional. This field can contain either an 8080 instruction, or an ASM directive.

4. **OPERANDS:** The operand field will either be empty or contain one or two operands, according to the requirements of the statement line instruction. An operand can be a constant, a label, or an expression. In general, ASM directives require one or more operands in this field.

5. **COMMENTS:** The comment field is used by the programmer to place noncode text that describes the function of the statement line. This field is optional, but, if it is used, the text must always begin with a semicolon (";"). When ASM detects the presence of a semicolon on a statement line, it ignores the text after the semicolon and treats it as commentary.

ASM SOURCE PROGRAM CONSTANTS AND EXPRESSIONS

The following is a brief summary of how constants and expressions in source-code programs can be used with ASM.



CONSTANTS: There are two basic types of constants which can be placed in the operand field of statement lines. These consist of numeric and string constants. These are described next.

1. **NUMERIC CONSTANTS:** A numeric constant consists of a fixed number that is one of four number bases, as follows.

   a. **Binary (Base 2):** A binary numeric constant consists of a sequence of digits using "0" and "1", and must be followed by the character "B".

   b. **Octal (Base 8):** An octal numeric constant consists of a sequence of digits, each in the range of 0 through 7. The sequence must also be terminated the character "O" or "Q".

   c. **Decimal (Base 10):** A decimal numeric constant consists of a sequence of digits, each in the range of 0 through 9. The sequence can optionally be terminated by the character "D" for clarity. This is not necessary since ASM will automatically consider numeric constants not terminated by a number base character as a decimal number.

d. **Hexadecimal (Base 16):** A hexadecimal numeric constant consists of a series of digits and characters, each of which must be in the range of 0 through 9 and A through F. This type of constant must begin with a digit (0 through 9). The sequence must be terminated with the character "H". This terminator is optional if the first digit in the sequence is "0", in which case, ASM will automatically treat the constant as a hexadecimal number.

2. **STRING CONSTANTS:** A string constant is a series of characters enclosed between apostrophes (') i.e., single quotation marks. This type of constant must not exceed 64 characters in length, and only normal printable characters are allowed (no nonprintable control or escape sequences). If the apostrophe is to be included in this string, two apostrophes should be typed in sequence. This is done because ASM will ignore the first apostrophe and treat the next one as a literal character. Note that once a file is assembled ASM will not translate lower-case characters to upper case. ASM computes the value of a string constant by adding a high-order bit of "0" to the 7-bit ASCII code for each character.

EXPRESSIONS: ASM permits the use of expressions in the place of operands. An expression can be a combination of constants, labels, arithmetic operators, logical operators, and parentheses. While ASM is assembling an expression, the expression is evaluated and reduced to a single value. The ASM program performs these operations on 16-bit unsigned values, and produces 16-bit unsigned results. The following arithmetic and logical operators are supported by ASM.

1. **ARITHMETIC OPERATORS:** ASM can perform simple arithmetic when evaluating an expression to determine its value. The following arithmetic operators can be used to join labels and constants into expressions.

- Addition ("A+B" is equivalent to the sum of A and B).
- Subtraction ("A−B" means B is subtracted from A).
- Multiplication ("A*B" means A is multiplied by B).
- Division ("A/B" means A is divided by B, and "A MOD B" is the remainder when A is divided by B).
- Positive Numbers ("+A" is equivalent to "A").
- Negative Numbers ("−A" is equivalent to "0−A").

2. **LOGICAL OPERATORS:** The ASM program can also perform logical (Boolean) operations. The following logical operators are supported.

   - NOT ("NOT B" is the inverse or complement of "B").
   - AND ("A AND B" produces the logical AND of A and B).
   - OR ("A OR B" produces the logical OR of A and B).
   - XOR ("A XOR B" produces the Exclusive-OR of A and B).
   - SHL and SHR ("A SHL B" will shift-left A the number of bit positions defined by B. Shifted-out high-order bits are discarded, and vacated low-order bits are replaced with zeros. "A SHR B" will shift-right in the same manner).

ASM DIRECTIVES

There are several special instructions not a part of the standard 8080 assembly language instructions set, which may be used with the ASM program. These are referred to as "assembler directives." These directives are used to control the way in which ASM assembles a program and, thus, affects the resulting machine code.

The assembler directives are placed in the statement lines of the source-code program basically in the same manner as normal assembly language statements. A directive name is placed in the mnemonic field of the statement line. You may find that many of these assembler directives are used by other assembler programs. It is important, however, to know how ASM deals with the ones described in this section.

ORG The ORG (Origin) assembler directive provides ASM with the memory address used for the sequence of statements which follow. The ORG directive is placed in a statement line in the following format:

```
LABEL    ORG EXPRESSION    ;COMMENTS
```

The expression field is used to contain the memory address of the next instruction or directive. The optional label is assigned the value in the expression field for later usage. More than one ORG directive may be used in a program.

EQU The EQU (Equate) assembler directive is used to assign a value or expression to a given label. This directive is placed in a statement line in the following format.

```
LABEL    EQU EXPRESSION    ;COMMENTS
```

The expression may be any valid number, address, constant or expression. Both the label and expression fields are required in a statement using the EQU directive. Other previously defined labels can be placed in the expression field together with operators and numeric values.

SET The SET directive permits the setting or assigning of a given label with a specific value. This directive can also be used to reassign the value of a label (EQU can set a label to a particular value only once). The SET directive may be used several times on the same label. This directive is placed in the statement line as follows.

```
LABEL    SET EXPRESSION    ;COMMENTS
```

In a statement line containing this directive, both the label and the expression fields must be present.

IF and ENDIF

The IF and ENDIF (End IF) directives are used to start and end a section of the program to be assembled only if the condition specified in the IF statement line is true. The IF statement line is the first line of the program's conditional section. The ENDIF statement line is the last. The format in which these are entered is as follows:

```
IF EXPRESSION
.
(conditional assembly language statement lines)
.
ENDIF
```

The conditions under which the IF statement is to operate are entered in the expression field. If the expression field contains all zeros, then ASM will ignore all statement lines between the IF and ENDIF statements. If the expression field contains a value greater than zero, the conditional statement lines will be assembled.

DB, DW, and DS

There are three directives that are used to establish storage areas in memory. These are.

DB– The DB (Define Byte) directive is used to establish an area in memory on a byte-by-byte basis. The format of the statement line containing this directive is as follows:

```
LABEL    DB    EXP1,EXP2,EXP3    ;COMMENTS
```

The "EXP1,EXP2,EXP3" field is the expression field, which may consist of any number of expressions, interspersed with commas. When this statement line is assembled, ASM evaluates the contents and stores them as 8-bit (1 byte) values in successive memory locations. Both the label and comments fields are optional in a DB statement.

DW– The DW (Define Word) directive is used to establish an area in memory, 1 word (2 bytes) at a time. The format of the statement line containing this directive is as follows:

```
LABEL    DW    EXP1,EXP2,EXP3    ;COMMENTS
```

The "EXP1,EXP2,EXP3" field is the expression field, which may consist of any number of expressions, interspersed with commas. When this statement line is assembled, ASM evaluates the contents and stores them as 16-bit (1-word or 2-byte) values in successive pairs of memory locations. Within each pair, the least significant byte is stored first, followed by the most significant byte. Both the label and comments fields are optional in a DW statement.

DS– The DS (Define Storage) directive is used to reserve an area of in memory of specified size. The format of the statement line containing this directive is as follows:

```
LABEL    DS    EXP1    ;COMMENTS
```

The "EXP1" field is a single expression, which consists of a value that defines the amount of memory to be reserved. Both the label and comments fields are optional in a DS statement.

The expression fields of statements using the DB, DW, and DS directives may consist of any valid expressions, labels, numeric or string constants, etc. Both the DB and DW directives may have any number of expressions, providing they are interspersed by commas. The DS directive may only have one expression.

This concludes the description on the ASM program. For more detailed information about 8080 assembly language programming, reference should be made to books on the subject. Some of the available publications are listed in Appendix D of this book.

Chapter 14

LOAD

CP/M Versions:

1.3 **1.4** **2.0** **2.2**

Command Type:

Transient

SUMMARY

LOAD is a CP/M utility that allows ASM and DDT program users to convert hexadecimal format (.HEX extensions) program files created by ASM into executable command machine-code files with the extension .COM.

PROGRAM ACCESS

To access the LOAD program, the file LOAD.COM must be stored on a disk inserted in one of the computer's drives. The file to be loaded must have the "LOAD" in response to the logged disk drive name (A>, B>, etc.), a space, followed by the filespec. Do not include the extension .HEX in the filespec, as the LOAD program will automatically search for the specified file name and read it if it had the correct extension.

Once the command is entered, the LOAD program displays information regarding the specified file. The following example shows how DIR is used to verify the existence of a file a disk. Also demonstrated is the use of the LOAD command to load the .HEX file and thus create the .COM file.

```
A>DIR TEST01.*
A : TEST01   ASM : TEST01   HEX
A : TEST01   PRN

A>
A>LOAD TEST01

FIRST ADDRESS 0100
LAST  ADDRESS 01F8
BYTES READ    00F9
RECORDS WRITTEN 02


A>
A>DIR TEST01.*
A : TEST01   ASM : TEST01   HEX
A : TEST01   PRN : TEST01   COM

A>
```

This example shows the loading of the file "TEST01". Before the LOAD command was specified, the DIR command was used to verify that all the necessary files of TEST01 were on the disk. After the LOAD command was used and the program was terminated, DIR was employed again to verify that the file TEST01.COM had been successfully stored on the disk. The information displayed by LOAD during program execution is defined as follows.

FIRST ADDRESS: The first address location of the program being loaded.

LAST ADDRESS: The last address location of the program being loaded.

BYTES READ: The total number of bytes in the program being loaded.

RECORDS WRITTEN: The total number of 128-byte records written into the program being loaded.

Note that the code in the .HEX file must consist of records with addresses in *ascending order*. If this is not the case, the .COM file created by LOAD will be incorrect.

NOTE TO CP/M PROGRAMMERS:

When loading assembly language programs using the LOAD command, make sure that the program is ORGed to 0100 hexadecimal, or above. Otherwise an error condition will occur when the LOAD program attempts to load the program. The ORG function is used to instruct CP/M as to the program's location in memory. When it is loaded into memory and executed, it is to reside and begin execution. 0100H is normally the starting location of the Transient Program Area (TPA) portion of memory.

Chapter 15

DDT

CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

Transient

SUMMARY

The DDT program is a CP/M utility which, among other things, permits assembled machine code programs to be viewed, tested, altered, debugged, and loaded into memory. In addition, DDT allows you to view text files of any type. The DDT program operates in a dynamic manner, allowing the user to run a program in memory and view each step of execution. It can be used for making small patches to existing programs (like WordStar) or for disassembling programs to determine how they work. This is not a simple utility. The DDT program is essentially an important assembly language programmer's tool. A good understanding of assembly language programming will help in effectively using this program. Also, DDT has a built-in mini-assembler that works like a stripped down version of the ASM assembler. The book *Soul of CP/M* uses DDT to teach you assembly language programming and modifying CP/M's BIOS.

Here is a summary of the features provided by DDT.

- Fill the memory with constant data, i.e., put certain bytes in consecutive memory locations.
- Establish a default file control block (FCB) by specifying a file name, and read the file named in the FCB into memory.
- Display the contents of portions of memory in both hexadecimal and ASCII formats.

- Display certain portions of memory containing object code as 8080 assembly language mnemonics by disassembling the code to ASCII format.
- Display the hexadecimal contents of a particular address and optionally change the contents.
- Insert assembly language statements anywhere in memory.
- Move specific blocks of memory to another location.
- Execute portions of memory containing programs, set breakpoints (places for the program to halt and return to DDT), and set optional stops to display the registers of the CPU.
- Trace (follow step by step) the execution of a number of program instructions or an entire program.
- Display the current contents of the CPU registers, and optionally make changes.
- Enable the built-in calculator for adding and subtracting hexadecimal numbers.

To execute the various functions mentioned, several commands are provided which are entered while DDT is resident in memory. The following paragraphs describe how DDT is accessed and used.

PROGRAM ACCESS AND DESCRIPTION

To execute the DDT command, the file DDT.COM must be stored on a disk inserted in one of the computer drives. When the command is issued, DDT is loaded into the Console Command Processor (CCP) section of RAM, replacing what is normally there. DDT has unique operating characteristics as compared to other CP/M commands. Even though DDT is a "Transient Program", it is not loaded into the normal TPA at 100 hex. Rather, it is loaded above the TPA so that it is sitting in and overlaying CP/M's CCP area of memory. The Transient Program Area can then be used for holding or running the user program or file to be "debugged" by DDT. When a machine code file is specified to be loaded into memory, the file is placed in the TPA, where it can be supervised by DDT for code displaying and possible modification.

When a file is specified for loading into memory, DDT will "overlay" the contents of the TPA with the contents of the specified file. Because of these unusual operating characteristics, a CTRL/C command (Warm Boot) or the DDT "G0" command (described under the "G" command description) must be issued to exit DDT. In this way, the normal contents of the CCP can be reloaded (rebooted) from the disk into memory.

The following describes two ways in which DDT can be accessed and loaded into memory.

LOADING DDT INTO MEMORY WITHOUT SPECIFYING A FILE FOR EXAMINATION AND MODIFICATION



The DDT program can be loaded into memory without specifying a file for examination and modification. This is accomplished by simply typing "DDT" in response to the logged drive prompt (A>, B>, etc.) and pressing the RETURN key. This will load DDT into the CCP portion of memory, and the TPA will retain the same contents it had prior to issuing the command. See the next example.


```
A>DDT
DDT VER X.X
-
```

Once the program has been loaded, a sign-on message indicating the program name and the version, as shown by "X.X", will be displayed. The version number will be that of DDT, although it usually reflects the version number of CP/M itself (1.4, 2.2, etc.).

<( )> Special Attention—Other DDT Sign-Ons

In some special versions of CP/M, DDT has been modified for the environment in which it is to operate. This modification is sometimes indicated by a character following the word "DDT", displayed in the form of "DDT-X" where "X" can be one of the following:

D	Digital Research standard version
M	Intel MDS version
I	IMSAI standard version
O	Omcron Systems version
S	Digital Systems standard version

Once DDT has been loaded into memory, the dash character "-", which is the DDT prompt, will be displayed (as seen following the "Version" line in

the previous example). The DDT commands are entered in response to this prompt.

Once in memory and running the I (insert name) and R (read file) commands can be used to load in a file from the disk into memory. Basically, the I and R commands let you set up the FCB so you can easily load files in at any time.

LOADING DDT INTO MEMORY BY SPECIFYING A FILE FOR EXAMINATION AND MODIFICATION

```
DDT FILENAME.COM
DDT FILENAME.HEX
DDT FILENAME.EXT
```

Instead of issuing a sequence of commands to load a file into memory after DDT has been loaded, you can load DDT and the specified file at the same time. This is done by typing "DDT", and a space, followed by the file name and its extension. Files that are normally loaded into memory with DDT are ".COM" (transient program command) files, or ".HEX" files (files containing hexadecimal code), but .TXT text files may be examined too. The .HEX and .COM files always contain machine code, and thus DDT provides a means by which they can both be examined and modified. Any other machine code file or text file can also be loaded into memory with DDT. These include data files (FILE.DAT or other), overlay files (FILE.OVR, FILE.OVL, or others), document files (whatever.TXT), etc.

When DDT is loaded into memory with the filename of a file after it, it will display information regarding the memory status of the specified file loaded into the TPA. The following example demonstrates this.

```
A>DDT TEST01.COM
DDT VER X.X
NEXT PC
1E00 0100
-
```

After the initial sign-on message has been presented, certain information is displayed regarding the memory status at the location where the file

"TEST01.COM" was loaded. In our example, the number under the word "NEXT" indicates the "next" free memory location (after the loaded program) in the TPA. In other words, it is the first free (unused) location in memory after the last location used by the loaded program. The number beneath the abbreviation "PC" (short for "Program Counter") represents the current DDT memory location setting. This value informs you of the place in memory at which DDT may begin data examination and modification. This "place" is an address in memory and corresponds to ED's cursor in many ways. DDT uses the PC as a reference point, so that memory locations after the PC setting can be examined.

The remaining part of this section describes the various commands that may be used with DDT.

COMMAND DESCRIPTIONS

DISPLAYING THE CONTENTS OF PORTIONS OF MEMORY

- D
- Dstart
- Dstart,end

The D (Display) command is used to display the contents of a portion of memory on the screen in hexadecimal and ASCII formats. This is often the first command typed when DDT has just been loaded into memory with a file and the user wishes to look at the values in RAM. When the D command is typed by itself, it will display the next 192 bytes of data starting from the location setting in the program counter (PC). This is shown in the next example. Here the PC is set to 100 hex, the default value when DDT is first loaded into memory.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT PC
0200 0100
-D
```

```
0100 21 00 00 39 22 FA 01 31 3C 02 AF 32 F9 01 0E 0F !..9"..1<..2
0110 11 5C 00 CD 05 00 11 D5 01 3C CA 77 01 01 00 00 .\.......<.w
0120 C5 0E 14 11 5C 00 CD 05 00 C1 B7 C2 67 01 21 80 ....\.......
0130 00 7E E6 7F FE 1A CA 67 01 FE 20 CA 59 01 FE 0D .~.....g.. .
0140 CA 59 01 FE 0A CA 59 01 3A F9 01 FE 00 CA 5E 01 .Y....Y.:.....
0150 03 3E 00 32 F9 01 C3 5E 01 3E 01 32 F9 01 23 7C .>.2...^.>.2
0160 B7 CA 31 01 C3 20 01 11 5C 00 C5 0E 10 CD 05 00 ..1.. ..\....
0170 C1 CD 81 01 11 E5 01 0E 09 CD 05 00 2A FA 01 F9 ............
0180 C9 C5 E1 06 00 11 F0 D8 CD A5 01 11 18 FC CD A5 ............
0190 01 11 9C FF CD A5 01 11 F6 FF CD A5 01 7D C6 30 ............
01A0 4F CD C8 01 C9 0E 2F 0C 19 DA A7 01 7A 2F 57 7B O...../.....
01B0 2F 5F 13 19 79 FE 31 D2 C2 01 78 B7 79 C8 CD C8 \_..y.1...x.
-
```

After the D command was typed in this example, the contents of the first 192 locations were displayed in hexadecimal. The numbers in the far left column represent the starting locations of each line. After the starting location number on each line, there follows 16 numbers, each of which represents the byte or data value of each location on that line. For example, the number next to "0100" on the first line is "21", which is the data value corresponding to location 0100. The locations occur in order from left to right. For example, the first line of text starts with location 0100, as shown in the far left column. The location numbers increase as you move to the right, producing the range of locations 0100 through 010F, hexadecimal (16 locations).

In the far right section of the display, a series of alpha-numeric characters interspersed with periods appears. This is the field where ASCII text corresponding to the hex data will be displayed. The first character position in this field is the ASCII representation of the data in the first location on the corresponding line. The second character position represents the data in the second location on the same line, and so on. Only printable ASCII characters will be displayed in this field. These characters range from 32 through 7E hexadecimal. Any lower number (00 through 31) or any higher number (7F through FF) will cause a period (".") to be displayed in the corresponding character position. See Appendix F for more information on the range and definitions of ASCII codes. Usually non-ASCII values shown as periods correspond to program codes, where the full 8-bits is used. A section of text loaded into DDT, for example, will show the ASCII area loaded with text characters, while a preponderance of periods signifies that program information is being viewed.

There are other ways in which the D command can be used. Instead of specifying the display of the next 192 memory locations from the program counter (PC) setting, you can specify that the next 192 locations be displayed

from the starting location of your choice. By entering "D" immediately followed by the starting memory location, the next 192 locations after this location will be displayed. The specified memory location can be any one existing in the system's memory from 0000 to a maximum of FFFF. This is shown in the next example.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT PC
0200 0100
-D0130
0130 00 7E E6 7F FE 1A CA 67 01 FE 20 CA 59 01 FE 0D .~.....g.. .Y...
0140 CA 59 01 FE 0A CA 59 01 3A F9 01 FE 00 CA 5E 01 .Y....Y.:.....^.
  :
  :
  V
01E0 2F 5F 13 19 79 FE 31 D2 C2 01 78 B7 79 C8 CD C8 /_..y.1...x.y...
-
```


The D command can also be used to display a portion of memory within a specified starting location and an ending location. This form of the command is similar to the one described previously. The difference is that a comma and the ending location is entered in the command string after the starting location. This is shown in the following example.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT PC
0200 0100
-D0100,012FF
0100 21 00 00 39 22 FA 01 31 3C 02 AF 32 F9 01 0E 0F !..9"..1<..2....
0110 11 5C 00 CD 05 00 11 D5 01 3C CA 77 01 01 00 00 .\.......<.w....
0120 C5 0E 14 11 5C 00 CD 05 00 C1 B7 C2 67 01 21 80 ....\.......g.!.
-D0120,012F
0120 C5 0E 14 11 5C 00 CD 05 00 C1 B7 C2 67 01 21 80 ....\.......g.!.
-D0120,012A
0120 C5 0E 14 11 5C 00 CD 05 00 C1 B7 ....\......
-D0120,0122
0120 C5 0E 14 ...
-
```

Note in this example that the first portion displayed consisted of the data in locations 0100 through 12FF. As these were the locations specified, only those locations were displayed. The program counter (PC) was set at the last specified location. The next display consisted of the data between locations 0120 through 012F. The third display showed the data in locations 0120 through 012A, revealing only the relevant sections of the previous line. When the sections are less than 16 addresses wide, the display is left justified, which results in the ASCII section sliding over to the left also.

Any section of the memory large or small, can be displayed by specifying this command string. If a large section of memory is specified, the CTRL/S command can be used to stop and resume scrolling.

FILLING THE MEMORY WITH CONSTANT DATA

–Fstart,end,data

The F (Fill) command is used to insert a specified data value in every memory location between a specified starting address and an ending address. This command is useful when DDT is loaded into memory without an accompanying file. By specifying different data each time the command is issued, parts of the memory system can be tested by filling them with certain data and then visually examining them for correctness. The D command (described previously) can be used after the Fill command to verify the contents of the specified memory locations.

The Fill command string is entered by first typing "F" in response to the DDT prompt. It is immediately followed by the hexadecimal starting address, a comma, the hexadecimal ending address, another comma, and the hexadecimal byte to be inserted. Once the command is entered, the specified data will overlay or replace any data previously contained in the specified memory locations.

The next example demonstrates the correct syntax for this command.

```
A>DDT
DDT VERS X.X
-F0100,01FF,26
-D0100,01FF
```

```
0100 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 &&&&&&&&&&&&&&&&
0110 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 &&&&&&&&&&&&&&&&
  :
  :
  :
  V
01E0 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 &&&&&&&&&&&&&&&&
01F0 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 &&&&&&&&&&&&&&&&
-
```

Any 2-digit hexadecimal code can be specified with the F command, ranging from 00 through FF. This example shows that a value of 26 hexadecimal was specified to be inserted in all locations between starting location 0100 and ending location 01FF. Note that the specified range has been filled with this code. The 26 hexadecimal is the ASCII code used to represent the "&" (ampersand) character. When the D command was used, the ASCII field of the display was filled with this character.

When using the F command, specify the starting address with care, so as not to fill the memory used by the CP/M operating system and the DDT program. Those portions of memory that should never be filled will generally depend on the amount of RAM in the computer and the length of your CP/M. As a general rule, however, the first 256 memory locations and the last 11K of memory locations should never be specified with the F command. If you fill these memory locations, the integrity of the CP/M operating system and the loaded executing program (in this case, DDT) will be lost. In addition to these horrors, the system will, more likely than not, crash, thus requiring a complete reboot (Cold Boot).

Note in the example, that location 0100 was specified as the starting location. This is normally the starting location available for manipulation in the TPA (Transient Program Area) of memory.

INSERTING AND ESTABLISHING A FILE CONTROL BLOCK FOR REFERENCING A FILE TO BE LOADED

–Ifilename.ext

The I (Insert or Identify) command is used to establish a name in a File Control Block (FCB) for subsequent loading of that file into memory. This I

command precedes the R (Read) command (described next), since Insert will not actually insert the file in memory, just its name in the FCB. The I command merely prepares for later insertion with R.

The I command is used if the DDT command was issued without specifying a file, or when different files must be loaded into DDT and examined one after the other. I and R can also be used when a file has already been loaded into memory, and a second file (a program patch, for example) is to be added in memory starting at a particular location. This is the case when you are modifying your CP/M BIOS (see *Soul of CP/M*). When using the I and R commands to insert a file with assembly language hexadecimal or machine code to overlay a file already in memory, DDT will automatically place the file's contents at the correct memory location. It will do this regardless of the program counter (PC) setting.

The following example shows the correct syntax for the I command.

```
A>DDT
DDT VERS X.X
-Itest01.com
-
```

The example shows that the file "TEST01.COM" was specified with the I command. The file name always follows directly after the "I" character. *No spaces are required.* Once the command is entered, the DDT prompt reappears. Now you can enter the R (Read) command, as described next.

The file specified with the I command *must* be on the logged drive. Drive names (A:, B:, etc.) cannot precede the file name in this command sequence or an error will occur. There is a way, however, to retrieve a file from a disk other than the logged disk. This is accomplished by changing the value of location 005C hexadecimal, which is a location in the area of RAM where DDT resides when it is loaded. To change the value assigned to this location, use the S (Set) command (described later in this section). Use the following table to determine the code necessary to represent the desired drive.

TABLE 4-1. DISK DRIVE CODE VALUES ASSOCIATED TO THE DEFAULT FILE CONTROL BLOCK

DEFAULT DISK DRIVE	CODE ASSOCIATED TO LOCATION 005C (HEXADECIMAL)
Currently Logged Drive	00
A	01
B	02

TABLE 4-1 CONTINUED. DISK DRIVE CODE VALUES ASSOCIATED TO THE DEFAULT FILE CONTROL BLOCK

DEFAULT DISK DRIVE	CODE ASSOCIATED TO LOCATION 005C (HEXADECIMAL)
C	03
D	04
E	05
F	06
G	07
H	08
I	09
J	0A
K	0B
L	0C
M	0D
N	0E
O	0F
P	10

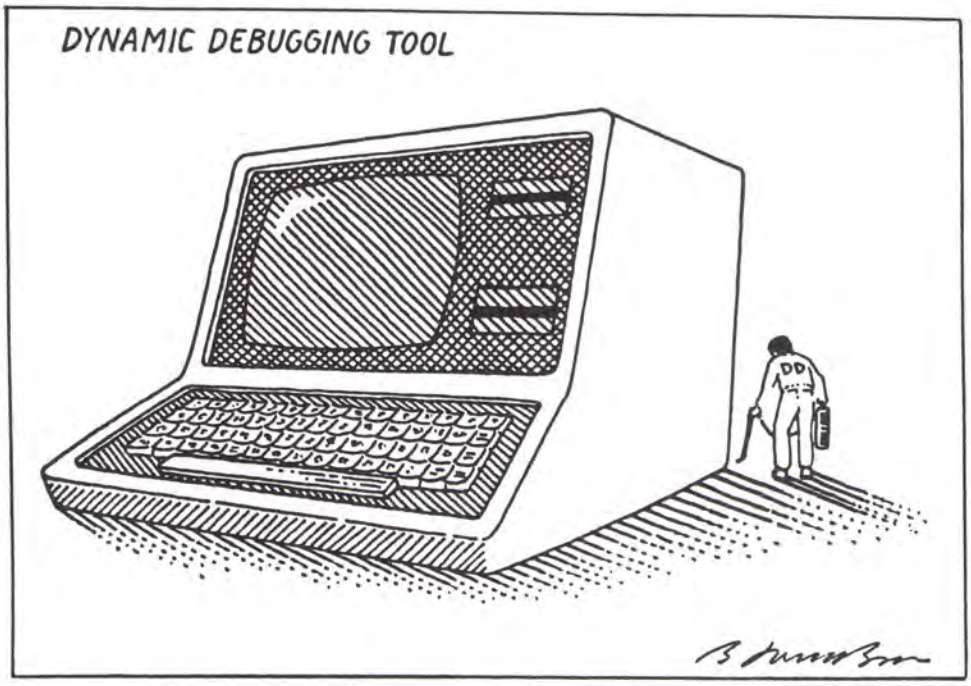
Note that the range of disk drives will depend on the version of CP/M installed in the system. For CP/M version 1.3, only Drives A and B can be used. For version 1.4, Drives A through D can be used. For CP/M Version 2.0 and newer, the full range of A through P can be used.

When setting a different value to location 005C, make sure that the selected drive is installed in the system, and that there is a disk inserted in it. Once the location has been set, and the file has been read into memory, don't forget to reset the location before exiting DDT.

READING A FILE ON A DISK AND LOADING IT INTO MEMORY

- R
- Rstart

The R (Read) command is used to cause a file specified by the I (Insert) command, as described previously, to be read and placed into memory. An R must never be used unless the I command was used first.



This command can be entered in one of two forms. The first is to enter "R" followed by the RETURN key. This will cause the file to be loaded into memory starting at location 0100 (first part of the TPA), unless the file contains hexadecimal or machine code with different starting address information. If this is the case, the file will contain an "offset" which will instruct DDT to load the file at a specific location usually higher than 100 hex. The second form of this command is to specify a loading location, by following "R" with the hexadecimal number representing the starting location for the file. Specifying a location is optional, and special care should be taken not to specify a location in some part of memory other than the TPA.

The next two examples show the correct syntaxes of the two forms of the R command. Included in the examples is the I command. We show both commands because neither should be used without the other.

```
A>DDT
DDT VERS X.X
-Itest01.com
-R
-
```

```
A>DDT
DDT VERS X.X
-Itest01.com
-R0200
-
```

When the second form of the command is used, the specified location number must not exceed the hexadecimal number FFFF. Otherwise, the starting location will wrap around to 0100 (normal default starting location with an offset of 0). You should, also, never specify a location between 0000 and 0100, since this is the portion of memory used by DDT and CP/M for their own variables. Also, you should not use an offset that, when combined with the file starting address, would have the file write over the DDT and CP/M operating system. The consequences of overlaying this data are obvious.

DISASSEMBLING AND DISPLAYING PORTIONS OF MEMORY WITH ASSEMBLY CODE

- L
- Lstart
- Lstart,end

The L (List) command is used to disassemble the hexadecimal code in memory and display the assembly language statements. When the L command is entered by itself, DDT displays the disassembled code of the next eleven assembly language instructions. In doing this, it displays the starting hexadecimal locations for each statement line. This is shown in the next example.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT  PC
0200 0100
-L
  0100  LXI  H,0000
  0103  DAD  SP
  0104  SHLD 01FA
  0107  LXI  SP,023C
```

```
  010A  XRA  A
  010B  STA  01F9
  010E  MVI  C,0F
  0110  LXI  D,005C
  0113  CALL 0005
  0116  LXI  D,01D5
  0119  INR  A
-
```

Once the eleven assembly language statements have been displayed, the DDT prompt will return and the program counter (PC) will be incremented to the last displayed location number.

The L command can also have a specified starting location number. This is accomplished by immediately following "L" with a number. We have demonstrated the results of this form of the command in the following example.

```
-L0119
  0119  INR  A
  011A  JZ   0177
    |
    |
    |
    V
  012E  LXI  H,0080
-
```

An ending location can also be specified by following the starting location number with a comma, and the ending location number. All assembly language mnemonics between the two locations specified will be displayed. The next example shows some variations on this form of the command.

```
-L0119,011D
  0119  INR  A
  011A  JZ   0177
  011D  LXI  B,
-
-L0100,0101
  0100  LXI  H,
-
```

Occasionally, when the L command is used, you may find the error symbol "??=" displayed. This error symbol normally means that the data (??=data) could not be interpreted by DDT because it is not an 8080 machine instruction opcode. Alternately, it could mean that the disassembled location contains noninstruction data or is the operand of an instruction.

DISPLAY AND OPTIONALLY CHANGE THE CONTENTS OF A LOCATION IN MEMORY



The S (Set) command performs two functions. It displays a series of locations beginning at a specified location, and optionally changes ("sets") the data contents. As each location is displayed, the current data associated with the displayed location is also displayed. Once the command is entered and the first location and its data are displayed, the console cursor or print head will be positioned to the right of the current data. At this point, new data can be entered which will replace the old data. Alternately, the RETURN key can be pressed to leave the original data intact. In either case, the next location and its data are displayed on the next line after the RETURN key is pressed. DDT will continue to perform in this manner until a period (".") is typed. This terminates the S command function, and the original DDT prompt reappears. This entire process is shown in the following example.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT  PC
0200 0100
-S0105
0105 FA <cr>
0106 01 <cr>
0107 31 <cr>
0108 3C 00
0109 02 01
010A AF <cr>
010B 32 .
-
```

The above example shows that the S command was specified starting with location 0105. The first line, therefore, reflects the data associated with that location as being FA. Since this value was not to be changed, the RETURN key was pressed, whereupon the next location and its data were displayed. Those lines in the example displayed with the "<cr>" symbol are those which did not have the data changed by pressing the RETURN key. At locations 0108 and 0109, new data was inserted or set, as shown by "00" and "01", respectively. When new data is set in this manner, it replaces the original data associated with the location displayed. The last displayed line shows that a period (".") was entered. This caused the S command function to terminate, and thus the DDT prompt was redisplayed. Set may be utilized for patching values in some part of memory for modification of a programs operation. WordStar can be patched in this manner, if you know the meaning of the values in its internal tables.

INSERT ASSEMBLY LANGUAGE STATEMENTS AT SPECIFIC LOCATIONS IN MEMORY



The A (Assemble) command is used to insert assembly language instructions in memory starting at a particular location. With this command you can enter 8080 assembly language mnemonics and operands. All numeric operands, however, must be hexadecimal numbers. Labels cannot be entered, nor can values be assigned to labels with this command. In this regard, we can refer to the A command as a mini-assembler.

The A command is entered by first specifying "A", followed (without a space), by the hexadecimal starting location number. Once this is entered, the starting location number will be displayed, and the console cursor or print head will be positioned to the right of the location number. This is where the assembly language statements are entered and followed by the RETURN key. Once text has been entered next to a location, the next location number (not occupied by the last assembly language statement) will be displayed on the next line. The command will await input. If the RETURN key is pressed without entering any text, the command will terminate and the DDT prompt will reappear. This is shown in the next example.

```
A>DDT DDT.COM
DDT VERS X.X
NEXT  PC
1400 0100
-L018B,0194
  018B  MVI  C,10
  018D  CALL 0005
  0190  STA  1E5F
  0193  RET
  0194
-
-A018B
018B  LHLD 1EC
018E  SHLD 8E7
0191  INX  H
0192  SHLD 8ED
0195  LDA  1EB
0198  .<cr>
-
```


This example displays a section of code (in disassembled form) from location 018B through 0194. When using DDT as a mini-assembler to write programs, the L command is normally used before any assembly language code is entered so that the locations to be modified can first be verified. This example shows the correct syntax of the A command, after the code is displayed. The specified starting address was the same as that beginning the displayed code, and once the command was entered, the first memory location to receive statements was displayed. Each time the RETURN key was pressed, the next location that would receive code was displayed by the A command. Note on the last line that a period (.) was used to terminate the A assembly command.

Note that all values to the far right are hexadecimal values. If any value that is not a hexadecimal address between 0 and FFFF is entered after the A command, the command function will be terminated and the "?" error symbol will be displayed. This error symbol will also occur when an assembly language mnemonic or operand value that could not be interpreted by DDT was entered. If this occurs, simply retype the A command with the starting location of where the error occurred and re-enter the statement properly.

As can be noted in the example, the A command function displays the next address location to be modified on the line below where text has been

entered. Note however, that the memory locations presented on each line may skip numbers. This is because each assembly language entry takes a different number of memory locations. As a result, the number you see on the next line will be +1, +2, or +3 greater than the previous memory location depending on the 8080 mnemonic typed in. To terminate this command function, press the RETURN key, and the DDT prompt will reappear. A period (".") can also be typed to terminate the program, but the use of the RETURN key is sufficient, since it will inform DDT that no further input is required.

COPY THE CONTENTS OF A SPECIFIC MEMORY BLOCK TO ANOTHER BLOCK STARTING AT A SPECIFIC ADDRESS



The M (Move) command is used to copy a specified range of data or 8080 program code, or whatever, to a new location in memory. The data to be copied is specified with a starting location (start1) and an ending location (end1). This range is called a block. Its size is the difference between these two parameters. The destination location that this block is moved to is specified by the parameter start2. It should be noted that this command will copy a block of data to a destination location exactly as it was found in the original locations. It should rarely be used to "relocate" machine language code, since the relocating process does not adjust addresses contained in the block moved. Consequently, a program that references itself will not work properly if the code is moved to a different starting address with M. The M command should be used with caution, since copying the data, as is, from one location to another usually destroys the integrity of any program that it overlays in memory. Any data in the destination locations is overlayed with the copied data and thus changed.

This command is entered by first typing "M" followed (without a space) by the starting location, a comma, the ending location of the block to be moved, another comma, and the starting location to which the block is to be copied. The next example demonstrates this.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT PC
0200 0100
-D0110,012F
0110 11 5C 00 CD 05 00 11 D5 01 3C CA 77 01 01 00 00 .\.......<.w....
0120 C5 0E 14 11 5C 00 CD 05 00 C1 B7 C2 67 01 21 80 ....\.......g.!.
-
-M0110,011F,0120
-
-D0110,012F
0110 11 5C 00 CD 05 00 11 D5 01 3C CA 77 01 01 00 00 .\.......<.w....
0120 11 5C 00 CD 05 00 11 D5 01 3C CA 77 01 01 00 00 .\.......<.w....
-
```


This example demonstrates the use of the M command. First, for clarity, the data in locations 0110 through 012F are displayed. Then the M command is issued, instructing DDT to copy the data from locations 0110 through 011F (a block exactly 16 bytes long) to the equivalent number of locations after starting location 0120. We then display the same area of memory. Note that the second line now contains the same data as the first line. Always use this command with caution so as not to inadvertently destroy the integrity of any program in memory. Be especially careful when specifying a destination starting location, so that reserved parts of memory are not overlayed.

The M command can be used to force any number of memory locations to fill with constant data.

EXECUTE PORTIONS OF A PROGRAM IN MEMORY AND SET BREAKPOINTS

- -G
- -Gstart
- -Gstart,end
- -G,end
- -G,end1,end2

The G "Go" command is used to initiate the execution of the program being examined by DDT, i.e., to make a program in memory run. There are various forms of this command, some of which are rarely used by the non-programmer.

Various forms of the G command allow starting the program at the current address in the computer's Program Counter, running of a program located at a specific memory address, specifying a particular ending point for the program, and so on.

Using forms of the G to start an 8080 program in memory requires a special consideration. A program run under DDT must end with a special 8080 Restart instruction (code RST 7 usually), rather then the 8080 RETURN instruction (RET) that usually terminates a program. If the RST instruction is not used DDT will probably do an illegal return (RET) and lose control of the program. As a result, the program will "hang" and only a reset will bring it back. When using G, one can either place the restart instruction at the end of the program with the A command or other forms of the G command can be used that automatically insert the RST instruction in the desired location or locations. The RST instruction is referred to as a "breakpoint" in the industry because the location is the place where the execution of the program "broke" out.

In CP/M when a RST or breakpoint is encountered by DDT, it prints out the address where it occurred lead by an asterisk character (i.e., *7FFF means a restart instruction was found at 7FFF). This information is useful to programmers so that they know where the program ended.

The following forms of the G command can be used.

G This form of the command will cause the program in memory to begin executing from the location set by the program counter (PC). This form is rarely used unless the user has set the program counter to the desired starting address. Otherwise DDT loses control of the program and it is impossible to return to DDT or CP/M without a system restart sequence (Cold Boot).

Gstart This form of the command will cause the program to execute beginning at the hexadecimal location in memory shown by "start" in the command string. This command string will have the same unpredictable results as the G command entered by itself, if the RST instruction is not first inserted.

Gstart,end This form of the command will cause the program to begin execution at location "start" and terminate when location "end" is reached. The program acts as if a breakpoint instruction was inserted at the address "end", and displays the address preceded by an asterisk. However, no real code will be

found at the "end" address. If another breakpoint instruction is detected before "end", it will cause the program to end. Note that this command string may have the same bad effects as those described for the "G" and "Gstart" command strings.

Gstart,end1, end2 This form of the command is the same as the previous one except that two ending locations are specified ("end1" and "end2"). When the CPU encounters either breakpoint, execution will stop and control will be returned to DDT. This form is used when the programmer has two places where the program may terminate and wants to differentiate them.

G,end This form of the command is the same as "Gstart,end", except that program execution will begin with the location setting of the program counter (PC). When the breakpoint "end" is encountered by the CPU, execution will stop.

G,end1, end2 This form of the command is the same as "Gstart, end1,end2" except that program execution will begin with the location setting of the program counter (PC).

Note that DDT is terminated by typing G0 or CTRL/C. In both cases, execution is forced to start at location 0. Address 0 contains the code to cause a warm boot of the computer system and brings the user back to CP/M's A> prompt.

EXECUTE A NUMBER OF PROGRAM INSTRUCTIONS IN MEMORY AND DISPLAY THE CONTENTS OF THE CPU REGISTER

–Unumber

The U (Untrace) command is used to execute a number of program instructions in memory without actually showing the instructions executed as it proceeds (like the Trace command does). A U will display the state of the 8080's registers once BEFORE commencing execution of the program. A T shows the registers at every executed instruction. Only the last instruction will be displayed by U. The U (and T Trace) both simulate the program being run. While U is faster than T, both are 500 times slower than G.

The U untrace command is entered by typing "U", followed (without a space) by a hexadecimal value representing the number of instructions to be executed. Once execution is complete (instructions used up or restart instruction encountered), the U command function will terminate. In addition, DDT will set the program counter (PC) to the contents of the register pair H and L. It also sets the starting location for the L (List) command to the location of the next instruction to be executed.

If you want to stop the U command and return to the DDT, any key on the console can be pressed while the command is in operation. This is shown in the following example.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT PC
0200 0100
-U5
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  H,0000*0
-U10
C0Z1M0E1I0 A=00 B=0000 D=0000 H=0100 S=023C P=010B STA  01F9*012
-U20
C0Z0M0E0I0 A=04 B=0014 D=01D5 H=0003 S=023A P=0123 LXI  D,005C*0
-
```

This example showed the results of using the U command first to execute 5 instructions, then 10 instructions, and finally 20 instructions. Each time, the U command displayed the current status of the CPU registers before the first instruction was executed. The fields of the displayed information in the first line of the above example have the following definitions.

C0Z0M0E0I0	is broken down as follows: "C0" means the Carry Flag = 0. "Z0" means the Zero Flag = 0. "M0" means the Minus Flag = 0. "E0" means the Even Parity Flag = 0. "I0" means the Interdigit Carry Flag = 0.
A=00	The Accumulator contains 00.
B=0000	Register B contains 00 and Register C contains 00.
D=0000	Register D contains 00 and Register E contains 00.
H=0000	Register H contains 00 and Register L contains 00.
S=0100	The Stack Pointer Register contains 0100.

P=0100	The Program Counter Register contains 0100.
LXI H,0000*010B	Assembly language instruction of location 0100 (shown by "P=0100").

This information will be useful in determining console output when using the T and X commands, described next.

TRACE THE EXECUTION OF A NUMBER OF PROGRAM INSTRUCTIONS

–Tnumber

The T (Trace) command has the same purpose as the U command described previously. The difference is that the status of the CPU registers and the actual 8080 instruction mnemonics are displayed after each instruction is executed. The command operates quite a bit slower than the U and G commands, since information has to be displayed for the execution of each instruction.

The command is entered by first typing "T", followed (without a space) by the hexadecimal number of instructions to be executed. Once the T command has finished execution, DDT will set the program counter (PC) to the contents of the register pair H and L. It also sets the starting location for the L (List) command to the location of the next instruction to be executed.

If you want to stop the T command and return to the DDT, any key on the console can be pressed while the command is in operation. The next example shows how this command is used.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT PC
0200 0100
-T5
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  H,0000
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0103 DAD  SP
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0100 P=0104 SHLD 01FA
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0100 P=0107 LXI  SP,023C
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=023C P=010A XRA  A*010B
```

```
-T10
COZ1MOE1IO A=00 B=0000 D=0000 H=0100 S=023C P=010B STA  01F9
COZ1MOE1IO A=00 B=0000 D=0000 H=0100 S=023C P=010E MVI  C,0F
COZ1MOE1IO A=00 B=000F D=0000 H=0100 S=023C P=0110 LXI  D,005C
COZ1MOE1IO A=00 B=000F D=005C H=0100 S=023C P=0113 CALL 0005
COZ1MOE1IO A=00 B=000F D=005C H=0100 S=023A P=0005 JMP  A800
COZ1MOE1IO A=00 B=000F D=005C H=0100 S=023A P=A800 JMP  AEA2
COZ1MOE1IO A=00 B=000F D=005C H=0100 S=023A P=AEA2 XTHL
COZ1MOE1IO A=00 B=000F D=005C H=0116 S=023A P=AEA3 SHLD B74A
COZ1MOE1IO A=00 B=000F D=005C H=0116 S=023A P=AEA6 XTHL
COZ1MOE1IO A=00 B=000F D= <space>
-
```

The second section in this example showed that displaying information by DDT was aborted after the space bar on the console was pressed. You can abort output in this manner by pressing any key (except CTRL/C, which will abort DDT and return you to CP/M).

DISPLAY THE CONTENTS OF THE CPU REGISTERS OR SPECIFIC REGISTER, AND OPTIONALLY MAKE CHANGES

–X
–Xregister

The X command is used to display the current values of the 8080 registers, or the contents of a specific register. It is also used to make any desired changes in a specific register. The information displayed when the X command is used similarly to that displayed with the U and T commands. The difference here is that the information will reflect the current status of the CPU's registers regardless of the program execution status.

When the X command is typed by itself, all of the CPU registers and their current contents are displayed on one line. Afterwards, the DDT prompt will be returned. Alternately, the X command can be entered following a specific CPU register name, causing the contents of the specified register to be displayed. When this form of the command is used, the console cursor or print head will be positioned to the right of the register name and its contents, where you can enter new replacement data, if you wish. To leave the register contents intact, press the RETURN key and the DDT prompt will reappear.

The X command can be used to display and modify any of the CPU registers displayed on the line in the following example.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT  PC
0200 0100
-X
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  H,0000
-
-XC
C0 <cr>
-Xz
Z0 <cr>
-XM
M0 <cr>
-XE
E0 <cr>
-XI
I0 <cr>
-XA
A=00 <cr>
-XB
B=0000 <cr>
-XD
D=0000 <cr>
-
```

The example shows the various uses of the X command. In the first use of the command, X was typed by itself. This caused all the CPU registers to be displayed with their current contents. Next, the example shows the X command being used to display the individual flags and registers. The "<cr>" symbol is used to indicate that the RETURN key was pressed after each display. Where this symbol occurs, however, is the position that you can enter new values to be placed in the registers or flags, if you wish. This is shown in the next example.

```
A>DDT TEST01.COM
DDT VERS X.X
NEXT PC
0200 0100
```

```
-XZ
 Z0 1
-XZ
 Z1 0
-XZ
 Z0
-
-XB
B=0000 FFFF
-XB
B=FFFF 0000
-XB
B=0000
-
-^C
```

This example first shows how the status of the Z (Zero) flag is changed from 0 to 1, and then back to 0 again. Then, the example shows how the contents of the B and C registers is changed from 0000 to FFFF, and then back to 0000 again.

USE THE BUILT-IN CALCULATOR FOR ADDING AND SUBTRACTING HEXADECIMAL NUMBERS

– Hxxxx,xxxx

The little known H (Hex) command is used to perform hexadecimal number addition and subtraction operations. This command performs the function of a calculator. It does not perform any functions affecting the contents in memory. The H command is useful when you want to obtain addition and subtraction results from any two given hexadecimal numbers during the execution of the DDT program. A frequent use of H is for figuring address offsets for overlaying code into CP/M's BIOS. (See *Soul of CP/M*).

The command is entered by first typing "H", followed (without a space) by the first hexadecimal number, a comma, and the second hexadecimal number. After the RETURN key is pressed, two numbers are displayed. The first number is the sum of the two numbers entered with the command (xxxx

+ xxxx). The second number is the difference between the two numbers (xxxx − xxxx). Only hexadecimal numbers (digits 0–9 and A–F, i.e., equivalent to 0 through 15, decimal), may be entered with this command. Numbers may be from one digit to a maximum of four digits in length. The H command will perform the addition correctly regardless of the order in which the numbers are entered. To obtain a positive difference (subtraction), however, the first number entered must be equal to or greater than the second number entered. If the first number entered is smaller than the second number, the command will begin calculating from FFFF to provide the subtraction result.

```
A>DDT
DDT VERS X.X

-H1,1                    (1 + 1 = 2) (1 - 1 = 0)
0002 0000
-H2,1
0003 0001
-H5,2
0007 0003
-HA,5
000F 0005
-H100,50
0150 00B0
-HABCD,9ABC
4689 1111

-H1,2                    (1 + 2 = 3) (1 - 2 = FFFF)
0003 FFFF
-H200,300
0500 FF00
-H5000,AAAA
FAAA A556
-
-^C
```

This example first shows a sequence of calculations with the numbers entered with its command in the correct sequence. Following these are some examples of the subtraction results displayed when the second number entered with the command is greater than the first.

This concludes the description of the DDT command. The following chapter describes the built-in SAVE command.

Chapter 16

SAVE

CP/M Versions:

1.3 **1.4** **2.0** **2.2**

Command Type:

Built-In

SUMMARY

The SAVE command is used in special circumstances that require the saving of the contents of RAM into a specified file on a disk. Usually only programmers require use of SAVE, most frequently it is used with assembly language programming for saving a RAM image to the disk media. SAVE is also required when using MOVCPM to modify the CP/M operating system's BIOS. The SAVE command is also used to create files to be placed in newly defined user areas (see previous USER command). If you are not involved in assembly language programming on your CP/M system, and you do not use different USER areas on disks, then you will probably not need to use this command.

PROGRAM ACCESS

The command line format for the SAVE command is very simple, although the information you need to obtain to create the command line is a little more involved. To save the contents of the memory into a file effectively, you must specify the number of "memory pages" to be saved, and the name of the file you wish to create. To save a file, you must use the SAVE command while the data to be saved is still in memory. You must not start any other program

until the SAVE command procedure is completed. If you run a program during the SAVE command procedure, the file you wanted to save will be erased when the memory is refreshed with the new program.

The SAVE command is usually used in conjunction with the transient command DDT (see the DDT description). You may use DDT to determine the number of "memory pages" to specify with the SAVE command. DDT may also be used to capture all the data in memory that you require for the file about to be specified. The DDT command provides you with the hexadecimal numbers (base 16) of the first and last memory pages when you bring up DDT with the file name to be measured following it. (A memory page is a block of data consisting of 256 bytes, or 100 hexadecimal bytes.) Then you will have to convert the hexadecimal number of the last memory page into a decimal number to be used with the SAVE command.

Once the correct decimal number of memory pages has been determined, you can use the SAVE command to save the file specified. The next example shows the command being used on Drive A. The number of memory pages is 12, and the file in which the information is to be stored is to be called "SPECIAL.COM." Because the logged drive is A, and no other drive was specified, the file will be stored on A.

```
A>SAVE 12 SPECIAL.COM

A>
A>STAT SPECIAL.COM

Recs   Bytes Ext Acc
  32      4k   1 R/W A:SPECIAL.COM
Bytes Remaining On A:   476k

A>
```

Note in the example, that the STAT command was used after the file was saved, to verify its size. Even though 12 memory pages was specified, which equates to 3,072 bytes (or 3K), the STAT command showed the file to have 4K. This is because CP/M stores files in 2K allocation units. So even though the data saved consisted of 3K, the actual space used on the disk will be approximately 4K.

The file can be stored on a disk in a drive other than the logged drive. This is done by placing the drive name (B:, C:, D:, etc.) after the number of memory pages and before the file name. Before using the SAVE command to store a file on a disk, make sure that there is enough free space left on that disk. If there is not enough room on the disk for you to store the specified file, the following error message will display.

```
A>SAVE 12 SPECIAL.COM

NO SPACE

A>
```

When the preceding occurs, you will have to calculate approximately how many kilobytes of disk space the file will require. This is a relatively simple calculation, since the number of bytes in a memory page is 256, and the number of bytes in a kilobyte is 1,024.

1,024 divided by 256 equals 4

Thus, we could say that a memory page is one quarter of a kilobyte, or that a kilobyte consists of four memory pages. By taking the 12 memory pages shown in this example, and dividing the number by 4, we can ascertain that the file will consist of 3K, or 3 kilobytes. With this knowledge, we can determine before the SAVE command is used whether we have sufficient space available on the disk for storage of the file.

Note:
If the answer for the above equation comes out an odd number (like 3) then we must round up to the nearest even number (here 4). This is because CP/M saves files in a minimum of 2K byte units.

When specifying a file to be saved, a specific file name must be given, as the SAVE command will not accept wildcards for the substitution of real characters. The following error message will be displayed if the SAVE command detects any wildcards in the file name, or if the number of memory pages was not specified.

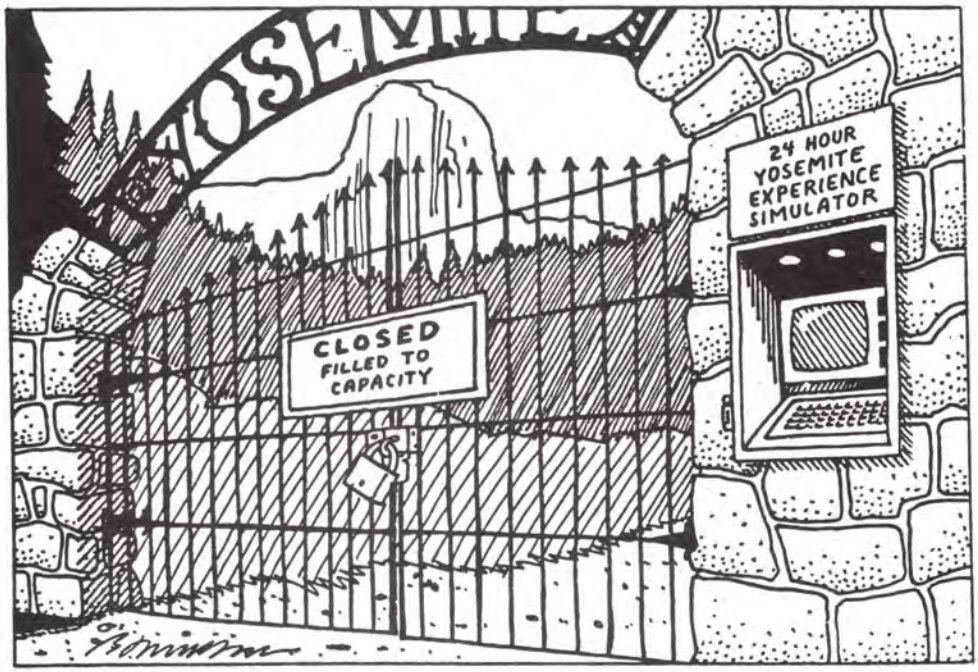
```
A>SAVE 12 SPECIAL.*

FILENAME?

A>
A>SAVE SPECIAL.COM

FILENAME?

A>
```



When this error message occurs, retype the SAVE command sequence properly. The information you need for the file will be still stored in the memory.

Chapter 17

DUMP

CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

Transient

SUMMARY

The DUMP command is a program which permits the CP/M user to display the contents of a machine code file in hexadecimal format. The ability to do this is known as "dumping" the contents of a file that could normally only be viewed via the DDT command. The DUMP command permits the viewing of assembled machine code files as well as transient program command file with a .COM extension. DUMP works like the D command of DDT.

PROGRAM ACCESS

To access the DUMP program, the file DUMP.COM must be stored on a disk in one of the computer disk drives. There is only one form of the DUMP command, which consists of entering "DUMP" in response to the logged drive, followed by the filespec. Once the command is entered, DUMP will display the entire contents of the machine code file in hexadecimal format on the console device. This is similar to the way DDT displays the code but without the ASCII representations.

DUMP permits the use of wildcards ("*" and "?" characters) to approximate the desired file name. When wildcards are used, the program dumps the contents of only the first file found on the disk that matches the specified wildcards.

The DUMP command operates in the same way as the built-in direct TYPE command in that the contents of the specified file are continuously displayed until the end of the file is reached. A CTRL/S command can be entered to stop and resume scrolling or another key can be pressed to terminate the command.

The hexadecimal code displayed by DUMP is organized in basically the same way as with the DDT command. The difference is that no ASCII representation is displayed.

The following example demonstrates how the DUMP command is used, and shows the format of the displayed hexadecimal code.

```
A>DUMP TEST01.COM

0000 21 00 00 39 22 FA 01 31 3C 02 AF 32 F9 01 0E 0F
0010 11 5C 00 CD 05 00 11 D5 01 3C CA 77 01 01 00 00
0020 C5 0E 14 11 5C 00 CD 05 00 C1 B7 C2 67 01 21 80
0030 00 7E E6 7F FE 1A CA 67 01 FE 20 CA 59 01 FE 0D
0040 CA 59 01 FE 0A CA 59 01 3A F9 01 FE 00 CA 5E 01
0050 03 3E 00 32 F9 01 C3 5E 01 3E 01 32 F9 01 23 7C
0060 B7 CA 31 01 C3 20 01 11 5C 00 C5 0E 10 CD 05 00
0070 C1 CD 81 01 11 E5 01 0E 09 CD 05 00 2A FA 01 F9
0080 C9 C5 E1 06 00 11 F0 D8 CD A5 01 11 18 FC CD A5
0090 01 11 9C FF CD A5 01 11 F6 FF CD A5 01 7D C6 30
00A0 4F CD C8 01 C9 0E 2F 0C 19 DA A7 01 7A 2F 57 7B
00B0 2F 5F 13 19 79 FE 31 D2 C2 01 78 B7 79 C8 CD C8
00C0 01 C9 06 FF CD C8 01 C9 D5 C5 E5 59 0E 02 CD 05
00D0 00 E1 C1 D1 C9 6E 6F 20 73 6F 75 72 63 65 20 66
00E0 69 6C 65 2E 24 20 77 6F 72 64 73 20 69 6E 20 74
00F0 68 65 20 66 69 6C 65 2E 24 00 00 00 00 00 00 00

A>
```

The example shows the hexadecimal representation of the specified file. The 4-digit numbers in the left-hand column are the starting address locations for each line of code. On each line of code, sixteen 2-digit numbers will be displayed. Each 2-digit number corresponds to the address location and represents the value of the data in it. The number immediately following the starting address location, for example, corresponds to the first (starting) address location. The second number corresponds to the second address location on that line, and so on. In the example, the first line has a starting

address of 0000. Since there are sixteen locations on that line, the data corresponds to address locations 0000 through 0009 and 000A through 000F, hexadecimal. The second line shows the data for locations 0010 through 001F. All the lines of code are organized in this manner.

Chapter 18

FORMAT

CP/M Versions:

1.3 1.4 2.0 2.2

As specified by distributor

Command Type:

Optional Transient

SUMMARY

Almost all CP/M operating system diskettes come with a disk-formatting program. This program is a necessary utility because it allows the CP/M user to prepare (initialize) new floppy disks or hard disks (removable or permanent) so CP/M can store information on them. This is necessary because there is no guarantee that new (fresh from the box) floppy disks will operate correctly with your configured CP/M or your disk drives, or that a hard disk will be properly formatted when you first turn on the system. (This may even be the case when each diskette has information regarding the correct format previously formatted on it by the disk manufacturer.)

The formatting program arranges the tracks and sectors in such a way that CP/M can reference them properly. If FORMAT is included on your CP/M distribution disk, it will normally be customized for the specific disk controlling hardware installed in your system. Manufacturers of disk drives and disk controlling hardware don't design their equipment to be compatible with those of other manufacturers. This means that your formatting program may not operate on your friend's disk equipment. We are about to describe how to use a *typical* disk formatting program. It may be slightly or very different from the one you are using, but the following examples will give you a feel for the way any FORMAT program should operate.

You will quickly discover that the disk-formatting program will be the first program you will use when receiving and installing the CP/M operating sys-

tem. Use it to format a new floppy disk so that you can then make a backup copy of the CP/M distribution disk. The distribution disk is almost always an 8-inch, single-density, diskette. Usually the disk is soft sectored, meaning there is only one index hole in the diskette and it is up to the computer to arrange the spacing of the sectors itself. The disk format for distribution CP/M diskettes is based on the IBM 3740 specification. CP/M may also be distributed to you on a 5-¼-inch disk. Since there is no established standard for 5-¼-inch disk media, the format will be one chosen by the manufacturer of the computer with which the CP/M is supplied.

Some versions of CP/M are distributed on hard sectored disks. In a hard sector disk, there are 32 individual "index" holes spaced evenly about the inner-most track of the disk. Electronics in the disk controller hardware sense these holes and use them to indicate the start locations of sectors physically of the disk itself, rather than from magnetic-based sector indications that are put down by a soft-sector-based system. Still the hard sectored disk comes only with the holes marked. It is still necessary to put additional track sector/information to allow the disk to hold CP/M files. If you imagine that a train is the data or program file that is to be "stored" on the disk media, you can imagine that formatting a disk lays down the tracks for the train to rest on.

Some systems utilize hard-sectored floppy disks, in which case, a formatting program will not be necessary as the format of these types of disks cannot be changed. If your system uses hard-sectored disks, then all you need to do is simply copy your distribution disk exactly, and you can skip this section of the chapter. If you are using soft-sectored disks, you will need to use an equivalent disk for backup purposes and format it to single density (as opposed to double density). Later, if your system supports it, you can format other disks to double density.

ACCESSING THE DISK FORMATTING PROGRAM

To format a new disk, the formatting program (perhaps called FORMAT.COM) should be stored on your system drive, i.e., Drive A, if it is a floppy disk drive. If the formatting program is on a hard disk, use whatever is the logical drive name. For some versions of these programs, you must have at least two disk drives: one containing the formatting program, and the other containing the disk you want formatted. Most formatting programs, however, permit you to format disks using only one drive. (Formatting with a one drive system is a fairly tedious process.)

Your formatting program may not be called "FORMAT.COM" but may be called one of the following names, or something similar.

DSKFMT.COM
DSKINIT.COM
FORMAT2.COM
IFORMAT.COM
IN.COM
INIT.COM
INITDSK.COM
MFORMAT.COM

. . . etc.

You can see that there is much confusion for the potential CP/M user. When you first bring up your CP/M you can look for these names. If you still can't identify it, you can simply try running various programs that might be it. The correct one can be identified by presenting a sequence of operations similar to the one explained later.

Throughout this section, we will assume that the formatting program is called "FORMAT.COM". You can rename the file containing your formatting program to anything you wish, as long as it has an extension or file type of ".COM". In this way, it can be accessed as an executable transient command file.

DISK DENSITY: Floppy diskettes can be formatted in one of many densities. The density refers to the number of tracks and sectors laid down on the disk by the format program. In single density, there are 40 tracks, 32 sectors, and 128 bytes per sector. Double density gives more than twice the storage capacity by using 77 tracks. And, while in single density the number of sectors per track and the number of bytes in each sector is fixed according to the IBM standard, in double density, the number of sectors and byte per sector varies with the manufacturer. There is no double-density standard, so all do it differently. Generally, however, the double-density format can be in one of three possible organizations: 256 byte per sector, 512 bytes per sector, or 1024 bytes per sector. The larger bytes per sector gives more storage efficiency since less overhead is wasted in less sectors per disk. The total capacity of the disk although basically about ½ megabyte, grows slightly as the number of sectors decreases.

From a compatibility perspective, it doesn't matter which of these formats you choose . . . you won't be able to read someone else's double-density standard, unless you have the same computer and the same CP/M.

The examples in this description pertain to a typical formatting program used with disk drives that support single- or double-sided, multidensity, soft-sectored floppy disks, in either the 5-¼-inch ("Mini") or 8-inch ("Standard") sizes. "Multidensity" means that these types of disks can normally be formatted to either single or double density, and in some rare cases, quadruple density. In this example, we will deal with single- and double-density formats.

FORMATTING A SOFT-SECTORED FLOPPY DISK

FORMAT

To format a floppy disk, place the disk containing the formatting program in the system drive, i.e., Drive A. If your system has only one drive, then you will have to execute the formatting program first, and then when instructed to remove the disk, do so, and replace it with the floppy disk to be formatted. If your system has more than one drive, place the floppy disk that is to be formatted in another drive, for example Drive B. Next perform the Warm Boot function (CTRL/C). Make sure that the logged drive is A, as indicated by the "A>" prompt. Type in the name of the command, in the case of the following example it is "FORMAT", and then press the RETURN key.

```
A>FORMAT

XYZ COMPANY FORMAT VERSION X.X
THIS COMMAND FORMATS A DISK, ERASING ALL DATA ON IT.
IS THIS WHAT YOU WANT? [Y/N]: Y
REMOVE SYSTEM DISK AND PRESS THE RETURN KEY: <CR>
ENTER DRIVE NAME CONTAINING THE DISK TO BE FORMATTED: B
DISK TO BE FORMATTED IN DRIVE "B". IS THIS OK? [Y/N]: Y
-------------------------------------------------
| 1 =  128 BYTES PER SECTOR, SINGLE DENSITY FORMAT |
| 2 =  256 BYTES PER SECTOR, DOUBLE DENSITY FORMAT |
```

```
| 3 = 1024 BYTES PER SECTOR, DOUBLE DENSITY FORMAT |
-----------------------------------------------------

SELECT ONE OF THE ABOVE FORMATS [1,2,3]: 1
"128 BYTES PER SECTOR, SINGLE DENSITY". IS THIS OK? [Y/N]: Y

FORMATTING IN PROGRESS. PLEASE WAIT UNTIL FINISHED.

FORMATTING COMPLETE. TO FORMAT ANOTHER DISK IN THE SAME
DRIVE, TYPE IN "Y", OTHERWISE PLACE SYSTEM DISK BACK IN
DRIVE A AND TYPE RETURN: <CR>

A>
```

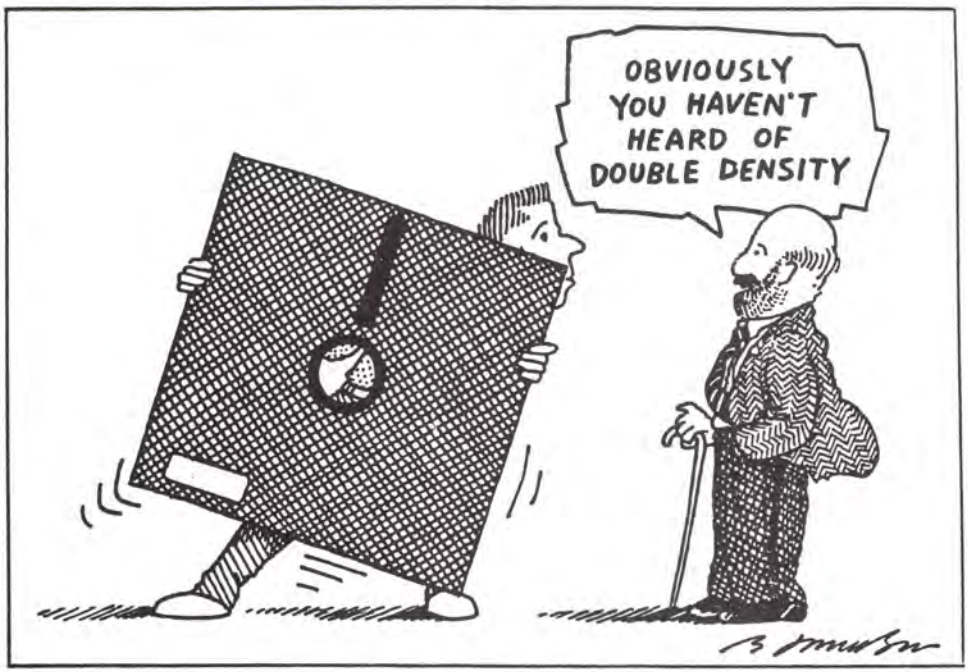
This example shows a program with easily understood prompts. Our users typing reaction to the CP/M prompts is shown in the figure in bold. The prompt that says "REMOVE SYSTEM DISK, AND TYPE RETURN" indicates that you could use this program on a system that has only one floppy disk drive. If your system has only one drive, then remove the disk from the drive at this point and insert the disk to be formatted. You should also do this if you had previously specified Drive A (even though there is more than one disk drive connected to the system). If your system has more than one drive, and a drive name other than the one containing the FORMAT program was specified, then this prompt can be ignored since the main disk need not be removed. Some people prefer always to remove the disk in drive A, regardless, since if the FORMAT user accidentally instructed the CP/M to format the distribution disk in drive A, everything on it would be erased.

Since the formatting program has been loaded into the computer's memory, until the end of the program has been reached, CP/M will not have to reference the system disk. However, upon answering the above prompts the FORMAT program will immediately begin accessing the disk to be formatted. A clicking sound can often be heard as the program systematically lays sector indications on the media surface.

Once the message "FORMATTING COMPLETE" has been received, the disk surface has been properly formatted. You now will be given the option either to insert another blank disk to be formatted and then type "Y" (to do it again), or to reinsert the system disk and press the RETURN key again to end the program and return to CP/M's A> prompt. This final part is important, because the formatting program can only terminate correctly if the disk containing the FORMAT.COM program is inserted in the drive.

During the formatting process, the program might not be able to properly format the disk as instructed. This could be due to the wrong type of disk (hard versus soft sectored, wrong density, etc.), or to physical damage on the disk, etc. If this happens, normally the formatting program may simply "hang up" when after the Y to go question and will only work if you shut the door. In other cases, the FORMAT program will present its own error message to indicate the door is open. There are some CP/M error messages, however, that may be displayed in case of such a malfunction.

If, for example, you initiated the formatting command and specified Drive B, and did not place a disk in that drive, then a CP/M error message, such as "DRIVE NOT READY", will probably be displayed, although some programs will not display any message at all. The actual error message displayed will depend on the program. In a case like this, the formatting program would terminate. If this occurred, you would have to reboot the system (Cold Boot), and start over again. If any other error messages are received, refer to Appendix H for detailed descriptions of the CP/M error messages.



You can use the DIR command to display the directory of the disk and to verify that a disk has been properly formatted. Since no data will, as yet, have

been stored on the newly formatted disk, the message "NO FILE" will be displayed if the disk was properly formatted. If you receive an error message such as

"BDOS ERR ON B: BAD SECTOR"

or something similar, it probably indicates that the formatting was unsuccessful. This means that an error was detected while CP/M was reading the disk. This could be due either to an improper format procedure, or a physical fault on the disk. When this occurs, repeat the formatting procedure to see if the problem corrects itself. Sometimes the CP/M program will offer you the chance to press different keys to take different reactions to the BDOS ERROR. You may be able, in some cases, to "TRY" again, return to CP/M, or ignore the error. If the error is repeated, then the disk might be faulty and should not be used. (If you get the same error message with every disk you try, then it's probably (but not totally) not the disks at all.) If the formatting was successful, you can use the STAT transient command (described previously) to determine how much space for storing bytes is available on the disk.

You will have to use the SYSGEN transient command (described in the next chapter) to copy over the CP/M system image onto the first two tracks of the disk to make the disk one that is "bootable" on your system. The PIP transient command (also described previously) can be used to copy files onto the disk. If you have only one drive in your system, you will not, however, be able to use the PIP command to copy files onto a diskette. Normally, in single-drive systems, a special copying utility program has been provided for this purpose. This program is often called "COPY.COM" but since it is non-standard, its method of operation may differ among distributors of the CP/M operating system or computer manufacturers.

OTHER FORMATTING PROGRAMS: As we mentioned earlier, not all formatting programs are alike. In the next example, we show a format program from another manufacturer. Although this program is accessed exactly as in the previous example, the action the program displays is different.

```
A>FORMAT

CompuPro Disk Sub-system FORMAT Version 2.2B

Specify drive (A: - D:)  :b Disk is Formatted as 1024 byte
sectors.
```

```
Select Disk format mode (0,1,2,3):
0 = 128          2 =  512
1 = 256          3 = 1024        >3
Confirm ready for format on disk drive B (y).y
         1        2        3        4        5        6
01234567890123456789012345678901234567890123456789012345678901234567890
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
FORMAT complete.
Do you want to FORMAT another disk ?
    R  - request new parameters.
    cr - duplicate last FORMAT.
    N  - exit back to system.          >n

A>stat b:

Bytes Remaining On B: 1192k
```

In this example, the program asks for the drive with the disk we wish to format. Here we answer the B drive. The program informs us that the current density (on the disk in drive A) is 1024 bytes per sector, then asks us which density we want to format the disk in B. We answer 1024 bytes per sector (double density). The program then asks us to confirm what we have requested (as a safety feature lest we specify the wrong disk, density, etc). Then the formatting begins. The row of numbers represents the tracks on the disk (actually this example is for a double sided disk so both sides of the disk are receiving the formatting) and 77 tracks are indicated. As each track is formatted, an F for format appears under the track number. When all the tracks are finished, the program then verifies that the format was successful and each track is checked one at a time. A good track receives the letter V for verify under the F. A bad track, due to a hard error, gets the letter E for error; and an error that was soft (i.e., an error occurred but a retry fixed it) is shown by the letter S for soft.

If you were to get a lot of S (soft errors), chances are the disk is old and worn out. A lot of E (hard errors) means the disk is not usable any longer.

Chapter 19

SYSGEN

CP/M Versions:

1.3 1.4 2.0 2.2

Command Type:

Transient

SUMMARY

The SYSGEN transient command, short for System Generation, is an elemental and necessary part of the CP/M operating system. It is used primarily for either copying or "generating" a CP/M operating system image from one disk to another. The CP/M system image is the actual operating system which is loaded into the computer's memory (RAM) each time the system is turned on and a "Cold Boot" is initiated. The first-time system user might logically assume that this CP/M system image is stored in a file somewhere on a disk. This is almost the case, but since a CP/M file can only be accessed *after* the CP/M operating system in RAM, there must be another way for the CP/M system to "start up" and get into RAM so it can run.

The technique used by CP/M is to store the operating system program on the first two tracks of the disk, called tracks 0 and 1. These two tracks are special and no other CP/M program can use them. They are referred to as the "system tracks." All floppies that are compatible with CP/M use these two tracks exclusively for storing the CP/M. The CP/M operating system cannot be accessed by the standard DR file manipulation commands, such as PIP, STAT, DIR, etc. Rather, a special program is used for moving the CP/M operating system program (copying it) from one disk to another. This program is called SYSGEN. SYSGEN stands for system generation and means it generates one system to another disk. Another pur-

pose of SYSGEN is in installing a fresh version of a modified operating system (one with its BIOS changed) onto another disk. This last use is usually only done by programmers or modifiers of the BIOS (see *Soul of CP/M*). Recall when the system is first turned on the boot or monitor program will get the operating system off the first two tracks and put in the proper place in RAM.

Some computers using CP/M have hard disks instead of, or in addition to, floppy disks. In some of these computers, the CP/M system image is stored somewhere other than the first two tracks. The way in which this is accomplished varies depending on the configuration. As a result, many manufacturers have modified the SYSGEN program to suit the needs of the particular computer.

This section will not describe how SYSGEN operates in such computers, since there are too many variations of system configurations. If you are using one of these systems, consult the manufacturer's technical manuals for information on how SYSGEN operates. This section will only explain how the standard, unmodified SYSGEN.COM program operates in computers using floppy disk drives as the main means of information storage.

ACCESSING SYSGEN

Normally, when the SYSGEN command is first used, a master floppy disk with the CP/M operating system image already stored on its first two tracks is needed. When your computer or new software is first installed, the initial step is to make a backup copy of the entire contents of the master disk. After the new backup floppy disk has been properly formatted (using the disk formatting program FORMAT, or equivalent), a copying program such as PIP (refer to the PIP description) is used to copy all the files, including the CP/M system image, over to the new disk. We use the SYSGEN command specifically to do this last step; copy the system image.

The SYSGEN transient program is usually stored in a file called "SYSGEN.COM." Some manufacturers or distributors may have named the file a little differently, such as "SYSGEN2.COM" to indicate that the command corresponds to the installed version of CP/M (Version 1.4, 2.0, 2.2, etc.). Whatever it's called, this file must be stored on a disk inserted in one of your computer drives; normally the A drive. In addition, the disk to which the CP/M system image is to be copied must be properly formatted. (For a description, see "FORMAT", Chapter 18.)

DESCRIPTION OF SYSGEN COMMAND VARIATIONS

COPYING THE CP/M SYSTEM IMAGE

SYSGEN

The next example shows the questions presented to the user when SYSGEN is used to copy the system image from the disk in Drive A to the disk in Drive B. Because the computer's questions are straightforward, answering them is quite simple.

```
A>SYSGEN
SYSGEN VER X.X
SOURCE DRIVE NAME (OR RETURN TO SKIP)A<CR>
SOURCE ON A, THEN TYPE RETURN<CR>
FUNCTION COMPLETE
DESTINATION DRIVE (OR RETURN TO REBOOT)B<CR>
DESTINATION ON B, THEN TYPE RETURN<CR>
FUNCTION COMPLETE
DESTINATION DRIVE (OR RETURN TO REBOOT)<CR>

A>
```

Note, the first prompt (request for information) displayed after typing SYSGEN is "SOURCE DRIVE NAME (OR RETURN TO SKIP)." This asks the name of the drive containing the disk on which the CP/M system image is stored. In response, type the name of the drive (A, B, C, etc.) containing the disk from which you wish to copy the system image. When typing the drive name, unlike the CP/M commands, there is no need to specify a colon (":") following the drive name. The SYSGEN program will automatically recognize your input as a valid drive name, as long as the drive name specified is valid and is recognized by CP/M. Our example shows that "A" was typed, since Drive A is usually the main system drive.

After pressing the RETURN key following the drive name in the first prompt, a second prompt, "SOURCE ON A, THEN TYPE RETURN" is displayed. The program is asking if you are sure that Drive A is to be the source drive. If you've made a mistake, then you can exit the program at this point by simply typing the CTRL/C (Warm Boot) sequence. If the specified drive is correct, press the RETURN key. Upon doing this, the SYSGEN program will load the contents of the first two tracks of the disk into the computer's memory. Once the system image has been successfully loaded into the memory, the confirmation prompt "FUNCTION COMPLETE" is displayed.

Once the system image is in the computer's main memory, the prompt "DESTINATION DRIVE (OR RETURN TO REBOOT)" is displayed. In response to this, enter the drive name containing the disk to which the system image is to be copied or generated. Enter it in the same manner as the response to the first "SOURCE" prompt. The same dialogue, mentioned previously, is repeated, confirming the destination drive. Again, once the system image has been successfully stored on the first two tracks of the disk in the destination drive, the prompt "FUNCTION COMPLETE" is displayed. The "DESTINATION DRIVE" prompt is then repeated. Now you can go ahead and copy the system image to another drive, or to a new disk inserted in the same original destination drive. If you want to do this, simply type in the destination drive name. Repeating the process is possible because the CP/M system image is still stored in the computer's memory. To exit from SYSGEN at this point, simply press the RETURN key, whereupon the original logged drive prompt (A>, B>, etc.) will reappear.

If an attempt is made to copy a system image from a disk without the correct information stored on its first two tracks, the following error message will be displayed.

```
A>SYSGEN
SYSGEN VER X.X
SOURCE DRIVE NAME (OR RETURN TO SKIP)A<CR>
SOURCE ON A, THEN TYPE RETURN<CR>
PERMANENT ERROR, TYPE RETURN TO IGNORE<CR>
PERMANENT ERROR, TYPE RETURN TO IGNORE^C

A>
```

This error message is displayed if the disk specified as the "SOURCE" has an incorrect format (i.e., wrong density, or non-CP/M compatible format). It

will also appear if the first two tracks are physically damaged, preventing SYSGEN from reading the data correctly. The same thing will happen if you attempt to copy the system image to an incorrectly formatted or unformatted disk in the "DESTINATION" disk drive. (Predictably, if there is no disk in the drive at all, the same error will appear.) Note that if the RETURN key is pressed in response to the error message, the SYSGEN program will again attempt to read from and write to the disk. If the program is unsuccessful, the same error message will be displayed. When this error message is received, you must enter the CTRL/C (Warm Boot) sequence to terminate SYSGEN. Once this is done, the original logged CP/M prompt (A>, B>, etc.) reappears and you can begin all over again.

<( )> SYSTEM TRACKS IN THE WRONG PLACE:

One drawback in the design of CP/M is the choice of locations for the storing of the operating system. It turns out that tracks 0 and 1 are the outermost tracks of the disk. They are therefore closest to the edge of the disk. Any electrical discharge occurring on the outside of the drive may also occur on the edge and therefore the operating system in tracks 0 and 1 will be the first to get zapped, scrambled, and put out of commission. Should this happen, your CP/M disk won't boot. In such a case, you can employ the SYSGEN program to reinstate the lost operating system and make the disk bootable again. Note that a good CP/M disk is needed along with the SYSGEN file.

SAVING THE CP/M SYSTEM IMAGE IN A FILE

```
SYSGEN
SAVE 41 CP/M-56K.SYS
```


The SYSGEN command can also be used to create a file containing the CP/M system image. Such a file can then be modified by the programmer. The SYSGEN command is used in conjunction with the built-in SAVE command to save the CP/M system image in a file. The SAVE command was described previously.

To use SYSGEN for putting the system image in a file, invoke the command in the usual manner by typing the command in response to the logged

drive name prompt. In response to the "SOURCE" drive prompt, type in the drive name containing the disk whose first two tracks store the desired system image. When this is done, the system image will be placed in the computer's memory, as indicated by the "FUNCTION COMPLETE" message. In response to the "DESTINATION" drive prompt, simply press the RETURN key, since you don't want to generate the system on the first two tracks of another disk. Once this is accomplished, the original prompt of the logged drive reappears and we are out of SYSGEN.

```
A>SYSGEN

SYSGEN VER X.X

SOURCE DRIVE NAME (OR RETURN TO SKIP)A<CR>

SOURCE ON A, THEN TYPE RETURN<CR>

FUNCTION COMPLETE

DESTINATION DRIVE (OR RETURN TO REBOOT)<CR>

A>
```

The system image is now temporarily residing in the scratch-pad area of RAM. The SAVE command is then used to save the contents of RAM into the specified file. Before using the SAVE command, please read the SAVE description presented earlier, since a good understanding of SAVE is very important when performing this function. While reading the description on SAVE, it will become apparent that to store all the necessary data in the specified file, the approximate number of memory pages must be calculated and specified.

The following example is a continuation of the previous example. It assumes that you have a CP/M system image generated for 56K. With the proper calculations, it can be determined that the required number of memory pages to be specified is approximately 41. This number may be higher than necessary, but it is better to work with a number slightly higher than one which is less than the exact amount. In this example, the name of the file specified to contain the system image is "CP/M-56K.SYS". The file name should indicate the size of the system image.

The extension should indicate that the file contains machine code as opposed to displayable text.




```
A>SAVE 41 CP/M-56K.SYS

A>

A>DIR CP/M-56K.SYS

A: CP/M-56K SYS

A>
```


Note that the extension ".SYS" tells us that the file "CP/M-56K.SYS" contains machine code. You can use the ".COM" extension instead of ".SYS" as it also indicates that the file contains machine code. ".COM" should be avoided for this type of file, however, as it usually indicates an executable "command" file. Since "CP/M-56K" is the CP/M system image and not an executable command, an extension such as ".SYS" or something similar is suggested.

COPYING A CP/M SYSTEM IMAGE FROM A FILE

SYSGEN CP/M-56K.SYS

As described in the previous paragraphs, SYSGEN can accept a file containing a CP/M system image, created with MOVCPM, and move this to the system tracks of any disk. The file specified for this purpose must be one that was originally created using MOVCPM or some other means. The file must contain the system image in a special form of relocatable machine code. To use SYSGEN this way, type in the SYSGEN command, a space, and the file name containing CP/M, followed by the RETURN key.

The next example shows how a file called CP/M-56K.SYS is moved onto a disk in drive B:.

```
A>SYSGEN CP/M-56K.SYS

DESTINATION DRIVE (OR RETURN TO REBOOT)B<CR>

FUNCTION COMPLETE

DESTINATION DRIVE (OR RETURN TO REBOOT)<CR>

A>
```

In this example, note that the "SOURCE" drive prompt is skipped, as CP/M is capable of passing on the file name information to the SYSGEN.COM program. Subsequently, the program immediately assumes that the file name specified is the source. The example also assumes that "CP/M-56K.SYS" is on Drive A, (as that is the logged drive), and that no drive name preceded the file name. You can, however, specify any drive name recognized by CP/M to precede the file name. Remember that there must always be a space between "SYSGEN" and the disk drive/file name.

If, by chance, a nonexistent file is typed in following SYSGEN, the following error message will appear.

```
A>SYSGEN CP/M-01K.SYS
SYSGEN VER X.X
NO SOURCE FILE ON DISK

A>
```

Note the original logged CP/M prompt (A>, B>, C>, etc.) reappears immediately following the error message.

Chapter 20

MOVCPM

CP/M Versions:

1.3 **1.4** **2.0** **2.2**

Command Type:

Transient

SUMMARY

The MOVCPM transient command, short for Move CP/M, is primarily used to "move" the CP/M operating system image to a different memory size. For every computer running CP/M, normally the system image must be configured to use the total RAM which is available. Since this amount may vary from computer to computer between 20K and 64K, it is necessary that CP/M be adjustable. When we use MOVCPM and the system image is changed from one size to another, it is, in effect, being moved to operate in a new location. In addition to being able to adjust CP/M to run in different amounts of RAM, MOVCPM is also used to relocate CP/M slightly downward to make room for any machine language programs which need to be stored outside (and above) CP/M.

When the CP/M operating system is first installed in a computer, the system disk contains the operating system and all of the commands. The system image on the disk is normally first configured for operation on a computer with 20K of memory for CP/M Version 2.0 and above. With CP/M versions below 2.0, this amount will be a minimum of 16K. This means your version of CP/M does not take advantage of the complete memory space. If your system contains more memory, however, such as 32K, 48K or 64K, you will want to configure CP/M for the maximum amount. This is important, because if you install a 20K system image on a system with 32K of memory, only 20K of the memory will be used by CP/M and the other 12K will be

ignored. The MOVCPM command is thus used to move the system image so that the system's memory is used to its fullest extent.

For CP/M Versions 1.3 and 1.4, the system image can be configured or moved to a memory size ranging from 16K to 64K. For CP/M Version 2.0 and newer, the range is from 20K to 64K. The number you select will depend on the actual size of the computer's available RAM. For most system configurations, the actual CP/M system image will require approximately 11K of memory, leaving the remaining amount specified with the MOVCPM command for programs in the TPA. For example, if you specify that the system be moved to 64K, approximately 11K will be used for the system image itself, and the remaining 53K will be available for use as scratch pad memory. You can find the locations of your CP/M and the exact amounts available for programs in *Soul of CP/M*.

Some microcomputers have a section of memory reserved for other than CP/M related uses. For example, if your computer is installed with a 64K RAM board, and 4K is reserved for use by the floppy disk controller hardware, then you will not be able to move the CP/M system image to exactly 64K. Only 60K is available for use by the CP/M operating system. Some computers have as much as 8K reserved for special purposes, leaving only 56K for CP/M.

There are two basic ways in which the MOVCPM command can be used. They are:

1. To move the CP/M system image so that it utilizes all or a specified amount of RAM, and transfer control over to the newly moved CP/M in memory.
2. To move the CP/M system image so that it utilizes all or a specified amount of RAM and then instead of transferring control to it immediately, rather saving the system image in a file and then moving it into the system tracks with SYSGEN.

ACCESSING MOVCPM

To access the MOVCPM command, the file "MOVCPM.COM" must exist on a disk inserted in a disk drive. The CP/M operating system must have been booted into RAM for the command to operate correctly.

When the command is initiated, the actual system image stored in the computer's memory is moved. The command does not reference the first two tracks of the disk inserted in the system disk drive like SYSGEN does.

DESCRIPTION OF MOVCPM COMMAND VARIATIONS

MOVE CP/M SYSTEM IMAGE USING ALL MEMORY AND TRANSFER CONTROL



This is the simplest form of the command. It instructs CP/M to move itself by utilizing all of the computer's available memory (16K, 32K, 48K, etc.). Once this is done, control of the computer is transferred over to the newly moved system image. With this form of the command, the system image is NOT configured so that it can be saved on a disk. The following example shows how this is done.

```
A>MOVCPM

CONSTRUCTING 48K CP/M ver. X.X

48K CP/M vers. X.X

A>
```

The example assumes that the maximum amount of system memory is 48K. Because the CP/M system image itself requires a certain amount (usually 11K) of this memory for its own storage, approximately 40K will be available for scratch pad purposes. In the first line of text in the example, the version number of CP/M (1.4, 2.0, etc.) is displayed along with the size of the system. The next line of text is the exact duplicate of the text received when first booting CP/M via the Cold Boot sequence, except that the size of CP/M will differ if the original system image utilized more or less memory.

MOVE CP/M USING SPECIFIED MEMORY AND TRANSFER CONTROL

MOVCPM nn

This form of the MOVCPM command is similar to the one described previously, except that, in this case, a specific amount of memory (16K, 32K, 48K, etc.) to be used is indicated. This form of the command is often used to optimize the size of the system image for your computer. It is also useful when certain areas in RAM are not operating correctly and need to be isolated so that CP/M does not reference them. (With this form of the command, the system image is not conditioned so that it can be saved on a disk.) It is also useful for quickly giving space for machine language programs to be stored above CP/M.

This form of the command is entered by typing "MOVCPM" and a space, followed by a value (nn) representing the amount of RAM you want CP/M to think your computer has in response to the logged drive prompt. If, for example, the final system image is moved to 56K, a value of "56" is what should be specified for nn. Once the command sequence has been entered, control of the computer is transferred over to the newly moved system image. The next example shows how this is done.

```
A>MOVCPM 56

CONSTRUCTING 56K CP/M ver. X.X

56K CP/M vers. X.X

A>
```

This example assumes that the maximum amount of system memory to be utilized by CP/M is 56K. Because the CP/M system image requires a certain

amount of this memory for its own storage, approximately 42K will be available for scratch pad purposes.

If the value you enter is greater than the available RAM, the MOVCPM program will still move the system image to the maximum permitted. If, for example, you had entered "50" for 50K when only 48K of memory was available for use by CP/M, the MOVCPM program would attempt to move the system image to that amount. As a result, an error would occur since all the memory specified could not be allocated. In your 48K machine, the same error would occur with any amount between 49 and 64, as there still would not be sufficient space in the memory for the entire system image. The error message received may vary between systems, but the most common would be "DRIVE NOT READY" because the new system image, or what is left of it, will not be able to reference your system disk drive. To solve this problem you have to reboot (Cold Boot) the system to load the original system image from the first two tracks of the disk in the system drive and start over. This is demonstrated in the following example.

```
A>MOVCPM 50

CONSTRUCTING 50K CP/M ver. X.X

DRIVE NOT READY
```

(A Cold Boot would be initiated at this point, to boot in the original system)

<( )> MAXIMUM AND MINIMUM RAM FOR MOVCPM:

MOVCPM nn requires a maximum and minimum amount which may be entered for the MOVCPM program to operate correctly. In CP/M Versions 1.3 and 1.4, the range is from 16K through 64K. In CP/M Version 2.0 and newer, the range is from 20K through 64K. If your computer has 64K of memory in and you specify 65 or higher with the MOVCPM command, you'll have a problem. The program will attempt to move the system image to "01K5" since 64K is the maximum available memory and exceeding it causes the computer to allocate 65K back at location 01K. This would be highly undesirable as the new system image would not operate. If this occurred, you would have to reboot (Cold Boot) the system, because an attempt was made to transfer control over to the inoperable newly moved system image.

MOVE CP/M USING ALL OF MEMORY AND PREPARE SYSTEM IMAGE FOR SAVING IN A FILE

**MOVCPM * *

This form of the command is the same as the first form of MOVCPM described in this section, except that control is not transferred to the newly moved system image. Instead, the new system image is placed in RAM so that it can be retrieved and saved in a disk file. MOVCPM * * instructs CP/M to move itself by utilizing all of the available memory in the system (16K, 32K, 48K, etc.). Once this is done, the SAVE command is used to put the memory image of CP/M (now moved) into a file, usually called CP/M-XX.COM or CP/M-XX.SYS, where xx is the size of the system being saved. Then the SYSGEN transient command (described previously) can be used to take the system image from this file and install it on the first two tracks of a disk in a specified drive (control is still with the original system until a reboot is performed). Alternately, the SYSGEN program can take the system image right out of memory (after MOVCPM has put it there) and install it onto the system tracks. Control will not be transferred to the new system until a reboot of the disk is performed.

The command sequence is entered by typing "MOVCPM" and a space, followed by an asterisk ("*"), another space and a final asterisk. If no space is placed between "MOVCPM" and the asterisks, or no space is placed between the two asterisks, the program will not operate as intended. The next example shows how the MOVCPM command is used and how the SYSGEN and SAVE commands are used to save the system image on a disk.

```
A>MOVCPM * *

CONSTRUCTING 64K CP/M ver. X.X
READY FOR "SYSGEN" OR
"SAVE 45 CPM64.COM"

A>
```

This example assumes that the maximum amount of system memory is 64K. Once the system image has been moved and placed in memory, the

program informs the user that either the SYSGEN or the SAVE command may be used to save the system image on a disk. Note that the MOVCPM program automatically calculates the number of memory pages that must be specified to save the system image in a file with the SAVE command. The program also presented a suggested file name for the file, as "CPM64.COM." You can call the file anything you wish, but it is important that the name identify the file as a machine or object code file rather than a text file. The extension ".COM" could be used, but it is suggested that an extension such as ".SYS" be used instead, because ".COM" usually indicates that the file is an executable command file. The file in which you store the system image will not be an executable command file.

<( )>SIZE OF YOUR CP/M SYSTEM IMAGE:
Note that the number in the "SAVE . . . " message tells us how big our CP/M is. As we saw in the SAVE description, the number represents the number of 256-byte pages to save in the file. In our example we show 45 pages; and since 45 x 256 = 11,250 bytes, our CP/M is about 11K bytes long.

The following example shows how the newly moved system image is saved in a file called "CPM-64K.SYS" by using the built-in or direct SAVE command.

```
A>SAVE 45 CPM-64K.SYS
A>
A>DIR CPM-64K.SYS

A: CPM-64K   SYS

A>
```

The next example shows how the SYSGEN transient command is used to place the newly moved system image on the first two tracks of the disk inserted in Drive A.


```
A>SYSGEN
SYSGEN VER X.X
SOURCE DRIVE NAME (OR RETURN TO SKIP)<CR>
DESTINATION DRIVE (OR RETURN TO REBOOT)A<CR>
```

```
DESTINATION ON A, THEN TYPE RETURN<CR>
FUNCTION COMPLETE
DESTINATION DRIVE (OR RETURN TO REBOOT)<CR>

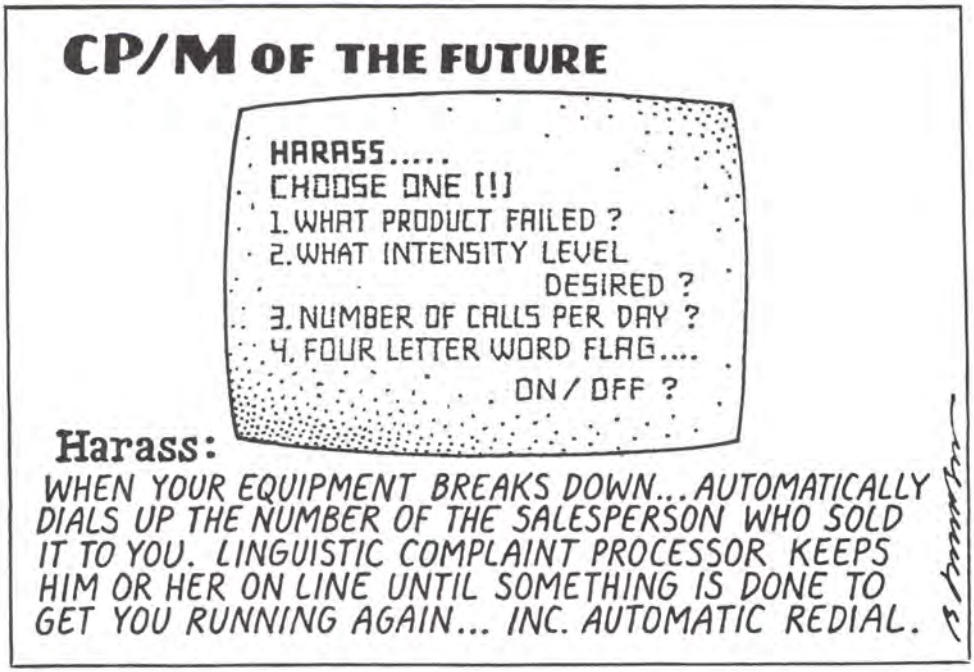
A>
```

Note that in this example, the RETURN key is pressed in response to the "SOURCE DRIVE" prompt. This is so because the system image already exists in memory. In response to the "DESTINATION DRIVE" prompt, the name of the drive is entered, in this case "A". This causes the SYSGEN program to over-write (i.e., write over) the first two tracks of the disk in Drive A with the newly moved system image. You could continue to use the SYSGEN program by repeatedly entering disk drive names in response to the "DESTINATION DRIVE" prompts as they are displayed. This would work since the system image is still in memory.

MOVE CP/M SYSTEM IMAGE USING SPECIFIED AMOUNT OF MEMORY AND PREPARE SYSTEM IMAGE FOR SAVING IN A FILE

MOVCPM nn *

This form of the command is similar to MOVCPM nn in that it allows you to specify how much memory you want allocated to CP/M. It is similar to MOVCPM in that it does not allow you to transfer control of CP/M. MOVCPM nn * instructs CP/M to move itself by utilizing a specified amount of RAM (16K, 32K, 48K, etc.). The command lets you create a system image that can be saved in a file that leaves room above CP/M for machine language programs. This form of the command is entered by typing "MOVCPM" and a space, followed by the specified amount RAM, another space, and an asterisk ("*"). Once the command sequence has been entered, the system image is placed in memory so that you can use the SYSGEN transient command or the built-in SAVE command to save the system image on a disk. By using this command, control of the computer is maintained by the original system image. The newly moved system image has no control. The following example shows how this is done.




```
A>MOVCPM 48 *

CONSTRUCTING 48K CP/M ver. X.X
READY FOR "SYSGEN" OR
"SAVE 45 CPM48.COM"

A>
```

Note that the system image was moved to 48K. The SYSGEN command can be used to save the system image on the first two tracks of a disk. Alternately, the built-in SAVE command can be used to save the system image in a file. If this was a computer with 64K of available working RAM memory, then the above CP/M system would free up $64 \times 48 = 16$K for other programs, or free space for buffers, etc.

CHAPTER 21

CP/M Plus (Ver. 3.0) Overview

INTRODUCTION

The latest version of the CP/M operating system, Version 3.0, called *CP/M Plus*™, succeeds Version 2.2 described in this book and contains many changes from previous versions which are proving to be significant advancements in 8-bit operating system technology. Although CP/M Plus retains most of the main and standard characteristics of previous versions of CP/M, the changes that have been made are significant both to experienced CP/M users and programmers. Several new commands have been added to the operating system, some of which are enhancements to the usage of CP/M in general, and others which are unique and essential to the environment in which CP/M Plus operates. Most of the commands provided by Version 2.2 are carried over, and almost all of them have had their operation improved upon. This chapter provides an overview of these new commands, as well as the changes made to commands carried over from previous versions.

OPERATING ENVIRONMENT

The command differences between CP/M Plus and previous versions are significant to both the computer operator and programmer. This is partially due to the operational changes made to the operating system itself which increases the types of environments in which it can operate. These changes

are immediately significant to the programmer, since system configuration, system interfacing, and program generation have become more complex due to the greater flexibility offered by the operating system. To the user, these changes are reflected by the new commands, enhancements to commands of previous versions, and the final configuration for the system on which CP/M Plus is operating.

CP/M Plus is available in two versions: Non-Banked and Banked. Due to the continuing development and decrease in cost of solid-state memory devices, it has become desirable and often essential to microcomputer users to increase the memory capacity of their systems. The Non-Banked version of CP/M Plus is designed for operation on a system with a maximum of 64 kilobytes of Random Access Memory (RAM), as is the case with the previous versions of CP/M. Although CP/M can be especially modified to operate with more memory in certain banked-memory systems, the Banked version of CP/M Plus has this modification built in. All the commands and functions available with the Non-Banked version are also available in the Banked version. The Banked version, however, has some command and system functions which are not available in the Non-Banked version. Throughout this chapter, these differences will be indicated.

Many changes have been made to the internal components or modules of CP/M Plus, such as the BIOS (Basic Input/Output System) procedures, BDOS (Basic Disk Operating System) calls, and the overall method in which memory is used. These changes are of particular importance to CP/M programmers modifying the operating system or writing utility programs. For brief reference information on BDOS calls in CP/M Plus, refer to the CP/M Versions comparative chart at the end of Chapter 2. For detailed information on the internal structure of CP/M Plus, it is suggested that you read the companion to this book, *The Soul Of CP/M*.

USING CP/M PLUS

The CP/M Plus operating system is used in much the same way as in previous versions of CP/M with some changes. These changes are described in the following paragraphs.

BOOTING THE CP/M PLUS SYSTEM: The booting (cold boot) process for CP/M Plus is the same as that described in Chapter 2, except for one consideration. In many applications, the system image is too large to be

stored on the first two tracks of the system disk, due to specially modified architectures, or components therein, such as the BIOS. When this is the case, the complete operating system image, or a part of it, is stored in a file called "CPM.SYS" which must be stored on the disk from which the system is to be booted. If the system image is not too large to be stored on the first two tracks of the system disk, then this file is optional and need not be present. The COPSYS transient utility command, which performs the same function as the SYSGEN command in previous versions of CP/M, provides the option to copy CPM.SYS along with the data on the first two tracks of the disk.

COLD-BOOT START-UP PROFILE FILE: Every time CP/M Plus is cold booted, the operating system will automatically search for a file called "PROFILE.SUB", which, if found, is submitted to the CP/M command line as if submitted by the SUBMIT command. This file can be structured like any SUBMIT file. If the file is not found, CP/M will continue operation in the normal fashion. This file can contain any commands the user wants executed each time the system is cold booted or reset, such as setting the time of day and date, making physical to logical device assignments, displaying a text file to the console device, etc.

USING DISK DRIVES AND USER NUMBERS: The user of CP/M Plus may use disk drives and user numbers in the same way as is done in CP/M Versions 2.0 and 2.2. The CP/M system (logged disk drive) prompt displayed by CP/M Plus has been enhanced so that it not only displays the disk drive name (A>, B>, C>, etc.), but also the User number (established with the USER command) currently being used, if the number is greater than 0. So, if the logged disk drive is A and the user area is 0, then "A>" will be displayed as the prompt, until another user number is specified, such as "1" or "2", at which point the prompt will change to "1A>" or "2A>". CP/M Plus will always include the user number in the prompt in this manner for Users 1 through 15, regardless of the logged disk drive. The user number will be absent from the prompt only if the current user number is "0".

ACCESSING DISKS AND FILES, AND THE USE OF PASSWORDS (BANKED VERSION OF CP/M PLUS ONLY): In CP/M Plus, disks and files are accessed in basically the same manner as described in Chapter 2. In the Banked version of CP/M Plus, however, the user has the option to utilize passwords with disks and files. Passwords can be assigned to disks and files by using special commands. If a disk is assigned a password, the user will be

prompted by the system for correct password before the disk can be used. If a file has a password assigned, the password is added to the filespec as a command trailer, in the following format:

x:filename.typ;password

where "x:" is the disk drive name, "filename" is the name of the file, ".typ" is the file type, and ";password" is the associated password. There are four types of password protection modes, as follows:

READ: A file in READ mode requires that a password be used before the file can be read, written to, copied, deleted, or renamed.

WRITE: A file in WRITE mode requires that a password be used before the file can be written to, deleted, or renamed. A password is not required to read or copy the file.

DELETE: A file in DELETE mode requires that a password be used before the file can be deleted or renamed. A password is not required to read, copy, or write to the file.

NONE: When in NONE mode, no password is required for the file.

DISK DRIVE CHAINING: CP/M Plus provides a new capability whereby transient command files and SUBMIT files can be automatically searched for through a specified "chain" of disk drives. Such a search will only occur if a disk drive name is not included in the filespec of the command line. A searched for file will only be found by CP/M if it has the "SYS" (system) attribute assigned. The SYS attribute is basically the same as that described for the STAT command in Chapter 8. If this attribute is not assigned the file being searched, CP/M will respond with a "file not found" message. The characteristics of drive chaining are controlled by the SETDEF command.

LOGICAL AND PHYSICAL DEVICES: The arrangement of logical and physical devices in CP/M Plus is considerably different as compared to the previous versions of CP/M. CP/M Plus supports five basic logical devices, as follows:

CONIN: or KEYBOARD: (console input) } or CON: (console device)

CONOUT: (console output)

AUXIN: (auxiliary input) } or AUX: or AUXILIARY:

AUXOUT:	(auxiliary output)	(auxiliary device)
LST: or PRINTER:	(listing output)	

There are a variety of physical characteristics that may be assigned to the logical devices in CP/M Plus. Not only can a variety of physical devices be assigned, but several standard characteristics can be assigned, in addition, such as direction of data flow (input-only, output-only, or bidirectional), data speed rate in bits per second (bps), type of interface (serial or non-serial), and protocol (such as X-ON/X-OFF). These characteristics, along with the physical devices, are assigned to logical devices by using the DEVICE command. The DEVICE command can also be used to display current assignments. In addition to the standard physical devices supported by previous versions of CP/M (CRT:, TTY:, LPT:, etc.), special physical devices can be defined in the BIOS of the system, each having unique names of up to six characters in length. These special physical devices could represent special device drivers and handlers built in to the BIOS, and can be assigned or reassigned at any time with the DEVICE command. Examples of special physical devices might be: MODEM1 for a certain type of modem or other data communications device, DIABLO for a special driver to control daisy wheel impact printers, DOT1 for high-speed dot-matrix printers, etc.

TIME AND DATE STAMPING: CP/M Plus has the built-in capability of being modified to reference realtime clock circuitry in the system's hardware. This capability permits the user to set the current time of day and date with the DATE command, as well as reference it at any time. If the time and date capability is enabled in the system, the INITDIR command can be used to enable time and date stamping on files. This stamping can be set up to occur only under certain circumstances, such as only when a file is created, only when a file is updated, or both. Time and date stamps for individual files can be displayed in file directories when command parameters are used either with the DIR or DIRSYS commands. When time and date stamping is enabled, and the system's clock is referenced correctly, the stamping process for individual files occurs automatically.

USING WILDCARDS: Wildcards are used in CP/M Plus in much the same way as in previous versions of CP/M (see Chapter 2), with the exception that they can now be used with the REN or RENAME command. To rename a group of files, the "*" or the "?" wildcards can be used, as long as they are placed in both the "old" filespec and the "new" filespec, in the same relative

positions. For example, the following sequences are legal for the RENAME command:

```
A>REN *.BK1=*.BAK
A>REN A*.BK?=B*.SK?
```

Whereas, the following sequence will cause an error message when used with the RENAME command:

```
A>REN A*.TXT=B.T*
```

Apart from the RENAME command, wildcards may be used in the same way as with previous versions of CP/M.

CP/M PLUS COMMAND OVERVIEWS

The commands provided by CP/M Plus consist of basically three types: Built-In, Transient commands, and Control Character commands. Many of the commands can be recognized as the same as those provided by previous versions of CP/M. However, almost all of these commands have had their operation enhanced. In the case of the built-in commands, some of them require accompanying transient command files for complete operation. The commands provided by CP/M Plus are listed in the next paragraphs.

BUILT-IN COMMANDS: The built-in commands provided by CP/M Plus are stored in the operating system itself, permitting the user to execute the commands regardless of the disk being used. There are three commands which are the exception, however; although they can be used as normal built-in files, to attain additional special parameters each must have an associated transient command file stored on the disk in the logged disk drive. This optional transient command file is used only for reference by the built-in command. CP/M Plus differs from previous versions of CP/M in that all built-in command names can either be typed in full or can be abbreviated to as few as three characters. The following list presents the CP/M Plus commands on the left side, any optional reference transient file in the middle, with the equivalent standard command or command sequence, if any, in CP/M Version 2.2.

CP/M Plus	Optional Ref. File	CP/M 2.2
DIR	DIR.COM	DIR (Replaces STAT functions)
DIRS or DIRSYS	DIR.COM	–(Replaces STAT functions)
REN or RENAME	RENAME.COM	REN
ERA or ERASE	none	ERA
TYP or TYPE	none	TYPE
USE or USER	none	USER
SAV or SAVE	none	SAVE

TRANSIENT COMMANDS: The transient commands in CP/M Plus operate in much the same way as with previous versions of CP/M. To execute a transient command, the command file with the file type ".COM" must be stored on the referenced disk. CP/M Plus will also consider files with the file type ".PRL" (Page Relocatable file) as valid transient command files. The ".PRL" file type is the standard file type used for transient command files in the MP/M-II and MP/M-86 multiuser, multitasking operating systems. Some of the transient commands from previous versions of CP/M are also used in CP/M Plus, whereas others are replaced with new commands. The following list presents the CP/M Plus commands on the left side, with the equivalent standard command or command sequence, if any, in CP/M Version 2.2.

CP/M Plus	CP/M 2.2
COPSYS	SYSGEN
DATE	–
DEVICE	– (replaces STAT functions)
DUMP	DUMP
ED	ED
FORMAT (special)	FORMAT (special)
GENCPM	MOVCPM
GENCOM	–
GET	–
HELP	–
HEXCOM	LOAD
INITDIR	–
LIB	optional
LINK	optional
MAC	optional (replaces ASM)
PATCH	–
PIP	PIP
PUT	–
RMAC	optional (replaces ASM)

CP/M Plus	CP/M 2.2
SET	– (replaces STAT functions)
SETDEF	–
SHOW	– (replaces STAT functions)
SID	optional (replaces DDT)
SUBMIT	SUBMIT
XREF	optional

CONTROL CHARACTER COMMANDS: Control character commands consist of such commands as CTRL/C which initiates a Warm Boot sequence, CTRL/S which freezes console output, CTRL/H which causes a backspace, etc. All the control character commands provided by previous versions of CP/M are also provided in CP/M Plus. In both the Non-Banked and Banked versions of CP/M Plus, the CTRL/Q control sequence command is used to resume scrolling on the console screen after a CTRL/S (stop scroll) command was issued. In the Banked version of CP/M Plus, some new line editing control character commands have been added. Note that these new commands are provided only in the Banked version, not in the Non-Banked version. They are:

New Line Editing Commands (CP/M Plus BANKED Version Only)	
CTRL/A	Moves the cursor one character to the left.
CTRL/B	Moves the cursor to the beginning of the command line without affecting the contents of the line. If the cursor is at the beginning, CTRL/B moves it to the end of the line.
CTRL/F	Moves the cursor one character to the right.
CTRL/G	Deletes the character at the cursor position. The cursor does not move.
CTRL/K	Deletes all characters from the cursor position to the end of the line.
CTRL/W	Recalls and displays previously entered command line both at the operating system level and within executing programs, if CTRL/W is entered as the first character after the prompt. Command lines can be recalled after either the CTRL/U, CTRL/M, or CTRL/J sequences were entered, or the RETURN key was pressed. If the recalled command line contains characters, the cursor is moved to the end of the command line. If RETURN is pressed after the recalled command, the command will be executed.

New Line Editing Commands (CP/M Plus BANKED Version Only) Cont.	
CTRL/X	This command has been improved over the previous versions of CP/M. When entered, it only deletes the characters to the left of the cursor, bringing remaining characters to the right of the cursor to the beginning of the command line.

The following are short overviews of each of the commands provided with CP/M Plus.

DIR AND DIRSYS: The DIR and DIRSYS (or DIRS) commands perform the basic function of displaying a disk's directory of files. DIR and DIRSYS perform the same function, except that DIRSYS is used to display "system" files assigned with the SYS attribute. Up to eighteen different optional parameters may be included with the command to perform different functions. The parameter used determines the type of information to be displayed, such as: display the directory with time and date stamps; display the directories of all or specified disk drives; display all except certain specified files and/or disk drives; display directories of all or certain USER areas; etc.

RENAME or REN: This command differs from previous versions of CP/M in the following ways:

1. Wildcards may be used as described previously in this chapter.
2. Either RENAME or REN may be entered to access the command.
3. A single command string may be entered, or the command name may be entered on its own for user dialog. If entered on its own, the command will ask for NEW NAME, and then OLD NAME.
4. Provides the option to delete an existing "NEW NAME" file.
5. When either option 3 or 4 is used, the built-in command will reference the transient command RENAME.COM. This must be on the logged disk drive or stored in one of the chained drives for these options to be used.

ERASE or ERA: This command differs from the previous versions of CP/M in the following ways:

1. The "ALL?" confirmation message after a "*.*" (all files) sequence is entered has been replaced by "CONFIRM (Y/N)?". In CP/M Plus,

this confirmation message will display when any wildcard is used in the command string (not just "*.*").

2. The "[CONFIRM]" optional parameter may be added to the command string to confirm each file to be deleted when wildcards are used.
3. In the Banked version of CP/M Plus only, the "[PASSWORD]" optional parameter may be used to prompt for the password before a file is erased.

TYPE: This command is identical to that of previous versions, except it will ask for the filespec to be entered if the command is entered by itself. A new optional parameter has been added. The "[PAGE]" parameter may be added to the command string to cause a pause at each 24th screen line until any key is pressed to resume.

USER: CP/M Plus provides the following changes to this command as compared to previous versions:

1. USER may be entered alone, displaying the dialog "ENTER USER #:"
2. The CP/M prompt will now display the user number along with the logged disk drive, as in MP/M (5A>, 10C>, etc.), except when current user is 0.
3. Files can now be copied into a user area with PIP command by specifying the [Gn] option with the destination filespec, and optionally another [Gn] option with the source filespec.
4. Files can be searched for in other user areas via the disk drive chaining arrangement, if the files are SYS (system) files.
5. The SHOW command replaces the function STAT performs in previous versions, with expanded information; DIR can display files in other user areas (individual user areas or all of them).

SAVE: The SAVE command is considerably different from previous versions of CP/M, and its operation is of significant importance to programmers. To save data stored in the system's memory into a file, the SAVE command is entered by itself *prior* to loading the data into memory. The primary method of loading the data into memory would be to use the SID (Symbolic Instruction Debugger) to load a particular file after the SAVE command has been entered. Upon exit from SID, the SAVE command will automatically execute, and will prompt the user for the name of the file to be saved. Once

this has been entered, the SAVE command will check for any existing file stored by the same name and, if found, will ask if it is to be deleted. The SAVE command then asks for starting location memory of the data to be saved, and then the ending location. Once all this information is entered, the data will be saved in the file specified.

COPSYS: This command provides the same function as SYSGEN (see Chapter 19) in previous versions of CP/M. Since the system image of CP/M Plus is sometimes stored or partially stored in a file called CPM.SYS, the COPSYS command will ask if it is to be considered in the copy operation.

DATE: The DATE command is provided by CP/M Plus to provide access to the time of day and date functions. The command can be used in the following ways:

1. Enter DATE to display time and date and then return to CP/M.
2. Enter DATE C to continually display time of day count.
3. Enter DATE (date-time) to set time and date, and then receive "press any key to set time" prompt.
4. Enter DATE SET to receive program dialog for entering date and time.

Note that this command will only be operational if your microcomputer has a system clock, and CP/M Plus has been modified to access the clock circuitry. The time of day and date will have to be set each time the system is powered up or reset, unless the clock circuitry has battery backup and CP/M Plus has been modified to use this backup process.

DEVICE: This command replaces the function of STAT in previous versions of CP/M for the process of assigning logical and physical devices. The DEVICE command may be used in the following ways:

1. Enter DEVICE alone to display the user predefined physical devices and their characteristics (Input, Output, Serial, XON/XOFF), and then the current physical-to-logical device assignments. Once displayed, program will pause for user input for device reassignments.
2. Enter DEVICE NAMES to display only the user predefined physical devices as above.
3. Enter DEVICE VALUES to display only the current physical to logical device assignments.

4. Enter DEVICE (logical or physical device name) to display information only about that device.
5. Assign a physical device to a logical device (as in STAT), or make a logical device not assigned to anything (i.e., logical device = NULL). Options can be added in the command string, enabling or disabling XON/XOFF, and setting the data speed (bps) rate. More than one physical device may be assigned to a logical device.
6. Assign attributes to a physical device (enable or disable XON/XOFF, and/or set the data speed (bps) rate). These attributes are carried over with a later assignment of the device to a logical device.
7. Display or set the current console device screen size (number of columns and lines).

DUMP: This command is much the same as that in previous versions of CP/M, although in CP/M Plus, it has the capability of displaying object code data in both hexadecimal and ASCII formats. The hexadecimal code is displayed on the left side, while the corresponding ASCII data is displayed to the right. The display format is almost identical to when the "D" (display) command is used in SID (see Appendix A) or DDT (see Chapter 15).

ED: The line text editor ED in CP/M Plus has the following differences from the previous versions of CP/M:

1. A different output file name may be specified along with the input file name. The ED command may be entered alone, causing the program to first ask for the input (source) file name, and then the output file name which may be optionally entered.
2. The following new ED commands have been added to CP/M Plus:
   a. :ncommand – This command string instructs ED to execute the command specified through to line "n".
   b. nXfilespec^Z – This command instructs ED to write "n" number of lines of text to the filespec specified. If a previous write operation was made using the same command to the same filespec, the lines written in a subsequent use of the command will append the previously written lines in the filespec.
   c. 0X^Z – This command will delete the library file X$$$$$$$.LIB.
   d. 0Xfilespec^Z – This command will delete the specified filespec.

FORMAT: The FORMAT command, as with previous versions of CP/M, is an optional, nonstandard utility in CP/M Plus. A program of this type is

needed to format floppy disks to different densities for CP/M compatibility. This command is normally provided by the distributor of your CP/M Plus package, and will have been modified for the particular type disk drive used with the system.

GENCPM: This command performs basically the same function as MOVCPM in previous versions of CP/M (see Chapter 20). It performs the basic function of changing the CP/M system image so that it utilizes the system's memory in the most efficient manner.

GENCOM: This command is used in CP/M Plus to attach special RSX (Resident System Extensions) to existing transient utility command files. An RSX is a special type of program that can be attached to the operating system to modify or extend the capability of the BDOS area of CP/M Plus. RSX files or modules are used to intercept BDOS functions and either perform them, translate them into other BDOS functions, or pass them through untouched. More information can be obtained on RSX modules and how they are used, in the companion to this book, *The Soul Of CP/M*.

GET: This command is used to specify a file that is to be the source of commands, entries, etc. that would otherwise be entered manually on the console device, and automatically execute them. The GET file contents may consist of CP/M commands, entries to programs already running, etc. Optional parameters are:

1. The SYSTEM parameter may be used to determine if a program executing is to have immediate input from the GET file; when omitted, the program executing will receive input from the GET file only when needed.
2. The ECHO parameter specifies that input from the GET file is to be echoed on the console device; the NO ECHO parameter specifies that input is to be invisible.
3. The GET CONSOLE parameter reverts everything back to manual console input only.

HELP: This is a standard command provided by CP/M Plus, which provides the user with the ability to ask for help information on specified commands. The command is accompanied by HELP.HLP, the source text file containing all of the help information referenced by the HELP.COM. This help file may be converted to HELP.DAT so that the source text may be

modified or expanded using a standard text editor or word processor. This command is used in these manners:

1. Help is obtained in the using the following command syntax:
   A>HELP {topic} {subtopic1} . . . {subtopic8}
   where "topic" is a CP/M Plus command, "subtopic1" is a first level parameter associated to "topic", "subtopic2" a second level parameter, and so on. Up to eight subtopics may be specified.
2. The [P] parameter may be used to prevent pausing every 23 lines.
3. The HELP> prompt is presented after each help display, until RETURN is pressed.
4. The [E] parameter is used to change the HELP.HLP source text file to HELP.DAT so that it may be edited with a text editor or word processor.
5. The [C] parameter is used to convert HELP.DAT back to HELP.HLP.

HEXCOM: This command is same as the LOAD command (See Chapter 14) in previous versions of CP/M. Only the name has changed.

INITDIR: This command is used to initialize a disk's directory to enable time of day and date stamping. This command can only be used if CP/M Plus utilizes the system's clock, if available.

LIB: This utility is provided by CP/M Plus on a standard basis, and is the same command as is optionally provided by previous versions of CP/M. See Appendix A for an overview of this command. The following basic functions are provided:

1. Create a library file consisting of several .REL (relocatable object code) files.
2. Create an indexed library file.
3. List module names within a library or other .REL file.

LINK: This is a standard CP/M Plus command, and is optionally available for previous versions of CP/M. It provides the function of linking several relocatable object code files and loading the result into a single executable transient command file. See Appendix A for an overview of this command.

MAC: This is a standard CP/M Plus command, and is optionally available for previous versions of CP/M. MAC is a macroassembler which greatly

expands upon the capabilities of the ASM command (see Chapter 13) in previous versions of CP/M. The ASM command is not provided in CP/M Plus. Refer to Appendix B for an overview of the MAC command.

PATCH: This is a special command which is used to install patches (program modifications and upgrades) to the CP/M Plus operating system or to CP/M Plus files. PATCH is used to automatically install upgrade patches sent to the system owner from the distributor or from Digital Research.

PIP: The PIP command (see Chapter 9) in CP/M Plus has the following changes from previous versions of CP/M:

1. The [Gn] option (USER number) can be specified with the destination filespec in the command string to copy a file TO a user area (another [Gn] parameter may be placed with the source filespec to specify a different user area to copy from).
2. The [A] parameter may be used to turn on the ARCHIVE attribute for the file(s) automatically so that only those files that have been modified since the last backup are copied (the files copied are then stamped as archived).
3. The [C] parameter may be used to instruct PIP to prompt for confirmation before each copy operation.
4. (CP/M Plus Banked Version Only) If the source file has the READ protection (see SET), a password is required as part of the filespec – if a destination filespec is not specified, all SET protection attributes will be carried over, otherwise the attributes will be those specified in the filespec command line.
5. The DIR, SYS, R/O, and R/W attributes are always carried over with copy operation.
6. All logical and physical devices may be used with the PIP command as in previous versions of CP/M, with the exceptions described earlier in this chapter.
7. Logical devices can be mixed with files in a PIP command string, implicating first, second, . . . last execution.

PUT: The PUT command is similar to the GET command, except that it directs console or printer output to a filespec. This command has the same optional parameters as the GET command, plus the FILTER/NO FILTER options: FILTER is used to translate all control characters into true ASCII character pairs, including ESCAPE sequences. To revert back to normal,

enter PUT CONSOLE or PUT PRINTER, depending on what was initiated at the beginning.

RMAC: This command is a standard command in CP/M Plus, and is optionally available in previous versions of CP/M. It is identical to the MAC command, except it is used to create relocatable object code files instead of absolute object code files. The LINK command is used link and load relocatable files into executable transient command files. Refer to Appendix B for an overview of RMAC.

SET: This command provides the following:

1. Assign protection status to filespecs, associated to its password for read, write, delete, all three, or none (CP/M Plus Banked Version Only).
2. It also assigns labels to disks, passwords to labels (i.e., to disks), and passwords to filespecs (CP/M Plus Banked Version Only).
3. Assign a default password, which will be used when no password is entered associated to a filespec; depending on the protection status of the filespec, the user may only do things with the filespec if its password equals the default password. (CP/M Plus Banked Version Only.)
4. Turn on and off CREATE, ACCESS, and/or UPDATE time of day and date stamping for all files on disk.
5. Turn on and off the ARCHIVE attribute for files (turned on automatically by the PIP [A] parameter on a filespec basis, the ARCHIVE attribute is shown both by DIR and SHOW.
6. Assign file attributes DIR, SYS, RO, RW as with STAT, but enclosed in brackets instead of using "$".
7. Turn on and off user-definable attributes F1 through F4.

SETDEF: This command is used for defining disk drive chaining. Disk drive chaining provides for the automatic searching of transient command files and SUBMIT files through more than one disk drive. A file searched for in this manner will only be accessed if it has the SYS (system) attribute assigned to it. The SETDEF command is used for the following:

1. Entering SETDEF alone displays the current search order disk drive chain.
2. Assigning a drive for storing temporary (.$$$) files.
3. Assigning the search order for up to 4 drives.

4. Assigning the file type search (.COM, .SUB, or both).

SHOW: This command provides basically the same functions as STAT for displaying disk characteristics in previous versions of CP/M. It is used as follows:

1. SHOW x: displays Read/Write space left on disk "x:".
2. SHOW LABEL displays information about the disk's label: label name, whether passwords are required (CP/M Plus Banked Version only), stamp create and update on or off, and the time and date first created and when updated.
3. SHOW USERS displays the same information as STAT USR:, including the total number of files for each user.
4. SHOW DIR displays the number of directory entries left for the disk.
5. SHOW DRIVE displays specific disk drive characteristics for the logged disk drive.

SID: This is a standard command in CP/M Plus, and is optionally available in previous versions of CP/M. It provides symbolic instruction debugging for programmers. This command replaces the function of DDT (see Chapter 15) in previous versions of CP/M. Refer to Appendix A for an overview of this command and its options.

SUBMIT: This command contains some major changes from previous versions of CP/M. These are as follows:

1. A SUBMIT source file may contain "<" symbols denoting the start of lines containing input to programs being submitted. This replaces the XSUB command function found in CP/M Versions 2.0 and 2.2. CP/M Plus does not include the XSUB command function. A line with a single "<" denotes a carriage return (equivalent to pressing RETURN), which can be used to exit from many CP/M programs.
2. SUBMIT can simulate control characters by simply having the "^" character and the associate character forming the required pair: SUBMIT translates these character pairs to the correct ASCII control sequence. If the "^" is required by itself, "^^" must be entered. This is the same as the use of the "$" character.
3. CP/M Plus provides PROFILE.SUB, a SUBMIT file containing commands and parameters that will be executed automatically after each cold boot or system reset.

4. SUBMIT files can be nested several deep. Upon termination of a nested SUBMIT file, control is returned to next higher SUBMIT file, continuing in this way until returning to the first SUBMIT file submitted. SUBMIT files can be nested with GET commands in any combination.

XREF: This command is used to provide a crossreference summary of variable usage in a program produced by MAC or RMAC. A summary of its operation and options is provided in Appendix A.

APPENDIX A

Optional Transient Utilities

INTRODUCTION

This appendix deals with utility programs that are not normally a part of the CP/M operating system package but may be purchased separately. Both Digital Research and other companies are considered. Only those programs that have gained considerable popularity in the past few years are described here.

All the programs described in this appendix are Transient Command Programs, i.e., they are all designed to reside on a disk. In addition, all of them have the file type or extension ".COM" (meaning "command"). Many of the utilities described are meant to be used together with certain high-level programming languages. This will be noted where appropriate. The languages themselves are described in Appendix B.

For each program described in this appendix, only the general operating characteristics are presented. Information about how to access each of these, and how they are handled in the CP/M environment is provided. In addition, we will describe which files are required for operation, and which are created by the use of the program.

The optional utility programs described in this appendix are:

- **DESPOOL:** Background print spooling utility (Digital Research).

- **SID:** Symbolic Instruction Debugger for 8080 and 8085 microprocessor-based computers (Digital Research).

- **ZSID:** Symbolic Instruction Debugger for Z80 microprocessor-based computers (Digital Research).

- **LINK-80** (Digital Research): Linking Loader for relocatable object files created by the Digital Research PL/I-80 programming language, the Digital Research RMAC macroassembler, Microsoft MACRO-80 macroassembler, and other programs which produce relocatable object files.

- **LINK-80** (Microsoft): Linking Loader for relocatable object files created by the following Microsoft high-level programming languages: MACRO-80 macroassembler, BASIC-80, FORTRAN-80, and COBOL-80.

- **LIB-80** (Digital Research): Librarian utility for relocatable files created by Digital Research high-level programming languages and utilities.

- **LIB-80** (Microsoft): Run-time library manager for subroutines created by Microsoft MACRO-80, FORTRAN-80, and COBOL-80.

- **XREF** (Digital Research): Utility for crossreferencing print and symbols files created by a Digital Research macroassembler (MAC or RMAC).

- **CREF-80** (Microsoft): Utility for crossreferencing and listing intermodule references and locations created by the Microsoft MACRO-80 macroassembler.

- **XLT-86** (Digital Research): Utility for translating 8080 microprocessor assembly language program files to equivalent 8088/8086 microprocessor assembly language program files.

The remainder of this appendix covers brief descriptions of the optional transient utilities just mentioned.



DESPOOL BACKGROUND PRINT UTILITY

SUMMARY

The DESPOOL program, which may be obtained from Digital Research, is an optional utility for CP/M. With it, you can accomplish the printing (or "spooling") of files on the system listing device (printer). The printing of these files can occur while other CP/M commands, such as PIP, STAT, etc. are being executed. This is done by setting up a reserved background printing area in RAM which will not be used by any other program while DESPOOL is in operation.

In the normal CP/M environment, only one command can be executed at a time. Consequently, when files to be printed are sent to the listing device, you have to wait until the printing is finished before another command can be issued. This can be particularly inconvenient when the printing device is slow, because it ties up the computer for a long time while printing a lengthy file. The DESPOOL program solves this problem by placing the printing task in a "background" mode, with a lower processing priority than other commands that are typed on the console device. In other words, characters are printed when more important things are not required by the computer.

PROGRAM ACCESS

To access and execute DESPOOL, the file DESPOOL.COM must be stored on a disk in one of the computer disk drives. When the "DESPOOL" command is typed in response to the logged drive prompt, a message indicating that DESPOOL has been loaded into memory and is awaiting console input will be displayed. To print a file, the CTRL/F command (simultaneously pressing the CTRL and F keys) is entered, whereupon the program will ask the name of the file to be printed.

While DESPOOL is in memory, any other CP/M command can be entered and executed, regardless of whether DESPOOL is printing a file. If a file is being printed, the printing process will have a lower processing priority than any other command being executed at the time. If, while DESPOOL is active

and printing a file, you access a command that usually requires a lot of processing and disk read/write operations, you may notice that the printing will slow down from time to time. How much it actually slows down and when depends on console input and other processing activities. While a file is printing, however, console input itself is not significantly slowed.

The DESPOOL program will remain in memory once a file has been printed, and can be reactivated to print another file at any time. The entering of the CTRL/C (Warm Boot) sequence will not affect DESPOOL while it is in memory. To remove DESPOOL from memory, the CTRL/B sequence, which simulates the CTRL/C function, is entered. DESPOOL will also be removed from memory if a system reset occurs and the CP/M operating system is rebooted (Cold Boot). DESPOOL can be reloaded into memory at any time. If an attempt is made to load DESPOOL while it is already in memory, however, a message will be displayed indicating that DESPOOL is already active.



8080/8085 *S*YMBOLIC *I*NSTRUCTION *D*EBUGGER UTILITY

SUMMARY

The SID program is a symbolic instruction debugging utility for 8080 or 8085 microprocessor assembly language programs. SID is available from Digital Research. This program basically functions similarly to the CP/M-provided DDT (Dynamic Debugging Tool) transient program. The difference is that SID includes many enhanced features for program checking and debugging.

The SID utility provides the following features:

- All DDT command functions, with some modifications and enhancements.
- Provides comprehensive memory display and fill functions.
- Hexadecimal arithmetic command.
- Displays complete information regarding program execution.
- Provides a means of setting real-time breakpoints.
- Provides symbolic disassembly of portions of code containing assembly language mnemonics.
- Provides a means by which the test program and its symbols table can be read (displayed).

- Provides two separate utilities, which can be dynamically loaded with SID to provide traceback and histogram features.

The commands provided by SID while it is loaded in memory are the same as those provided by the DDT program. In addition, there are some commands which are unique to SID.

PROGRAM ACCESS

Basically, the SID program is accessed in the same manner as the DDT program (described in Chapter 15). To access SID, the file SID.COM must be stored on a disk in one of the computer disk drives. The command "SID" can be entered by itself in response to the logged drive prompt (A>, B>, etc.). By doing this, the program will be loaded into memory without specifying a file to be examined or debugged. A command can then be later entered to insert a file to be debugged into memory.

The "SID" command can also be entered with a file name. This method will load both programs into memory. The file specified with SID should be a machine language object code file. It should either be in machine code nonprintable form (".COM" or other files) or in hexadecimal text form (".HEX" files).

Two special utility files are provided with the SID program package. These consist of TRACE.UTL and HIST.UTL, which can both be loaded into memory with SID to provide special functions. TRACE.UTL provides the traceback function to SID, and HIST.UTL provides the histogram facility. Either or both of these files can be loaded into memory during or after SID's loading process.

More than one specified file can be loaded into memory with SID. For example, both a .HEX file and its equivalent symbols table file (extension .SYM) created by the Digital Research LINK-80 program can be loaded. SID will automatically deal with memory management and crossreferencing. To specify more than one file, first enter "SID", a space, the first file (COM, HEX, or other file), another space, and the second file (SYM or other file).

Once SID has been loaded into memory, the SID prompt will display and will consist of the "#" character. Whenever this prompt is displayed, it means that SID is waiting for console input. All CP/M control character commands (CTRL/S, CTRL/C, etc.) can be used, as usual. The CTRL/C command (Warm Boot) will terminate SID and return control to CP/M. Once the SID program has terminated, the program file that was modified can be saved on the disk using the built-in direct SAVE command (described in Chapter 16), in the same manner as with the DDT command.



Z-80 *S*YMBOLIC *I*NSTRUCTION *D*EBUGGER UTILITY

SUMMARY

The ZSID program, available from Digital Research, is a symbolic instruction debugging utility for Z80 microprocessor assembly language programs. This program provides the same function as the CP/M-provided DDT (Dynamic Debugging Tool) transient program and the optional 8080/8085 SID program previously described. The difference is that ZSID includes many enhanced features for program checking and debugging.

The ZSID utility provides the following features:

- All DDT and SID command functions, with some modifications and enhancements.
- Provides comprehensive memory display and fill functions.
- Hexadecimal arithmetic command.
- Displays complete information regarding program execution.
- Provides a means of setting real-time breakpoints.
- Provides symbolic disassembly of portions of code containing 8080, 8085, and Z80 microprocessor assembly language mnemonics.
- Provides a means by which the test program and its symbols table can be read (displayed).
- Provides two separate utilities, which can be loaded dynamically with ZSID to provide traceback and histogram features.

The commands provided by ZSID while it is loaded in memory are the same as those provided by the DDT and SID programs. In addition, there are some commands which are unique to ZSID.

PROGRAM ACCESS

Basically, the ZSID program is accessed in the same manner as the DDT program (described in Chapter 15), and the SID program just described. To

access ZSID, the file ZSID.COM must be stored on a disk in one of the computer disk drives. To load the program itself into memory without specifying a file for examination or debugging, the "ZSID" command can be entered in response to the logged drive prompt (A>, B>, etc.). When this is done, ZSID will be loaded into memory and a command can be entered later to insert a file into memory. Alternately, the "ZSID" command can be entered with a file name. In that case, both programs will be loaded into memory. The file specified with ZSID should be a machine language object code file. It should either be in machine code nonprintable form (".COM" or other files) or hexadecimal text form (".HEX" files).

Two special utility files are provided with the ZSID program package. These consist of TRACE.UTL and HIST.UTL, both of which can be loaded into memory with ZSID to provide special functions. TRACE.UTL provides the traceback function to ZSID, and HIST.UTL provides the histogram facility. Either, or both, of these files can be loaded into memory during or after the ZSID loading process.

More than one file can be specified to load into memory with ZSID. For example, both a .HEX file and its equivalent symbols table file will automatically deal with memory management and crossreferencing. This type of command line is entered by first typing "ZSID", a space, the first file (COM, HEX, or other file), another space, and the second file (SYM or other file).

Once ZSID has been loaded into memory, the ZSID prompt consisting of the "#" character will be displayed. Whenever this prompt is displayed, it means that ZSID is waiting for console input. All CP/M control character commands (CTRL/S, CTRL/C, etc.) can be used as usual. The CTRL/C command (Warm Boot) will terminate ZSID and return control to CP/M. Once the ZSID program has terminated, the program file that was modified can be saved on the disk using the built-in direct SAVE command (described in Chapter 16), in the same manner as with the DDT and SID commands.



CP/M ASSEMBLY LANGUAGE *LINK*ING/LOADER UTILITY

GENERAL

The LINK-80 is an optional utility program which may be added to the CP/M operating system. There are two variations of this program, both of which are called LINK-80. One is supplied by Digital Research and the other comes from Microsoft. Basically, both variations of the program provide the same function, although the general operation and the applications of each program often differ. The following paragraphs describe the operating characteristics of each variation of the program.

DIGITAL RESEARCH LINK-80 SUMMARY

Digital Research's LINK-80 utility is an optional program which may be added to the CP/M operating system. With it, relocatable object code files created by the PL/I-80 high-level programming language, the MAC macroassembler, or any other language translator producing Microsoft relocatable code can be loaded to create executable machine code command files. This program also provides a means by which several relocatable files can be linked together to create a single file to be loaded at a fixed address.

Basically, the LINK-80 program performs the same load function as the standard CP/M LOAD transient command program. The difference is that LINK-80 will produce two types of files. One type is the executable command file (.COM for CP/M or .PRL for MP/M), and the other is the symbols table file (.SYM), which can later be used with the SID or ZSID programs.

DIGITAL RESEARCH *LINK-80* PROGRAM ACCESS

To access LINK-80, the file LINK.COM must be stored on a disk in one of the computer's disk drives. The command is entered by first typing "LINK" in response to the logged drive prompt (A>, B>, etc.). This is followed either by a single file or a series of files. When only one relocatable file is to be loaded, the file name is entered after the command "LINK" on the same command line. If several relocatable files are to be linked together into one file to be loaded, all the relocatable files to be linked are entered. Files are separated from one another with a comma (",").

The LINK-80 program will produce an output file using the same name as the first relocatable file entered in the command string. The output file can have a different specified file name. If this is desired, enter the new file name after "LINK-80". Follow it by an equals sign ("="), and all the relocatable files to be linked.

During the linking and loading process, LINK-80 may create up to eight temporary files (indicated by the ".$$$" extension), which will reside on the disk until the linking operation is complete. You should, therefore, check the amount of free disk space to verify that there will be enough room for the complete operation.

When using LINK-80 to link and load PL/I-80 programs, the program will search the PL/I-80 run-time library file for any subroutines used by the PL/I-80 programs. This file is called PLILIB.IRL, and must always reside on the disk when LINK-80 is used with PL/I-80 programs.

There are several parameters or "switches" that may be specified in a LINK-80 command string. These switches instruct LINK-80 to operate in a particular manner. When they are used, they are enclosed between square brackets ("[" and "]"). This is similar to the way parameters are used with the PIP program, except that here multiple parameters are separated from each other with commas. Some parameters are entered next to the first specified file in a command string, whereas others are placed after the last file.

LINK-80 switches are defined as follows:

[A] This switch is used to provide LINK-80 with additional internal buffer space for symbol table storage. The switch is optional, and should only be used if more than the normal amount of memory space is required.

[D] This switch is called the Data Origin switch, and is used to specify the origin of data and common segments. If the switch is not specified,

LINK-80 will place the data and common segments immediately after the program segment. The switch is entered with a hexadecimal number, "Dnnnn", where "nnnn" is the specified data origin.

[G] This switch is called the Go switch. It is used to specify the label where the program is to begin execution. If the switch is not specified, execution will begin with the first byte in the program. The switch is entered with the label name, "Glabel", where "label" is the alphanumeric name of the label at the load address.

[L] This switch is called the Load Address switch. It is used to define the base address of the output .COM file generated by LINK-80. If this switch is not used, LINK-80 will specify 100 hexadecimal, which is the base of the Transient Program Area (TPA) in a standard CP/M system. The switch is entered with a hexadecimal number representing the desired load address, "Lnnnn", where "nnnn" is the number.

[M] This switch is called the Memory Size switch, and is used when creating output command files for the MP/M operating system (.PRL files). Its function is to indicate that additional data space is required for the .PRL file's correct execution. The switch is entered with a hexadecimal number representing the amount of space needed, "Mnnnn", where "nnnn" is the memory size. This switch is never used for .COM files in the CP/M operating system.

[NL] This switch is used to suppress the listing of the symbols table file (.SYM) to the console device while LINK-80 is executing.

[NR] This switch is used to disable the output of the symbols table file (.SYM) to the disk.

[OC] This switch is called the Output COM switch, and is used to direct LINK-80 to output a .COM file to the disk. This switch need never be used since its function is the default in LINK-80.

[OP] This switch is called the Output PRL switch. It is only used in MP/M to enable the output of a .PRL file instead of a .COM file.

[P] This switch is called the Program Origin switch, and is used to specify the origin of a program segment. If the switch is not used, LINK-80 will place the program segment at the load address, which is 100 hexadecimal (unless otherwise specified by the L switch). The switch is entered by specifying the hexadecimal number of the program origin, "Pnnnn", where "nnnn" is the number.

[Q] This switch is called the ? Symbol switch. It is used to cause the listing of this symbol when LINK-80 is referencing the PL/I-80 run-time library file. If this switch is not used, no occurrences of this symbol will be listed.

[S] This switch is called the Search switch. It is used to indicate that the file with this switch in the command line is to be treated as a library file. This will cause the specified file to be searched. In doing so, LINK-80 will include only those modules containing symbols which are to be referenced but are not defined in the modules already linked.

As was noted previously, the Digital Research LINK-80 can also be used to link relocatable files created with Microsoft high-level languages. The switches just mentioned, however, should be used with caution when linking Microsoft format files, since not all of them are appropriate.

Note also that when many switches are combined in a fixed manner it is convenient to put them in a submit file so only one brief command statement need be typed.

MICROSOFT LINK-80 SUMMARY

Microsoft's LINK-80 utility provided is an optional program which may be added to the CP/M operating system. With it, relocatable object code files created by the Microsoft MACRO-80 macroassembler, and various Microsoft high-level languages such as FORTRAN-80 and COBOL-80 can be loaded to create executable machine code command files. This program also provides a means by which several relocatable files can be linked together to create a single file that is to be loaded.

The LINK-80 program basically performs the same load function as the standard CP/M LOAD transient command program. The Microsoft LINK-80 program differs from the Digital Research LINK-80 in one fundamental way. Instead of typing all the link and load specifiers on the same command line, Microsoft LINK-80 can be executed by itself. In this way, it continues to provide a special prompt for user command input until an exit command is given. This is similar to the way the PIP program can be used. Another difference is that by specifying a special command, the Microsoft LINK-80 program can create symbol table files for later usage by SID or ZSID.

MICROSOFT *LINK-80* PROGRAM ACCESS

To access LINK-80, the file L80.COM must be stored on a disk in one of the computer disk drives. The command can be entered in one of two ways. The first is to enter "L80" in response to the logged drive prompt (A>, B>, etc.). This is followed by all the required linking and loading specifiers. The second way is the enter "L80" by itself followed by the RETURN key. With this method, the program will be loaded into memory, and a special LINK-80 prompt will display consisting of the "*" character. It is in response to this prompt that all the linking and loading specifiers as well as other commands are entered.

When the "L80" command is entered with all the linking and loading specifiers on the same command line, all files to be linked are entered in sequence, each separated with the forward slant ("/") character. The command "E" (Exit) must terminate the line *before* the RETURN key is pressed. If the "E" command is absent from the command line, LINK-80 will accept all the specified parameters, but the program will enter the line command mode, as indicated by the prompt consisting of the "*" character. LINK-80 will wait for another command. (By typing "E" at this point, you will exit LINK-80 and return to CP/M.) Once all the linking has been accomplished, LINK-80 will load the results in the output command .COM file. This file will have the same name as the first relocatable file specified in the command line.

When the "L80" command is entered by itself, the program will be loaded and the "*" prompt will be displayed. In response to this prompt, the file names to be linked and loaded can be entered, as well as other commands dictating the operation of LINK-80. As each file or command is entered on a line, LINK-80 will begin execution. The "*" prompt will be displayed again once it is ready to receive another command.

There are several parameters or "switches" that may be specified in a LINK-80 command string, or in response to the LINK-80 command prompt ("*"). These switches are also known as commands. They are used to instruct LINK-80 to operate in a particular manner. When they are used in a single command line with other files, they are separated from files to which they relate by commas. They are, however, placed with the file inside the "/" characters. When the switches are entered by themselves (as a group) in response to the command prompt "*", they are separated from each other by the "/" character. When single switches are entered in response to the "*" prompt, each one is preceded with the "/" character.

These switches or commands are defined as follows:

/G This switch is used to instruct LINK-80 to execute the .COM file after it is loaded, and then exit to CP/M when finished. The switch can also be used to specify the desired start address of the .COM file once it is loaded for execution. This is done by entering "/G:nnnn" where "nnnn" is the starting address in hexadecimal.

/E This switch is used to exit LINK-80 and return to CP/M once everything has been executed. The switch can also be used to specify the starting address of the .COM file generated for its execution. This is done by entering "E:nnnn" where "nnnn" is the starting address in hexadecimal. When this form of the command is used, you will not exit from LINK-80 until the program is finished executing.

/N This switch is used to specify the saving of all previously loaded programs and subroutines using the file name immediately preceding the switch. An alternate form of this switch is to enter "/N:P". This will save only the contents of the Transient Program Area (TPA).

/P /D /R These three switches are called the Address Setting switches. A /P is used to set the starting address for programs and data. A /D is used to set the starting address of the data area only. If /P is used with /D, it will only set the program starting address. A /R is used to reset LINK-80.

/S This switch is used to specify the file immediately following as a library file. LINK-80 will search through library files created by the Microsoft LIB-80 program for modules that can, but have not yet been used in the linking process.

/U /M These two switches are called the Global Listing switches. A /U is used to list undefined globals. A /M is used to list a complete global reference map.

/O /H These two switches are called the Radix Setting switches. A /O sets the radix to octal (base 8), and /H sets the radix to hexadecimal (base 16).

/X /Y These two switches are called the Special Code switches. The /X switch is used to specify that the loaded output file is to be loaded in hexadecimal format (.HEX) instead of executable machine language format (.COM). When /X is used, /N should also be specified. The /Y switch is used to instruct LINK-80 to also output a symbols table

file (.SYM) for later use with the SID or ZSID utilities. When this switch is used, the /N and /E switches should also be specified.

As was noted previously, the Digital Research LINK-80 can be used to link relocatable files created in Microsoft high-level languages. Care should be taken when using library files, since there may be many incompatibilities. Also, Microsoft LINK-80 cannot be used with PL/I-80 files if reference the PL/I-80 run-time library is required. This is the case, since this referencing capability is not provided.



CP/M *LIB*RARIAN UTILITY

GENERAL

The LIB-80 utility is a library management program. It permits the grouping together and storing of various assembly language and executable code subroutines that are often used with various programs in one neat "library" file. There two library utilities, both of which are called LIB-80. One is available from Digital Research, and the other from Microsoft. These two variations of the program basically perform the same functions, although there are some differences. Both LIB-80 variations have a fundamental characteristic in common. They both provide a means by which commonly used subroutines can be stored for later incorporation into program files when needed. This eliminates the need for the user to type these routines in each time.

The following paragraphs describe each variation of the LIB-80 program.

DIGITAL RESEARCH *LIB-80* SUMMARY

Digital Research's LIB-80 utility is an optional program which may be added to the CP/M operating system. This program performs the basic function of handling library files containing any number of relocatable object code modules. The LIB-80 program is used to concatenate (group together)

several relocatable files into a library file.* It is also used to create an indexed library, select modules, and print module names from a library.

DIGITAL RESEARCH *LIB-80* PROGRAM ACCESS

To access the LIB-80 program, the file LIB.COM must be stored on a disk in one of the computer drives. The command is entered by typing "LIB" in response to the logged drive prompt (A>, B>, etc.). This is followed by file names and parameters determining the function the program is to accomplish. The various ways LIB-80 can be used are as follows:

CREATE A LIBRARY FILE: After "LIB" is typed, enter the name of the output library file which LIB-80 is to create. This is followed (without a space) by the equals sign ("="), and the series of relocatable files that are to be grouped together and placed into the output library file. Each of these files is separated from one another with a comma. If no file type or extension is typed for either the output library file or the relocatable files, LIB-80 will assume the extension .REL. Other extensions can be typed for the output library file (for example, .LIB). The relocatable files can also have other extensions entered, as long as the files contain relocatable code.

CREATE AN INDEXED LIBRARY FILE: To create an indexed library file, the command string is nearly the same as mentioned in the paragraph above. The difference is that the command line is terminated with the "[I]" switch or parameter. If no extension is specified with the output indexed library file name, LIB-80 will automatically assign the extension .IRL.

LIST MODULE NAMES: To list all module names in an existing relocatable file (.REL) or a library file, enter the relocatable file name after "LIB". This is followed (without a space) by either the "[M]" switch or "[P]" switch. The [M] switch will list to the console all module names. The [P] switch will also list all module names, but will include all PUBLICS. A PUBLIC is a symbol which may be referenced in another module.

---

*The routines stored in a library must be in a special format that allows them to be reassembled to work anywhere in memory. This format is necessary because the programmer will not know it in advance. This format is called relocatable. It means the addresses in the library programs are "movable" and is actually selected when the final linking is done.

MICROSOFT *LIB-80* SUMMARY

Microsoft's LIB-80 utility is an optional program which may be added to the CP/M operating system. This program performs the basic function of handling library files containing any number of relocatable object code modules created by Microsoft programs MACRO-80 macroassembler, FORTRAN-80, and COBOL-80. The LIB-80 program is used to concatenate (group together) several relocatable files into a library file. LIB-80 can also be used to group together various assembly language programs that are subroutines to MACRO-80, FORTRAN-80 or COBOL-80. In addition, the program can be employed for the creation of specialized run-time library files containing specific execution requirements designed by the programmer.

Microsoft LIB-80 differs from Digital Research LIB-80, in that the command can be entered with all parameters and files on a single line. Alternately, LIB-80 can be loaded by itself so that a prompt is displayed, thus permitting you to enter all the necessary files and parameters on individual lines, until an exit command is given.

MICROSOFT *LIB-80* PROGRAM ACCESS

To access the LIB-80 program, the file LIB.COM must be stored on a disk in one of the computer drives. The command is entered by typing "LIB" in response to the logged drive prompt (A>, B>, etc.). This is followed by file names and parameters which determine the function the program is to accomplish. The "LIB" command may also be entered by itself, if the file names and parameters are to be entered on individual lines. When "LIB" is entered by itself, the program will be loaded into memory, and the "*" character will be displayed as a prompt. This prompt indicates that the LIB-80 program is awaiting for a command to be input.

The various functions LIB-80 provides are as follows:

CREATE A LIBRARY FILE: After "LIB" is typed, enter the output library file name which LIB-80 is to create. This is followed (without a space) by the equals sign ("="), and the series of files to be grouped together and placed into the output library file. Each file is separated from one another by a comma.

All source files to be grouped must be relocatable files. If no file type or extension is typed either output for the library file or the source relocatable

files, LIB-80 will assume the extension REL. Other extensions can be typed for the output library file (for example, .LIB). As long as the relocatable files contain relocatable code, they can also have other extensions entered. Module names pertaining to specific source files can also be included in this command line. They are entered *after* the corresponding source file, but *before* the comma, and must be enclosed with angled brackets ("<" and ">"). If more than one module name is specified for a file, the modules can be typed within the same angled brackets, and each module name must be separated by a comma.

There are several switches or parameters which may be included in a LIB-80 command line. These switches are known as commands, and can be typed when single-line command strings are entered. They can also be typed when entering parameters in response to the "*" prompt. All switches must be entered with a preceding forward slant ("/"), in the form of "/switch". They are usually placed next to the file to which the switch corresponds. These switches are defined as follows:

/E This switch is used to exit LIB-80 (while in command line mode), and return to CP/M. *Use this switch only if you are creating a new library file, or if you are revising one. Otherwise, use the CTRL/C command (Warm Boot) to exit.* This is important, because this switch will instruct LIB-80 to rename the library file under construction (.LIB) to .REL and delete any previous version of the file. If /E was used with an existing library file, and the file had not been updated, it will be deleted. (This could be disastrous if it was unintentional.) This will not occur, however, if an output file was specified in the command line as well as the input (existing file), since LIB-80 will then save the output file on the disk.

/R This switch is used to rename the library file under construction (.LIB) to .REL. The same cautionary note as mentioned previously applies to this switch. Use this switch only if you are creating a new library file. An /R performs the same functions as /E, except an exit will not occur from the program. Use this switch when you want to exit from the current library file, but want to continue using LIB-80.

/L This switch will list all the modules in the file specified, and all symbol definitions contained in the modules to the console device.

/U This switch will list to the console the symbols which it might find to be undefined during a single pass of the library file. If a symbol in a

library module refers back to a preceding module, it will be listed when this switch is used.

/C This switch is used to clear all commands entered from LIB-80 without exiting the LIB-80 program. The library file under construction is deleted, and LIB-80 session starts over. This switch is useful when an incorrect module has been specified, or a series of modules were entered in the wrong order. In either of these cases, the switch is used, as it might be desired to start over from the beginning.

/O This switch is used to set the radix mode to octal (base 8). The /L switch should also be used when this switch is specified.

/H This switch is used to set the radix mode to hexadecimal (base 16). The hexadecimal radix mode is the default, so this switch need not be used unless the /O switch was used previously.



CP/M ASSEMBLY LANGUAGE *CROSSREF*ERENCE UTILITY

SUMMARY

Digital Research's XREF utility is an optional program which may be added to the CP/M operating system. The XREF program is an assembly language crossreference utility used with the print and symbols table files created by the MAC or RMAC macroassemblers to provide a summary of variable usage throughout a given program.

PROGRAM ACCESS

To access the XREF program, the file XREF.COM must be stored on a disk in one of the computer drives. The command "XREF" is entered in response to the logged drive prompt (A>, B>, etc.). It is followed by a file name without an extension, and terminated by the RETURN key. The file

name specified in the command line refers to two files on the disk. They are the output print .PRN file, and the output symbols table .SYM file. Both have the same file name.

For XREF to operate correctly, both of these files must be stored on the disk, or an error will occur. Once the command has been successfully entered, XREF will begin reading each line in the print .PRN file. In so doing, it attaches a prefix line number to each line, and copies the prefixed lines to an output file with the same file name but with the extension .XRF. While the lines are being written to the output file, XREF scans each line for any symbols that occur in the symbols table .SYM file. Once this copy and comparison procedure is complete, XREF will append to the output crossreference XRF file a crossreference report. This reports lists all the line numbers where each symbol in the symbols table of the .SYM file occurs. Together with this report, XREF will flag with the "#" character, symbols which occurred as first tokens on each line. For each symbol reported, their values, as assigned by MAC or RMAC, are also presented.

When XREF command string is entered, a drive name can precede the file name. When this is done, both source files (the print .PRN and symbols table crossreference file (.XRF)) will be output to the same drive.

A special optional parameter can be added to the XREF command string to direct the output .XRF file information to the list printer device (assigned to the logical device name LST:). This is an option to storing the file on the disk. This is accomplished by typing the file name, a space and "$P".



CP/M ASSEMBLY LANGUAGE *CROSSREF*ERENCE UTILITY

SUMMARY

Microsoft's CREF-80 utility is an optional program which may be added to the CP/M operating system. The CREF-80 program is an assembly language crossreference utility used to process a specially assembled listing file created by the Microsoft MACRO-80 macroassembler. It does this so that a listing can be provided of the locations of all intermodule references and

locations of their definitions. The program produces an output crossreference listing file containing this information, which can later be used to aid in the debugging of a program.

PROGRAM ACCESS

To access the CREF-80 program, its command file must be stored on a disk in one of the computer's disk drives. The file may be called CREF80.COM, CREF.COM, or C80.COM. The latter is the most common.

The command "C80" is entered in response to the logged drive prompt (A>, B>, etc.). This is followed by a space, the equals ("=") sign, a file name without an extension. The command string is terminated by the RETURN key. The specified file name in the command line refers to a special file on the disk which was created by MACRO-80, and has the extension .CRF. Only this type of file may be used with CREF-80.

Once the command is executed, CREF-80 will create an output file that is similar in content to an output print .PRN file. The difference is that it will have the extension .LST. Once the command has been successfully entered, CREF-80 will begin reading each line in the special .CRF file. It attaches a prefix line number to each line, and copies the prefixed lines to the output file with the same file name but with the extension .LST. While the lines are being written to the output file, CREF-80 scans each line for any symbols that occurred in the statements. Once this copy and comparison procedure is complete, CREF-80 will append to the output crossreference .LST file a crossreference report that lists all the line numbers where each symbol occurs. Together with this report, CREF-80 will flag with the "#" character, symbols which occurred as first tokens on each line. For each symbol reported, their values, as assigned by MACRO-80, are also presented.

When CREF-80 command string is entered, a drive name can precede the file name. This will instruct CREF-80 to reference the .CRF file on the specified drive. If the output crossreference .LST file is to be output on a different disk, the desired drive name can precede the equals sign ("=") in front of the file name in the command line. If you want the output crossreference .LST file to be stored under a name other than the .CRF file, the desired file name can be placed (without a space) in front of the equals sign.

Instead of storing the file on the disk, CREF-80 will also accept the use of the LST: logical device and TTY: physical device names for the output of the .LST file. When one of these device names is used, it must precede the equals

sign before the .PRL file name (without a space). LST: is used to send the .LST file to the printing device, and TTY: is specified to list the contents of the file on the console device.



8080 TO 8086 ASSEMBLY LANGUAGE *TRANSLATOR* UTILITY

SUMMARY

Digital Research's XLT86 utility is an optional program that may be added to the CP/M operating system. This program serves the purpose of translating 8080 microprocessor assembly language programs to equivalent 8086 or 8088 microprocessor programs. During the translating process, XLT86 takes into account the present CP/M or MP/M operating system environment. As a result, the translated programs can operate either in the CP/M-86 or MP/M-86 operating systems. The XLT86 program uses global data flow analysis techniques to determine 8080 microprocessor register usage and thus reduces the number of 8086/8088 microprocessor assembly language instructions.

PROGRAM ACCESS

To access the XLT86 program, XLT86.COM (command file), and XLT00.0VL and XLT01.0VL (overlay files) must be stored on a disk in one of the computer drives. The program is accessed by typing "XLT86" in response to the logged drive prompt (A>, B>, etc.). This is followed by a space, and the file name. If an extension is not specified with the file name, XLT86 will assume the extension .ASM (CP/M ASM assembly language source-code file). The specified file must contain assembly language source code acceptable by the CP/M ASM assembler, or the MAC or RMAC macroassemblers.

Once the translation is complete, XLT86 will create and place the translated code in an output file by the same name as the source file. This output

file will, however, have the extension .A86 (CP/M-86 assembler default extension). In addition, XLT86 will create a print file with the extension .PRN. This print file will contain a list of any errors encountered during the translating process. The output .PRN file will be in text form, and will have imbedded tab and form-feed characters. As a result, it can be immediately output to the printing device.

During the translating process, XLT86 will process conditional assembly language statements and place the results in the output file. Macro definitions, macro invocations, and repeat loops, however, are not altered in the translating process. To properly translate macros, the source file must first be assembled using MAC or RMAC. The resulting output print PRN file must be modified with a text editor to delete all the beginning column positions. Then, by renaming the modified file to the extension .ASM, you will produce a file acceptable to XLT86.

There are several parameters or switches which may be added to the command line to instruct XLT86 to perform in a certain manner. These switches are entered following the file name, and must be enclosed within a pair of square brackets ("[" and "]"). More than one parameter may be entered on a command line. All switches must be side-by-side (without spaces) and enclosed within the same square brackets.

These switches are defined as follows:

[Ax] This switch instructs XLT86 to place the translated output .A86 file on a specified disk drive. "x" represents the disk drive name (entered without a colon).

[B] This switch is used to produce a list of basic blocks in the output .PRN file.

[C] This switch is used to instruct XLT86 to assume the 8086 "compact module" for execution.

[J] This switch is used to specify the translation of conditional jump instructions to short conditional instructions.

[L] This switch is used to output the .PRN file directly to the printing device.

[N] This switch is used to display on the console device each line and statement number as it is being translated.

[Px] This switch is used to specify the disk drive where the output .PRN file is to be stored. The "x" is the disk drive name (entered without the colon).

[R] This switch instructs XLT86 to assume that all flags are active upon returns from subroutines.

[S] This switch instructs XLT86 to assume nonoverlapping of the 8086 code and segments.

[Tx] This switch is used to specify the disk drive where the temporary file created by XLT86 is to be stored. In this switch, "x" is the the disk drive name (entered without a colon).

[80] This switch is used to specify the creation of an 8080 assembly language listing in the output .PRN file.

[86] This switch is used to specify the creation of an 8086 assembly line and statement listing in the output .PRN file.

APPENDIX B

CP/M-Compatible Languages

INTRODUCTION

This appendix deals with high-level language programs that are normally not a part of the CP/M operating system package but may be added. Only the programs that have gained considerable popularity in the past few years are described in this appendix. Many of the utilities described in Appendix A are used together with certain high-level programming languages described in this appendix. This will be noted where appropriate.

For each program described in this appendix, only the general operating characteristics are presented. Provided information tells how to access each of these programs and how they are handled in the CP/M environment. We will also disclose which files are required for operation, and which files are created with the use of the program. In other words, the descriptions provide information about how the various programs interface with CP/M.

The optional high-level language programs described in this appendix are:

- **MAC:** Macroassembler provided by Digital Research.
- **RMAC:** Relocating macroassembler provided by Digital Research.
- **MACRO-80:** Macroassembler provided by Microsoft.
- **PL/I-80:** Subset software package of the Programming Language/One provided by Digital Research.
- **BASIC-80:** BASIC language interpreter provided by Microsoft.
- **BASCOM:** Compiler for BASIC-80 provided by Microsoft.

- **CBASIC2:** Compiler BASIC provided by Digital Research.
- **CB-80:** Native code compiler language compatible with CBASIC2 provided by Digital Research.
- **PASCAL/MT+:** Pascal software package provided by Digital Research.
- **FORTRAN-80:** FORTRAN software package provided by Microsoft.
- **COBOL-80:** COBOL software package provided by Microsoft.

The remainder of this appendix covers brief descriptions of the optional transient programs just mentioned.



*MAC MA*CROASSEMBLER

SUMMARY

Digital Research's MAC macroassembler is an optional program which may be added to the CP/M operating system. The MAC program is used to assemble 8080 microprocessor assembly language program files containing macros. Macros are sections of code that are referenced and execute when the user has a certain keyword appear in the source listing. During the assembly process, MAC will create an output hexadecimal format file. This file can be loaded later with the CP/M LOAD command, a print file similar to that created by the CP/M ASM program, and a symbols table file. Reference should be made to the MAC manual for information about how macros are used to create assembly language source files. MAC cannot produce relocatable files. RMAC does that.

PROGRAM ACCESS

To access MAC, the file MAC.COM must be stored on a disk in one of the computer drives. The command "MAC" is entered in response to the logged disk prompt (A>, B>, etc.), followed by the name of the file to be assembled. The specified file must have the extension .ASM, but the extension is not typed in the command line.

While MAC is performing the assembly process, three output files will be created. The first is the object code file in hexadecimal format (.HEX). The second is the output print file (.PRN). The third is the symbols table file (.SYM). (The last is not produced by ASM.) All three of these files will have the same name as the source-code .ASM file. The format of the hexadecimal file (.HEX) will be standard Intel hexadecimal. Using the CP/M LOAD command (described in Chapter 14), the file later will be loaded into an executable command file. The format of the print (.PRN) file will be similar to the one created when the ASM program is used. The difference is that it will contain all of the macro assembler routines expanded out. In the source listing, only the symbolic macro reference is given (i.e., something like EOUT appears in the source). The symbols table (.SYM) file will contain all of the symbols used in the assembly routines, and can later be used for debugging purposes with the optional SID or ZSID utilities (described in Appendix A).

Special parameters added after the specified file can be included in the MAC command string. Several parameters can be included, and they are separated from one another by a space. These parameters are preceded by a dollar sign ("$"), and are defined as follows:

Ax This parameter is used to specify the name of the drive which contains the source .ASM file. The "x" is the drive name (entered without the colon).

Hx This parameter is used to specify the name of the drive on which the output .HEX file is to be stored. The "x" is the drive name. Alternately, a choice of three symbols can be used instead of a drive name. An "X" means list the file to the console device, "P" means send the file to the printer device, and "Z" means suppress output of the file. These three options can be used with any parameter normally used to specify a disk drive name. Any parameter, that is, except the "A" parameter described previously.

+Lx or −Lx

This parameter determines the disk drive name from which the library (.LIB) files are to be referenced. "x" is the drive name or special symbol. The "+L" parameter causes the input lines read from the library file to be listed on the console device. The "−L" parameter is the default setting and is used to suppress this listing.

+M or −M

This parameter is used to control the listing of macros in the .PRN file. When +M is entered, all macro lines are listed as they are processed. When −M is used, the listing of macros is suppressed.

Px This parameter is used to specify the disk drive on which the print .PRN file is to be printed. The "x" is the drive name or special symbol.

+Q or −Q

This parameter is used for the listing of LOCAL symbols in the symbols table .SYM file. When +Q is entered, all LOCAL symbols are listed. When −Q is used, the listing of these symbols is suppressed.

+S or −S

This parameter is used to specify whether the symbols table .SYM file is to be appended to the end of the list .PRN file. When +S is entered, the .SYM file is appended to the .PRN file. When −S is entered, no appending takes place.

+1 or −1

This parameter is used during the assembly process to specify the output of the .PRN listing file after the first pass of the source file. When +1 is entered, the listing will be created on the first pass, which can later be used for macro debugging. When −1 is used, the listing is suppressed on the first pass (this is the default setting).



RMAC RELOCATABLE *MAC*ROASSEMBLER

SUMMARY

Digital Research's RMAC utility is an optional relocatable macroassembler program. RMAC is almost identical to MAC. The difference is that RMAC is used to create relocatable object code files instead of loadable hexadecimal files. A relocatable file can then be linked together with other relocatable files using the optional LINK-80 program (described in Appendix A).

The results are placed in a final loadable file, which then is loaded into executable machine code format.

PROGRAM ACCESS

The access and operation of RMAC is identical to MAC (described previously), except the name of a parameter function has been changed. The "H" parameter defined for MAC, is called the "R" parameter when using RMAC. This switch is used to control the output of the final relocatable file, which will have the extension .REL instead of .HEX. The relocatable .REL output file can then be used with the LINK-80 program to link it with other .REL files. The results can be loaded into a final executable command file.

The following optional parameters or switches are defined for use with MAC. Any or all parameters may be specified on the command line, and first parameter presented must be preceded by "$" (dollar sign).

$A Specifies the disk on which the .ASM file is stored. Enter $AA for disk A, $AB for disk B, etc.

$H Specifies the disk on which the output .HEX file is to be stored. Enter $HA for disk A, $HB for disk B, etc.

$L Specifies the disk on which the .LIB files are stored. Enter $LA for disk A, $LB for for disk B, etc.

$+L and $−L
Controls the listing of input lines read from the macro library. $+L enables the listing, $−L disables it.

$+M and $−M
Controls the listing of all macro lines as they are processed during assembly. $+M enables listing, $−M disables it.

$*M Specifies that only hexadecimal code be generated by macro expansions.

$P Specifies the disk on which the output .PRN listing file is to be stored. Enter $PA for disk A, $PB for disk B, etc.

$+Q and $−Q
Controls the listing of all LOCAL symbols in the symbols table. $+Q enables listing, and $−Q disables it.

$+S and $−S

Controls the appending of the .SYM (output symbols table file) to the end of the output .PRN file. $+S enables appending, and $−S disables it.

$+1 and $−1

Controls the output of a listing file on the first pass during assembly. This is used for macro debugging purposes. $+1 enables the listing, and $−1 disables it (default).



*MACRO-80 MACRO*ASSEMBLER

SUMMARY

Microsoft's MACRO-80 macroassembler program is an optional program to the CP/M operating system. The purpose and operation of this program is almost identical the to the MAC and RMAC programs. The MACRO-80 program is used to assemble assembly language programs containing macros. This is done to produce intermediate relocatable object-code files. These relocatable object-code files can be linked together with other relocatable files with the optional Microsoft LINK-80 program (described in Appendix A). As a result, they can be used to produce a final file loaded to create an executable machine language object-code file.

The MACRO-80 program also creates a special file that can later be used with the optional Microsoft CREF-80 program (described in Appendix A). This is used to produce a crossreference file containing symbols table information, and its purpose is similar in nature to the symbols table (.SYM) file created by MAC and RMAC.

PROGRAM ACCESS

To access the MACRO-80 program, the file M80.COM must be stored on a disk in one of the computer disk drives. To load the program, "M80" is

entered in response to the logged disk prompt (A>, B>, etc.). "M80" may be followed by a string of characters representing the source file to be assembled and special parameters. On the other hand, pressing the RETURN key offers the option to enter the files and parameters on individual lines while MACRO-80 is loaded in the memory. When latter form of the command line is entered, MACRO-80 will be loaded into memory, and the "*" character, which represents the MACRO-80 command prompt, will be displayed. Files and special parameters can be entered, one line at a time, in response to this prompt.

When file names and parameters are entered on the same line as the "M80" command, the source file to be assembled is entered preceded with an equals sign ("="). If the names of the output relocatable object file and the print file are specified, they must precede the equals sign, with the relocatable file entered first, followed by a comma and the name of the print file. Alternately, this command string, and other variations of it, can be entered in response to the "*" prompt.

There are a number of special parameters or switches that can be specified in the command string. If used, they must be entered at the end of the command string. Each switch is preceded with the forward slant ("/") character. These switches are defined as follows:

/O This switch is used to specify that all address listings in the print .LST file are to be octal (base 8) numbers.

/H This switch is used to specify that all address listings in the print .LST file are to be hexadecimal (base 16) numbers.

/R This switch forces the generation of the relocatable object-code file with the same name as the source file. This switch can be used instead of specifying the object-code file name in the command line.

/L This switch forces the generation of the print .LST file with the same name as the source file. This switch can be used instead of specifying the print file name in the command line.

/C This switch instructs MACRO-80 to generate the special crossreference file (for later usage with CREF-80). This file will have the extension .CRF.

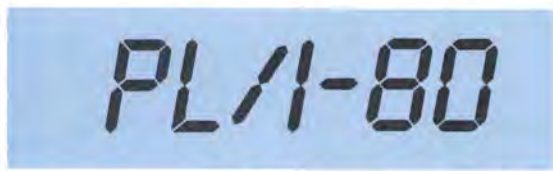
/Z This switch specifies that the source code file contains Z80 microprocessor assembly language instructions. In this way, MACRO-80 can assemble them properly. This switch is only necessary if the ".Z80" pseudo-op code is not specified in the source file.

/I This switch specifies that the source code file contains 8080 microprocessor assembly language instructions. In this way, MACRO-80 can assemble them properly. This switch is only necessary if the ".8080" pseudo-op code is not specified in the source file.

/P This switch allocates an extra 256 bytes of stack space for use during the assembly process. This switch is normally not needed, but should be specified if a stack overflow condition occurred during the first assembly process attempt.

/M This switch initializes Block data areas. This switch need only be used when the source file contains the DS (Define Space) pseudo-op code initialized to zeros.

/X This switch enables the listing of false conditionals. If the switch is not used, all false conditionals will not be listed.



PL/I-80 8080 PROGRAMMING LANGUAGE/*1*

SUMMARY

Digital Research's PL/I-80 (8080 Programming Language/One) is an optional program which may be added to CP/M. This programming language is based on the ANSI General-Purpose Subset (Subset G) of PL/I, as specified by the ANS PL/I Standardization Committee X3J1. PL/I-80 is almost identical with Subset G, with some differences as indicated in the manual. This language is also similar to the Pascal programming language and the Intel PL/M-80 language.

PL/I-80 is often referred to as a "system development" language, since it is considered by many system programmers as a useful tool for developing operating systems, high-level languages, and system utilities. You can use the PL/I-80 program to assemble or compile files containing PL/I-80-compatible source-code programs. This is done to create relocatable object code files. These relocatable object-codes files can then be linked with other relocatable library files using the optional Digital Research LINK-80 utility (described in Appendix A). The result is the creation of a final file that can then be loaded into executable machine language format.

PROGRAM ACCESS

To access the PL/I-80 program, the files PLI.COM (main command file) and PLILIB.IRL (special PL/I-80 run-time librarian file) must be stored on a disk in one of the computer drives. The command "PLI" is entered in response to the logged drive prompt (A>, B>, etc.). This is followed by the name of the file containing the source code to be compiled. Unless an extension is typed with the source-code file name, PL/I-80 will assume that the file has the extension .PLI.

While the program is being compiled, PL/I-80 will automatically retrieve necessary information from the library file PLILIB.IRL. The resulting compiled file is a relocatable object-code file, with the extension .REL. This file can then be used with the LINK-80 program (described in Appendix A) to link with other relocatable files, and load the result into an executable command file.



BASIC-80 8080 *BASIC* LANGUAGE INTERPRETER

SUMMARY

Microsoft's BASIC-80 program is a BASIC language interpreter which may be optionally added to the CP/M operating system. Version 5 of this program is designed to meet the requirements of the ANSI subset standard for the BASIC language. BASIC-80 also provides many other unique features and can be used to create BASIC source-code programs and interpret existing ones so that they may be executed.

PROGRAM ACCESS

To access the BASIC-80 program, the file MBASIC.COM must be stored on a disk in one of the computer drives. Although MBASIC.COM is the most common name for this file, it can be stored under a different name.

The command can be entered in one of two ways. The first way is to enter "MBASIC" in response to the logged disk prompt (A>, B>, etc.), followed by the RETURN key. This will cause MBASIC to be loaded into the memory, whereupon the prompt "Ok" will be displayed. This prompt indicates that BASIC-80 is ready to receive commands. These commands can be used to create a new BASIC file, modify an existing one, or interpret (run) a file. Alternately, the command "MBASIC" can be entered followed by the BASIC source-code program file name that you want interpreted and executed. If no extension is entered with the file name, BASIC-80 will assume the extension .BAS.

There are two types of BASIC program source-code files that can be used with BASIC-80. The first is the .BAS file, which is normally the file used for run-time interpretation. Usually, this file does not have the same format as it contains many control sequences which make it difficult to view using the CP/M TYPE command. The other type of file is the source-code program in displayable ASCII format. This type of file usually has the extension .ASC. It is the .ASC file that is most often used when source program modifications are made with a text editor instead of using BASIC-80.

When the "MBASIC" command is entered with a file name for interpretation, several optional parameters may follow on the command line. These parameters determine certain operating characteristics of BASIC-80. They are entered with a preceding forward slant character ("/"), and are usually followed by a specific numerical value.

The BASIC-80 parameters are defined as follows:

/F:nnnn This parameter sets the number of disk data files that may be open at any one time during the execution of a BASIC-80 program. The "nnnn" is the number of files in decimal. If this option is not entered, the default value is set to 3.

/M:nnnn This parameter is used to set the highest memory location that will be used by BASIC-80; "nnnn" is the amount of memory. This option allows setting memory above MBASIC that can store user programs written in 8080 assembly code.

/S:nnnn This parameter specifies the maximum record size for use with random files. The "nnnn" is the number of bytes in decimal. If this parameter is not used, the default setting is 128 bytes.



BASCOM, MICROSOFT BASIC-80 COMPILER

SUMMARY

The Microsoft BASIC-80 Compiler (often referred to as "BASCOM") is an optional software package that may be added to the CP/M operating system and the Microsoft BASIC-80 interpreter program. The BASCOM software package provides a means by which BASIC language program source-code files created by or for use by the BASIC-80 program, can be compiled into intermediate relocatable object-code files. These relocatable files can be used by the Microsoft LINK-80 program (described in Appendix A) to link with other relocatable files for eventual loading into an executable .COM command file. Several files are also included with the package that provide reference sources for subroutines found in various versions of the compiler program and BASIC-80.

In summary, the BASCOM package provides a means by which BASIC-80 programs can be linked and loaded into executable command files. When a compiled, linked and loaded program is executed, run-time speed is significantly increased over the interpretation process of the BASIC-80 program.

PROGRAM ACCESS

The BASCOM software package consists of the BASCOM.COM file, which is the compiler command, and several library files containing reference subroutines. To make full use of the relocatable files created by the compiler, the package also includes the LINK-80, MACRO-80, CREF-80, and LIB-80 programs.

To compile a BASIC-80 source-code program file, the "BASCOM" command is entered in response to the logged disk prompt (A>, B>, etc.). This is followed by the name of the file to be compiled. If the extension of the file is not entered, BASCOM will assume the .BAS extension. Once the compilation process is complete, BASCOM will create the output file containing the relocatable compiled code. This file will have the extension .REL. BASCOM

will also create a listing file with the extension .PRN, which will contain the BASIC language code plus the compiled parameters, where appropriate.

To specify a .REL file name different from the source file name, the .REL file name can be specified first in the command line. Both file names are separated from each other by an equals sign ("="). To specify a different file name for the .PRN file, both the .REL and the .PRN file names can be placed before the source file. The first two files are separated by a comma. Finally, to specify only the .PRN file, the file name can be entered preceded by a comma, and followed by the equals sign and the source file name.

Once the output files have been created, other Microsoft utilities, such as LINK-80, can be used to create the final files.

The following parameters or switches are defined for use with BASCOM.

/C Specify that constraints on line numbering be relaxed. (Note that this switch may not be used together with the /4 switch.)

/D Instruct the compiler to generate debug code for run-time error checking.

/E Inform the compiler that the program to be processed has an ON ERROR GOTO statement(s) with RESUME <line number>.

/N Suppress listing of disassembled object code in the listing file.

/O Subtitute the OBSLIB.REL run-time library for BASLIB.REL as the default run-time library searched by the Microsoft LINK-80 after the linker switches /E or /G are used.

/S Instruct the compiler to write quoted strings to the .REL file on the disk and not to the data area in RAM.

/T Instruct the compiler to use execution conventions supported by Microsoft's BASIC-80 Version 4.51. (Note that this switch may not be used together with the /C switch.)

/X Inform the compiler that the program to be processed has an ON ERROR GOTO statement(s) with either RESUME, RESUME 0, or RESUME NEXT.

/Z Instruct the compiler to use Z80 microprocessor op codes where possible.

/4 Instruct the compiler to use lexical conventions supported by Microsoft's BASIC-80 Version 4.51. (Note that this switch may not be used together with the /C switch.)



CBASIC COMPILER BASIC LANGUAGE

SUMMARY

The CBASIC software package which may be added to the CP/M operating system consists of a BASIC language compiler and run-time program. This BASIC compiler program provides the means by which CBASIC language source program files can be compiled into intermediate run-time files. These run-time program files can then be executed by another program which is also provided in the package.

There are two major versions of this program. Version 1 is known as CBASIC, and Version 2 is known as CBASIC2. Both versions are essentially the same, although some differences exist in the way the statements and commands in the source-code programs are used.

PROGRAM ACCESS

To use either the CBASIC or CBASIC2 package, the following two files must be stored on a disk in one of the computer drives: CBAS.COM (CBASIC) or CBAS2.COM (CBASIC2), which is the compiler program, and CRUN.COM (CBASIC) or CRUN2.COM (CBASIC2), which is the run-time command to run the compiled programs.

To compile a program, enter "CBAS" or "CBAS2" in response to the logged disk prompt (A>, B>, etc.). This is followed by the name of the file without the extension, since the program will assume the extension .BAS. After the compiling process is complete, the program will list the number of errors (if any) which it encountered in the source file. It will also create an output intermediate file (with the extension .INT) which contains the compiled code. To execute an intermediate .INT file, enter the "CRUN" or "CRUN2" command, followed by the .INT file name, without the extension (this command will assume the extension .INT).

Several optional parameters may be entered on the command line when using the compiler program. These parameters (known as "toggles") specify the operating characteristics of the program. Toggles are entered after the

specified file name, preceded by a space and the "$" character. Any number of toggles may be entered after the "$" character.

These toggles are defined as follows:

$B This toggle suppresses the listing to the console device of the source program as it is being compiled. If the program detects any errors during compilation, they will be listed even if this toggle is used.

$C This toggle suppresses the generation of the output intermediate .INT file. It is used when compiling a program for debugging purposes only.

$D This toggle suppresses the translation of lower-case characters in the source file to upper case when listed on the console device. If this toggle is used, all keywords in the source file must be in upper case.

$E This toggle causes all errors encountered during compilation to be placed in the output intermediate .INT file. As a result, when the program is run, these error messages will display where appropriate. This toggle is useful for debugging purposes.

$F This toggle causes the compiled output listing to be printed on the printer device, as well as the console device.

$G This toggle causes the compiled output listing to be sent to a file on a disk with the extension .LST.



8080 NATIVE CODE COMPILER FOR *CB*ASIC

SUMMARY

Digital Research's CB-80 software package consists of various programs. They are used to implement BASIC source-code programs written for the CBASIC and CBASIC2 compiler packages into relocatable object-code programs that can be loaded as executable machine language programs. The CB-80 package consists of a compiler program to produce the relocatable programs, and a library and linker editor program called LK-80.

PROGRAM ACCESS

To use the CB-80 package, the following files must be stored on a disk in one of the computer drives: CB80.COM, which is the compiler program, and LK80.COM, which is the librarian/linker editor program.

To compile a CBASIC or CBASIC2 compatible source program file, enter the command "CB80" in response to the logged drive prompt (A>, B>, etc.). This is followed by the name of the file to be compiled. If an extension is not specified, the extension .BAS will be assumed. Once the compilation process is complete, CB-80 will create an output relocatable object-code file with the extension .REL. This file can later be linked and loaded into executable machine code format with the LK-80 program.

There are several parameters or switches which may be included in the compiler program command line to specify certain operating characteristics. Switches are entered following the source-code file name in the command line. When a switch is used, it must be enclosed in a pair of square brackets ("[" and "]"). Any number of switches may be specified on the command line, all of which can be entered in sequence and enclosed within a single pair of square brackets.

These switches are defined as follows:

[B] Suppresses the listing of the source file contents to the console device.

[C(x)]
Changes the default disk to a specified disk from which the Include file will be retrieved. The Include file is controlled by the "%INCLUDE" directive in the source file. The toggler is entered as shown, where "(x)" is the disk drive ("(A)" for Drive A, "(B)" for Drive B, etc.).

[D] Enables the listing of errors caused by undeclared variables in the source program.

[I] This switch is used to interlist the generated code with the source code to the console device.

[L(n)] This switch is used to change the page length when the [P] toggler is used to send the listing information to the printer device. "n" is a decimal number determining the page length.

[N] Causes code to be generated for each physical line in the source program.

[O] Suppresses the generation of the output .REL file.

[P] Specifies the listing of the source file to the printer device. The printed output will be paginated, and each page will have a heading with the page number and source file name.

[S] Causes the program symbols information to be included in the output .REL file.

[T] Causes the listing of the symbols table following the source file listing.

[W(n)]

Sets the page width when using the [P] switch to output the source file listing to the printer device. The page width is "n", where a decimal number is entered ("(65)" for 65 columns, "(75)" for 75 columns, etc.).

[X(x)]

This switch is used to change the default disk drive to a selected drive. The selected drive is where the CB-80 temporary work files are to reside during the compilation process. "x" is the drive name ("(A)" for Drive A, "(B)" for Drive B, etc.).



PASCAL/MT+ PROGRAMMING LANGUAGE

SUMMARY

Digital Research's Pascal/MT+ programming language package may be added to the CP/M operating system. Pascal/MT+ complies with the entire International Standards Organization (ISO) standard, DPS/7185, and includes many other unique features. The package provides a compiler, linker, disassembler, debugger, and a librarian utility. With these programs, Pascal source files compatible with Pascal/MT+ can be compiled into relocatable object-code files, for later linking and loading into executable machine language command files.

PROGRAM ACCESS

To use Pascal/MT+ the following command files must be stored on a disk in one of the system disk drives: MTPLUS.COM (compiler command), LINKMT.COM (linker command), DIS8080.COM (disassembler command), and LIBMT.COM (librarian command). There are also many other files required for these programs to work. These consist of reference command and relocatable files. (Refer to the Pascal/MT+ manual to determine which files are required and which are optional.)

To use the compiler program, enter "MTPLUS" in response to the logged drive prompt (A>, B>, etc.). This is followed by the name of the source file to be compiled. If an extension is not entered, the program will assume the extension .SRC. The program will display compilation information, and list any errors it encounters in the source program. Once the compiling process is complete, the program will produce an output relocatable file with the extension .ERL. This relocatable .ERL file can then be linked and loaded using the "LINKMT" command.

On the "MTPLUS" command line, optional parameters or switches which determine the operating characteristics of the program may be entered following the source file name. Enter a switch by preceding it with the "$" or the "#" character. Any number of switches may be entered on the same command line. All must be preceded by a single "$" or "#" character.

These switches are defined as follows:

$Rd or #Rd

This switch is used to specify the disk drive on which the output .ERL file is to be stored. The "d" is the disk drive name (entered without the colon).

$Od or #Od

This switch is used to specify the disk drive on which the Root and Overlay files are to be referenced. The "d" is the disk drive name (entered without the colon).

$Pd or #Pd

This switch is used to specify the disk drive on which the output .PRN file is to be stored. The "d" is the disk drive name (entered without the colon).

$X or #X

This switch is used to generate an extended .REL file with disassembler records.

$D or #D

This switch specifies the generation of debugger information in the object code. It also writes the output .PSY file to the drive specified by the P switch.

$Ed or #Ed

This switch informs the program that the errors file MTERRS.TXT is stored on a specific disk drive. The "d" is the drive name (entered without the colon).

$Td or #Td

This switch specifies the disk drive on which the file PASTEMP.TOK is to be stored. The "d" is the disk drive name (entered without the colon).

$Q or #Q

This switch is called the Quiet switch and is used to suppress the listing to the console device of any unnecessary messages.

$C or #C

This switch is used to cause the program to continue past any encountered errors.

$A or #A

This switch will cause the program automatically to access the linker program after compilation is complete.

$B or #B

This switch specifies the use of BCD (Binary Coded Decimal) instead of floating point on real numbers.

$Z or #Z

This switch specifies the generation of Z80 microprocessor optimized code. When this switch is not used, the program will generate only 8080 microprocessor code.

$V or #V

This switch is used to list the name of each procedure and function when found in the source-code file. It is used as an aid to determining error locations during the first phase.

$@ or #@

This switch specifies that the "@" character is to be made equivalent to the "^" character.



FORTRAN-80 8080 *FORTRAN* LANGUAGE

SUMMARY

Microsoft's FORTRAN-80 program is an optional program which may be added to the CP/M operating system. The FORTRAN-80 program is a compiler for FORTRAN source program files, which creates relocatable object-code files for later linking and loading into executable command machine code files. The FORTRAN-80 program includes the American National Standard FORTRAN as described in ANSI document X3.9-1966. It also includes a number of extensions and some restrictions.

PROGRAM ACCESS

To use the FORTRAN-80 program, the file F80.COM must be stored on a disk in one of the computer drives. To access the program, enter "F80" in response to the logged drive prompt (A>, B>, etc.). This is followed by the name of the source file. If an extension is not entered for the source file, the extension FOR is assumed. Alternately, the command "F80" can be entered by itself. Doing this will load the program in memory and the "*" character will be displayed. This prompt indicates that FORTRAN-80 is awaiting a command.

Once compilation is complete, the program will generate a relocatable object-code file with the extension .REL. This relocatable file can then be linked with other relocatable files using the LINK-80 program (described in Appendix A). The final results can then be loaded in an executable command machine language file. The FORTRAN-80 program will also generate an output listing file with the extension .PRN. The command line syntax when

specifying various files is the same as that used by the Microsoft MACRO-80 program.

Several optional parameters or switches which determine the operational characteristics of the program may be entered on the command line. These switches are entered after the source file name. Each is preceded by a forward slant ("/") character.

These FORTRAN-80 switches are defined as follows:

/O Specifies the printing of all listing addresses in octal (base 8).

/H Specifies the printing of all listing addresses in hexadecimal (base 16).

/N Disables the listing of generated code to the console device.

/R Forces the generation of the .REL relocatable object-code file.

/L Forces the generation of the .PRN listing file.

/P Specifies the allocation of an extra 100 bytes of stack space for use during compilation. This switch can be entered a number of times for multiples of 100 bytes.

/M Specifies that the generated code is to be in a format for loading into ROM devices.



COBOL-80 8080 *COBOL* LANGUAGE

SUMMARY

The COBOL-80 package is available from Microsoft, and can be added to the CP/M operating system. The package consists of a compiler which generates relocatable object-code files, and a run-time program used to run compiled COBOL-80 programs.

PROGRAM ACCESS

To use the COBOL-80 compiler and run-time programs, the following files must be stored on a disk in one of the computer drives: COBOL.COM (compiler command file), COBOL1.OVR through COBOL4.OVR (overlay files for COBOL.COM), RUNCOB.COM (run-time executor command), COBLBX.REL (library for RUNCOB.COM), CRTDRV.REL (basic video terminal driver), and DEBUG.REL (interactive COBOL-80 debugger).

To compile a COBOL-80 source program file, enter "COBOL" in response to the logged drive prompt (A>, B>, etc.). This is followed by the name of the source file. If an extension for the source file is not entered, .COB is assumed. Once compilation is complete, COBOL-80 will generate a relocatable object-code file with the extension .REL, and a listing file with the extension .PRN. The command line syntax when specifying several files is the same as that for the Microsoft MACRO-80 and FORTRAN-80 programs.

To specify certain operating characteristics of the program, special optional parameters or switches may be included in the command line. These switches are entered after the source file name. Each one is preceded with the forward slant ("/") character.

These switches are defined as follows:

/R Force the generation of the relocatable object code file.

/L Force the generation of the listing file.

/P Allocate an extra 100 bytes of stack space for use by the compiler. This switch may be repeated for multiples of 100 bytes.

/D Suppress the generation of a debug file, object code information about source file line numbers.

/X Prepare the object code file so that it can only be linked with the run-time system contained in the library.

/Fn Generate FIPS flagging messages in the listing file, where the value of "n" can be the following:

0 List all FIPS flagging features and Microsoft extensions.
1 Exclude low-intermediate features.
2 Exclude both low- and high-intermediate features.
3 Exclude all except extensions.
4 List no FIPS flagging features.

APPENDIX C

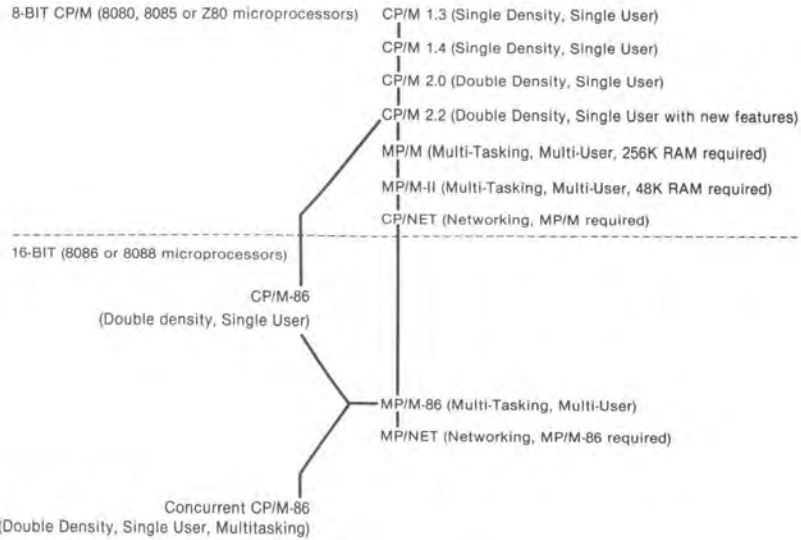
CP/M-Related Operating Systems

INTRODUCTION

This chapter presents basic information on the similarities between CP/M and CP/M-related operating systems. Specifically, this chapter contains information on the MP/M-II, CP/M-86, and MP/M-86 operating systems, including information on the CP/NET network operating system for CP/M and MP/M-II systems. Since the time when CP/M became very popular in the microcomputer industry and community, many operating systems have been developed that were either direct modifications of CP/M or were very much influenced by it. In fact, there are now so many variations of CP/M that have been specially modified to operate on different machines, that the task of describing them all and being up to date would be much too great for inclusion here. It is for this reason that only the CP/M and MP/M series of operating systems available from Digital Research are covered in this chapter.

The information presented will deal with the basic differences between the CP/M and MP/M series of operating systems; information is provided regarding the different commands available, although details concerning how these commands are used will not be dealt with. The purpose of this chapter is to provide the newcomer to these operating systems with a glimpse into what each contains. The next figure provides a basic profile of the evolution of the CP/M operating system into its derivations and close cousins. Only operating systems from Digital Research are included, as other operating sys-

tems based on CP/M are too numerous. Each operating system shown has associated text describing the basic, inherent features, such as support of single or double-density disks, single- or multi-user, etc.



THE MP/M-II OPERATING SYSTEM

MP/M-II is an acronym for Multi-Programming Monitor Control Program for Microcomputers. It is a multi-user, multi-tasking operating system for 8080, 8085, and Z80 microprocessor-based microcomputer systems. MP/M-II is the second version after MP/M version 1, with significantly reduced system memory requirements and with new and enhanced features. MP/M-II is, however, downward compatible with MP/M version 1. Since MP/M-II has pretty much superseded MP/M version 1, only MP/M-II will be discussed in this section. MP/M-II has the same hardware requirements as CP/M, with the following exceptions:

- At least 48K (48 kilobytes) of system Random Access Memory (RAM).
- A clock/timer interrupt.

Apart from these requirements, MP/M-II may have the same hardware configuration as CP/M, although MP/M-II provides the means by which the hardware can be expanded up to sixteen printer devices, and up to sixteen input/output devices, up to eight of which may be console devices (video

terminals or teletypewriters). MP/M-II uses the same 8-inch IBM standard single-density floppy disk format as CP/M, and thus files can easily be transferred between systems. CP/M executable command files can also be used in MP/M-II, although some must be modified for correct operation since MP/M-II uses the system memory in a different manner.

The operation of MP/M-II from the user standpoint is very similar to CP/M, except that many of the console prompts are different, and many of the commands have different names. MP/M-II makes full use of the CP/M USER command function, in that it can support up to 16 users simultaneously. So that the operating system and the user can keep track of the users using the system, the main MP/M-II logged drive prompt includes the current user number, as follows:



All of the CP/M built-in and transient commands are also used by MP/M-II, although in many cases the functions of these commands have been expanded. There are also many commands unique to MP/M-II supplied with the operating system. In general, the command syntax format is the same as CP/M, although many variations exist, including enhanced functions which can make the entering of commands considerably more complex than in CP/M.

THE CP/M-86 OPERATING SYSTEM

The CP/M-86 operating system is based on CP/M, and is designed for operation with 8086 or 8088 microprocessor-based microcomputers. CP/M-86 is basically the same as CP/M insofar as it is a single-user system, supports basically the same commands, and user interaction is virtually indentical. CP/M-86, however, is designed for operation with different microprocessors, and thus supports greatly enhanced hardware interfacing characteristics over CP/M. CP/M-86 retains all of the CP/M built-in (direct) and transient commands, although some of the commands are named differently, and many of the commands have enhanced capabilities. For example, the CP/M-86 assembler command is called "ASM86" instead of ASM, because it is used to assemble only 8086/8088 assembly language programs, and thus is not compatible with CP/M. Several other commands that have specific applica-

tions of this kind are named differently from the corresponding commands in CP/M.

THE CONCURRENT CP/M-86 OPERATING SYSTEM

The Concurrent CP/M-86 operating system is based on CP/M-86 and includes many of the features also provided by CP/M Version 3.0. This operating system differs from CP/M-86 in that it supports concurrency, that is, it permits the user to run more than one program at the same time. This is accomplished with what are called "virtual consoles." A virtual console is created by the operating system itself when a program or task is submitted by the user from the hardware console device. It is treated as if it were a real console, but it is created and is completely controlled by the operating system. If another program is submitted for execution while one is already running, a second virtual console is established to handle the request. Up to three different programs can be handled concurrently.

Concurrent CP/M-86 also provides real-time capability, making it a very useful tool for system development, system emulations, and other real-time applications.

Up to one megabyte of RAM memory is supported, as well as up to 16 disk drives each consisting of up to 512 megabytes each.

THE MP/M-86 OPERATING SYSTEM

The MP/M-86 operating system is a multi-tasking, multi-user operating system that is implemented on 8086 or 8088 microprocessor-based microcomputer. MP/M-86 is designed to basically complement the single-user CP/M-86 operating system, and contains basically the same operating characteristics of the MP/M-II operating system. Compared to MP/M-II, MP/M-86 provides greatly enhanced hardware interfacing characteristics by retaining many of the characteristics supported by CP/M-86, as well as increasing the number of users that can be supported. MP/M-86 can support up to 254 input/output devices, which can be used in almost any combination to support console and printer devices. MP/M-86 also retains all the commands supported by both MP/M-II and CP/M-86, with some enhanced capabilities, including some new commands. The system prompts and command syntax entries are the same as MP/M-II.

THE CP/NET NETWORK OPERATING SYSTEM

The CP/NET operating system provides a means by which several microcomputers installed with CP/M, MP/M (version 1), or MP/M-II can be linked together in a network, so that they may all share each other's resources. The types of resources that can be shared consist of disk files, printer devices, console terminals, and various programs and data bases. The network basically consists of a "master" system installed with MP/M or MP/M-II, and a "slave" system installed with CP/M. The master systems operate as host computers which manage the shared resources that can be accessed by the slave systems in the network. Many types of networks can be established with CP/NET, and several slave systems can be linked with a single master system. Several features are provided by CP/NET, such as the transferring of messages between master and slave systems, and an electronic mail system is provided which allows the master and slave systems to send each other mail.

APPENDIX D

Annotated Bibliography

BOOKS ON CP/M

Fernandez, Judi, and Ashley, Ruth. *Using CP/M*. New York: John Wiley Sons, 1980.
A self-teaching book; it uses a question/answer format while conveying technical information. A quick primer of CP/M commands and syntax.

Brigham, Bruce, ed. *CP/M Summary Guide*. Glastonbury, CN. The Rosseta Stone, 1980.
The book reprints and summarizes the commands available with CP/M, DESPOOL, MAC, TEX, CBASIC, and BASIC-80 from Microsoft. A useful computer-side reference.

Cortesi, David E. *Inside CP/M: A Guide For Users And Programmers*. New York: Holt, Rinehart and Winston, 1982.
Provides very thorough information on the internal aspects of CP/M, including reference information on the usage of commands. It includes information on CP/M-86 and MP/M-II. A very useful book for programmers and those interested in the programming aspects of CP/M.

Hogan, Thom. *CP/M User Guide*. Berkeley, CA. Osborne/McGraw-Hill, 1981.
Provides detailed technical information on all aspects of CP/M and MP/M, in a reference style. A must for all serious CP/M users.

Murtha, Stephen, and Waite, Mitchell. *CP/M Primer*. Indianapolis, IN. Howard W. Sams & Co., Inc., 1980.
Provides an easy-to-understand introduction and description of CP/M and its commands. The book is particularly useful to those who have little or no experience with disk operating systems; and yet it is also useful and interesting to those who already have CP/M experience.

Waite, Mitchell, and Lafore, Robert. *Soul Of CP/M*. Indianapolis, IN. Howard W. Sams & Co., Inc., 1983.

A companion book to the CP/M Bible, it carries on where this one left off with detailed information on the internal aspects of CP/M and programming tips for programmers and those interested in CP/M programming.

Zaks, Rodnay. *The CP/M Handbook With MP/M*. Berkeley, CA. Sybex, 1980.

Discusses CP/M and MP/M commands, programs, and facilities.

MAGAZINE ARTICLES ON CP/M

"A Seminar for Independent CP/M Software Vendors." *DataCast*, No. 001, pp. 24-28.

Discusses methods used by independent CP/M software developers and vendors.

Anderton, Craig. "In Search of the Processed Word." *Kilobaud Microcomputing*, May, 1981, pp. 34-39.

Provides some useful information on word processing concepts, systems and software, some programs of which are compatible with CP/M.

Ballinger, Charles. "HDOS or CP/M?" *Interface Age*, September, 1980 pp. 88-91.

Compares two disk operating systems available for Heath computers.

Barbier, Ken. "CP/M For Single-Drive Systems." *Kilobaud Microcomputing*, September, 1980, pp. 94-98.

Explains the problems with using a single-drive CP/M system; includes an assembly language source listing for a single-drive file copying program.

Barbier, Ken. "CP/M Is for Me." *Kilobaud Microcomputing*, August, 1980, pp. 176-78.

Takes a look at CP/M's way of handling disk space, and provides some useful suggestions.

Barbier, Ken. "Memory Map." *Kilobaud Microcomputing*, March, 1980, pp. 160-163.

Describes how to create a memory map program for any computer, in particular any system based on the 8080 microprocessor. A useful diagnostics tool.

Barbier, Ken. "Start/Exit Routine for CP/M." *Kilobaud Microcomputing*, October, 1980, pp. 82-83.

Provides the source listing to an assembly language program which provides orderly linkages between CP/M and user programs. A useful addtion to CP/M.

Barbier, Ken. "Relocating the Dynamic Debugging Tool." *Kilobaud Microcomputing*, November, 1980, pp. 181-182.

Provides information on how to relocate CP/M's DDT program so that it does not overlay the Console Command Processor (CCP), allowing you to run DDT along with the entire CP/M operating system.

Barbier, Ken. "A Print Utility for CP/M." *Kilobaud Microcomputing*, February, 1981, pp. 150-54.

Provides an assembly language source program for a simple and useful printing utility that runs under CP/M.

Barbier, Ken. "The Secret Life Of the 8085." *Kilobaud Microcomputing*, September, 1980, pp. 114-19.

Provides some useful information for owners of systems that are based on the 8085 microprocessor.

Barker, Lee. "Help With OSI's CP/M." *Dr. Dobbs Journal*, May, 1980, pp. 36-37.

A modification of the BIOS for OSI computers using CP/M to eradicate problems encountered.

Barkstrom, Bruce. "What Time Does the Sun Set?" *BYTE*, July, 1981, pp. 94-114.

A source listing for a program written in CBASIC2 to calculate the rising and setting of the sun throughout the year.

Barley, Kathryn S., and Driscoll, James R. "A Survey of Data-Base Management Systems for Microcomputers." *BYTE*, November, 1981, pp. 208-234.

Provides information on several Data-Base Management Systems (DBMS) available for microcomputers, including those which run directly under CP/M and those which run under CP/M-compatible programming languages, such as Microsoft BASIC, Microsoft FORTRAN, etc.

Biese, Leo, and Iannuccillo, Emilio. "MASTHEAD: Why Not Title Your Printouts?" *Interface Age*, August, 1980, pp. 122-27.

A method of printing large titles on program listings, with source code in both assembly language and Microsoft BASIC.

Blank, Marc S., and Galley, S.W. "How to Fit a Large Program Into a Small Machine (or How to Fit the Great Underground Empire on Your Desk-Top)." *Creative Computing*, July, 1980, pp. 80-87.

Provides information on how to install a large program on your microcomputer, such as large adventure games.

Bove, Tony. "Approaching CP/M." *DataCast*, No. 001, p. 29.

Provides an introduction to the CP/M articles presented in this issue of the magazine. The articles are as follows:

The Illustrated CP/M Turorial - Davies, Dick, pp. 29-38
The Impatient User's Guide to CP/M, pp. 39-41
CP/M User's Reference and Command Summary, pp. 42-43
CP/M User's Reference, pp. 43-51
CP/M Command Summary, pp. 52-55
An Index to CP/M Software and Vendors - Press, Larry, pp. 61-64.

Bove, Tony. "Writing With a CP/M System." *DataCast*, No. 002, pp. 6-11.

Provides information on a using a CP/M system for writing books, manuscripts, manuals, etc.; and covers the operation of a popular CP/M- compatible word processing program called WordStar, available from MicroPro International. ED, the CP/M text editor, is also covered. The following articles are included:

Writing With WordStar, pp. 12-35
The Impatient User's Guide to WordStar, pp. 36-42
ED: A Text Editor, pp. 43-50.

Bozinovic, Dragan. "Software Addressing Modes for the 8080." *BYTE*, March, 1981, pp. 236-240.

Provides some notes on 8080 assembly language programming.

Brockman, D. M. "Binary-to-BCD Converter Program for the 8080." *BYTE*, August, 1981, pp. 418-419.

The article provides a binary-to-BCD assembly language program listing for the 8080.

Calkins, Clark A. "Image Processing With a Printer." *BYTE*, February, 1981, pp. 220-248.

Provides a source listing to an assembly language program written for CP/M version 1.4.

Carbrey, Bruce D. "An Integer Math Package for the 8080." *BYTE*, May, 1981, pp. 204-226.

Provides source listings of a program written in 8080 assembly language for an integer math package.

Cecil, Alex. "ACT: An 8080 Macroprocessor." *Dr. Dobbs Journal*, March, 1978, pp. 20-45.

A "TRAC"-like text interpreter, ACT is written in PL/M. Computer programming students will be interested in the source code.

Charnock, Glen. "Put a Dungeon In Your Computer." *Creative Computing*, July, 1980, pp. 66-78.

Provides a simple BASIC program listing to install a Dungeons & Dragons game on your system. The BASIC program can be easily adapted to a BASIC running under CP/M.

Christensen, Ward. "An 8080 Disassembler." *Dr. Dobbs Journal*, February, 1977, pp. 30-43.

A complete assembly language source listing for an 8080-based disassembler. Documentation explains disassembly to assembly language programmers.

Collins, Rosann; Hines, Theodore; and Rowan, George. "Manipulating Pencil Files; Convert Them to BASIC." *Creative Computing*, August, 1979, pp. 98-99.

Illustrates how to use files created with Electric Pencil using Processor Technology BASIC; applicable to many different CP/M-compatible BASICs.

Conboy, Terry A. "Data Communication Error Measurement." *Kilobaud Microcomputing*, May, 1980, pp. 52-56.

Provides an 8080 machine language program that checks for errors in data-transmission systems.

Corbin, Harold. "An Introduction to Data Compression." *BYTE*, April, 1981, pp. 218-250.

Provides an introduction to data compression in computers, and provides an assembly language listing of an 8080 program.

Cotton, Gene. "How to Solve Your Damaged Disk Dilemma." *Interface Age*, September, 1980, pp. 80-86, 130-31.

An assembly language program to save bad spots on a diskette as a new file name [unused].bad. Contains a fairly detailed explanation of how CP/M maintains information on the diskette.

Craig, John. "A New Kind of Pencil!" *Creative Computing*, February, 1979, pp. 30-33.

Thoroughly reviews the Electric Pencil word processor.

Cotter, Bill. "Microsoft Adventure." *Creative Computing*, May, 1980, pp. 20-21.
Reviews an adventure game program from Microsoft, and discusses games of this type in general. Microsoft Adventure is CP/M-compatible.

Dahmke, Mark. "Wordsmith." *BYTE*, May, 1981, pp. 254-258.
A review of a popular CP/M-compatible word processing package called Wordsmith, available from Scion Corp.

Dennon, Jack. "PUSH! POP! RAM! WOW!" *Kilobaud Microcomputing*, May, 1980, p. 200.
Provides a short program to test RAM memory in an 8080 system.

Didday, Rich. "Universal Data Entry System; In a Car Pooling Application." *Creative Computing*, May, 1980, pp. 102-110.
Reviews UDE from The Software Store and demonstrates the program.

Doan, Daniel. "Computer Simulation of a Solar-Energy System." *BYTE*, July, 1981, pp. 158-172.
A source listing to a program written in CBASIC for simulating parameters of a home solar-energy system.

Epstein, Jake, and Terry, Chris. "Introduction to CP/M; The CP/M Connection." *S100 Microsystems*, September/October, 1980, pp. 10-32.
Continuation of a series of CP/M articles.

Epstein, Jake. "An Introduction to CP/M." *S-100 Microsystems*, January/February, 1980, pp. 6-10; March/April, 1980, pp. 28-33; May/June, 1980, pp. 12-17.
An excellent introduction to CP/M for computer programmers; details specific information.

Eubanks, Gordon. "Notes on CP/M's BASIC-E." *Dr. Dobbs Journal*, October, 1977, p. 35.
Letter to the editor describing the versions of BASIC-E, its public domain status, and how to obtain revised revisions. (Note: Gordon Eubanks developed BASIC-E for a Master's thesis project; it is the ancestor of CBASIC and CBASIC2.)

Firebaugh, M.; Fossum, T.; Sorensen, P.; and, Stone, W. "Comparative Evaluations of Basic Systems." *Creative Computing*, December, 1980, pp. 70-80.
A comparative study of BASIC programs, comparing those that run under CP/M to those of several other popular machines and operating systems.

Fitzgerald, Jim. "Off-the-Shelf Word Processing System." *Kilobaud Microcomputing*, September, 1979, pp. 92-94.
Describes a CP/M system using Electric Pencil.

Foster, Charlie. "Pascal With a Z80." *Interface Age*, November, 1980, pp. 60-62.
Reviews Pascal/Z from Ithaca Intersystems, emphasizes its differences from UCSD Pascal.

Foster, Charlie, and Meador, Richard. "8080 Dynatrace." *S-100 Microsystems*, July/August, 1980, pp. 22-31.
A dynamic screen-orientated assembly language debugging system: Source code is in 8080 assembly language. Article assumes expertise required to modify the output routines.

Frantz, James. "Turn-Key CP/M Systems." *Creative Computing*, December, 1979, pp. 104-07.
Step-by-step directions to make CP/M automatically execute a program after a cold start.

Freed, Edwin E. "Secret Codes Revealed." *Kilobaud Microcomputing*, April, 1981, pp. 58-60.
Provides information on previously undocumented Z80 microprocessor codes. This article is useful to Z80 system users running CP/M.

Friedman, David. "Las Vegas Super Slot: A CP/M Game Machine Program Using Flashwriter I Graphics." *Dr. Dobbs Journal*, November/December, 1980, pp. 10-22.
A game program in assembly language for CP/M systems using the Vector Graphic Flashwriter I board.

Fritzson, Richard. "Data on Disk: Implementing File Systems." *Kilobaud Microcomputing*, January, 1981, pp. 111-115.
Provides an overview of some popular disk filing methods, including the one used by CP/M.

Fritzon, Richard. "The New CP/M: Is It Worth It?" *Kilobaud Microcomputing*, July, 1980, p. 66.
Quickly summarizes the differences between Versions 1.4 and 2.0 of CP/M.

Fritzson, Richard. "Write Your Own FORTH Interpreter." *Kilobaud Microcomputing*, February, 1981, pp. 76-92.
Provides a tutorial on how to write a FORTH interpreter in 8080 assembly language.

Fritzson, Richard. "Write Your Own Pseudo-FORTH Compiler." *Kilobaud Microcomputing*, March, 1981, pp. 44-64.
Provides a tutorial on how to write a FORTH compiler using the FORTH interpreter described in the previous article (see above).

Gagne, Jim. "Vice Versa - Pencil to CP/M and Reverse." *Dr. Dobbs Journal*, March, 1979, pp. 26-29.
How to reformat Electric Pencil files to CP/M format and vice versa. Assembly language source code provided.

Guralnick, Steven. "The Verdict Is In On Spellbinder." *Kilobaud Microcomputing*, May, 1981, pp. 72-78.
Reviews a popular CP/M-compatible word processing program called Spellbinder, available from Lexisoft, Inc.

Hayes Microcomputer Products, Inc. "Modem Control." *Kilobaud Microcomputing*, January, 1981, pp. 192-198.
Provides a listing of a Microsoft BASIC program that runs under CP/M with the Hayes' Micromodem 100 or 80-103A.

Haanstra, Bruce. "Optional Printing with CP/M and Microsoft BASIC." *Interface Age*, November, 1980, pp. 84-86.
Illustrates how to circumvent Microsoft BASIC's interception of the ^P character used to turn the printer on and off in CP/M.

Hallen, Rod. "Back-Space Mod for CP/M and Microsoft BASIC." *Kilobaud Microcomputing*, August, 1980, pp. 48-49.

Provides an assembly language listing of code to add to CP/M CBIOS to eliminate the echoing characteristics when using the delete key with CP/M and Microsoft BASIC.

Hallen, Rod. "Battle of the Assemblers." *Creative Computing*, December, 1979, pp. 42-45.

Compares ASM, supplied with CP/M with ASMB, an assembler available from Technical Systems Consultants. Briefly describes an assembler called "SASSY."

Hallen, Rod. "Double Your Memory, Double Your Fun." *Kilobaud Microcomputing*, May, 1981, pp. 190-192.

Provides some useful information for systems utilizing the Morrow Designs Discus 2D floppy disk drives with CP/M. The article is appropriate to all CP/M users.

Hallen, Rod. "Super Word Processors." *Kilobaud Microcomputing*, June, 1980, pp. 214-217.

Summarizes the features of ED, EDIT, Electric Pencil, WordStar, and The Magic Wand; makes no conclusions or recommendations. Provides a brief bibliography of articles on word processing.

Hallen, Rod. "Tarbell Disk BASIC." *Kilobaud Microcomputing*, May, 1980, pp. 168-170.

Summarizes the features of Tarbell Disk BASIC; emphasizes those features that are distinct from other interpreters.

Hallen, Rod. "The Battle of The Word Processors." *Creative Computing*, November, 1979, pp. 48-53.

Reviews ED, TEX, EDIT, and Electric Pencil.

Hamilton, R.W. "WPDaisy Word Processing System." *Creative Computing*, May, 1979, pp. 36-41.

Reviews a little-known, but powerful, word processing system.

Harrell, Martin D. "Operation Codes of the 8080, 8085, and Z80 Processors." *BYTE*, March, 1980, pp. 194-207.

Provides tables listing the operation codes for the mentioned microprocessors, showing differences. This is useful information for assembly language programming.

Hart, Glenn. "A New BASIC From Tarbell." *Creative Computing*, January, 1980, pp. 20-23.

Reviews a slightly different BASIC available for CP/M systems.

Hart, Glenn. "Magic Wand Word Processor." *Creative Computing*, August, 1980, pp. 38-45.

Reviews the features of the Magic Wand word processing system.

Hasset, Greg. "How to Write an Adventure." *Creative Computing*, July, 1980, pp. 88-90.

Provides general information on how to write adventure games on your microcomputer.

Heintz, Carl. "Analyst: Another Data-base Manager." *Interface Age*, December, 1980, pp. 42-44.

Reviews the Analyst data base management system from Structured Systems.

Heintz, Carl. "Maxiledger." *Interface Age*, September, 1980, pp. 42-44.

Reviews the Maxiledger general ledger accounting system available from CompuMax Associates.

Heintz, Carl. "A Peach of a General Ledger Program." *Interface Age*, October, 1980, pp. 46-48.
Reviews the Peachtree General Ledger system.

Heintz, Carl. "Pearl - A Novel Programming Gem." *Interface Age*, November, 1980, p. 47.
A program which "writes" programs; Pearl writes custom application packages from user input. Explains the power inherent in such a concept.

Heyman, Victor. "IDSWORD - The Comprehensive Processing System for Home and Business." *Creative Computing*, May, 1979, pp. 43-44.
Reviews another little-known word processing system compatible with CP/M systems.

Hicks, Bruce. "Lost & Forgotten Island." *Creative Computing*, March, 1980, pp. 124-131.
Provides source listing to a game named after the article's title, written in Microsoft BASIC.

Hoffer, W. C. "Data and Time for the CP/M Operating System." *Interface Age*, August, 1978, pp. 152-156.
Assembly language routines enable a Compu Time board to perform automatic time and dating of program listings.

Hogan, Thom. "CP/M and You." *Kilobaud Microcomputing*, February, 1980, pp. 183-184.
Provides some enlightening information on CP/M, its use in the industry, and how it affects you as a microcomputer user.

Hogg, Douglass. "How Good is Microsoft's FORTRAN-80?" *Creative Computing*, January, 1979, pp. 62-67.
Reviews Microsoft's FORTRAN-80 and examines the FORTRAN implementation on a microcomputer.

Hunt, Daniel S. "DATALINE." *BYTE*, March, 1981, pp. 216-222.
Provides a listing for a program written in Microsoft BASIC used to take the drudgery out of writing assembly language subroutines for BASIC programs.

Jackson, David R. and Callahan, John M. "Energy Conservation with a Microcomputer." *BYTE*, July, 1981, pp. 178-208.
A program written in Commodore PET BASIC, that can easily be adapted to Microsoft BASIC. The program produces a table of conduction and ventilation loads and the yearly energy requirements for a given house.

Johnson, Bob. "Business Software Review." *Interface Age*, August, 1979, pp. 38-39.
Reviews the Graham Dorian Apartment Management and Payroll software packages.

Kaapke, Richard. "DUKEDOM." *Creative Computing*, February, 1980, pp. 100-110.
Provides the source listing for a game called DUKEDOM written in Microsoft BASIC.

Kendall, Wallace. "Prettyprinting With Microsoft BASIC." *Kilobaud Microcomputing*, May, 1979, p. 80.
Presents a simple way to format program listings using Microsoft BASIC.

Kern, Christopher. "The BDS C Compiler." *BYTE*, June, 1981, pp. 356-362.
Reviews the BDS C Compiler, available from Lifeboat Associates, which runs on an 8080-based CP/M system. The article compares C with other popular programming languages such as Pascal and BASIC.

Kern, Christopher. "Printf for the C Function Library." *BYTE*, May, 1981, pp. 430-434.
Provides a source listing for a program that provides the Printf function to early versions of the BDS C programming language. The BDS C language is compatible with 8080 CP/M systems.

Kildall, Gary. "CP/M: A Family of 8- and 16-Bit Operating Systems." *BYTE*, June, 1981, pp. 216-232.
The author of CP/M provides general information on CP/M: its past, present, and future; and how it interacts with today's and tommorow's microcomputers.

Kildall, Gary. "Simple Technique for Static Relocation." *Dr. Dobbs Journal*, February, 1978, pp. 10-13.
The author of CP/M describes the CP/M relocation technique.

Klem, David C. "Solar Specs." *Kilobaud Microcomputing*, March, 1980, pp. 68-70.
Provides a CBASIC program that computes solar heat potential. The program can run on a CP/M system.

Knecht, Ken. "CBASIC Review." *80 Microcomputing*, April, 1980, pp. 130-32.
Compares CBASIC features with Microsoft BASIC; introduces FMG's CP/M operating system for the TRS-80.

Larson, James. "The Ins and Outs of CP/M." *BYTE*, June, 1981, pp. 268-300.
This article clarifies input/output and disk-interfacing aspects of CP/M, that are otherwise not provided in the documentation, or are poorly documented. Provides sample I/O assembly language routines, including some other useful programming information.

Lehman, John A. "A Reformatter for CP/M and IBM Floppy Disks." *BYTE*, April, 1981, pp. 94, 96.
Reviews a file-transfer program called Reformatter, available from MicroTech Exports. The program is used to transfer files on floppy diskettes between CP/M and IBM machines.

Lemmons, Phil. "Five Spelling-Correction Programs for CP/M-Based Systems." *BYTE*, November, 1981, pp. 434-448.
The article provides a software review of five popular spelling-correction programs that run under CP/M. Provides specifications for each, with comparative notes. The article also provides general and basic information on the use of these types of programs on microcomputers used for word processing by writers, secretaries and engineers.

Levitan, Steven P., and Bonar, Jeffrey G. "Three Microcomputer LISPs." *BYTE*, September, 1981, pp. 388-412.
The article provides a software review of three LISP programming languages, two of which are CP/M-compatible. Discusses the use of these languages, provides detailed listings of examples, and provides information on their use in the development of "artificial intelligence."

Lindsay, John. "New Version of BASIC." *Kilobaud Microcomputing*, May, 1980, pp. 72-74.
Summarizes the features of Microsoft BASIC version 5. Discusses the differences between versions.

Lindsay, Jon. "For CP/M: Automatic Program Execution on Start-up." *Kilobaud Microcomputing*, January, 1981, pp. 184-186.
Provides a means by which CP/M can be modified to provide automatic execution of user program upon system start-up.

Lubar, David. "Pearl." *Creative Computing*, December, 1980, p. 68.
Provides a review of the popular program called Pearl, available from Computer Pathways Unlimited. The program is CP/M-compatible, and is a program which is used to generate user application programs, in response to simple input.

Lutz, Dick. "Sharpening Your Pencil." *Creative Computing*, March, 1980, pp. 30-35.
Reviews a program designed to add to the features of Electric Pencil.

Lutz, Dick. "WordStar vs. Electric Pencil." *Creative Computing*, February, 1980, pp. 17-22.
Provides a detailed comparison between two popular CP/M-compatible word processing programs.

Magruder, Bob. "The New WPDaisy: Word-Processing Software." *onComputing*, Fall 1980, pp. 68-74.
Extensively reviews WPDaisy.

Marino, Albert J. "Sorta Super Fast." *Kilobaud Microcomputing*, April, 1981, pp. 164-166.
Provides useful information on 8080 assembly language programming.

Maurer, W. D. "8080 Assembly Language for the Totally Lost." *onComputing*, Spring 1980, pp. 81-85.
Provides some useful tips for 8080 assembly language programming for the beginner.

Maxey, Dr. E. Stanton. "Tracking the Planets." *Kilobaud Microcomputing*, March, 1981, pp. 130-34.
Provides a simple program written in Microsoft BASIC to calculate planetary coordinates.

McClure, James. "CBASIC - A Review." *Creative Computing*, September, 1979, pp. 48-51.
Reviews a frequently used high level language. Provides timing comparisons for a series of standardized benchmarks, indicating CBASIC's relatively slow speed on many tests.

McClure, James. "Microsoft vs. MicroFocus COBOL." *Creative Computing*, March, 1980, pp. 20-29.
Compares the features of the two primary implementations of COBOL available to CP/M users.

McClure, James. "A Personal Finance System." *Kilobaud Microcomputing*, June, 1979, pp. 74-78; July, 1979, pp. 50-56; August, 1979, pp. 66-75.
An extensive CBASIC2 program for managing personal finances.

McClure, James. "Shoplist: The Latest Kitchen Utensil." *Creative Computing*, May, 1980, pp. 74-80.

Provides a source listing for a program written in Microsoft BASIC.

Miller, Alan. "An Interrupt-Driven Keyboard Buffer." *Interface Age*, October, 1980, pp. 106-107, 137-141.

A different approach for implementing a keypress detection routine for North Star BIOS in CP/M.

Miller, Alan. "BASCOM: Microsoft's BASIC Compiler for the 8080/Z80." *Interface Age*, July, 1980, pp. 124-26.

Discusses the differences between compilers and interpreters, emphasizing the features of Microsoft's compiler BASIC. Presents sample programs.

Miller, Alan. "CBASIC: A Business-Orientated Language for CP/M." *Interface Age*, August, 1979, pp. 116-119.

Summarizes and reviews the CBASIC high level language for CP/M.

Miller, Alan. "CP/M for the TRS-80 Model II: Lifeboat and FMG Corp. Versions." *Interface Age*, November, 1980, pp. 94-98.

A detailed comparison of two different implementations of CP/M for the Model II.

Miller, Alan. "CP/M Part 2 - A Macro Assembler & Other Goodies." *Interface Age*, December, 1978, pp. 130-135.

Reviews the MAC assembler available from Digital Research, the SID symbolic instruction debugger also from Digital Research, and a sample CBIOS for the North Star CP/M system.

Miller, Alan. "Diagnostics Package for CP/M." *Interface Age*, October, 1980, p. 104.

Reviews the Supersoft CP/M Diagnostics package; these programs test memory, CPU, disk, printer and console.

Miller, Alan. "Structured Assembly-Language Programming for the 8080." *Interface Age*, November, 1979, pp. 153-155.

Reviews SP80, a structured assembly language set of macro routines available for CP/M systems. Compares the Digital Research MAC assembler with SP80.

Miller, Alan. "ZSID Z80 Debugger for CP/M." *Interface Age*, August, 1980, pp. 88-90.

Reviews the ZSID symbolic debugger for assembly language programs available from Digital Research; also mentions the DESPOOL program, from Digital Research.

Miller, Alan. "The Electric Pencil for CP/M." *Interface Age*, August, 1978, pp. 148-149.

A review of the popular word processing package.

Miller, Alan. "Pascal for CP/M: Digital Marketing's PascalM." *Interface Age*, September, 1980, pp. 96-103.

Reviews Pascal/M and discusses the difference between Pascal and other high level languages.

Monagan, James. "Efficient Data Storage For Microsoft BASIC." *Kilobaud Microcomputing*, November, 1980, pp. 68-69.

Provides information on how to improve packing and unpacking numeric data for efficient data storage on random disk files, using Microsoft BASIC running under CP/M.

Morgan, Chris. "The New 16-Bit Operating Systems, or, The Search for Benutzerfreundlichkeit." *BYTE*, June, 1981, Editorial by the Editor in Chief, pp. 6, 10.

An editorial on the 16-bit version of CP/M, also mentioning other similar operating systems.

North, Steve. "The CP/M Disk Operating System." *Creative Computing*, November/December, 1978, pp. 52-53.

A short summary of the CP/M operating system and the programs that accompany it.

North, Steve. "Creative Computing Reviews Five Software Packages - Tiny C, Microsoft BASIC 5.0, Research Machines Z80 Algol, Structural Analysis SP80 Macros, Digital Research CP/M 2.0 and MP/M." *Creative Computing*, March, 1980, pp. 40-44.

Summarizes the features of the titled packages.

Parsons, Ronald. "UCSD Pascal to CP/M File Transfer Program." *Dr. Dobbs Journal*, August, 1979, pp. 12-16.

Assembly language program transfers from UCSD Pascal format to CP/M file structure.

Parsons, Thomas W. "All About ASCII." *Kilobaud Microcomputing*, November, 1980, pp. 129-132.

Provides general, useful information about the American Standard Code for Information Interchange (ASCII), and how it is utilized by several programs and operating systems, including CP/M.

Pelczarski, Mark. "Microsoft Softcard." *BYTE*, November, 1981, pp. 152-162.

The article reviews the Softcard for the Apple-II, available from Microsoft. The Softcard is plug-compatible with the Apple-II and is used to provide a Z80 microprocessor so that the CP/M operating system can be run.

Platt, Charles; Dowden, Tony; and, Head, Gene. "Word Processing Roundup: WordStar & WP6502, Spellguard, VEDIT." *Kilobaud Microcomputing*, May, 1981, p. 40, p. 46 and pp. 50-51 respectively.

Provides a review of a word processor program called WordStar, available from MicroPro International; a spelling-error checking program called Spellguard, available from Innovative Software Applications; and, a text editor called VEDIT, available from CompuView Products, Inc.

Pournelle, Jerry. "BASIC, Computer Languages, and Computer Adventures." *BYTE*, December, 1980, pp. 222-238.

The article contains general information on computer adventure games and the machines and programming languages they run under. Provides extensive information on the use of these programs with CP/M.

Pournelle, Jerry. "New Software, New Hardware, Computer Languages, and Games." *BYTE*, November, 1981, pp. 449-457.

The article covers new hardware and software, much of which is CP/M-compatible.

Pournelle, Jerry. "Omikron TRS-80 Boards, NEWDOS+, and Sundry Other Matters." *BYTE*, July, 1980, pp. 198-208.

A rambling look at Pournelle's experiences with microcomputers in writing; a popular science fiction writer (*Mote in God's Eye, Lucifer's Hammer*, and others). Pournelle offers a personal account, and discusses the use of CP/M in writing.

Pournelle, Jerry. "Operating Systems, Languages, Statistics, Pirates, and the Lone Wolf." *BYTE*, January, 1982, pp. 132-158.

Provides interesting information on operating systems in general, comparing CP/M to other operating systems, and discusses both the present and future of these operating systems. The article also provides information in CP/M-compatible languages.

Press, Larry. "A Review of Four Text-Formatting Programs." *onComputing*, Fall 1980, pp. 48-54.

Reviews S-80, TEX, TPS, and Textwriter.

Press, Larry. "Word Processors: A Look at Four Popular Programs." *onComputing*, Summer 1980, pp. 38-52.

A detailed review of Auto Scribe, Electric Pencil, Magic Wand, and WordStar.

Pugh, Tim. "Intelligent Terminal Implementation on S100 Bus." *Dr. Dobbs Journal*, June/July, 1978, pp. 4-16.

A well-documented assembly language program that allows communication between any CP/M system using a IDS-88 modem board and any other computer system.

Raymer, Paul. "Weather Station." *Creative Computing*, December, 1980, pp. 142-143.

A simple experimental program for weather forecasting written in Applesoft BASIC, and can be easily adaptable to Microsoft BASIC.

Rhodes, Cheryl. "The DIALOG Information Retrieval System." *DataCast*, No. 002, pp. 51-62.

Covers the information retrieval system called DIALOG, which is compatible with many computers. Describes ways in which a similar system can be used with CP/M.

Rolander, Thomas A., "Microcomputer Software Meshes with Local Nets." *Electronics*, January, 1982, pp. 96-99.

The author, who works for Digital Research, discusses the use of CP/M and similar operating systems by microcomputers in local networks. The article provides some interesting insights into the workings of CP/M, and sheds light on the future of CP/M and other based-on or similar operating systems.

Rose, Lance E. "Position-Independent Code for the 8080." *Kilobaud Microcomputing*, June, 1980, pp. 202-203.

Provides information on how to create a code-relocation method for the 8080 that requires no special loader or systems software changes.

Roxburgh, Alastair. "Fast Fourier Comes Back." *BYTE*, May, 1981, pp. 458-461.

The author has provided fourier program written for the 8080 microprocessor, converted from an original program written for the M6800 microprocessor.

Sanger M.D., Joseph Jay. "The Electronic Librarian." *Kilobaud Microcomputing*, November, 1979, pp. 44-62.

A BASIC-E program for maintaining an information data base.

Sclawy, Alan. "CP/M Encryption Prescription." *Kilobaud Microcomputing*, November, 1980, pp. 42-46.
Provides information on how to encrypt files for security, using BASIC under CP/M.

Sellier, Jane. "Auto Scribe." *Creative Computing*, January, 1980, pp. 24-25.
Provides a general overview of the capabilities of Auto Scribe, also providing hardware compatiblity information.

Sjowall, Tor. "CP/M to UCSD Pascal File Conversion." *Dr. Dobbs Journal*, October, 1980, pp. 16-19.
Pascal source code for converting CP/M files to the UCSD Pascal format.

Stewart, John. "CP/M Primer: a Most Sophisticated Operating System." *Kilobaud Microcomputing*, April, 1978, pp. 30-34.
An early detailed description of the CP/M operating system.

Stok, Glenn. "Video HARDCOPY For CP/M." *Kilobaud Microcomputing*, October, 1980, pp. 168-173.
Provides a source listing of an assembly language program to enable the CP/M user to print a hard copy of any information shown on the console screen.

Stoutemyer, Devid R. "Symbolic Math Using BASIC." *BYTE*, October, 1980, pp. 232-246.
The article provides a BASIC listing of a program to provide the expansion of polynomials. The program can be adapted to most CP/M-compatible BASIC programming languages.

Suits, David B. "Base Arithmetic." *Creative Computing*, November, 1980, pp. 146-149.
Provides a source listing for a program written in Compucolor Disk BASIC, which can easily be adaptable to Microsoft BASIC. The program is used to convert certain base number to another given base.

Swayne, Patrick. "Disassembler For the Heath H8." *Kilobaud Microcomputing*, August, 1980, pp. 88-94.
Provides a source listing of a disassembler program for Heath H8, and indicates that it can be easily adaptable to other 8080 and 8085 systems.

Terry, Chris. "The CP/M Connection." *S-100 Microsystems*, July/August, 1980, pp. 32-35.
Discusses the CP/M relocation method and how to interface CBIOS and make other modifications to a relocated system.

"Upgraded CP/M Floppy Disc Operating System." *Dr. Dobbs Journal*, November, 1976, p. 51.
A detailed summary of the features of CP/M revision 1.4. Includes descriptions of the accompanying 1.4 manuals.

Van Buer, Darrel. "A Table-Driven Assembler on CP/M." *Dr. Dobbs Journal*, February, 1980, pp. 18-25.
A PL/M source code for a macro assembler.

VanHorn, Eric. "SuperSort by MicroPro International." *Creative Computing*, July, 1979, pp. 34-37.
Reviews the features of SuperSort.

Warren, Jim. "First Word on a Floppy-Disc Operating System." *Dr. Dobbs*, April, 1976, p. 5.
Provides first available information on CP/M.

Warren, Jim. "The Time for Floppys is Just About Now!" *Dr. Dobbs Journal*, August, 1976, p. 5.
Reviews the available floppy disk systems of the time, and summarizes CP/M features.

Williams, Gregg., Editor. "The muSIMP/muMATH-79 Symbolic Math System." *BYTE*, November, 1980, pp. 324-338.
Provides a product review of the muSIMP and muMATH-79 programs, available from The Soft Warehouse. These programs are compatible with the 8080, 8085, and Z80 microprocessors, and run under CP/M and other similar operating systems.

Williams, Greg., sr. ed. "Three Versions of APL." *BYTE*, April, 1981, pp. 188-208.
Reviews three APL ("A Programming Language") programs compatible with CP/M: Softronics APL, Vanguard APL/DTC, and Vanguard APL/V80. The article indicates differences between these programs, and provides information on how each interacts with CP/M.

Willoughby, Steve. "Hardcopy Device Driver Programs for CP/M." *Dr. Dobbs Journal*, September, 1980, pp. 34-37.
Assembly language routines to be included in CP/M CBIOS section for Diablo 1650 or Teletype printers.

"Word Processing Directory." *Kilobaud Microcomputing*, May, 1981, pp. 68-70.
Provides a directory of several popular word processing programs, many of which are CP/M-compatible.

Work, Jim. "Hippity-Hop Memory Test." *Kilobaud Microcomputing*, May, 1981, pp. 166-75.
Provides an 8080 assembly language program to test the memory of your system. The code was written for the North Star system, but can be easily adaptable to any 8080, 8085, or Z80 CP/M system.

Zarucki, Paul. "Indirect I/O Addressing on the 8080." *BYTE*, August, 1981, pp. 402-403.
Provides a useful 8080 assembly program listing to provide subroutines for performing indirect input and output operations with the 8080 microprocessor.

BIBLIOGRAPHY

ASSEMBLY LANGUAGE PROGRAMMING BOOKS

8080/8085 Assembly Language Programming Manual. Santa Clara, CA. Intel Corp., 1980.

Leventhal, Lance, *8080A/8085 Assembly Language Programming.* Berkeley, CA. Osborne/McGraw-Hill, 1979.

Leventhal, Lance, *Z80 Assembly Language Programming.* Berkeley, CA. Osborne/McGraw-Hill, 1979.

Myers, Glenford, *Reliable Software Through Composite Design.* Van Nostrand Reinhold, 1975.

Zaks, Rodnay. *Programming the Z80.* Berkeley, CA. Sybex, 1979.

APPENDIX E

Glossary of Commonly Used Terms

Allocation, Dynamic – The reassignment of peripherals within a given program.

Allocation, Static – The assignment of peripherals to a given job.

Allocation Technique – The method of providing a process with access to a shared resource.

BDOS – Basic Disk Operating System or Basic DOS; portion of an operating system which handles all of the disk file transactions.

Binding – The act of assigning absolute addresses to a program.

BIOS – Basic Input/Output (I/O) System; portion of an operating system used to handle all machine-dependent transactions.

Blocked List – A catalog of the processes waiting for microprocessor time or for completion of an I/O operation.

Blocking – The process of combining more than one record into one block to make data transfers more efficient.

Buffering – The process of using areas of memory to isolate I/O devices from one another and from the CPU.

Catalog – A means by which several items can be grouped together for easy reference and access. Also, see **CPU Directory**.

CCP – Console Command Processor.

Chaining – The ability of an executing program to call another program that resides on disk.

Common – An area of memory maintained for the purpose of passing data or parameters between programs.

Console Command Processor (CCP) – The portion of an operating system that controls all of the interaction with the operator of the computer while in the main command mode of operation.

Constrained Allocation – A resource-allocation strategy that specifies all the resource a process will need, but does not prevent execution unless a deadlock occurs.

Contiguous Allocation – An allocation method that assigns physically adjacent sectors to a file.

CPU Directory – A file containing information concerning the other files on a mass-storage device such as a diskette; also known as a catalog.

Deadlock – A condition that exists when a process is blocked in a state and in all future states that the system can reach.

Deadly Embrace – A situation in which two processes each unknowingly wait for resources held by the other.

Device Independence – An operating system feature that frees the user from considering device-specific details. It uses mnemonics to refer to specific devices; changing these mnemonics redirects I/O to another device.

Direct Memory Access (DMA) – A means of providing fast peripherals with access to system memory without going through the CPU Directory or catalog.

DOS – Disk Operating System. The CP/M operating system is a DOS.

Executive – An operating system routine responsible for decision making.

Extension – A portion of file name, usually between 1 to 3 characters in length which appends the file name, used to define the type of file.

File – A specific collection of data stored on a disk or other media, referenced by a specific name.

File Control Block (FCB) – A data structure in main memory used for keeping track of files in use.

File Management System – The part of an operating system that controls the organization and allocation of disk files, which might consist of one or more sectors.

File Type – See Extension.

Index – A number representing the relative position of a byte in either a record or a byte.

Interrupt – A break in the normal flow of a system or routine from which flow can resume later.

I/O – Input/Output.

I/O Supervisor – The portion of an operating system that provides routines for I/O procedures.

Job – The collection of activities needed to accomplish a specified amount of work.

Linked List – A list formed by typing together (with pointers) several items such as sectors on a disk.

Lock Byte – An entity used to represent a resource in synchronization schemes; also termed a semaphore.

Memory Protection – A method of ensuring that the contents of main memory within certain variable limits are not altered or INADVERTENTLY destroyed.

Noncontiguous Allocation – An allocation method that assigns physically nonadjacent sectors to a file.

Nucleus – The most basic level of an operating system; creates and destroys software processes used to implement abstract processes.

Operating System – An organized collection of techniques and procedures used for operating a computer.

Overlaying – The technique of repeatedly using the same blocks of internal storage during different stages of a program; for example, when one routine is no longer needed, another routine can replace all or part of it.

Polling Procedure – A routine for sequentially checking each I/O device to determine whether it requires servicing.

Primitive – An operating provided by a nucleus for use in synchronization.

Priority – A parameter designating the relative urgency of a task or process.

Process – A computation that can occur concurrently with other computations; an operating system's basic unit of computation.

Process Control Block (PCB) – A data structure that uniquely defines a given process.

Program (Code) – A set of instructions that tells a computer step by step exactly how to handle a job.

Re-Entrant Code – A program task or routine that can be executed simultaneously by more than one process.

Relocation, Dynamic – The act of assigning absolute memory addresses when a program is loaded into memory.

Relocation, Static – The act of assigning absolute addresses at linking time.

Rotational Ordering – A method of organizing I/O requests to a disk to reduce total service time.

Segmentation – A technique for managing variable-sized areas of memory, termed segments, that contain logical parts of a program.

Semaphore – See **Lock Byte**.

Service Request – The appeal by a process or task for access to a system resource.

Single Contiguous Allocation – A memory-allocation scheme that assigns all available memory as one block.

Spin Block – A loop created when a process keeps checking the state of a flag of status bit while waiting for an event to occur.

Spooling – Simultaneous peripheral operations on line; used to convert a dedicated device into a shared one.

Standard Allocation Pattern – A resource-allocation scheme designed to prevent deadlocks.

Swapping – A technique similar to overlaying; involves moving processes between main memory and auxiliary storage in order to multiplex main memory.

Sysgen – System generation, the process by which an operating system is configured out of individual system components to accommodate a particular hardware configuration.

Task – A routine that forms the lowest self-contained unit of a job or process.

Time Slicing – A technique that shares microprocessor time among several processes. The quantum of time allocated to a process is termed a time slice.

TPA – Transient Program Area.

Transient Area – The space in memory available for user programs and system utilities.

Transient Program Area (TPA) – See **Transient Area**.

Urgency – The degree to which a task or process requires attention; determined by the priority of the task or process.

Utilities – Routines used in housekeeping functions and I/O.

APPENDIX F

ASCII Crossreference

The American Standard Code for Information Interchange (ASCII) is a seven-bit code also known as the USA Standard Code for Information Interchange (USASCII). The ASCII code is used as a means by which people can easily communicate with computers. It translates the binary numbers of machine data to familiar alpha and numeric symbols which can be printed or displayed on a terminal, and also interprets signals created by the pressing of keys on specially laid out keyboards for translation to appropriate machine binary data.

Because eight-bit codes are much more common on computers than seven-bit codes, ASCII is commonly embedded in an eight-bit code, ASCII-8, which is what is shown in the following tables. The arrangement of the tables shows the 128 ($=2^7$) possible combinations for the seven-bit USASCII code. The 128 seven-bit ASCII definitions and sequence do not change between eight-bit and seven-bit ASCII. The eighth bit (the leftmost bit) is used in ASCII-8 as a *parity* bit. The parity bit is equal to a 1 when there is an odd number of 1-bits in the rightmost seven-bit code, and is equal to a 0-bit when there is an even number. The parity bit is often used in computers to verify the integrity of transmitted data. If a received parity bit does not properly represent the seven-bit code to which it is attached, then a "Parity Error" will occur, thus informing the computer user that data is not being transmitted properly.

The codes provided by ASCII are divided into three major categories, they are:

1. **CONTROL CHARACTERS:** These are special nonprintable characters which are used to control the way in which the computer, terminal, and other devices are to operate. All control characters have a binary representation, but only a few will have symbol representations which can be interpreted by a terminal user, and will only be displayed when instructed to do so. ASCII Control Characters are further divided into three categories:

   A. **Communication Control Characters:** These characters are used to control the transmission and reception of data. They are used as markers to indicate the beginning and end of blocks of data, and to control characteristics of transmission.

   B. **Format Effector Characters:** These characters are used to control the way in which data is formatted, whether it be in the stage of transmission, displaying or printing on a terminal, or being stored on storage media. They consist of characters which control such things as carriage returns, line feeds, tabs, etc.

   C. **Information Separator Characters:** These characters are used to control the way in which blocks of data are separated from each other. These characters produce file, group, record and unit separators so that data can be organized in an intelligible and organized manner for the computer and the computer user.

   There are other special control characters which do not fall under any of the preceding three sub-categories. These characters are usually used for the operation of terminals (teleprinters, video terminals, etc.) or other devices, and do not directly affect the way the computer itself operates.

2. **ALPHA-NUMERIC CHARACTERS:** These characters consist of all letters of the English alphabet, and the decimal digits 0 through 9.

3. **SPECIAL GRAPHIC CHARACTERS:** These characters consist of grammatical symbols (periods, commas, colons, etc.), economics symbols (dollar sign, percent, etc.), and general and scientific symbols (less than, braces, circumflex, etc.).

DEC X_{10}	HEX X_{16}	OCT X_8	ASCII	Terminal Key *
0	00	00	NUL	CTRL/1
1	01	01	SOH	CTRL/A
2	02	02	STX	CTRL/B
3	03	03	ETX	CTRL/C
4	04	04	EOT	CTRL/D
5	05	05	ENQ	CTRL/E
6	06	06	ACK	CTRL/F
7	07	07	BEL	CTRL/G
8	08	10	BS	CTRL/H, BACKSPACE, LEFT-ARROW
9	09	11	HT	CTRL/I, TAB
10	0A	12	LF	CTRL/J, LINE FEED, DOWN-ARROW
11	0B	13	VT	CTRL/K, UP-ARROW
12	0C	14	FF	CTRL/L, RIGHT-ARROW
13	0D	15	CR	CTRL/M, RETURN
14	0E	16	SO	CTRL/N
15	0F	17	SI	CTRL/O
16	10	20	DLE	CTRL/P
17	11	21	DC1	CTRL/Q, XON, NO SCROLL (OFF)
18	12	22	DC2	CTRL/R, TAPE
19	13	23	DC3	CTRL/S, XOFF, NO SCROLL (ON)
20	14	24	DC4	CTRL/T, TAPE
21	15	25	NAK	CTRL/U
22	16	26	SYN	CTRL/V
23	17	27	ETB	CTRL/W
24	18	30	CAN	CTRL/X
25	19	31	EM	CTRL/Y
26	1A	32	SUB	CTRL/Z, CLEAR SPACE
27	1B	33	ESC	ESC, ESCAPE
28	1C	34	FS	CTRL/\ (Control/Reverse-Slant)
29	1D	35	GS	CTRL/^ (Control/Grave Accent)
30	1E	36	RS	CTRL/= (Control/Equals sign), HOME
31	1F	37	US	CTRL/- (Control/Hyphen), NEW LINE
32	20	40	SP	Space (SPACE BAR)
33	21	41	!	! (Exclamation mark)
34	22	42	"	" (Quotation Mark)
35	23	43	#	# (Number Sign or Octothorpe)
36	24	44	$	$ (Dollar Sign)
37	25	45	%	% (Percent)
38	26	46	&	& (Ampersand)
39	27	47	'	' (Apostrophe or Acute Accent)
40	28	50	(	((Opening Parenthesis)
41	29	51	)	) (Closing Parenthesis)
42	2A	52	*	* (Asterisk)
43	2B	53	+	+ (Plus)

DEC X_{10}	HEX X_{16}	OCT X_8	ASCII	Terminal Key *
44	2C	54	,	, (Comma)
45	2D	55	-	- (Hyphen, Dash, or Minus)
46	2E	56	.	. (Period)
47	2F	57	/	/ (Forward Slant)
48	30	60	0	0
49	31	61	1	1
50	32	62	2	2
51	33	63	3	3
52	34	64	4	4
53	35	65	5	5
54	36	66	6	6
55	37	67	7	7
56	38	70	8	8
57	39	71	9	9
58	3A	72	:	: (Colon)
59	3B	73	;	; (Semicolon)
60	3C	74	<	< (Less Than)
61	3D	75	=	= (Equals)
62	3E	76	>	> (Greater Than)
63	3F	77	?	? (Question Mark)
64	40	100	@	@ (Commercial At)
65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F
71	47	107	G	G
72	48	110	H	H
73	49	111	I	I
74	4A	112	J	J
75	4B	113	K	K
76	4C	114	L	L
77	4D	115	M	M
78	4E	116	N	N
79	4F	117	O	O
80	50	120	P	P
81	51	121	Q	Q
82	52	122	R	R
83	53	123	S	S
84	54	124	T	T
85	55	125	U	U
86	56	126	V	V
87	57	127	W	W

DEC X_{10}	HEX X_{16}	OCT X_8	ASCII	Terminal Key *
88	58	130	X	X
89	59	131	Y	Y
90	5A	132	Z	Z
91	5B	133	[	[(Opening Bracket)
92	5C	134	\	\ (Reverse Slant)
93	5D	135	]	] (Closing Bracket)
94	5E	136	^	^ (Carat or Circumflex)
95	5F	137	_	_ (Underscore or Underline)
96	60	140	`	` (Grave Accent)
97	61	141	a	a
98	62	142	b	b
99	63	143	c	c
100	64	144	d	d
101	65	145	e	e
102	66	146	f	f
103	67	147	g	g
104	68	150	h	h
105	69	151	i	i
106	6A	152	j	j
107	6B	153	k	k
108	6C	154	l	l
109	6D	155	m	m
110	6E	156	n	n
111	6F	157	o	o
112	70	160	p	p
113	71	161	q	q
114	72	162	r	r
115	73	163	s	s
116	74	164	t	t
117	75	165	u	u
118	76	166	v	v
119	77	167	w	w
120	78	170	x	x
121	79	171	y	y
122	7A	172	z	z
123	7B	173	{	{ (Opening Brace)
124	7C	174	|	| (Vertical Line or Logical OR)
125	7D	175	}	} (Closing Brace)
126	7E	176	~	~ (Tilde)
127	7F	177	DEL	DEL, DELETE, RUBOUT

*Those key sequences consisting of "CTRL/ " are typed in by simultaneously pressing the CTRL key and the key indicated. These sequences are based on those defined for most stan-

dard terminals, such as the Diablo 1640 keyboard and the Televideo series of terminals. These key sequences may be defined differently on other keyboards.

ABBREVIATIONS:

DEC = Decimal (Base 10)
HEX = Hexadecimal (Base 16)
OCT = Octal (Base 8)
ASCII = American Standard Code for Information Interchange
CCITT = Comité Consultatif International Télégraphe et Téléphone (International Telegraph and Telephone Consultative Committee).

ASCII CONTROL CHARACTER DEFINITIONS

ACK (Acknowledgement)—This is a communication control character which is used as a general "yes" answer to various queries, but it also sometimes indicates "I received your last transmission and I'm ready for your next."

BEL (Bell)—This is a general-purpose control character which is used to activate a bell, beeper, or other audible alarm on a device to which it was sent.

BS (Backspace)—This is a format effector control character which is used to move the carriage, print head, or cursor back one space or position.

CAN (Cancel)—This is a general-purpose control character which is used to indicate that the material in the previous transmission is to be disregarded. The amount of material is decided by the user.

CR (Carriage Return, or Return)—This is a format effector control character which moves the carriage, print head, or cursor on a terminal back to beginning of line. On most terminals, the RETURN key causes both a CR and a LF (Line Feed).

DC1-DC4 (Device Controls)—These are general-purpose control characters which are used to control the user's terminal or similar devices. There are no standard functions assigned, except that DC4 frequently means "stop." The CCITT suggests a number of possible assignments; in general, they prefer using the first two controls for "on", and the last two for "off," and DC2 and DC4 to refer to the more important device. In some systems, these codes are labeled XON, TAPE, XOFF, and TAPE, respectively. X means "transmitter," and TAPE and TAPE means "tape on" and "tape off." These labels are found on the keytops of some terminals.

DEL (Delete)—This is a general-purpose control character which is used to delete a character. It is called RUBOUT on some terminals. It is not strictly a control character since it is not grouped with the other ASCII control characters. The DEL function has binary all-ones bit pattern ($1111\ 1111_2$), and the reason for this is historic: the only way to erase a bit pattern punched into paper tape was to punch out all the holes so that the resulting pattern was equivalent to a null. ASCII still considers DEL equivalent to a null, although many operating systems use it to erase the preceding character.

DLE (Data Link Escape)—This communication control character uses a special type of escape sequence specifically used for controlling the data line and transmission facilities.

EM (End of Medium)—This general-purpose control character is used to indicate the end of paper tape (or other storage medium) or that this is the end of the material on the medium.

ENQ (Enquiry)—This communication control character is usually used for requesting identification or status information. In some systems, this code is called WRU - "Who are you?"

EOT (End Of Transmission)—This communication control character is used to mark the end of transmission after one or more messages.

ESC (Escape)—This general-purpose character marks the beginning of an escape sequence. An escape sequence consists of a series of codes, which as a group, have a special meaning, usually a control function. On some terminals, ESC is called ALT MODE.

ETB (End of Transmission Block)—This communication control character is used when it is desired to break up a long message into blocks. ETB is used to mark block boundaries. The blocks usually have nothing to do with the format of the message being transmitted.

ETX (End of Text)—This communication control character is used to mark the end of a text. See SOH. This code was originally called EOM, "End Of Message," and may be labeled as such on some terminals.

FF (Form Feed)—This format effector control character causes the carraige, print wheel, or cursor to advance to the top of next page.

FS, GS, RS, US (File, Group, Record and Unit Separator)—These are a set of information separator control characters that are used for delimiting portions of information. There is no standard usage, except that FS is expected to refer to the largest division, and US to the smallest.

HT (Horizontal Tab)—This is a format effector control character which is used to tab the carriage, print wheel, or cursor to the next predetermined stop on the same line. It is usually up to the user to decide where the horizontal tab stops are to be positioned.

LF (Line Feed)—This is a format effector control character which moves the carriage, print head, or cursor down one line. Most systems combine CR (carriage return) with LF, and the new line is then called New Line (NL).

NAK (Negative Acknowledgement)—This is a communications control character which is used to indicate "no" in answer to various queries. It is sometimes defined as "I received your last transmission, but it had errors and I'm awaiting a retransmission."

NUL (Null)—This is a general-purpose control character which is used mainly as a space filler. See also **SYN**.

SI (Shift In)—This is a general-purpose control character which is used after a SO code to indicate that codes revert to normal ASCII meaning.

SO (Shift Out)—This is a general-purpose control character used to indicate that bit patterns to follow will have meanings outside the standard ASCII set and will continue to do so until SI is entered.

SOH (Start Of Heading)—This is a communication control character which is used to mark the beginning of a heading, when headings are used in messages along with text. Headings

usually state the name and location of an addressee. This code was originally called SOM, "Start Of Message."

STX (Start of Text)—This is a communication control character which is used as a marker for beginning of text, and end of heading (if used). Was originally called EOA, "End Of Address."

SUB (Substitute)—This is a general-purpose control character which is used to indicate a character that is to take the place of a character known to be wrong.

SYN (Synchronous Idle)—This is a communication control character used by some high-speed data communications systems which use synchronized clocks at the transmitter and receiver ends. During idle periods, when there are no bit patterns to enable the receiver clock to track the transmitter, the receiver may drift out of sync. Every transmission following an idle period is therefore replaced by three or four SYN characters. The SYN code has a bit pattern that enables the receiver not only to lock onto the transmitter clock, but also to determine the beginning and end points of each character. SYN characters may also be used to fill short idle periods in order to maintain synchronization, hence the name.

VT (Vertical Tab)—This is a format effector control character which is used to tab the carriage, print head, or cursor to the next predetermined stop (usually a line).

APPENDIX G

CP/M Prompts

X>	CP/M waiting for a command; drive X is the currently logged drive.
NX>	MP/M waiting for a command; drive X is the currently logged drive; and the current user number is N.
*	PIP or ED waiting for a command. This prompt is also used by Microsoft BASIC-80, EDIT, FORTRAN-80, COBOL-80, and Pascal when waiting for a command.
nnnn:*	ED waiting for a command; the "*" prompt is at line number nnnn.
-	DDT waiting for a command.

In the following space write in the prompts of other programs that you use.

APPENDIX H

CP/M Error Messages

Through the course of using CP/M and its various commands, it is unusual not to encounter error messages of some kind. As with all computers, errors do occur from time to time. However, in the case of CP/M, the error messages are often cryptic in meaning, and are, for the most part, very poorly documented, or not documented at all. We have also found that many of the messages are inappropriate when displayed for many types of errors (hardware and software related). For these reasons, we have decided to list any errors that you might encounter while using the CP/M.

BDOS ERR ON X: BAD SECTOR

This message will appear when the disk controller hardware of your system has detected an error condition while issuing, for example, the command string "STAT *.*," or any other STAT command sequence which causes CP/M to read the disk or write to the disk. The error is usually due to malfunctioning disk controller hardware, or an extremely worn disk. This error may also be caused by slight differences in the format of the disk, if the disk was formatted on another system with different disk controller hardware. Even though most CP/M-compatible disk controllers are considered as IBM-standard, you may find differences in recording formats, even though the CP/M formatting program used was the same as yours.

When this message is received, the cursor will remain on the same line as the message. It is suggested that a Warm Boot (CTRL/C) be issued to recover, whereupon the cursor will move to the next line and the original CP/M prompt will reappear. If the RETURN key is used instead, CP/M will ignore the error and resume with the command originally issued. It is suggested that this not be done as the format integrity of the disk may be damaged. Once recovered, the disk should either be reformatted or the contents of the disk copied to another disk on your system if you have a copy program that will ignore or translate the errors. You should not attempt to use PIP, as it is more than likely the same error message will occur.

BDOS ERR ON X: FILE R/O

This error message exists only in CP/M Version 2.0 or above. The message will appear, if you use the ERA command to erase a file that has the R/O (Read/Only) file-protection attribute assigned. The error will also occur, if any attempt is made to the write to the file, such as opening and closing the file using a text editor or word processor. Any circumstance involving the modification of the contents of the R/O file will cause this error message. When the error occurs, the cursor will be on the same line as the error message. Press the RETURN key or perform a Warm Boot (CTRL/C). The cursor will move to the next line where the original CP/M prompt (A>, B>, C>, etc.) will reappear. The file can then be changed to Read/Write (R/W) using STAT, if it is desired that the file have its contents modified or that it be erased.

BDOS ERROR ON X: READ ONLY

This error message exists only in CP/M Version 1.4, 2.0, or above. The message will appear when an attempt is made to write to a disk that has been designated as read-only (R/O) by a STAT command sequence or by CP/M's BDOS. A disk is designated as R/O when the STAT command sequence "STAT X:=$R/O" is issued, or when the BDOS section of CP/M has been set up so that it issues the write-protect when certain situations are encountered. When the message is presented, the cursor will remain on the same line as the error message. Press the RETURN key or any key to perform a Warm Boot, whereupon the cursor will move to the next line where the original CP/M prompt will reappear, and the disk will be set to Read/Write (R/W).

BDOS ERR ON X: SELECT

This message will appear if a nonexistent drive was specified (X:). When the error occurs, the cursor will be on the same line as the error message. Any input from the console device (RETURN key or CTRL/C) will cause a Warm Boot. The cursor will be move to the next line where the original CP/M prompt (A>, B>, C>, etc.) will reappear.

This error message will also occur if the disk drive specified in a command sequence does not have a disk inserted in it. The system may "die" when this occurs. To recover, insert a disk in the drive (drive "X:") and then perform a Warm Boot (CTRL/C). The cursor will move to the next line, where the original CP/M prompt will reappear.

FILE NOT FOUND

This message will appear when the file or files specified in a command sequence cannot be found on the disk, or if there are no files on the disk. This message will also appear when an error was made in the specification of a disk drive in the command sequence, such as omitting the colon after the disk specifier. After the message is given, the cursor automatically moves to the next line and is positioned at the original CP/M prompt.

INVALID ASSIGNMENT

This message will appear if a logical or physical device name is misspelled, or if the device does not exist, when assigning physical devices to logical devices using the STAT command. When the error occurs, the cursor will automatically move to the next line, where the original CP/M prompt will reappear. To display the menu of all allowed logical-to-physical device assignments, type the command sequence "STAT VAL:." To display the existing assignments, type in "STAT DEV:."

INVALID DISK ASSIGNMENT

This error message exists only in CP/M Version 2.0 or above. The message will appear if an error was made while assigning protection status to a disk. Allowed command sequences are: "STAT X:=R/O" or "STAT X:=R/W." If anything other than "R/O" or "R/W" is typed after "STAT X:=" then this message will be given. When the message is given, the cursor automatically moves to the next line where the original CP/M prompt will reappear.

INVALID FILE INDICATOR

This error message only exists in CP/M Version 2.0 or above. The message will appear when an error was made while assigning STAT command file attributes. Allowed attributes are: $R/W, $R/O, $DIR, and $SYS. If any other STAT attribute is typed, this message will appear. When the message occurs, the cursor will automatically move to the next line where the original CP/M prompt will reappear.

APPENDIX I

Command Summary

RESIDENT (DIRECT BUILT-IN) COMMANDS

DIR (Directory):

DIR	List directory of logged disk.
DIR *.*	List all files on logged disk.
DIR *.ext	List all files with extension ".ext".
DIR filename.*	List all files called "filename".
DIR filename.ext	List specific file only.

REN (Rename):

REN newname.ext = oldname.ext

Rename file "oldname.ext" to "newname.ext".

ERA (Erase):

ERA filename.ext	Erase specific file.
ERA filename.*	Erase all files called "filename".
ERA *.ext	Erase all files with extension ".ext".
ERA *.*	Erase all files on disk.

TYPE (Display Text File):

TYPE filename.ext	Display contents of "filename.ext" on console device.

USER (Select Disk User Area):

USER x	Select disk User area "x" (where "x" = 0 through 15).

SAVE (Save memory contents in file on disk):

SAVE n filename.COM Save "n" memory pages (page = 256 bytes) in file "filename.COM" on disk.

NOTE: An optional disk drive name (A:, B:, etc.) can be entered before any of the file name command strings (specific file names or wildcards) with the above commands, except the USER command, to specify a disk drive other than the logged one. Wildcards can consist of the "*" character (multiple-character substitution), or the "?" character (single character substitution). Filenames may consist of up to eight characters, and extensions or file types may have up to three.

CONTROL CHARACTER COMMANDS

Control character commands are entered while in the main CP/M command mode, and sometimes while running certain programs, by simultaneously pressing the CTRL key and the indicated key on the console device.

CTRL/C	Warm Boot.
CTRL/M	Carriage Return.
CTRL/J	Line Feed.
CTRL/H	Backspace.
DELETE or **RUBOUT**	Delete character to left.
CTRL/U	Cancel currently entered command string.
CTRL/X	Cancel currently entered command string and erase it.
CTRL/E	Permit long command strings to overflow onto next line.
CTRL/R	Retype an edited command string on next line, and omit all echoed characters deleted with DELETE or RUBOUT key.
CTRL/S	Stop/Restart scrolling of text on console device.
CTRL/P	Echo all console device input and output on the LST: printer device.

TRANSIENT COMMANDS

SYSGEN (Generate or Copy CP/M System Image):

SYSGEN	Initiate the SYSGEN dialog.
SYSGEN filename.ext	Retrieve system image stored in "filename.ext" on disk.

MOVCPM (Move or Relocate CP/M System Image):

MOVCPM	Move system image using all available memory and transfer control to it when finished.
MOVCPM nn	Move system image using "nn" amount of memory (in kilobytes) and transfer control to it when finished.
MOVCPM * *	Move system image using all available memory, do not transfer control to it when finished, and prepare system image for saving in a file on a disk.
MOVCPM nn *	Move system image using "nn" amount of memory (in kilobytes), do not transfer control to it when finished, and prepare system image for saving in a file on a disk.

NOTE: When using the last two forms of the MOVCPM command, the system image will be left in memory, and the SAVE command is subsequently used to save it on a disk. The SYSGEN command can then be used to generate a bootable system image.

STAT (Disk and File Status/Statistics Command):

STAT	Display amount of unused space on all disks.

NOTE: This command will normally only display unused space of the logged drive. However, unused space of all other drives with disks will also be displayed, if each had a previous read operation. A disk drive name (A:, B:, etc.) can follow the STAT command to display unused space of a specific disk drive.

STAT filename.ext	Display size in kilobytes of file "filename.ext".
STAT filename.*	Display sizes in kilobytes of all files called "filename".
STAT *.ext	Display sizes in kilobytes of all files with the extension ".ext".
STAT *.*	Display sizes of all files on the disk.

NOTE: Optionally, the "$S" parameter can append the preceding command strings for more detailed size information.

STAT filename.ext $R/O STAT filename.* $R/O STAT *.ext $R/O STAT *.* $R/O	Set file(s) to Read-Only.
STAT filename.ext $R/W STAT filename.* $R/W STAT *.ext $R/W STAT *.* $R/W	Set file(s) to Read/Write.
STAT x:=R/O	Temporarily set disk drive "x:" to Read-Only.
STAT filename.ext $SYS STAT filename.* $SYS STAT *.ext $SYS STAT *.* $SYS	Set file(s) to System file(s) so that they will not display using the DIR command.
STAT filename.ext $DIR STAT filename.* $DIR STAT *.ext $DIR STAT *.* $DIR	Set file(s) to Directory file(s) so that they can be displayed using the DIR command.
STAT USR:	Display current USER number and number of established USER areas.
STAT DSK:	Display disk drive characteristics of all disks inserted in system.
STAT x:DSK:	Display disk drive characteristics of disk drive "x:".
STAT DEV:	Display current physical-to-logical device assignments.

STAT VAL: — Display information on allowed device assignments and other STAT commands.

STAT log:=phy: — Assign physical device "phy:" to logical device "log:", where device assignments can be:

LOGICAL	PHYSICAL
CON: =	TTY:, CRT:, BAT:, or UC1:
RDR: =	TTY:, PTR:, UR1:, or UR2:
PUN: =	TTY:, PTP:, UP1:, or UP2:
LST: =	TTY:, CRT:, LPT:, or UL1:

PIP (Peripheral Transfer Program):

PIP — Load PIP to perform individual copy operations.

PIP y:=x:filename.ext
PIP y:=x:filename.*
PIP y:=x:*.ext
PIP y:=x:*.* — Copy file(s) from disk drive "x:" to disk drive "y:".

PIP y:newfile.ext=x:oldfile.ext — Copy file "oldfile.ext" on disk drive "x:" to disk drive "y:" and rename the copied file to "newfile.ext".

PIP y:newfile.ext=x:oldfile1.ext,oldfile2.ext,oldfile3.ext — Copy all files on disk drive "x:" in the order shown, and place all copies in the same order (concatenate) into file "newfile.ext" on disk drive "y:".

PIP CON:=x:filename.ext
PIP TTY:=x:filename.ext
PIP CRT:=x:filename.ext
PIP UC1:=x:filename.ext — Copy file "filename.ext" on disk drive "x:" to the currently assigned console device.

PIP PUN:=x:filename.ext
PIP PTP:=x:filename.ext
PIP UP1:=x:filename.ext
PIP UP2:=x:filename.ext
PIP OUT:=x:filename.ext — Copy file "filename.ext" on disk drive "x:" to an output-only device.

PIP LST:=x:filename.ext PIP LPT:=x:filename.ext PIP UL1:=x:filename.ext PIP PRN:=x:filename.ext	Copy file "filename.ext" on disk drive "x:" to the currently assigned listing device.
PIP x:filename.ext=CON: PIP x:filename.ext=TTY: PIP x:filename.ext=CRT: PIP x:filename.ext=UC1:	Copy all input from currently assigned console device to file "filename.ext" on disk drive "x:", until a CTRL/Z is entered.
PIP x:filename.ext=RDR: PIP x:filename.ext=PTR: PIP x:filename.ext=UR1: PIP x:filename.ext=UR2: PIP x:filename.ext=INP:	Copy all input from an input-only device to file "filename.ext" on disk drive "x:", until a CTRL/Z is entered on the console device.
PIP PUN:=NUL: PIP PUN:=NUL:, x:filename.ext PIP PUN:=NUL:, x:filename.ext,NUL: PIP PUN:= x:filename.ext,NUL:	Specify null leaders and trailers while copying file "filename.ext" on disk drive "x:" to an output-only device.
PIP PUN:= x:filename.ext,EOF:	Specify an end-of-file marker while copying file "filename.ext" on disk drive "x:" to an output-only device.

OPTIONAL PIP PARAMETERS:

[V] Verify accuracy of copied data by comparing destination file with source file.

[E] Echo and display all text on the console device as it is being copied.

[Sstring^Z]
Begin copying a file after "string" string of text is encountered (terminate "string" with a CTRL/Z command).

[Qstring^Z]
Quit copying a file after "string" string of text is encountered (terminate "string" with a CTRL/Z command).

[L] Convert all uppercase characters to lower case during a copy operation.

[U] Convert all lowercase characters to upper case during a copy operation.

[Dn] Delete all characters on each line of text after column "n" during a copy operation.

[Tn] Convert all ASCII tab characters to "n" number of spaces during a copy operation.

[F] Delete all form-feed characters during a copy operation.

[Pn] Insert form-feed characters after each "n" number of lines during a copy operation.

[N] Add sequential line numbers without leading zeros to text during a copy operation.

[N2] Add sequential line numbers with leading zeros to text during a copy operation.

[R] Copy a "System" file (set by the STAT command).

[W] Copy over a "Read-Only" file (set by the STAT command) that has the same file name as the specified destination file name.

[Gn] Copy a file or files from USER disk area number "n" (from 0 to 15), to the current USER disk area number.

[O] Interpret file or files being copied as non-ASCII object/machine code file(s).

[B] Specify "Block Mode" format during a copy operation.

[H] Specify Intel Hexadecimal format during a data transfer operation between input-only and output-only devices.

[I] Ignore any Null Records in Intel Hexadecimal format during a copy operation.

[Z] Set the parity bit of all ASCII bytes to zero during a copy operation.

ED (Text Editor Program):

ED filename.ext	Load ED into memory and open file "filename.ext" for creation and/or for editing.
ED Commands:	
nA 0A #A	Append "n" number lines of text from source file to editing buffer, or all text until editing buffer is half full ("0"), or all lines until editing buffer is full ("#").
I Itext Itext^Z	Insert text until a CTRL/Z command is given ("I"), insert text and remain in insert mode ("Itext"), or insert text and immediately exit from insert mode ("Itext^Z").
U -U	Engage ("U") or disengage ("-U") character translation to upper case.
nC -nC	Move CP "n" number of characters to right ("nC"), or to left ("-nC").
nL -nL	Move CP "n" number of lines forward ("nL"), or back ("-nL").
V -V	Display ("V") or suppress displaying ("-V") of line numbers.
n: B -B	Move CP to line number "n:". Move the CP to the beginning ("B") or the end ("-B") of the editing buffer.
T 0T 0TT	Display text on current line before the CP position ("T"), after the CP ("0T"), or the complete line ("0TT").
nT -nT #T -#T	Display "n" number of lines after the CP position ("nT"), before CP ("-nT"); or all lines after ("#T") or before ("-#T") CP position.
nP -nP	Move CP "n" number of pages forward ("nP") or back ("-nP") and display text.
nD -nD	Delete "n" number of characters on a line after ("nD") or before ("-nD") the CP position.

nK -nK	Kill (delete) "n" number of lines after ("nK") or before ("-nK") the CP position.
Ftext^Z nFtext^Z	Find "text" string of text in the editing buffer ("Ftext^Z"), or find the "n"th occurrence of "text" ("nFtext^Z"). Terminate "text" with the CTRL/Z command.
Ntext^Z nNtext^Z	Same as above, only perform automatic append operations from the source file as necessary.
Sfind^Zreplace^Z nSfind^Zreplace^Z	Search for "find" text string, and replace it with "replace" text string. Alternately, perform operation "n" number of times. "^Z" means use the CTRL/Z command.
Jfind^Zinsert^Zend^Z nJfind^Zinsert^Zend^Z	Search for "find" text string, and juxtaposition it by inserting "insert" text after "find", and delete all text thereafter until "end" text string is found. Alternately, perform operation "n" number of times. "^Z" means use the CTRL/Z command.
nMcommands	Specify the macro repetition of a subsequent group of commands ("commands"), "n" number of times.
0V	Verify total editing buffer usage and free space.
W nW	Write the current line ("W") from the editing buffer to the temporary output 0W file, or "n" number of lines ("nW"). #W Alternately, write all lines until the editing buffer is half empty ("0W"), or all lines until buffer is completely empty ("#W").
nX	Write "n" number of lines from the editing buffer to a temporary library file.
R	Read the entire contents of the temporary library file (created by "nX") into the editing buffer.

Rfilename	Read the contents of the library file "filename" (with the extension .LIB) into the editing buffer.
H	Save all editing in the final output file without exiting ED.
E	Exit ED (return to CP/M) and save all editing in the final output file.
O	Erase all editing in the editing buffer and final output file without exiting ED.
Q	Exit ED (return to CP/M) and erase all editing in the editing buffer and the final output file.

SUBMIT (Submit Several Commands to CP/M):

SUBMIT filename a b c — Load SUBMIT command and submit commands in "filename" (with extension .SUB), and make optional text substitutions "a" for $1, "b" for $2, "c" for $3, etc.

XSUB (Extended SUBMIT Program):

This program is placed in SUBMIT files to permit the submitting of commands to other programs loaded by SUBMIT. This command is not entered by itself on the command line.

ASM (CP/M 8080 Microprocessor Assembler Program):

ASM filename.123 — Assemble "filename" with the extension .ASM, with optional parameters "123" as follows:

"1" position is disk drive name where the source file is stored.

"2" position is the disk drive name where the output .HEX file is to be stored, or "Z" to suppress output of the file.

"3" position is the disk drive name where the output .PRN file is to be stored, or "Z" to suppress its output, or "X" if listing to be sent to the console device.

DDT (Dynamic Debugging Tool):

DDT filename.ext	Load DDT into memory and also place file "filename.ext" into memory for examination.
DDT Commands:	
D Dstart Dstart,end	Display next 192 locations of memory in hexadecimal format from current location ("D"), or begin displaying from location "start" ("Dstart"). Alternately, display from location "start" until location "end" is reached ("Dstart,end").
Fstart,end,data	Fill all memory locations between locations "start" and "end", with the byte "data" (in hexadecimal).
Ifilename.ext	Insert and establish the File Control Block (FCB) of "filename.ext" so that file can be subsequently read into memory with the "R" command.
R Rstart	Read the contents of "filename.ext" into memory starting at the current location, as specified by the "I" command. Alternately, insert into memory beginning from location "start" ("Rstart").
L Lstart Lstart,end	Disassemble and display the contents the next 192 locations of memory in assembly code starting at the current location ("L"), or beginning from location "start" ("Lstart"). Alternately, specify the ending location "end" ("Lstart,end").
Sstart	Display and optionally change the hexadecimal contents of location "start", and continue on to subsequent locations until "." is entered.
Astart	Insert assembly language statements starting at location "start", and continue with subsequent locations until "." is entered.
Mstart1,end1,start2	Copy the contents of a specific memory block between "start1" and "end1" to another block in memory starting at location "start2".
G Gstart	Execute all of program in memory with no breakpoints ("G"), starting at location "start" un-

Gstart,end Gstart,end1,end2 G,end G,end1,end2	til the last location ("Gstart"), or by specifying breakpoint (location) "end" ("Gstart,end"). Alternately, specify two breakpoints after a starting location ("Gstart,end1,end2"), or from the current location ("G,end" or "G,end1,end2").
Unumber	Execute "number" number of program instructions in memory and then stop to display the contents of the CPU registers.
Tnumber	Trace the execution of "number" number of program executions, displaying the contents of the CPU registers for each instruction.
X Xregister	Display the current contents of all the CPU registers ("X"), or of the specific register "register" ("Xregister").

DUMP (Dump contents of machine code file in hexadecimal format):

DUMP filename.ext	Dump the contents of file "filename.ext" in hexadecimal format.

LOAD (Load a .HEX file to create a .COM file):

LOAD filename	Read file "filename" with the extension .HEX created with the ASM command, and create "filename" with the extension .COM.

Index

A

"A" (Append) command, 192
A (Assemble) command, 256
Acc (Access), 98
Accessing
  commands and files, 45-50
  DIR, 68-70
  disk formatting program, 278-284
  disks and files, and use of passwords, 307-308
  ERA, 75-76
  files, 47-49
  MOVCPM, 296-297
  STAT, 93-94
  SYSGEN, 286-293
  USER, 85-89
Accessing and using
  PIP, 119
  submit, 204-207
Add sequential line numbers, 150-151
Allocation units, 97
Alpha-numeric characters, 396
Appending
  portions or all of text from source file until editing buffer is half or completely full, 166-167
  specific lines of text from source file into editing buffer, 164-166
Ascending order, 238
ASCII cross reference, 25, 395-402
ASM (Assembler) command, 34, 219-236
  and CP/M console error messages, 226-227
  directives, 233-236
  source program
    constants and expressions, 230-233
    format, 229-230
.ASM, 55
  file, 224
ASM.COM (Assembler), 46
Assembling a file on the default drive, 222-226
Assigning
  logical devices, 308-309
  physical devices, 308-309
    to logical names, 115-116
*(used to represent a group of characters), 50

B

"B" Command, 177
Backing up the CP/M system disk, 41-42
Backspace, 168
.BAK, 55
.BAS, 56
BASCOM, 347, 357-358
BASIC-80, 347, 355-356
BAT:—Batch processing device, 114, 133
BDOS, 34
Begin copying a file, 140
Bibliography, annotated, 25, 375-389
BIOS, 34, 37
"Block Mode" parameter [B], 155
Book
  how to use, 23-26
  purpose, 23
Booting, 39
  CP/M
    Plus system, 306-307
    system, 38-41
Bootstrap loader, 38-39
"BREAK" message, 201
Built-in (direct) commands, 31-32, 46-47, 58, 310-311

Bytes, 97
  read, 238

C

Cannot
  close file, 227
  open library file after "R" command, 201
Carat, 163
CBASIC2, 348, 359-360
CB-80, 348, 360-362
Character
  deletion commands, 53-54
  Pointer (CP), 162-164
Characters
  alpha-numeric, 396
  control, 396
  definitions, 400-402
  format effector, 396
  information separator, 396
  special graphic, 396
.COB, 56
COBOL-80, 333, 348, 366-367
Cold boot, 39-40, 83
Cold-boot start-up profile file, 307
.COM, 47, 56
Combining CP movement commands, 187-189
Command
  descriptions, 245-267
  summary, 26, 409-420
Commands
  built-in, 31-32, 46-47, 58, 409-410
  control character, 410
  direct, 31-32, 45, 409-410
  resident, 32, 45, 409-410
  transient, 32-35, 46-47, 59-61, 409-410
Commonly used terms, glossary of, 391-394
Comparison of CP/M versions, 57
Compatible languages, CP/M, 347-367
CON:—Console device, 112
Concurrent CP/M-86, 31
  operating system, 372
Console
  Command Processor (CCP), 31-32
  device, 30
Constants
  numeric 231-232
    binary, 231
    decimal, 231
    hexadecimal, 231
    octal, 232
  string, 232
Control
  character commands, 52-55, 61, 312-313, 410
  characters, 396
  key (CTRL), 52
  programs, 26
Convert all
  ASCII tab characters, 147-148
  lower-case characters to upper-case, 145-146
  upper-case characters to lower-case, 143-144
COPSYS, 315
Copy
  contents of specific memory block, 258-259
  files from other disk user areas, 153
  from
    input-only device into file on a disk, 135
    source "system" file, 152
Copying
  (concatenating) several files into one file, 118
  CP/M system image from file, 287-289, 292-293
  data between standard CP/M devices and special PIP devices, 118
  file on same disk drive, 118
  files between
    different disk drives, 118, 120-133
    disk drive and a special disk device, 118
  input on console device into file on disk, 134-135
  nonexistent files, 125
  protected files, 126
  several files into one file, 128-130
  system files, 127
Copying to
  console destination device (video terminal or teletypewriter), 131
  list devices (line printer or other device), 132-133
  output only devices (paper tape punch or other device), 131-132
CP/M
  command(s)
    overviews, 310-322
    summary of, 31-35
  error messages, 405-407
  Librarian utility, 336-340
  operating system, 28-30
  prompts, 26, 403
  sign-on messages, 40-41
  version 3.0, overview, 25
CP/M assembly language
  crossreference

CP/M assembly language–cont
crossreference
program access, 342-343
summary, 341-342
utility
program access, 340-341
summary, 340
Link Loader utility, 330
CP/M Plus (Ver. 3.0) Overview, 305-322
CP/M-based operating systems, other, 30-31
CP/M-compatible languages, 25
CP/M-related operating systems, 25, 369-373
CP/M-86 operating system, 371-372
CP/M-68K, 31
CP/NET network operating system, 30, 373
Create
indexed library file, 337
library file, 337
Creating
or changing USER areas, 86-87, 88
USER area for the first time, 87-88
CR (carriage return), 168
CREF-80, 324
utility, 341-343
.CRF, 56
CRT device, 30
CRT:–High-Speed Console (cathode ray tube) device, 114
CTRL key, 83
CTRL/E, 54, 61
CTRL/H and backspace, 53-54
CTRL/J, 53
CTRL/L (insert a form feed) character, 169
CTRL/M, 53
CTRL/R (Reprint), 55, 61, 169
CTRL/S, 55, 61
CTRL/U (delete line of text), 54, 61, 169
CTRL/X (delete line of text), 54, 61, 169
Cursor, 165

D

D (Display) command, 181-182, 245
.DAT, 56
DATE, 315
Date stamping, time and, 309
DB (Define Byte), 234-235
D–data error, 228
DDT (Dynamic Debugging Tool), 34, 46, 135, 241-267, 326
Default drive, 43
Delete [D], 146
DELETE (RUBOUT), 54, 168-169, 308
Delete
(kill) one or several lines before or after CP, 182-183
one or several characters on line before or after CP, 181-182
Delete all characters on each line of text after specified column, 146-147
form-feed characters during copy operation, 148-149
Description
and program access (ED), 158-162
of MOVCPM command variations, 297-303
of STAT command variations, 94-95
DESPOOL, 323, 325-326
Device, 315-316
console, 30
input-only, 30
listing, 30
output-only, 30
DIR (Directory) command, 32, 45-46, 58, 67-70, 313
Direct commands, 45
"Directory" (DIR (nonsystem)), 152
DIRSYS, 313
Digital Research, 30-31
LIB-80
general, 336
program access, 337
summary, 336-337
LINK-80
program access, 331-333
summary, 330
Disassembling and displaying portions of memory with assembly code, 253-255
Disk
density, 279-280
drive chaining, 308
hard, 93
hard-sectored, 93
soft-sectored, 93
specifications, 62
DISKINIT.SUB, 205-208
Disks
protecting, 92
Display
and optionally change contents of location in memory, 255-256
contents of CPU registers, 264-266
control commands, 54-55
or echo all text, 138-139
or suppress displaying line numbers, 175-176

Display–cont
  text before or after CP, 177-180
Displaying
  allowed device names, 112-115
  contents of
    file using TYPE, 81-83
    portions of memory, 245-248
  complete directory of disk, 68-69
  directory of specific file or group of files, 69-70
  disk drive characteristics, 106
  device assignments, 111-112
  protection status, 95-96
  status
    all files, 98-99
    specific files, 96-98
  unused disk space, 91, 94-95
DOC, 69-70
.DOC, 56
.$$$, 57
DOS (Disk Operating System), 26, 28-29
Double
  checking files, 105
  density, 93
  -sided, double-density floppy disk, 108-109
DS (Define Storage), 234-235
DUMP command, 34, 273-275, 316
DUMP.COM, 46
DW (Define Word), 234-235

E

"E" (Exit) command, 197-198
E–expression error, 228
Echo [E], 139
ED (EDitor) command, 34, 157-200, 216-217, 316
  character pointer (CP), 162-164
  description of, 164-200
  error messages, 200-201
ED.COM (Text Editor), 46
Editing
  a file, 160-161
  buffer full, 201
8080 to 8086 Assembly language Translator utility
  program access, 343-345
  summary, 343
8080/8085 Symbolic Instruction Debugger utility (SID), 326-327
Engage or disengage character translation to upper case, 172
EOF: (End-of-file marker), 136
EQU (Equate) assembler, 233-234
ERA (Erase) command, 32, 45, 58, 75-79, 313-314
Erase, 45
  a read-only file having same name as destination file name, 152-153
  all editing in editing buffer and final file without exiting ED, 199
Erasing
  all files, 78-79
  groups of files, 77
  single file, 75-76
Error messages, 26
  CP/M, 405-407
Execute a number of program instructions in memory, 261-263
Exit ED (return to CP/M) and save all editing in final file, 198-199
Expressions
  arithmetic operators, 232
  logical operators, 233
Ext (Extent), 98

F

F (Fill) command, 248
"F" (Find command, 183
File
  Control Block (FCB), 249
  editing, 160-161
  management, 27
  type (typ), 47
Filename, 47
Files
  double checking, 105
  hiding, 92, 103-105
  listing, 92
  protecting, 92, 100-101
Filespec, 47
Filling the memory with constant data, 248-249
Find text string
  and substitute it with specified text, 185-186
  in editing buffer, 183-184
    and juxtaposition it with specified text, 186-187
    and perform append and write operations, 184-185
First address, 238
Floppy disk format standard, 29
.FOR, 56
Form-feed [F], 148

FORMAT command, 34-35, 277-284, 316-317
Format effector characters, 396
Formatting a soft-sectored floppy disk, 280-283
FORTRAN-80, 333, 348, 365-366

G

GENCOM, 317
GENCPM, 317
GET, 317
G "Go" command, 259-261
[G] parameter, 153
Glossary of commonly used terms, 25, 391-394
Graphic characters, special, 396

H

"H" Command, 197
H (Hex) command, 266
Hard disk, 93
Hard-sectored disk, 93
HELP, 317-318
.HEX, 56
file, 224
Hexadecimal parameter [H], 156
HEXCOM, 318
Hiding files, 92, 103-105
How to use this book, 23-26

I

I (insert) command, 167, 249
IBM 3740, 28-29
IF and ENDIF (End IF) directives, 234
Ignore
any null records in Intel Hexadecimal format, 156
parameter [I], 156
Improper FORMAT, 124
Information separator characters, 396
INITDIR, 318
Initiating
ED, 160
single-line PIP commands, 127-128
Input or source, 133
Input/output (I/O), 27
Input-only device, 30
Insert
assembly language statements, 256-258
form-feed characters after each specific number of lines, 149
Inserting
and establishing file control block for referencing, 249-251
comments in .SUB files, 209-210
text into the editing buffer, 167-172
.INT, 56
Intel
8080, 30
8085, 28, 30
8086, 31
Interpret files being copied as NON-ASCII, 153-155
Introduction, 23-35

J

"J" (Juxtaposition) command, 186

K

"K" (Kill) command, 182-183
Kilobyte(s), 40
drive capacity, 109

L

L–label error, 228
L (List) command, 253
Labels, 230
Languages, CP/M compatible, 347-367
Last address, 238
LF (line feed), 168
LIB, 318
.LIB, 56
LIB-80, 324
Line
deletion commands, 54
numbers, 229
termination commands, 53
LINK, 318
LINK-80, 324
List module names, 337
Listing
device, 30
files, 92
LOAD Command, 34, 237-239
LOAD.COM, 46
Loading, 39
DDT into memory
by specifying a file, 244-245
without specifying a file, 243-244
PIP and performing copy operations, 120-127

Logged drive, 43
Logical
  and physical devices, 308-309
  Extent, 98
  (symbolic) devices, 111-112
Lower [L], 143
LPT:–Line Printer device, 115
.LST, 56
LST:–Output List device, 112

M

M (Move) command, 258
MAC, 318-319, 347, 348-350
.MAC, 56
MACRO-80, 333, 347, 352-354
Macro repetition of command groupings, 189-191
Memory Management, 42
Microcomputer operating systems, 26-28
Microsoft LIB-80
  program access, 338-340
  summary, 328
Microsoft LINK-80
  program access, 334-336
  summary, 333
Mnemonics, 230
Monitor program, 39
Motorola 68000, 31
MOVCPM (Move CP/M) command, 35, 46, 295-303
Move CP backward or forward one or several
  characters on a line, 173-174
  lines, 174-175
Move CP
  one or several pages and display text, 180-181
  to beginning or end of editing buffer, 177
  to specific line number, 176
Move CP/M system image using
  all memory and transfer control, 297
  specified amount of memory, 302-303
Move CP/M using
  all of memory and prepare system image for saving in a file, 300-302
  specified memory and transfer control, 298-299
MP/M-86, 31
MP/M operating systems, 30, 372
MP/M-II operating system, 370-371
.MSG, 56

N

"N" Command, 184
National 16000, 31
"nC" Command, 173-175
N-feature not yet implemented, 228
New or changed BDOS functions, 63-65
No directory space, 227
No source file present, 226
NONE, 308
"nP" Command, 180
NUL, 136
Number [N] and [N2], 150

O

"O" Command, 199
O–overflow error, 228
Object parameter [O], 154
128 Byte record capacity, 109
Operands, 230
Operating
  environment, 305-306
  system(s), 26
    CP/M-Related, 369-373
Optional
  PIP parameters, 118-119, 137-156
  transient utilities, 25, 323-345
ORG (Origin) assembler, 233
OS (operating system), 26
Other formatting programs, 283-284
Ouput file write error, 227
Output-only device, 30
Over-writing existing files during a copy operation, 125-126
.OVR, 56

P

P–phase error, 228
Pagination [P], 149
Parity bit, 395
.PAS, 57
Pascal/MT+, 348, 362-365
Password, 28
PATCH, 319
Pausing display of
  directory, 70
  text file, 82-83
Peripheral devices, 29
Peripherals, 93
Permanent storage, 41

Physical devices, logical and, 308-309
PIP (Peripheral Interchange Program) command, 34, 46, 87, 117-156, 319
PL/I-80, 347, 354-355
[Pn] parameter, 149
Printed on paper (teletype-writer), 131
Printer control commands, 55
.PRN, 57
  file, 225-226
Program
  access, 221, 237-239, 269-272, 273-275
    description, 242
    operation, 215-217
  counter (PC), 254
Programmable Read-Only Memory (PROM), 38
Prompt, 40
Protecting
  complete disks temporarily, 101-103
  files and disks, 92, 100-101
Protection status, 73, 91
PTP:–Paper-Tape Punch device, 115
PTR:–Paper Tape Reader device, 114
PUN:–Send-Information/Paper Tape Punch drive, 112, 131
Purpose of book, 23
PUT, 319-320

Q

"Q"(Quit) command, 199-200
? (used to represent a single character), 50, 52
Quit
  copying a file, 141-142
  ED (Return to CP/M) and erase all editing in editing buffer and final file buffer, 199-200
  [Q], 141

R

"R" (Read) command, 194-195, 251
Random Access Memory (RAM), 38
RDR:–Receive-Information/Paper Tape Reader device, 112
READ, 308
Read contents of
  library file into editing buffer, 196-197
  temporary file created by the nX command into editing buffer, 195-196
Read parameter [R], 152
Read/Only (R/O), 79
  file, 73, 125
Read/Write (R/W), 79
  file, 73, 125
Reading a file on a disk and loading it into memory, 251-253
Records
  /Block, 110
  /Extent, 110
  Written, 238
Recs (records), 97
REL, 57
REN (Rename) command, 32, 45, 58, 71-74, 313
Renaming a file on
  another drive, 73-74
  logged drive, 71-73
Repeat count too high, 201
Resident commands, 32, 45
  (direct built-in), 409-410
Return (CTRL/M), 169
Revealing
  current disk users, 93
  disk drive characteristics, 93
RMAC, 320, 347, 350-352
R–register error, 228
Rubout (Delete), 168-169

S

S (Set) command, 255
"S" (Substitute) command, 185
S–symbol error, 229
SAVE command, 32, 45, 58, 269-272, 314-315
Save all editing in final file without exiting ED, 197-198
Saving the CP/M system image in a file, 289-291
Screen (video terminal), 131
Scrolling control of STAT, 99
Sectors, 110
SET, 320
  directive, 234
SETDEF, 320-321
Set the parity bit of all ASCII bytes to zero, 156
SHOW, 321
SID, 321, 324
Single density, 93
Single-sided
  double-density floppy disk, 107-108
  single-density floppy disk, 107
Size, 97
Soft-sectored disk, 93

Source
  code program (.ASM) error messages, 227-229
  file
    name error, 226
    read error, 226
  input or, 133
Special
  control-character command, 35
  graphic characters, 396
  SUBMIT features, 211
Specify
  "Block Mode" format, 155
  Intel Hexadecimal format, 155-156
Specifying
  an end-of-file marker, 136
  leader and trailers while copying source files to output-only device, 136
Standard file types, 55-57
Start [S], 140
STAT (Status or Statistics) command, 33, 91-116
STAT.COM (Status or Statistics), 46, 48-49
Stopping the DIR command, 70
.SUB, 57
SUBMIT, 321-322
SUBMIT.COM, 46
SUBMIT command, 34, 203-210
.SYM, 57
Syntax, 190
SYSGEN command, 35, 40-41, 60, 285-293
SYSGEN.COM (System Generation), 46
"System"(SYS), 152

T

T (Trace) command, 263
"T"(Type) command, 177-180
Tab [T], 147
Terminating ED and saving the file, 161-162
Terms, glossary of commonly used, 25
.TEX, 57
TEXT (TXT), 51, 57
32 Byte directory entries, 109-110
Time and date stamping, 309
TPA, 252
Trace the execution of a number of program instruction, 263-264
Track(s), 110
  reserved, 111
Transient
  commands, 32-35, 46-47, 59-61, 311-312, 411-420
  utilities, optional, 323-345
TTY device, 30
TTY:—Slow-Speed Console (Teletype Device), 114
.TXT, 57
TYPE command, 32, 45, 58, 81-83, 314

U

U—undefined symbol, 229
U (Untrace) command, 261
"U"(upper case) command, 172
^U (editing command), 168
UC1:—User-Defined Console device, 114
UL1:—User-Defined List device #1, 115
Unrecognized command, 201
Up arrow, 163
UP1:—User-Defined Punch #1 device, 115
UP2:—User-Defined Punch #2 device, 115
Upper [U], 145
UR1:—User-Defined Reader #1 device, 114
UR2:—User-Defined Reader #2 device, 114
Use the built-in calculator for adding and subtracting hexadecimal numbers, 266-267
USER command, 32, 45, 58, 85-89, 314
Using
  ASM, 221-226
  both the "start" and "quit" parameters, 142-143
  CP/M, 23, 37-66
  CP/M Plus, 306-310
  disk drives, 43-45
    and user numbers, 307
  PIP, accessing and, 119
  standard and special destination devices with PIP, 130-131
  variables in .SUB files, 207-209
  wildcards, 50-52, 309-310
  XSUB, 215-217
Utilities, optional transient, 323-345
Utility programs, 32

V

"V" Command, 175, 191
Verify [V], 137
  accuracy of copied data, 137
  total editing buffer usage and free space, 191
Verifying number of disk users, 105-106

Viewing
  logical devices, 93
  physical devices, 93
v–value error, 229

W

"W"(Write) command, 192
Warm boot, 40, 42, 53, 83
When answering
  "No" to quit, 200
  "Yes" to quit, 200
Wildcards ("*" or "?"), 82
Winchester disk drives, 29
WRITE, 308
Write
  parameter [W], 152
  -protected (R/O) disk, 102
  several lines from editing buffer to temporary
    library file, 193-194
    output file, 192-193

X

X command, 264
^X (editing command), 168
XLT-86, 324
  utility, 343-345
XREF, 322, 324
  utility, 340-341
XSUB (Extended SUBMIT) command, 34, 213-217
X SUB.COM (Extended Submit), 46

Y

^Y (editing command), 168

Z

Z key, 167-168
Z-80 SID utility, 328-329
Zero parameter [Z], 156
"0" Zero character, 166
Zilog Z80, 28, 30-31
ZSID, 324

TO THE READER

Sams Computer books cover Fundamentals — Programming — Interfacing — Technology written to meet the needs of computer engineers, professionals, scientists, technicians, students, educators, business owners, personal computerists and home hobbyists.

Our Tradition is to meet your needs and in so doing we invite you to tell us what your needs and interests are by completing the following:

1. I need books on the following topics:

_____

_____

_____

_____

_____

_____

2. I have the following Sams titles:

_____

_____

_____

_____

_____

_____

3. My occupation is:

_____ Scientist, Engineer
_____ Personal computerist
_____ Technician, Serviceman
_____ Educator
_____ Student
_____ D P Professional
_____ Business owner
_____ Computer store owner
_____ Home hobbyist
Other _____

_____

Name (print) _____

Address _____

City _____ State _____ Zip _____

Mail to: **Howard W. Sams & Co., Inc.**
Marketing Dept. #CBS1/80
4300 W. 62nd St., P.O. Box 7092
Indianapolis, Indiana 46206

22015

Need Other Books in the Sams/Waite Primer Series?

UNIX® PRIMER PLUS
This newest of the Sams/Waite Primers presents the elements of UNIX clearly, simply, and accurately, for ready understanding by anyone in any field who needs to learn, use, or work with UNIX in some way. Fully illustrated. Includes two handy, removable summary cards. By Mitchell Waite, Donald Martin, and Stephen Prata.
Ask for No. 22028 ..... **$19.95**

CP/M® PRIMER (2nd Edition)
Completely updated to give you the know-how you need to begin using and working with new or old CP/M versions immediately, in *any* application. Includes CP/M terminology, operation, capabilities, internal structure, and more, plus a convenient, tear-out reference card listing CP/M commands. By Mitchell Waite and Stephen Murtha.
Ask for No. 22170 ..... **$16.95**

SOUL OF CP/M®: HOW TO USE THE HIDDEN POWER OF YOUR CP/M SYSTEM
Teaches you how to use and modify CP/M's internal features, including how to modify BIOS, use CP/M system calls in your own programs, and more! Excellent if you've read *CP/M PRIMER* or are otherwise familiar with CP/M. By Mitchell Waite and Robert Lafore.
Ask for No. 22030 ..... **$18.95**

CP/M® BIBLE: THE AUTHORITATIVE REFERENCE GUIDE TO CP/M
Highly detailed, all-version CP/M reference manual that gives you instant, one-stop access to all CP/M keywords, commands, utilities, conventions, and more. A must if you use any version of CP/M. By Mitchell Waite and John Angermeyer.
Ask for No. 22015 ..... **$19.95**

BASIC PROGRAMMING PRIMER (2nd Edition)
Improved, expanded edition, now usable with the IBM PC and similar computers. Features new advanced BASIC coverage, game-program listings and self-tests with answers! Highly useful, user-friendly, how-to-program material for beginners and more advanced users alike. By Mitchell Waite and Michael Pardee.
Ask for No. 22014 ..... **$17.95**

PASCAL PRIMER
You'll generate powerful programs in UCSD® Pascal as this Sams powerhouse guides you through Pascal program structure, procedures, variables, decision-making statements, and numeric functions. A remarkable book that also contains eight appendixes. By Mitchell Waite and David Fox.
Ask for No. 21793 ..... **$17.95**

COMPUTER GRAPHICS PRIMER
Almost every page of this Sams best-seller has a color drawing, a photograph, a picture, or a schematic that helps you learn graphics programming quickly and easily—including computer animation! Examples can be easily adapted to any computer running a version of Microsoft BASIC. By Mitchell Waite.
Ask for No. 21650 ..... **$14.95**

MICROCOMPUTER PRIMER (2nd Edition)
A little more technical than the other Primers to better cover basic computer concepts, the electronics behind the logic, what happens inside the computer as a program runs, and a little about languages and operating systems. Excellent introductory material. By Mitchell Waite and Michael Pardee.
Ask for No. 21653 ..... **$14.50**

YOUR OWN COMPUTER (2nd Edition)
Shows how you can go about choosing and buying a computer for your own uses. Explores applications, buzzwords, programs, hardware, and peripherals. Compares 30 different models of personal and small-business computers. By Mitchell Waite and Michael Pardee.
Ask for No. 21860 ..... **$8.95**

Apple II, Apple II+, and Apple IIe are registered trademarks of Apple Computers, Inc. • CP/M is a registered trademark of Digital Research, Inc. • UCSD is a trademark of UC Regents, San Diego campus • UNIX is a trademark of Bell Laboratories, Inc.

SAMS BOOK & SOFTWARE ORDER CARD

Catalog No.	Qty.	Price	Total	Catalog No.	Qty.	Price	Total

Subtotal _____
☐ Check ☐ Money Order — Add local tax where applicable _____
☐ MasterCard ☐ Visa — Add Handling Charge 2.00
Total Amount Enclosed _____

Account Number _____ Expires _____

Name (print _____

Signature _____

Address _____

City _____ State _____ Zip _____

Offer good in USA only. Prices subject to change without notice. Full payment must accompany your order. **X0438**

Call 1-800-428-3696 for the name of your nearest Sams retailer or distributor (residents in IN, AK, HI call 317-298-5566).

If your retailer or distributor doesn't stock the Sams publication you need, you can order it through him or directly from us. Orders placed directly with us are subject to a $2.00 additional handling charge per order.

PHONE ORDERS

Depending on where you live, call either number listed above and charge your Sams publication to your VISA or MASTERCARD.

MAIL ORDERS

Use the order form below, or if it's missing, send your order on a plain piece of paper.

(1) Include your name, street or RD address, city, state, and zip.

(2) Tell us the titles of the books you need, the product numbers (see other side of this card) and the quantity of each one you'd like.

(3) Add up the total cost for the books, add your state's sales tax if applicable, and then add $2.00 for handling.

(4) Include your check or money order for the full amount due, or

(5) Charge it to your VISA or MASTER CARD. To charge it, list your account number and the card expiration date. Shipping charges will be added to credit card orders.

(6) Mail your order to:
Howard W. Sams & Co., Inc.
Dept. No. X0438
4300 West 62nd St., P.O. Box 7092
Indianapolis, Indiana 46206

In Canada, contact Lenbrook Electronics, Markham, Ontario L3R 1H2

Prices subject to change without notice. All books available from Sams distributors, bookstores, and computer stores. Offer good in U.S. only. Note: Distributor, computer store and dealer inquiries are welcome.



See other side for list of Sams/Waite Primers and their prices.

Place Stamp Here

HOWARD W. SAMS & CO., INC.

4300 WEST 62ND STREET P.O.BOX 7092

INDIANAPOLIS, INDIANA 46206

TM

cp/m® bible

The Authoritative Reference Guide to CP/M®

There has long been a need for a comprehensive, clearly written reference guide to CP/M. CP/M Bible fills this need. It provides complete description of all of the built-in commands, transient commands, and operating procedures. Geared to the current version 2.2 (and the older 1.3 and 1.4 versions) it also provides an overview of 3.0. It includes descriptions of CP/M compatible languages and related operating systems. The material is presented in an easy-to-look-up, color-keyed format. Complete and authoritative, this is the one book you need beside your CP/M terminal for constant reference.

Mitchell Waite



Mitchell Waite is president of The Waite Group, Inc., a California company specializing in software, book writing, and documentation for the computer marketplace. Mr. Waite has experience in electronics, communications, and programming and is fluent in 10 computer languages. He is the originator of the popular Sams/Waite "Primer" computer book series. Mr. Waite is a coauthor of *Microcomputer Primer, Your Own Computer, Pascal Primer, CP/M® Primer, UNIX® Primer Plus,* and *Soul of CP/M®*, and is sole author of *Computer Graphics Primer* published by Howard W. Sams & Co., Inc.

John Angermeyer



John Angermeyer is a software engineer/documentation specialist for the telecommunications industry and a consultant on CP/M® applications and commercial software programs. In his work in CP/M®, Mr. Angermeyer evaluates software for word processing, data base management, programming, spreadsheet analysis and budget management. He is also competent in foreign language translations of technical journals and marine electronics. Mr. Angermeyer is an avid sailor and has made two transatlantic crossings from Europe to South America. His other interests include natural history and its preservation, camping, and scuba diving.

HOWARD W. SAMS & CO., INC.
4300 West 62nd Street, Indianapolis, Indiana 46268 USA

$19.95/22015

ISBN: 0-672-22015-6