System Software Dokumentation

sirius COMPUTER

SIRIUS 1
User's Guide
for
CP/M-86

sirius

NOTICES

COPYRIGHT

TRADEMARKS

CP/M is a registered trademark of Digital Research, Inc. ASM-86, CBASIC-86, CP/M-86, and DDT-86 are trademarks of Digital Research, Inc.

DISCLAIMER

Sirius Systems Technology, Inc. makes no representations or warranties of any kind with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Sirius shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Further, Sirius reserves the right periodically to revise this publication and to make changes in the content hereof without obligation of Sirius to notify any person of such revision or changes.

PROPRIETARY NOTICE

PART NUMBER

100919-01
Printed in U.S.A.

CONTENTS

USER GUIDE

FIGURES

TABLES

MANUAL PRINTING CONVENTIONS

This manual uses two-color printing in examples to differentiate computer messages from user entries. Computer output is black, and user entries are grey.

The following symbols and abbreviations are used in examples to represent pressing certain keys: ↵ represents pressing the Return key, and ␣ represents pressing the Space bar.

Examples and parts of tables show Alternate function (Alt key) characters as bold face upper case letters. For example, the key sequence Alt C is indicated as **C.**

1

INTRODUCTION

1. INTRODUCTION

This manual describes the CP/M-86 operating system, as used with the SIRIUS 1 Computer. The manual is an introduction to the system for new users. It can also serve as a quick reference guide for users experienced with CP/M-based operating systems. The manual contains basic operating-system information necessary for using CP/M-86 and for developing programs that run on the SIRIUS 1 Computer.

1.1 MANUAL CONTENTS

This manual contains the following six chapters and five appendixes:

- "Introduction" describes manual contents, conventions, and related documentation.
- "CP/M-86 Overview" provides an overview of the operating system.
- "Basic CP/M-86 Operating Procedures" describes loading the operating system, changing the default disk drive, and controlling console and printer output.
- "Device and Media Management" describes formatting and copying diskettes; displaying diskette file statistics; marking files or drives as read-only or as read-write; and assigning physical devices to logical devices.
- "File System Management" describes displaying the diskette directory and the contents of ASCII files; deleting, renaming, and moving files; and executing command files.
- "Creating and Editing Text Files" describes using the ED program to create and modify ASCII files.
- Appendix A, "CP/M-86 Command Format Summary," provides an alphabetic list of the syntax of each CP/M-86 command.
- Appendix B, "CP/M-86 Error Messages," describes the error messages generated by the CP/M-86 operating system.
- Appendix C, "BDOS Error Messages," describes the error messages issued by the Basic Disk Operating System portion of CP/M-86, as used with SIRIUS 1.
- Appendix D, "SIRIUS Error Codes and Definitions," error messages.
- Appendix E, "SIRIUS Utility Error Messages," describes the error messages generated by the SIRIUS utility programs DCOPY, FORMAT, and BOOTCOPY.

A form for readers' comments has been included at the back of the manual. If the comment form has been removed, please send comments to your SIRIUS 1 representative. A list of Sirius offices, subsidiaries, and distributors appears at the back of this manual.

1.2 MANUAL CONVENTIONS

This manual uses the following conventions:

- Examples present computer- and user-generated material differently: computer output is black; user entries are grey.
- In examples and command-format illustrations, nonprinting ASCII characters, such as line feeds and carriage returns, appear as symbols. For instance, **J** represents the character for a line feed; ↵ represents a carriage return; ␣ represents a space.
- In examples and parts of tables, Alternate function (Alt key) characters are represented with boldface upper case letters. For example, the key sequence Alt C appears in examples as **C**. (Use of the Alt key is described in Section 2.3.)
- Command names, file names, and system prompts or responses embedded in text appear in upper case.
- When command formats and standard user entries appear in text for the first time, they are presented in bold-faced single lines.
- In command formats and in Appendix A:
  - Braces ({ }) indicate that one item in the enclosed group must be selected.
  - A vertical bar (|) separates alternatives.
  - Brackets ([ ]) indicate that the enclosed item(s) are optional.
  - An ellipsis in brackets ([...]) indicates that the preceding parameter can be repeated any number of times. A comma preceding the ellipsis ([,...]) indicates that each parameter must be separated from the others by a comma.
  - General forms (which represent specific entries to be supplied by the user - such as file names, drive names, and device names) appear as single words, such as filename, drivename, and logicaldevice. In command and error message formats, general forms are italicized.
  - The term system diskette refers to any diskette that contains the CP/M-86 operating system.

1.3 RELATED PUBLICATIONS

Several publications on SIRIUS 1 and on CP/M and CP/M-86 are available. *Introduction to the SIRIUS 1 Computer* presents basic operating procedures for SIRIUS 1. Users who are inexperienced with computers should familiarize themselves with these procedures before using this manual. *SIRIUS 1 User's Guide for CP/M-86*.

The following manuals provide additional technical information on CP/M-86:

- *SIRIUS 1 Programmer's Guide for CP/M-86* describes the 8086 assembler (ASM-86) and the Dynamic Debug Tool (DDT-86) used to test and debug assembly language programs.

- *SIRIUS 1 System Guide for CP/M-86* describes the internal structure of CP/M-86, how to create an executable command file, and the use of device, disk, and memory-management routines provided by the operating system.

2

CPM/-86 OVERVIEW

CP/M-86 is an operating system for the SIRIUS 1 Computer. The structure of the CP/M-86 file system allows dynamic allocation of file space and both sequential and random file access. Using this file system, many programs can be stored in both source and machine-executable form on one diskette. CP/M-86 includes a text editor (ED), an 8086/8088 assembler (ASM86), and a dynamic test and debug program (DDT).

2.1 COMPONENTS

There are four main components of CP/M-86: (1) the Basic Input/Output System (BIOS); (2) the Basic Diskette Operating System (BDOS); (3) the Console Command Process (CCP); and (4) the Transient Program Area (TPA). Component functions follow:

- BIOS — provides the primitive input/output interface to the diskette drives and input/output devices.
- BDOS — controls the diskette drives and file system.
- CCP — reads the keyboard and processes commands.
- TPA — holds programs loaded from diskette.

2.2 INPUT/OUTPUT DEVICES

CP/M-86 performs its input and output based on the four "logical" devices described in Table 2-1. A logical device is the name of a device (e.g., LST:) that programmers or operators refer to when doing input or output processing. Special-purpose programs, called drivers, handle the transfer of data to and from these logical devices. These drivers interface with specific physical devices, which are actual input/output devices such as disk drives and line printers.

The operator can assign a physical device to a logical device name through the STAT command (as described in Section 4.4). For example, the operator can direct output to any of four physical devices by changing the physical-to-logical device association, as described in Section 4.4.8.

Table 2-1 describes the CP/M-86 logical and physical devices.

The names of the actual devices attached to the SIRIUS 1 computer may or may not correspond to the physical device names listed in Table 2-1. For example, the device TTY: is actually an RS-232-C serial communications port to which a printer, plotter, or modem might be attached. The driver programs in the BIOS define the correspondence of physical device to actual device.

Section 4.4.8 contains instructions for assigning a physical device to a logical device name.

Table 2-1: Logical and Physical Devices

DEVICE TYPE/NAME	DESCRIPTION
Logical Devices	
CON:	Console device — the principal interactive console which communicates with the operator. Typically, CON: is a device such as a CRT or teletype.
LST:	List device — the principal listing device; usually a hard-copy device, such as a printer or teletype.
RDR:	Paper tape reader device — the principal tape punching device; normally a high speed paper tape punch or teletype.
PUN:	Paper tape punch device — the principal tape reading device, such as a simple optical reader or teletype.
BAT:	Batch mode-reader (RDR:) as input; a printer (LST:) as output.
Physical Devices	
TTY:	Serial output-port A (teletype-style printer — RS-232-C)
CRT:	Keyboard and cathode ray tube display
UC1:	External console (to be developed)
PTR:	High speed read (to be developed)
UR1:	(To be developed)
UR2:	(To be developed)
PTP:	High speed punch (to be developed)
UP1:	(To be developed)
UP2:	(To be developed)
LPT:	Parallel port line printer (Centronics)
UL1:	Serial printer — port B (RS-232-C)

2.3 BASIC OPERATING PRINCIPLES AND NOTES

This section (1) describes basic operating principles for using CP/M-86's commands and (2) notes certain general characteristics of the operating system. Some of the operating principles are expanded in Chapter 3.

- The operator initiates CP/M-86 functions by entering commands at the keyboard. The CP/M-86 commands and utility programs that can be used to manipulate the data contained on diskettes are described in Chapters 4, 5, and 6.

- The "default drive" is the currently selected drive, the drive to or from which CP/M-86 is recording or obtaining data. Commands entered to manipulate data act on the diskette in the default drive, unless the operator specifies another drive name in the command.

- Once CP/M-86 is loaded (as described in Section 3.1), the console displays the letter name of the default drive, followed by the command prompt. At the operating-system level, CP/M-86 always displays the name of the default drive with the system command prompt (e.g., **A** or **B**). Because drive A is preferred for loading the operating system, the screen always displays A after loading.

- Commands the user enters in response to the prompt are referred to as "system-level" commands.

- All the commands described in this manual — except the ED commands described in Section 6.2 — can be entered in upper or lower case. CP/M-86 automatically converts commands entered in lower case at the keyboard into upper case.

- The SIRIUS Alt (Alternate function) key has two functions:

  — It displays the symbol or invokes the function shown on the front face of another key.

  — It acts as the Control key (for example, the function generally referred to as a Control C is an Alt C on SIRIUS 1 and is accomplished by striking C while keeping the Alt key depressed).

2.4 ENTERING AND MODIFYING CP/M-86 COMMANDS

The user communicates with the operating system by entering CP/M-86 commands at the keyboard. Pressing the Return key at the end of a command line sends the command to the operating system. A command line can be up to 127 characters long.

The user can modify CP/M-86 commands at any time during entry, until the Return key is pressed. To modify the command line, use the Alt key functions described in Table 2-2.

Table 2-2: Alt-Key Command-Line Editing Functions

ALT CHARACTER	FUNCTION
Alt E	Causes physical end of line; display is shifted to beginning of next line. but line is not sent to CPU until Return key is pressed. Useful for entering command lines that are longer than physical line on screen.
Alt H or Backspace key	Moves cursor back one character position and deletes last character entered.
Alt J	Terminates input (line feed).
Alt M or Return key	Terminates input (carriage return).
Alt R	Redisplays current command line.
Alt U	Cancels current command line.
Alt X	Backspaces (deletes) to beginning of current line.
Alt Z	ASCII end-of-file character; ends input from console.

2.5 FILE SYSTEM

The SIRIUS 1 computer stores information on diskettes in the form of files; a file is one or a group of related characters. A single file can be any length, up to the data storage capacity of a diskette (600-kbytes per side); a diskette directory can contain up to 128 entries. All CP/M-86 file-related functions can be used in user-written programs.

2.5.1 FILE SPECIFIERS

A CP/M-86 file is identified by its file specifier, which consists of one to three parts: a file specifier must contain a file name; it may also contain a file extension and/or a drive name (the name of the drive containing the diskette on which the file is recorded). If the drive name is omitted from a file specifier, CP/M-86 assumes that the file is on the diskette in the default drive.

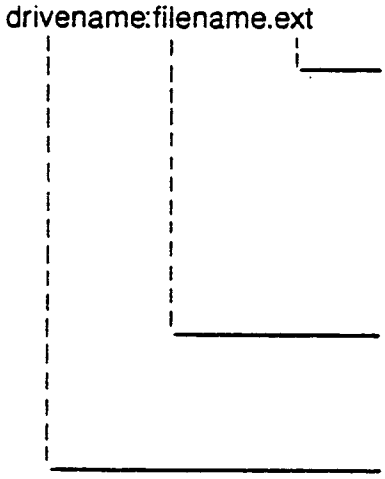
The three elements of a file specifier are arranged in the following sequence, with no intervening spaces:

- Drive name — one letter followed by a colon.
- File name — one to eight characters.
- File extension — one to three characters.

Characters in a file specifier may be entered in upper or lower case. The following characters may not be used in a file name or a file extension: < > . , ; : = ? * [ ]

Examples and command formats in this manual present file specifier elements as follows: "filespecifier" represents a file specifier, "drivename" represents a file specifier's drive name, "filename" represents a file name, and "ext" represents a file extension. The three parts of a file specifier are illustrated in Figure 2-1 and are further described in the next subsections.

Figure 2-1: File Specifier Structure



File extension — contains up to 3 characters. Any character can be used, except < > . , ; : = ? * [ ]

Extensions are an optional part of file specifiers, except for the file types shown in Table 2-3.

File name — contains up to 8 characters. Any character can be used except < > . , ; : = ? * [ ]

Drive name — either A or B. The drive name need not be specified when the file resides on the diskette in the default drive.

2.5.2 CP/M-86 FILE EXTENSION CONVENTIONS

A file specifier's file extension typically describes the form of the data in the file. Conventions govern some types of files, and some languages and application software packages assign file extensions automatically. For example, BASIC-86 assigns the extension .BAS to programs saved from that language. In addition, CP/M-86 defines the meanings of several file extensions, as presented in Table 2-3.

Files can also be assigned extensions at the operator's discretion. For example, an operator might use the extension .TXT for all text files.

Table 2-3: CP/M-86 File Extension Conventions

FILE EXTENSION	CP/M-86 INTERPRETATION
.A86	8086 assembly language source code
.BAK	Backup file created by ED program and some other text editors
.BAS	BASIC source code (Microsoft BASIC)
.CMD	Executable command file
.COB	COBOL source code
.DAT	Data file (assumed to be ASCII)
.FOR	FORTRAN source code
.H86	Hex file produced by assembler
.INT	Intermediate compiled code (CBASIC/86, CIS COBOL)
.LST	Listing of compilation or assembly
.OVR	Overlay module
.PRN	Listing of compilation or assembly
.REL	Relocatable object code module
.SUB	Command file executed by SUBMIT command
.SYM	Symbol table of assembly or compilation
.XRF	Cross-reference
.$$$	Temporary, system-generated file

2.5.3 VALID AND INVALID FILE SPECIFIERS

Examples of valid file specifiers follow:

Examples:

```
PAYROLL
B:1REPORT
PIP.CMD
BJACK.BAS
A:JOE.SMO
```

The following examples present invalid file specifiers; the notations describe why the examples are invalid:

Examples:

MARCHPAYROL	File name contains more than 8 characters.
B:SALES[2]	File name contains brackets, which are illegal characters.
2:JOE.SMO	Drive name is entered as a number, rather than a letter.

If the user attempts to name or save a file with a file name or extension of more than the legal number of characters, CP/M-86 accepts the first eight letters of the file name and the first three letters of the extension as a valid file specifier.

However, if the user attempts to name or save a file with a file specifier that is invalid in any other way, CP/M-86 responds with the message Invalid Format.

2.5.4 WILD-CARD CHARACTERS

CP/M-86 wild-card characters allow the operator to use a single command to perform one task on a group of files.

Use of wild-card characters in CP/M-86 is similar to the use of wild cards in card games; the wild-card characters take on the meaning that the system operator assigns to them by matching existing file names.

The two CP/M-86 wild-card characters are the question mark (?) and the asterisk (*).

The ? wild-card character means "match any character — or no character — in this particular location in the file name or file extension." For example, PAY-???.ROL matches each of these file specifiers:

PAY-JAN.ROL

PAY-FEB.ROL

PAY-MAR.ROL

PAY-JL.ROL

The * wild-card character means "pad with ?s." This enables the operator to refer to entire families of files. For example, *.BAS refers to all files with the extension BAS. As another example, WS*.* identifies any file whose file name (no matter how long) starts with WS, regardless of extension.

Note that * pads all possible characters that follow it, within the file name or the extension where it appears. This means that CP/M-86 cannot read or match any characters following the * in a file name or extension. For example, *86.CMD matches all files with extension .CMD (????????.CMD), not just files ending in 86 with extension .CMD.

Wild-card characters can be used in combination. For example, ?TEST.* refers to all files whose file name is four or five characters long, the last four of which are TEST, regardless of file extension.

Note that the file specifier *.* refers to all files on the default or designated drive.

3

CP/M-86 BASIC OPERATING PROCEDURES

This chapter elaborates on some of the operating procedures presented in Section 2.3 and describes procedures for loading CP/M-86, changing the default drive, and controlling console and printer output via the operating system.

Additional background information on basic operating procedures for SIRIUS 1 appears in the *Introduction to the SIRIUS 1 Computer.*

3.1 LOADING CP/M-86

Use the following procedure to load CP/M-86:

1. Be sure both diskette drives are empty.

2. Turn on the system by pressing the switch located at the left rear corner of the processor unit. The LED busy indicator on diskette drive A lights up, and the system displays a power-up display sequence which includes a symbolic request for insertion of the system diskette (e.g., a flashing arrow and a diskette symbol).

3. Insert the system diskette (label up and label-edge last) into drive A and close the drive door. When CP/M-86 is loaded, the system displays a sign-on message (which presents system information, including a keyboard configuration summary) and the CP/M-86 command prompt for the default or logged drive (A>).

3.2 CHANGING THE DEFAULT DRIVE

As described in Section 2.3, the default drive is the working or currently selected drive. The name of the default drive appears in the CP/M-86 system prompt; for example, the system prompt B> indicates that drive B is the default drive. To change the default drive, respond to the system prompt by entering the letter of the desired drive, a colon (:), and a carriage return. The following example illustrates changing the default drive to B from A.

Example:

```
A>b: ↵
B>
```

3.3 CHANGING DISKETTES AT THE CP/M-86 SYSTEM LEVEL

As a general practice, enter Alt C whenever you change diskettes at the CP/M-86 system level (i.e., when the prompt is >). Entering Alt C at the > prompt resets or clears the disk system and logs all active drives. Active drives include the default drive, drive A (if it is not the default drive), and any drive specified in CP/M-86 operations since the last loading of the operating system.

CP/M-86 "logs" a drive by building and holding in memeory an allocation map of the directory of the diskette in that drive. When the user enters Alt C at the > prompt, the operating system always logs drive A. However, if drive B is the default drive or has been active in CP/M-86 operations, entering Alt C logs both drives.

If you attempt to write to a newly inserted diskette without first resetting the disk system, CP/M-86 will not perform the write operation and will display the following error message:

Bdos Err On X: R/O

where X represents the name of the drive containing the diskette. See Appendix C for explanation of error messages.

3.4 CONTROLLING CONSOLE OUTPUT

When operating at the system level (i.e., when the command prompt is >) anc when working with DDT and PIP, the user can start or stop computer output to either the console or the printer. Starting output to the printer this way causes the printer to "echo" console display. Stopping output to the console temporarily freezes the screen display. Table 3-1 lists the Alternate characters used to control console output.

Table 3-1: Console Output Alternate Characters

ALTERNATE CHARACTER	FUNCTION
Alt P	Sends all console output to both LST: (for hardcopy) and CON: (for screen display, if CON: is assigned as CRT:). Re-entering Alt P stops output to LST:. CAUTION: Unless a printer is on-line, entering Alt P freezes the operating system; the operating system must be reloaded, and all data in memory is lost.
Alt S or Pause/Cont Key	Temporarily stops output to CON:. Console (and printer, if output is being echoed to printer) remains frozen until any key is pressed. Alt S sends Device Control 3 (X-OFF) to CPU.

3.5 LOADING CP/M-86 OR PROGRAMS FROM DRIVE B

Although CP/M-86 always attempts to load from drive A, the operating system can be loaded from drive B. The option of loading from B is useful if drive A is inoperable for any reason. To load the system from drive B, follow steps 1 and 2 in the procedure in Section 3.1. Then insert the system diskette in drive B. After the sign-on message appears, the screen displays this disk operating system error message:

Drive = 0, Track = 0, Sector = 0, Error = FA
Bdos Err On A: Bad Sector

Enter Alt C to reset the disk system and to display the system prompt, A>. Then log onto drive B by entering "b:" followed by a Return.

In addition, you can load any CP/M-86 utility or program from a drive other than the default drive by entering the drive name before the command. For example, to load PIP from the B drive while working on the A drive —

Example:

```
A>b:pip ↵
```

4

DEVICE AND MEDIA MANAGEMENT

CP/M-86 includes these four programs for managing diskette files:

- FORMAT formats diskettes for system use.
- DCOPY copies a diskette's contents onto another diskette.
- BOOTCOPY copies the operating system tracks from one diskette to another.
- STAT reports statistics and controls certain characteristics of diskettes and files.

4.1 DISKETTE FORMATTING — THE FORMAT PROGRAM

The FORMAT program prepares diskettes to receive data. In the process, FORMAT automatically erases any previous files on the diskette. New diskettes must be formatted before they can be used by the system.

4.1.1 USING FORMAT

The basic procedure for formatting a diskette follows:

1. Insert the system diskette in drive A and enter

   format ↵

   SIRIUS 1 loads FORMAT and displays the utility's one-line sign-on banner near the top of the screen.

2. Once FORMAT is loaded, the system diskette can be removed. Insert the diskette to be formatted in either drive.

3. At the bottom of the screen, FORMAT asks which drive contains the diskette to be formatted, as follows:

   Format drive? (A or B; press return key to end)

   Enter the letter name of the correct drive.

4. The system responds by displaying

   Format drive X. Press space bar when ready.

   where X is the drive name entered in step 3.

   Check that the diskette to be formatted is in the correct drive (to avoid possible data loss) and press the Space bar.

5. The formatting procedure takes approximately one minute for a single-sided diskette. At the bottom of the screen, FORMAT displays the number of each track as it is formatted. Near the top of the screen, FORMAT displays the message —

   Format drive X

   where X is the drive name entered in step 3.

6. When formatting is complete, near the top of the screen, FORMAT displays —

   Format drive X complete.

7. FORMAT prompts to repeat the process. To format another diskette, repeat steps 2 through 5. To end the FORMAT program, insert the system diskette in drive A, and enter Alt C or press the Return key.

The following example illustrates a sample session using FORMAT to format one diskette. The example shows FORMAT prompts and messages consecutively, as they appear on the screen, but does not indicate where on the screen the prompts and messages are displayed.

Example:

```
A>format ↵

Diskette FORMAT Utility - Version 2.7

Format drive? (A or B; press return key to end)b

Format drive B. Press space bar when ready. —

Format drive B

Format drive B complete.

Format drive? (A or B; press return key to end) ↵

A>
```

4.1.2 USING FORMAT SWITCHES

The user can expand the FORMAT command by including a drive name and optional "switches" that modify FORMAT program operation. These additions must appear in the command line with the FORMAT command stem; a switch value or group of switches must be preceded by a $ (dollar sign). There are four switches:

C — Display the count of the tracks copied and the number of soft errors encountered.

E — Display the locations of soft errors encountered.

Z — Display disk zone information (size of tracks and gaps).

D — Format a double-sided diskette (to be developed).

Tracks are circular sections of a diskette; SIRIUS diskettes have 80 tracks on a side. These tracks are grouped into eight zones; the drive motor runs at a different speed on each zone. Soft errors are hard-to-read sections or worn or flawed spots on the diskette that may make it unreliable for use. The $C switch (for counting tracks and soft errors) allows the user to gauge the reliability of a diskette for recording data. If 15 to 20 or more errors appear, repeat the FORMAT process a few times. If this number of errors persists, discard the diskette and try another.

One to four switches may be included in the FORMAT command line, as follows:

format [drivename:] [$C | E | Z | D] ↵

The next example presents the command to format a disk in drive B and then to display (1) the number of tracks copied and (2) the number and the locations of soft errors encountered in the process. (See Appendixes D and E for explanation of FORMAT error messages.)

Example:

```
A>format b:$ ce ↵

Soft format error: D=1, T=01, S=12, E=4C
Soft format error: D=1, T=01, S=FF, E=41
Soft format error: D=1, T=01, S=06, E=4A

Format complete.
80 tracks formatted: 3 soft errors.

A>
```

4.2 DISKETTE BACKUP — THE DCOPY PROGRAM

The DCOPY program copies the contents, including the system tracks, of one diskette onto another diskette, creating a literal twin of the source or copy-from diskette. In the process, DCOPY formats the destination or copy-to diskette for CP/M-86 (eliminating the need to run the FORMAT program separately).

There are two methods — long and short — for using DCOPY. With the long method, the program prompts the user for the names of the copy-from and copy-to drives. With the short method, the user enters the program name and the copy-from and copy-to drive names in a single command line, without prompts from DCOPY. The long DCOPY method (described in Section 4.2.1) can be used to copy the contents of one or more diskettes in either drive and to make more than one copy. The short method (described in Section 4.2.2) can be used to make a single copy of the diskette in the default drive; the copy-from diskette must contain DCOPY. With this method, the system exits the program immediately after the copy is completed.

Several of the optional switches described for the FORMAT program can be included in the DCOPY command (see Sections 4.1.2 and 4.2.2).

4.2.1 USING DCOPY WITH INTERACTIVE SYSTEM PROMPTS

Use the following procedure to copy a diskette from either drive and to make multiple copies.

1. Insert the system diskette in drive A, clear the disk system by entering Alt C, and enter the following command:

   dcopy ↵

   SIRIUS 1 loads DCOPY and displays the utility's one-line sign-on banner at the top of the screen.

2. Remove the system diskette from drive A if it is not the copy-from diskette.

3. At the bottom of the screen, DCOPY asks for the name of the drive containing the diskette to be copied:

   Copy from drive? (A or B; press return key to end)

   Insert the copy-from diskette in either drive and insert the copy-to diskette in the other drive. Answer the DCOPY query with the correct copy-from drive name. DCOPY assumes that the remaining drive is the copy-to drive.

4. DCOPY repeats the particulars of the copy command and asks for confirmation:

   Copy from drive f to drive t. Press space bar when ready.

   where f is the name of the copy-from drive, and t is the name of the copy-to drive. Press the Space bar to start the DCOPY process. Pressing any other key cancels the DCOPY command.

5. During the copy process, at the bottom of the screen, DCOPY displays the number of each track as it is copied.

   Near the top of the screen, DCOPY displays the message:

   Copy from drive f to drive t

   At the bottom of the screen, DCOPY displays the number of each track as it is copied. When the copy is complete, the system prompts with:

   Copy from drive f to drive t complete.
   Copy from drive? (A or B; press return key to end)

6. To make another copy, follow steps 3 and 4 above.

   To exit DCOPY (return to the operating system), insert the system diskette in drive A and enter Alt C or press the Return key.

The following example illustrates a sample session using DCOPY to copy one diskette. The example shows DCOPY prompts and messages consecutively, as they appear on the screen, but does not indicate where on the screen the prompts and messages are displayed.

Example:

```
A>dcopy ↵

Diskette COPY Utility - Version 2.4

Copy from drive? (A or B; press return key to end)a

Copy from drive A to drive B. Press space bar when ready. —

Copy from drive A to drive B

Copy from drive A to drive B complete.

Copy from drive? (A or B; press return key to end) ↵

A>
```

4.2.2 USING DCOPY WITHOUT SYSTEM PROMPTS

Use the following procedure for the short, one-command DCOPY method:

1. To copy the system diskette or any diskette containing DCOPY, insert the copy-from diskette in the default drive, insert the copy-to diskette in the other drive, and clear the disk system with Alt C. Then enter the following command:

   dcopy f: to t: ↵

   where f represents the copy-from drive name, and t represents the copy-to drive name.

2. At the bottom of the screen, DCOPY displays the number of each track as it is copied. When DCOPY has completed the copy process, the system displays the following message:

   Copy from drive f to drive t complete.

NOTE: DCOPY's short command form can include the switches C, E, and Z, as for the FORMAT program (see Section 4.1.2). For example, to display the count of tracks copied and of soft errors encountered, and to copy the contents of a diskette in drive A to a diskette in drive B, enter —

EXAMPLE

```
A>dcopy a: to b: $c ↵
```

4.3 OPERATING-SYSTEM COPY — THE BOOTCOPY PROGRAM

The CP/M-86 BOOTCOPY program creates a system diskette by copying the system tracks from a system diskette onto another diskette. The system tracks contain the portion of the operating system that "boots" or loads into memory when the user inserts the system diskette at power-up or when the user presses the Reset button to re-load the operating system. The system tracks also contain the resident CP/M-86 commands, such as DIR and TYPE, that do not appear on the diskette directory. If the destination diskette already contains an operating system, BOOTCOPY writes over it. For BOOTCOPY to run, the destination diskette must have been formatted with the CP/M-86 FORMAT program. The BOOTCOPY process does not affect any files on either diskette's directory.

Like DCOPY, the BOOTCOPY program has both a short and a long command form (see Section 4.2). The following procedure and example depict the long form of BOOTCOPY (with interactive queries from the program). The short, one-command version is described at the end of this section.

To copy the operating system tracks from one diskette to another, use the following procedure:

1. Load the operating system (if it is not already loaded) and enter the following command:

   bootcopy ↵

   After loading, the BOOTCOPY program identifies itself with a sign-on banner and then asks which drive contains the source or copy-from diskette:

   Source drive (A/B) ?

2. Enter the source drive name. The program identifies the selected drive and requests confirmation:

   Source disk in drive X and press <space>.

   where X is the name of the drive entered in step 1.

   Make sure that drive X contains the diskette whose system tracks you want to copy.

3. Press the Space bar to begin the BOOTCOPY process. BOOTCOPY reads the operating-system sectors into memory and then xasks which drive contains the destination or copy-to diskette:

   Destination drive (A/B) ?

4. Place the destination diskette in either drive and enter the drive name. BOOTCOPY displays the following request for confirmation:

   Destination disk in drive X and press <space>.

   where X is the name of the drive entered in step 3. Press the Space bar to complete the BOOTCOPY process.

5. When the copy is complete, BOOTCOPY displays a confirming message and again prompts for a destination drive:

   Bootcopy complete.
   Destination drive (A/B) ?

   If you want to BOOTCOPY the same system tracks to another diskette, insert another destination diskette and follow step 4 above.

   To exit the program, press the Return key.

   To BOOTCOPY from a different source diskette, exit the program and then enter a new BOOTCOPY command.

The following example illustrates a sample session of BOOTCOPY:

Example:

```
A>bootcopy ↵

BOOTCOPY Version 1.6

Source drive (A/B) ?a
Source disk in drive A and press <space>. ⌴

Reading system sectors...

Destination drive (A/B) ?b
Destination disk in drive B and press <space>. ⌴

Writing system sectors...

Bootcopy complete.

Destination drive (A/B) ? ↵

A>
```

The command format for the short, one-command version of BOOTCOPY follows:

bootcopy [drivename: to drivename:] ↵

With the short command form, BOOTCOPY exits to CP/M-86 after completing one copy. To BOOTCOPY again, enter another BOOTCOPY command.

4.4 DEVICE AND FILE STATUS — THE STAT PROGRAM

Use the STAT program to display information about diskettes and files, to specify whether file names are to be displayed in the directory, to specify whether diskette contents can be user-modified, and to change physical-to-logical device assignments.

Parameters specified in the STAT command determine what information about the system, the diskettes, or the files is displayed. Refer to Table 4-1 for STAT commands, parameters, and formats.

Table 4-1: STAT Command Options

COMMAND/PARAMETER	BRIEF DESCRIPTION
stat ↵	Displays amount of available storage space on diskette in default or active drive(s).
stat dev: ↵	Displays physical-to-logical device assignments.
stat *drivename:* ↵	Displays amount of available storage space in specified drive.
stat [*drivename:*] dsk: ↵	Displays diskette characteristics for active drive(s) or specified drive.
stat *drivename:* = r/o ↵	Temporarily sets a drive to read-only; enter Alt C to restore R/W status
stat *filespecifier* ↵	Displays file characteristics and read-write status.
stat *filespecifier* $s ↵	Displays file size and read-write status.
stat *filespecifier* {$r/o|r/w} ↵	Sets file access attribute to read-only (R/O) or read-write (R/W).
stat *filespecifier* {$sys\|$dir} ↵	Takes file a system file or a directory file.
stat *logicaldevice=physicaldevice* ↵	Changes physical-to-logical device assignments.
stat *val:* ↵	Lists available STAT command parameter options.

4.4.1 AVAILABLE SPACE ON ALL DRIVES

When entered with no parameters, the STAT command displays the amount of available disk space and the read/write status of all active diskette drives.

Active drives, as defined in Section 3.3, include the default drive and any drive previously specified in CP/M-86 operations. As an illustration, once the DIR B: command has displayed the directory of drive B, that drive is an active drive even if it is not the default drive.

To display file information concerning the diskettes in drives A and B when both drives are active, enter the following command:

stat ↵

The following example demonstrates using STAT when drive A contains a diskette with 120K bytes of available storage and drive B contains a read-only diskette with 215K bytes of available storage.

Example:

```
A>stat ↵
A: R/W, Space: 120k
B: R/O, Space: 215k

A>
```

4.4.2 AVAILABLE SPACE ON A SPECIFIED DRIVE

The STAT command followed by a drive name displays space information for the diskette in the specified drive.

The following example illustrates using STAT to display the statistics for a diskette in drive B that contains 120K bytes of available memory.

Example:

```
A>stat b: ↵
Bytes Remaining On B:120k

A>
```

4.4.3 FILE INFORMATION

When followed by a file specifier, the STAT command displays information about the specified file(s). Wild-card characters (? and *) can be used in the manner described in Section 2.5.

The CP/M-86 STAT display incorporates the following column heads:

Recs — The number of 128-byte records in the file.

Bytes — The number of 8-bit bytes allocated to the file.

Ext — The number of 16-Kbyte extents allocated to the file.

Acc — The accessing attribute of the file, which indicates whether the file is read-only (i.e., write-protected) or read-write.

The next example demonstrates using STAT to display file information for all files on the diskette in the logged drive that have names beginning with S and extensions of .CMD.

Example:

```
A> stat s*.cmd ↵

 Recs      Bytes     Ext    Acc
------------------------------------------
    4        2K       1     R/W A:SET.CMD
    5        2K       1     R/W A:SETUP.CMD
   84       12K       1     R/W A:STAT.CMD
   32        4K       1     R/W A:SUBMIT.CMD

Bytes Remaining on A: 364K

A>
```

When working with random (sparse) files, the $S parameter following the file specifier adds an additional column, Size, to the display. The following example illustrates using STAT to display information for a random file. The final record number is 10000, although far fewer records have actually been allocated.

Example:

```
A>stat test.dat $s ↵
   Size        Recs        Bytes       Ext    Acc
   0000        17          4K          2      R/W A:TEST.DAT
A>
```

4.4.4 SETTING A FILE OR DISKETTE TO READ-ONLY OR READ-WRITE STATUS

As indicated in Section 4.4.3, if the user sets the accessing attribute of a diskette or file to read-only, the data in the file or diskette cannot be written over or erased (except under FORMAT or DCOPY). If a diskette or file is set to read-write status, the user can modify or erase the data.

Use STAT to change the accessing attribute of a file or diskette. The command format to change a file's access attribute follows:

stat filespecifier { $r/o| $r/w} ↵

The command format to set a diskette to temporary read/only status follows:

stat drivename: = r/o ↵

where drivename is the name of the drive containing the diskette.

The following example sets the file JOE.TXT on drive B to read-only.

Example:

```
A>stat b:joe.txt $r/o ↵
```

You will not be able to write to (record on) the file JOE.TXT unless you reset its accessing attribute. To reset the file to read-write status, replace $r/o with $r/w in the STAT command in the preceding example.

The next example demonstrates temporarily setting an entire diskette on drive B to read-only status.

Example:

```
A>stat b:=r/o ↵
```

To restore a diskette to read-write status, enter Alt C at the system prompt.

You can ensure permanent write-protection for a diskette by using a write-protect tab. Covering the write-enable notch on the diskette jacket with an adhesive write-protect tab will prevent any writing to the disk until the tab is removed.

4.4.5 EXCLUDING FILE NAMES FROM THE DIRECTORY

STAT can be used to exclude file names from the diskette directory display. The excluded files, known as system files, are still available for use although their presence is not indicated in the directory. The user might set machine-readable files, such as command files, to system file status in order to protect them from being accidentally written over or altered. Names of system files appear in parentheses when file status information is listed by STAT.

Use the following command format to exclude a file name from the diskette directory:

stat filespecifier $sys ↵

To reinstate a file name in the diskette directory, use the following command format:

stat filespecifier $dir ↵

4.4.6 DISPLAYING STAT COMMAND PARAMETERS

Enter the following command to display all the parameters that can be manipulated with STAT:

stat val: ↵

The system responds with a listing of the CP/M-86 STAT parameters, as follows:

Example:

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

4.4.7 DISPLAYING DRIVE CHARACTERISTICS

To display disk drive characteristics, enter the STAT command followed by a disk drive name and the special term, DSK:. The system responds with a listing of the sizes of the tracks, blocks, and sectors on the diskette, as well as the capacity of the diskette and its directory.

The following example illustrates using STAT to display the disk drive characteristics of a diskette in drive B.

Example:

```
A>stat b:dsk: ↵

     B:  Drive Characteristics
  4592:  128 Byte Record Capacity
   574:  Kilobyte Drive Capacity
   128:  32 Byte Directory Entries
   128:  Checked Directory Entries
   128:  Records/Extent
    16:  Records/Block
    64:  Sectors/Track
     0:  Reserved Tracks

A>
```

For explanation of the technical terms that appear in the preceding display, see the *SIRIUS 1 System Guide for CP/M-86.*

4.4.8 PHYSICAL-TO-LOGICAL DEVICE ASSIGNMENTS

The STAT command assigns a physical device to a logical device name, within allowable physical-to-logical device relationships, as listed in Table 4-2. Four logical devices appear in the first table column. For each logical device, the user can assign any of the four physical device names listed beside it in the second column. Physical devices and logical device names are described in Section 2.2.

Table 4-2: Physical-To-Logical Device Assignments

LOGICAL DEVICE	ASSIGNABLE PHYSICAL DEVICES			
CON:	TTY:	CRT:	BAT:	UC1:
LST:	TTY:	CRT:	LPT:	UC1:
RDR:	TTY:	PTR:	URL:	UR2:
PUN:	TTY:	PTP:	UP1:	UP2:

One STAT command can assign all four logical devices, provided they are separated by single commas. The command format for assigning physical to logical devices follows:

stat logicaldevice = physicaldevice [,..] ↵

The next example demonstrates changing the punch device to TTY:, changing the list device to LPT:, and changing the paper tape reader to TTY:.

Example:

```
A>stat pun:=tty:,lst:=lpt:,rdr:=tty: ↵
```

Use the STAT command with the DEV: parameter to check the physical/logical device assignments. The next example demonstrates displaying the four logical device assignments.

Example:

```
A>stat dev: ↵
CON:  is CRT:
RDR:  is TTY:
PUN:  is TTY:
LST:  is TTY:

A>
```

5

FILE SYSTEM MANAGEMENT

CP/M-86 provides four system-level resident or built-in commands and two utility programs that assist in managing files:

- The DIR command displays the file names and extensions for the files stored on the logged or specified diskette(s).
- The TYPE command displays the information stored in ASCII files.
- The ERA command deletes files from a diskette.
- The REN command renames an existing file.
- The PIP program (the Peripheral Interchange Program) moves files (or file parts) from one storage device or media to another.
- The SUBMIT program allows the operator to string several commands together in one file.

All CP/M-86 commands can be entered in either upper or lower case.

5.1 THE DIR COMMAND

The DIR command displays all or part of the contents of a diskette's directory, a computer-maintained list of the file specifiers for the files and programs residing on the diskette.

5.1.1 DIR COMMAND FORMAT

The format of the DIR command is —

dir { **drivename:** ↵ / **filespecifier** ↵ }

If the DIR command is entered without a drive name or file specifier, the system displays the directory for the diskette in the logged drive. If a drive name is specified, the system displays the directory for the diskette in that drive. If a file specifier is given, the system lists the file or files (wild-card characters can be used in the file specifier) that match it.

If the diskette directory is empty or if DIR cannot find a file specifier that matches the given file specifier, the system responds with a message such as NO FILE.

5.1.2 DIR COMMAND EXAMPLES

To display the file directory for the diskette in the default drive, enter the following:

dir ↵

The system responds as shown in the following hypothetical example:

Example:

```
A>dir ↵
A: PIP          CMD : STAT     CMD : ASM86   TXT  : SUBMIT  CMD
A: BOOTCOPY     CMD : DCOPY    CMD : ED      CMD  : FORMAT  CMD
A>
```

Three DIR command line examples follow. To display all files on the diskette in drive B (regardlesss of which drive is currently logged) —

Example:

```
A>dir b: ↵
```

To display all files with the extension .CMD on the diskette in the default drive —

Example:

```
A>dir *.cmd ↵
```

To verify the existence of a file named ACCRED.INT on the diskette in drive B —

Example:

```
A>dir b:accred.int ↵
```

5.3 THE TYPE (DISPLAY) COMMAND

The TYPE resident command displays the contents of an ASCII text file. Use the Cont key to stop and start the display. A specific file name must be given; wild-card characters cannot be used with TYPE.

The format of the command follows:

type *filespecifier* ↵

Two command line examples follow. To display a file named SUMMARY on the diskette in the default drive —

Example:

```
A>type summary ↵
```

To display the file named MEMO.BAK on the diskette in drive B —

Example:

```
A>type b:memo.bak ↵
```

5.3 THE ERA (DELETE) COMMAND

The ERA resident command deletes the directory entries of one or more files. Unlike the FORMAT program, which erases the entire diskette, ERA deletes only the directory entry, thereby freeing previously allocated file space for re-use.

Wild-card characters can be used (with caution) to delete groups of files with one ERA command.

The format of this command follows —

era *filespecifier* ↵

If no file is found with the file specifier entered in a command line, ERA responds with the message NO FILE.

The following example shows erasing a file named ACCOUNTS.BAS on a diskette in the default drive.

Example:

```
A>era accounts.bas ↵
```

All files on a diskette can be erased via use of the *.* wild-card characters.

Example:

```
A>era *.* ↵
```

The system responds to the preceding command (to erase all files on a diskette) with the message —

ALL (Y/N) ?

Enter Y to erase all files on the logged or specified drive(s). Any other response cancels the command.

5.4 THE REN (RENAME) COMMAND

The REN resident command changes the directory entry (the file specifier) for a file. The format of the command follows:

ren newfilespecifier=oldfilespecifier ↵

Wild-card characters cannot be used with REN. If the new file specifier named in the REN command already exists, the system responds with the message FILE EXISTS and cancels the command.

Each REN operation takes place on one drive only — either the default drive or the drive named in the new file specifier. If the REN command line names two different drives, the system responds with a non-recognition query and cancels the command.

For example, to rename ACCOUNTS.BAS as BOOKS.BAS on the default drive —

Example:

```
A>ren books.bas=accounts.bas: ↵
```

To rename PIP.CMD to COPY.CMD on drive B —

Example:

```
A>ren b:copy.cmd=pip.cmd ↵
```

If two drive names appear in a REN command, the system will repeat the second portion of the command, followed by a question mark.

Example:

```
A>ren b:copy.cmd=:pip.cmd ↵
A:PIP.CMD?

A>
```

When REN (or any CP/M-86 program) responds to a command with a non-recognition query such as the one above, check the format and spelling of the problematic command line. Then re-enter the command while making the necessary changes.

5.5 PIP — PERIPHERAL INTERCHANGE PROGRAM

The user can employ PIP and its command parameters to perform many file-handling functions. PIP copies, prints, displays, and combines diskette files. PIP can read one or more source files and copy them individually (with the same or new file specifiers) onto another diskette, or concatenate (combine) them into a single file on the source diskette or on another diskette. It can also copy a portion of a file to another file. Further, PIP can output a file to a peripheral device such as a CRT or printer. Optional PIP command parameters, described in Table 5-1 and Section 5.5.3, can modify data as it is copied.

5.5.1 LOADING PIP

PIP commands can be entered in either of two ways. The first method is to load the PIP program without any file specifiers or parameters (i.e., enter PIP followed by a Return). Once loaded in this manner, the program displays its prompt, an asterisk (*) to indicate that it is ready to accept the completion of a PIP command. The PIP program remains loaded until Alt C or a Return — each entered as a complete command line — returns the system to CP/M-86.

Example:

```
A>pip ↵
*
```

This method of loading the PIP program enables the user to perform a number of PIP operations without re-entering the command stem for each one.

The second method of loading the PIP program is to enter the PIP command stem on the same line with the file specifiers and any PIP parameters for the operation you are requesting. Pressing the Return key loads PIP and simultaneously enters and executes the command particulars (file specifiers and parameters). After execution, control of the system returns to CP/M-86 (which responds with a>). To load and run PIP again, you must re-enter the PIP command stem plus the command particulars for the next operation.

The examples in this manual show the second (one-line) method of entering PIP commands.

5.5.2 COPYING FILES WITH PIP

The general format for PIP commands that copy files follows:

pip destination=source[,source2...] [[parameterlist]] ↵

The definitions of the three general forms in the PIP command format follow:

Destination — the name of the file or device that will receive the data; destination can include a disk drive or a logical device name followed by a colon.

Source — the name(s) of the file(s) or device(s) that will be copied onto the destination; source can include a disk drive or device name followed by a colon.

Parameterlist — one or more command parameters (described in Table 5-1) enclosed in brackets, immediately following the name(s) of the affected file(s) or device(s).

Note that PIP parameters, when used, must be enclosed by square brackets entered with the PIP command. In the command format above, the first pair of brackets around parameterlist function as a typographical device that indicates optional elements (see Section 1.2, "Manual Conventions"). The second pair of brackets are literal characters to be entered in the command line. The use of PIP parameters is discussed in Section 5.5.3.

The following example illustrates using PIP to make a copy of file X, calling the new file Y, on the logged drive.

Example:

```
A> pip y=x ↵
```

You can abbreviate PIP commands under some circumstances. Only one file specifier requires a file name when a file is copied from one drive or device to another with no change in the file name itself; CP/M-86 assumes the file name to be the same for both destination and source.

The next three examples demonstrate alternative abbreviated command lines that can be entered to copy file JOE.TXT from the logged drive (A) to a different drive (B), without changing the file name. Note that PIP always assumes the default drive in any file specifier that contains no specific drive name.

Example:

```
A>pip b=joe.txt ↵
```

Example:

```
A>pip b:joe.txt=a: ↵
```

Example:

```
A>pip b:=a joe.txt ↵
```

The wild-card character * can be used to manipulate groups of files. For example, to copy all files with an extension of .CMD from drive A (as the default drive) to drive B, without changing the file names —

Example:

```
A>pip b:=*.cmd ↵
```

The next example demonstrates file concatenation with PIP. To concatenate (combine) files X, Y, and Z on drive A into one file named Q on drive B, regardless of which is the default drive —

Example:

```
A>pip b:q=a:x,a:y,a:z ↵
```

5.5.3 USING PIP PARAMETERS

The user can modify copy operations by including command parameters in brackets in the PIP command. (Section 5.5.1 describes the use of brackets in PIP commands.) Table 5-1 describes the PIP command parameters included in CP/M-86; examples illustrating these parameters follow the table.

Table 5-1: PIP Command Parameters

PARAMETER	EFFECT
B	Transfers data in block mode; data is buffered until X-OFF (Alts) is received from the source device. This allows transfer of data to a file from a continuous reading device, such as a cassette. Diskette buffer is cleared and more input data is read.
Dn	Deletes characters extending past column n. Use this parameter to truncate long lines sent to printer or display.
E	Echoes or outputs to the console the characters in the file affected by the PIP operation.
F	Filters or removes the existing form feeds from the file. Use the P parameter in the same PIP parameter list to insert new form feeds.
G	User number (to be developed).
H	Transfers hex data. Checks all data for proper hex file format. Removes non-essential characters between hex records. System then prompts for corrective action if errors occur.
I	Ignores :00 records in transfer of hex format file. (I parameter automatically sets H parameter.)
L	Translates upper case letters to lower case.
N	Numbers each line, starting at 1 and incrementing by 1. Suppresses leading zeros; line numbers are followed by colons. If N2 is specified, includes leading zeros and inserts tabs following the line numbers. Expands tabs if T parameter is set.
O	Transfers object files (non-ASCII); normal CP/M-86 end-of-file terminator is ignored.
Pn	Includes one form feed (page eject) every n lines (with initial form feed). If n equals 1 or is omitted, form feeds occur every 60 lines. If F parameter is used, deletes existing form feeds before new ones are inserted.
Qs**Z**	Stops copying from source device or file when string s is encountered. If no Q parameter is entered, copying stops at end-of-file. Use S parameter to specify the string from which to start copying.
R	Reads system files (files with directory display suppressed by STAT $SYS command).
Ss**Z**	Starts copying when string s is encountered. Use S and Q parameters (start and quit strings) to select a portion of a file for copying. Copying will start at the file beginning if no S parameter is entered. PIP translates start and quit strings into upper case if the user enters the string on the same line as the PIP command itself. Strings are not translated if the string is entered in response to the PIP prompt (*).
Tn	Expands tabs (**I** characters) to every nth column.
U	Translates lower case letters to upper case.
V	Verifies that data has been copied correctly by rereading after write operations (destination must be file).
W	Writes over R/O files without console interrupts.
Z	Forces parity (high order) bit of each ASCII character to zero.

Five examples illustrating the use of PIP parameters follow.

To make a copy of X.TXT, renaming it Y.TXT (when both files reside on drive B) and to instruct PIP to echo the file to the console, translate lower case letters to upper case, and verify the results —

Example:

```
A>pip b:y.txt=b:x.txt[euv] ↵
```

To copy a portion of file Y.TXT on drive B, starting at the beginning of the source file and ending at the words "Once upon" and continuing to the end of the file —

Example:

```
A>pip a:x.txt=b:y.txt[sOnce upon Z] ↵
```

To copy a portion of file X.TXT on the default drive (A) into a file Y.TXT on drive B, starting at the beginning of the source file and ending at the words "last straw." —

Example:

```
A>pip y.txt=x.txt[qlast straw.Z] ↵
```

You can use PIP to create and format disk files so that files entered with a text editor or word processing program can be printed within CP/M-86. For example, to output or copy the file X.TXT from drive B to the LST: device, expand tabs every 8 columns, and start a new page every 60 lines (standard text-entry format translated into CP/M-86 PIP parameters) —

Example:

```
A>pip lst:=b:x.txt[t8p60] ↵
```

To copy the file X.TXT on drive B to the diskette in drive A (keeping the same file name), delete the embedded form feeds, change the page length to 52 lines, and number each line —

Example:

```
A>pip a:=b:x.txt[nfp52] ↵
```

5.5.4 OUTPUTTING FILES TO LOGICAL DEVICES WITH PIP

PIP can "copy" or output files to logical devices as well as to storage media. Table 5-2 lists and describes the logical devices and the special terms you can use with PIP; examples follow Table 5-2 to illustrate the use of these terms.

When a logical device is specified as the source, PIP reads output from that device until encountering an end-of-file marker. When copying a keyboard entry to a logical device, enter Alt Z to indicate end-of-file.

When a logical device is specified as the destination, PIP copies or outputs the specified file to that device. For example, specifying the logical device CON: as the destination in a PIP command will display the file contents on the console screen.

The specification of logical devices in PIP commands is subject to physical constraints. When a logical device name appears in either the destination or source fields of a PIP command, the physical device assigned to that logical device (see Section 4.4.8) must be able to receive or send data as specified. For example, the logical device LST: is an output device and cannot be a source of data.

Table 5-2: Logical Devices and Special Terms Used With PIP

DEVICE	FUNCTION
CON:	Sends the specified data to the console device.
LST:	Sends the specified data to the list device.
PRN:	Similar to LST:, except that tabs are expanded at every eighth character position, lines are numbered, and form feeds are inserted at the beginning of the file and at every 60th line. PRN: is equivalent to LST: [t8np60].
PUN:	Sends the specified data to the paper tape punch.
EOF:	Sends CP/M-86 end-of-file marker (Alt Z) to destination device.
NUL:	Sends 40 nulls (ASCII 0s) to destination device (this can be issued at the end of punched output).
RDR:	Sends the specified data to the reader device.

Two examples follow.

To concatenate three .A86 files and output them to the CON: device, followed by an end-of-file marker —

Example:

```
A>pip con:=x.a86,y.a86,z.a86,eof: ↵
```

To send 40 nulls to PUN: and to copy the file X.A86 to PUN:, followed by an end-of-file marker and 40 more null characters —

Example:

```
A>pip pun:=x.a86,eof:,nul: ↵
```

5.6 THE SUBMIT COMMAND AND SUB FILES

SUBMIT is a CP/M-86 utility that enables the user to perform a series of CP/M-86 tasks with one command. Each SUBMIT command executes a user-generated list of CP/M-86 commands stored in a specified diskette file identified by the file extension .SUB. The commands in the .SUB file can contain substitution parameters for which specific values — such as file names — are substituted via the SUBMIT command. For SUBMIT to run, SUBMIT.CMD must be on the diskette in the default drive (see Sections 2.3 and 3.2); if the submitted file is on another drive, the drive name must be included with the .SUB file name.

To create .SUB files, use the text editor ED (see Chapter 6) or a word processing application program. Each .SUB file is composed of a list of CP/M-86 commands that the computer performs when executing the .SUB file.

You can create master .SUB files by including in the .SUB file substitution parameters for which you insert values when entering the SUBMIT command. A variable for which a value must be supplied each time the .SUB file is executed is represented by a substitution parameter (a dollar sign $ followed by an integer, called the parameter number). The first variable in a command file is represented by the parameter $1; the second by $2; etc. If the .SUB file contains no substitution parameters, you cannot vary the contents of the submitted file.

When the .SUB file is executed, SUBMIT pairs the parameters specified in the SUBMIT command with subsitution parameters $1 through $n in the .SUB file. If the number of parameters in the SUBMIT command and in the .SUB file do not correspond, the SUBMIT command is canceled, and the system displays an error message. Entering another SUBMIT command as the last command in a .SUB file creates chained command files.

The format of the SUBMIT command follows:

submit filename [parameterlist] ↵

where filename is the .SUB file name (entered with or without the .SUB extension) that contains the prototype commands to be executed, and parameterlist is the list of parameters, separated by blanks, to replace the substitution parameters named in the master .SUB file.

For example, suppose the file ASMBL.SUB contains the following CP/M-86 commands:

```
ASM86  $1
DIR  $1.*
ERA  *.BAK
PIP  $2:=$1.PRN
ERA  $1.PRN
```

To execute the commands listed in file ASMBL.SUB and to substitute REPORT1 for $1 and PRN for $2 in the .SUB file —

Example:

```
A>submit asmbl report1 prn ↵
```

SUBMIT reads the .SUB file, substitutes REPORT1 for all occurrences of $1 and PRN for $2, and creates a file named $$$.SUB that contains the following commands:

```
ASM86  REPORT1
DIR  REPORT1.*
ERA  *.BAK
PIP  PRN:=REPORT1.PRN
ERA  REPORT1.PRN
```

SUBMIT creates the $$$.SUB file on drive A. When SUBMIT executes the $$$.SUB file, drive A is automatically logged. Therefore, all the commands listed in the .SUB file must reside on the diskette in drive A, and the diskette must be R/W. Command processing is aborted if any system error occurs during execution of a $$$.SUB file.

6

CREATING AND EDITING TEXT FILES

Use the CP/M-86 text editor (ED) to create or modify ASCII files. ASCII files can be programs, documents, or data for programs.

ED creates files from text entered at the keyboard. When used to modify an existing file, ED reads the file to be edited into a portion of memory called the edit buffer and modifies it according to operator commands. The edit buffer is empty until the operator loads it with either new text or text from an existing file. During the edit, the changed file can be written to a temporary file, to which ED assigns the same file name as the original file, with the extension of .$$$. ED keeps the original file, which is assigned the extension .BAK, as a backup file. After the edit, ED renames the temporary file to the original file specifier.

For example, to modify the existing file, SAMPLE.TXT, ED creates a temporary file named SAMPLE.$$$ to be used during the ED session to contain the newly modified file. When the operator terminates the session, ED renames the original file as SAMPLE.BAK and renames SAMPLE.$$$, the temporary file, SAMPLE.TXT. Refer to Section 6.7 for a sample ED session.

6.1 ACTIVATING THE TEXT EDITOR

Use the ED command to activate the CP/M-86 text editor. To initiate ED, after the system prompt, enter —

ed filespecifier ↵

where ED is the command stem, and filespecifier is the name of the file to be created or edited.

If no file with the given specifier exists, ED creates a new file, assigning it that file specifier as a name, as described in Section 6.3. If the file specifier is omitted from the command line, the system responds with the error message DISK OR DIRECTORY FULL.

When activated, ED displays an asterisk (*) as its command prompt.

6.2 BASIC OPERATING PRINCIPLES

While inputting at the keyboard, terminate each line with a carriage return (a line feed is supplied automatically with each carriage return).

ED assigns line numbers to each line of text for use during the editing session. The line numbers are not written into the file. Use the line numbers to refer to specific lines of text that are to be modified.

ED commands should be entered as lower case characters, even though ED accepts both upper and lower case. If the editing commands I, F, S, N, J, M, and R are entered in upper case, subsequent text will be translated to upper case, regardless of how it is entered or displayed on the screen.

Save in-process work at regular intervals. If data is accidentally deleted from the edit buffer, this step will minimize the loss.

6.3 CREATING A TEXT FILE

To create a text file, after the system prompt (>) enter the ED command and the file specifier you wish to assign to the new file:

ed filespecifier ↵

Unless a file with the given file specifier already exists, the system responds with the message NEW FILE and its command prompt.

Next prepare ED to accept text input by entering the I (insert)) command:

i ↵

ED indicates it is ready to accept input by displaying the first line number. The following example illustrates opening a new text file named JOE.TXT.

Example:

```
A>ed joe.txt ↵

NEW FILE
     : *i ↵
    1:
```

To terminate text insertion, enter Alt Z. ED responds to an Alt Z by displaying its prompt, an asterisk (*).

6.4 EDITING AN EXISTING FILE

To edit an existing file, enter the file's specifier with the ED command. After the ED command prompt, load the file to be edited into the edit buffer by entering the A (append) command:

#a ↵

Enter a number before the A command to indicate the number of lines of the file to be loaded into the edit buffer. The pound sign (#) entered in the preceding command represents the number 65535, the largest number of lines that ED will allow; enter the pound sign before the A command to load the entire file into the edit buffer. If no integer is included in the A command, ED assumes the number 1 (see Table 6-1).

In the following example, the file SAMPLE.TXT is loaded into the edit buffer for modification.

Example:

```
A>ed sample.txt ↵
      : *#a ↵
     1: *
```

Use of the A command is explained in detail in Table 6-1.

6.5 TEXT TRANSFERS BETWEEN THE EDIT BUFFER AND ASCII FILES

The edit buffer is a text storage area in the computer memory. Editing commands affect text in the editor buffer; they do not affect files on the diskette. Table 6-1 describes the commands that move lines of text between the edit buffer and the temporary file (with given extension .$$$) created by ED to store work during an editing session. Figure 6-1 illustrates use of these commands.

Table 6-1: Edit Buffer - Text File Transfer Commands

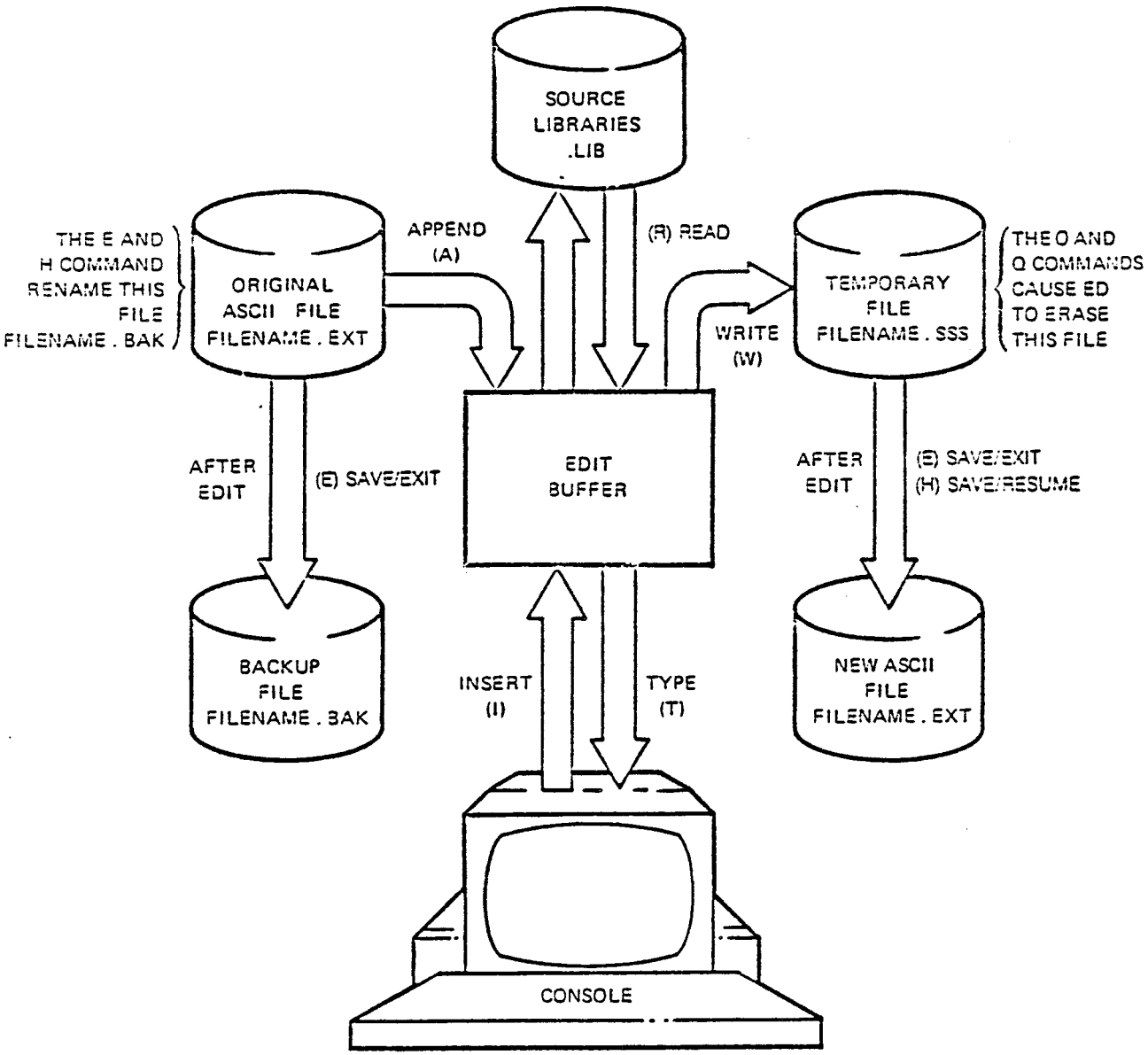
COMMAND FORMAT	COMMAND FUNCTION
nA ↵	Appends to the last character in the edit buffer the next n lines from original text file to be edited. Entering #A loads 65535 lines of text or the entire file, whichever is smaller, into the edit buffer from the original text file.
nW ↵	Writes the first n lines of the edit buffer into temporary file. Remaining lines are moved up to the top of buffer.
E ↵	Ends edit. Edit buffer is copied into temporary file, and files are renamed.
H ↵	Allows work accomplished so far to be saved. Buffer is emptied, temporary file becomes new source file, and a new temporary file is created (equivalent to entering an E command, followed by the ED command with the file specifier for the source file).
O ↵	Returns to original file. Edit buffer is emptied, temporary file is deleted, and edit starts over again on original file.
Q ↵	Quits the edit with no file changes, erases the .BAK file if one existed, and returns control to the operating system.
rfilename* ↵	Reads specified file (whose extension must be .LIB) into the edit buffer. If the file specifier is omitted, X$$$$$$$.LIB is read in. Input file is left intact. This command is useful in inserting boilerplate information into more than one file.
nX ↵	Transfers n lines of data from edit buffer to temporary file to which ED assigns the name X$$$$$$$.LIB.

NOTES: Many ED commands can be preceded by an integer represented in the command format column by n. If no integer is entered, ED assigns a value of 1. If a pound sign (#) is entered for n, 65535 (the largest allowable integer) is assigned.

Commands can be entered in either upper or lower case, unless otherwise noted.

* To avoid translation of all text following an R command into upper case, enter the R command in lower case (see Section 6.2).

Figure 6-1: Edit Buffer/Edit File Interaction



6.6 THE CHARACTER POINTER

Once ASCII characters are in the edit buffer, ED creates a character pointer (CP) in the buffer. The CP, not visible on the screen, is moved throughout the edit buffer under operator control. The location of the CP determines where the commands perform their tasks. The line that contains the CP is called the current line (CL).

The CP points to the imaginary space between two characters (spaces are considered characters); it never points to the character itself. The CP can also point to the beginning of the edit buffer (before the first character) or to the end of the buffer (the end of the last active character).

Always position the CP before editing text, or data may be accidentally lost or modified.

The commands that manipulate the CP or display text in the vicinity of the CP are described in Table 6-2. In the table, an n preceding a command indicates that an optional value can be specified. If no value is specified, ED assigns a value of 1. If the pound sign (#) is entered for n, ED assigns a value of 65535. In the table, commands may be entered preceded by either a plus (+) or a minus (—) sign. If neither sign is entered, ED assigns a +.

Table 6-2: Character-Pointer Related Commands

COMMAND FORMAT	COMMAND FUNCTION
±B	Moves CP to beginning (+) or end (-) of the edit buffer.
±nC	Moves CP n characters toward the beginning (-) or end (+) of buffer. (Carriage return and line feed are counted as two characters.)
±nL	If n equals 0, moves CP to beginning of CL. If n does not equal 0, moves CP to beginning of CL, then up (-) or down (+) n lines. CP stops at top or bottom of edit buffer if n is too large.
±nT	If n equals 0, displays CL up to CP. If n equals 1, displays CL from CP to end of the line. If n is greater than 1, displays CL and the following n-1 lines (+) or the preceding n lines (-). This command does not move the CP. (To display the CL, enter 0TT.)
±n	Moves CP as does the L command, and then displays new CL.
n:	Moves CP to beginning of line number n.
:n	Defines a range of text between the current line and line number n. This does not move the CP.

NOTE: An n preceding a command indicates that an optional value can be specified. If no value is specified, ED assigns a value of 1. If the pound sign (#) is entered for n, ED assigns a value of 65535. In the table, commands preceded by a plus or minus sign (+) may be entered preceded by either a plus (+) or a minus (-) sign to indicate CP movement forward or backward in the file. If neither sign is entered, ED assigns a plus (+).

6.7 MODIFYING TEXT

Once the CP has been positioned, text in the edit buffer can be modified. Single characters or complete lines of text can be deleted. Text from a different ASCII file can be added, or new data can be inserted. Existing strings — groups of contiguous characters — can be replaced with new strings. (Strings are limited to a maximum of 100 characters; a string of 0-length, called a null string, is allowed.) The commands used to modify text are described in Table 6-3.

Table 6-3: Text Modification Commands

COMMAND FORMAT	COMMAND FUNCTION
±nD	Deletes n characters preceding (-) or following (+) CP.
±nK	Deletes n lines preceding (-) or following (+) CL. If CP is not at beginning of line, CL characters preceding (+) or following (-) CP are not deleted.
nfstring or nfstring**Z**	Finds specified string. Use Alt Z if additional commands will be entered in the same line. Search begins at CP. Match is made n times, and CP is positioned after last character in last matched string. If n matches cannot be made, CP remains at its initial position. String can include Alt L, which matches a carriage return/line feed in a string.
jstring1**Z** string2**Z** string 3 or njstring1**Z** string2**Z** string3**Z**	Text between string 1 and string 3 is replaced with string 2. Searches for string 1, inserts string 2, places CP after string 2, and deletes original characters between string 1 and string 3. Search begins at the CP, and substitution is made n times. If string 3 is not found, nothing is deleted.
nmstring or nmstring**Z**	Allows more than one command to be entered and or executed repeatedly. String represents a string of commands, rather than text. If n is greater than 1, command string is executed n times. If n equals 0 or 1, commandstring is executed repeatedly until an error condition is encountered (e.g., end of buffer is reached with F command). For example, the command string MSGAMMA **Z**DELTA **Z**OTT substitutes DELTA for each GAMMA in the buffer, and displays each changed line.
nnstring or nnstring**Z**	Finds specified string. Search begins at current or position of CP. If string cannot be found in edit buffer n times, contents of buffer are written to temporary file (automatic #W command). Input lines from original file are read into edit buffer (automatic A command) until buffer is half full or original file has been completely transferred. Search continues in this manner until string has been found n times (or original file has been completely transferred). This command is used when entire original file cannot fit in edit buffer.
istring or *istring***Z**	Inserts string in front of the CP. (A carriage return without Alt Z is entered after string to put a carriage return/line feed at the end.)
*nsstring1***Z** *string 2* *or nsstring1***Z** *string2***Z**	Searches for string 1 and substitutes string 2 for string 1 n times. Search begins at current position of CP. Search is limited to contents of edit buffer.
Rstring	Inserts file into edit buffer before CP. String represents a file specifier whose extension is assumed to be .LIB. (i.e., the command RMACRO inserts MACRO.LIB file into buffer after CP).

6.8 COMMAND STRING USAGE

Two or more commands can be entered as a string (up to 128 characters); as in the case of single command entries, ED executes them only after a carriage return is entered. Table 6-4 lists the CP/M-86 Alt characters that manipulate the input command string.

Table 6-4: Command String Alternate-Characters

ALTERNATE CHARACTER	FUNCTION
Alt H or Backspace	Deletes the preceding character.
Alt X	Deletes the entire command string.
Alt C	Clears the disk system and re-initializes the CP/M-86 operating system. Returns control to the operating system without saving any of the text modifications that are in the edit buffer.
Alt E	Returns carriage without transmitting command line (128 characters maximum).

6.9 SAMPLE ED SESSION

The following example illustrates use of some of the basic ED functions described in this chapter. Brief explanations of the user entries are shown on the right.

Example:

```
A>ed sample.txt                    Edit a file called SAMPLE.TXT.
NEW FILE
 :  *i                             Insert text into edit buffer.
1:  This is a simple
2:  file to be used
3:  to demonsttrate
4:  how to use
5:  a program called
6:  ED.
7:  Z                              Terminate text insertion.
 :  *b                             Move CP to beginning of buffer.
1:  *#t                            Display entire buffer.
1:  This is a simple
2:  file to be used
3:  to demonsttrate
4:  how to use
5:  a program called
6:  ED.
1:  *ssimZsamZOtt                  Substitute SAM for SIM, changing
                                   simple to sample.
1:  This is a sample
1:  *3:sttZtZOtt                   Move CP to line 3 and change tt to t and then
                                   display corrected line.
3:  to demonstrate
3:  *5:kb#t                        Move CP to line 5 and delete it, then move CP
                                   to beginning of buffer and
                                   display entire contents.
1:  This is a sample
2:  file to be used
3:  to demonstrate
4:  how to use
5:  ED.
1:  *5:i                           Move CP to line 5 and insert new text.
5:  the text Editor
6:  known as
7:  Z
7:  *3:#t                          Move CP to line 3 and display the rest
                                   of the buffer
3:  to demonstrate
4:  how to use
5:  the text Editor
6:  known as
7:  ED.
3:  *3c11dishowZOtt                Move CP 3 columns, delete next 11 characters,
                                   insert "show," display new line.
3:  to show
3:  *b3t                           Move CP to beginning of buffer
                                   and display 3 lines.
1:  This is a sample
2:  file to be used
3:  to show
1:  *2:i                           Move CP to line 2 and insert text.
2:  ASCII text
3:  Z
3:  *b#t                           Move CP to beginning and display entire
                                   buffer.
1:  This is a sample
2:  ASCII text
4:  to show
5:  how to use
6:  the text Editor
7:  known as
8:  ED. Z
1:  *e                             Exit ED.
A>
```

6.10 ED ERROR MESSAGE AND ERROR INDICATORS

When errors occur, ED displays the last character read before the error and an error indicator. The following line shows the error message format:

BREAK errorindicator AT lastcharacter

Table 6-5 lists and describes the ED error indicators.

Table 6-5: ED Error Indicators

ERROR INDICATOR	MEANING
?	Command is not recognized.
>	Memory buffer full (use D, K, N, S, or W command to remove characters); F, N, or S strings are too long.
#	Command cannot be executed number of times specified.
O	LIB file cannot be opened (R command).

6.11 ALTERNATE CHARACTER SUMMARY

Table 6-6 summarizes the alternate characters available in ED.

Table 6-6: Text Editor Alternate-Character Commands

ALTERNATE CHARACTER	FUNCTION
Alt C	Restart (use with extreme care — loss of data may occur).
Alt E	Physical carriage return (not actually entered in command) which allows commands longer than one line to be entered at console.
Alt H	Character delete (same as BACKSPACE key).
Alt I or Tab Key	Logical tab (columns 1, 8, 15, etc.).
Alt L	Logical carriage return/line feed for use in search and substitute strings.
Alt Z	String terminator.

6.12 COMMAND SUMMARY

Table 6-7 summarizes the ED commands in an alphabetical listing.

Table 6-7: Alphabetical List of ED Commands

COMMAND	FUNCTION
nA	Appends lines.
±B	Moves CP to beginning of buffer.
±nC	Moves CP by characters.
±nD	Deletes characters.
E	Ends edit and closes file (normal end).
nf	Finds string.
H	Ends edit; closes and reopens file.
i	Inserts characters.
nj	Places string in juxtaposition (between specified strings).
±nK	Deletes lines.
±nL	Moves CP by lines.
nm	Enters and executes more than one command at a time.
nn	Finds characters throughout original file.
O	Returns to original file.
nP	Displays next 23 lines (1 screen) in buffer.
Q	Quits edit with no file changes.
R	Reads library file into buffer.
nS	Substitutes one string for another.
±nT	Displays lines.
nW	Writes lines into temporary file.
nX	Transfers text from edit buffer to temporary library file.
±n	Moves CP and displays (±nLT).
n:	Moves CP to line n.
±v	Enables/disables line numbering.

APPENDIX A: CP/M-86 COMMAND-FORMAT SUMMARY

Table A-1 contains an alphabetical list of command formats for CP/M-86 commands. For descriptions of ED and PIP subcommand formats, refer to Chapters 5 and 6. Note that the brackets in the PIP command format are literal characters, as described in Section 5.5.2.

Table A-1: CP/M-86 Command Formats

bootcopy [*drivename:* to *drivename:*] ↵

dcopy [*drivename:* to *drivename:*] ↵

dir [*drivename:* | *filespecifier*] ↵

ed *filespecifier* ↵

era *filespecifier* ↵

format *drivename:*[$switch[...]] ↵

pip *destination*=*source*[,*source2*..][[*parameterlist*]] ↵

ren *newfilespecifier*=*oldfilespecifier*

stat [*filespecifier* [{$r/o|$r/w}][{$sys|$dir}] | {vat} | [*drivename:*]dsk: | *drivename:*=r/o | {dev:} | {*logicaldevice*=*physicaldevice*[,..]}] ↵

submit *filespecifier*[*parameterlist*] ↵

type *filespecifier* ↵

NOTE: See Chapters 5 and 6 for descriptions of PIP and ED command formats.

APPENDIX B: CP/M-86 ERROR MESSAGES

Table B-1 presents the CP/M-86 error messages. Note that X: represents the drive name in the error message formats, and n represents a number.

Table B-1: CP/M-86 Error Messages

Message	Meaning
Bdos err on X: bad sector	This can indicate a hardware problem, or a worn, improperly formatted or missing disk. Enter Alt C to terminate the program and return to CP/M-86; then enter Alt C again to reset the disk system. Or press the Return key to ignore the error.
Bdos err on X: R/O	The drive has been assigned read-only status with a STAT command, or the disk in the drive has been changed without resetting the disk system with an Alt C. CP/M-86 terminates the current program as soon as you press any key.
COMMANDNAME?	If CP/M-86 cannot find the command you specified, it returns the command name you entered followed by a question mark. Check that you have typed the command name correctly, or that the command you requested exists as a .CMD file on the default or specified disk.
DESTINATION IS R/O, DELETE (Y/N)?	PIP. The destination file specified in a PIP command already exists and it is read/only. If you type Y, the destination file is deleted before the file copy is done.
DISK READ ERROR: *filespecifier*	PIP. The input disk file specified in a PIP command could not be read properly. This is usually the result of an unexpected end of file. Correct the problem in your file.
DISK WRITE ERROR: filespecifier	PIP. A disk write operation could not be successfully performed during a PIP command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space and execute PIP again.
NO FILE:=*filespecifier*	PIP. Input file specified does not exist.
CANNOT WRITE: *devicename*	PIP. The destination specified in the PIP command is illegal. An input device has probably been specified as a destination.
INVALID FORMAT: invalidentry	PIP. The format of your PIP command is illegal. See the description of the PIP command.
CANNOT READ: *devicename* or *filename*	PIP. The source specified in the PIP command is illegal. An output device has probably been specified as a source.
QUIT NOT FOUND: =*source*	PIP. The string argument to a Q parameter was not found in your input file.
START NOT FOUND:= *source*	PIP. The string argument to an S parameter could not be found in the source file.
ABORTED: *filespecifier*	PIP. The user has aborted a PIP operation by pressing a key.
VERIFY ERROR: *filespecifier*	PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. Usually this indicates a bad copy on the destination disk or drive.
FILE EXISTS	CP/M-86 has been asked to create a new file using a file specification that is already assigned to another file. Either delete the existing file or use another file specifier.
File Not Found	CP/M-86 could not find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive.
Invalid Assignment	STAT. An invalid device was specified in a STAT device assignment. Use the STAT VAL: display to list the valid assignments for each of the four logical STAT devices: CON:, RDR:, PUN:, and LST:. See Section 4.4.
NO FILE	CP/M-86 could not find the specified file, or no files exist.
Invalid File Indicator	STAT. This message results from an invalid command to set file attributes. These are the only options valid in a STAT *filespecifier* [option] command: $r/o, $r/w, $sys, $dir.
Invalid Disk Assignment	STAT. An invalid STAT drive command was given. The only valid drive assignment in STAT is STAT d:=R/O.

APPENDIX C: BDOS ERROR MESSAGES

The SIRIUS 1 Basic Disk Operating System generates the error messages described below. The appropriate recovery procedure for each error situation is also presented. The upper case X: in the error messages represents the name of the drive on which the error occurred.

- Drive=nn, Track=nn, Sector=nn: Error=nn
  Bdos Err On X:=Bad Sector

  The system generates this message in response to a read or write error. The nn represents the hexadecimal number of the drive, track, sector, or error-code number. The significance of the error-code number is explained in Appendix D.

  To correct this error condition, press the Return key and the system will skip the problem sector. If the same error message continues to appear, enter Alt C to abort the operation.

  This error message may also indicate that a non-existent drive has been selected or an attempt has been made to access a drive with no diskette inserted. Enter Alt C to return to the system prompt >; then be sure a diskette is inserted in the correct drive.

- Bdos Err On X: R/O

  The system generates this message when the user attempts to write on a disk in a drive set for read only (R/O) with the STAT command or if the user forgets to enter an Alt C to clear the disk system when inserting a new diskette. To correct this error condition, enter Alt C to return to the operating system. When the prompt appears, enter Alt C again to clear the disk system.

APPENDIX D: ERROR CODES AND DEFINITIONS

Many of the error messages generated under CP/M-86 include error code numbers (in hexadecimal). This appendix defines the error codes. There are four general types of error codes:

1x — Cannot find sector

2x — Cannot read sector

3x — Sector verify error

4x — Formatting error

Table D-1 lists and defines the error codes that CP/M-86 and its utility programs display when an error condition arises.

Table D-1 Sirius Error Code Definitions

ERROR CODE NUMBER	GENERAL DEFINITION / DETAILED DEFINITION
	COULD NOT COMPLETE READ-RELATED COMMAND
11	Noise encountered on sync line
12	Bad header block ID
13	Checksum error in header
14	Header GCR error
15	Wrong track
16	Wrong sector
17	Bad job code
	INVALID DATA ON DISKETTE
21	Bad data block ID
22	Checksum error in data
23	GCR error
24	Sync time out
	DEFECTIVE DRIVE OR DISKETTE
31	Bad data block ID
32	Verify error
33	Checksum error
34	GCR error
	FORMAT PROGRAM ERROR CODES
41	No sync found (bad or missing diskette)
42	Bad header ID
43	Wrong track
44	Wrong sector
45	Bad header checksum
46	Gap error
47	GCR error
48	No data sync
49	Bad data ID
4A	Data verify error
4B	Data checksum
4C	Gap 2 error
4D	GCR error
	MISCELLANEOUS
F3	Data not written due to disk change
F4	Cannot write to disk until logged (C)
F5	Wrong diskette type
F6	Cannot start disk operation
F7	Illegal track number
F8	Illegal drive number
F9	Illegal disk operation
FB	Drive motor not up to speed
FC	Write-protected diskette
FD	Bad track on diskette
FE	Cannot complete disk operation
FF	Bad diskette or unformatted diskette

APPENDIX E: SIRIUS UTILITY ERROR MESSAGES

Table E-1 presents the error messages for the SIRIUS utility programs DCOPY, FORMAT, and BOOTCOPY.

Table E-1: DCopy and Format Error Messages

MESSAGE	MEANING
Bad DCOPY command Bad FORMAT command	The DCOPY or FORMAT command was entered incorrectly. Re-enter the command.
Copy aborted at track xx. Format aborted at track xx.	The disk has a bad track and the program is unable to continue. Run program again, or try moving to another drive. If error persists, discard diskette.
Cannot COPY double-sided diskettes. Cannot Format double-sided diskettes.	The computer being used does not have double sided disk drives.
Sector Write Error: D=xx, T=xx, S=xx, E=xx. CANNOT WRITE DISK LABEL	The program is unable to write the disk label to the disk. The drive, track, sector and error code numbers are indicated.
Operating system version mismatch.	The operating system being used is not compatible with the version of DCOPY or FORMAT. Use the correct version of the program or load the correct operating system.
DRIVE DOOR OPEN	Close the drive door and restart the program.
WRITE PROTECTED DISK	The disk being written to has a write-protect label on it. Remove the label and run the program again. Files on the disk may be altered if you do this.
Bad Drive Number: press any key to continue.	An incorrect drive name was entered.
Read error: D=xx, T=xx, S=xx.E=xx CANNOT READ TRACK	There was a disk read error on the indicated track and sector. Re-attempt the program; discard the diskette if the error persists.
Soft Format error: D=xx, T=xx, S=xx, E=ee	A soft format error of type ee occurred at the indicated track and sector on the indicated drive. If there are more than 9 soft errors on a track, the program will abort. Soft errors in moderate numbers are not harmful to diskette performance.
CANNOT SIZE TRACK	The disk that you are trying to format or copy to may be bad. Run the program again to see if the same error occurs. Discard the diskette if the error persists.
CANNOT FORMAT TRACK	You have a possibly bad disk. Run the program again to see if the same error occurs. Discard the diskette if the error persists.

Table E-1: Bootcopy Error Messages

MESSAGE	MEANING
OPERATING SYSTEM MISMATCH	This version of BOOTCOPY is not compatible with the currently running operating system. Use the correct version of BOOTCOPY or load the correct operating system.
BAD BOOTCOPY COMMAND	The BOOTCOPY command was incorrectly typed. Re-enter the command.
NOT ENOUGH MEMORY	There is not enough memory in the system to do the BOOTCOPY. Boot up a smaller operating system and then run BOOTCOPY or try to build a smaller system.
CANNOT OPEN FILE	BOOTCOPY could not open the input file. Make sure that the file exists and that you spelled the file name correctly.
FILE READ ERROR	BOOTCOPY could not read the input file. The file is probably corrupt. Try to re-create the file.
NO BOOT SECTORS ON SOURCE DISK	The source disk does not have boot tracks. Therefore there is no operating system to copy. Use a source diskette with boot tracks.
NOT A SYSTEM IMAGE	The boot tracks on the source disk do not contain an operating system. Use a system disk.
WRONG DESTINATION DISK LABEL TYPE.	The destination disk has an incorrect software label. Format the disk with the correct version of FORMAT; any files on the disk will be lost.
READ ERROR: Drive=xx, Track=xx, Sector=xx, Error=xx	There was a read or write error on the given track and sector. Try to run the program again to see if the same error occurs.
WRITE ERROR: Drive=xx, Track=xx, Sector=xx, Error=xx	

OFFICES, SUBSIDIARIES, AND DISTRIBUTORS

FRANCE
Sirius Computer S.A.R.L.
28, rue Jean Jaures
92800 Puteaux
Phone: (33) 1-773-8564
Telex: 614764

ITALY
Harden S.p.A. Divisione Elettronica
Via Giuseppina 110
26048 Sospiro (Cremona)
Phone: (39) 372-63136
Telex: 320588

UNITED KINGDOM
ACT (Microsystems) Ltd.
Shenstone House
Dudley Road
Halesowen
West Midlands B63 3NT
Phone: (44) 021-501-2284
Telex: 339396

UNITED STATES
Sirius Systems Technology, Inc.
380 El Pueblo Road
Scotts Valley, CA 95066
Phone: (408) 438-6680
Telex: 357403

WEST GERMANY
Sirius Computer GmbH
Orber Strasse 24
6000 Frankfurt 61
Phone: (49) 611-410223
Telex: 4185558

READER'S COMMENTS FORM

Your comments are a main source of ideas for improvement. Please use this form to provide us with feedback on this document.

DOCUMENT

TITLE: _____

PART NUMBER: _____

YOUR GENERAL REACTION:

Overall quality:	☐ Excellent	☐ Adequate	☐ Poor
Text clarity:	☐ Very clear	☐ Adequate	☐ Difficult
Usefulness of format:	☐ Helpful	☐ Adequate	☐ Inconvenient

YOUR SPECIFIC COMMENTS:

Did you find any errors in the document? _____

If so, describe: _____

_____

_____

Was any important information omitted from the document? _____

If so, describe: _____

_____

_____

What sections of the document were especially useful to you?

_____

_____

What sections were of no use to you? _____

_____

_____

How could material be presented to be more helpful to you?

_____

_____

READER'S NAME: _____

JOB TITLE: _____

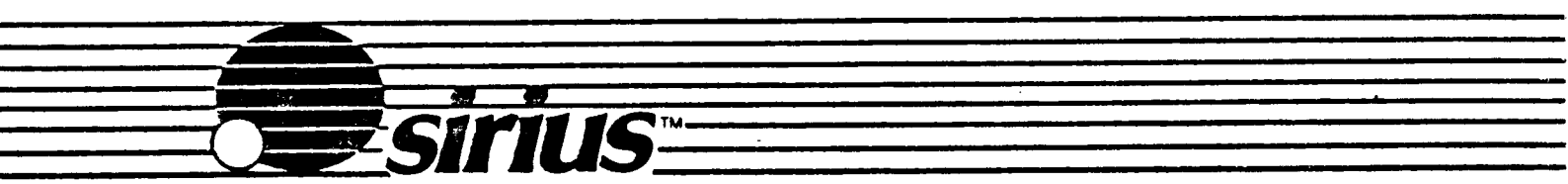
COMPANY: _____

ADDRESS: _____

Please complete and return this form to the office, subsidiary or distributor nearest you.

SIRIUS 1

CP/M-86™
System Guide



CP/M-86™
System Guide

Digital Research
P.O. Box 579
801 Lighthouse Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

The "CP/M-86 System Guide" was prepared using the Digital Research TEX-80™ Text Formatter and printed in the United States of America by Commercial Press/ Monterey.

****
* Second Printing: June 1981 *
****

CPM System Guide

Table of Contents

Appendixes

Appendixes

Foreword

The CP/M-86 System Guide presents the system programming aspects of CP/M-86™, a single-user operating system for the Intel 8086 and 8088 16-bit microprocessors. The discussion assumes the reader is familiar with CP/M the Digital Research 8-bit operating system. To clarify specific differences with CP/M-86, this document refers to the 8-bit version of CP/M as CP/M-80™. Elements common to both systems are simply called CP/M features.

CP/M-80 and CP/M-86 are equivalent at the user interface level and thus the Digital Research documents:

- An Introduction to CP/M Features and Facilities
- ED: A Context Editor for the CP/M Disk System
- CP/M 2 User's Guide

are shipped with the CP/M-86 package. Also included is the CP/M-86 Programmer's Guide, which describes ASM-86™ and DDT-86™, Digital Research's 8086 assembler and interactive debugger.

This System Guide presents an overview of the CP/M-86 programming interface conventions. It also describes procedures for adapting CP/M-86 to a custom hardware enviornment. This information parallels that presented in the CP/M 2 Interface Guide and the CP/M 2 Alteration Guide.

Section 1 gives an overview of CP/M-86 and summarizes its differences with CP/M-80. Section 2 describes the general execution environment while Section 3 tells how to generate command files. Sections 4 and 5 respectively define the programming interfaces to the Basic Disk Operating System and the Basic Input/Output System. Section 6 discusses alteration of the BIOS to support custom disk configurations, and Section 7 describes the loading operation and the organization of the CP/M-86 system file.

Section 1
CP/M-86 System Overview

1.1 CP/M-86 General Characteristics

CP/M-86 contains all facilities of CP/M-80 with additional features to account for increased processor address space of up to a megabyte (1,048,576) of main memory. Further, CP/M-86 maintains file compatibility with all previous versions of CP/M. The file structure of version 2 of CP/M is used, allowing as many as sixteen drives with up to eight megabytes on each drive. Thus, CP/M-80 and CP/M-86 systems may exchange files without modifying the file format.

CP/M-86 resides in the file CPM.SYS, which is loaded into memory by a cold start loader during system initialization. The cold start loader resides on the first two tracks of the system disk. CPM.SYS contains three program modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the user-configurable Basic I/O System (BIOS). The CCP and BDOS portions occupy approximately 10K bytes, while the size of the BIOS varies with the implementation. The operating system executes in any portion of memory above the reserved interrupt locations, while the remainder of the address space is partitioned into as many as eight non-contiguous regions, as defined in a BIOS table. Unlike CP/M-80, the CCP area cannot be used as a data area subsequent to transient program load; all CP/M-86 modules remain in memory at all times, and are not reloaded at a warm start.

Similar to CP/M-80, CP/M-86 loads and executes memory image files from disk. Memory image files are preceded by a "header record," defined in this document, which provides information required for proper program loading and execution. Memory image files under CP/M-86 are identified by a "CMD" file type.

Unlike CP/M-80, CP/M-86 does not use absolute locations for system entry or default variables. The BDOS entry takes place through a reserved software interrupt, while entry to the BIOS is provided by a new BDOS call. Two variables maintained in low memory under CP/M-80, the default disk number and I/O Byte, are placed in the CCP and BIOS, respectively. Dependence upon absolute addresses is minimized in CP/M-86 by maintaining initial "base page" values, such as the default FCB and default command buffer, in the transient program data area.

Utility programs such as ED, PIP, STAT and SUBMIT operate in the same manner under CP/M-86 and CP/M-80. In its operation, DDT-86 resembles DDT supplied with CP/M-80. It allows interactive debugging of 8086 and 8088 machine code. Similarly, ASM-86 allows assembly language programming and development for the 8086 and 8088 using Intel-like mnemonics.

The GENCMD (Generate CMD) utility replaces the LOAD program of CP/M-80, and converts the hex files produced by ASM-86 or Intel utilities into memory image format suitable for execution under CP/M-86. Further, the LDCOPY (Loader Copy) program replaces SYSGEN, and is used to copy the cold start loader from a system disk for replication. In addition, a variation of GENCMD, called LMCMD, converts output from the Intel LOC86 utility into CMD format. Finally, GENDEF (Generate DISKDEF) is provided as an aid in producing custom disk parameter tables. ASM-86, GENCMD, LMCMD, and GENDEF are also supplied in "COM" file format for cross-development under CP/M-80.

Several terms used throughout this manual are defined in Table 1-1 below:

Table 1-1. CP/M-86 Terms

Term	Meaning
Nibble	4-bit half-byte
Byte	8-bit value
Word	16-bit value
Double Word	32-bit value
Paragraph	16 contiguous bytes
Paragraph Boundary	An address divisible evenly by 16 (low order nibble 0)
Segment	Up to 64K contiguous bytes
Segment Register	One of CS, DS, ES, or SS
Offset	16-bit displacement from a segment register
Group	A segment-register-relative relocatable program unit
Address	The effective memory address derived from the composition of a segment register value with an offset value

A group consists of segments that are loaded into memory as a single unit. Since a group may consist of more than 64K bytes, it is the responsibility of the application program to manage segment registers when code or data beyond the first 64K segment is accessed.

CP/M-86 supports eight program groups: the code, data, stack and extra groups as well as four auxiliary groups. When a code, data, stack or extra group is loaded, CP/M-86 sets the respective segment register (CS, DS, SS or ES) to the base of the group. CP/M-86 can also load four auxiliary groups. A transient program manages the location of the auxiliary groups using values stored by CP/M-86 in the user's base page.

1.2 CP/M-80 and CP/M-86 Differences

The structure of CP/M-86 is as close to CP/M-80 as possible in order to provide a familiar programming environment which allows application programs to be transported to the 8086 and 8088 processors with minimum effort. This section points out the specific differences between CP/M-80 and CP/M-86 in order to reduce your time in scanning this manual if you are already familiar with CP/M-80. The terms and concepts presented in this section are explained in detail throughout this manual, so you will need to refer to the Table of Contents to find relevant sections which provide specific definitions and information.

Due to the nature of the 8086 processor, the fundamental difference between CP/M-80 and CP/M-86 is found in the management of the various relocatable groups. Although CP/M-80 references absolute memory locations by necessity, CP/M-86 takes advantage of the static relocation inherent in the 8086 processor. The operating system itself is usually loaded directly above the interrupt locations, at location 0400H, and relocatable transient programs load in the best fit memory region. However, you can load CP/M-86 into any portion of memory without changing the operating system (thus, there is no MOVCPM utility with CP/M-86), and transient programs will load and run in any non-reserved region.

Three general memory models are presented below, but if you are converting 8080 programs to CP/M-86, you can use either the 8080 Model or Small Model and leave the Compact Model for later when your addressing needs increase. You'll use GENCMD, described in Section 3.2, to produce an executable program file from a hex file. GENCMD parameters allow you to specify which memory model your program requires.

CP/M-86 itself is constructed as an 8080 Model. This means that all the segment registers are placed at the base of CP/M-86, and your customized BIOS is identical, in most respects, to that of CP/M-80 (with changes in instruction mnemonics, of course). In fact, the only additions are found in the SETDMAB, GETSEGB, SETIOB, and GETIOB entry points in the BIOS. Your warm start subroutine is simpler since you are not required to reload the CCP and BDOS under CP/M-86. One other point: if you implement the IOBYTE facility, you'll have to define the variable in your BIOS. Taking these changes into account, you need only perform a simple translation of your CP/M-80 BIOS into 8086 code in order to implement your 8086 BIOS.

If you've implemented CP/M-80 Version 2, you already have disk definition tables which will operate properly with CP/M-86. You may wish to attach different disk drives, or experiment with sector skew factors to increase performance. If so, you can use the new GENDEF utility which performs the same function as the DISKDEF macro used by MAC under CP/M-80. You'll find, however, that GENDEF provides you with more information and checks error conditions better than the DISKDEF macro.

Although generating a CP/M-86 system is generally easier than generating a CP/M-80 system, complications arise if you are using single-density floppy disks. CP/M-86 is too large to fit in the two-track system area of a single-density disk, so the bootstrap operation must perform two steps to load CP/M-86: first the bootstrap must load the cold start loader, then the cold start loader loads CP/M-86 from a system file. The cold start loader includes a LDBIOS which is identical to your CP/M-86 BIOS with the exception of the INIT entry point. You can simplify the LDBIOS if you wish because the loader need not write to the disk. If you have a double-density disk or reserve enough tracks on a single-density disk, you can load CP/M-86 without a two-step boot.

To make a BDOS system call, use the reserved software interrupt #244. The jump to the BDOS at location 0005 found in CP/M-80 is not present in CP/M-86. However, the address field at offset 0006 is present so that programs which "size" available memory using this word value will operate without change. CP/M-80 BDOS functions use certain 8080 registers for entry parameters and returned values. CP/M-86 BDOS functions use a table of corresponding 8086 registers. For example, the 8086 registers CH and CL correspond to the 8080 registers B and C. Look through the list of BDOS function numbers in Table 4-2. and you'll find that functions 0, 27, and 31 have changed slightly. Several new functions have been added, but they do not affect existing programs.

One major philosophical difference is that in CP/M-80, all addresses sent to the BDOS are simply 16-bit values in the range 0000H to 0FFFFH. In CP/M-86, however, the addresses are really just 16-bit offsets from the DS (Data Segment) register which is set to the base of your data area. If you translate an existing CP/M-80 program to the CP/M-86 environment, your data segment will be less than 64K bytes. In this case, the DS register need not be changed following initial load, and thus all CP/M-80 addresses become simple DS-relative offsets in CP/M-86.

Under CP/M-80, programs terminate in one of three ways: by returning directly to the CCP, by calling BDOS function 0, or by transferring control to absolute location 0000H. CP/M-86, however, supports only the first two methods of program termination. This has the side effect of not providing the automatic disk system reset following the jump to 0000H which, instead, is accomplished by entering a CONTROL-C at the CCP level.

You'll find many new facilities in CP/M-86 that will simplify your programming and expand your application programming capability. But, we've designed CP/M-86 to make it easy to get started: in short, if you are converting from CP/M-80 to CP/M-86, there will be no major changes beyond the translation to 8086 machine code. Further, programs you design for CP/M-86 are upward compatible with MP/M-86, our multitasking operating system, as well as CP/NET-86 which provides a distributed operating system in a network environment.

Section 2
Command Setup and Execution Under CP/M-86

This section discusses the operation of the Console Command Processor (CCP), the format of transient programs, CP/M-86 memory models, and memory image formats.

2.1 CCP Built-in and Transient Commands

The operation of the CP/M-86 CCP is similar to that of CP/M-80. Upon initial cold start, the CP/M sign-on message is printed, drive A is automatically logged in, and the standard prompt is issued at the console. CP/M-86 then waits for input command lines from the console, which may include one of the built-in commands

DIR ERA REN TYPE USER

(note that SAVE is not supported under CP/M-86 since the equivalent function is performed by DDT-86).

Alternatively, the command line may begin with the name of a transient program with the assumed file type "CMD" denoting a "command file." The CMD file type differentiates transient command files used under CP/M-86 from COM files which operate under CP/M-80.

The CCP allows multiple programs to reside in memory, providing facilities for background tasks. A transient program such as a debugger may load additional programs for execution under its own control. Thus, for example, a background printer spooler could first be loaded, followed by an execution of DDT-86. DDT-86 may, in turn, load a test program for a debugging session and transfer control to the test program between breakpoints. CP/M-86 keeps account of the order in which programs are loaded and, upon encountering a CONTROL-C, discontinues execution of the most recent program activated at the CCP level. A CONTROL-C at the DDT-86 command level aborts DDT-86 and its test program. A second CONTROL-C at the CCP level aborts the background printer spooler. A third CONTROL-C resets the disk system. Note that program abort due to CONTROL-C does not reset the disk system, as is the case in CP/M-80. A disk reset does not occur unless the CONTROL-C occurs at the CCP command input level with no programs residing in memory.

When CP/M-86 receives a request to load a transient program from the CCP or another transient program, it checks the program's memory requirements. If sufficient memory is available, CP/M-86 assigns the required amount of memory to the program and loads the program. Once loaded, the program can request additional memory from the BDOS for buffer space. When the program is terminated, CP/M-86 frees both the program memory area and any additional buffer space.

2.2 Transient Program Execution Models

The initial values of the segment registers are determined by one of three "memory models" used by the transient program, and described in the CMD file header. The three memory models are summarized in Table 2-1 below.

Table 2-1. CP/M-86 Memory Models

Model	Group Relationships
8080 Model	Code and Data Groups Overlap
Small Model	Independent Code and Data Groups
Compact Model	Three or More Independent Groups

The 8080 Model supports programs which are directly translated from CP/M-80 when code and data areas are intermixed. The 8080 model consists of one group which contains all the code, data, and stack areas. Segment registers are initialized to the starting address of the region containing this group. The segment registers can, however, be managed by the application program during execution so that multiple segments within the code group can be addressed.

The Small Model is similar to that defined by Intel, where the program consists of an independent code group and a data group. The Small Model is suitable for use by programs taken from CP/M-80 where code and data is easily separated. Note again that the code and data groups often consist of, but are not restricted to, single 64K byte segments.

The Compact Model occurs when any of the extra, stack, or auxiliary groups are present in program. Each group may consist of one or more segments, but if any group exceeds one segment in size, or if auxiliary groups are present, then the application program must manage its own segment registers during execution in order to address all code and data areas.

The three models differ primarily in the manner in which segment registers are initialized upon transient program loading. The operating system program load function determines the memory model used by a transient program by examining the program group usage, as described in the following sections.

2.3 The 8080 Memory Model

The 8080 Model is assumed when the transient program contains only a code group. In this case, the CS, DS, and ES registers are initialized to the beginning of the code group, while the SS and SP registers remain set to a 96-byte stack area in the CCP. The Instruction Pointer Register (IP) is set to 100H, similar to CP/M-80, thus allowing base page values at the beginning of the code group. Following program load, the 8080 Model appears as shown in Figure 2-1, where low addresses are shown at the top of the diagram:

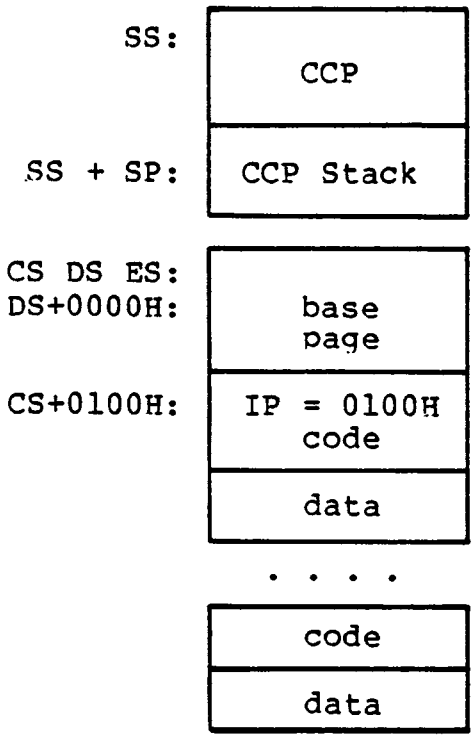


Figure 2-1. CP/M-86 8080 Memory Model

The intermixed code and data regions are indistinguishable. The "base page" values, described below, are identical to CP/M-80, allowing simple translation from 8080, 8085, or Z80 code into the 8086 and 8088 environment. The following ASM-86 example shows how to code an 8080 model transient program.

```
            eseg
            org     100h
            .
            .       (code)
endcs       equ     $
            dseg
            org     offset endcs
            .
            .     (data)
            end
```

2.4 The Small Memory Model

The Small Model is assumed when the transient program contains both a code and data group. (In ASM-86, all code is generated following a CSEG directive, while data is defined following a DSEG directive with the origin of the data segment independent of the code segment.) In this model, CS is set to the beginning of the code group, the DS and ES are set to the start of the data group, and the SS and SP registers remain in the CCP's stack area as shown in Figure 2-2.

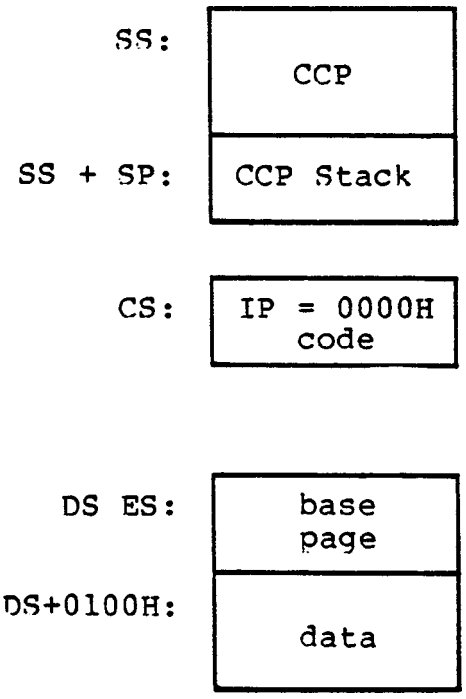


Figure 2-2. CP/M-86 Small Memory Model

The machine code begins at CS+0000H, the "base page" values begin at DS+0000H, and the data area starts at DS+0100H. The following ASM-86 example shows how to code a small model transient program.

```
cseg
.
.       (code)
dseg
org       100h
.
.       (data)
end
```

2.5 The Compact Memory Model

The Compact Model is assumed when code and data groups are present, along with one or more of the remaining stack, extra, or auxiliary groups. In this case, the CS, DS, and ES registers are set to the base addresses of their respective areas. Figure 2-3 shows the initial configuration of segment registers in the Compact Model. The values of the various segment registers can be programmatically changed during execution by loading from the initial values placed in base page by the CCP, thus allowing access to the entire memory space.

If the transient program intends to use the stack group as a stack area, the SS and SP registers must be set upon entry. The SS and SP registers remain in the CCP area, even if a stack group is defined. Although it may appear that the SS and SP registers should be set to address the stack group, there are two contradictions. First, the transient program may be using the stack group as a data area. In that case, the Far Call instruction used by the CCP to transfer control to the transient program could overwrite data in the stack area. Second, the SS register would logically be set to the base of the group, while the SP would be set to the offset of the end of the group. However, if the stack group exceeds 64K the address range from the base to the end of the group exceeds a 16-bit offset value.

The following ASM-86 example shows how to code a compact model transient program.

```
cseg
.
.       (code)
dseg
org     100h
.
.       (data)
eseg
.
.       (more data)
sseg
.
.       (stack area)
end
```

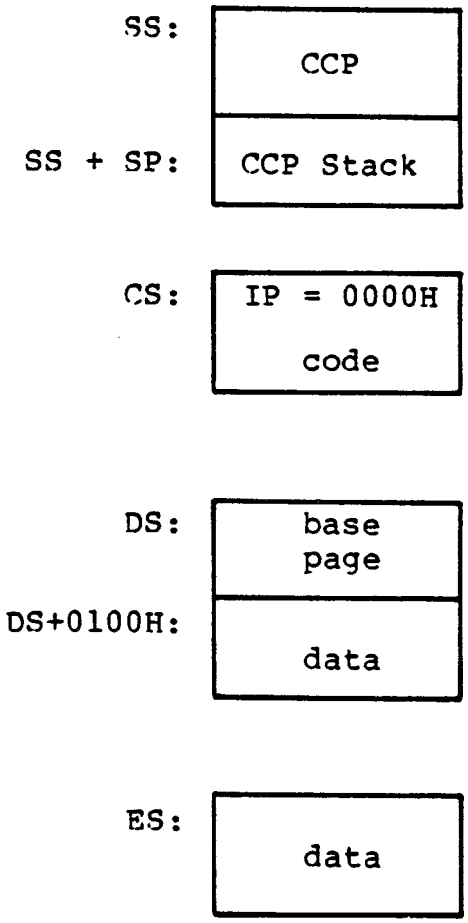


Figure 2-3. CP/M-86 Compact Memory Model

2.6 Base Page Initialization

Similar to CP/M-80, the CP/M-86 base page contains default values and locations initialized by the CCP and used by the transient program. The base page occupies the regions from offset 0000H through 00FFH relative to the DS register. The values in the base page for CP/M-86 include those of CP/M-80, and appear in the same relative positions, as shown in Figure 2-4.

DS + 0000:	LC0	LC1	LC2
DS + 0003:	BC0	BC1	M80
DS + 0006:	LD0	LD1	LD2
DS + 0009:	BD0	BD1	xxx
DS + 000C:	LE0	LE1	LE2
DS + 000F:	BE0	BE1	xxx
DS + 0012:	LS0	LS1	LS2
DS + 0015:	BS0	BS1	xxx
DS + 0018:	LX0	LX1	LX2
DS + 001B:	BX0	BX1	xxx
DS + 001E:	LX0	LX1	LX2
DS + 0021:	BX0	BX1	xxx
DS + 0024:	LX0	LX1	LX2
DS + 0027:	BX0	BX1	xxx
DS + 002A:	LX0	LX1	LX2
DS + 002D:	BX0	BX1	xxx
DS + 0030:	Not Currently Used		
. . .			
DS + 005B:			
DS + 005C:	Default FCB		
DS + 0080:	Default Buffer		
DS + 0100:	Begin User Data		

Figure 2-4. CP/M-86 Base Page Values

Each byte is indexed by 0, 1, and 2, corresponding to the standard Intel storage convention of low, middle, and high-order (most significant) byte. "xxx" in Figure 2-4 marks unused bytes. LC is the last code group location (24-bits, where the 4 high-order bits equal zero).

In the 8080 Model, the low order bytes of LC (LC0 and LC1) never exceed 0FFFFH and the high order byte (LC2) is always zero. BC is base paragraph address of the code group (16-bits). LD and BD provide the last position and paragraph base of the data group. The last position is one byte less than the group length. It should be noted that bytes LD0 and LD1 appear in the same relative positions of the base page in both CP/M-80 and CP/M-86, thus easing the program translation task. The M80 byte is equal to 1 when the 8080 Memory Model is in use. LE and BE provide the length and paragraph base of the optional extra group, while LS and BS give the optional stack group length and base. The bytes marked LX and BX correspond to a set of four optional independent groups which may be required for programs which execute using the Compact Memory Model. The initial values for these descriptors are derived from the header record in the memory image file, described in the following section.

2.7 Transient Program Load and Exit

Similar to CP/M-80, the CCP parses up to two filenames following the command and places the properly formatted FCB's at locations 005CH and 006CH in the base page relative to the DS register. Under CP/M-80, the default DMA address is initialized to 0080H in the base page. Due to the segmented memory of the 8086 and 8088 processors, the DMA address is divided into two parts: the DMA segment address and the DMA offset. Therefore, under CP/M-86, the default DMA base is initialized to the value of DS, and the default DMA offset is initialized to 0080H. Thus, CP/M-80 and CP/M-86 operate in the same way: both assume the default DMA buffer occupies the second half of the base page.

The CCP transfers control to the transient program through an 8086 "Far Call." The transient program may choose to use the 96-byte CCP stack and optionally return directly to the CCP upon program termination by executing a "Far Return." Program termination also occurs when BDOS function zero is executed. Note that function zero can terminate a program without removing the program from memory or changing the memory allocation state (see Section 4.2). The operator may terminate program execution by typing a single CONTROL-C during line edited input which has the same effect as the program executing BDOS function zero. Unlike the operation of CP/M-80, no disk reset occurs and the CCP and BDOS modules are not reloaded from disk upon program termination.

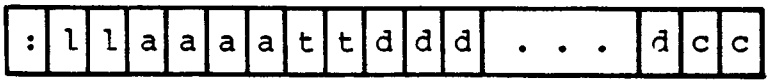
Section 3
Command (CMD) File Generation

As mentioned previously, two utility programs are provided with CP/M-86, called GENCMD and LMCMD, which are used to produce CMD memory image files suitable for execution under CP/M-86. GENCMD accepts Intel 8086 "hex" format files as input, while LMCMD reads Intel L-module files output from the standard Intel LOC86 Object Code Locator utility. GENCMD is used to process output from the Digital Research ASM-86 assembler and Intel's OH86 utility, while LMCMD is used when Intel compatible developmental software is available for generation of programs targeted for CP/M-86 operation.

3.1 Intel 8086 Hex File Format

GENCMD input is in Intel "hex" format produced by both the Digital Research ASM-86 assembler and the standard Intel OH86 utility program (see Intel document #9800639-03 entitled "MCS-86 Software Development Utitities Operating Instructions for ISIS-II Users"). The CMD file produced by GENCMD contains a header record which defines the memory model and memory size requirements for loading and executing the CMD file.

An Intel "hex" file consists of the traditional sequence of ASCII records in the following format:



where the beginning of the record is marked by an ASCII colon, and each subsequent digit position contains an ASCII hexadecimal digit in the range 0-9 or A-F. The fields are defined in Table 3-1.

Table 3-1. Intel Hex Field Definitions

Field	Contents
ll	Record Length 00-FF (0-255 in decimal)
aaaa	Load Address
tt	Record Type: 00 data record, loaded starting at offset aaaa from current base paragraph 01 end of file, cc = FF 02 extended address, aaaa is paragraph base for subsequent data records 03 start address is aaaa (ignored, IP set according to memory model in use) The following are output from ASM-86 only: 81 same as 00, data belongs to code segment 82 same as 00, data belongs to data segment 83 same as 00, data belongs to stack segment 84 same as 00, data belongs to extra segment 85 paragraph address for absolute code segment 86 paragraph address for absolute data segment 87 paragraph address for absolute stack segment 88 paragraph address for absolute extra segment
d	Data Byte
cc	Check Sum (00 - Sum of Previous Digits)

All characters preceding the colon for each record are ignored. (Additional hex file format information is included in the ASM-86 User's Guide, and in Intel's document #9800821A entitled "MCS-86 Absolute Object File Formats.")

3.2 Operation of GENCMD

The GENCMD utility is invoked at the CCP level by typing

```
GENCMD filename parameter-list
```

where the filename corresponds to the hex input file with an assumed (and unspecified) file type of H86. GENCMD accepts optional parameters to specifically identify the 8080 Memory Model and to describe memory requirements of each segment group. The GENCMD parameters are listed following the filename, as shown in the command line above where the parameter-list consists of a sequence of keywords and values separated by commas or blanks. The keywords are:

```
8080  CODE  DATA  EXTRA  STACK  X1  X2  X3  X4
```

The 8080 keyword forces a single code group so that the BDOS load function sets up the 8080 Memory Model for execution, thus allowing intermixed code and data within a single segment. The form of this command is

```
GENCMD filename 8080
```

The remaining keywords follow the filename or the 8080 option and define specific memory requirements for each segment group, corresponding one-to-one with the segment groups defined in the previous section. In each case, the values corresponding to each group are enclosed in square brackets and separated by commas. Each value is a hexadecimal number representing a paragraph address or segment length in paragraph units denoted by hhhh, prefixed by a single letter which defines the meaning of each value:

```
Ahhhh  Load the group at absolute location hhhh
Bhhhh  The group starts at hhhh in the hex file
Mhhhh  The group requires a minimum of hhhh * 16 bytes
Xhhhh  The group can address a maximum of hhhh * 16 bytes
```

Generally, the CMD file header values are derived directly from the hex file and the parameters shown above need not be included. The following situations, however, require the use of GENCMD parameters.

- The 8080 keyword is included whenever ASM-86 is used in the conversion of 8080 programs to the 8086/8088 environment when code and data are intermixed within a single 64K segment, regardless of the use of CSEG and DSEG directives in the source program.

- An absolute address (A value) must be given for any group which must be located at an absolute location. Normally, this value is not specified since CP/M-86 cannot generally ensure that the required memory region is available, in which case the CMD file cannot be loaded.

- The B value is used when GENCMD processes a hex file produced by Intel's OH86, or similar utility program that contains more than one group. The output from OH86 consists of a sequence of data records with no information to identify code, data, extra, stack, or auxiliary groups. In this case, the B value marks the beginning address of the group named by the keyword, causing GENCMD to load data following this address to the named group (see the examples below). Thus, the B value is normally used to mark the boundary between code and data segments when no segment information is included in the hex file. Files produced by ASM-86 do not require the use of the B value since segment information is included in the hex file.

- The minimum memory value (M value) is included only when the hex records do not define the minimum memory requirements for the named group. Generally, the code group size is determined precisely by the data records loaded into the area. That is, the total space required for the group is defined by the range between the lowest and highest data byte addresses. The data group, however, may contain uninitialized storage at the end of the group and thus no data records are present in the hex file which define the highest referenced data item. The highest address in the data group can be defined within the source program by including a "DB 0" as the last data item. Alternatively, the M value can be included to allocate the additional space at the end of the group. Similarly, the stack, extra, and auxiliary group sizes must be defined using the M value unless the highest addresses within the groups are implicitly defined by data records in the hex file.

- The maximum memory size, given by the X value, is generally used when additional free memory may be needed for such purposes as I/O buffers or symbol tables. If the data area size is fixed, then the X parameter need not be included. In this case, the X value is assumed to be the same as the M value. The value XFFFF allocates the largest memory region available but, if used, the transient program must be aware that a three-byte length field is produced in the base page for this group where the high order byte may be non-zero. Programs converted directly from CP/M-80 or programs that use a 2-byte pointer to address buffers should restrict this value to XFFF or less, producing a maximum allocation length of 0FFF0H bytes.

The following GENCMD command line transforms the file X.H86 into the file X.CMD with the proper header record:

```
gencmd x code[a40] data[m30,xfff]
```

In this case, the code group is forced to paragraph address 40H, or equivalently, byte address 400H. The data group requires a minimum of 300H bytes, but can use up to 0FFF0H bytes, if available.

Assuming a file Y.H86 exists on drive B containing Intel hex records with no interspersed segment information, the command

```
gencmd b:y data[b30,m20] extra[b50] stack[m40] xl[m40]
```

produces the file Y.CMD on drive B by selecting records beginning at address 0000H for the code segment, with records starting at 300H allocated to the data segment. The extra segment is filled from records beginning at 500H, while the stack and auxiliary segment #1 are uninitialized areas requiring a minimum of 400H bytes each. In this example, the data area requires a minimum of 200H bytes. Note again, that the B value need not be included if the Digital Research ASM-86 assembler is used.

3.3 Operation of LMCMD

The LMCMD utility operates in exactly the same manner as GENCMD, with the exception that LMCMD accepts an Intel L-module file as input. The primary advantage of the L-module format is that the file contains internally coded information which defines values which would otherwise be required as parameters to GENCMD, such the beginning address of the group's data segment. Currently, however, the only language processors which use this format are the standard Intel development packages, although various independent vendors will, most likely, take advantage of this format in the future.

3.4 Command (CMD) File Format

The CMD file produced by GENCMD and LMCMD consists of the 128-byte header record followed immediately by the memory image. Under normal circumstances, the format of the header record is of no consequence to a programmer. For completeness, however, the various fields of this record are shown in Figure 3-1.

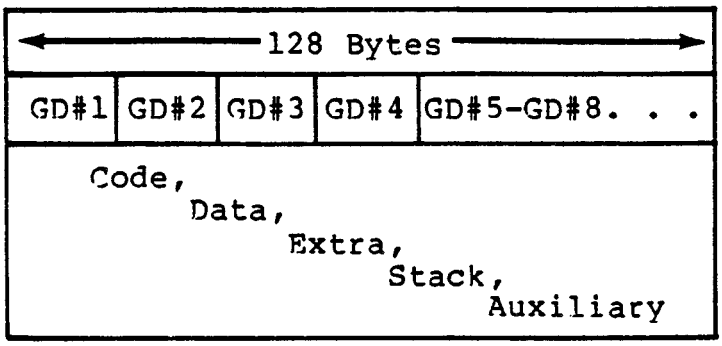
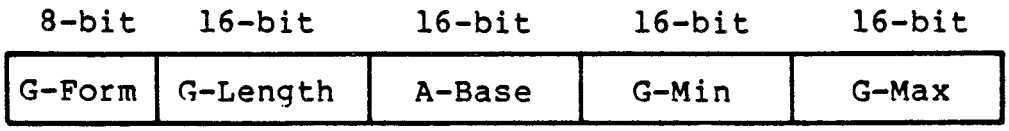
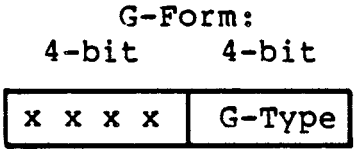


Figure 3-1. CMD File Header Format

In Figure 3-1, GD#2 through GD#8 represent "Group Descriptors." Each Group Descriptor corresponds to an independently loaded program unit and has the following fields:



where G-Form describes the group format, or has the value zero if no more descriptors follow. If G-Form is non-zero, then the 8-bit value is parsed as two fields:



The G-Type field determines the Group Descriptor type. The valid Group Descriptors have a G-Type in the range 1 through 9, as shown in Table 3-2 below.

Table 3-2. Group Descriptors

G-Type	Group Type
1	Code Group
2	Data Group
3	Extra Group
4	Stack Group
5	Auxiliary Group #1
6	Auxiliary Group #2
7	Auxiliary Group #3
8	Auxiliary Group #4
9	Shared Code Group
10 - 14	Unused, but Reserved
15	Escape Code for Additional Types

All remaining values in the group descriptor are given in increments of 16-byte paragraph units with an assumed low-order 0 nibble to complete the 20-bit address. G-Length gives the number of paragraphs in the group. Given a G-length of 0080H, for example, the size of the group is 00800H = 2048D bytes. A-Base defines the base paragraph address for a non-relocatable group while G-Min and G-Max define the minimum and maximum size of the memory area to allocate to the group. G-Type 9 marks a "pure" code group for use under MP/M-86 and future versions of CP/M-86. Presently a Shared Code Group is treated as a non-shared Program Code Group under CP/M-86.

The memory model described by a header record is implicitly determined by the Group Descriptors. The 8080 Memory Model is assumed when only a code group is present, since no independent data group is named. The Small Model is implied when both a code and data group are present, but no additional group descriptors occur. Otherwise, the Compact Model is assumed when the CMD file is loaded.

Section 4
Basic Disk Operating System Functions

This section presents the interface conventions which allow transient program access to CP/M-86 BDOS and BIOS functions. The BDOS calls correspond closely to CP/M-80 Version 2 in order to simplify translation of existing CP/M-80 programs for operation under CP/M-86. BDOS entry and exit conditions are described first, followed by a presentation of the individual BDOS function calls.

4.1 BDOS Parameters and Function Codes

Entry to the BDOS is accomplished through the 8086 software interrupt #224, which is reserved by Intel Corporation for use by CP/M-86 and MP/M-86. The function code is passed in register CL with byte parameters in DL and word parameters in DX. Single byte values are returned in AL, word values in both AX and BX, and double word values in ES and BX. All segment registers, except ES, are saved upon entry and restored upon exit from the BDOS (corresponding to PL/M-86 conventions). Table 4-1 summarizes input and output parameter passing:

Table 4-1. BDOS Parameter Summary

BDOS Entry Registers	BDOS Return Registers
CL Function Code	Byte value returned in AL
DL Byte Parameter	Word value returned in both AX and BX
DX Word Parameter	Double-word value returned with
DS Data Segment	offset in BX and
	segment in ES

Note that the CP/M-80 BDOS requires an "information address" as input to various functions. This address usually provides buffer or File Control Block information used in the system call. In CP/M-86, however, the information address is derived from the current DS register combined with the offset given in the DX register. That is, the DX register in CP/M-86 performs the same function as the DE pair in CP/M-80, with the assumption that DS is properly set. This poses no particular problem for programs which use only a single data segment (as is the case for programs converted from CP/M-80), but when the data group exceeds a single segment, you must ensure that the DS register is set to the segment containing the data area related to the call. It should also be noted that zero values are returned for function calls which are out-of-range.

A list of CP/M-86 calls is given in Table 4-2 with an asterisk following functions which differ from or are added to the set of CP/M-80 Version 2 functions.

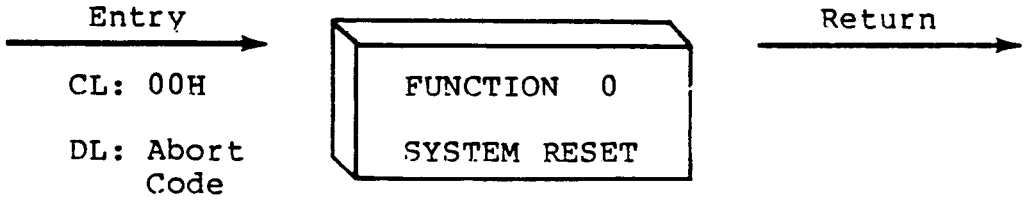
Table 4-2. CP/M-86 BDOS Functions

F#	Result	F#	Result
0*	System Reset	24	Return Login Vector
1	Console Input	25	Return Current Disk
2	Console Output	26	Set DMA Address
3	Reader Input	27*	Get Addr(Alloc)
4	Punch Output	28	Write Protect Disk
5	List Output	29	Get Addr(R/O Vector)
6*	Direct Console I/O	30	Set File Attributes
7	Get I/O Byte	31*	Get Addr(Disk Parms)
8	Set I/O Byte	32	Set/Get User Code
9	Print String	33	Read Random
10	Read Console Buffer	34	Write Random
11	Get Console Status	35	Compute File Size
12	Return Version Number	36	Set Random Record
13	Reset Disk System	37*	Reset drive
14	Select Disk	40	Write Random with Zero Fill
15	Open File	50*	Direct BIOS Call
16	Close File	51*	Set DMA Segment Base
17	Search for First	52*	Get DMA Segment Base
18	Search for Next	53*	Get Max Memory Available
19	Delete File	54*	Get Max Mem at Abs Location
20	Read Sequential	55*	Get Memory Region
21	Write Sequential	56*	Get Absolute Memory Region
22	Make File	57*	Free memory region
23	Rename File	58*	Free all memory
		59*	Program load

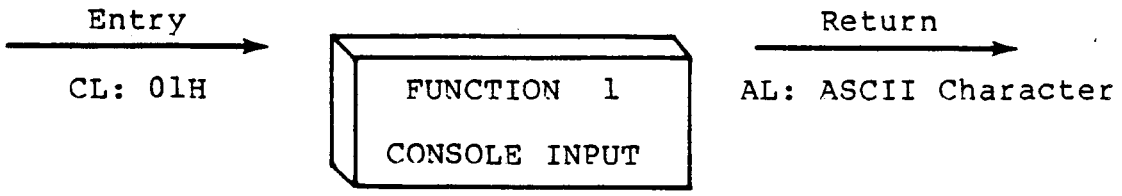
The individual BDOS functions are described below in three sections which cover the simple functions, file operations, and extended operations for memory management and program loading.

4.2 Simple BDOS Calls

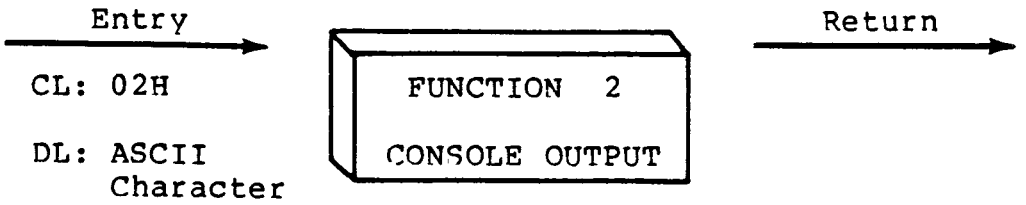
The first set of BDOS functions cover the range 0 through 12, and perform simple functions such as system reset and single character I/O.



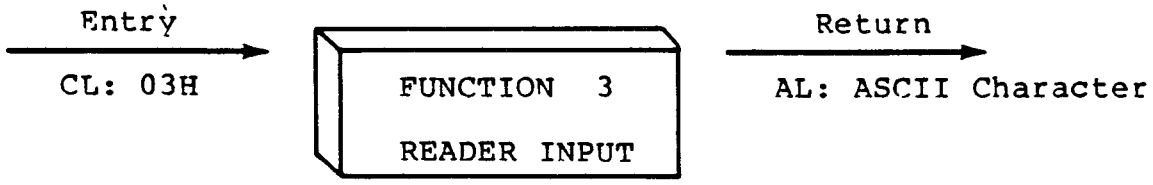
The system reset function returns control to the CP/M operating system at the CCP command level. The abort code in DL has two possible values: if DL = 00H then the currently active program is terminated and control is returned to the CCP. If DL is a 01H, the program remains in memory and the memory allocation state remains unchanged.



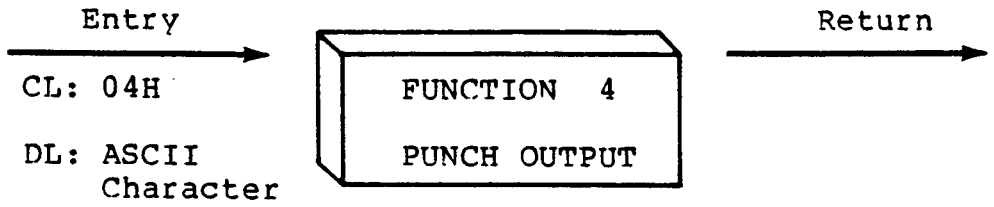
The console input function reads the next character from the logical console device (CONSOLE) to register AL. Graphic characters, along with carriage return, line feed, and backspace (CONTROL-H) are echoed to the console. Tab characters (CONTROL-I) are expanded in columns of eight characters. The BDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.



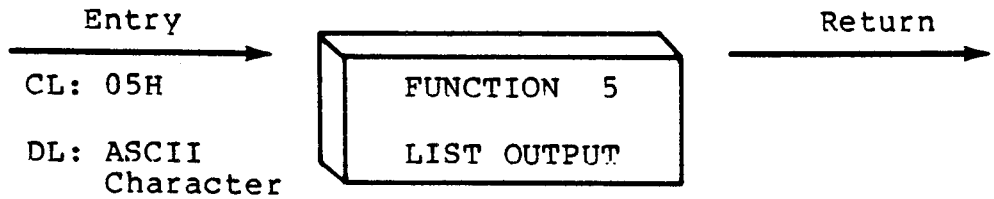
The ASCII character from DL is sent to the logical console. Tab characters expand in columns of eight characters. In addition, a check is made for start/stop scroll (CONTROL-S).



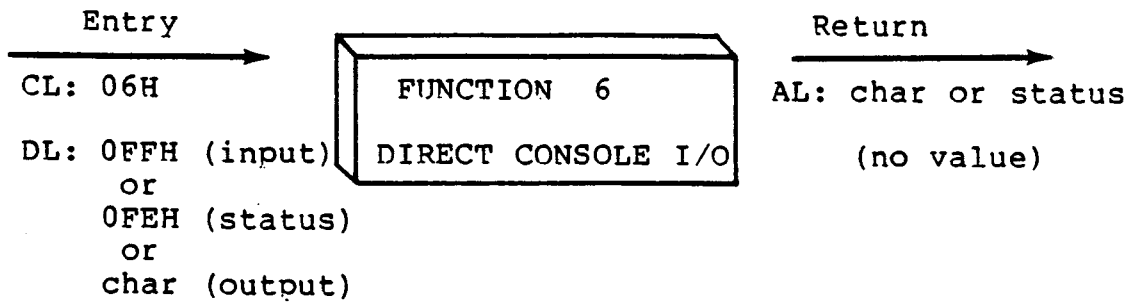
The Reader Input function reads the next character from the logical reader (READER) into register AL. Control does not return until the character has been read.



The Punch Output function sends the character from register DL to the logical punch device (PUNCH).

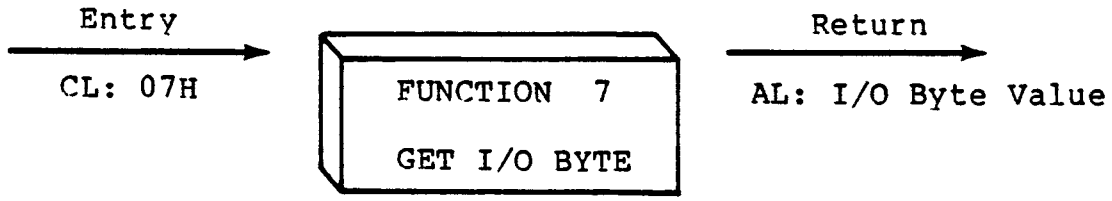


The List Output function sends the ASCII character in register DL to the logical list device (LIST).

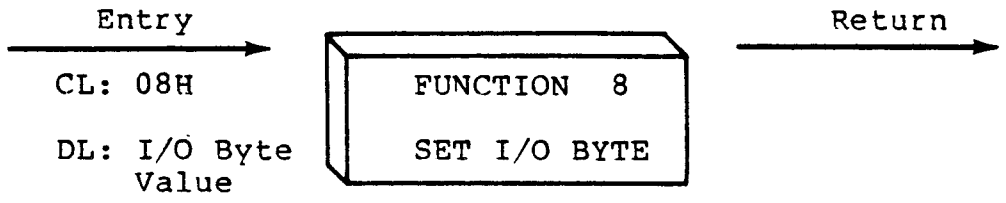


Direct console I/O is supported under CP/M-86 for those specialized applications where unadorned console input and output is required. Use of this function should, in general, be avoided since it bypasses all of CP/M-86's normal control character functions (e.g., CONTROL-S and CONTROL-P). Programs which perform direct I/O through the BIOS under previous releases of CP/M-80, however, should be changed to use direct I/O under the BDOS so that they can be fully supported under future releases of MP/M and CP/M.

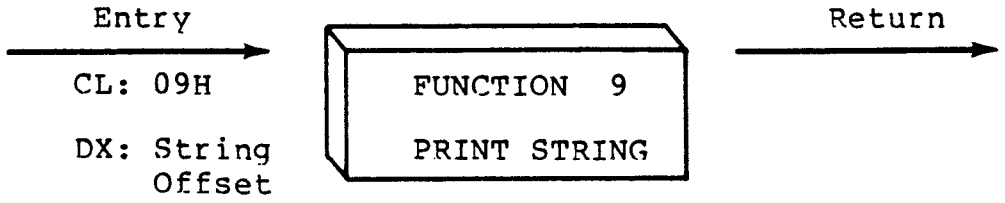
Upon entry to function 6, register DL either contains (1) a hexadecimal FF, denoting a CONSOLE input request, or (2) a hexadecimal FE, denoting a CONSOLE status request, or (3) an ASCII character to be output to CONSOLE where CONSOLE is the logical console device. If the input value is FF, then function 6 directly calls the BIOS console input primitive. The next console input character is returned in AL. If the input value is FE, then function 6 returns AL = 00 if no character is ready and AL = FF otherwise. If the input value in DL is not FE or FF, then function 6 assumes that DL contains a valid ASCII character which is sent to the console.



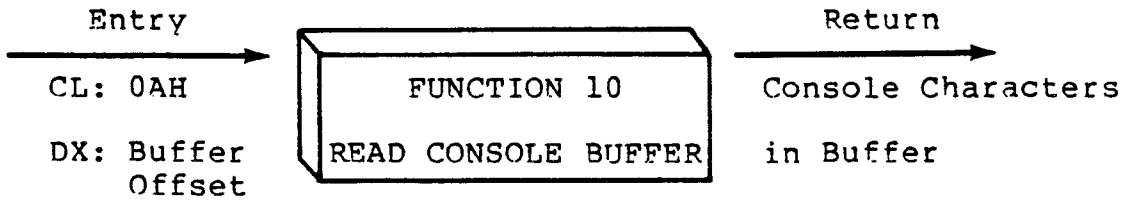
The Get I/O Byte function returns the current value of IOBYTE in register AL. The IOBYTE contains the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST provided the IOBYTE facility is implemented in the BIOS.



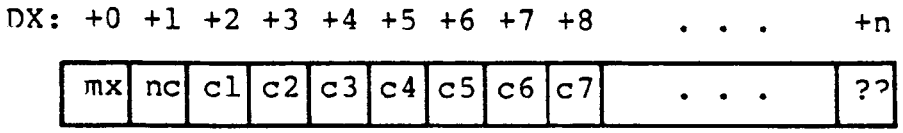
The Set I/O Byte function changes the system IOBYTE value to that given in register DL. This function allows transient program access to the IOBYTE in order to modify the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST.



The Print String function sends the character string stored in memory at the location given by DX to the logical console device (CONSOLE), until a "$" is encountered in the string. Tabs are expanded as in function 2, and checks are made for start/stop scroll and printer echo.



The Read Buffer function reads a line of edited console input into a buffer addressed by register DX from the logical console device (CONSOLE). Console input is terminated when either the input buffer is filled or when a return (CONTROL-M) or a line feed (CONTROL-J) character is entered. The input buffer addressed by DX takes the form:



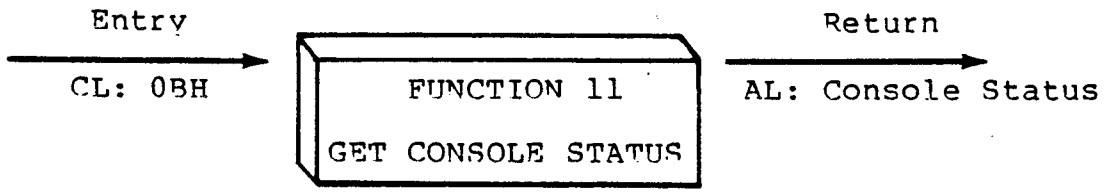
where "mx" is the maximum number of characters which the buffer will hold, and "nc" is the number of characters placed in the buffer. The characters entered by the operator follow the "nc" value. The value "mx" must be set prior to making a function 10 call and may range in value from 1 to 255. Setting mx to zero is equivalent to setting mx to one. The value "nc" is returned to the user and may range from 0 to mx. If nc < mx, then uninitialized positions follow the last character, denoted by "??" in the above figure. Note that a terminating return or line feed character is not placed in the buffer and not included in the count "nc".

A number of editing control functions are supported during console input under function 10. These are summarized in Table 4-3.

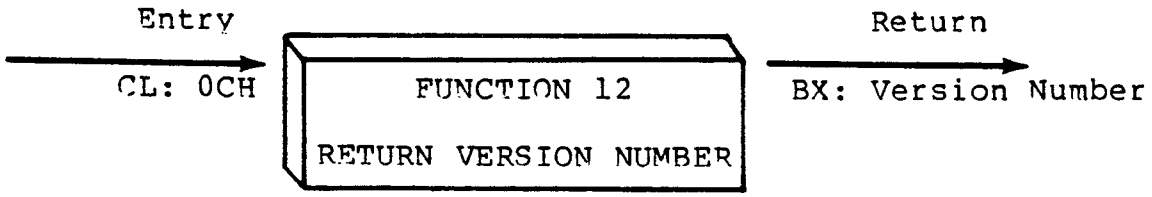
Table 4-3. Line Editing Controls

Keystroke	Result
rub/del	removes and echoes the last character
CONTROL-C	reboots when at the beginning of line
CONTROL-E	causes physical end of line
CONTROL-H	backspaces one character position
CONTROL-J	(line feed) terminates input line
CONTROL-M	(return) terminates input line
CONTROL-R	retypes the current line after new line
CONTROL-U	removes current line after new line
CONTROL-X	backspaces to beginning of current line

Certain functions which return the carriage to the leftmost position (e.g., CONTROL-X) do so only to the column position where the prompt ended. This convention makes operator data input and line correction more legible.



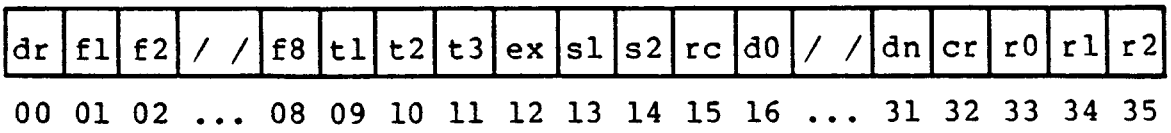
The Console Status function checks to see if a character has been typed at the logical console device (CONSOLE). If a character is ready, the value 01H is returned in register AL. Otherwise a 00H value is returned.



Function 12 provides information which allows version independent programming. A two-byte value is returned, with BH = 00 designating the CP/M release (BH = 01 for MP/M), and BL = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register BL, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. To provide version number compatibility, the initial release of CP/M-86 returns a 2.2.

4.3 BDOS File Operations

Functions 12 through 52 are related to disk file operations under CP/M-86. In many of these operations, DX provides the DS-relative offset to a file control block (FCB). The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access, or a sequence of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at offset 005CH from the DS register can be used for random access files, since bytes 007DH, 007EH, and 007FH are available for this purpose. Here is the FCB format, followed by definitions of each of its fields:



where

dr — drive code (0 - 16)
0 => use default drive for file
1 => auto disk select drive A,
2 => auto disk select drive B,
...
16=> auto disk select drive P.

f1...f8 — contain the file name in ASCII upper case, with high bit = 0

t1,t2,t3 — contain the file type in ASCII upper case, with high bit = 0
t1′, t2′, and t3′ denote the high bit of these positions,
t1′ = 1 => Read/Only file,
t2′ = 1 => SYS file, no DIR list

ex — contains the current extent number, normally set to 00 by the user, but in range 0 - 31 during file I/O

s1 — reserved for internal system use

s2 — reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH

rc — record count for extent "ex," takes on values from 0 - 128

d0...dn — filled-in by CP/M, reserved for system use

cr — current record to read or write in a sequential file operation, normally set to zero by user

r0,r1,r2 — optional random record number in the range 0-65535, with overflow to r2, r0,r1 constitute a 16-bit value with low byte r0, and high byte r1

For users of earlier versions of CP/M, it should be noted in passing that both CP/M Version 2 and CP/M-86 perform directory operations in a reserved area of memory that does not affect write buffer content, except in the case of Search and Search Next where the directory record is copied to the current DMA address.

There are three error situations that the BDOS may encounter during file processing, initiated as a result of a BDOS File I/O function call. When one of these conditions is detected, the BDOS issues the following message to the console:

BDOS ERR ON x: error

where x is the drive name of the drive selected when the error condition is detected, and "error" is one of the three messages:

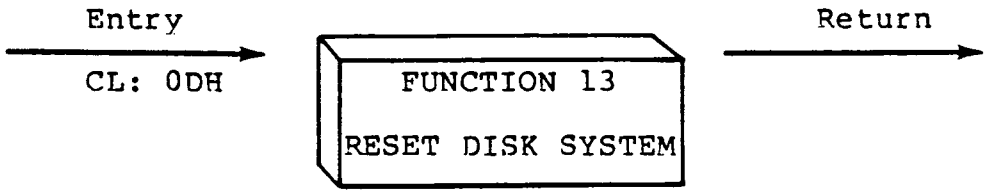
BAD SECTOR SELECT R/O

These error situations are trapped by the BDOS, and thus the executing transient program is temporarily halted when the error is detected. No indication of the error situation is returned to the transient program.

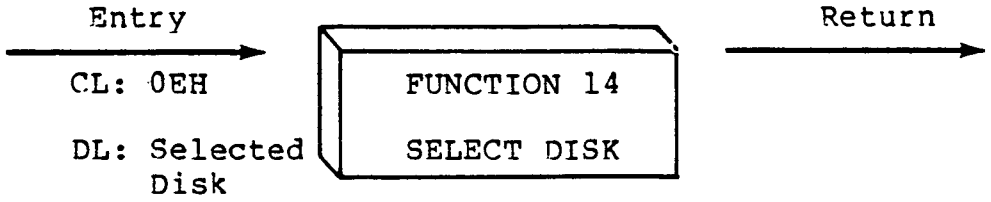
The "BAD SECTOR" error is issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes BIOS sector read and write commands as part of the execution of BDOS file related system calls. If the BIOS read or write routine detects a hardware error, it returns an error code to the BDOS resulting in this error message. The operator may respond to this error in two ways: a CONTROL-C terminates the executing program, while a RETURN instructs CP/M-86 to ignore the error and allow the program to continue execution.

The "SELECT" error is also issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS disk select call prior to issuing any BIOS read or write to a particular drive. If the selected drive is not supported in the BIOS module, it returns an error code to the BDOS resulting in this error message. CP/M-86 terminates the currently running program and returns to the command level of the CCP following any input from the console.

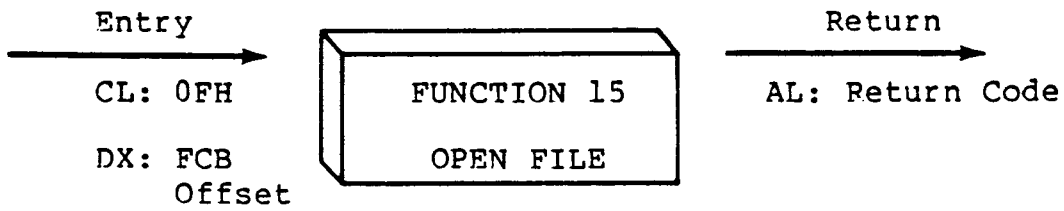
The "R/O" message occurs when the BDOS receives a command to write to a drive that is in read-only status. Drives may be placed in read-only status explicitly as the result of a STAT command or BDOS function call, or implicitly if the BDOS detects that disk media has been changed without performing a "warm start." The ability to detect changed media is optionally included in the BIOS, and exists only if a checksum vector is included for the selected drive. Upon entry of any character at the keyboard, the transient program is aborted, and control returns to the CCP.



The Reset Disk Function is used to programmatically restore the file system to a reset state where all disks are set to read/write (see functions 28 and 29), only disk drive A is selected. This function can be used, for example, by an application program which requires disk changes during operation. Function 37 (Reset Drive) can also be used for this purpose.

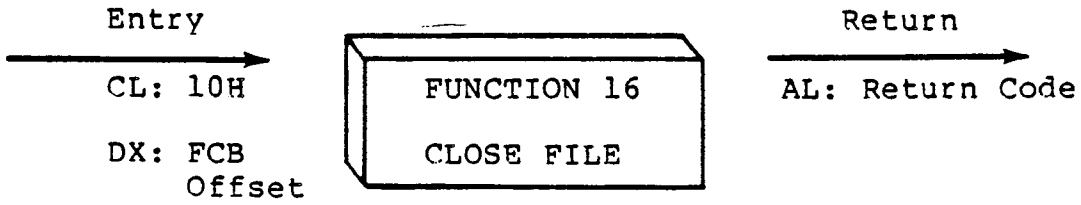


The Select Disk function designates the disk drive named in register DL as the default disk for subsequent file operations, with DL = 0 for drive A, 1 for drive B, and so-forth through 15 corresponding to drive P in a full sixteen drive system. In addition, the designated drive is logged-in if it is currently in the reset state. Logging-in a drive places it in "on-line" status which activates the drive's directory until the next cold start, warm start, disk system reset, or drive reset operation. FCB's which specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

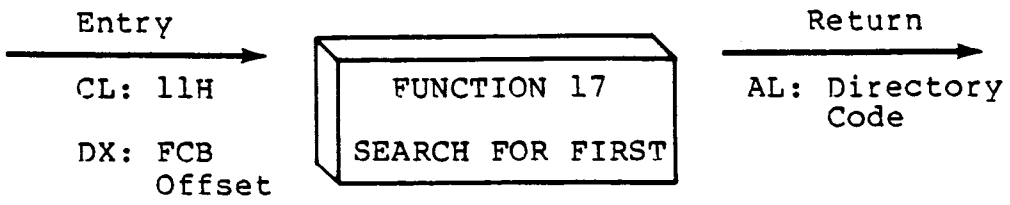


The Open File operation is used to activate a FCB specifying a file which currently exists in the disk directory for the currently active user number. The BDOS scans the disk directory of the drive specified by byte 0 of the FCB referenced by DX for a match in positions 1 through 12 of the referenced FCB, where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included and, further, byte "ex" of the FCB is set to zero before making the open call.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of the FCB, thus allowing access to the files through subsequent read and write operations. Note that an existing file must not be accessed until a successful open operation is completed. Further, an FCB not activated by either an open or make function must not be used in BDOS read or write commands. Upon return, the open function returns a "directory code" with the value 0 through 3 if the open was successful, or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB then the first matching FCB is activated. Note that the current record ("cr") must be zeroed by the program if the file is to be accessed sequentially from the first record.

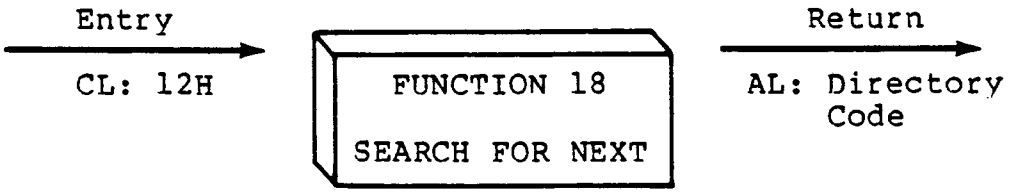


The Close File function performs the inverse of the open file function. Given that the FCB addressed by DX has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to permanently record the new directory information.

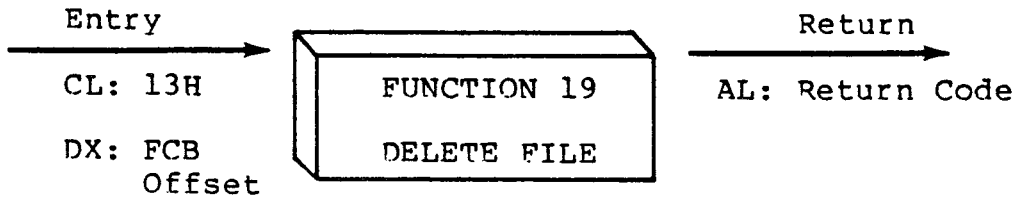


Search First scans the directory for a match with the file given by the FCB addressed by DX. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the buffer at the current DMA address is filled with the record containing the directory entry, and its relative starting position is AL * 32 (i.e., rotate the AL register left 5 bits). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

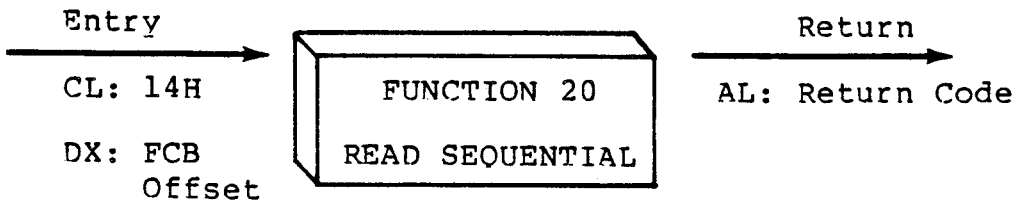
An ASCII question mark (63 decimal, 3F hexadecimal) in any position from "f1" through "ex" matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the "dr" field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the "dr" field is not a question mark, the "s2" byte is automatically zeroed.



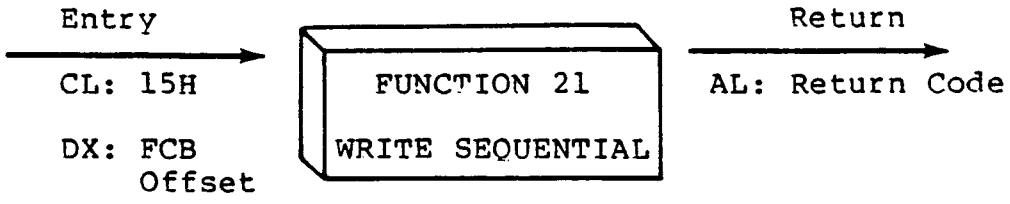
The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match. In terms of execution sequence, a function 18 call must follow either a function 17 or function 18 call with no other intervening BDOS disk related function calls.



The Delete File function removes files which match the FCB addressed by DX. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions. Function 19 returns a 0FFH (decimal 255) if the referenced file or files cannot be found, otherwise a value of zero is returned.



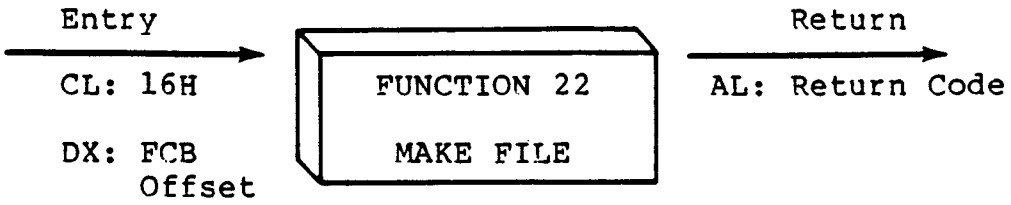
Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128 byte record from the file into memory at the current DMA address. The record is read from position "cr" of the extent, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next read operation. The "cr" field must be set to zero following the open call by the user if the intent is to read sequentially from the beginning of the file. The value 00H is returned in the AL register if the read operation was successful, while a value of 01H is returned if no data exists at the next record position of the file. Normally, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block which has not been previously written, or an extent which has not been created. These situations are usually restricted to files created or appended by use of the BDOS Write Random commmand (function 34).



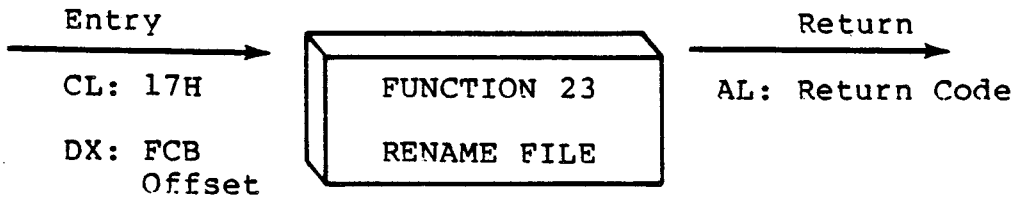
Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128 byte data record at the current DMA address to the file named by the FCB. The record is placed at position "cr" of the file, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly written records overlay those which already exist in the file. The "cr" field must be set to zero following an open or make call by the user if the intent is to write sequentially from the beginning of the file. Register AL = 00H upon return from a successful write operation, while a non-zero value indicates an unsuccessful write due to one of the following conditions:

01 No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

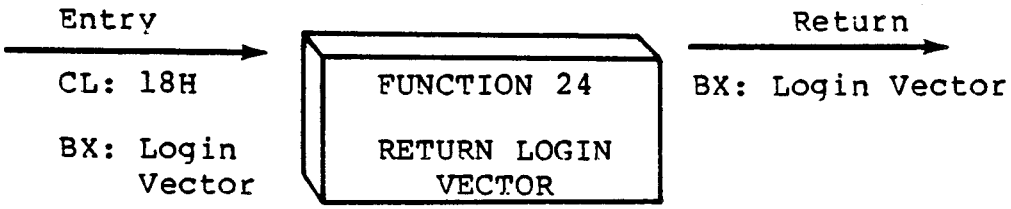
02 No available data block - This condition is encountered when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.



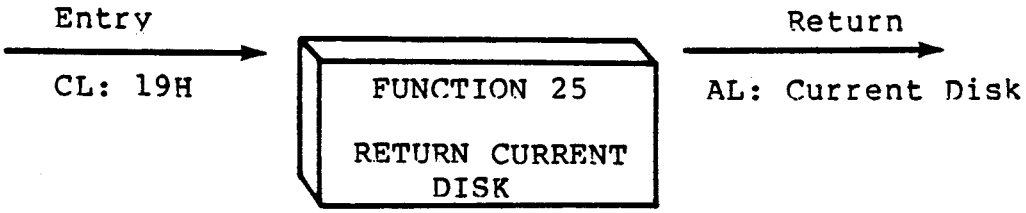
The Make File operation is similar to the open file operation except that the FCB must name a file which does not exist in the currently referenced disk directory (i.e., the one named explicitly by a non-zero "dr" code, or the default disk if "dr" is zero). The BDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The make function has the side-effect of activating the FCB and thus a subsequent open is not necessary.



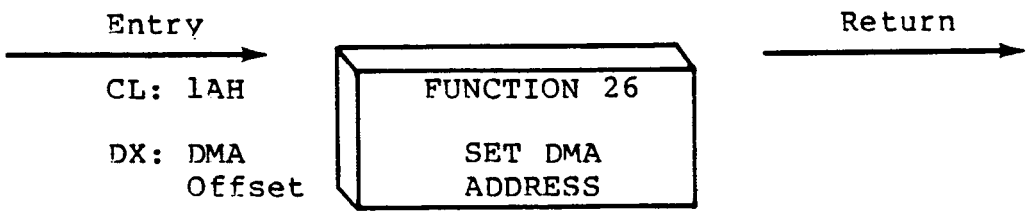
The Rename function uses the FCB addressed by DX to change all directory entries of the file specified by the file name in the first 16 bytes of the FCB to the file name in the second 16 bytes. It is the user's responsibility to insure that the file names specified are valid CP/M unambiguous file names. The drive code "dr" at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is ignored. Upon return, register AL is set to a value of zero if the rename was successful, and 0FFH (255 decimal) if the first file name could not be found in the directory scan.



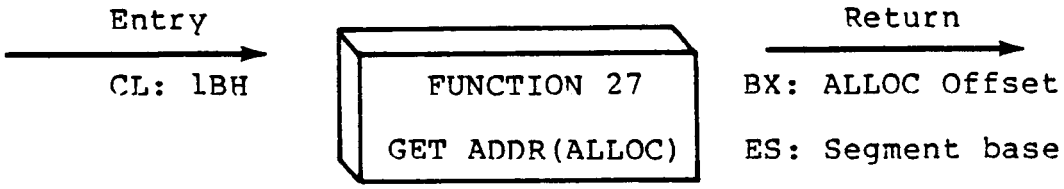
The login vector value returned by CP/M-86 is a 16-bit value in BX, where the least significant bit corresponds to the first drive A, and the high order bit corresponds to the sixteenth drive, labelled P. A "0" bit indicates that the drive is not on-line, while a "1" bit marks an drive that is actively on-line due to an explicit disk drive selection, or an implicit drive select caused by a file operation which specified a non-zero "dr" field.



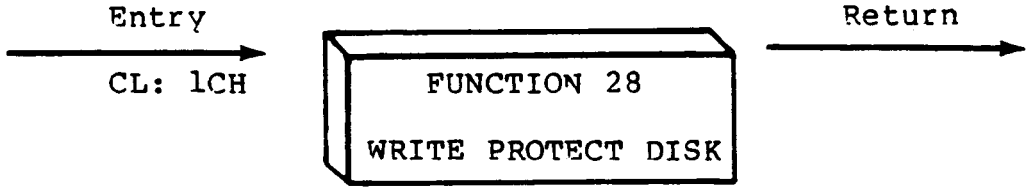
Function 25 returns the currently selected default disk number in register AL. The disk numbers range from 0 through 15 corresponding to drives A through P.



"DMA" is an acronym for Direct Memory Address, which is often used in connection with disk controllers which directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data is transfered through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128 byte data record resides before a disk write and after a disk read. In the CP/M-86 environment, the Set DMA function is used to specify the offset of the read or write buffer from the current DMA base. Therefore, to specify the DMA address, both a function 26 call and a function 51 call are required. Thus, the DMA address becomes the value specified by DX plus the DMA base value until it is changed by a subsequent Set DMA or set DMA base function.

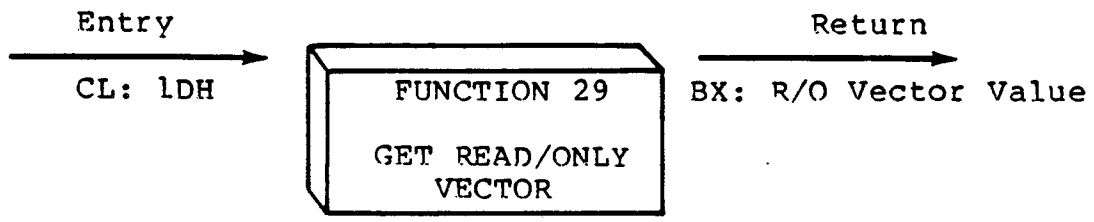


An "allocation vector" is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the segment base and the offset address of the allocation vector for the currently selected disk drive. The allocation information may, however, be invalid if the selected disk has been marked read/only.

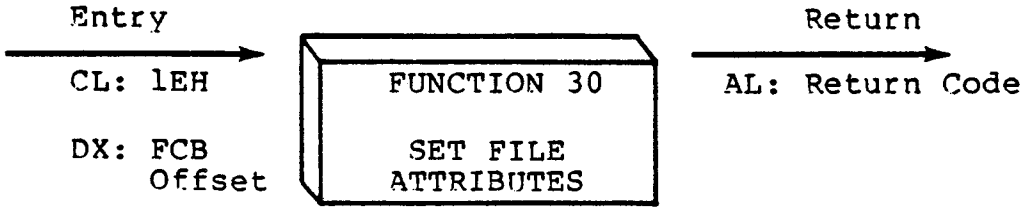


The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold start, warm start, disk system reset, or drive reset operation produces the message:

```
Bdos Err on d: R/O
```



Function 29 returns a bit vector in register BX which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M-86 which detect changed disks.



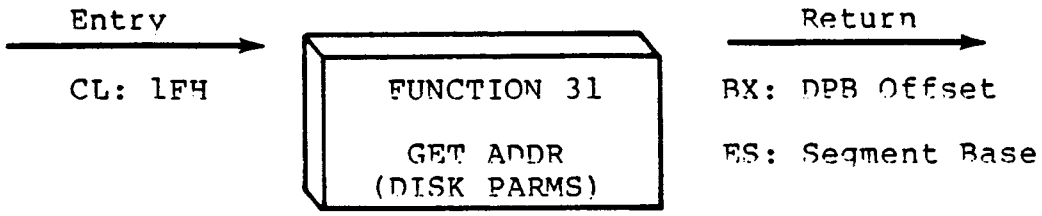
The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O, System and Archive attributes (t1', t2', and t3') can be set or reset. The DX pair addresses a FCB containing a file name with the appropriate attributes set or reset. It is the user's responsibility to insure that an ambiguous file name is not specified. Function 30 searches the default disk drive directory area for directory entries that belong to the current user number and that match the FCB specified name and type fields. All matching directory entries are updated to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' are reserved for future system expansion. The currently assigned attributes are defined as follows:

t1': The R/O attribute indicates if set that the file is in read/only status. BDOS will not allow write commands to be issued to files in R/O status.

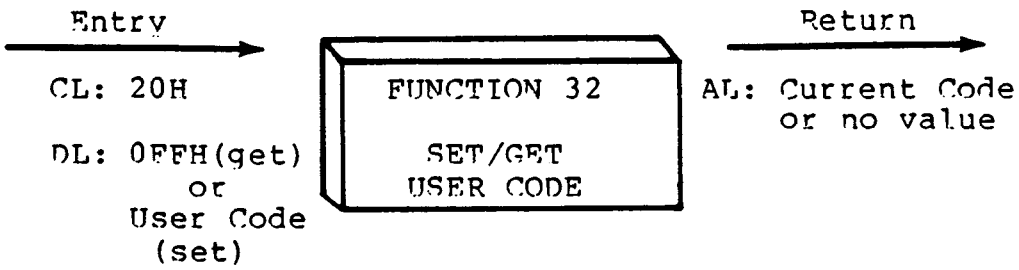
t2': The System attribute is referenced by the CP/M DIR utility. If set, DIR will not display the file in a directory display.

t3': The Archive attribute is reserved but not actually used by CP/M-86 If set it indicates that the file has been written to back up storage by a user written archive program. To implement this facility, the archive program sets this attribute when it copies a file to back up storage; any programs updating or creating files reset this attribute. Further, the archive program backs up only those files that have the Archive attribute reset. Thus, an automatic back up facility restricted to modified files can be easily implemented.

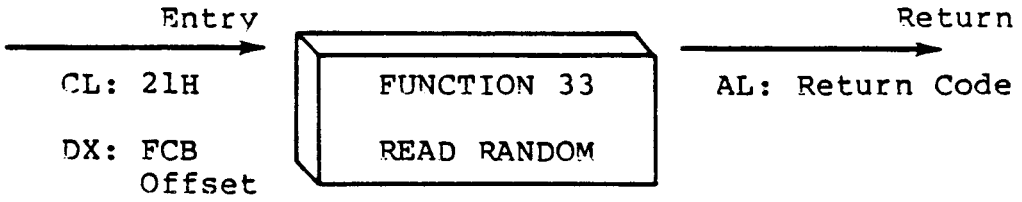
Function 30 returns with register AL set to 0FFH (255 decimal) if the referenced file cannot be found, otherwise a value of zero is returned.



The offset and the segment base of the BIOS resident disk parameter block of the currently selected drive are returned in BX and ES as a result of this function call. This control block can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility. Section 6.3 defines the BIOS disk parameter block.



An application program can change or interrogate the currently active user number by calling function 32. If register DL = 0FFH, then the value of the current user number is returned in register AL, where the value is in the range 0 to 15. If register DL is not 0FFH, then the current user number is changed to the value of DL (modulo 16).



The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of any size file. In order to access a file using the Read Random function, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the FCB is properly initialized for subsequent random access operations. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register AL either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the buffer at the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

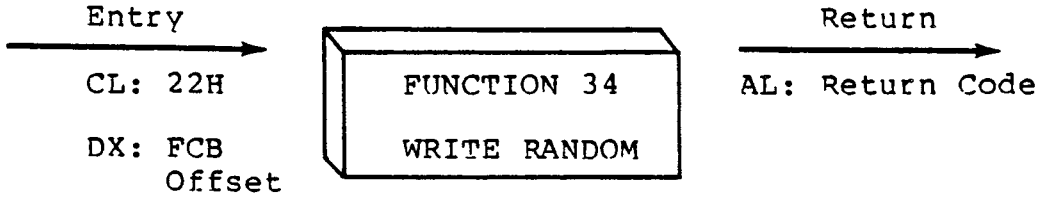
Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register AL following a random read are listed in Table 4-4, below.

Table 4-4. Function 33 (Read Random) Error Codes

Code	Meaning
01	Reading unwritten data - This error code is returned when a random read operation accesses a data block which has not been previously written.
02	(not returned by the Random Read command)
03	Cannot close current extent - This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a read random operation on an FCB that has not been opened.
04	Seek to unwritten extent - This error code is returned when a random read operation accesses an extent that has not been created. This error situation is equivalent to error 01.
05	(not returned by the Random Read command)
06	Random record number out of range - This error code is returned whenever byte r2 of the FCB is non-zero.

Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.



The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Sequential read or write operations can commence following a random write, with the note that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. In particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

In order to access a file using the Write Random function, the base extent (extent 0) must first be opened. As in the Read Random function, this ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, a Make File function must be issued for the base extent. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests.

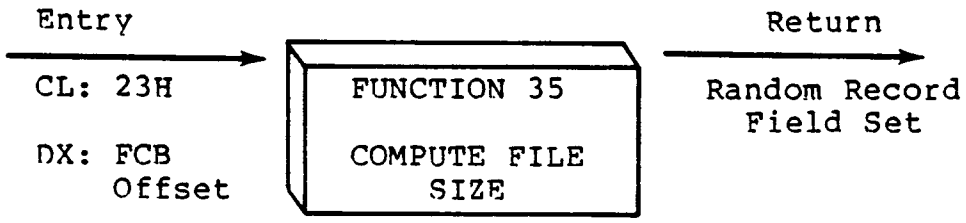
Upon return from a Write Random call, register AL either contains an error code, as listed in Table 4-5 below, or the value 00 indicating the operation was successful.

Table 4-5. Function 34 (WRITE RANDOM) Error Codes

Code	Meaning
01	(not returned by the Random Write command)
02	No available data block - This condition is encountered when the Write Random command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

Table 4-5. (continued)

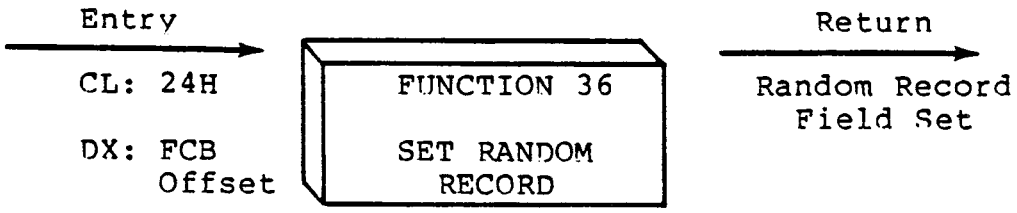
Code	Meaning
03	Cannot close current extent - This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a write random operation on an FCB that has not been opened.
04	(not returned by the Random Write command)
05	No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.
06	Random record number out of range - This error code is returned whenever byte r2 of the FCB is non-zero.



When computing the size of a file, the DX register addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

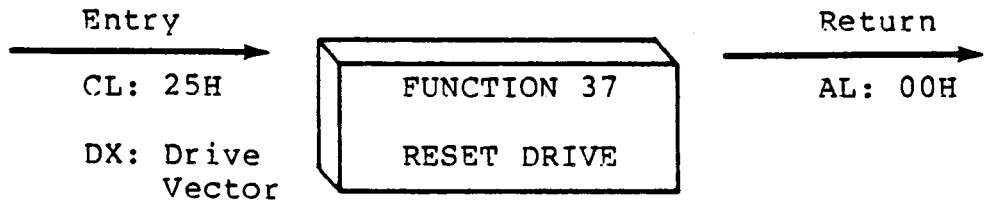
The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, a single record with record number 65535 (CP/M's maximum record number) is written to a file using the Write Random function, then the virtual size of the file is 65536 records, although only one block of data is actually allocated.



The Set Random Record function causes the BDOS to automatically produce the random record position of the next record to be accessed from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

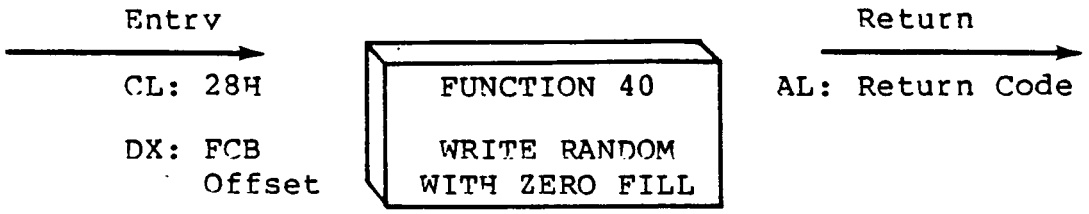
First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position minus one is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the next record in the file.

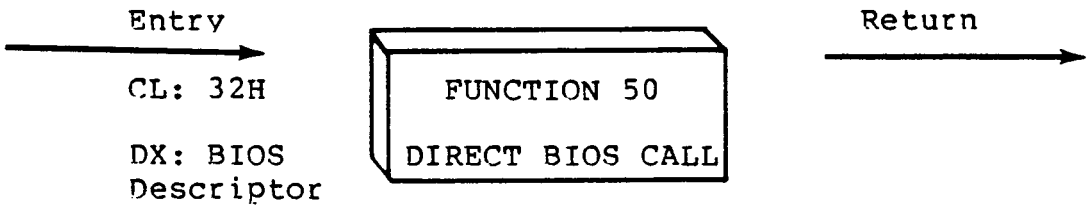


The Reset Drive function is used to programmatically restore specified drives to the reset state (a reset drive is not logged-in and is in read/write status). The passed parameter in register DX is a 16 bit vector of drives to be reset, where the least significant bit corresponds to the first drive, A, and the high order bit corresponds to the sixteenth drive, labelled P. Bit values of "1" indicate that the specified drive is to be reset.

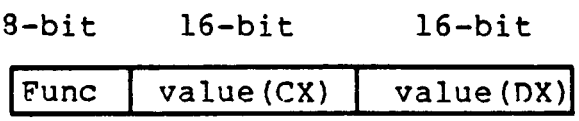
In order to maintain compatibility with MP/M, CP/M returns a zero value for this function.



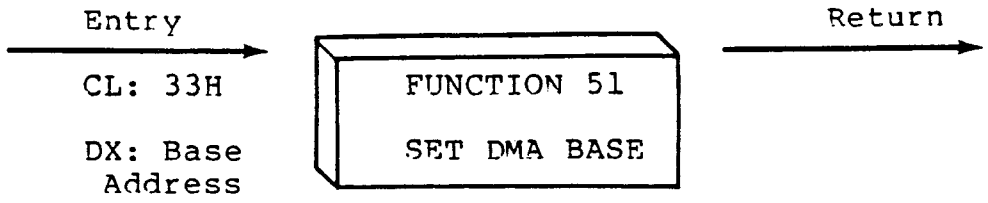
The Write Random With Zero Fill function is similar to the Write Random function (function 34) with the exception that a previously unallocated data block is initialized to records filled with zeros before the record is written. If this function has been used to create a file, records accessed by a read random operation that contain all zeros identify unwritten random record numbers. Unwritten random records in allocated data blocks of files created using the Write Random function contain uninitialized data.



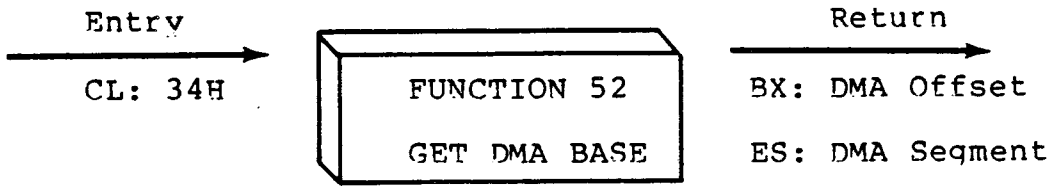
Function 50 provides a direct BIOS call and transfers control through the BDOS to the BIOS. The DX register addresses a five-byte memory area containing the BIOS call parameters:



where Func is a BIOS function number, (see Table 5-1), and value(CX) and value(DX) are the 16-bit values which would normally be passed directly in the CX and DX registers with the BIOS call. The CX and DX values are loaded into the 8086 registers before the BIOS call is initiated.



Function 51 sets the base register for subsequent DMA transfers. The word parameter in DX is a paragraph address and is used with the DMA offset to specify the address of a 128 byte buffer area to be used in the disk read and write functions. Note that upon initial program loading, the default DMA base is set to the address of the user's data segment (the initial value of DS) and the DMA offset is set to 0080H, which provides access to the default buffer in the base page.



Function 52 returns the current DMA Base Segment address in ES, with the current DMA Offset in DX.

4.4 BDOS Memory Management and Load

Memory is allocated in two distinct ways under CP/M-86. The first is through a static allocation map, located within the BIOS, that defines the physical memory which is available on the host system. In this way, it is possible to operate CP/M-86 in a memory configuration which is a mixture of up to eight non-contiguous areas of RAM or ROM, along with reserved, missing, or faulty memory regions. In a simple RAM-based system with contiguous memory, the static map defines a single region, usually starting at the end of the BIOS and extending up to the end of available memory.

Once memory is physically mapped in this manner, CP/M-86 performs the second level of dynamic allocation to support transient program loading and execution. CP/M-86 allows dynamic allocation of memory into, again, eight regions. A request for allocation takes place either implicitly, through a program load operation, or explicitly through the BDOS calls given in this section. Programs themselves are loaded in two ways: through a command entered at the CCP level, or through the BDOS Program Load operation (function 59). Multiple programs can be loaded at the CCP level, as long as each program executes a System Reset (function 0) and remains in memory (DL = 01H). Multiple programs of this type only receive control by intercepting interrupts, and thus under normal circumstances there

is only one transient program in memory at any given time. If, however, multiple programs are present in memory, then CONTROL-C characters entered by the operator delete these programs in the opposite order in which they were loaded no matter which program is actively reading the console.

Any given program loaded through a CCP command can, itself, load additional programs and allocate data areas. Suppose four regions of memory are allocated in the following order: a program is loaded at the CCP level through an operator command. The CMD file header is read, and the entire memory image consisting of the program and its data is loaded into region A, and execution begins. This program, in turn, calls the BDOS Program Load function (59) to load another program into region B, and transfers control to the loaded program. The region B program then allocates an additional region C, followed by a region D. The order of allocation is shown in Figure 4-1 below:

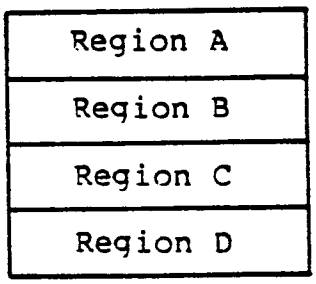


Figure 4-1. Example Memory Allocation

There is a hierarchical ownership of these regions: the program in A controls all memory from A through D. The program in B also controls regions B through D. The program in A can release regions B through D, if desired, and reload yet another program. DDT-86, for example, operates in this manner by executing the Free Memory call (function 57) to release the memory used by the current program before loading another test program. Further, the program in B can release regions C and D if required by the application. It must be noted, however, that if either A or B terminates by a System Reset (BDOS function 0 with DL = 00H) then all four regions A through D are released.

A transient program may release a portion of a region, allowing the released portion to be assigned on the next allocation request. The released portion must, however, be at the beginning or end of the region. Suppose, for example, the program in region B above receives 800H paragraphs at paragraph location 100H following its first allocation request as shown in Figure 4-2 below.

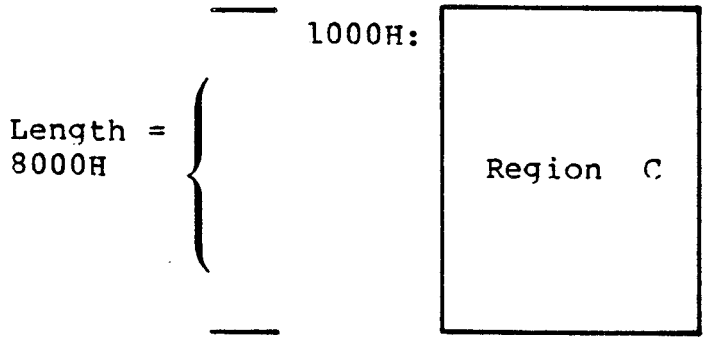


Figure 4-2. Example Memory Region

Suppose further that region D is then allocated. The last 200H paragraphs in region C can be returned without affecting region D by releasing the 200H paragraphs beginning at paragraph base 700H, resulting in the memory arrangement shown in Figure 4-3.

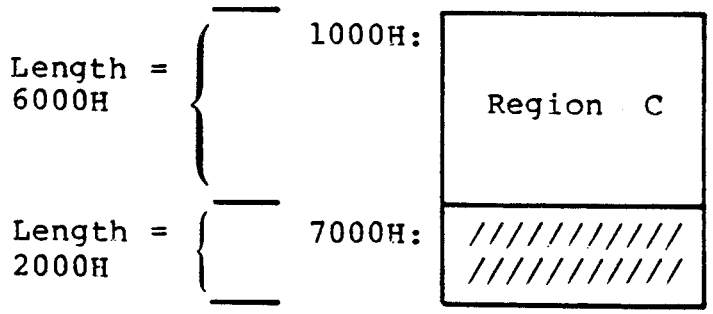
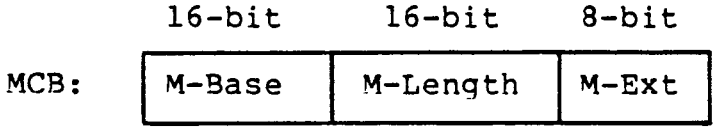


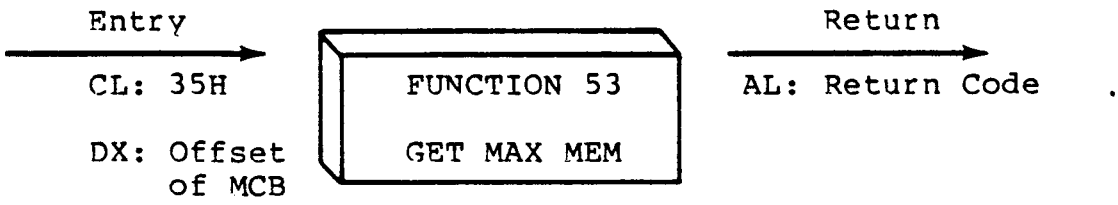
Figure 4-3. Example Memory Regions

The region beginning at paragraph address 700H is now available for allocation in the next request. Note that a memory request will fail if eight memory regions have already been allocated. Normally, if all program units can reside in a contiguous region, the system allocates only one region.

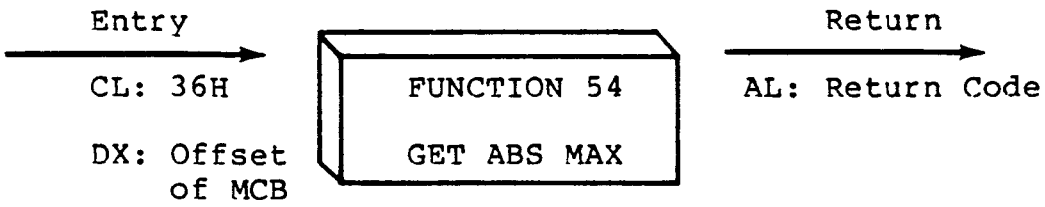
Memory management functions beginning at 53 reference a Memory Control Block (MCB), defined in the calling program, which takes the form:



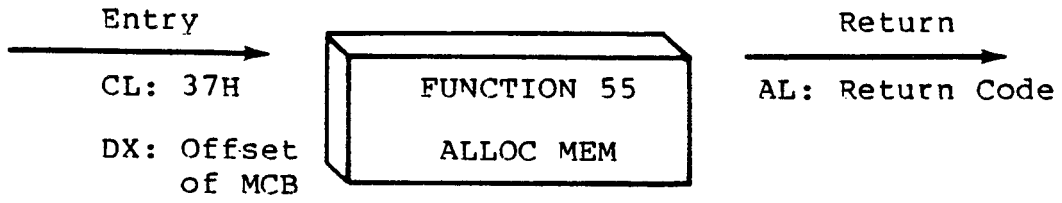
where M-Base and M-Length are either input or output values expressed in 16-byte paragraph units, and M-Ext is a returned byte value, as defined specifically with each function code. An error condition is normally flagged with a 0FFH returned value in order to match the file error conventions of CP/M.



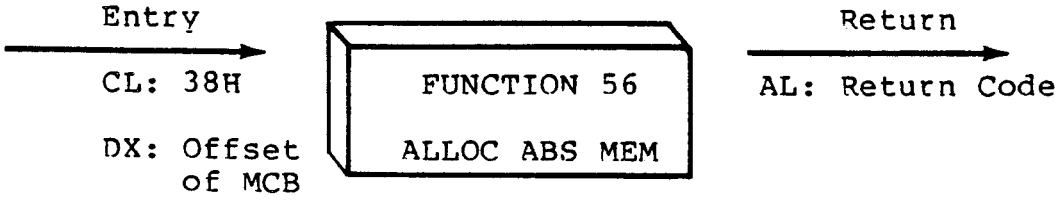
Function 53 finds the largest available memory region which is less than or equal to M-Length paragraphs. If successful, M-Base is set to the base paragraph address of the available area, and M-Length to the paragraph length. AL has the value 0FFH upon return if no memory is available, and 00H if the request was successful. M-Ext is set to 1 if there is additional memory for allocation, and 0 if no additional memory is available.



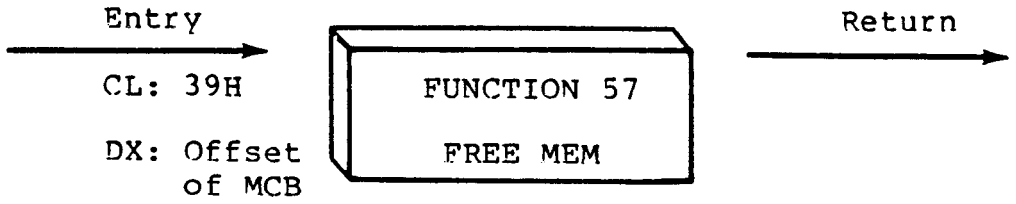
Function 54 is used to find the largest possible region at the absolute paragraph boundary given by M-Base, for a maximum of M-Length paragraphs. M-Length is set to the actual length if successful. AL has the value 0FFH upon return if no memory is available at the absolute address, and 00H if the request was successful.



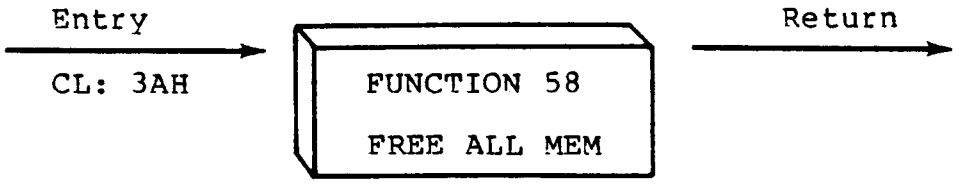
The allocate memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length. Function 55 returns in the user's MCB the base paragraph address of the allocated region. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.



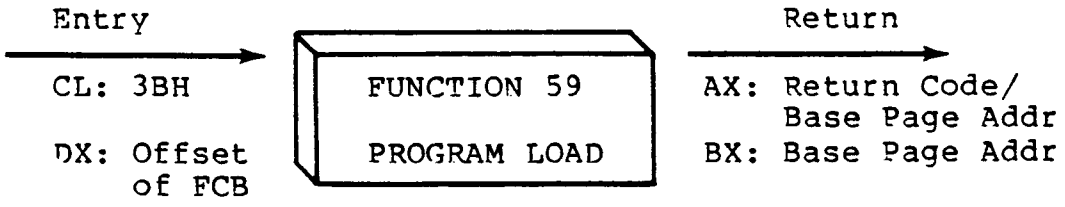
The allocate absolute memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length and the absolute base address from M-Base. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.



Function 57 is used to release memory areas allocated to the program. The value of the M-Ext field controls the operation of this function: if M-Ext = 0FFH then all memory areas allocated by the calling program are released. Otherwise, the memory area of length M-Length at location M-Base given in the MCB addressed by DX is released (the M-Ext field should be set to 00H in this case). As described above, either an entire allocated region must be released, or the end of a region must be released: the middle section cannot be returned under CP/M-86.



Function 58 is used to release all memory in the CP/M-86 environment (normally used only by the CCP upon initialization).



Function 59 loads a CMD file. Upon entry, register DX contains the DS relative offset of a successfully opened FCB which names the input CMD file. AX has the value 0FFFFH if the program load was unsuccessful. Otherwise, AX and BX both contain the paragraph address of the base page belonging to the loaded program. The base address and segment length of each segment is stored in the base page. Note that upon program load at the CCP level, the DMA base address is initialized to the base page of the loaded program, and the DMA offset address is initialized to 0080H. However, this is a function of the CCP, and a function 59 does not establish a default DMA address. It is the responsibility of the program which executes function 59 to execute function 51 to set the DMA base and function 26 to set the DMA offset before passing control to the loaded program.

Section 5
Basic I/O System (BIOS) Organization

The distribution version of CP/M-86 is setup for operation with the Intel SBC 86/12 microcomputer and an Intel 204 diskette controller. All hardware dependencies are, however, concentrated in subroutines which are collectively referred to as the Basic I/O System, or BIOS. A CP/M-86 system implementor can modify these subroutines, as described below, to tailor CP/M-86 to fit nearly any 8086 or 8088 operating environment. This section describes the actions of each BIOS entry point, and defines variables and tables referenced within the BIOS. The discussion of Disk Definition Tables is, however, treated separately in the next section of this manual.

5.1 Organization of the BIOS

The BIOS portion of CP/M-86 resides in the topmost portion of the operating system (highest addresses), and takes the general form shown in Figure 5-1, below:

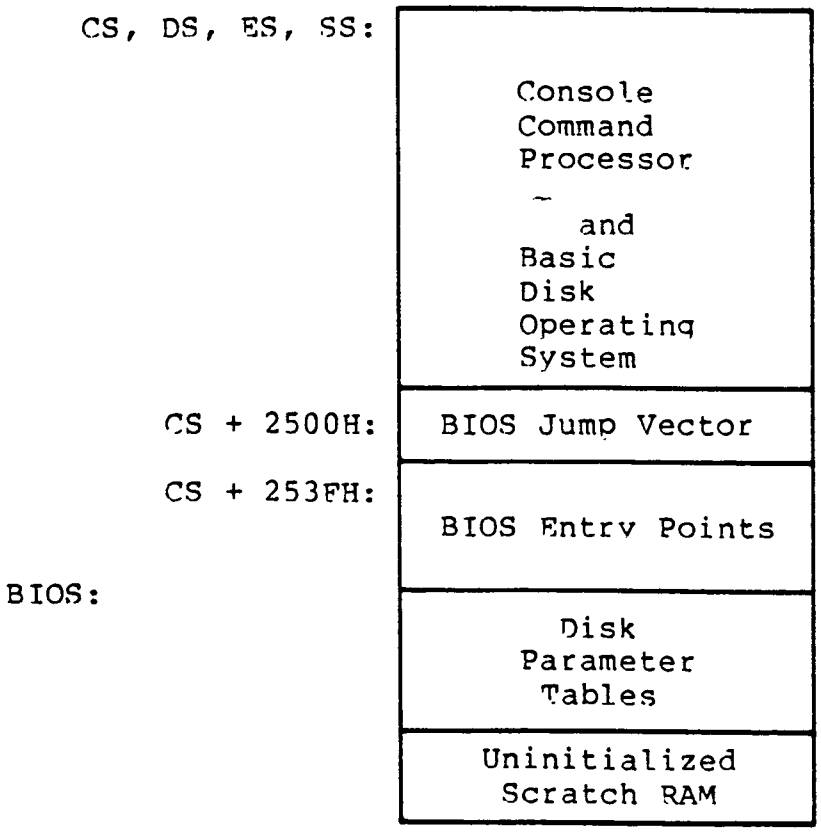


Figure 5-1. General CP/M-86 Organization

As described in the following sections, the CCP and BDOS are supplied with CP/M-86 in hex file form as CPM.H86. In order to implement CP/M-86 on non-standard hardware, you must create a BIOS which performs the functions listed below and concatenate the resulting hex file to the end of the CPM.H86 file. The GENCMD utility is then used to produce the CPM.SYS file for subsequent load by the cold start loader. The cold start loader that loads the CPM.SYS file into memory contains a simplified form of the BIOS, called the LDBIOS (Loader BIOS). It loads CPM.SYS into memory at the location defined in the CPM.SYS header (usually 0400H). The procedure to follow in construction and execution of the cold start loader and the CP/M-86 Loader is given in a later section.

Appendix D contains a listing of the standard CP/M-86 BIOS for the Intel SBC 86/12 system using the Intel 204 Controller Board. Appendix E shows a sample "skeletal" BIOS called CBIOS that contains the essential elements with the device drivers removed. You may wish to review these listings in order to determine the overall structure of the BIOS.

5.2 The BIOS Jump Vector

Entry to the BIOS is through a "jump vector" located at offset 2500H from the base of the operating system. The jump vector is a sequence of 21 three-byte jump instructions which transfer program control to the individual BIOS entry points. Although some non-essential BIOS subroutines may contain a single return (RET) instruction, the corresponding jump vector element must be present in the order shown below in Table 5-1. An example of a BIOS jump vector may be found in Appendix D, in the standard CP/M-86 BIOS listing.

Parameters for the individual subroutines in the BIOS are passed in the CX and DX registers, when required. CX receives the first parameter; DX is used for a second argument. Return values are passed in the registers according to type: Byte values are returned in AL. Word values (16 bits) are returned in BX. Specific parameters and returned values are described with each subroutine.

Table 5-1. BIOS Jump Vector

Offset from Beginning of BIOS	Suggested Instruction	BIOS F#	Description
2500H	JMP INIT	0	Arrive Here from Cold Boot
2503H	JMP WBOOT	1	Arrive Here for Warm Start
2506H	JMP CONST	2	Check for Console Char Ready
2509H	JMP CONIN	3	Read Console Character In
250CH	JMP CONOUT	4	Write Console Character Out
250FH	JMP LIST	5	Write Listing Character Out
2512H	JMP PUNCH	6	Write Char to Punch Device
2515H	JMP READER	7	Read Reader Device
2518H	JMP HOME	8	Move to Track 00
251BH	JMP SELDSK	9	Select Disk Drive
251EH	JMP SETTRK	10	Set Track Number
2521H	JMP SETSEC	11	Set Sector Number
2524H	JMP SETDMA	12	Set DMA Offset Address
2527H	JMP READ	13	Read Selected Sector
252AH	JMP WRITE	14	Write Selected Sector
252DH	JMP LISTST	15	Return List Status
2530H	JMP SECTRAN	16	Sector Translate
2533H	JMP SETDMAB	17	Set DMA Segment Address
2536H	JMP GETSEGB	18	Get MEM DESC Table Offset
2539H	JMP GETIOB	19	Get I/O Mapping Byte
253CH	JMP SETIOB	20	Set I/O Mapping Byte

There are three major divisions in the BIOS jump table: system (re)initialization subroutines, simple character I/O subroutines, and disk I/O subroutines.

5.3 Simple Peripheral Devices

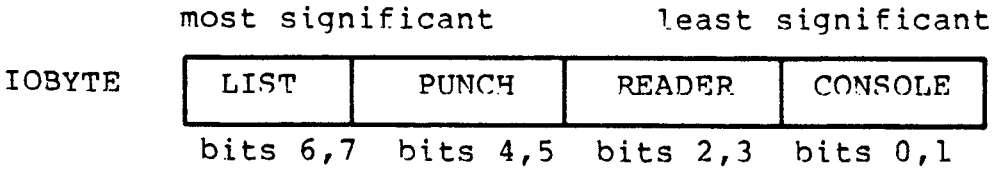
All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M-86 as "logical" devices, and are assigned to physical devices within the BIOS. Device characteristics are defined in Table 5-2.

Table 5-2. CP/M-86 Logical Device Characteristics

Device Name	Characteristics
CONSOLE	The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or Teletype.
LIST	The principal listing device, if it exists on your system, which is usually a hard-copy device, such as a printer or Teletype.
PUNCH	The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or Teletype.
READER	The principal tape reading device, such as a simple optical reader or teletype.

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, your CBIOS should give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other transient program. Alternately, the PUNCH and LIST subroutines can just simply return, and the READER subroutine can return with a 1AH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, you can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M-86 processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in the BIOS is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:



The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given in Table 5-3, below.

Table 5-3. IOBYTE Field Definitions

```
CONSOLE field (bits 0,1)
    0  - console is assigned to the console printer (TTY:)
    1  - console is assigned to the CRT device (CRT:)
    2  - batch mode: use the READER as the CONSOLE input,
         and the LIST device as the CONSOLE output (BAT:)
    3  - user defined console device (UC1:)

READER field (bits 2,3)
    0  - READER is the Teletype device (TTY:)
    1  - READER is the high-speed reader device (RDR:)
    2  - user defined reader # 1 (UR1:)
    3  - user defined reader # 2 (UR2:)

PUNCH field (bits 4,5)
    0  - PUNCH is the Teletype device (TTY:)
    1  - PUNCH is the high speed punch device (PUN:)
    2  - user defined punch # 1 (UP1:)
    3  - user defined punch # 2 (UP2:)

LIST field (bits 6,7)
    0  - LIST is the Teletype device (TTY:)
    1  - LIST is the CRT device (CRT:)
    2  - LIST is the line printer device (LPT:)
    3  - user defined list device (UL1:)
```

Note again that the implementation of the IOBYTE is optional, and affects only the organization of your CBIOS. No CP/M-86 utilities use the IOBYTE except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and displayed. In any case, you should omit the IOBYTE implementation until your basic CBIOS is fully implemented and tested, then add the IOBYTE to increase your facilities.

5.4 BIOS Subroutine Entry Points

The actions which must take place upon entry to each BIOS subroutine are given below. It should be noted that disk I/O is always performed through a sequence of calls on the various disk access subroutines. These setup the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) offset and segment addresses involved in the I/O operation. After all these parameters have been setup, a call is made to the READ or WRITE function to perform the actual I/O operation. Note that there is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a call to set the DMA segment base and a call to set the DMA offset followed by several calls which read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE subroutines should perform several retries (10 is standard) before reporting the error condition to the BDOS. The HOME subroutine may or may not actually perform the track 00 seek, depending upon your controller characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

Table 5-4. BIOS Subroutine Summary

Subroutine	Description
INIT	This subroutine is called directly by the CP/M-86 loader after the CPM.SYS file has been read into memory. The procedure is responsible for any hardware initialization not performed by the bootstrap loader, setting initial values for BIOS variables (including IOBYTE), printing a sign-on message, and initializing the interrupt vector to point to the BDOS offset (0B11H) and base. When this routine completes, it jumps to the CCP offset (0H). All segment registers should be initialized at this time to contain the base of the operating system.
WBOOT	This subroutine is called whenever a program terminates by performing a BDOS function #0 call. Some re-initialization of the hardware or software may occur here. When this routine completes, it jumps directly to the warm start entry point of the CCP (06H).
CONST	Sample the status of the currently assigned console device and return 0FFH in register AL if a character is ready to read, and 00H in register AL if no console characters are ready.

Table 5-4. (continued)

Subroutine	Description
CONIN	Read the next console character into register AL, and set the parity bit (high order bit) to zero. If no console character is ready, wait until a character is typed before returning.
CONOUT	Send the character from register CL to the console output device. The character is in ASCII, with high order parity bit set to zero. You may want to include a time-out on a line feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can, if you wish, filter out control characters which have undesirable effects on the console device.
LIST	Send the character from register CL to the currently assigned listing device. The character is in ASCII with zero parity.
PUNCH	Send the character from register CL to the currently assigned punch device. The character is in ASCII with zero parity.
READER	Read the next character from the currently assigned reader device into register AL with zero parity (high order bit must be zero). An end of file condition is reported by returning an ASCII CONTROL-Z (1AH).
HOME	Return the disk head of the currently selected disk to the track 00 position. If your controller does not have a special feature for finding track 00, you can translate the call into a call to SETTRK with a parameter of 0.

Table 5-4. (continued)

Subroutine	Description
SELDSK	Select the disk drive given by register CL for further operations, where register CL contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P (the standard CP/M-86 distribution version supports two drives). On each disk select, SELDSK must return in BX the base address of the selected drive's Disk Parameter Header. For standard floppy disk drives, the content of the header and associated tables does not change. The sample BIOS included with CP/M-86 called CBIOS contains an example program segment that performs the SELDSK function. If there is an attempt to select a non-existent drive, SELDSK returns BX=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is performed. This is due to the fact that disk select operations may take place without a subsequent disk operation and thus disk access may be substantially slower using some disk controllers. On entry to SELDSK it is possible to determine whether it is the first time the specified disk has been selected. Register DL, bit 0 (least significant bit) is a zero if the drive has not been previously selected. This information is of interest in systems which read configuration information from the disk in order to set up a dynamic disk definition table.
SETTRK	Register CX contains the track number for subsequent disk accesses on the currently selected drive. You can choose to seek the selected track at this time, or delay the seek until the next read or write actually occurs. Register CX can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives, and 0-65535 for non-standard disk subsystems.
SETSEC	Register CX contains the translated sector number for subsequent disk accesses on the currently selected drive (see SECTRAN, below). You can choose to send this information to the controller at this point, or instead delay sector selection until a read or write operation occurs.

Table 5-4. (continued)

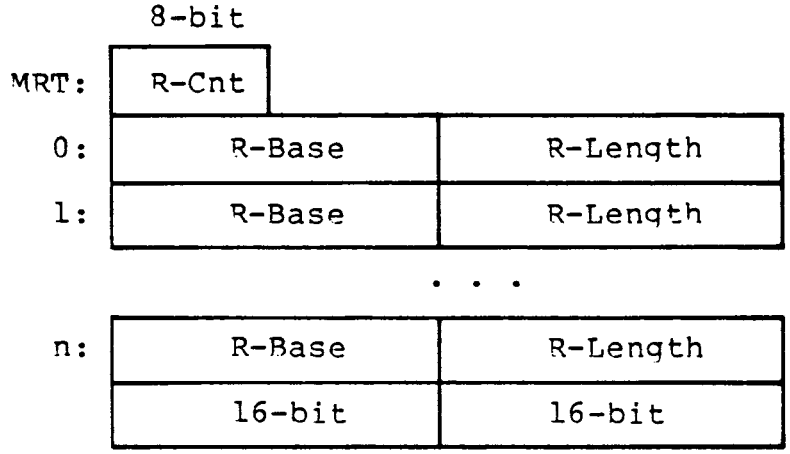
Subroutine	Description
SETDMA	Register CX contains the DMA (disk memory access) offset for subsequent read or write operations. For example, if CX = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH offset from the current DMA segment base, and all subsequent write operations get their data from that address, until the next calls to SETDMA and SETDMAB occur. Note that the controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the CBIOS which you construct will use the 128 byte area starting at the selected DMA offset and base for the memory buffer during the following read or write operations.
READ	Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA offset and segment base have been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register AL: 0 no errors occurred 1 non-recoverable error condition occurred Currently, CP/M-86 responds only to a zero or non-zero value as the return code. That is, if the value in register AL is 0 then CP/M-86 assumes that the disk operation completed properly. If an error occurs, however, the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing RETURN to ignore the error, or CONTROL-C to abort.
WRITE	Write the data from the currently selected DMA buffer to the currently selected drive, track, and sector. The data should be marked as "non-deleted data" to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register AL, with error recovery attempts as described above.
LISTST	Return the ready status of the list device. The value 00 is returned in AL if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer.

Table 5-4. (continued)

Subroutine	Description
SECTRAN	Performs logical to physical sector translation to improve the overall response of CP/M-86. Standard CP/M-86 systems are shipped with a "skew factor" of 6, where five physical sectors are skipped between sequential read or write operations. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In computer systems that use fast processors, memory and disk subsystems, the skew factor may be changed to improve overall response. Note, however, that you should maintain a single density IBM compatible version of CP/M-86 for information transfer into and out of your computer system, using a skew factor of 6. In general, SECTRAN receives a logical sector number in CX. This logical sector number may range from 0 to the number of sectors -1. Sectran also receives a translate table offset in DX. The sector number is used as an index into the translate table, with the resulting physical sector number in BX. For standard systems, the tables and indexing code is provided in the CBIOS and need not be changed. If DX = 0000H no translation takes place, and CX is simply copied to BX before returning. Otherwise, SECTRAN computes and returns the translated sector number in BX. Note that SECTRAN is called when no translation is specified in the Disk Parameter Header.
SETDMAB	Register CX contains the segment base for subsequent DMA read or write operations. The BIOS will use the 128 byte buffer at the memory address determined by the DMA base and the DMA offset during read and write operations.
GETSEGB	Returns the address of the Memory Region Table (MRT) in BX. The returned value is the offset of the table relative to the start of the operating system. The table defines the location and extent of physical memory which is available for transient programs.

Table 5-4. (continued)

Subroutine	Description
	Memory areas reserved for interrupt vectors and the CP/M-86 operating system are not included in the MRT. The Memory Region Table takes the form:



	where R-Cnt is the number of Memory Region Descriptors (equal to n+1 in the diagram above), while R-Base and R-Length give the paragraph base and length of each physically contiguous area of memory. Again, the reserved interrupt locations, normally 0-3FFH, and the CP/M-86 operating system are not included in this map, because the map contains regions available to transient programs. If all memory is contiguous, the R-Cnt field is 1 and n = 0, with only a single Memory Region Descriptor which defines the region.
GETIOB	Returns the current value of the logical to physical input/output device byte (IOBYTE) in AL. This eight-bit value is used to associate physical devices with CP/M-86's four logical devices.
SETIOB	Use the value in CL to set the value of the IOBYTE stored in the BIOS.

The following section describes the exact layout and construction of the disk parameter tables referenced by various subroutines in the BIOS.

Section 6
BIOS Disk Definition Tables

Similar to CP/M-80, CP/M-86 is a table-driven operating system with a separate field-configurable Basic I/O System (BIOS). By altering specific subroutines in the BIOS presented in the previous section, CP/M-86 can be customized for operation on any RAM-based 8086 or 8088 microprocessor system.

The purpose of this section is to present the organization and construction of tables within the BIOS that define the characteristics of a particular disk system used with CP/M-86. These tables can be either hand-coded or automatically generated using the GENDEF utility provided with CP/M-86. The elements of these tables are presented below.

6.1 Disk Parameter Table Format

In general, each disk drive has an associated (16-byte) disk parameter header which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below.

Disk Parameter Header							
XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is given in Table 6-1.

Table 6-1. Disk Parameter Header Elements

Element	Description
XLT	Offset of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e, the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.
0000	Scratchpad values for use within the BDOS (initial value is unimportant).

Table 6-1. (continued)

Element	Description
DIRBUF	Offset of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.
DPB	Offset of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.
CSV	Offset of a scratchpad area used for software check for changed disks. This offset is different for each DPH.
ALV	Offset of a scratchpad area used by the BDOS to keep disk storage allocation information. This offset is different for each DPH.

Given n disk drives, the DPH's are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as

DPBASE

00	XLT 00	0000	0000	0000	DIRBUF	DBP 00	CSV 00	ALV 00
01	XLT 01	0000	0000	0000	DIRBUF	DBP 01	CSV 01	ALV 01
	(and so-forth through)							
n-1	XLTn-1	0000	0000	0000	DIRBUF	DBPn-1	CSVn-1	ALVn-1

where the label DPBASE defines the offset of the DPH table relative to the beginning of the operating system.

A responsibility of the SELDSK subroutine, defined in the previous section, is to return the offset of the DPH from the beginning of the operating system for the selected drive. The following sequence of operations returns the table offset, with a 0000H returned if the selected drive does not exist.

```
NDISKS  EQU     4    ;NUMBER OF DISK DRIVES
......
SELDSK:
        ;SELECT DISK N GIVEN BY CL
        MOV     BX,0000H  ;READY FOR ERR
        CPM     CL,NDISKS ;N BEYOND MAX DISKS?
        JNB     RETURN    ;RETURN IF SO
                          ;0 <= N < NDISKS
        MOV     CH,0      ;DOUBLE (N)
        MOV     BX,CX     ;BX = N
        MOV     CL,4      ;READY FOR * 16
        SHL     BX,CL     ;N = N * 16
        MOV     CX,OFFSET DPBASE
        ADD     BX,CX     ;DPBASE + N * 16
RETURN: RET               ;BX - .DPH (N)
```

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count-1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPH's, takes the general form:

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b

where each is a byte or word value, as shown by the "8b" or "16b" indicator below the field. The fields are defined in Table 6-2.

Table 6-2. Disk Parameter Block Fields

Field	Definition
SPT	is the total number of sectors per track
BSH	is the data allocation block shift factor, determined by the data block allocation size.
BLM	is the block mask which is also determined by the data block allocation size.
EXM	is the extent mask, determined by the data block allocation size and the number of disk blocks.
DSM	determines the total storage capacity of the disk drive
DRM	determines the total number of directory entries which can be stored on this drive

Table 6-2. (continued)

Field	Definition
AL0,AL1	determine reserved directory blocks.
CKS	is the size of the directory check vector
OFF	is the number of reserved tracks at the beginning of the (logical) disk.

Although these table values are produced automatically by GENDEF, it is worthwhile reviewing the derivation of each field so that the values may be cross-checked when necessary. The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the disk parameter block. Given that you have selected a value for BLS, the values of BSH and BLM are shown in Table 6-3 below, where all values are in decimal.

Table 6-3. BSH and BLM Values for Selected BLS

BLS	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

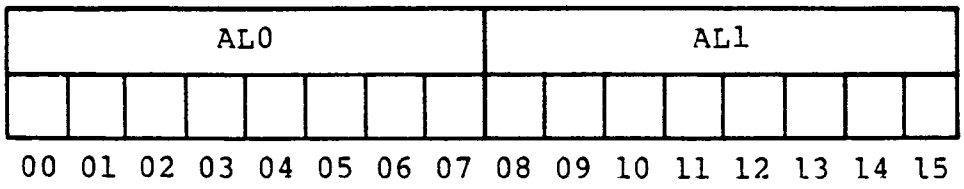
The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in the following table.

Table 6-4. Maximum EXM Values

BLS	DSM < 256	DSM > 255
1,024	0	N/A
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is one less than the total number of directory entries, which can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The two values AL0 and AL1 can together be considered a string of 16-bits, as shown below.



where position 00 corresponds to the high order bit of the byte labeled AL0, and 15 corresponds to the low order bit of the byte labeled AL1. Each bit position reserves a data block for a number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, as shown in Table 6-5.

Table 6-5. BLS and Number of Directory Entries

BLS	Directory Entries
1,024	32 times # bits
2,048	64 times # bits
4,096	128 times # bits
8,192	256 times # bits
16,384	512 times # bits

Thus, if DRM = 127 (128 directory entries), and BLS = 1024, then there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media is fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks which are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called, and can be used as a mechanism for skipping reserved operating system tracks, or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, recall that several DPH's can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, note that the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If CKS = (DRM+1)/4, then you must reserve (DRM+1)/4 bytes for directory check use. If CKS = 0, then no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is computed as (DSM/8)+1.

The BIOS shown in Appendix D demonstrates an instance of these tables for standard 8" single density drives. It may be useful to examine this program, and compare the tabular values with the definitions given above.

6.2 Table Generation Using GENDEF

The GENDEF utility supplied with CP/M-86 greatly simplifies the table construction process. GENDEF reads a file

x.DEF

containing the disk definition statements, and produces an output file

x.LIB

containing assembly language statements which define the tables necessary to support a particular drive configuration. The form of the GENDEF command is:

GENDEF x parameter list

where x has an assumed (and unspecified) filetype of DEF. The parameter list may contain zero or more of the symbols defined in Table 6-6.

Table 6-6. GENDEF Optional Parameters

Parameter	Effect
$C	Generate Disk Parameter Comments
$O	Generate DPBASE OFFSET $
$Z	Z80, 8080, 8085 Override
$COZ	(Any of the Above)

The C parameter causes GENDEF to produce an accompanying comment line, similar to the output from the "STAT DSK:" utility which describes the characteristics of each defined disk. Normally, the DPBASE is defined as

```
DPBASE  EQU  $
```

which requires a MOV CX,OFFSET DPBASE in the SELDSK subroutine shown above. For convenience, the $O parameter produces the definition

```
DPBASE  EQU  OFFSET $
```

allowing a MOV CX,DPBASE in SELDSK, in order to match your particular programming practices. The $Z parameter is included to override the standard 8086/8088 mode in order to generate tables acceptable for operation with Z80, 8080, and 8085 assemblers.

The disk definition contained within x.DEF is composed with the CP/M text editor, and consists of disk definition statements identical to those accepted by the DISKDEF macro supplied with CP/M-80 Version 2. A BIOS disk definition consists of the following sequence of statements:

```
DISKS    n
DISKDEF  0,...
DISKDEF  1,...
......
DISKDEF  n-1
......
ENDEF
```

Each statement is placed on a single line, with optional embedded comments between the keywords, numbers, and delimiters.

The DISKS statement defines the number of drives to be configured with your system, where n is an integer in the range 1 through 16. A series of DISKDEF statements then follow which define the characteristics of each logical disk, 0 through n-1, corresponding to logical drives A through P. Note that the DISKS and DISKDEF statements generate the in-line fixed data tables described in the previous section, and thus must be placed in a non-executable portion of your BIOS, typically at the end of your BIOS, before the start of uninitialized RAM.

The ENDEF (End of Diskdef) statement generates the necessary uninitialized RAM areas which are located beyond initialized RAM in your BIOS.

The form of the DISKDEF statement is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

dn	is the logical disk number, 0 to n-1
fsc	is the first physical sector number (0 or 1)
lsc	is the last sector number
skf	is the optional sector skew factor
bls	is the data allocation block size
dks	is the disk size in bls units
dir	is the number of directory entries
cks	is the number of "checked" directory entries
ofs	is the track offset to logical track 00
[0]	is an optional 1.4 compatibility flag

The value "dn" is the drive number being defined with this DISKDEF statement. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted or equal to 0.

The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes because there are fewer directory references. Also, logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the amount of BIOS work space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired.

The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold start or system reset has not occurred (when this situation is detected, CP/M-86 automatically marks the disk read/only so that data is not subsequently destroyed). As stated in the previous section, the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low.

The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of CP/M-80, version 1.4 which have been modified for higher density disks (typically double density). This parameter ensures that no directory compression takes place, which would cause incompatibilities with these non-standard CP/M 1.4 versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF    i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with CP/M-80 Version 1.4, and upwardly compatible with CP/M-80 Version 2 implementations, is defined using the following statements:

```
DISKS     4
DISKDEF   0,1,26,6,1024,243,64,64,2
DISKDEF   1,0
DISKDEF   2,0
DISKDEF   3,0
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with a skew of 6 between sequential accesses, 1024 bytes per data block, 243 data blocks for a total of 243K byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS statement generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the statement. Each disk header block contains sixteen bytes, as described above, and corresponds one-for-one to each of the defined drives. In the four drive standard system, for example, the DISKS statement generates a table of the form:

```
DPBASE  EQU  $
DPE0    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail earlier in this section. The check and allocation vector addresses are generated by the ENDEF statement for inclusion in the RAM area following the BIOS code and tables.

Note that if the "skf" (skew factor) parameter is omitted (or equal to 0), the translation table is omitted, and a 0000H value is inserted in the XLT position of the disk parameter header for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN receives a translation table address of DX = 0000H, and simply returns the original logical sector from CX in the BX register. A translate table is constructed when the skf parameter is present, and the (non-zero) table address is placed into the corresponding DPH's. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF statement call:

```
XLT0    EQU   OFFSET $
        DB    1,7,13,19,25,5,11,17,23,3,9,15,21
        DB    2,8,14,20,26,6,12,18,24,4,10,16,22
```

Following the ENDEF statement, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS which is loaded upon cold start, but must be available between the BIOS and the end of operating system memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF statement. For a standard four-drive system, the ENDEF statement might produce

```
1C72 =      BEGDAT EQU OFFSET $
            (data areas)
1DB0 =      ENDDAT EQU OFFSET $
013C =      DATSIZ EQU OFFSET $-BEGDAT
```

which indicates that uninitialized RAM begins at offset 1C72H, ends at 1DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

After modification, you can use the STAT program to check your drive characteristics, since STAT uses the disk parameter block to decode the drive information. The comment included in the LIB file by the $C parameter to GENCMD will match the output from STAT. The STAT command form

```
STAT d:DSK:
```

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below:

```
r: 128 Byte Record Capacity
k: Kilobyte Drive  Capacity
d: 32  Byte Directory Entries
c: Checked  Directory Entries
e: Records/ Extent
b: Records/ Block
s: Sectors/ Track
t: Reserved Tracks
```

6.3 GENDEF Output

GENDEF produces a listing of the statements included in the DEF file at the user console (CONTROL-P can be used to obtain a printed listing, if desired). Each source line is numbered, and any errors are shown below the line in error, with a "?" beneath the item which caused the condition. The source errors produced by GENCMD are listed in Table 6-7, followed by errors that can occur when producing input and output files in Table 6-8.

Table 6-7. GENDEF Source Error Messages

Message	Meaning
Bad Val	More than 16 disks defined in DISKS statement.
Convert	Number cannot be converted, must be constant in binary, octal, decimal, or hexadecimal as in ASM-86.
Delimit	Missing delimiter between parameters.
Duplic	Duplicate definition for a disk drive.
Extra	Extra parameters occur at the end of line.
Length	Keyword or data item is too long.
Missing	Parameter required in this position.
No Disk	Referenced disk not previously defined.
No Stmt	Statement keyword not recognized.
Numeric	Number required in this position
Range	Number in this position is out of range.
Too Few	Not enough parameters provided.
Quote	Missing end quote on current line.

Table 6-8. GENDEF Input and Output Error Messages

Message	Meaning
Cannot Close ".LIB" File	LIB file close operation unsuccessful, usually due to hardware write protect.
"LIB" Disk Full	No space for LIB file.
No Input File Present	Specified DEF file not found.
No ".LIB" Directory Space	Cannot create LIB file due to too many files on LIB disk.
Premature End-of-File	End of DEF file encountered unexpectedly.

Given the file TWO.DEF containing the following statements

```
disks   2
diskdef 0,1,26,6,2048,256,128,128,2
diskdef 1,1,58,,2048,1024,300,0,2
endef
```

the command

```
gencmd two $c
```

produces the console output

```
DISKDEF Table Generator, Vers 1.0
1               DISKS   2
2               DISKDEF 0,1,58,,2048,256,128,128,2
3               DISKDEF 1,1,58,,2048,1024,300,0,2
4               ENDEF
No Error(s)
```

The resulting TWO.LIB file is brought into the following skeletal assembly language program, using the ASM-86 INCLUDE directive. The ASM-86 output listing is truncated on the right, but can be easily reproduced using GENDEF and ASM-86.

```
                        ;       Sample Program Including TWO.LI
                        ;
                        SELDSK:
                        ;       ....
  0000 B9 03 00                 MOV     CX,OFFSET DPBASE
                        ;       ....
=                               INCLUDE TWO.LIB
=                       ;               DISKS   2
=    0003               dpbase  equ     $                       ;Base o
= 0003 32 00 00 00      dpe0    dw      xlt0,0000h              ;Transl
= 0007 00 00 00 00              dw      0000h,0000h             ;Scratc
= 000B 5B 00 23 00              dw      dirbuf,dpb0             ;Dir Bu
= 000F FB 00 DB 00              dw      csv0,alv0               ;Check,
= 0013 00 00 00 00      dpe1    dw      xlt1,0000h              ;Transl
= 0017 00 00 00 00              dw      0000h,0000h             ;Scratc
= 001B 5B 00 4C 00              dw      dirbuf,dpb1             ;Dir Bu
= 001F 9B 01 1B 01              dw      csv1,alv1               ;Check,
=                       ;               DISKDEF 0,1,26,6,2048,2
=                       ;
=                       ;       Disk 0 is CP/M 1.4 Single Densi
=                       ;        4096:  128 Byte Record Capacit
=                       ;         512:  Kilobyte Drive  Capacit
=                       ;         128:  32 Byte Directory Entri
=                       ;         128:  Checked Directory Entri
=                       ;         256:  Records / Extent
=                       ;          16:  Records / Block
=                       ;          26:  Sectors / Track
=                       ;           2:  Reserved  Tracks
=                       ;           6:  Sector Skew Factor
=                       ;
=    0023               dpb0    equ     offset $                ;Disk P
= 0023 1A 00                    dw      26                      ;Sector
= 0025 04                       db      4                       ;Block
= 0026 0F                       db      15                      ;Block
= 0027 01                       db      1                       ;Extnt
= 0028 FF 00                    dw      255                     ;Disk S
= 002A 7F 00                    dw      127                     ;Direct
= 002C C0                       db      192                     ;Alloc0
= 002D 00                       db      0                       ;Allocl
= 002E 20 00                    dw      32                      ;Check
= 0030 02 00                    dw      2                       ;Offset
=    0032               xlt0    equ     offset $                ;Transl
= 0032 01 07 0D 13              db      1,7,13,19
= 0036 19 05 0B 11              db      25,5,11,17
= 003A 17 03 09 0F              db      23,3,9,15
= 003E 15 02 08 0E              db      21,2,8,14
= 0042 14 1A 06 0C              db      20,26,6,12
= 0046 12 18 04 0A              db      18,24,4,10
= 004A 10 16                    db      16,22
=    0020               als0    equ     32                      ;Alloca
=    0020               css0    equ     32                      ;Check
=                       ;               DISKDEF 1,1,58,,2048,10
=                       ;
=                       ;       Disk 1 is CP/M 1.4 Single Densi
=                       ;       16384:  128 Byte Record Capacit
```

```
=                       ;         2048:  Kilobyte Drive  Capacit
=                       ;          300:  32 Byte Directory Entri
=                       ;            0:  Checked Directory Entri
=                       ;          128:  Records / Extent
=                       ;           16:  Records / Block
=                       ;           58:  Sectors / Track
=                       ;            2:  Reserved  Tracks
=                       ;
=   004C                dpbl    equ     offset $        ;Disk P
= 004C 3A 00                    dw      58              ;Sector
= 004E 04                       db      4               ;Block
= 004F 0F                       db      15              ;Block
= 0050 00                       db      0               ;Extnt
= 0051 FF 03                    dw      1023            ;Disk S
= 0053 2B 01                    dw      299             ;Direct
= 0055 F8                       db      248             ;Alloc0
= 0056 00                       db      0               ;Alloc1
= 0057 00 00                    dw      0               ;Check
= 0059 02 00                    dw      2               ;Offset
=   0000                xlt1    equ     0               ;No Tra
=   0080                als1    equ     128             ;Alloca
=   0000                css1    equ     0               ;Check
=                       ;               ENDEF
=                       ;
=                       ;       Uninitialized Scratch Memory Fo
=                       ;
=   005B                begdat  equ     offset $        ;Start
= 005B                  dirbuf  rs      128             ;Direct
= 00DB                  alv0    rs      als0            ;Alloc
= 00FB                  csv0    rs      css0            ;Check
= 011B                  alv1    rs      als1            ;Alloc
= 019B                  csv1    rs      css1            ;Check
=   019B                enddat  equ     offset $        ;End of
=   0140                datsiz  equ     offset $-begdat ;Size o
= 019B 00                       db      0               ;Marks
                                END
```

Section 7
CP/M-86 Bootstrap and Adaptation Procedures

This section describes the components of the standard CP/M-86 distribution disk, the operation of each component, and the procedures to follow in adapting CP/M-86 to non-standard hardware.

CP/M-86 is distributed on a single-density IBM compatible 8" diskette using a file format which is compatible with all previous CP/M-80 operating systems. In particular, the first two tracks are reserved for operating system and bootstrap programs, while the remainder of the diskette contains directory information which leads to program and data files. CP/M-86 is distributed for operation with the Intel SBC 86/12 single-board computer connected to floppy disks through an Intel 204 Controller. The operation of CP/M-86 on this configuration serves as a model for other 8086 and 8088 environments, and is presented below.

The principal components of the distribution system are listed below:

- The 86/12 Bootstrap ROM (BOOT ROM)
- The Cold Start Loader (LOADER)
- The CP/M-86 System (CPM.SYS)

When installed in the SBC 86/12, the BOOT ROM becomes a part of the memory address space, beginning at byte location 0FF000H, and receives control when the system reset button is depressed. In a non-standard environment, the BOOT ROM is replaced by an equivalent initial loader and, therefore, the ROM itself is not included with CP/M-86. The BOOT ROM can be obtained from Digital Research or, alternatively, it can be programmed from the listing given in Appendix C or directly from the source file which is included on the distribution disk as BOOT.A86. The responsibility of the BOOT ROM is to read the LOADER from the first two system tracks into memory and pass program control to the LOADER for execution.

7.1 The Cold Start Load Operation

The LOADER program is a simple version of CP/M-86 that contains sufficient file processing capability to read CPM.SYS from the system disk to memory. When LOADER completes its operation, the CPM.SYS program receives control and proceeds to process operator input commands.

Both the LOADER and CPM.SYS programs are preceded by the standard CMD header record. The 128-byte LOADER header record contains the following single group descriptor.

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	0400	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

where G-Form = 1 denotes a code group, "x" fields are ignored, and A-Base defines the paragraph address where the BOOT ROM begins filling memory (A-Base is the word value which is offset three bytes from the beginning of the header). Note that since only a code group is present, an 8080 memory model is assumed. Further, although the A-Base defines the base paragraph address for LOADER (byte address 04000H), the LOADER can, in fact be loaded and executed at any paragraph boundary that does not overlap CP/M-86 or the BOOT ROM.

The LOADER itself consists of three parts: the Load CPM program (LDCPM), the Loader Basic Disk System (LDBDOS), and the Loader Basic I/O System (LDBIOS). Although the LOADER is setup to initialize CP/M-86 using the Intel 86/12 configuration, the LDBIOS can be field-altered to account for non-standard hardware using the same entry points described in a previous section for BIOS modification. The organization of LOADER is shown in Figure 7-1 below:

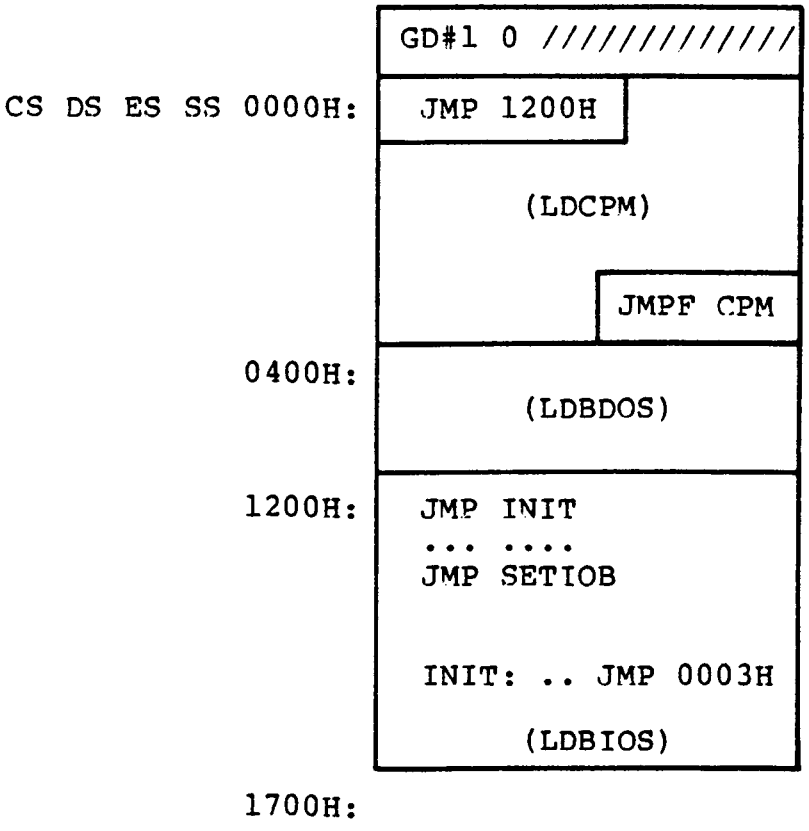


Figure 7-1. LOADER Organization

Byte offsets from the base registers are shown at the left of the diagram. GD#1 is the Group Descriptor for the LOADER code group described above, followed immediately by a "0" group terminator. The entire LOADER program is read by the BOOT ROM, excluding the header record, starting at byte location 04000H as given by the A-Field. Upon completion of the read, the BOOT ROM passes control to location 04000H where the LOADER program commences execution. The JMP 1200H instruction at the base of LDCPM transfers control to the beginning of the LDBIOS where control then transfers to the INIT subroutine. The subroutine starting at INIT performs device initialization, prints a sign-on message, and transfers back to the LDCPM program at byte offset 0003H. The LDCPM module opens the CPM.SYS file, loads the CP/M-86 system into memory and transfers control to CP/M-86 through the JMPF CPM instruction at the end of LDCPM execution, thus completing the cold start sequence.

The files LDCPM.H86 and LDBDOS.H86 are included with CP/M-86 so that you can append your own modified LDBIOS in the construction of a customized loader. In fact, BIOS.A86 contains a conditional assembly switch, called "loader_bios," which, when enabled, produces the distributed LDBIOS. The INIT subroutine portion of LDBIOS is listed in Appendix C for reference purposes. To construct a custom LDBIOS, modify your standard BIOS to start the code at offset 1200H, and change your initialization subroutine beginning at INIT to perform disk and device initialization. Include a JMP to offset 0003H at the end of your INIT subroutine. Use ASM-86 to assemble your LDBIOS.A86 program:

```
ASM86 LDBIOS
```

to produce the LDBIOS.H86 machine code file. Concatenate the three LOADER modules using PIP:

```
PIP LOADER.H86=LDCPM.H86,LDBDOS.H86,LDBIOS.H86
```

to produce the machine code file for the LOADER program. Although the standard LOADER program ends at offset 1700H, your modified LDBIOS may differ from this last address with the restriction that the LOADER must fit within the first two tracks and not overlap CP/M-86 areas. Generate the command (CMD) file for LOADER using the GENCMD utility:

```
GENCMD LOADER 8080 CODE[A400]
```

resulting in the file LOADER.CMD with a header record defining the 8080 Memory Model with an absolute paragraph address of 400H, or byte address 4000H. Use DDT to read LOADER.CMD to location 900H in your 8080 system. Then use the 8080 utility SYSGEN to copy the loader to the first two tracks of a disk.

```
A>DDT
-ILOADER.CMD
-R800
-^C
A>SYSGEN
SOURCE DRIVE NAME (or return to skip) <cr>
DESTINATION DRIVE NAME (or return to skip) B
```

Alternatively, if you have access to an operational CP/M-86 system, the command

```
LDCOPY LOADER
```

copies LOADER to the system tracks. You now have a diskette with a LOADER program which incorporates your custom LDBIOS capable of reading the CPM.SYS file into memory. For standardization, we assume LOADER executes at location 4000H. LOADER is statically relocatable, however, and its operating address is determined only by the value of A-Base in the header record.

You must, of course, perform the same function as the BOOT ROM to get LOADER into memory. The boot operation is usually accomplished in one of two ways. First, you can program your own ROM (or PROM) to perform a function similar to the BOOT ROM when your computer's reset button is pushed. As an alternative, most controllers provide a power-on "boot" operation that reads the first disk sector into memory. This one-sector program, in turn, reads the LOADER from the remaining sectors and transfers to LOADER upon completion, thereby performing the same actions as the BOOT ROM. Either of these alternatives is hardware-specific, so you'll need to be familiar with the operating environment.

7.2 Organization of CPM.SYS

The CPM.SYS file, read by the LOADER program, consists of the CCP, BDOS, and BIOS in CMD file format, with a 128-byte header record similar to the LOADER program:

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	040	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

where, instead, the A-Base load address is paragraph 040H, or byte address 0400H, immediately following the 8086 interrupt locations. The entire CPM.SYS file appears on disk as shown in Figure 7-2.

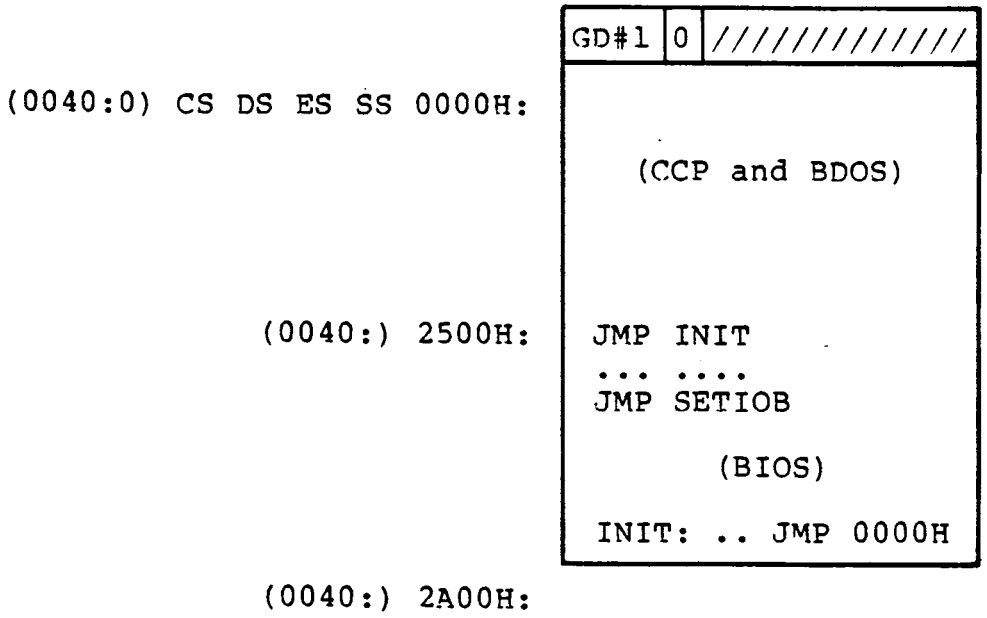


Figure 7-2. CPM.SYS File Organization

where GD#1 is the Group Descriptor containing the A-Base value followed by a "0" terminator. The distributed 86/12 BIOS is listed in Appendix D, with an "include" statement that reads the SINGLES.LIB file containing the disk definition tables. The SINGLES.LIB file is created by GENDEF using the SINGLES.DEF statements shown below:

```
disks 2
diskdef 0,1,26,6,1024,243,64,64,2
diskdef 1,0
endef
```

The CPM.SYS file is read by the LOADER program beginning at the address given by A-Base (byte address 0400H), and control is passed to the INIT entry point at offset address 2500H. Any additional initialization, not performed by LOADER, takes place in the INIT subroutine and, upon completion, INIT executes a JMP 0000H to begin execution of the CCP. The actual load address of CPM.SYS is determined entirely by the address given in the A-Base field which can be changed if you wish to execute CP/M-86 in another region of memory. Note that the region occupied by the operating system must be excluded from the BIOS memory region table.

Similar to the LOADER program, you can modify the BIOS by altering either the BIOS.A86 or skeletal CBIOS.A86 assembly language files which are included on your source disk. In either case, create a customized BIOS which includes your specialized I/O drivers, and assemble using ASM-86:

```
ASM86 BIOS
```

to produce the file BIOS.H86 containing your BIOS machine code.

Concatenate this new BIOS to the CPM.H86 file on your distribution disk:

```
PIP CPMX.H86 = CPM.H86,BIOS.H86
```

The resulting CPMX hex file is then converted to CMD file format by executing

```
GENCMD CPMX 8080 CODE[A40]
```

in order to produce the CMD memory image with A-Base = 40H. Finally, rename the CPMX file using the command

```
REN CPM.SYS = CPMX.CMD
```

and place this file on your 8086 system disk. Now the tailoring process is complete: you have replaced the BOOT ROM by either your own customized BOOT ROM, or a one-sector cold start loader which brings the LOADER program, with your custom LDBIOS, into memory at byte location 04000H. The LOADER program, in turn, reads the CPM.SYS file, with your custom BIOS, into memory at byte location 0400H. Control transfers to CP/M-86, and you are up and operating. CP/M-86 remains in memory until the next cold start operation takes place.

You can avoid the two-step boot operation if you construct a non-standard disk with sufficient space to hold the entire CPM.SYS file on the system tracks. In this case, the cold start brings the CP/M-86 memory image into memory at the location given by A-Base, and control transfers to the INIT entry point at offset 2500H. Thus, the intermediate LOADER program is eliminated entirely, although the initialization found in the LDBIOS must, of course, take place instead within the BIOS.

Since ASM-86, GENCMD and GENDEF are provided in both COM and CMD formats, either CP/M-80 or CP/M-86 can be used to aid the customizing process. If CP/M-80 or CP/M-86 is not available, but you have minimal editing and debugging tools, you can write specialized disk I/O routines to read and write the system tracks, as well as the CPM.SYS file.

The two system tracks are simple to access, but the CPM.SYS file is somewhat more difficult to read. CPM.SYS is the first file on the disk and thus it appears immediately following the directory on the diskette. The directory begins on the third track, and occupies the first sixteen logical sectors of the diskette, while the CPM.SYS is found starting at the seventeenth sector. Sectors are "skewed" by a factor of six beginning with the directory track (the system tracks are sequential), so that you must load every sixth sector in reading the CPM.SYS file. Clearly, it is worth the time and effort to use an existing CP/M system to aid the conversion process.

Appendix A
Sector Blocking and Deblocking

Upon each call to the BIOS WRITE entry point, the CP/M-86 BDOS includes information that allows effective sector blocking and deblocking where the host disk subsystem has a sector size which is a multiple of the basic 128-byte unit. This appendix presents a general-purpose algorithm that can be included within your BIOS and that uses the BDOS information to perform the operations automatically.

Upon each call to WRITE, the BDOS provides the following information in register CL:

0	=	normal sector write
1	=	write to directory sector
2	=	write to the first sector of a new data block

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128-byte sectors in sequence, and thus there is little overhead involved in either operation when blocking and deblocking records since pre-read operations can be avoided when writing records.

This appendix lists the blocking and deblocking algorithm in skeletal form (the file is included on your CP/M-86 disk). Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer which is the size of the host disk sector. Throughout the program, values and variables which relate to the CP/M sector involved in a seek operation are prefixed by "sek," while those related to the host disk system are prefixed by "hst." The equate statements beginning on line 24 of Appendix F define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The SELDSK entry point clears the host buffer flag whenever a new disk is logged-in. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETSEC, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial function of returning the physical sector number.

The principal entry points are READ and WRITE. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host disk number, hsttrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). You must insert code at this point which performs the full host sector read or write into, or out of, the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

```
;*********************************************************
;*                                                       *
;*         Sector Blocking / Deblocking                  *
;*                                                       *
;* This algorithm is a direct translation of the         *
;* CP/M-80 Version, and is included here for refer-      *
;* ence purposes only.  The file DEBLOCK.LIB is in-      *
;* cluded on your CP/M-86 disk, and should be used       *
;* for actual applications.  You may wish to contact     *
;* Digital Research for notices of updates.              *
;*                                                       *
;*********************************************************
;
;*********************************************************
;*                                                       *
;*         CP/M to host disk constants                   *
;*                                                       *
;* (This example is setup for CP/M block size of 16K     *
;* with a host sector size of 512 bytes, and 12 sec-     *
;* tors per track.  Blksiz, hstsiz, hstspt, hstblk       *
;* and secshf may change for different hardware.)        *
;*********************************************************
una     equ     byte ptr [BX]   ;name for byte at BX
;
blksiz  equ     16384           ;CP/M allocation size
hstsiz  equ     512             ;host disk sector size
hstspt  equ     12              ;host disk sectors/trk
hstblk  equ     hstsiz/128      ;CP/M sects/host buff
;
;*********************************************************
;*                                                       *
;* secshf is log2(hstblk), and is listed below for       *
;* values of hstsiz up to 2048.                          *
;*                                                       *
;*           hstsiz    hstblk    secshf                  *
;*              256         2         1                  *
;*              512         4         2                  *
;*             1024         8         3                  *
;*             2048        16         4                  *
;*                                                       *
```

```
41: ;*******************************************************
42: secshf  equ     2               ;log2(hstblk)
43: cpmspt  equ     hstblk * hstspt ;CP/M sectors/track
44: secmsk  equ     hstblk-1        ;sector mask
45: ;
46: ;*******************************************************
47: ;*                                                     *
48: ;*        BDOS constants on entry to write             *
49: ;*                                                     *
50: ;*******************************************************
51: wrall   equ     0               ;write to allocated
52: wrdir   equ     1               ;write to directory
53: wrual   equ     2               ;write to unallocated
54: ;
55: ;*******************************************************
56: ;*                                                     *
57: ;*      The BIOS entry points given below show the     *
58: ;*      code which is relevant to deblocking only.     *
59: ;*                                                     *
60: ;*******************************************************
61: seldsk:
62:         ;select disk
63:         ;is this the first activation of the drive?
64:         test DL,1               ;lsb = 0?
65:         jnz selset
66:         ;this is the first activation, clear host buff
67:         mov hstact,0
68:         mov unacnt,0
69: selset:
70:         mov al,cl ! cbw         ;put in AX
71:         mov sekdsk,al           ;seek disk number
72:         mov cl,4 ! shl al,cl    ;times 16
73:         add ax,offset dpbase
74:         mov bx,ax
75:         ret
76: ;
77: home:
78:         ;home the selected disk
79:         mov al,hstwrt           ;check for pending write
80:         test al,al
81:         jnz homed
82:         mov hstact,0            ;clear host active flag
83: homed:
84:         mov cx,0                ;now, set track zero
85: ;       (continue HOME routine)
86:         ret
87: ;
88: settrk:
89:         ;set track given by registers CX
90:         mov sektrk,CX           ;track to seek
91:         ret
92: ;
93: setsec:
94:         ;set sector given by register cl
95:         mov seksec,cl           ;sector to seek
```

```
 96:         ret
 97: ;
 98: setdma:
 99:         ;set dma address given by CX
100:         mov dma_off,CX
101:         ret
102: ;
103: setdmab:
104:         ;set segment address given by CX
105:         mov dma_seg,CX
106:         ret
107: ;
108: sectran:
109:         ;translate sector number CX with table at [DX]
110:         test DX,DX      ;test for hard skewed
111:         jz notran       ;(blocked must be hard skewed)
112:         mov BX,CX
113:         add BX,DX
114:         mov BL,[BX]
115:         ret
116: no_tran:
117:         ;hard skewed disk, physical = logical sector
118:         mov BX,CX
119:         ret
120: ;
121: read:
122:         ;read the selected CP/M sector
123:         mov unacnt,0            ;clear unallocated counter
124:         mov readop,1            ;read operation
125:         mov rsflag,1            ;must read data
126:         mov wrtype,wrual        ;treat as unalloc
127:         jmp rwoper              ;to perform the read
128: ;
129: write:
130:         ;write the selected CP/M sector
131:         mov readop,0            ;write operation
132:         mov wrtype,cl
133:         cmp cl,wrual            ;write unallocated?
134:         jnz chkuna              ;check for unalloc
135: ;
136: ;       write to unallocated, set parameters
137: ;
138:         mov unacnt,(blksiz/128) ;next unalloc recs
139:         mov al,sekdsk           ;disk to seek
140:         mov unadsk,al           ;unadsk = sekdsk
141:         mov ax,sektrk
142:         mov unatrk,ax           ;unatrk = sektrk
143:         mov al,seksec
144:         mov unasec,al           ;unasec = seksec
145: ;
146: chkuna:
147:         ;check for write to unallocated sector
148: ;
149:         mov bx,offset unacnt    ;point "UNA" at UNACNT
150:         mov al,una ! test al,al ;any unalloc remain?
```

```
151:          jz alloc                  ;skip if not
152: ;
153: ;        more unallocated records remain
154:          dec al                    ;unacnt = unacnt-1
155:          mov una,al
156:          mov al,sekdsk             ;same disk?
157:          mov BX,offset unadsk
158:          cmp al,una                ;sekdsk = unadsk?
159:          jnz alloc                 ;skip if not
160: ;
161: ;        disks are the same
162:          mov AX, unatrk
163:          cmp AX, sektrk
164:          jnz alloc                 ;skip if not
165: ;
166: ;        tracks are the same
167:          mov al,seksec             ;same sector?
168: ;
169:          mov BX,offset unasec      ;point una at unasec
170: ;
171:          cmp al,una                ;seksec = unasec?
172:          jnz alloc                 ;skip if not
173: ;
174: ;        match, move to next sector for future ref
175:          inc una                   ;unasec = unasec+1
176:          mov al,una,               ;end of track?
177:          cmp al,cpmspt             ;count CP/M sectors
178:          jb noovf                  ;skip if below
179: ;
180: ;        overflow to next track
181:          mov una,0                 ;unasec = 0
182:          inc unatrk                ;unatrk=unatrk+1
183: ;
184: noovf:
185:          ;match found, mark as unnecessary read
186:          mov rsflag,0              ;rsflag = 0
187:          jmps rwoper               ;to perform the write
188: ;
189: alloc:
190:          ;not an unallocated record, requires pre-read
191:          mov unacnt,0              ;unacnt = 0
192:          mov rsflag,1              ;rsflag = 1
193:                                    ;drop through to rwoper
194: ;
195: ;*************************************************
196: ;*                                               *
197: ;*      Common code for READ and WRITE follows   *
198: ;*                                               *
199: ;*************************************************
200: rwoper:
201:          ;enter here to perform the read/write
202:          mov erflag,0              ;no errors (yet)
203:          mov al, seksec            ;compute host sector
204:          mov cl, secshf
205:          shr al,cl
```

```
206:          mov sekhst,al             ;host sector to seek
207: ;
208: ;        active host sector?
209:          mov al,1
210:          xchg al,hstact            ;always becomes 1
211:          test al,al                ;was it already?
212:          jz filhst                 ;fill host if not
213: ;
214: ;        host buffer active, same as seek buffer?
215:          mov al,sekdsk
216:          cmp al,hstdsk             ;sekdsk = hstdsk?
217:          jnz nomatch
218: ;
219: ;        same disk, same track?
220:          mov ax,hsttrk
221:          cmp ax,sektrk             ;host track same as seek track
222:          jnz nomatch
223: ;
224: ;        same disk, same track, same buffer?
225:          mov al,sekhst
226:          cmp al,hstsec             ;sekhst = hstsec?
227:          jz match                  ;skip if match
228: nomatch:
229:          ;proper disk, but not correct sector
230:          mov al, hstwrt
231:          test al,al                ;"dirty" buffer ?
232:          jz filhst                 ;no, don't need to write
233:          call writehst             ;yes, clear host buff
234: ;        (check errors here)
235: ;
236: filhst:
237:          ;may have to fill the host buffer
238:          mov al,sekdsk ! mov hstdsk,al
239:          mov ax,sektrk ! mov hsttrk,ax
240:          mov al,sekhst ! mov hstsec,al
241:          mov al,rsflag
242:          test al,al                ;need to read?
243:          jz filhst1
244: ;
245:          call readhst              ;yes, if 1
246: ;        (check errors here)
247: ;
248: filhst1:
249:          mov hstwrt,0              ;no pending write
250: ;
251: match:
252:          ;copy data to or from buffer depending on "readop"
253:          mov al,seksec             ;mask buffer number
254:          and ax,secmsk             ;least signif bits are masked
255:          mov cl, 7 ! shl ax,cl     ;shift left 7 (* 128 = 2**7)
256: ;
257: ;        ax has relative host buffer offset
258: ;
259:          add ax,offset hstbuf      ;ax has buffer address
260:          mov si,ax                 ;put in source index register
```

```
261:          mov di,dma_off          ;user buffer is dest if readop
262: ;
263:          push DS ! push ES       ;save segment registers
264: ;
265:          mov ES,dma_seg          ;set destseg to the users seg
266:                                  ;SI/DI and DS/ES is swapped
267:                                  ;if write op
268:          mov cx,128/2            ;length of move in words
269:          mov al,readop
270:          test al,al              ;which way?
271:          jnz     rwmove          ;skip if read
272: ;
273: ;        write operation, mark and switch direction
274:          mov hstwrt,1            ;hstwrt = 1 (dirty buffer now)
275:          xchg si,di              ;source/dest index swap
276:          mov ax,DS
277:          mov ES,ax
278:          mov DS,dma_seg          ;setup DS,ES for write
279: ;
280: rwmove:
281:          cld ! rep movs AX,AX    ;move as 16 bit words
282:          pop ES ! pop DS         ;restore segment registers
283: ;
284: ;        data has been moved to/from host buffer
285:          cmp wrtype,wrdir        ;write type to directory?
286:          mov al,erflag           ;in case of errors
287:          jnz return_rw           ;no further processing
288: ;
289: ;        clear host buffer for directory write
290:          test al,al              ;errors?
291:          jnz return_rw           ;skip if so
292:          mov hstwrt,0            ;buffer written
293:          call writehst
294:          mov al,erflag
295: return_rw:
296:          ret
297: ;
298: ;*****************************************************
299: ;*                                                   *
300: ;* WRITEHST performs the physical write to the host   *
301: ;* disk, while READHST reads the physical disk.       *
302: ;*                                                   *
303: ;*****************************************************
304: writehst:
305:          ret
306: ;
307: readhst:
308:          ret
309: ;
310: ;*****************************************************
311: ;*                                                   *
312: ;* Use the GENDEF utility to create disk def tables  *
313: ;*                                                   *
314: ;*****************************************************
315: dpbase   equ     offset $
```

```
316: ;       disk parameter tables go here
317: ;
318: ;*******************************************************
319: ;*                                                     *
320: ;* Uninitialized RAM areas follow, including the       *
321: ;* areas created by the GENDEF utility listed above.   *
322: ;*                                                     *
323: ;*******************************************************
324: sek_dsk rb      1               ;seek disk number
325: sek_trk rw      1               ;seek track number
326: sek_sec rb      1               ;seek sector number
327: ;
328: hst_dsk rb      1               ;host disk number
329: hst_trk rw      1               ;host track number
330: hst_sec rb      1               ;host sector number
331: ;
332: sek_hst rb      1               ;seek shr secshf
333: hst_act rb      1               ;host active flag
334: hst_wrt rb      1               ;host written flag
335: ;
336: una_cnt rb      1               ;unalloc rec cnt
337: una_dsk rb      1               ;last unalloc disk
338: una_trk rw      1               ;last unalloc track
339: una_sec rb      1               ;last unalloc sector
340: ;
341: erflag  rb      1               ;error reporting
342: rsflag  rb      1               ;read sector flag
343: readop  rb      1               ;1 if read operation
344: wrtype  rb      1               ;write operation type
345: dma_seg rw      1               ;last dma segment
346: dma_off rw      1               ;last dma offset
347: hstbuf  rb      hstsiz          ;host buffer
348:         end
```

Appendix B
Sample Random Access Program

This appendix contains a rather extensive and complete example of random access operation. The program listed here performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.CMD, the CCP level command:

```
RANDOM X.DAT
```

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

```
next command?
```

and is followed by operator input, terminated by a carriage return. The input commands take the form

```
nW    nR    Q
```

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

```
type data:
```

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. The only error message is

```
error, try again
```

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at offset 005CH and the default buffer at offset 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development. In fact, with some work, this program could evolve into a simple data base management system.

One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed which first reads a sequential file and extracts a specific field defined by the operator. For example, the command

```
GETKEY NAMES.DAT  LASTNAME 10 20
```

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list, and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an "inverted index" in information retrieval parlance.)

Rename the program shown above as QUERY, and enhance it a bit so that it reads a sorted key file into memory. The command line might appear as:

```
QUERY NAMES.DAT LASTNAME.KEY
```

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, you can find a particular entry quite rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, you examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You'll quickly reach the item you're looking for (in log2(n) steps) where you'll find the corresponding record number. Fetch and display this record at the console, just as we have done in the program shown above.

At this point you're just getting started. With a little more work, you can allow a fixed grouping size which differs from the 128 byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing boolean expressions which compute the set of records which satisfy several relationships, such as a LASTNAME between HARDY and LAUREL, and an AGE less than 45. Display all the records which fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well.

```
 1: ;
 2: ;*******************************************************
 3: ;*                                                     *
 4: ;*    Sample Random Access Program for CP/M-86         *
 5: ;*                                                     *
 6: ;*******************************************************
 7: ;
 8: ;    BDOS Functions
 9: ;
10: coninp  equ     1       ;console input function
11: conout  equ     2       ;console output function
12: pstring equ     9       ;print string until '$'
13: rstring equ     10      ;read console buffer
14: version equ     12      ;return version number
15: openf   equ     15      ;file open function
16: closef  equ     16      ;close function
17: makef   equ     22      ;make file function
18: readr   equ     33      ;read random
19: writer  equ     34      ;write random
20: ;
21: ;   Equates for non graphic characters
22: cr      equ     0dh     ;carriage return
23: lf      equ     0ah     ;line feed
24: ;
25: ;
26: ;  load SP, ready file for random access
27: ;
28:         cseg
29:         pushf                   ;push flags in CCP stack
30:         pop     ax              ;save flags in AX
31:         cli                     ;disable interrupts
32:         mov     bx,ds           ;set SS register to base
33:         mov     ss,bx           ;set SS, SP with interru
34:         mov     sp,offset stack ;    for 80888
35:         push    ax              ;restore the flags
36:         popf
37: ;
38: ;       CP/M-86 initial release returns the file
39: ;       system version number of 2.2:  check is
40: ;       shown below for illustration purposes.
41: ;
42:         mov     cl,version
43:         call    bdos
44:         cmp     al,20h          ;version 2.0 or later?
45:         jnb     versok
46:         ;       bad version, message and go back
47:         mov     dx,offset badver
48:         call    print
49:         jmp     abort
50: ;
51: versok:
52: ;       correct version for random access
53:         mov     cl,openf        ;open default fct
54:         mov     dx,offset fcb
55:         call    bdos
```

```
56:         inc     al              ;err 255 becomes zero
57:         jnz     ready
58: ;
59: ;       cannot open file, so create it
60:         mov     cl,makef
61:         mov     dx,offset fcb
62:         call    bdos
63:         inc     al              ;err 255 becomes zero
64:         jnz     ready
65: ;
66: ;       cannot create file, directory full
67:         mov     dx,offset nospace
68:         call    print
69:         jmp     abort           ;back to ccp
70: ;
71: ;   loop back to "ready" after each command
72: ;
73: ready:
74: ;       file is ready for processing
75: ;
76:         call    readcom         ;read next command
77:         mov     ranrec,dx       ;store input record#
78:         mov     ranovf,0h       ;clear high byte if set
79:         cmp     al,'Q'          ;quit?
80:         jnz     notq
81: ;
82: ;       quit processing, close file
83:         mov     cl,closef
84:         mov     dx,offset fcb
85:         call    bdos
86:         inc     al      ;err 255 becomes 0
87:         jz      error   ;error message, retry
88:         jmps    abort   ;back to ccp
89: ;
90: ;
91: ;  end of quit command, process write
92: ;
93: ;
94: notq:
95: ;       not the quit command, random write?
96:         cmp     al,'W'
97:         jnz     notw
98: ;
99: ;       this is a random write, fill buffer until cr
100:        mov     dx,offset datmsg
101:        call    print           ;data prompt
102:        mov     cx,127          ;up to 127 characters
103:        mov     bx,offset buff  ;destination
104: rloop: ;read next character to buff
105:        push    cx              ;save loop conntrol
106:        push    bx              ;next destination
107:        call    getchr          ;character to AL
108:        pop     bx              ;restore destination
109:        pop     cx              ;restore counter
110:        cmp     al,cr           ;end of line?
```

```
111:            jz      erloop
112: ;          not end, store character
113:            mov     byte ptr [bx],al
114:            inc     bx                  ;next to fill
115:            loop    rloop               ;decrement cx ..loop if
116: erloop:
117: ;          end of read loop, store 00
118:            mov     byte ptr [bx],0h
119: ;
120: ;          write the record to selected record number
121:            mov     cl,writer
122:            mov     dx,offset fcb
123:            call    bdos
124:            or      al,al               ;error code zero?
125:            jz      ready   ;for another record
126:            jmps     error   ;message if not
127: ;
128: ;
129: ;
130: ;  end of write command, process read
131: ;
132: ;
133: notw:
134: ;          not a write command, read record?
135:            cmp     al,'R'
136:            jz      ranread
137:            jmps     error   ;skip if not
138: ;
139: ;          read random record
140: ranread:
141:            mov     cl,readr
142:            mov     dx,offset fcb
143:            call    bdos
144:            or      al,al               ;return code 00?
145:            jz      readok
146:            jmps     error
147: ;
148: ;          read was successful, write to console
149: readok:
150:            call    crlf                ;new line
151:            mov     cx,128              ;max 128 characters
152:            mov     si,offset buff      ;next to get
153: wloop:
154:            lods    al                  ;next character
155:            and     al,07fh             ;mask parity
156:            jnz     wloop1
157:            jmp     ready               ;for another command if
158: wloop1:
159:            push    cx                  ;save counter
160:            push    si                  ;save next to get
161:            cmp     al,' '              ;graphic?
162:            jb      skipw               ;skip output if not grap
163:            call    putchr              ;output character
164: skipw:
165:            pop     si
```

```
166:            pop     cx
167:            loop    wloop           ;decrement CX and check
168:            jmp     ready
169: ;
170: ;
171: ;  end of read command, all errors end-up here
172: ;
173: ;
174: error:
175:            mov     dx,offset errmsg
176:            call    print
177:            jmp     ready
178: ;
179: ;  BDOS entry subroutine
180: bdos:
181:            int     224             ;entry to BDOS if by INT
182:            ret
183: ;
184: abort:                            ;return to CCP
185:            mov     cl,0
186:            call    bdos            ;use function 0 to end e
187: ;
188: ;  utility subroutines for console i/o
189: ;
190: getchr:
191:            ;read next console character to a
192:            mov     cl,coninp
193:            call    bdos
194:            ret
195: ;
196: putchr:
197:            ;write character from a to console
198:            mov     cl,conout
199:            mov     dl,al           ;character to send
200:            call    bdos            ;send character
201:            ret
202: ;
203: crlf:
204:            ;send carriage return line feed
205:            mov     al,cr           ;carriage return
206:            call    putchr
207:            mov     al,lf           ;line feed
208:            call    putchr
209:            ret
210: ;
211: print:
212:            ;print the buffer addressed by dx until $
213:            push    dx
214:            call    crlf
215:            pop     dx              ;new line
216:            mov     cl,pstring
217:            call    bdos            ;print the string
218:            ret
219: ;
220: readcom:
```

```
221:          ;read the next command line to the conbuf
222:          mov     dx,offset prompt
223:          call    print           ;command?
224:          mov     cl,rstring
225:          mov     dx,offset conbuf
226:          call    bdos            ;read command line
227: ;        command line is present, scan it
228:          mov     ax,0            ;start with 0000
229:          mov     bx,offset conlin
230: readc:   mov     dl,[bx]         ;next command character
231:          inc     bx              ;to next command positio
232:          mov     dh,0            ;zero high byte for add
233:          or      dl,dl           ;check for end of comman
234:          jnz     getnum
235:          ret
236: ;        not zero, numeric?
237: getnum:
238:          sub     dl,'0'
239:          cmp     dl,10           ;carry if numeric
240:          jnb     endrd
241:          mov     cl,10
242:          mul     cl              ;multipy accumulator by
243:          add     ax,dx           ;+digit
244:          jmps     readc          ;for another char
245: endrd:
246: ;        end of read, restore value in a and return value
247:          mov     dx,ax           ;return value  in DX
248:          mov     al,-1[bx]
249:          cmp     al,'a'          ;check for lower case
250:          jnb     transl
251:          ret
252: transl:  and     al,5fH  ;translate to upper case
253:          ret
254: ;
255: ;
256: ; Template for Page 0 of Data Group
257: ;   Contains default FCB and DMA buffer
258: ;
259:          dseg
260:          org     05ch
261: fcb      rb      33              ;default file control bl
262: ranrec   rw      1               ;random record position
263: ranovf   rb      1               ;high order (overflow) b
264: buff     rb      128             ;default DMA buffer
265: ;
266: ;  string data area for console messages
267: badver        db      'sorry, you need cp/m version 2$'
268: nospace       db      'no directory space$'
269: datmsg        db      'type data: $'
270: errmsg        db      'error, try again.$'
271: prompt        db      'next command? $'
272: ;
273: ;
274: ;    fixed and variable data area
275: ;
```

```
276: conbuf db      conlen  ;length of console buffer
277: consiz rs      1       ;resulting size after read
278: conlin rs      32      ;length 32 buffer
279: conlen equ     offset $ - offset consiz
280: ;
281:        rs       31      ;16 level stack
282: stack  rb       1
283:        db       0       ;end byte for GENCMD
284:        end
```

Appendix C
Listing of the Boot ROM

```
*********************************************************
*                                                       *
* This is the original BOOT ROM distributed with CP/M   *
* for the SBC 86/12 and 204 Controller.  The listing    *
* is truncated on the right, but can be reproduced by   *
* assembling ROM.A86 from the distribution disk.  Note  *
* that the distributed source file should always be     *
* referenced for the latest version                     *
*                                                       *
*********************************************************
                    ;
                    ; ROM bootstrap for CP/M-86 on an iSBC86/12
                    ;                with the
                    ;   Intel SBC 204 Floppy Disk Controller
                    ;
                    ;       Copyright (C) 1980,1981
                    ;       Digital Research, Inc.
                    ;       Box 579, Pacific Grove
                    ;       California, 93950
                    ;
                    ;*********************************************
                    ;* This is the BOOT ROM which is initiated   *
                    ;* by a system reset.  First, the ROM moves  *
                    ;* a copy of its data area to RAM at loca-   *
                    ;* tion 00000H, then initializes the segment*
                    ;* registers and the stack pointer.  The     *
                    ;* various peripheral interface chips on the*
                    ;* SBC 86/12 are initialized.  The 8251      *
                    ;* serial interface is configured for a 9600*
                    ;* baud asynchronous terminal, and the in-   *
                    ;* terrupt controller is setup for inter-    *
                    ;* rupts 10H-17H (vectors at 00040H-0005FH)  *
                    ;* and edge-triggered auto-EOI (end of in-   *
                    ;* terrupt) mode with all interrupt levels   *
                    ;* masked-off.  Next, the SBC 204 Diskette   *
                    ;* controller is initialized, and track 1    *
                    ;* sector 1 is read to determine the target  *
                    ;* paragraph address for LOADER.  Finally,   *
                    ;* the LOADER on track 0 sectors 2-26 and    *
                    ;* track 1 sectors 1-26 is read into the     *
                    ;* target address.  Control then transfers   *
                    ;* to LOADER.  This program resides in two   *
                    ;* 2716 EPROM's (2K each) at location        *
                    ;* 0FF000H on the SBC 86/12 CPU board.  ROM  *
                    ;* 0 contains the even memory locations, and*
                    ;* ROM 1 contains the odd addresses.  BOOT   *
                    ;* ROM uses RAM between 00000H and 000FFH    *
                    ;* (absolute) for a scratch area, along with*
                    ;* the sector 1 buffer.                      *
                    ;*********************************************
```

```
00FF            true            equ     0ffh
FF00            false           equ     not true
                ;
00FF            debug           equ     true
                ;debug = true indicates bootstrap is in same roms
                ;with SBC 957 "Execution Vehicle" monitor
                ;at FE00:0 instead of FF00:0
                ;
000D            cr              equ     13
000A            lf              equ     10
                ;
                ;       disk ports and commands
                ;
00A0            base204         equ     0a0h
00A0            fdccom          equ     base204+0
00A0            fdcstat         equ     base204+0
00A1            fdcparm         equ     base204+1
00A1            fdcrslt         equ     base204+1
00A2            fdcrst          equ     base204+2
00A4            dmacadr         equ     base204+4
00A5            dmaccont        equ     base204+5
00A6            dmacscan        equ     base204+6
00A7            dmacsadr        equ     base204+7
00A8            dmacmode        equ     base204+8
00A8            dmacstat        equ     base204+8
00A9            fdcsel          equ     base204+9
00AA            fdcsegment      equ     base204+10
00AF            reset204        equ     base204+15
                ;
                ;actual console baud rate
2580            baud_rate       equ     9600
                ;value for 8253 baud counter
0008            baud            equ     768/(baud_rate/100)
                ;
00DA            csts            equ     0DAh    ;i8251 status port
00D8            cdata           equ     0D8h    ; "    data port
                ;
00D0            tch0            equ     0D0h    ;8253 PIC channel 0
00D2            tch1            equ     tch0+2  ;ch 1 port
00D4            tch2            equ     tch0+4  ;ch 2 port
00D6            tcmd            equ     tch0+6  ;8253 command port
                ;
00C0            icp1            equ     0C0h    ;8259a port 0
00C2            icp2            equ     0C2h    ;8259a port 1
                ;
                ;
                        IF NOT DEBUG
                ROMSEG          EQU     0FF00H  ;normal
                        ENDIF
                ;
                        IF DEBUG                ;share prom with SB
FE00            ROMSEG          EQU     0FE00H
                        ENDIF
                ;
                ;
```

```
                        ;       This long jump prom'd in by hand
                        ;       cseg    0ffffh          ;reset goes to here
                        ;       JMPF    BOTTOM          ;boot is at bottom
                        ;       EA 00 00 00 FF          ;cs = bottom of pro
                        ;                                       ip = 0
                        ;       EVEN PROM       ODD PROM
                        ;       7F8 - EA        7F8 - 00
                        ;       7F9 - 00        7F9 - 00
                        ;       7FA - FF                ;this is not done i
                        ;
  FE00                          cseg    romseg
                        ;
                        ;First, move our data area into RAM at 0000:0200
                        ;
 0000 8CC8                      mov ax,cs
 0002 8ED8                      mov ds,ax       ;point DS to CS for source
 0004 BE3F01                    mov SI,drombegin        ;start of data
 0007 BF0002                    mov DI,offset ram_start ;offset of destinat
 000A B80000                    mov ax,0
 000D 8EC0                      mov es,ax       ;destination segment is 000
 000F B9E600                    mov CX,data_length      ;how much to move i
 0012 F3A4                      rep movs al,al          ;move out of eprom
                        ;
 0014 B80000                    mov ax,0
 0017 8ED8                      mov ds,ax       ;data segment now in RAM
 0019 8ED0                      mov ss,ax
 001B BC2A03                    mov sp,stack_offset     ;Initialize stack s
 001E FC                        cld                     ;clear the directio
                        ;
                                IF NOT DEBUG
                        ;
                        ;Now, initialize the console USART and baud rate
                        ;
                                mov al,0Eh
                                out csts,al     ;give 8251 dummy mode
                                mov al,40h
                                out csts,al     ;reset 8251 to accept mode
                                mov al,4Eh
                                out csts,al     ;normal 8 bit asynch mode,
                                mov al,37h
                                out csts,al     ;enable Tx & Rx
                                mov al,0B6h
                                out tcmd,al     ;8253 ch.2 square wave mode
                                mov ax,baud
                                out tch2,al     ;low of the baud rate
                                mov al,ah
                                out tch2,al     ;high of the baud rate
                        ;
                                ENDIF
                        ;
                        ;Setup the 8259 Programmable Interrupt Controller
                        ;
 001F B013                      mov al,13h
 0021 E6C0                      out icp1,al     ;8259a ICW 1  8086 mode
 0023 B010                      mov al,10h
```

```
0025 E6C2                       out icp2,al     ;8259a ICW 2  vector @ 40-5
0027 B01F                       mov al,1Fh
0029 E6C2                       out icp2,al     ;8259a ICW 4  auto EOI mast
002B B0FF                       mov al,0FFh
002D E6C2                       out icp2,al     ;8259a OCW 1  mask all leve
                        ;
                        ;Reset and initialize the iSBC 204 Diskette Interfa
                        ;
                        restart:        ;also come back here on fatal error
002F E6AF                       out reset204,AL ;reset iSBC 204 logic and
0031 B001                       mov AL,1
0033 E6A2                       out fdcrst,AL   ;give 8271 FDC
0035 B000                       mov al,0
0037 E6A2                       out fdcrst,AL   ;  a reset command
0039 BB1502                     mov BX,offset specs1
003C E8E100                     CALL sendcom  ;program
003F BB1B02                     mov BX,offset specs2
0042 E8DB00                     CALL sendcom  ;  Shugart SA-800 drive
0045 BB2102                     mov BX,offset specs3
0048 E8D500                     call sendcom  ;    characteristics
004B BB1002             homer:  mov BX,offset home
004E E85800                     CALL execute    ;home drive 0
                        ;
0051 BB2A03                     mov bx,sector1  ;offset for first sector DM
0054 B80000                     mov ax,0
0057 8EC0                       mov es,ax       ;segment "    "     "     "
0059 E8A700                     call setup_dma
                        ;
005C BB0202                     mov bx,offset read0
005F E84700                     call execute    ;get T0 S1
                        ;
0062 8E062D03                   mov es,ABS
0066 BB0000                     mov bx,0        ;get loader load address
0069 E89700                     call setup_dma  ;setup DMA to read loader
                        ;
006C BB0602                     mov bx,offset read1
006F E83700                     call execute    ;read track 0
0072 BB0B02                     mov bx,offset read2
0075 E83100                     call execute    ;read track 1
                        ;
0078 8C06E802                   mov leap_segment,ES
                        ;       setup far jump vector
007C C706E6020000               mov leap_offset,0
                        ;
                        ;       enter LOADER
0082 FF2EE602                   jmpf dword ptr leap_offset
                        ;
                        pmsg:
0086 8A0F                       mov cl,[BX]
0088 84C9                       test cl,cl
008A 7476                       jz return
008C E80400                     call conout
008F 43                         inc BX
0090 E9F3FF                     jmp pmsg
                        ;
```

```
                conout:
0093 E4DA               in al,csts
0095 A801               test al,1
0097 74FA               jz conout
0099 8AC1               mov al,cl
009B E6D8               out cdata,al
009D C3                 ret
                ;
                conin:
009E E4DA               in al,csts
00A0 A802               test al,2
00A2 74FA               jz conin
00A4 E4D8               in al,cdata
00A6 247F               and al,7Fh
00A8 C3                 ret
                ;
                ;
                ;
                execute:        ;execute command string @ [BX]
                                ;<BX> points to length,
                                ;followed by Command byte
                                ;followed by length-1 parameter byt
                ;
00A9 891E0002           mov     lastcom,BX      ;remember what it w
                retry:                          ;retry if not ready
00AD E87000             call    sendcom         ;execute the comman
                                                ;now, let's see wha
                                                ;of status poll was
                                                ;for that command t
00B0 8B1E0002           mov     BX,lastcom      ;point to command s
00B4 8A4701             mov     AL,1[BX]        ;get command op cod
00B7 243F               and     AL,3fh          ;drop drive code bi
00B9 B90008             mov     CX,0800h        ;mask if it will be
00BC 3C2C               cmp     AL,2ch          ;see if interrupt t
00BE 720B               jb      execpoll
00C0 B98080             mov     CX,8080h        ;else we use "not c
00C3 240F               and     AL,0fh          ;unless . . .
 0C5 3C0C               cmp     AL,0ch          ;there isn't
00C7 B000               mov AL,0
00C9 7737               ja return               ;any result at all
                ;
                execpoll:       ;poll for bit in b, toggled with c
00CB E4A0               in AL,FDCSTAT
00CD 22C5               and AL,CH
00CF 32C174F8           xor AL,CL ! JZ execpoll
                ;
00D3 E4A1               in      AL,fdcrslt      ;get result registe
00D5 241E               and     AL,1eh          ;look only at resul
00D7 7429               jz      return          ;zero means it was
                ;
00D9 3C10               cmp al,10h
00DB 7513               jne fatal               ;if other than "Not
                ;
00DD BB1302             mov bx,offset rdstat
00E0 E83D00             call sendcom            ;perform read statu
```

```
                rd_poll:
00E3 E4A0               in al,fdc_stat
00E5 A880               test al,80h             ;wait for command n
00E7 75FA               jnz rd_poll
00E9 8B1E0002           mov bx,last_com         ;recover last attem
00ED E9BDFF             jmp retry               ;and try it over ag
                ;
                fatal:                          ; fatal error
00F0 B400               mov ah,0
00F2 8BD8               mov bx,ax               ;make 16 bits
00F4 8B9F2702           mov bx,errtbl[BX]
                ;       print appropriate error message
00F8 E88BFF             call pmsg
00FB E8A0FF             call conin              ;wait for key strik
00FE 58                 pop ax                  ;discard unused ite
00FF E92DFF             jmp restart             ;then start all ove
                ;
                return:
0102 C3                 RET                     ;return from EXECUT
                ;
                setupdma:
0103 B004               mov AL,04h
0105 E6A8               out dmacmode,AL         ;enable dmac
0107 B000               mov al,0
0109 E6A5               out dmaccont,AL         ;set first (dummy)
010B B040               mov AL,40h
010D E6A5               out dmaccont,AL         ;force read data mo
010F 8CC0               mov AX,ES
0111 E6AA               out fdcsegment,AL
0113 8AC4               mov AL,AH
0115 E6AA               out fdcsegment,AL
0117 8BC3               mov AX,BX
0119 E6A4               out dmacadr,AL
011B 8AC4               mov AL,AH
011D E6A4               out dmacadr,AL
011F C3                 RET
                ;
                ;
                ;
                sendcom:        ;routine to send a command string t
0120 E4A0               in AL,fdcstat
0122 2480               and AL,80h
0124 75FA               jnz sendcom     ;insure command not busy
0126 8A0F               mov CL,[BX]     ;get count
0128 43                 inc BX
0129 8A07               mov al,[BX]     ;point to and fetch command
012B E6A0               out fdccom,AL   ;send command
                parmloop:
012D FEC9               dec CL
012F 74D1               jz return       ;see if any (more) paramete
0131 43                 inc BX          ;point to next parameter
                parmpoll:
0132 E4A0               in AL,fdcstat
0134 2420               and AL,20h
0136 75FA               jnz parmpoll    ;loop until parm not full
```

```
0138 8A07                   mov AL,[BX]
013A E6A1                   out fdcparm,AL  ;output next parameter
013C E9EEFF                 jmp parmloop    ;go see about another
                    ;
                    ;
                    ;       Image of data to be moved to RAM
                    ;
  013F              drombegin equ offset $
                    ;
013F 0000           clastcom        dw      0000h   ;last command
                    ;
0141 03             creadstring     db      3       ;length
0142 52                             db      52h     ;read function code
0143 00                             db      0       ;track #
0144 01                             db      1       ;sector #
                    ;
0145 04             creadtrk0       db      4
0146 53                             db      53h     ;read multiple
0147 00                             db      0       ;track 0
0148 02                             db      2       ;sectors 2
0149 19                             db      25      ;through 26
                    ;
014A 04             creadtrk1       db      4
014B 53                             db      53h
014C 01                             db      1       ;track 1
014D 01                             db      1       ;sectors 1
014E 1A                             db      26      ;through 26
                    ;
014F 026900         chome0          db      2,69h,0
0152 016C           crdstat0        db      1,6ch
0154 05350D         cspecs1         db      5,35h,0dh
0157 0808E9                         db      08h,08h,0e9h
015A 053510         cspecs2         db      5,35h,10h
015D FFFFFF                         db      255,255,255
0160 053518         cspecs3         db      5,35h,18h
0163 FFFFFF                         db      255,255,255
                    ;
0166 4702           cerrtbl dw      offset er0
0168 4702                   dw      offset er1
016A 4702                   dw      offset er2
016C 4702                   dw      offset er3
016E 5702                   dw      offset er4
0170 6502                   dw      offset er5
0172 7002                   dw      offset er6
0174 7F02                   dw      offset er7
0176 9002                   dw      offset er8
0178 A202                   dw      offset er9
017A B202                   dw      offset erA
017C C502                   dw      offset erB
017E D302                   dw      offset erC
0180 4702                   dw      offset erD
0182 4702                   dw      offset erE
0184 4702                   dw      offset erF
                    ;
0186 0D0A4E756C6C Cer0      db      cr,lf,'Null Error ??',0
```

```
     204572726F72
     203F3F00
  0186            Cer1    equ     cer0
  0186            Cer2    equ     cer0
  0186            Cer3    equ     cer0
0196 0D0A436C6F63 Cer4    db      cr,lf,'Clock Error',0
     6B204572726F
     7200
01A4 0D0A4C617465 Cer5    db      cr,lf,'Late DMA',0
     20444D4100
01AF 0D0A49442043 Cer6    db      cr,lf,'ID CRC Error',0
     524320457272
     6F7200
01BE 0D0A44617461 Cer7    db      cr,lf,'Data CRC Error',0
     204352432045
     72726F7200
01CF 0D0A44726976 Cer8    db      cr,lf,'Drive Not Ready',0
     65204E6F7420
     526561647900
01E1 0D0A57726974 Cer9    db      cr,lf,'Write Protect',0
     652050726F74
     65637400
01F1 0D0A54726B20 CerA    db      cr,lf,'Trk 00 Not Found',0
     3030204E6F74
     20466F756E64
     00
0204 0D0A57726974 CerB    db      cr,lf,'Write Fault',0
     65204661756C
     7400
0212 0D0A53656374 CerC    db      cr,lf,'Sector Not Found',0
     6F72204E6F74
     20466F756E64
     00
  0186            CerD    equ     cer0
  0186            CerE    equ     cer0
  0186            CerF    equ     cer0
                  ;
  0225            dromend equ offset $
                  ;
  00E6            data_length     equ dromend-drombegin
                  ;
                  ;       reserve space in RAM for data area
                  ;       (no hex records generated here)
                  ;
  0000                    dseg    0
                          org     0200h
                  ;
  0200            ram_start       equ     $
0200              lastcom         rw      1       ;last command
0202              read0           rb      4       ;read track 0 secto
0206              read1           rb      5       ;read T0 S2-26
020B              read2           rb      5       ;read T1 S1-26
0210              home            rb      3       ;home drive 0
0213              rdstat          rb      2       ;read status
0215              specs1          rb      6
```

```
 21B                specs2          rb      6
0221                specs3          rb      6
0227                errtbl          rw      16
0247                er0             rb      length cer0     ;16
  0247              er1             equ     er0
  0247              er2             equ     er0
  0247              er3             equ     er0
0257                er4             rb      length cer4     ;14
0265                er5             rb      length cer5     ;11
0270                er6             rb      length cer6     ;15
027F                er7             rb      length cer7     ;17
0290                er8             rb      length cer8     ;18
02A2                er9             rb      length cer9     ;16
02B2                erA             rb      length cerA     ;19
02C5                erB             rb      length cerB     ;14
02D3                erC             rb      length cerC     ;19
  0247              erD             equ     er0
  0247              erE             equ     er0
  0247              erF             equ     er0
                    ;
02E6                leap_offset     rw      1
02E8                leap_segment    rw      1
                    ;
                    ;
02EA                                rw      32      ;local stack
  032A              stack_offset    equ     offset $;stack from here do
                    ;
                    ;               T0 S1 read in here
  032A              sector1         equ offset $
                    ;
032A                Ty              rb      1
032B                Len             rw      1
032D                Abs             rw      1       ;ABS is all we care
032F                Min             rw      1
0331                Max             rw      1
                                    end
```

Appendix D
LDBIOS Listing

```
*****************************************************
*                                                   *
* This the the LOADER BIOS, derived from the BIOS   *
* program by enabling the "loader_bios" condi-      *
* tional assembly switch.  The listing has been     *
* edited to remove portions which are duplicated    *
* in the BIOS listing which appears in Appendix D   *
* where elipses "..." denote the deleted portions   *
* (the listing is truncated on the right, but can   *
* be reproduced by assembling the BIOS.A86 file     *
* provided with CP/M-86)                            *
*                                                   *
*****************************************************

;*************************************************
;*                                               *
;* Basic Input/Output System (BIOS) for          *
;* CP/M-86 Configured for iSBC 86/12 with        *
;* the iSBC 204 Floppy Disk Controller           *
;*                                               *
;* (Note:  this file contains both embedded      *
;* tabs and blanks to minimize the list file     *
;* width for printing purposes.  You may wish    *
;* to expand the blanks before performing        *
;* major editing.)                               *
;*************************************************

;       Copyright (C) 1980,1981
;       Digital Research, Inc.
;       Box 579, Pacific Grove
;       California, 93950
;
;       (Permission is hereby granted to use
;       or abstract the following program in
;       the implementation of CP/M, MP/M or
;       CP/NET for the 8086 or 8088 Micro-
;       processor)

FFFF            true            equ -1
0000            false           equ not true
```

```
                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.  Blc_list is true if we     *
                ;* have a serial printer attached to BLC8538 *
                ;* Bdos_int is interrupt used for earlier    *
                ;* versions.                                 *
                ;*                                           *
                ;*********************************************

FFFF            loader_bios     equ true
FFFF            blc_list        equ true
00E0            bdos_int        equ 224 ;reserved BDOS Interrupt

                        IF      not loader_bios
                ;-------------------------------------------
                ;|                                         |
                        . . .
                ;|                                         |
                ;-------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;-------------------------------------------
                ;|                                         |
1200            bios_code       equ 1200h ;start of LDBIOS
0003            ccp_offset      equ 0003h ;base of CPMLOADER
0406            bdos_ofst       equ 0406h ;stripped BDOS entry
                ;|                                         |
                ;-------------------------------------------
                        ENDIF   ;loader_bios
                        . . .

                        cseg
                        org     ccpoffset
                ccp:
                        org     bios_code

                ;*********************************************
                ;*                                           *
                ;* BIOS Jump Vector for Individual Routines  *
                ;*                                           *
                ;*********************************************

1200 E93C00      jmp INIT       ;Enter from BOOT ROM or LOADER
1203 E96100      jmp WBOOT      ;Arrive here from BDOS call 0
                        . . .
1239 E96400      jmp GETIOBF    ;return I/O map byte (IOBYTE)
123C E96400      jmp SETIOBF    ;set I/O map byte (IOBYTE)
```

```
                ;*********************************************
                ;*                                           *
                ;* INIT Entry Point, Differs for LDBIOS and  *
                ;* BIOS, according to "Loader_Bios" value    *
                ;*                                           *
                ;*********************************************

                INIT:   ;print signon message and initialize hardwa
123F 8CC8               mov ax,cs       ;we entered with a JMPF so
1241 8ED0               mov ss,ax       ; CS: as the initial value
1243 8ED8               mov ds,ax       ;       DS:,
1245 8EC0               mov es,ax       ;       and ES:
                        ;use local stack during initialization
1247 BCA916             mov sp,offset stkbase
124A FC                 cld             ;set forward direction

                        IF      not loader_bios
                ;-----------------------------------------------
                ;|                                             |
                        ; This is a BIOS for the CPM.SYS file.
                ;|      . . .                                  |
                ;-----------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;-----------------------------------------------
                ;|                                             |
                        ;This is a BIOS for the LOADER
124B 1E                 push ds         ;save data segment
124C B80000             mov ax,0
124F 8ED8               mov ds,ax       ;point to segment zero
                        ;BDOS interrupt offset
1251 C70680030604       mov bdos_offset,bdos_ofst
1257 8C0E8203           mov bdos_segment,CS ;bdos interrupt segment
125B 1F                 pop ds          ;restore data segment
                ;|                                             |
                ;-----------------------------------------------
                        ENDIF   ;loader_bios

125C BB1514             mov bx,offset signon
125F E85A00             call pmsg       ;print signon message
1262 B100               mov cl,0        ;default to dr A: on coldst
1264 E99CED             jmp ccp         ;jump to cold start entry o

1267 E99FED     WBOOT:  jmp ccp+6       ;direct entry to CCP at com

                        IF      not loader_bios
                ;-----------------------------------------------
                ;|                                             |
                        . . .
                ;|                                             |
                ;-----------------------------------------------
                        ENDIF   ;not loader_bios
```

```
                ;*********************************************
                ;*                                           *
                ;*   CP/M Character I/O Interface Routines   *
                ;*   Console is Usart (i8251a) on iSBC 86/12 *
                ;*   at ports D8/DA                          *
                ;*                                           *
                ;*********************************************

                CONST:          ;console status
126A E4DA               in al,csts
                        . . .
                const_ret:
1272 C3                 ret             ;Receiver Data Available

                CONIN:                  ;console input
1273 E8F4FF             call const
                        . . .
                 CONOUT:         ;console output
127D E4DA               in al,csts
                        . . .

                LISTOUT:                ;list device output

                        IF      blc_list
                ;-----------------------------------------
                ;|                                       |
1288 E80700             call LISTST
                          . . .
                ;|                                       |
                ;-----------------------------------------
                        ENDIF   ;blc_list

1291 C3                 ret

                LISTST:                 ;poll list status

                        IF      blc_list
                ;-----------------------------------------
                ;|                                       |
1292 E441               in al,lsts
                        . . .
                ;|                                       |
                ;-----------------------------------------
                        ENDIF   ;blc_list

129C C3                 ret

                PUNCH:  ;not implemented in this configuration
                READER:
129D B01A               mov al,1ah
129F C3                 ret             ;return EOF for now
```

```
                GETIOBF:
12A0 B000               mov al,0        ;TTY: for consistency
12A2 C3                 ret             ;IOBYTE not implemented

                SETIOBF:
12A3 C3                 ret             ;iobyte not implemented

                zero_ret:
12A4 2400               and al,0
12A6 C3         ret                     ;return zero in AL and flag

                ; Routine to get and echo a console character
                ;       and shift it to upper case

                uconecho:
12A7 E8C9FF             call CONIN      ;get a console character
                        . . .
                ;***********************************************
                ;*                                             *
                ;*          Disk Input/Output Routines         *
                ;*                                             *
                ;***********************************************

                SELDSK:         ;select disk given by register CL
12CA BB0000             mov bx,0000h
                        . . .

                HOME:   ;move selected disk to home position (Track
12EB C606311500         mov trk,0       ;set disk i/o to track zero
                        . . .

                SETTRK: ;set track address given by CX
1300 880E3115           mov trk,cl      ;we only use 8 bits of trac
1304 C3                 ret

                SETSEC: ;set sector number given by cx
1305 880E3215           mov sect,cl     ;we only use 8 bits of sect
1309 C3                 ret

                SECTRAN: ;translate sector CX using table at [DX]
130A 8BD9               mov bx,cx
                        . . .

                SETDMA: ;set DMA offset given by CX
1311 890E2A15           mov dma_adr,CX
1315 C3                 ret

                SETDMAB: ;set DMA segment given by CX
1316 890E2C15           mov dma_seg,CX
131A C3                 ret
                ;
                GETSEGT:  ;return address of physical memory table
131B BB3815             mov bx,offset seg_table
131E C3                 ret
```

```
                ;*********************************************
                ;*                                           *
                ;*  All disk I/O parameters are setup:  the  *
                ;*  Read and Write entry points transfer one *
                ;*  sector of 128 bytes to/from the current  *
                ;*  DMA address using the current disk drive *
                ;*                                           *
                ;*********************************************

                READ:
131F B012               mov al,12h      ;basic read sector command
1321 EB02               jmps r_w_common

                WRITE:
1323 B00A               mov al,0ah      ;basic write sector command

                r_w_common:
1325 BB2F15             mov bx,offset io_com ;point to command stri
                        . . .

                ;*********************************************
                ;*                                           *
                ;*                Data Areas                 *
                ;*                                           *
                ;*********************************************
  1415          data_offset     equ offset $

                        dseg
                        org     data_offset     ;contiguous with co

                        IF      loader_bios
                ;-------------------------------------------
                ;|                                         |
1415 0D0A0D0A   signon  db      cr,lf,cr,lf
1419 43502F4D2D38       db      'CP/M-86 Version 2.2',cr,lf,0
     362056657273
     696F6E20322E
     320D0A00
                ;|                                         |
                ;-------------------------------------------
                        ENDIF   ;loader_bios

                        IF      not loader_bios
                ;-------------------------------------------
                ;|                                         |
                        . . .
                ;|                                         |
                ;-------------------------------------------
                        ENDIF   ;not loader_bios

142F 0D0A486F6D65 bad_hom db    cr,lf,'Home Error',cr,lf,0
                        . . .
=                       include singles.lib ;read in disk definitio
=               ;               DISKS 2
```

```
  1541              dpbase  equ     $               ;Base of Disk Param
                            . . .
=1668 00                    db      0               ;Marks End of Modul

 1669               loc_stk rw  32  ;local stack for initialization
   16A9             stkbase equ offset $

                            . . .
 16A9 00                    db 0    ;fill last address for GENCMD

                    ;***********************************************
                    ;*                                             *
                    ;*          Dummy Data Section                 *
                    ;*                                             *
                    ;***********************************************
   0000                     dseg    0       ;absolute low memory
                            org     0       ;(interrupt vectors)
                            . . .
                            END
```

Appendix E
BIOS Listing

```
**************************************************
*                                                *
* This is the CP/M-86 BIOS, derived from the BIOS *
* program by disabling the "loader_bios" condi-  *
* tional assembly switch.  The listing has been  *
* truncated on the right, but can be reproduced  *
* by assembling the BIOS.A86 file provided with  *
* CP/M-86.  This BIOS allows CP/M-86 operation   *
* with the Intel SBC 86/12 with the SBC 204 con- *
* troller.  Use this BIOS, or the skeletal CBIOS *
* listed in Appendix E, as the basis for a cus-  *
* tomized implementation of CP/M-86.             *
* provided with CP/M-86)                         *
*                                                *
**************************************************
```

```
;**********************************************
;*                                            *
;* Basic Input/Output System (BIOS) for       *
;* CP/M-86 Configured for iSBC 86/12 with     *
;* the iSBC 204 Floppy Disk Controller        *
;*                                            *
;* (Note:  this file contains both embedded   *
;* tabs and blanks to minimize the list file  *
;* width for printing purposes.  You may wish*
;* to expand the blanks before performing     *
;* major editing.)                            *
;**********************************************

;       Copyright (C) 1980,1981
;       Digital Research, Inc.
;       Box 579, Pacific Grove
;       California, 93950
;
;       (Permission is hereby granted to use
;       or abstract the following program in
;       the implementation of CP/M, MP/M or
;       CP/NET for the 8086 or 8088 Micro-
;       processor)

FFFF            true            equ -1
0000            false           equ not true
```

```
                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.  Blc_list is true if we     *
                ;* have a serial printer attached to BLC8538 *
                ;* Bdos_int is interrupt used for earlier    *
                ;* versions.                                 *
                ;*                                           *
                ;*********************************************

0000            loader_bios     equ false
FFFF            blc_list        equ true
00E0            bdos_int        equ 224 ;reserved BDOS Interrupt

                        IF      not loader_bios
                ;-------------------------------------------
                ;|                                         |
2500            bios_code       equ 2500h
0000            ccp_offset      equ 0000h
0B06            bdos_ofst       equ 0B06h ;BDOS entry point
                ;|                                         |
                ;-------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;-------------------------------------------
                ;|                                         |
                bios_code       equ 1200h ;start of LDBIOS
                ccp_offset      equ 0003h ;base of CPMLOADER
                bdos_ofst       equ 0406h ;stripped BDOS entry
                ;|                                         |
                ;-------------------------------------------
                        ENDIF   ;loader_bios

00DA            csts            equ 0DAh  ;i8251 status port
00D8            cdata           equ 0D8h  ;    "  data port

                        IF      blc_list
                ;-------------------------------------------
                ;|                                         |
0041            lsts            equ 41h ;2651 No. 0 on BLC8538 stat
0040            ldata           equ 40h ; "   "   "  "    "    data
0060            blc_reset       equ 60h ;reset selected USARTS on B
                ;|                                         |
                ;-------------------------------------------
                        ENDIF   ;blc_list

                ;*********************************************
                ;*                                           *
                ;*    Intel iSBC 204 Disk Controller Ports   *
                ;*                                           *
                ;*********************************************
```

```
00A0                base204         equ 0a0h        ;SBC204 assigned ad

00A0                fdc_com         equ base204+0   ;8271 FDC out comma
00A0                fdc_stat        equ base204+0   ;8271 in status
00A1                fdc_parm        equ base204+1   ;8271 out parameter
00A1                fdc_rslt        equ base204+1   ;8271 in result
00A2                fdc_rst         equ base204+2   ;8271 out reset
00A4                dmac_adr        equ base204+4   ;8257 DMA base addr
00A5                dmac_cont       equ base204+5   ;8257 out control
00A6                dmac_scan       equ base204+6   ;8257 out scan cont
00A7                dmac_sadr       equ base204+7   ;8257 out scan addr
00A8                dmac_mode       equ base204+8   ;8257 out mode
00A8                dmac_stat       equ base204+8   ;8257 in status
00A9                fdc_sel         equ base204+9   ;FDC select port (n
00AA                fdc_segment     equ base204+10  ;segment address re
00AF                reset_204       equ base204+15  ;reset entire inter

000A                max_retries     equ 10          ;max retries on dis
                                                    ;before perm error
000D                cr              equ 0dh         ;carriage return
000A                lf              equ 0ah         ;line feed

                        cseg
                        org     ccpoffset
                ccp:
                        org     bios_code

                ;***********************************************
                ;*                                             *
                ;* BIOS Jump Vector for Individual Routines    *
                ;*                                             *
                ;***********************************************

2500 E93C00             jmp INIT        ;Enter from BOOT ROM or LOADER
2503 E98400             jmp WBOOT       ;Arrive here from BDOS call 0
2506 E99000             jmp CONST       ;return console keyboard status
2509 E99600             jmp CONIN       ;return console keyboard char
250C E99D00             jmp CONOUT      ;write char to console device
250F E9A500             jmp LISTOUT     ;write character to list device
2512 E9B700             jmp PUNCH       ;write character to punch device
2515 E9B400             jmp READER      ;return char from reader device
2518 E9FF00             jmp HOME        ;move to trk 00 on cur sel drive
251B E9DB00             jmp SELDSK      ;select disk for next rd/write
251E E90E01             jmp SETTRK      ;set track for next rd/write
2521 E91001             jmp SETSEC      ;set sector for next rd/write
2524 E91901             jmp SETDMA      ;set offset for user buff (DMA)
2527 E92401             jmp READ        ;read a 128 byte sector
252A E92501             jmp WRITE       ;write a 128 byte sector
252D E99100             jmp LISTST      ;return list status
2530 E90601             jmp SECTRAN     ;xlate logical->physical sector
2533 E90F01             jmp SETDMAB     ;set seg base for buff (DMA)
2536 E91101             jmp GETSEGT     ;return offset of Mem Desc Table
2539 E99300             jmp GETIOBF     ;return I/O map byte (IOBYTE)
253C E99300             jmp SETIOBF     ;set I/O map byte (IOBYTE)
```

```
                ;***********************************************
                ;*                                             *
                ;* INIT Entry Point, Differs for LDBIOS and    *
                ;* BIOS, according to "Loader_Bios" value      *
                ;*                                             *
                ;***********************************************

                INIT:   ;print signon message and initialize hardwa
253F 8CC8               mov ax,cs       ;we entered with a JMPF so
2541 8ED0               mov ss,ax       ; CS: as the initial value
2543 8ED8               mov ds,ax       ;       DS:,
2545 8EC0               mov es,ax       ;       and ES:
                        ;use local stack during initialization
2547 BCE429             mov sp,offset stkbase
254A FC                 cld             ;set forward direction

                        IF      not loader_bios
                ;-----------------------------------------------
                ;|                                             |
                        ; This is a BIOS for the CPM.SYS file.
                        ; Setup all interrupt vectors in low
                        ; memory to address trap

254B 1E                 push ds         ;save the DS register
254C B80000             mov ax,0
254F 8ED8               mov ds,ax
2551 8EC0               mov es,ax       ;set ES and DS to zero
                        ;setup interrupt 0 to address trap routine
2553 C70600008D25       mov int0_offset,offset int_trap
2559 8C0E0200           mov int0_segment,CS
255D BF0400             mov di,4
2560 BE0000             mov si,0        ;then propagate
2563 B9FE01             mov cx,510      ;trap vector to
2566 F3A5               rep movs ax,ax  ;all 256 interrupts
                        ;BDOS offset to proper interrupt
2568 C7068003060B       mov bdos_offset,bdos_ofst
256E 1F                 pop ds          ;restore the DS register

                ;***********************************************
                ;*                                             *
                ;* National "BLC 8538" Channel 0 for a serial* 
                ;* 9600 baud printer - this board uses 8 Sig-* 
                ;* netics 2651 Usarts which have on-chip baud* 
                ;* rate generators.                            *
                ;*                                             *
                ;***********************************************

256F B0FF               mov al,0FFh
2571 E660               out blc_reset,al ;reset all usarts on 8538
2573 B04E               mov al,4Eh
2575 E642               out ldata+2,al  ;set usart 0 in async 8 bit
2577 B03E               mov al,3Eh
2579 E642               out ldata+2,al  ;set usart 0 to 9600 baud
257B B037               mov al,37h
257D E643               out ldata+3,al  ;enable Tx/Rx, and set up R
```

```
                        ;|                                          |
                        ;-------------------------------------------
                                ENDIF   ;not loader_bios

                                IF      loader_bios
                        ;-------------------------------------------
                        ;|                                          |
                                ;This is a BIOS for the LOADER
                                push ds         ;save data segment
                                mov ax,0
                                mov ds,ax       ;point to segment zero
                                ;BDOS interrupt offset
                                mov bdos_offset,bdos_ofst
                                mov bdos_segment,CS ;bdos interrupt segment
                                pop ds          ;restore data segment
                        ;|                                          |
                        ;-------------------------------------------
                                ENDIF   ;loader_bios

257F BB4427                     mov bx,offset signon
2582 E86600                     call pmsg       ;print signon message
2585 B100                       mov cl,0        ;default to dr A: on coldst
2587 E976DA                     jmp ccp         ;jump to cold start entry o

258A E979DA             WBOOT:  jmp ccp+6       ;direct entry to CCP at com

                                IF      not loader_bios
                        ;-------------------------------------------
                        ;|                                          |
                        int_trap:
258D FA                         cli             ;block interrupts
258E 8CC8                       mov ax,cs
2590 8ED8                       mov ds,ax       ;get our data segment
2592 BB7927                     mov bx,offset int_trp
2595 E85300                     call pmsg
2598 F4                         hlt             ;hardstop
                        ;|                                          |
                        ;-------------------------------------------
                                ENDIF   ;not loader_bios

                        ;*********************************************
                        ;*                                           *
                        ;*   CP/M Character I/O Interface Routines   *
                        ;*   Console is Usart (i8251a) on iSBC 86/12 *
                        ;*   at ports D8/DA                          *
                        ;*                                           *
                        ;*********************************************

                        CONST:          ;console status
259A E4DA                       in al,csts
259B 2402                       and al,2
259D 7402                       jz const_ret
259F 0CFF                       or al,255       ;return non-zero if RDA
                        const_ret:
25A1 C3                         ret             ;Receiver Data Available
```

```
                CONIN:                  ;console inout
25A2 E8F4FF             call const
25A5 74FB               jz CONIN        ;wait for RDA
25A7 E4D8               in al,cdata
25A9 247F               and al,7fh      ;read data and remove parit
25AB C3                 ret

                CONOUT:         ;console output
25AC E4DA               in al,csts
25AE 2401               and al,1        ;get console status
25B0 74FA               jz CONOUT       ;wait for TBE
25B2 8AC1               mov al,cl
25B4 E6D8               out cdata,al    ;Transmitter Buffer Empty
25B6 C3                 ret             ;then return data

                LISTOUT:                ;list device output

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
25B7 E80700             call LISTST
25BA 74FB               jz LISTOUT      ;wait for printer not busy
25BC 8AC1               mov al,cl
25BE E640               out ldata,al    ;send char to TI 810
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

25C0 C3                 ret

                LISTST:                 ;poll list status

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
25C1 E441               in al,lsts
25C3 2481               and al,81h      ;look at both TxRDY and DTR
25C5 3C81               cmp al,81h
25C7 750A               jnz zero_ret    ;either false, printer is b
25C9 0CFF               or al,255       ;both true, LPT is ready
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

25CB C3                 ret

                PUNCH:  ;not implemented in this configuration
                READER:
25CC B01A               mov al,1ah
25CE C3                 ret             ;return EOF for now

                GETIOBF:
25CF B000               mov al,0        ;TTY: for consistency
25D1 C3                 ret             ;IOBYTE not implemented
```

```
                SETIOBF:
25D2 C3                 ret             ;iobyte not implemented

                zero_ret:
25D3 2400               and al,0
25D5 C3         ret                     ;return zero in AL and flag

                ; Routine to get and echo a console character
                ;       and shift it to upper case

                uconecho:
25D6 E8C9FF             call CONIN      ;get a console character
25D9 50                 push ax
25DA 8AC8               mov cl,al       ;save and
25DC E8CDFF             call CONOUT
25DF 58                 pop ax          ;echo to console
25E0 3C61               cmp al,'a'
25E2 7206               jb uret         ;less than 'a' is ok
25E4 3C7A               cmp al,'z'
25E6 7702               ja uret         ;greater than 'z' is ok
25E8 2C20               sub al,'a'-'A'  ;else shift to caps
                uret:
25EA C3                 ret

                ;       utility subroutine to print messages

                pmsg:
25EB 8A07               mov al,[BX]     ;get next char from message
25ED 84C0               test al,al
25EF 7428               jz return       ;if zero return
25F1 8AC8               mov CL,AL
25F3 E8B6FF             call CONOUT     ;print it
25F6 43                 inc BX
25F7 EBF2               jmps pmsg       ;next character and loop

                ;***********************************************
                ;*                                             *
                ;*          Disk Input/Output Routines         *
                ;*                                             *
                ;***********************************************

                SELDSK:         ;select disk given by register CL
25F9 BB0000             mov bx,0000h
25FC 80F902             cmp cl,2        ;this BIOS only supports 2
25FF 7318               jnb return      ;return w/ 0000 in BX if ba
2601 B080               mov al, 80h
2603 80F900             cmp cl,0
2606 7502               jne sel1        ;drive 1 if not zero
2608 B040               mov al, 40h     ;else drive is 0
260A A26928     sel1:   mov sel_mask,al ;save drive select mask
                                        ;now, we need disk paramete
260D B500               mov ch,0
260F 8BD9               mov bx,cx       ;BX = word(CL)
2611 B104               mov cl,4
```

```
2613 D3E3               shl bx,cl       ;multiply drive code * 16
                        ;create offset from Disk Parameter Base
2615 81C37C28           add bx,offset dp_base
                return:
2619 C3                 ret

                HOME:   ;move selected disk to home position (Track
261A C6066C2800         mov trk,0       ;set disk i/o to track zero
261F BB6E28             mov bx,offset hom_com
2622 E83500             call execute
2625 74F2               jz return       ;home drive and return if O
2627 BB6A27             mov bx,offset bad_hom   ;else print
262A E8BEFF             call pmsg       ;"Home Error"
262D EBEB               jmps home       ;and retry

                SETTRK: ;set track address given by CX
262F 880E6C28           mov trk,cl      ;we only use 8 bits of trac
2633 C3                 ret

                SETSEC: ;set sector number given by cx
2634 880E6D28           mov sect,cl     ;we only use 8 bits of sect
2638 C3                 ret

                SECTRAN: ;translate sector CX using table at [DX]
2639 8BD9               mov bx,cx
263B 03DA               add bx,dx       ;add sector to tran table a
263D 8A1F               mov bl,[bx]     ;get logical sector
263F C3                 ret

                SETDMA: ;set DMA offset given by CX
2640 890E6528           mov dma_adr,CX
2644 C3                 ret

                SETDMAB: ;set DMA segment given by CX
2645 890E6728           mov dma_seg,CX
2649 C3                 ret
                ;
                GETSEGT:  ;return address of physical memory table
264A BB7328             mov bx,offset seg_table
264D C3                 ret

                ;***********************************************
                ;*                                             *
                ;*  All disk I/O parameters are setup:  the    *
                ;*  Read and Write entry points transfer one   *
                ;*  sector of 128 bytes to/from the current    *
                ;*  DMA address using the current disk drive   *
                ;*                                             *
                ;***********************************************

                READ:
264E B012               mov al,12h      ;basic read sector command
2650 EB02               jmps r_w_common

                WRITE:
```

```
2652 B00A               mov al,0ah      ;basic write sector command

                    r_w_common:
2654 BB6A28             mov bx,offset io_com ;point to command stri
2657 884701             mov byte ptr 1[BX],al ;put command into str
                    ;   fall into execute and return

                    execute:  ;execute command string.
                            ;[BX] points to length,
                            ;       followed by Command byte,
                            ;       followed by length-1 parameter byte

265A 891E6328           mov last_com,BX ;save command address for r
                    outer_retry:
                            ;allow some retrying
265E C60662280A         mov rtry_cnt,max_retries
                    retry:
2663 8B1E6328           mov BX,last_com
2667 E88900             call send_com   ;transmit command to i8271
                    ;   check status poll

266A 8B1E6328           mov BX,last_com
266E 8A4701             mov al,1[bx]    ;get command op code
2671 B90008             mov cx,0800h    ;mask if it will be "int re
2674 3C2C               cmp al,2ch
2676 720B               jb exec_poll    ;ok if it is an interrupt t
2678 B98080             mov cx,8080h    ;else we use "not command b
267B 240F               and al,0fh
267D 3C0C               cmp al,0ch      ;unless there isn't
267F B000               mov al,0
2681 7736               ja exec_exit    ;       any result
                                        ;poll for bits in CH,
                    exec_poll:          ;  toggled with bits in CL

2683 E4A0               in al,fdc_stat  ;read status
2685 22C5               and al,ch
2687 32C1               xor al,cl       ;  isolate what we want to
2689 74F8               jz exec_poll    ;and loop until it is done

                                        ;Operation complete,
268B E4A1               in al,fdc_rslt  ; see if result code indica
268D 241E               and al,1eh
268F 7428               jz exec_exit    ;no error, then exit
                                        ;some type of error occurre
2691 3C10               cmp al,10h
2693 7425               je dr_nrdy      ;was it a not ready drive ?
                                        ;no,
                    dr_rdy: ; then we just retry read or write
2695 FE0E6228           dec rtry_cnt
2699 75C8               jnz retry       ;  up to 10 times

                    ;   retries do not recover from the
                    ;   hard error

269B B400               mov ah,0
```

```
269D 8BD8               mov bx,ax       ;make error code 16 bits
269F 8B9F9127           mov bx,errtbl[BX]
26A3 E845FF             call pmsg       ;print appropriate message
26A6 E4D8               in al,cdata     ;flush usart receiver buffe
26A8 E82BFF             call uconecho   ;read upper case console ch
26AB 3C43               cmp al,'C'
26AD 7425               ie wboot_1      ;cancel
26AF 3C52               cmp al,'R'
26B1 74AB               ie outer_retry  ;retry 10 more times
26B3 3C49               cmp al,'I'
26B5 741A               ie z_ret        ;ignore error
26B7 0CFF               or al,255       ;set code for permanent err
                exec_exit:
26B9 C3                 ret

                dr_nrdy:        ;here to wait for drive ready
26BA E81A00             call test_ready
26BD 75A4               jnz retry       ;if it's ready now we are d
26BF E81500             call test_ready
26C2 759F               jnz retry       ;if not ready twice in row,
26C4 BB0228             mov bx,offset nrdymsg
26C7 E821FF             call pmsg ;"Drive Not Ready"
                nrdy01:
26CA E80A00             call test_ready
26CD 74FB               iz nrdy01       ;now loop until drive ready
26CF EB92               jmps retry      ;then go retry without decr
                zret:
26D1 2400               and al,0
26D3 C3                 ret             ;return with no error code

                wboot_1:                ;can't make it w/ a short 1
26D4 E9B3FE             jmp WBOOT

                ;***********************************************
                ;*                                             *
                ;*  The i8271 requires a read status command   *
                ;*  to reset a drive-not-ready after the       *
                ;*  drive becomes ready                        *
                ;*                                             *
                ;***********************************************

                test_ready:
26D7 B640               mov dh, 40h     ;proper mask if dr 1
26D9 F606692880         test sel_mask,80h
26DE 7502               jnz nrdy2
26E0 B604               mov dh, 04h     ;mask for dr 0 status bit
                nrdy2:
26E2 BB7128             mov bx,offset rds_com
26E5 E80B00             call send_com
                dr_poll1:
26E8 E4A0               in al,fdc_stat  ;get status word
26EA A880               test al,80h
26EC 75FA               jnz dr_poll     ;wait for not command busy
26EE E4A1               in al,fdc_rslt  ;get "special result"
26F0 84C6               test al,dh      ;look at bit for this drive
```

```
26F2 C3                 ret             ;return status of ready

                ;*********************************************
                ;*                                           *
                ;*  Send_com sends a command and parameters  *
                ;*  to the i8271:  BX addresses parameters.  *
                ;*  The DMA controller is also initialized   *
                ;*  if this is a read or write               *
                ;*                                           *
                ;*********************************************

                send_com:
  26F3 E4A0             in al,fdc_stat
  26F5 A880             test al,80h     ;insure command not busy
  26F7 75FA             jnz send_com    ;loop until ready

                        ;see if we have to initialize for a DMA ope

  26F9 8A4701           mov al,1[bx]    ;get command byte
  26FC 3C12             cmp al,12h
  26FE 7504             jne write_maybe ;if not a read it could be
  2700 B140             mov cl,40h
  2702 EB06             jmps init_dma   ;is a read command, go set
                write_maybe:
  2704 3C0A             cmp al,0ah
  2706 7520             jne dma_exit    ;leave DMA alone if not rea
  2708 B180             mov cl,80h      ;we have write, not read
                init_dma:
                ;we have a read or write operation, setup DMA contr
                ;       (CL contains proper direction bit)
  270A B004             mov al,04h
  270C E6A8             out dmac_mode,al    ;enable dmac
  270E B000             mov al,00
  2710 E6A5             out dmac_cont,al    ;send first byte to con
  2712 8AC1             mov al,cl
  2714 E6A5             out dmac_cont,al   ;load direction register
  2716 A16528           mov ax,dma_adr
  2719 E6A4             out dmac_adr,al    ;send low byte of DMA
  271B 8AC4             mov al,ah
  271D E6A4             out dmac_adr,al    ;send high byte
  271F A16728           mov ax,dma_seg
  2722 E6AA             out fdc_segment,al ;send low byte of segmen
  2724 8AC4             mov al,ah
  2726 E6AA             out fdc_segment,al ;then high segment addre
                dma_exit:
  2728 8A0F             mov cl,[BX]     ;get count
  272A 43               inc BX
  272B 8A07             mov al,[BX]     ;get command
  272D 0A066928         or al,sel_mask  ;merge command and drive co
  2731 E6A0             out fdc_com,al  ;send command byte
                parm_loop:
  2733 FEC9             dec cl
  2735 7482             jz exec_exit    ;no (more) parameters, retu
  2737 43               inc BX          ;point to (next) parameter
                parm_poll:
```

```
2738 E4A0               in al,fdc_stat
273A A820               test al,20h     ;test "parameter register f
273C 75FA               jnz parm_poll   ;idle until parm req not fu
273E 8A07               mov al,[BX]
2740 E6A1               out fdc_parm,al ;send next parameter
2742 EBEF               jmps parm_loop  ;go see if there are more p

                ;***********************************************
                ;*                                             *
                ;*               Data Areas                    *
                ;*                                             *
                ;***********************************************
  2744          data_offset     equ offset $

                        dseg
                        org     data_offset     ;contiguous with co

                        IF      loader_bios
                ;-----------------------------------------------
                ;|                                             |
                signon  db      cr,lf,cr,lf
                        db      'CP/M-86 Version 2.2',cr,lf,0
                ;|                                             |
                ;-----------------------------------------------
                        ENDIF   ;loader_bios

                        IF      not loader_bios
                ;-----------------------------------------------
                ;|                                             |
2744 0D0A0D0A   signon  db      cr,lf,cr,lf
2748 202053797374       db      '  System Generated  - 11 Jan 81',c
     656D2047656E
     657261746564
     20202D203131
     204A616E2038
     310D0A00
                ;|                                             |
                ;-----------------------------------------------
                        ENDIF   ;not loader_bios

276A 0D0A486F6D65 bad_hom db    cr,lf,'Home Error',cr,lf,0
     204572726F72
     0D0A00
2779 0D0A496E7465 int_trp db    cr,lf,'Interrupt Trap Halt',cr,lf,0
     727275707420
     547261702048
     616C740D0A00

2791 B127B127B127 errtbl  dw er0,er1,er2,er3
     B127
2799 C127D127DE27         dw er4,er5,er6,er7
     EF27
27A1 022816282828         dw er8,er9,erA,erB
     3D28
27A9 4D28B127B127         dw erC,erD,erE,erF
```

```
B127

27B1 0D0A4E756C6C er0     db  cr,lf,'Null Error ??',0
     204572726F72
     203F3F00
  27B1             er1     equ er0
  27B1             er2     equ er0
  27B1             er3     equ er0
27C1 0D0A436C6F63 er4     db  cr,lf,'Clock Error :',0
     6B204572726F
     72203A00
27D1 0D0A4C617465 er5     db  cr,lf,'Late DMA :',0
     20444D41203A
     00
27DE 0D0A49442043 er6     db  cr,lf,'ID CRC Error :',0
     524320457272
     6F72203A00
27EF 0D0A44617461 er7     db  cr,lf,'Data CRC Error :',0
     204352432045
     72726F72203A
     00
2802 0D0A44726976 er8     db  cr,lf,'Drive Not Ready :',0
     65204E6F7420
     526561647920
     3A00
2816 0D0A57726974 er9     db  cr,lf,'Write Protect :',0
     652050726F74
     656374203A00
2828 0D0A54726B20 erA     db  cr,lf,'Trk 00 Not Found :',0
     3030204E6F74
     20466F756E64
     203A00
283D 0D0A57726974 erB     db  cr,lf,'Write Fault :',0
     65204661756C
     74203A00
284D 0D0A53656374 erC     db  cr,lf,'Sector Not Found :',0
     6F72204E6F74
     20466F756E64
     203A00
  27B1             erD     equ er0
  27B1             erE     equ er0
  27B1             erF     equ er0
  2802             nrdymsg equ er8

2862 00          rtry_cnt db 0   ;disk error retry counter
2863 0000        last_com dw 0   ;address of last command string
2865 0000        dma_adr  dw 0   ;dma offset stored here
2867 0000        dma_seg  dw 0   ;dma segment stored here
2869 40          sel_mask db 40h ;select mask, 40h or 80h

                 ;       Various command strings for i8271

286A 03          io_com   db 3   ;length
286B 00          rd_wr    db 0   ;read/write function code
286C 00          trk      db 0   ;track #
```

```
286D 00          sect    db 0    ;sector #

286E 022900      hom_com db 2,29h,0      ;home drive command
2871 012C        rds_com db 1,2ch        ;read status command

                 ;       System Memory Segment Table

2873 02          segtable db 2   ;2 segments
2874 DF02                dw tpa_seg      ;1st seg starts after BIOS
2876 2105                dw tpa_len      ;and extends to 08000
2878 0020                dw 2000h        ;second is 20000 -
287A 0020                dw 2000h        ;3FFFF (128k)

=                        include singles.lib ;read in disk definitio
=                ;               DISKS 2
=  287C          dpbase  equ     $               ;Base of Disk Param
=287C AB280000   dpe0    dw      xlt0,0000h      ;Translate Table
=2880 00000000           dw      0000h,0000h     ;Scratch Area
=2884 C5289C28           dw      dirbuf,dpb0     ;Dir Buff, Parm Blo
=2888 64294529           dw      csv0,alv0       ;Check, Alloc Vecto
=288C AB280000   dpe1    dw      xlt1,0000h      ;Translate Table
=2890 00000000           dw      0000h,0000h     ;Scratch Area
=2894 C5289C28           dw      dirbuf,dpb1     ;Dir Buff, Parm Blo
=2898 93297429           dw      csv1,alv1       ;Check, Alloc Vecto
=                ;               DISKDEF 0,1,26,6,1024,243,64,64,2
=  289C          dpb0    equ     offset $        ;Disk Parameter Blo
=289C 1A00               dw      26              ;Sectors Per Track
=289E 03                 db      3               ;Block Shift
=289F 07                 db      7               ;Block Mask
=28A0 00                 db      0               ;Extnt Mask
=28A1 F200               dw      242             ;Disk Size - 1
=28A3 3F00               dw      63              ;Directory Max
=28A5 C0                 db      192             ;Alloc0
=28A6 00                 db      0               ;Alloc1
=28A7 1000               dw      16              ;Check Size
=28A9 0200               dw      2               ;Offset
=  28AB          xlt0    equ     offset $        ;Translate Table
=28AB 01070D13           db      1,7,13,19
=28AF 19050B11           db      25,5,11,17
=28B3 1703090F           db      23,3,9,15
=28B7 1502080E           db      21,2,8,14
=28BB 141A060C           db      20,26,6,12
=28BF 1218040A           db      18,24,4,10
=28C3 1016               db      16,22
=  001F          als0    equ     31              ;Allocation Vector
=  0010          css0    equ     16              ;Check Vector Size
=                ;               DISKDEF 1,0
=  289C          dpb1    equ     dpb0            ;Equivalent Paramet
=  001F          als1    equ     als0            ;Same Allocation Ve
=  0010          css1    equ     css0            ;Same Checksum Vect
=  28AB          xlt1    equ     xlt0            ;Same Translate Tab
=                ;               ENDEF
=                ;
=                ;       Uninitialized Scratch Memory Follows:
=  28C5          begdat  equ     offset $        ;Start of Scratch A
```

```
=28C5                   dirbuf  rs      128             ;Directory Buffer
=2945                   alv0    rs      als0            ;Alloc Vector
=2964                   csv0    rs      css0            ;Check Vector
=2974                   alv1    rs      als1            ;Alloc Vector
=2993                   csv1    rs      css1            ;Check Vector
=  29A3                 enddat  equ     offset $        ;End of Scratch Are
=  00DE                 datsiz  equ     offset $-begdat ;Size of Scratch Ar
=29A3 00                        db      0               ;Marks End of Modul


 29A4                   loc_stk rw  32  ;local stack for initialization
   29E4                 stkbase equ offset $

   29E4                 lastoff equ offset $
   02DF                 tpa_seg equ (lastoff+0400h+15) / 16
   0521                 tpa_len equ 0800h - tpa_seg
 29E4 00                        db 0    ;fill last address for GENCMD

                        ;***********************************************
                        ;*                                             *
                        ;*          Dummy Data Section                 *
                        ;*                                             *
                        ;***********************************************
   0000                         dseg    0       ;absolute low memory
                                org     0       ;(interrupt vectors)
 0000                   int0_offset     rw      1
 0002                   int0_segment    rw      1
                        ;       pad to system call vector
 0004                           rw      2*(bdos_int-1)

 0380                   bdos_offset     rw      1
 0382                   bdos_segment    rw      1
                                END
```

Appendix F
CBIOS Listing

```
*****************************************************
*                                                   *
* This is the listing of the skeletal CBIOS which   *
* you can use as the basis for a customized BIOS    *
* for non-standard hardware.  The essential por-    *
* tions of the BIOS remain, with "rs" statements    *
* marking the routines to be inserted.              *
*                                                   *
*****************************************************

                ;*********************************************
                ;*                                           *
                ;* This Customized BIOS adapts CP/M-86 to    *
                ;* the following hardware configuration      *
                ;*     Processor:                            *
                ;*     Brand:                                *
                ;*     Controller:                           *
                ;*                                           *
                ;*                                           *
                ;*     Programmer:                           *
                ;*     Revisions :                           *
                ;*                                           *
                ;*********************************************

FFFF            true            equ -1
0000            false           equ not true
000D            cr              equ 0dh ;carriage return
000A            lf              equ 0ah ;line feed

                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.                             *
                ;*                                           *
                ;*********************************************

0000            loader_bios     equ false
00E0            bdos_int        equ 224 ;reserved BDOS interrupt

                        IF      not loader_bios
                ;-------------------------------------------
                ;|                                         |
2500            bios_code       equ 2500h
0000            ccp_offset      equ 0000h
0B06            bdos_ofst       equ 0B06h ;BDOS entry point
                ;|                                         |
                ;-------------------------------------------
```

```
                ENDIF   ;not loader_bios

                IF      loader_bios
;-----------------------------------------------
;|                                             |
bios_code       equ 1200h ;start of LDBIOS
ccp_offset      equ 0003h ;base of CPMLOADER
bdos_ofst       equ 0406h ;stripped BDOS entry
;|                                             |
;-----------------------------------------------
                ENDIF   ;loader_bios

                cseg
                org     ccpoffset
ccp:
                org     bios_code

;***********************************************
;*                                             *
;* BIOS Jump Vector for Individual Routines    *
;*                                             *
;***********************************************

2500 E93C00      jmp INIT       ;Enter from BOOT ROM or LOADER
2503 E97900      jmp WBOOT      ;Arrive here from BDOS call 0
2506 E98500      jmp CONST      ;return console keyboard status
2509 E98D00      jmp CONIN      ;return console keyboard char
250C E99A00      jmp CONOUT     ;write char to console device
250F E9A200      jmp LISTOUT    ;write character to list device
2512 E9B500      jmp PUNCH      ;write character to punch device
2515 E9BD00      jmp READER     ;return char from reader device
2518 E9F600      jmp HOME       ;move to trk 00 on cur sel drive
251B E9D900      jmp SELDSK     ;select disk for next rd/write
251E E90101      jmp SETTRK     ;set track for next rd/write
2521 E90301      jmp SETSEC     ;set sector for next rd/write
2524 E90C01      jmp SETDMA     ;set offset for user buff (DMA)
2527 E91701      jmp READ       ;read a 128 byte sector
252A E94701      jmp WRITE      ;write a 128 byte sector
252D E98F00      jmp LISTST     ;return list status
2530 E9F900      jmp SECTRAN    ;xlate logical->physical sector
2533 E90201      jmp SETDMAB    ;set seg base for buff (DMA)
2536 E90401      jmp GETSEGT    ;return offset of Mem Desc Table
2539 E9A400      jmp GETIOBF    ;return I/O map byte (IOBYTE)
253C E9A500      jmp SETIOBF    ;set I/O map byte (IOBYTE)

;***********************************************
;*                                             *
;* INIT Entry Point, Differs for LDBIOS and    *
;* BIOS, according to "Loader_Bios" value      *
;*                                             *
;***********************************************

INIT:   ;print signon message and initialize hardwa
253F 8CC8       mov ax,cs       ;we entered with a JMPF so
```

```
2541 8ED0               mov ss,ax       ;CS: as the initial value o
2543 8ED8               mov ds,ax       ;DS:,
2545 8EC0               mov es,ax       ;and ES:
                        ;use local stack during initialization
2547 BC5928             mov sp,offset stkbase
254A FC                 cld             ;set forward direction

                        IF      not loader_bios
                ;-----------------------------------------------
                ;|                                             |
                        ; This is a BIOS for the CPM.SYS file.
                        ; Setup all interrupt vectors in low
                        ; memory to address trap

254B 1E                 push ds         ;save the DS register
254C C606A72600         mov IOBYTE,0    ;clear IOBYTE
2551 B80000             mov ax,0
2554 8ED8               mov ds,ax
2556 8EC0               mov es,ax       ;set ES and DS to zero
                        ;setup interrupt 0 to address trap routine
2558 C70600008225       mov int0_offset,offset int_trap
255E 8C0E0200           mov int0_segment,CS
2562 BF0400             mov di,4
2565 BE0000             mov si,0        ;then propagate
2568 B9FE01             mov cx,510      ;trap vector to
256B F3A5               rep movs ax,ax  ;all 256 interrupts
                        ;BDOS offset to proper interrupt
256D C70680030608       mov bdos_offset,bdos_ofst
2573 1F                 pop ds          ;restore the DS register

                ;       (additional CP/M-86 initialization)
                ;|                                            |
                ;-----------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;-----------------------------------------------
                ;|                                             |
                        ;This is a BIOS for the LOADER
                        push ds         ;save data segment
                        mov ax,0
                        mov ds,ax       ;point to segment zero
                        ;BDOS interrupt offset
                        mov bdos_offset,bdos_ofst
                        mov bdos_segment,CS ;bdos interrupt segment
                ;       (additional LOADER initialization)
                        pop ds          ;restore data segment
                ;|                                            |
                ;-----------------------------------------------
                        ENDIF   ;loader_bios

2574 BBB126             mov bx,offset signon
2577 E86F00             call pmsg       ;print signon message
257A B100               mov cl,0        ;default to dr A: on coldst
257C E981DA             jmp ccp         ;jump to cold start entry o
```

```
257F E984DA       WBOOT:  jmp ccp+6       ;direct entry to CCP at com

                          IF      not loader_bios
                  ;-----------------------------------------
                  ;|                                       |
                  int_trap:
2582 FA                   cli             ;block interrupts
2583 8CC8                 mov ax,cs
2585 8ED8                 mov ds,ax       ;get our data segment
2587 BBD126               mov bx,offset int_trp
258A E85C00               call pmsg
258D F4                   hlt             ;hardstop
                  ;|                                       |
                  ;-----------------------------------------
                          ENDIF   ;not loader_bios

                  ;*********************************************
                  ;*                                           *
                  ;*   CP/M Character I/O Interface Routines   *
                  ;*                                           *
                  ;*********************************************

                  CONST:            ;console status
258E                      rs      10      ;(fill-in)
2598 C3                   ret

                  CONIN:                  ;console input
2599 E8F2FF               call CONST
259C 74FB                 jz CONIN        ;wait for RDA
259E                      rs      10      ;(fill-in)
25A8 C3                   ret

                  CONOUT:           ;console output
25A9                      rs      10      ;(fill-in)
25B3 C3                   ret             ;then return data

                  LISTOUT:                ;list device output
25B4                      rs      10      ;(fill-in)
25BE C3                   ret

                  LISTST:                 ;poll list status
25BF                      rs      10      ;(fill-in)
25C9 C3                   ret

                  PUNCH:            ;write punch device
25CA                      rs      10      ;(fill-in)
25D4 C3                   ret

                  READER:
25D5                      rs      10      ;(fill-in)
25DF C3                   ret

                  GETIOBF:
25E0 A0A726               mov al,IOBYTE
```

```
25E3 C3                 ret

                SETIOBF:
25E4 880EA726           mov IOBYTE,cl   ;set iobyte
25E8 C3                 ret             ;iobyte not implemented

                pmsg:
25E9 8A07               mov al,[BX]     ;get next char from message
25EB 84C0               test al,al
25ED 7421               jz return       ;if zero return
25EF 8AC8               mov CL,AL
25F1 E8B5FF             call CONOUT     ;print it
25F4 43                 inc BX
25F5 EBF2               jmps pmsg       ;next character and loop

                ;***********************************************
                ;*                                             *
                ;*          Disk Input/Output Routines         *
                ;*                                             *
                ;***********************************************

                SELDSK:         ;select disk given by register CL
  0002          ndisks  equ     2 ;number of disks (up to 16)
25F7 880EA826           mov disk,cl     ;save disk number
25FB BB0000             mov bx,0000h    ;ready for error return
25FE 80F902             cmp cl,ndisks   ;n beyond max disks?
2601 730D               jnb return      ;return if so
2603 B500               mov ch,0        ;double(n)
2605 8BD9               mov bx,cx       ;bx = n
2607 B104               mov cl,4        ;ready for *16
2609 D3E3               shl bx,cl       ;n = n * 16
260B B9F126             mov cx,offset dpbase
260E 03D9               add bx,cx       ;dpbase + n * 16
2610 C3         return: ret             ;bx = .dph

                HOME:   ;move selected disk to home position (Track
2611 C706A9260000       mov trk,0       ;set disk i/o to track zero
2617                    rs      10      ;(fill-in)
2621 C3                 ret

                SETTRK: ;set track address given by CX
2622 890EA926           mov trk,CX
2626 C3                 ret

                SETSEC: ;set sector number given by cx
2627 890EAB26           mov sect,CX
262B C3                 ret

                SECTRAN: ;translate sector CX using table at [DX]
262C 8BD9               mov bx,cx
262E 03DA               add bx,dx       ;add sector to tran table a
2630 8A1F               mov bl,[bx]     ;get logical sector
2632 C3                 ret

                SETDMA: ;set DMA offset given by CX
```

```
2633 890EAD26           mov dma_adr,CX
2637 C3                 ret

                SETDMAB: ;set DMA segment given by CX
2638 890EAF26           mov dma_seg,CX
263C C3                 ret
                ;
                GETSEGT:  ;return address of physical memory table
263D BBE826             mov bx,offset seg_table
2640 C3                 ret

                ;***********************************************
                ;*                                             *
                ;*  All disk I/O parameters are setup:         *
                ;*     DISK     is disk number      (SELDSK)  *
                ;*     TRK      is track number     (SETTRK)  *
                ;*     SECT     is sector number    (SETSEC)  *
                ;*     DMA_ADR  is the DMA offset   (SETDMA)  *
                ;*     DMA_SEG  is the DMA segment  (SETDMAB) *
                ;*  READ reads the selected sector to the DMA *
                ;*  address, and WRITE writes the data from   *
                ;*  the DMA address to the selected sector    *
                ;*  (return 00 if successful,  01 if perm err)*
                ;*                                             *
                ;***********************************************

                READ:
2641                    rs      50      ;fill-in
2673 C3                 ret

                WRITE:
2674                    rs      50      ;(fill-in)
26A6 C3                 ret


                ;***********************************************
                ;*                                             *
                ;*                Data Areas                   *
                ;*                                             *
                ;***********************************************
  26A7          data_offset     equ offset $

                        dseg
                        org     data_offset     ;contiguous with co
26A7 00         IOBYTE  db      0
26A8 00         disk    db      0       ;disk number
26A9 0000       trk     dw      0       ;track number
26AB 0000       sect    dw      0       ;sector number
26AD 0000       dma_adr dw      0       ;DMA offset from DS
26AF 0000       dma_seg dw      0       ;DMA Base Segment

                        IF      loader_bios
                ;-----------------------------------------------
                ;|                                             |
                signon  db      cr,lf,cr,lf
```

```
                        db      'CP/M-86 Version 1.0',cr,lf,0
                    ;|                                      |
                    ;----------------------------------------
                        ENDIF   ;loader_bios

                        IF      not loader_bios
                    ;----------------------------------------
                    ;|                                      |
26B1 0D0A0D0A       signon  db      cr,lf,cr,lf
26B5 53797374656D           db      'System Generated 00/00/00'
     2047656E6572
     617465642030
     302F30302F30
     30
26CE 0D0A00                 db      cr,lf,0
                    ;|                                      |
                    ;----------------------------------------
                        ENDIF   ;not loader_bios

26D1 0D0A           int_trp db      cr,lf
26D3 496E74657272           db      'Interrupt Trap Halt'
     757074205472
     61702048616C
     74
26E6 0D0A                   db      cr,lf

                    ;       System Memory Segment Table

26E8 02             segtable db 2   ;2 segments
26E9 C602                   dw tpa_seg      ;1st seg starts after BIOS
26EB 3A05                   dw tpa_len      ;and extends to 08000
26ED 0020                   dw 2000h        ;second is 20000 -
26EF 0020                   dw 2000h        ;3FFFF (128k)

=                           include singles.lib ;read in disk definitio
=                   ;                       DISKS 2
=  26F1             dpbase  equ     $               ;Base of Disk Param
=26F1 20270000      dpe0    dw      xlt0,0000h      ;Translate Table
=26F5 00000000              dw      0000h,0000h     ;Scratch Area
=26F9 3A271127              dw      dirbuf,dpb0     ;Dir Buff, Parm Blo
=26FD D927BA27              dw      csv0,alv0       ;Check, Alloc Vecto
=2701 20270000      dpe1    dw      xlt1,0000h      ;Translate Table
=2705 00000000              dw      0000h,0000h     ;Scratch Area
=2709 3A271127              dw      dirbuf,dpb1     ;Dir Buff, Parm Blo
=270D 0828E927              dw      csv1,alv1       ;Check, Alloc Vecto
=                   ;                       DISKDEF 0,1,26,6,1024,243,64,64,2
=  2711             dpb0    equ     offset $        ;Disk Parameter Blo
=2711 1A00                  dw      26              ;Sectors Per Track
=2713 03                    db      3               ;Block Shift
=2714 07                    db      7               ;Block Mask
=2715 00                    db      0               ;Extnt Mask
=2716 F200                  dw      242             ;Disk Size - 1
=2718 3F00                  dw      63              ;Directory Max
=271A C0                    db      192             ;Alloc0
=271B 00                    db      0               ;Alloc1
```

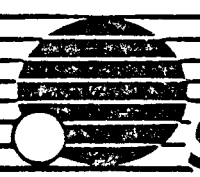
```
=271C 1000                      dw      16              ;Check Size
=271E 0200                      dw      2               ;Offset
=  2720                 xlt0    equ     offset $        ;Translate Table
=2720 01070D13                  db      1,7,13,19
=2724 19050B11                  db      25,5,11,17
=2728 1703090F                  db      23,3,9,15
=272C 1502080E                  db      21,2,8,14
=2730 141A060C                  db      20,26,6,12
=2734 1218040A                  db      18,24,4,10
=2738 1016                      db      16,22
=  001F                 als0    equ     31              ;Allocation Vector
=  0010                 css0    equ     16              ;Check Vector Size
=                       ;               DISKDEF 1,0
=  2711                 dpb1    equ     dpb0            ;Equivalent Paramet
=  001F                 als1    equ     als0            ;Same Allocation Ve
=  0010                 css1    equ     css0            ;Same Checksum Vect
=  2720                 xlt1    equ     xlt0            ;Same Translate Tab
=                       ;               ENDEF
=                       ;
=                       ;       Uninitialized Scratch Memory Follows:
=                       ;
=  273A                 begdat  equ     offset $        ;Start of Scratch A
=273A                   dirbuf  rs      128             ;Directory Buffer
=27BA                   alv0    rs      als0            ;Alloc Vector
=27D9                   csv0    rs      css0            ;Check Vector
=27E9                   alv1    rs      als1            ;Alloc Vector
=2808                   csv1    rs      css1            ;Check Vector
=  2818                 enddat  equ     offset $        ;End of Scratch Are
=  00DE                 datsiz  equ     offset $-begdat ;Size of Scratch Ar
=2818 00                        db      0               ;Marks End of Modul


 2819                   loc_stk rw  32  ;local stack for initialization
   2859                 stkbase equ offset $

   2859                 lastoff equ offset $
   02C6                 tpa_seg equ (lastoff+0400h+15) / 16
   053A                 tpa_len equ 0800h - tpa_seg
 2859 00                        db 0    ;fill last address for GENCMD

                        ;***********************************************
                        ;*                                             *
                        ;*          Dummy Data Section                 *
                        ;*                                             *
                        ;***********************************************
   0000                         dseg    0       ;absolute low memory
                                org     0       ;(interrupt vectors)
 0000                   int0_offset     rw      1
 0002                   int0_segment    rw      1
                        ;       pad to system call vector
 0004                           rw      2*(bdos_int-1)

 0380                   bdos_offset     rw      1
 0382                   bdos_segment    rw      1
                                END
```

Index

Index

SIRIUS 1

CP/M-86™
Programmer's Guide



CP/M-86™
Programmer's Guide

Digital Research
P.O. Box 579
801 Lighthouse Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

COPYRIGHT

DISCLAIMER

TRADEMARKS

The "CP/M-86 Programmer's Guide" was prepared using the Digital Research TEX-80™ text formatter and printed in the United States of America by Commercial Press/Monterey.

****
* Second Printing: June 1981 *
****

Table of Contents

Table of Contents
(continued)

Table of Contents
(continued)

Table of Contents
(continued)

Foreword

This manual assists the 8086 assembly language programmer working in a CP/M-86™ environment. It assumes you are familiar with the CP/M-86 implementation of CP/M and have read the following Digital Research publications:

- CP/M 2 Documentation
- CP/M-86 System Guide

The reader should also be familiar with the 8086 assembly language instruction set, which is defined in Intel's 8086 Family User's Manual.

The first section of this manual discusses ASM-86 operation and the various assembler options which may be enabled when invoking ASM-86™. One of these options controls the hexadecimal output format. ASM-86 can generate 8086 machine code in either Intel or Digital Research format. These two hexadecimal formats are described in Appendix A.

The second section discusses the elements of ASM-86 assembly language. It defines ASM-86's character set, constants, variables, identifiers, operators, expressions, and statements.

The third section discusses the ASM-86 directives, which perform housekeeping functions such as requesting conditional assembly, including multiple source files, and controlling the format of the listing printout.

The fourth section is a concise summary of the 8086 instruction mnemonics accepted by ASM-86. The mnemonics used by the Digital Research assembler are the same as those used by the Intel assembler except for four instructions: the intra-segment short jump, and inter-segment jump, return and call instructions. These differences are summarized in Appendix B.

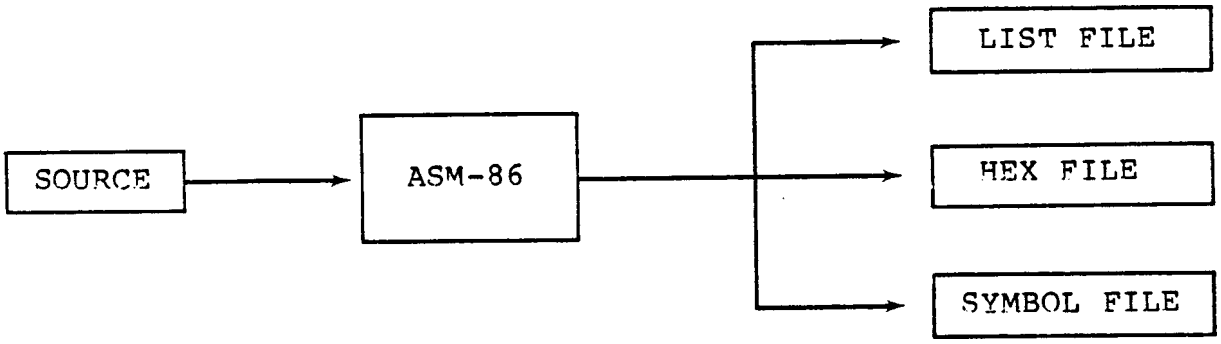
The fifth section of this manual discusses the code-macro facilities of ASM-86. Code-macro definition, specifiers and modifiers as well as nine special code-macro directives are discussed. This information is also summarized in Appendix H.

The sixth section discusses the DDT-86 program, which allows the user to test and debug programs interactively in the CP/M-86 enviornment. Section 6 includes a DDT-86 sample debugging session.

Section 1
Introduction

1.1 Assembler Operation

ASM-86 processes an 8086 assembly language source file in three passes and produces three output files, including an 8086 machine language file in hexadecimal format. This object file may be in either Intel or Digital Research hex format, which are described in Appendix C. ASM-86 is shipped in two forms: an 8086 cross-assembler designed to run under CP/M on an Intel 8080 or Zilog Z-80 based system, and a 8086 assembler designed to run under CP/M-86 on an Intel 8086 or 8088 based system. ASM-86 typically produces three output files from one input file as shown in Figure 1-1, below.



<file name>.A86 - contains source
<file name>.LST - contains listing
<file name>.H86 - contains assembled program in hexadecimal format
<file name>.SYM - contains all user-defined symbols

Figure 1-1. ASM-86 Source and Object Files

Figure 1-1 also lists ASM-86 filename extensions. ASM-86 accepts a source file with any three letter extension, but if the extension is omitted from the invoking command, it looks for the specified filename with the extension .A86 in the directory. If no filename is specified and the file has an extension other than .A86 or has no extension at all, ASM-86 returns an error message.

The other extensions listed in Figure 1-1 identify ASM-86 output files. The .LST file contains the assembly language listing with any error messages. The .H86 file contains the machine language program in either Digital Research or Intel hexadecimal format. The .SYM file lists any user-defined symbols.

Invoke ASM-86 by entering a command of the following form:

ASM86 <source filename> [$ <optional parameters>]

Section 1.2 explains the optional parameters. Specify the source file in the following form:

[<optional drive>:]<filename>[.<optional extension>]

where

<optional drive>	is a valid drive letter specifying the source file's location. Not needed if source is on current drive.
<filename>	is a valid CP/M filename of 1 to 8 characters.
<optional extension>	is a valid file extension of 1 to 3 characters, usually .A86.

Some examples of valid ASM-86 commands are:

A>ASM86 B:BIOS88

A>ASM86 BIOS88.ASM $FI AA HB PB SB

A>ASM86 D:TEST

Once invoked, ASM-86 responds with the message:

CP/M 8086 ASSEMBLER VER x.x

where x.x is the ASM-86 version number. ASM-86 then attempts to open the source file. If the file does not exist on the designated drive, or does not have the correct extension as described above, the assembler displays the message:

NO FILE

If an invalid parameter is given in the optional parameter list, ASM-86 displays the message:

PARAMETER ERROR

After opening the source, the assembler creates the output files. Usually these are placed on the current disk drive, but they may be redirected by optional parameters, or by a drive specification in the the source file name. In the latter case, ASM-86 directs the output files to the drive specified in the source file name.

During assembly, ASM-86 aborts if an error condition such as disk full or symbol table overflow is detected. When ASM-86 detects an error in the source file, it places an error message line in the listing file in front of the line containing the error. Each error message has a number and gives a brief explanation of the error. Appendix H lists ASM-86 error messages. When the assembly is complete, ASM-86 displays the message:

```
END OF ASSEMBLY. NUMBER OF ERRORS: n
```

1.2 Optional Run-time Parameters

The dollar-sign character, $, flags an optional string of run-time parameters. A parameter is a single letter followed by a single letter device name specification. The parameters are shown in Table 1-1, below.

Table 1-1. Run-time Parameter Summary

Parameter	To Specify	Valid Arguments
A	source file device	A, B, C, ... P
H	hex output file device	A ... P, X, Y, Z
P	list file device	A ... P, X, Y, Z
S	symbol file device	A ... P, X, Y, Z
F	format of hex output file	I, D

All parameters are optional, and can be entered in the command line in any order. Enter the dollar sign only once at the beginning of the parameter string. Spaces may separate parameters, but are not required. No space is permitted, however, between a parameter and its device name.

A device name must follow parameters A, H, P and S. The devices are labeled:

```
A, B, C, ... P  or  X, Y, Z
```

Device names A through P respectively specify disk drives A through P. X specifies the user console (CON:), Y specifies the line printer (LST:), and Z suppresses output (NUL:).

If output is directed to the console, it may be temporarily stopped at any time by typing a control-S. Restart the output by typing a second control-S or any other character.

The F parameter requires either an I or a D argument. When I is specified, ASM-86 produces an object file in Intel hex format. A D argument requests Digital Research hex format. Appendix C discusses these formats in detail. If the F parameter is not entered in the command line, ASM-86 produces Digital Research hex format.

Table 1-2. Run-time Parameter Examples

Command Line	Result
ASM86 IO	Assemble file IO.A86, produce IO.HEX, IO.LST and IO.SYM, all on the default drive.
ASM86 IO.ASM $ AD SZ	Assemble file IO.ASM on device D, produce IO.LST and IO.HEX on the default device, suppress symbol file.
ASM86 IO $ PY SX	Assemble file IO.A86, produce IO.HEX, route listing directly to printer, output symbols on console.
ASM86 IO $ FD	Produce Digital Research hex format.
ASM86 IO $ FI	Produce Intel hex format.

1.3 Aborting ASM-86

You may abort ASM-86 execution at any time by hitting any key on the console keyboard. When a key is pressed, ASM-86 responds with the question:

```
USER BREAK. OK(Y/N)?
```

A Y response aborts the assembly and returns to the operating system. An N response continues the assembly.

Section 2
Elements of ASM-86 Assembly Language

2.1 ASM-86 Character Set

ASM-86 recognizes a subset of the ASCII character set. The valid characters are the alphanumerics, special characters, and non-printing characters shown below:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9

+ - * / = ( ) [ ] ; ' . ! , _ : @ $

space, tab, carriage-return, and line-feed
```

Lower-case letters are treated as upper-case except within strings. Only alphanumerics, special characters, and spaces may appear within a string.

2.2 Tokens and Separators

A token is the smallest meaningful unit of an ASM-86 source program, much as a word is the smallest meaningful unit of an English composition. Adjacent tokens are commonly separated by a blank character or space. Any sequence of spaces may appear wherever a single space is allowed. ASM-86 recognizes horizontal tabs as separators and interprets them as spaces. Tabs are expanded to spaces in the list file. The tab stops are at each eighth column.

2.3 Delimiters

Delimiters mark the end of a token and add special meaning to the instruction, as opposed to separators, which merely mark the end of a token. When a delimiter is present, separators need not be used. However, separators after delimiters can make your program easier to read.

Table 2-1 describes ASM-86 separators and delimiters. Some delimiters are also operators and are explained in greater detail in Section 2.6.

Table 2-1. Separators and Delimiters

Character	Name	Use
20H	space	separator
09H	tab	separator, legal in source files, expanded in list files
CR	carriage return	terminate source lines
LF	line feed	legal after CR; if within source lines, it is interpreted as a space
;	semicolon	start comment field
:	colon	identifies a label, used in segment override specification
.	period	forms variables from numbers
$	dollar sign	notation for "present value of location pointer"
+	plus	arithmetic operator for addition
-	minus	arithmetic operator for subtraction
*	asterisk	arithmetic operator for multiplication
/	slash	arithmetic operator for division
@	at-sign	legal in identifiers
_	underscore	legal but ignored in identifiers
!	exclamation point	logically terminates a statement, thus allowing multiple statements on a single source line
'	apostrophe	delimits string constants

2.4 Constants

A constant is a value known at assembly time that does not change while the assembled program is executed. A constant may be either an integer or a character string.

2.4.1 Numeric Constants

A numeric constant is a 16-bit value in one of several bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. The radix indicators are shown in Table 2-2, below.

Table 2-2. Radix Indicators for Constants

Indicator	Constant Type	Base
B	binary	2
O	octal	8
Q	octal	8
D	decimal	10
H	hexadecimal	16

ASM-86 assumes that any numeric constant not terminated with a radix indicator is a decimal constant. Radix indicators may be upper or lower case.

A constant is thus a sequence of digits followed by an optional radix indicator, where the digits are in the range for the radix. Binary constants must be composed of 0's and 1's. Octal digits range from 0 to 7; decimal digits range from 0 to 9. Hexadecimal constants contain decimal digits as well as the hexadecimal digits A (10D), B (11D), C (12D), D (13D), E (14D), and F (15D). Note that the leading character of a hexadecimal constant must be either a leading 0 or a decimal digit so that ASM-86 cannot confuse a hex constant with an identifier. The following are valid numeric constants:

```
1234     1234D    1100B    1111000011110000B
1234H    0FFEH    3377O    13772Q
33770    0FE3H    1234d    0ffffh
```

2.4.2 Character Strings

ASM-86 treats an ASCII character string delimited by apostrophes as a string constant. All instructions accept only one- or two-character string constants as valid arguments. Instructions treat a one-character string as an 8-bit number. A two-character string is treated as a 16-bit number with the value of the second character in the low-order byte, and the value of the first character in the high-order byte.

The numeric value of a character is its ASCII code. ASM-86 does not translate case within character strings, so both upper- and lower-case letters can be used. Note that only alphanumerics, special characters, and spaces are allowed within strings.

A DB assembler directive is the only ASM-86 statement that may contain strings longer than two characters. The string may not exceed 255 bytes. Include any apostrophe to be printed within the string by entering it twice. ASM-86 interprets the two keystrokes '' as a single apostrophe. Table 2-3 shows valid strings and how they appear after processing:

Table 2-3. String Constant Examples

```
                'a'  -> a
           'Ab''Cd'  -> Ab'Cd
      'I like CP/M'  -> I like CP/M
              ''''   -> '
'ONLY UPPER CASE'    -> ONLY UPPER CASE
'only lower case'    -> only lower case
```

2.5 Identifiers

Identifiers are character sequences which have a special, symbolic meaning to the assembler. All identifiers in ASM-86 must obey the following rules:

1. The first character must be alphabetic (A,...Z, a,...z).

2. Any subsequent characters can be either alphabetical or a numeral (0,1,.....9). ASM-86 ignores the special characters @ and _, but they are still legal. For example, a_b becomes ab.

3. Identifiers may be of any length up to the limit of the physical line.

Identifiers are of two types. The first are keywords, which have predefined meanings to the assembler. The second are symbols, which are defined by the user. The following are all valid identifiers:

```
NOLIST
WORD
AH
Third_street
How_are_you_today
variable@number@1234567890
```

2.5.1 Keywords

A keyword is an identifier that has a predefined meaning to the assembler. Keywords are reserved; the user cannot define an identifier identical to a keyword. For a complete list of keywords, see Appendix D.

ASM-86 recognizes five types of keywords: instructions, directives, operators, registers and predefined numbers. 8086 instruction mnemonic keywords and the actions they initiate are defined in Section 4. Directives are discussed in Section 3. Section 2.6 defines operators. Table 2-4 lists the ASM-86 keywords that identify 8086 registers.

Three keywords are predefined numbers: BYTE, WORD, and DWORD. The values of these numbers are 1, 2 and 4, respectively. In addition, a Type attribute is associated with each of these numbers. The keyword's Type attribute is equal to the keyword's numeric value. See Section 2.5.2 for a complete discussion of Type attributes.

Table 2-4. Register Keywords

Register Symbol	Size	Numeric Value	Meaning
AH	1 byte	100 B	Accumulator-High-Byte
BH	1 "	111 B	Base-Register-High-Byte
CH	1 "	101 B	Count-Register-High-Byte
DH	1 "	110 B	Data-Register-High-Byte
AL	1 "	000 B	Accumulator-Low-Byte
BL	1 "	011 B	Base-Register-Low-Byte
CL	1 "	001 B	Count-Register-Low-Byte
DL	1 "	010 B	Data-Register-Low-Byte
AX	2 bytes	000 B	Accumulator (full word)
BX	2 "	011 B	Base-Register "
CX	2 "	001 B	Count-Register "
DX	2 "	010 B	Data-Register "
BP	2 "	101 B	Base Pointer
SP	2 "	100 B	Stack Pointer
SI	2 "	110 B	Source Index
DI	2 "	111 B	Destination Index
CS	2 "	01 B	Code-Segment-Register
DS	2 "	11 B	Data-Segment-Register
SS	2 "	10 B	Stack-Segment-Register
ES	2 "	00 B	Extra-Segment-Register

2.5.2 Symbols and Their Attributes

A symbol is a user-defined identifier that has attributes which specify what kind of information the symbol represents. Symbols fall into three categories:

- variables
- labels
- numbers

Variables identify data stored at a particular location in memory. All variables have the following three attributes:

- Segment - tells which segment was being assembled when the variable was defined.
- Offset - tells how many bytes there are between the beginning of the segment and the location of this variable.
- Type - tells how many bytes of data are manipulated when this variable is referenced.

A Segment may be a code-segment, a data-segment, a stack-segment or an extra-segment depending on its contents and the register that contains its starting address (see Section 3.2). A segment may start at any address divisible by 16. ASM-86 uses this boundary value as the Segment portion of the variable's definition.

The Offset of a variable may be any number between 0 and 0FFFFH or 65535D. A variable must have one of the following Type attributes:

- BYTE
- WORD
- DWORD

BYTE specifies a one-byte variable, WORD a two-byte variable and DWORD a four-byte variable. The DB, DW, and DD directives respectively define variables as these three types (see Section 3). For example, a variable is defined when it appears as the name for a storage directive:

```
VARIABLE  DB 0
```

A variable may also be defined as the name for an EQU directive referencing another label, as shown below:

```
VARIABLE  EQU  ANOTHER_VARIABLE
```

Labels identify locations in memory that contain instruction statements. They are referenced with jumps or calls. All labels have two attributes:

- Segment
- Offset

Label segment and offset attributes are essentially the same as variable segment and offset attributes. Generally, a label is defined when it precedes an instruction. A colon, :, separates the label from instruction; for example:

```
LABEL:   ADD  AX,BX
```

A label may also appear as the name for an EQU directive referencing another label; for example:

```
LABEL   EQU   ANOTHER_LABEL
```

Numbers may also be defined as symbols. A number symbol is treated as if you had explicitly coded the number it represents. For example:

```
Number_five     EQU   5
MOV   AL,Number_five
```

is equivalent to:

```
MOV   AL,5
```

Section 2.6 describes operators and their effects on numbers and number symbols.

2.6 Operators

ASM-86 operators fall into the following categories: arithmetic, logical, and relational operators, segment override, variable manipulators and creators. Table 2-5 defines ASM-86 operators. In this table, a and b represent two elements of the expression. The validity column defines the type of operands the operator can manipulate, using the or bar character, |, to separate alternatives.

Table 2-5. ASM-86 Operators

Syntax	Result	Validity
	Logical Operators	
a XOR b	bit-by-bit logical EXCLUSIVE OR of a and b.	a, b = number
a OR b	bit-by-bit logical OR of a and b.	a, b = number
a AND b	bit-by-bit logical AND of a and b.	a, b = number
NOT a	logical inverse of a: all 0's become 1's, 1's become 0's.	a = 16-bit number

Table 2-5. (continued)

Syntax	Result	Validity
Relational Operators		
a EQ b	returns 0FFFFH if a = b, otherwise 0.	a, b = unsigned number
a LT b	returns 0FFFFH if a < b, otherwise 0.	a, b = unsigned number
a LE b	returns 0FFFFH if a <= b, otherwise 0.	a, b = unsigned number
a GT b	returns 0FFFFH if a > b, otherwise 0.	a, b = unsigned number
a GE b	returns 0FFFFH if a >= b otherwise 0.	a, b = unsigned number
a NE b	returns 0FFFFH if a <> b, otherwise 0.	a, b = unsigned number
Arithmetic Operators		
a + b	arithmetic sum of a and b.	a = variable, label or number b = number
a - b	arithmetic difference of a and b.	a = variable, label or number b = number
a * b	does unsigned multiplication of a and b.	a, b = number
a / b	does unsigned division of a and b.	a, b = number
a MOD b	returns remainder of a / b.	a, b = number
a SHL b	returns the value which results from shifting a to left by an amount b.	a, b = number
a SHR b	returns the value which results from shifting a to the right by an amount b.	a, b = number
+ a	gives a.	a = number
- a	gives 0 - a.	a = number

Table 2-5. (continued)

Syntax	Result	Validity
Segment Override		
<seg reg>: <addr exp>	overrides assembler's choice of segment register.	<seg reg> = CS, DS, SS or ES
Variable Manipulators, Creators		
SEG a	creates a number whose value is the segment value of the variable or label a.	a = label | variable
OFFSET a	creates a number whose value is the offset value of the variable or label a.	a = label | variable
TYPE a	creates a number. If the variable a is of type BYTE, WORD or DWORD, the value of the number will be 1, 2 or 4, respectively.	a = label | variable
LENGTH a	creates a number whose value is the LENGTH attribute of the variable a. The length attribute is the number of bytes associated with the variable.	a = label | variable
LAST a	if LENGTH a > 0, then LAST a = LENGTH a - 1; if LENGTH a = 0, then LAST a = 0.	a = label | variable
a PTR b	creates virtual variable or label with type of a and attributes of b	a = BYTE | WORD, | DWORD b = <addr exp>
.a	creates variable with an offset attribute of a. Segment attribute is current segment.	a = number
$	creates label with offset equal to current value of location counter; segment attribute is current segment.	no argument

2.6.1 Operator Examples

Logical operators accept only numbers as operands. They perform the boolean logic operations AND, OR, XOR, and NOT. For example:

```
  00FC              MASK    EQU     0FCH
  0080              SIGNBIT EQU     80H
0000 B180                   MOV     CL,MASK AND SIGNBIT
0002 B003                   MOV     AL,NOT MASK
```

Relational operators treat all operands as unsigned numbers. The relational operators are EQ (equal), LT (less than), LE (less than or equal), GT (greater than), GE (greater than or equal), and NE (not equal). Each operator compares two operands and returns all ones (0FFFFH) if the specified relation is true and all zeros if it is not. For example:

```
  000A              LIMIT1  EQU     10
  0019              LIMIT2  EQU     25
                            .
                            .
                            .
0004 B8FFFF                 MOV     AX,LIMIT1 LT LIMIT2
0007 B80000                 MOV     AX,LIMIT1 GT LIMIT2
```

Addition and subtraction operators compute the arithmetic sum and difference of two operands. The first operand may be a variable, label, or number, but the second operand must be a number. When a number is added to a variable or label, the result is a variable or label whose offset is the numeric value of the second operand plus the offset of the first operand. Subtraction from a variable or label returns a variable or label whose offset is that of first operand decremented by the number specified in the second operand. For example:

```
  0002              COUNT   EQU     2
  0005              DISP1   EQU     5
000A FF             FLAG    DB      0FFH
                            .
                            .
                            .
000B 2EA00B00               MOV     AL,FLAG+1
000F 2E8A0E0F00             MOV     CL,FLAG+DISP1
0014 B303                   MOV     BL,DISP1-COUNT
```

The multiplication and division operators *, /, MOD, SHL, and SHR accept only numbers as operands. * and / treat all operators as unsigned numbers. For example:

```
0016 BE5500                 MOV     SI,256/3
0019 B310                   MOV     BL,64/4
  0050          BUFFERSIZE  EQU     80
001B B8A000                 MOV     AX,BUFFERSIZE * 2
```

Unary operators accept both signed and unsigned operators as shown below:

```
001E B123                MOV     CL,+35
0020 B007                MOV     AL,2--5
0022 B2F4                MOV     DL,-12
```

When manipulating variables, the assembler decides which segment register to use. You may override the assembler's choice by specifying a different register with the segment override operator. The syntax for the override operator is <segment register> : <address expression> where the <segment register> is CS, DS, SS, or ES. For example:

```
0024 368B472D            MOV     AX,SS:WORDBUFFER[BX]
0028 268B0E5B00          MOV     CX,ES:ARRAY
```

A variable manipulator creates a number equal to one attribute of its variable operand. SEG extracts the variable's segment value, OFFSET its offset value, TYPE its type value (1, 2, or 4), and LENGTH the number of bytes associated with the variable. LAST compares the variable's LENGTH with 0 and if greater, then decrements LENGTH by one. If LENGTH equals 0, LAST leaves it unchanged. Variable manipulators accept only variables as operators. For example:

```
002D 000000000000 WORDBUFFER     DW      0,0,0
0033 0102030405   BUFFER         DB      1,2,3,4,5
                                 .
                                 .
                                 .
0038 B80500              MOV     AX,LENGTH BUFFER
003B B80400              MOV     AX,LAST BUFFER
003E B80100              MOV     AX,TYPE BUFFER
0041 B80200              MOV     AX,TYPE WORDBUFFER
```

The PTR operator creates a virtual variable or label, one valid only during the execution of the instruction. It makes no changes to either of its operands. The temporary symbol has the same Type attribute as the left operator, and all other attributes of the right operator as shown below.

```
0044 C60705              MOV     BYTE PTR [BX], 5
0047 8A07                MOV     AL,BYTE PTR [BX]
0049 FF04                INC     WORD PTR [SI]
```

The Period operator, ., creates a variable in the current data segment. The new variable has a segment attribute equal to the current data segment and an offset attribute equal to its operand. Its operand must be a number. For example:

```
004B A10000              MOV     AX, .0
004E 268B1E0040          MOV     BX, ES: .4000H
```

The Dollar-sign operator, $, creates a label with an offset attribute equal to the current value of the location counter. The label's segment value is the same as the current code segment. This operator takes no operand. For example:

```
0053 E9FDFF          JMP     $
0056 EBFE            JMPS    $
0058 E9FD2F          JMP     $+3000H
```

2.6.2 Operator Precedence

Expressions combine variables, labels or numbers with operators. ASM-86 allows several kinds of expressions which are discussed in Section 2.7. This section defines the order in which operations are executed should more than one operator appear in an expression.

In general, ASM-86 evaluates expressions left to right, but operators with higher precedence are evaluated before operators with lower precedence. When two operators have equal precedence, the left-most is evaluated first. Table 2-6 presents ASM-86 operators in order of increasing precedence.

Parentheses can override normal rules of precedence. The part of an expression enclosed in parentheses is evaluated first. If parentheses are nested, the innermost expressions are evaluated first. Only five levels of nested parentheses are legal. For example:

```
15/3 + 18/9 = 5 + 2 = 7
15/(3 + 18/9) = 15/(3 + 2) = 15/5 = 3
```

Table 2-6. Precedence of Operations in ASM-86

Order	Operator Type	Operators
1	Logical	XOR, OR
2	Logical	AND
3	Logical	NOT
4	Relational	EO, LT, LE, GT, GE, NE
5	Addition/subtraction	+, -
6	Multiplication/division	*, /, MOD, SHL, SHR
7	Unary	+, -
8	Segment override	<segment override>:
9	Variable manipulators, creators	SEG, OFFSET, PTR, TYPE, LENGTH, LAST
10	Parentheses/brackets	( ), [ ]
11	Period and Dollar	., $

2.7 Expressions

ASM-86 allows address, numeric, and bracketed expressions. An address expression evaluates to a memory address and has three components:

- A segment value
- An offset value
- A type

Both variables and labels are address expressions. An address expression is not a number, but its components are. Numbers may be combined with operators such as PTR to make an address expression.

A numeric expression evaluates to a number. It does not contain any variables or labels, only numbers and operands.

Bracketed expressions specify base- and index- addressing modes. The base registers are BX and BP, and the index registers are DI and SI. A bracketed expression may consist of a base register, an index register, or a base register and an index register.

Use the + operator between a base register and an index register to specify both base- and index-register addressing. For example:

```
MOV  variable[bx],0
MOV  AX,[BX+DI]
MOV  AX,[SI]
```

2.8 Statements

Just as "tokens" in this assembly language correspond to words in English, so are statements analogous to sentences. A statement tells ASM-86 what action to perform. Statements are of two types: instructions and directives. Instructions are translated by the assembler into 8086 machine language instructions. Directives are not translated into machine code but instead direct the assembler to perform certain clerical functions.

Terminate each assembly language statement with a carriage return (CR) and line feed (LF), or with an exclamation point, !, which ASM-86 treats as an end-of-line except in comments. Multiple assembly language statements can be written on the same physical line if separated by exclamation points.

The ASM-86 instruction set is defined in Section 4. The syntax for an instruction statement is:

```
[label:]  [prefix]  mnemonic [ operand(s)]  [;comment]
```

where the fields are defined as:

label:

> A symbol followed by ":" defines a label at the current value of the location counter in the current segment. This field is optional.

prefix

> Certain machine instructions such as LOCK and REP may prefix other instructions. This field is optional.

mnemonic

> A symbol defined as a machine instruction, either by the assembler or by an EQU directive. This field is optional unless preceded by a prefix instruction. If it is omitted, no operands may be present, although the other fields may appear. ASM-86 mnemonics are defined in Section 4.

operand(s)

An instruction mnemonic may require other symbols to represent operands to the instruction. Instructions may have zero, one or two operands.

comment

Any semicolon (;) appearing outside a character string begins a comment, which is ended by a carriage return. Comments improve the readability of programs. This field is optional.

ASM-86 directives are described in Section 3. The syntax for a directive statement is:

[name] directive operand(s) [;comment]

where the fields are defined as:

name

Unlike the label field of an instruction, the name field of a directive is never terminated with a colon. Directive names are legal for only DB, DW, DD, RS and EQU. For DB, DW, DD and RS the name is optional; for EQU it is required.

directive

One of the directive keywords defined in Section 3.

operand(s)

Analogous to the operands to the instruction mnemonics. Some directives, such as DB, DW, and DD, allow any operand while others have special requirements.

comment

Exactly as defined for instruction statements.

Section 3
Assembler Directives

3.1 Introduction

Directive statements cause ASM-86 to perform housekeeping functions such as assigning portions of code to logical segments, requesting conditional assembly, defining data items, and specifying listing file format. General syntax for directive statements appears in Section 2.8.

In the sections that follow, the specific syntax for each directive statement is given under the heading and before the explanation. These syntax lines use special symbols to represent possible arguments and other alternatives. Square brackets, [], enclose optional arguments. Angle brackets, <>, enclose descriptions of user-supplied arguments. Do not include these symbols when coding a directive.

3.2 Segment Start Directives

At run-time, every 8086 memory reference must have a 16-bit segment base value and a 16-bit offset value. These are combined to produce the 20-bit effective address needed by the CPU to physically address the location. The 16-bit segment base value or boundary is contained in one of the segment registers CS, DS, SS, or ES. The offset value gives the offset of the memory reference from the segment boundary. A 16-byte physical segment is the smallest relocatable unit of memory.

ASM-86 predefines four logical segments: the Code Segment, Data Segment, Stack Segment, and Extra Segment, which are respectively addressed by the CS, DS, SS, and ES registers. Future versions of ASM-86 will support additional segments such as multiple data or code segments. All ASM-86 statements must be assigned to one of the four currently supported segments so that they can be referenced by the CPU. A segment directive statement, CSEG, DSEG, SSEG, or ESEG, specifies that the statements following it belong to a specific segment. The statements are then addressed by the corresponding segment register unless a segment override is included with the instruction. ASM-86 assigns statements to the specified segment until it encounters another segment directive.

Instruction statements must be assigned to the Code Segment. Directive statements may be assigned to any segment. ASM-86 uses these assignments to change from one segment register to another. For example, when an instruction accesses a memory variable, ASM-86 must know which segment contains the variable so it can generate a segment override prefix byte if necessary.

3.2.1 The CSEG Directive

```
CSEG    <numeric expression>
CSEG
CSEG    $
```

This directive tells the assembler that the following statements belong in the Code Segment. All instruction statements must be assigned to the Code Segment. All directive statements are legal within the Code Segment.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Code Segment after it has been interrupted by a DSEG, SSEG, or ESEG directive. The continuing Code Segment starts with the same attributes, such as location and instruction pointer, as the previous Code Segment.

3.2.2 The DSEG Directive

```
DSEG    <numeric expression>
DSEG
DSEG    $
```

This directive specifies that the following statements belong to the Data Segment. The Data Segment primarily contains the data allocation directives DB, DW, DD and RS, but all other directive statements are also legal. Instruction statements are illegal in the Data Segment.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Data Segment after it has been interrupted by a CSEG, SSEG, or ESEG directive. The continuing Data Segment starts with the same attributes as the previous Data Segment.

3.2.3 The SSEG Directive

```
SSEG    <numeric expression>
SSEG
SSEG    $
```

The SSEG directive indicates the beginning of source lines for the Stack Segment. Use the Stack Segment for all stack operations. All directive statements are legal in the Stack Segment, but instruction statements are illegal.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Stack Segment after it has been interrupted by a CSEG, DSEG, or ESEG directive. The continuing Stack Segment starts with the same attributes as the previous Stack Segment.

3.2.4 The ESEG Directive

```
ESEG    <numeric expression>
ESEG
ESEG    $
```

This directive initiates the Extra Segment. Instruction statements are not legal in this segment, but all directive statements are.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Extra Segment after it has been interrupted by a DSEG, SSEG, or CSEG directive. The continuing Extra Segment starts with the same attributes as the previous Extra Segment.

3.3 The ORG Directive

```
ORG     <numeric expression>
```

The ORG directive sets the offset of the location counter in the current segment to the value specified in the numeric expression. Define all elements of the expression before the ORG directive because forward references may be ambiguous.

In most segments, an ORG directive is unnecessary. If no ORG is included before the first instruction or data byte in a segment, assembly begins at location zero relative to the beginning of the segment. A segment can have any number of ORG directives.

3.4 The IF and ENDIF Directives

```
IF      <numeric expression>
        < source line 1 >
        < source line 2 >
                .
                .
                .
        < source line n >
        ENDIF
```

The IF and ENDIF directives allow a group of source lines to be included or excluded from the assembly. Use conditional directives to assemble several different versions of a single source program.

When the assembler finds an IF directive, it evaluates the numeric expression following the IF keyword. If the expression evaluates to a non-zero value, then <source line 1> through <source line n> are assembled. If the expression evaluates to zero, then all lines are listed but not assembled. All elements in the numeric expression must be defined before they appear in the IF directive. Nested IF directives are not legal.

3.5 The INCLUDE Directive

```
INCLUDE     <file name>
```

This directive includes another ASM-86 file in the source text. For example:

```
INCLUDE  EQUALS.A86
```

Use INCLUDE when the source program resides in several different files. INCLUDE directives may not be nested; a source file called by an INCLUDE directive may not contain another INCLUDE statement. If <file name> does not contain a file type, the file type is assumed to be .A86. If no drive name is specified with <file name>, ASM-86 assumes the drive containing the source file.

3.6 The END Directive

```
END
```

An END directive marks the end of a source file. Any subsequent lines are ignored by the assembler. END is optional. If not present, ASM-86 processes the source until it finds an End-Of-File character (1AH).

3.7 The EQU Directive

```
symbol  EQU  <numeric expression>
symbol  EQU  <address expression>
symbol  EQU  <register>
symbol  EQU  <instruction mnemonic>
```

The EQU (equate) directive assigns values and attributes to user-defined symbols. The required symbol name may not be terminated with a colon. The symbol cannot be redefined by a subsequent EQU or another directive. Any elements used in numeric or address expressions must be defined before the EQU directive appears.

The first form assigns a numeric value to the symbol, the second a memory address. The third form assigns a new name to an 8086 register. The fourth form defines a new instruction (sub)set. The following are examples of these four forms:

```
0005            FIVE    EQU     2*2+1
0033            NEXT    EQU     BUFFER
0001            COUNTER EQU     CX
                MOVVV   EQU     MOV
                                .
                                .
                                .
005D 8BC3                       MOVVV   AX,BX
```

3.8 The DB Directive

```
[symbol] DB <numeric expression>[,<numeric expression>..]
[symbol] DB <string constant>[,<string constant>...]
```

The DB directive defines initialized storage areas in byte format. Numeric expressions are evaluated to 8-bit values and sequentially placed in the hex output file. String constants are placed in the output file according to the rules defined in Section 2.4.2. A DB directive is the only ASM-86 statement that accepts a string constant longer than two bytes. There is no translation from lower to upper case within strings. Multiple expressions or constants, separated by commas, may be added to the definition, but may not exceed the physical line length.

Use an optional symbol to reference the defined data area throughout the program. The symbol has four attributes: the Segment and Offset attributes determine the symbol's memory reference, the Type attribute specifies single bytes, and Length tells the number of bytes (allocation units) reserved.

The following statements show DB directives with symbols:

```
005F 43502F4D2073 TEXT    DB      'CP/M system',0
     797374656D00
006B E1           AA      DB      'a' + 80H
006C 0102030405   X       DB      1,2,3,4,5
                                  .
                                  .
                                  .
0071 B90C00                MOV     CX,LENGTH TEXT
```

3.9 The DW Directive

```
[symbol] DW <numeric expression>[,<numeric expression>..]
[symbol] DW <string constant>[,<string constant>...]
```

The DW directive initializes two-byte words of storage. String constants longer than two characters are illegal. Otherwise, DW uses the same procedure to initialize storage as DB except that the low-order byte is stored first, followed by the high-order byte. The following are examples of DW statements:

```
0074 0000          CNTR    DW      0
0076 63C166C169C1  JMPTAB  DW      SUBR1,SUBR2,SUBR3
007C 010002000300          DW      1,2,3,4,5,6
     040005000600
```

3.10 The DD Directive

```
[symbol] DD <numeric expression>[,<numeric expression>..]
```

The DD directive initializes four bytes of storage. The Offset attribute of the address expression is stored in the two lower bytes, the Segment attribute in the two upper bytes. Otherwise, DD follows the same procedure as DB. For example:

```
  1234                     CSEG    1234H
                                   .
                                   .
                                   .
0000 6CC134126FC1 LONG_JMPTAB      DD      ROUT1,ROUT2
     3412
0008 72C1341275C1                  DD      ROUT3,ROUT4
     3412
```

3.11 The RS Directive

[symbol] RS <numeric expression>

The RS directive allocates storage in memory but does not initialize it. The numeric expression gives the number of bytes to be reserved. An RS statement does not give a byte attribute to the optional symbol. For example:

```
0010            BUF     RS      80
0060                    RS      4000H
4060                    RS      1
```

3.12 The RB Directive

[symbol] RB <numeric expression>

The RB directive allocates byte storage in memory without any initialization. This directive is identical to the RS directive except that it does give the byte attribute.

3.13 The RW Directive

[symbol] RW <numeric expression>

The RW directive allocates two-byte word storage in memory but does not initialize it. The numeric expression gives the number of words to be reserved. For example:

```
4061            BUFF    RW      128
4161                    RW      4000H
C161                    RW      1
```

3.14 The TITLE Directive

TITLE <string constant>

ASM-86 prints the string constant defined by a TITLE directive statement at the top of each printout page in the listing file. The title character string should not exceed 30 characters. For example:

```
TITLE 'CP/M monitor'
```

3.15 The PAGESIZE Directive

PAGESIZE <numeric expression>

The PAGESIZE directive defines the number of lines to be included on each printout page. The default pagesize is 66.

3.16 The PAGEWIDTH Directive

```
PAGEWIDTH <numeric expression>
```

The PAGEWIDTH directive defines the number of columns printed across the page when the listing file is output. The default pagewidth is 120 unless the listing is routed directly to the terminal; then the default pagewidth is 79.

3.17 The EJECT Directive

```
EJECT
```

The EJECT directive performs a page eject during printout. The EJECT directive itself is printed on the first line of the next page.

3.18 The SIMFORM Directive

```
SIMFORM
```

The SIMFORM directive replaces a form-feed (FF) character in the print file with the correct number of line-feeds (LF). Use this directive when printing out on a printer unable to interpret the form-feed character.

3.19 The NOLIST and LIST Directives

```
NOLIST
LIST
```

The NOLIST directive blocks the printout of the following lines. Restart the listing with a LIST directive.

Section 4
The ASM-86 Instruction Set

4.1 Introduction

The ASM-86 instruction set includes all 8086 machine instructions. The general syntax for instruction statements is given in Section 2.7. The following sections define the specific syntax and required operand types for each instruction, without reference to labels or comments. The instruction definitions are presented in tables for easy reference. For a more detailed description of each instruction, see Intel's MCS-86 Assembly Language Reference Manual. For descriptions of the instruction bit patterns and operations, see Intel's MCS-86 User's Manual.

The instruction-definition tables present ASM-86 instruction statements as combinations of mnemonics and operands. A mnemonic is a symbolic representation for an instruction, and its operands are its required parameters. Instructions can take zero, one or two operands. When two operands are specified, the left operand is the instruction's destination operand, and the two operands are separated by a comma.

The instruction-definition tables organize ASM-86 instructions into functional groups. Within each table, the instructions are listed alphabetically. Table 4-1 shows the symbols used in the instruction-definition tables to define operand types.

Table 4-1. Operand Type Symbols

Symbol	Operand Type
numb	any NUMERIC expression
numb8	any NUMERIC expression which evaluates to an 8-bit number
acc	accumulator register, AX or AL
reg	any general purpose register, not segment register
reg16	a 16-bit general purpose register, not segment register
segreg	any segment register: CS, DS, SS, or ES

Table 4-1. (continued)

Symbol	Operand Type
mem	any ADDRESS expression, with or without base- and/or index- addressing modes, such as: variable variable+3 variable[bx] variable[SI] variable[BX+SI] [BX] [BP+DI]
simpmem	any ADDRESS expression WITHOUT base- and index- addressing modes, such as: variable variable+4
mem|reg	any expression symbolized by "reg" or "mem"
mem|reg16	any expression symbolized by "mem|reg", but must be 16 bits
label	any ADDRESS expression which evaluates to a label
lab8	any "label" which is within +/- 128 bytes distance from the instruction

The 8086 CPU has nine single-bit Flag registers which reflect the state of the CPU. The user cannot access these registers directly, but can test them to determine the effects of an executed instruction upon an operand or register. The effects of instructions on Flag registers are also described in the instruction-definition tables, using the symbols shown in Table 5-2 to represent the nine Flag registers.

Table 4-2. Flag Register Symbols

AF	Auxiliary-Carry-Flag
CF	Carry-Flag
DF	Direction-Flag
IF	Interrupt-Enable-Flag
OF	Overflow-Flag
PF	Parity-Flag
SF	Sign-Flag
TF	Trap-Flag
ZF	Zero-Flag

4.2 Data Transfer Instructions

There are four classes of data transfer operations: general purpose, accumulator specific, address-object and flag. Only SAHF and POPF affect flag settings. Note in Table 4-3 that if acc = AL, a byte is transferred, but if acc = AX, a word is transferred.

Table 4-3. Data Transfer Instructions

Syntax		Result
IN	acc,numb8|numb16	transfer data from input port given by numb8 or numb16 (0-255) to accumulator
IN	acc,DX	transfer data from input port given by DX register (0-0FFFFH) to accumulator
LAHF		transfer SF, ZF, AF, PF, and CF flags to the AH register
LDS	reg16,mem	transfer the segment part of the memory address (DWORD variable) to the DS segment register, transfer the offset part to a general purpose 16-bit register
LEA	reg16,mem	transfer the offset of the memory address to a (16-bit) register
LES	reg16,mem	transfer the segment part of the memory address to the ES segment register, transfer the offset part to a 16-bit general purpose register
MOV	reg,mem|reg	move memory or register to register
MOV	mem|reg,reg	move register to memory or register

Table 4-3. (continued)

Syntax		Result
MOV	mem|reg,numb	move immediate data to memory or register
MOV	segreg,mem|reg16	move memory or register to segment register
MOV	mem|reg16,segreg	move segment register to memory or register
OUT	numb8|numb16,acc	transfer data from accumulator to output port (0-255) given by numb8 or numb16
OUT	DX,acc	transfer data from accumulator to output port (0-0FFFFH) given by DX register
POP	mem|reg16	move top stack element to memory or register
POP	segreg	move top stack element to segment register; note that CS segment register not allowed
POPF		transfer top stack element to flags
PUSH	mem|reg16	move memory or register to top stack element
PUSH	segreg	move segment register to top stack element
PUSHF		transfer flags to top stack element
SAHF		transfer the AH register to flags
XCHG	reg,mem|reg	exchange register and memory or register
XCHG	mem|reg,reg	exchange memory or register and register
XLAT	mem|reg	perform table lookup translation, table given by "mem|reg", which is always BX. Replaces AL with AL offset from BX.

4.3 Arithmetic, Logical, and Shift Instructions

The 8086 CPU performs the four basic mathematical operations in several different ways. It supports both 8- and 16-bit operations and also signed and unsigned arithmetic.

Six of the nine flag bits are set or cleared by most arithmetic operations to reflect the result of the operation. Table 4-4 summarizes the effects of arithmetic instructions on flag bits. Table 4-5 defines arithmetic instructions and Table 4-6 logical and shift instructions.

Table 4-4. Effects of Arithmetic Instructions on Flags

CF	is set if the operation resulted in a carry out of (from addition) or a borrow into (from subtraction) the high-order bit of the result; otherwise CF is cleared.
AF	is set if the operation resulted in a carry out of (from addition) or a borrow into (from subtraction) the low-order four bits of the result; otherwise AF is cleared.
ZF	is set if the result of the operation is zero; otherwise ZF is cleared.
SF	is set if the result is negative.
PF	is set if the modulo 2 sum of the low-order eight bits of the result of the operation is 0 (even parity); otherwise PF is cleared (odd parity).
OF	is set if the operation resulted in an overflow; the size of the result exceeded the capacity of its destination.

Table 4-5. Arithmetic Instructions

Syntax		Result
AAA		adjust unpacked BCD (ASCII) for addition - adjusts AL
AAD		adjust unpacked BCD (ASCII) for division - adjusts AL
AAM		adjust unpacked BCD (ASCII) for multiplication - adjusts AX
AAS		adjust unpacked BCD (ASCII) for subtraction - adjusts AL
ADC	reg,mem|reg	add (with carry) memory or register to register
ADC	mem|reg,reg	add (with carry) register to memory or register
ADC	mem|reg,numb	add (with carry) immediate data to memory or register
ADD	reg,mem|reg	add memory or register to register
ADD	mem|reg,reg	add register to memory or register
ADD	mem|reg,numb	add immediate data to memory or register
CBW		convert byte in AL to word in AH by sign extension
CWD		convert word in AX to double word in DX/AX by sign extension
CMP	reg,mem|reg	compare register with memory or register
CMP	mem|reg,reg	compare memory or register with register
CMP	mem|reg,numb	compare data constant with memory or register
DAA		decimal adjust for addition, adjusts AL
DAS		decimal adjust for subtraction, adjusts AL
DEC	mem|reg	subtract 1 from memory or register

Table 4-5. (continued)

Syntax		Result
INC	mem|reg	add 1 to memory or register
DIV	mem|reg	divide (unsigned) accumulator (AX or AL) by memory or register. If byte results, AL = quotient, AH = remainder. If word results, AX = quotient, DX = remainder
IDIV	mem|reg	divide (signed) accumulator (AX or AL) by memory or register - quotient and remainder stored as in DIV
IMUL	mem|reg	multiply (signed) memory or register by accumulator (AX or AL) - if byte, results in AH, AL. If word, results in DX, AX
MUL	mem|reg	multiply (unsigned) memory or register by accumulator (AX or AL) - results stored as in IMUL
NEG	mem|reg	two's complement memory or register
SBB	reg,mem|reg	subtract (with borrow) memory or register from register
SBB	mem|reg,reg	subtract (with borrow) register from memory or register
SBB	mem|reg,numb	subtract (with borrow) immediate data from memory or register
SUB	reg,mem|reg	subtract memory or register from register
SUB	mem|reg,reg	subtract register from memory or register
SUB	mem|reg,numb	subtract data constant from memory or register

Table 4-6. Logic and Shift Instructions

Syntax		Result
AND	reg,mem|reg	perform bitwise logical "and" of a register and memory register
AND	mem|reg,reg	perform bitwise logical "and" of memory register and register
AND	mem|reg,numb	perform bitwise logical "and" of memory register and data constant
NOT	mem|reg	form ones complement of memory or register
OR	reg,mem|reg	perform bitwise logical "or" of a register and memory register
OR	mem|reg,reg	perform bitwise logical "or" of memory register and register
OR	mem|reg,numb	perform bitwise logical "or" of memory register and data constant
RCL	mem|reg,1	rotate memory or register 1 bit left through carry flag
RCL	mem|reg,CL	rotate memory or register left through carry flag, number of bits given by CL register
RCR	mem|reg,1	rotate memory or register 1 bit right through carry flag
RCR	mem|reg,CL	rotate memory or register right through carry flag, number of bits given by CL register
ROL	mem|reg,1	rotate memory or register 1 bit left
ROL	mem|reg,CL	rotate memory or register left, number of bits given by CL register
ROR	mem|reg,1	rotate memory or register 1 bit right
ROR	mem|reg,CL	rotate memory or register right, number of bits given by CL register
SAL	mem|reg,1	shift memory or register 1 bit left, shift in low-order zero bits

Table 4-6. (continued)

Syntax		Result
SAL	mem|reg,CL	shift memory or register left, number of bits given by CL register, shift in low-order zero bits
SAR	mem|reg,1	shift memory or register 1 bit right, shift in high-order bits equal to the original high-order bit
SAR	mem|reg,CL	shift memory or register right, number of bits given by CL register, shift in high-order bits equal to the original high-order bit
SHL	mem|reg,1	shift memory or register 1 bit left, shift in low-order zero bits - note that SHL is a different mnemonic for SAL
SHL	mem|reg,CL	shift memory or register left, number of bits given by CL register, shift in low-order zero bits - note that SHL is a different mnemonic for SAL
SHR	mem|reg,1	shift memory or register 1 bit right, shift in high-order zero bits
SHR	mem|reg,CL	shift memory or register right, number of bits given by CL register, shift in high-order zero bits
TEST	reg,mem|reg	perform bitwise logical "and" of a register and memory or register - set condition flags but do not change destination
TEST	mem|reg,reg	perform bitwise logical "and" of memory register and register - set condition flags but do not change destination
TEST	mem|reg,numb	perform bitwise logical "and" - test of memory register and data constant - set condition flags but do not change destination

Table 4-6. (continued)

Syntax		Result
XOR	reg,mem|reg	perform bitwise logical "exclusive OR" of a register and memory or register
XOR	mem|reg,reg	perform bitwise logical "exclusive OR" of memory register and register
XOR	mem|reg,numb	perform bitwise logical "exclusive OR" of memory register and data constant

4.4 String Instructions

String instructions take one or two operands. The operands specify only the operand type, determining whether operation is on bytes or words. If there are two operands, the source operand is addressed by the SI register and the destination operand is addressed by the DI register. The DI and SI registers are always used for addressing. Note that for string operations, destination operands addressed by DI must always reside in the Extra Segment (ES).

Table 4-7. String Instructions

Syntax		Result
CMPS	mem|reg,mem|reg	subtract source from destination, affect flags, but do not return result.
LODS	mem|reg	transfer a byte or word from the source operand to the accumulator.
MOVS	mem|reg,mem|reg	move 1 byte (or word) from source to destination.
SCAS	mem|reg	subtract destination operand from accumulator (AX or AL), affect flags, but do not return result.
STOS	mem|reg	transfer a byte or word from accumulator to the destination operand.

Table 4-8 defines prefixes for string instructions. A prefix repeats its string instruction the number of times contained in the CX register, which is decremented by 1 for each iteration. Prefix mnemonics precede the string instruction mnemonic in the statement line as shown in Section 2.8.

Table 4-8. Prefix Instructions

Syntax	Result
REP	repeat until CX register is zero
REPZ	repeat until CX register is zero and zero flag (ZF) is not zero
REPE	equal to "REPZ"
REPNZ	repeat until CX register is zero and zero flag (ZF) is zero
REPNE	equal to "REPNZ"

4.5 Control Transfer Instructions

There are four classes of control transfer instructions:

- calls, jumps, and returns
- conditional jumps
- iterational control
- interrupts

All control transfer instructions cause program execution to continue at some new location in memory, possibly in a new code segment. The transfer may be absolute or depend upon a certain condition. Table 4-9 defines control transfer instructions. In the definitions of conditional jumps, "above" and "below" refer to the relationship between unsigned values, and "greater than" and "less than" refer to the relationship between signed values.

Table 4-9. Control Transfer Instructions

Syntax		Result
CALL	label	push the offset address of the next instruction on the stack, jump to the target label
CALL	mem|reg16	push the offset address of the next instruction on the stack, jump to location indicated by contents of specified memory or register
CALLF	label	push CS segment register on the stack, push the offset address of the next instruction on the stack (after CS), jump to the target label
CALLF	mem	push CS register on the stack, push the offset address of the next instruction on the stack, jump to location indicated by contents of specified double word in memory
INT	numb8	push the flag registers (as in PUSHF), clear TF and IF flags, transfer control with an indirect call through any one of the 256 interrupt-vector elements - uses three levels of stack
INTO		if OF (the overflow flag) is set, push the flag registers (as in PUSHF), clear TF and IF flags, transfer control with an indirect call through interrupt-vector element 4 (location 10H) - if the OF flag is cleared, no operation takes place
IRET		transfer control to the return address saved by a previous interrupt operation, restore saved flag registers, as well as CS and IP - pops three levels of stack
JA	lab8	jump if "not below or equal" or "above" ((CF or ZF)=0)

Table 4-9. (continued)

Syntax		Result
JAE	lab8	jump if "not below" or "above or equal" (CF=0)
JB	lab8	jump if "below" or "not above or equal" (CF=1)
JBE	lab8	jump if "below or equal" or "not above" ((CF or ZF)=1)
JC	lab8	same as "JB"
JCXZ	lab8	jump to target label if CX register is zero
JE	lab8	jump if "equal" or "zero" (ZF=1)
JG	lab8	jump if "not less or equal" or "greater" (((SF xor OF) or ZF)=0)
JGE	lab8	jump if "not less" or "greater or equal" ((SF xor OF)=0)
JL	lab8	jump if "less" or "not greater or equal" ((SF xor OF)=1)
JLE	lab8	jump if "less or equal" or "not greater" (((SF xor OF) or ZF)=1)
JMP	label	jump to the target label
JMP	mem|reg16	jump to location indicated by contents of specified memory or register
JMPF	label	jump to the target label possibly in another code segment
JMPS	lab8	jump to the target label within +/- 128 bytes from instruction
JNA	lab8	same as "JBE"
JNAE	lab8	same as "JB"
JNB	lab8	same as "JAE"
JNBE	lab8	same as "JA"
JNC	lab8	same as "JNB"

Table 4-9. (continued)

Syntax		Result
JNE	lab8	jump if "not equal" or "not zero" (ZF=0)
JNG	lab8	same as "JLE"
JNGE	lab8	same as "JL"
JNL	lab8	same as "JGE"
JNLE	lab8	same as "JG"
JNO	lab8	jump if "not overflow" (OF=0)
JNP	lab8	jump if "not parity" or "parity odd"
JNS	lab8	jump if "not sign"
JNZ	lab8	same as "JNE"
JO	lab8	jump if "overflow" (OF=1)
JP	lab8	jump if "parity" or "parity even" (PF=1)
JPE	lab8	same as "JP"
JPO	lab8	same as "JNP"
JS	lab8	jump if "sign" (SF=1)
JZ	lab8	same as "JE"
LOOP	lab8	decrement CX register by one, jump to target label if CX is not zero
LOOPE	lab8	decrement CX register by one, jump to target label if CX is not zero and the ZF flag is set - "loop while zero" or "loop while equal"
LOOPNE	lab8	decrement CX register by one, jump to target label if CX is not zero and ZF flag is cleared - "loop while not zero" or "loop while not equal"
LOOPNZ	lab8	same as "LOOPNE"
LOOPZ	lab8	same as "LOOPE"

Table 4-9. (continued)

Syntax	Result
RET	return to the return address pushed by a previous CALL instruction, increment stack pointer by 2
RET numb	return to the address pushed by a previous CALL, increment stack pointer by 2+numb
RETF	return to the address pushed by a previous CALLF instruction, increment stack pointer by 4
RETF numb	return to the address pushed by a previous CALLF instruction, increment stack pointer by 4+numb

4.6 Processor Control Instructions

Processor control instructions manipulate the flag registers. Moreover, some of these instructions can synchronize the 8086 CPU with external hardware.

Table 4-10. Processor Control Instructions

Syntax	Results
CLC	clear CF flag
CLD	clear DF flag, causing string instructions to auto-increment the operand pointers
CLI	clear IF flag, disabling maskable external interrupts
CMC	complement CF flag
ESC numb8,mem|reg	do no operation other than compute the effective address and place it on the address bus (ESC is used by the 8087 numeric co-processor), "numb8" must be in the range 0 to 63

Table 4-10. (continued)

Syntax	Results
LOCK	PREFIX instruction, cause the 8086 processor to assert the "bus-lock" signal for the duration of the operation caused by the following instruction - the LOCK prefix instruction may precede any other instruction - buslock prevents co-processors from gaining the bus; this is useful for shared-resource semaphores
HLT	cause 8086 processor to enter halt state until an interrupt is recognized
STC	set CF flag
STD	set DF flag, causing string instructions to auto-decrement the operand pointers
STI	set IF flag, enabling maskable external interrupts
WAIT	cause the 8086 processor to enter a "wait" state if the signal on its "TEST" pin is not asserted

Section 5
Code-Macro Facilities

5.1 Introduction to Code-macros

ASM-86 does not support traditional assembly-language macros, but it does allow the user to define his own instructions by using the Code-macro directive. Like traditional macros, code-macros are assembled wherever they appear in assembly language code, but there the similarity ends. Traditional macros contain assembly language instructions, but a code-macro contains only code-macro directives. Macros are usually defined in the user's symbol table; ASM-86 code-macros are defined in the assembler's symbol table. A macro simplifies using the same block of instructions over and over again throughout a program, but a code-macro sends a bit stream to the output file and in effect adds a new instruction to the assembler.

Because ASM-86 treats a code-macro as an instruction, you can invoke code-macros by using them as instructions in your program. The example below shows how MAC, an instruction defined by a code-macro, can be invoked.

```
        .
        .
        .
XCHG BX,WORD3
MAC  PAR1,PAR2
MUL  AX,WORD4
        .
        .
        .
```

Note that MAC accepts two operands. When MAC was defined, these two operands were also classified as to type, size, and so on by defining MAC's formal parameters. The names of formal parameters are not fixed. They are stand-ins which are replaced by the names or values supplied as operands when the code-macro is invoked. Thus formal parameters "hold the place" and indicate where and how the operands are to be used.

The definition of a code-macro starts with a line specifying its name and its formal parameters, if any:

CodeMacro <name> [<formal parameter list>]

where the optional <formal parameter list> is defined:

<formal name>:<specifier letter>[<modifier letter>][<range>]

As stated above, the formal name is not fixed, but a place holder. If formal parameter list is present, the specifier letter is required and the modifier letter is optional. Possible specifiers are A, C, D, E, M, R, S, and X. Possible modifier letters are b, d, w, and sb. The assembler ignores case except within strings, but for clarity, this section shows specifiers in upper-case and modifiers in lower-case. Following sections describe specifiers, modifiers, and the optional range in detail.

The body of the code-macro describes the bit pattern and formal parameters. Only the following directives are legal within code-macros:

```
SEGFIX
NOSEGFIX
MODRM
RELB
RELW
DB
DW
DD
DBIT
```

These directives are unique to code-macros, and those which appear to duplicate ASM-86 directives (DB, DW, and DD) have different meanings in code-macro context. These directives are discussed in detail in later sections. The definition of a code-macro ends with a line:

```
EndM
```

CodeMacro, EndM, and the code-macro directives are all reserved words. Code-macro definition syntax is defined in Backus-Naur-like form in Appendix H. The following examples are typical code-macro definitions.

```
CodeMacro AAA
  DB 37H
EndM

CodeMacro DIV divisor:Eb
  SEGFIX divisor
  DB     6FH
  MODRM  divisor
EndM

CodeMacro ESC opcode:Db(0,63),src:Eb
  SEGFIX src
  DBIT 5(1BH),3(opcode(3))
  MODRM  opcode,src
EndM
```

5.2 Specifiers

Every formal parameter must have a specifier letter that indicates what type of operand is needed to match the formal parameter. Table 5-1 defines the eight possible specifier letters.

Table 5-1. Code-macro Operand Specifiers

Letter	Operand Type
A	Accumulator register, AX or AL.
C	Code, a label expression only.
D	Data, a number to be used as an immediate value.
E	Effective address, either an M (memory address) or an R (register).
M	Memory address. This can be either a variable or a bracketed register expression.
R	A general register only.
S	Segment register only.
X	A direct memory reference.

5.3 Modifiers

The optional modifier letter is a further requirement on the operand. The meaning of the modifier letter depends on the type of the operand. For variables, the modifier requires the operand to be of type: "b" for byte, "w" for word, "d" for double-word and "sb" for signed byte. For numbers, the modifiers require the number to be of a certain size: "b" for -256 to 255 and "w" for other numbers. Table 5-2 summarizes code-macro modifiers.

Table 5-2. Code-macro Operand Modifiers

Variables		Numbers	
Modifier	Type	Modifier	Size
b	byte	b	-256 to 255
w	word	w	anything else
d	dword		
sb	signed byte		

5.4 Range Specifiers

The optional range is specified within parentheses by either one expression or two expressions separated by a comma. The following are valid formats:

```
(numberb)
(register)
(numberb,numberb)
(numberb,register)
(register,numberb)
(register,register)
```

Numberb is 8-bit number, not an address. The following example specifies that the input port must be identified by the DX register:

```
CodeMacro IN dst:Aw,port:Rw(DX)
```

The next example specifies that the CL register is to contain the "count" of rotation:

```
CodeMacro ROR dst:Ew,count:Rb(CL)
```

The last example specifies that the "opcode" is to be immediate data, and may range from 0 to 63 inclusive:

```
CodeMacro ESC opcode:Db(0,63),adds:Eb
```

5.5 Code-macro Directives

Code-macro directives define the bit pattern and make further requirements on how the operand is to be treated. Directives are reserved words, and those that appear to duplicate assembly language instructions have different meanings within a code-macro definition. Only the nine directives defined here are legal within code-macro definitions.

5.5.1 SEGFIX

If SEGFIX is present, it instructs the assembler to determine whether a segment-override prefix byte is needed to access a given memory location. If so, it is output as the first byte of the instruction. If not, no action is taken. SEGFIX takes the form:

```
SEGFIX <formal name>
```

where <formal name> is the name of a formal parameter which represents the memory address. Because it represents a memory address, the formal parameter must have one of the specifiers E, M or X.

5.5.2 NOSEGFIX

Use NOSEGFIX for operands in instructions that must use the ES register for that operand. This applies only to the destination operand of these instructions: CMPS, MOVS, SCAS, STOS. The form of NOSEGFIX is:

```
NOSEGFIX   segreg,<formname>
```

where segreg is one of the segment registers ES, CS, SS, or DS and <formname> is the name of the memory-address formal parameter, which must have a specifier E, M, or X. No code is generated from this directive, but an error check is performed. The following is an example of NOSEGFIX use:

```
CodeMacro MOVS si_ptr:Ew,di_ptr:Ew
  NOSEGFIX    ES,di_ptr
  SEGFIX      si_ptr
  DB          0A5H
EndM
```

5.5.3 MODRM

This directive intructs the assembler to generate the ModRM byte, which follows the opcode byte in many of the 8086's instructions. The ModRM byte contains either the indexing type or the register number to be used in the instruction. It also specifies which register is to be used, or gives more information to specify an instruction.

The ModRM byte carries the information in three fields: The mod field occupies the two most significant bits of the byte, and combines with the register memory field to form 32 possible values: 8 registers and 24 indexing modes.

The reg field occupies the three next bits following the mod field. It specifies either a register number or three more bits of opcode information. The meaning of the reg field is determined by the opcode byte.

The register memory field occupies the last three bits of the byte. It specifies a register as the location of an operand, or forms a part of the address-mode in combination with the mod field described above.

For further information of the 8086's instructions and their bit patterns, see Intel's 8086 Assembly Language Programing Manual and the Intel 8086 Family User's Manual. The forms of MODRM are:

```
MODRM  <form name>,<form name>
MODRM  NUMBER7,<form name>
```

where NUMBER7 is a value 0 to 7 inclusive and <form name> is the name of a formal parameter. The following examples show MODRM use:

```
CodeMacro RCR dst:Ew,count:Rb(CL)
  SEGFIX      dst
  DB          0D3H
  MODRM       3,dst
EndM

CodeMacro OR dst:Rw,src:Ew
  SEGFIX      src
  DB          0BH
  MODRM       dst,src
EndM
```

5.5.4 RELB and RELW

These directives, used in IP-relative branch instructions, instruct the assembler to generate displacement between the end of the instruction and the label which is supplied as an operand. RELB generates one byte and RELW two bytes of displacement. The directives the following forms:

```
RELB <form name>
RELW <form name>
```

where <form name> is the name of a formal parameter with a "C" (code) specifier. For example:

```
CodeMacro LOOP place:Cb
  DB          0E2H
  RELB        place
EndM
```

5.5.5 DB, DW and DD

These directives differ from those which occur outside of code-macros. The form of the directives are:

```
DB <form name> | NUMBERB
DW <form name> | NUMBERW
DD <form name>
```

where NUMBERB is a single-byte number, NUMBERW is a two-byte number, and <form name> is a name of a formal parameter. For example:

```
CodeMacro XOR dst:Ew,src:Db
  SEGFIX      dst
  DB          81H
  MODRM       6,dst
  DW          src
EndM
```

5.5.6 DBIT

This directive manipulates bits in combinations of a byte or less. The form is:

```
DBIT <field description>[,<field description>]
```

where a <field description>, has two forms:

```
<number><combination>
<number>(<form name>(<rshift>))
```

where <number> ranges from 1 to 16, and specifies the number of bits to be set. <combination> specifies the desired bit combination. The total of all the <number>s listed in the field descriptions must not exceed 16. The second form shown above contains <form name>, a formal parameter name that instructs the assembler to put a certain number in the specified position. This number normally refers to the register specified in the first line of the code-macro. The numbers used in this special case for each register are:

```
AL:    0
CL:    1
DL:    2
BL:    3
AH:    4
CH:    5
DH:    6
BH:    7
AX:    0
CX:    1
DX:    2
BX:    3
SP:    4
BP:    5
SI:    6
DI:    7
ES:    0
CS:    1
SS:    2
DS:    3
```

<rshift>, which is contained in the innermost parentheses, specifies a number of right shifts. For example, "0" specifies no shift, "1" shifts right one bit, "2" shifts right two bits, and so on. The definition below uses this form.

```
CodeMacro DEC dst:Rw
  DBIT 5(9H),3(dst(0))
EndM
```

The first five bits of the byte have the value 9H. If the remaining bits are zero, the hex value of the byte will be 48H. If the instruction:

```
DEC     DX
```

is assembled and DX has a value of 2H, then 48H + 2H = 4AH, which is the final value of the byte for execution. If this sequence had been present in the definition:

```
DBIT 5(9H),3(dst(1))
```

then the register number would have been shifted right once and the result would had been 48H + 1H = 49H, which is erroneous.

Section 6
DDT-86

6.1 DDT-86 Operation

The DDT-86™ program allows the user to test and debug programs interactively in a CP/M-86 environment. The reader should be familiar with the 8086 processor, ASM-86 and the CP/M-86 operating system as described in the CP/M-86 System Guide.

6.1.1 Invoking DDT-86

Invoke DDT-86 by entering one of the following commands:

```
DDT86
DDT86 filename
```

The first command simply loads and executes DDT-86. After displaying its sign-on message and prompt character, - , DDT-86 is ready to accept operator commands. The second command is similar to the first, except that after DDT-86 is loaded it loads the file specified by filename. If the file type is omitted from filename, .CMD is assumed. Note that DDT-86 cannot load a file of type .H86. The second form of the invoking command is equivalent to the sequence:

```
A>DDT86
DDT86 x.x
-Efilename
```

At this point, the program that was loaded is ready for execution.

6.1.2 DDT-86 Command Conventions

When DDT-86 is ready to accept a command, it prompts the operator with a hyphen, -. In response, the operator can type a command line or a CONTROL-C (represented in this chapter as ↑C) to end the debugging session (see Section 6.1.4). A command line may have up to 64 characters, and must be terminated with a carriage return. While entering the command, use standard CP/M line-editing functions (↑X, ↑H, ↑R, etc.) to correct typing errors. DDT-86 does not process the command line until a carriage return is entered.

The first character of each command line determines the command action. Table 6-1 summarizes DDT-86 commands. DDT-86 commands are defined individually in Section 6.2.

Table 6-1. DDT-86 Command Summary

Command	Action
A	enter assembly language statements
D	display memory in hexadecimal and ASCII
E	load program for execution
F	fill memory block with a constant
G	begin execution with optional breakpoints
H	hexadecimal arithmetic
I	set up file control block and command tail
L	list memory using 8086 mnemonics
M	move memory block
R	read disk file into memory
S	set memory to new values
T	trace program execution
U	untraced program monitoring
V	show memory layout of disk file read
W	write contents of memory block to disk
X	examine and modify CPU state

The command character may be followed by one or more arguments, which may be hexadecimal values, file names or other information, depending on the command. Arguments are separated from each other by commas or spaces. No spaces are allowed between the command character and the first argument.

6.1.3 Specifying a 20-Bit Address

Most DDT-86 commands require one or more addresses as operands. Because the 8086 can address up to 1 megabyte of memory, addresses must be 20-bit values. Enter a 20-bit address as follows:

```
ssss:oooo
```

where ssss represents an optional 16-bit segment number and oooo is a 16-bit offset. DDT-86 combines these values to produce a 20-bit effective address as follows:

```
 ssss0
+ oooo
 -----
 eeeee
```

The optional value ssss may be a 16-bit hexadecimal value or the name of a segment register. If a segment register name is specified, the value of ssss is the contents of that register in the user's CPU state, as displayed by the X command. If omitted, a default value appropriate to the command being executed is used as described in Section 6.4.

6.1.4 Terminating DDT-86

Terminate DDT-86 by typing a ↑C in response to the hyphen prompt. This returns control to the CCP. Note that CP/M-86 does not have the SAVE facility found in CP/M for 8-bit machines. Thus if DDT-86 is used to patch a file, write the file to disk using the W command before exiting DDT-86.

6.1.5 DDT-86 Operation with Interrupts

DDT-86 operates with interrupts enabled or disabled, and preserves the interrupt state of the program being executed under DDT-86. When DDT-86 has control of the CPU, either when it is initially invoked, or when it regains control from the program being tested, the condition of the interrupt flag is the same as it was when DDT-86 was invoked, except for a few critical regions where interrupts are disabled. While the program being tested has control of the CPU, the user's CPU state determines the state of the interrupt flag.

6.2 DDT-86 Commands

This section defines DDT-86 commands and their arguments. DDT-86 commands give the user control of program execution and allow the user to display and modify system memory and the CPU state.

6.2.1 The A (Assemble) Command

The A command assembles 8086 mnemonics directly into memory. The form is:

```
As
```

where s is the 20-bit address where assembly is to start. DDT-86 responds to the A command by displaying the address of the memory location where assembly is to begin. At this point the operator enters assembly language statements as described in Section 4 on Assembly Language Syntax. When a statement is entered, DDT-86 converts it to machine code, places the value(s) in memory, and displays the address of the next available memory location. This process continues until the user enters a blank line or a line containing only a period.

DDT-86 responds to invalid statements by displaying a question mark, ? , and redisplaying the current assembly address.

6.2.2 The D (Display) Command

The D command displays the contents of memory as 8-bit or 16-bit hexadecimal values and in ASCII. The forms are:

```
D
Ds
Ds,f
DW
DWs
DWs,f
```

where s is the 20-bit address where the display is to start, and f is the 16-bit offset within the segment specified in s where the display is to finish.

Memory is displayed on one or more display lines. Each display line shows the values of up to 16 memory locations. For the first three forms, the display line appears as follows:

```
ssss:oooo bb bb . . . bb cc . . . c
```

where ssss is the segment being displayed and oooo is the offset within segment ssss. The bb's represent the contents of the memory locations in hexadecimal, and the c's represent the contents of memory in ASCII. Any non-graphic ASCII characters are represented by periods.

In response to the first form shown above, DDT-86 displays memory from the current display address for 12 display lines. The response to the second form is similar to the first, except that the display address is first set to the 20-bit address s. The third form displays the memory block between locations s and f. The next three forms are analogous to the first three, except that the contents of memory are displayed as 16-bit values, rather than 8-bit values, as shown below:

```
ssss:oooo wwww wwww . . . wwww cccc . . . cc
```

During a long display, the D command may be aborted by typing any character at the console.

6.2.3 The E (Load for Execution) Command

The E command loads a file into memory so that a subsequent G, T or U command can begin program execution. The E command takes the form:

```
E<filename>
```

where <filename> is the name of the file to be loaded. If no file type is specified, .CMD is assumed. The contents of the user segment registers and IP register are altered according to the information in the header of the file loaded.

An E command releases any blocks of memory allocated by any previous E or R commands or by programs executed under DDT-86. Thus only one file at a time may be loaded for execution.

When the load is complete, DDT-86 displays the start and end addresses of each segment in the file loaded. Use the V command to redisplay this information at a later time.

If the file does not exist or cannot be successfully loaded in the available memory, DDT-86 issues an error message.

6.2.4 The F (Fill) Command

The F command fills an area of memory with a byte or word constant. The forms are:

```
Fs,f,b
FWs,f,w
```

where s is a 20-bit starting address of the block to be filled, and f is a 16-bit offset of the final byte of the block within the segment specified in s.

In response to the first form, DDT-86 stores the 8-bit value b in locations s through f. In the second form, the 16-bit value w is stored in locations s through f in standard form, low 8 bits first followed by high 8 bits.

If s is greater than f or the value b is greater than 255, DDT-86 responds with a question mark. DDT-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existent RAM at the location indicated.

6.2.5 The G (Go) Command

The G command transfers control to the program being tested, and optionally sets one or two breakpoints. The forms are:

```
G
G,b1
G,b1,b2
Gs
Gs,b1
Gs,b1,b2
```

where s is a 20-bit address where program execution is to start, and b1 and b2 are 20-bit addresses of breakpoints. If no segment value is supplied for any of these three addresses, the segment value defaults to the contents of the CS register.

In the first three forms, no starting address is specified, so DDT-86 derives the 20-bit address from the user's CS and IP registers. The first form transfers control to the user's program without setting any breakpoints. The next two forms respectively set one and two breakpoints before passing control to the user's program. The next three forms are analogous to the first three, except that the user's CS and IP registers are first set to s.

Once control has been transferred to the program under test, it executes in real time until a breakpoint is encountered. At this point, DDT-86 regains control, clears all breakpoints, and indicates the address at which execution of the program under test was interrupted as follows:

```
*ssss:oooo
```

where ssss corresponds to the CS and oooo corresponds to the IP where the break occurred. When a breakpoint returns control to DDT-86, the instruction at the breakpoint address has not yet been executed.

6.2.6 The H (Hexadecimal Math) Command

The H command computes the sum and difference of two 16-bit values. The form is:

```
Ha,b
```

where a and b are the values whose sum and difference are to be computed. DDT-86 displays the sum (ssss) and the difference (dddd) truncated to 16 bits on the next line as shown below:

```
ssss dddd
```

6.2.7 The I (Input Command Tail) Command

The I command prepares a file control block and command tail buffer in DDT-86's base page, and copies this information into the base page of the last file loaded with the E command. The form is:

```
I<command tail>
```

where <command tail> is a character string which usually contains one or more filenames. The first filename is parsed into the default file control block at 005CH. The optional second filename (if specified) is parsed into the second part of the default file control block beginning at 006CH. The characters in <command tail> are also copied into the default command buffer at 0080H. The length of <command tail> is stored at 0080H, followed by the character string terminated with a binary zero.

If a file has been loaded with the E command, DDT-86 copies the file control block and command buffer from the base page of DDT-86 to the base page of the program loaded. The location of DDT-86's base page can be obtained from the SS register in the user's CPU state when DDT-86 is invoked. The location of the base page of a program loaded with the E command is the value displayed for DS upon completion of the program load.

6.2.8 The L (List) Command

The L command lists the contents of memory in assembly language. The forms are:

```
L
Ls
Ls,f
```

where s is a 20-bit address where the list is to start, and f is a 16-bit offset within the segment specified in s where the list is to finish.

The first form lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s and then lists twelve lines of code. The last form lists disassembled code from s through f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. When DDT-86 regains control from a program being tested (see G, T and U commands), the list address is set to the current value of the CS and IP registers.

Long displays may be aborted by typing any key during the list process. Or, enter ↑S to halt the display temporarily.

The syntax of the assembly language statements produced by the L command is described in Section 4.

6.2.9 The M (Move) Command

The M command moves a block of data values from one area of memory to another. The form is:

```
Ms,f,d
```

where s is the 20-bit starting address of the block to be moved, f is the offset of the final byte to be moved within the segment described by s, and d is the 20-bit address of the first byte of the area to receive the data. If the segment is not specified in d, the same value is used that was used for s. Note that if d is between s and f, part of the block being moved will be overwritten before it is moved, because data is transferred starting from location s.

6.2.10 The R (Read) Command

The R command reads a file into a contiguous block of memory. The form is:

R<filename>

where <filename> is the name and type of the file to be read.

DDT-86 reads the file into memory and displays the start and end addresses of the block of memory occupied by the file. A V command can redisplay this information at a later time. The default display pointer (for subsequent D commands) is set to the start of the block occupied by the file.

The R command does not free any memory previously allocated by another R or E command. Thus a number of files may be read into memory without overlapping. The number of files which may be loaded is limited to seven, which is the number of memory allocations allowed by the BDOS, minus one for DDT-86 itself.

If the file does not exist or there is not enough memory to load the file, DDT-86 issues an error message.

6.2.11 The S (Set) Command

The S command can change the contents of bytes or words of memory. The forms are:

Ss
SWs

where s is the 20-bit address where the change is to occur.

DDT-86 displays the memory address and its current contents on the following line. In response to the first form, the display is:

ssss:oooo bb

and in response to the second form

ssss:oooo wwww

where bb and wwww are the contents of memory in byte and word formats, respectively.

In response to one of the above displays, the operator may choose to alter the memory location or to leave it unchanged. If a valid hexadecimal value is entered, the contents of the byte (or word) in memory is replaced with the value. If no value is entered, the contents of memory are unaffected and the contents of the next address are displayed. In either case, DDT-86 continues to display successive memory addresses and values until either a period or an invalid value is entered.

DDT-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existent RAM at the location indicated.

6.2.12 The T (Trace) Command

The T command traces program execution for 1 to 0FFFFH program steps. The forms are:

```
T
Tn
TS
TSn
```

where n is the number of instructions to execute before returning control to the console.

Before DDT-86 traces an instruction, it displays the current CPU state and the disassembled instruction. In the first two forms, the segment registers are not displayed, which allows the entire CPU state to be displayed on one line. The next two forms are analogous to the first two, except that all the registers are displayed, which forces the disassembled instruction to be displayed on the next line as in the X command.

In all of the forms, control transfers to the program under test at the address indicated by the CS and IP registers. If n is not specified, one instruction is executed. Otherwise DDT-86 executes n instructions, displaying the CPU state before each step. A long trace may be aborted before n steps have been executed by typing any character at the console.

After a T command, the list address used in the L command is set to the address of the next instruction to be executed.

Note that DDT-86 does not trace through a BDOS interrupt instruction, since DDT-86 itself makes BDOS calls and the BDOS is not reentrant. Instead, the entire sequence of instructions from the BDOS interrupt through the return from BDOS is treated as one traced instruction.

6.2.13 The U (Untrace) Command

The U command is identical to the T command except that the CPU state is displayed only before the first instruction is executed, rather than before every step. The forms are:

```
U
Un
US
USn
```

where n is the number of instructions to execute before returning control to the console. The U command may be aborted by striking any key at the console.

6.2.14 The V (Value) Command

The V command displays information about the last file loaded with the E or R commands. The form is:

```
V
```

If the last file was loaded with the E command, the V command displays the start and end addresses of each of the segments contained in the file. If the last file was read with the R command, the V command displays the start and end addresses of the block of memory where the file was read. If neither the R nor E commands have been used, DDT-86 responds to the V command with a question mark, ?.

6.2.15 The W (Write) Command

The W command writes the contents of a contiguous block of memory to disk. The forms are:

```
W<filename>
W<filename>,s,f
```

where <filename> is the filename and file type of the disk file to receive the data, and s and f are the 20-bit first and last addresses of the block to be written. If the segment is not specified in f, DDT-86 uses the same value that was used for s.

If the first form is used, DDT-86 assumes the s and f values from the last file read with an R command. If no file was read with an R command, DDT-86 responds with a question mark, ?. This first form is useful for writing out files after patches have been installed, assuming the overall length of the file is unchanged.

In the second form where s and f are specified as 20-bit addresses, the low four bits of s are ignored. Thus the block being written must always start on a paragraph boundary.

If a file by the name specified in the W command already exists, DDT-86 deletes it before writing a new file.

6.2.16 The X (Examine CPU State) Command

The X command allows the operator to examine and alter the CPU state of the program under test. The forms are:

```
X
Xr
Xf
```

where r is the name of one of the 8086 CPU registers and f is the abbreviation of one of the CPU flags. The first form displays the CPU state in the format:

```
             AX   BX   CX  . . .  SS   ES   IP
--------- xxxx xxxx xxxx . . . xxxx xxxx xxxx
<instruction>
```

The nine hyphens at the beginning of the line indicate the state of the nine CPU flags. Each position may be either a hyphen, indicating that the corresponding flag is not set (0), or a one-character abbreviation of the flag name, indicating that the flag is set (1). The abbreviations of the flag names are shown in Table 2-1. <instruction> is the disassembled instruction at the next location to be executed, which is indicated by the CS and IP registers.

Table 6-2. Flag Name Abbreviations

Character	Name
O	Overflow
D	Direction
I	Interrupt Enable
T	Trap
S	Sign
Z	Zero
A	Auxiliary Carry
P	Parity
C	Carry

The second form allows the operator to alter the registers in the CPU state of the program being tested. The r following the X is the name of one of the 16-bit CPU registers. DDT-86 responds by displaying the name of the register followed by its current value. If a carriage return is typed, the value of the register is not changed. If a valid value is typed, the contents of the register are changed to that value. In either case, the next register is then displayed. This process continues until a period or an invalid value is entered, or the last register is displayed.

The third form allows the operator to alter one of the flags in the CPU state of the program being tested. DDT-86 responds by displaying the name of the flag followed by its current state. If a carriage return is typed, the state of the flag is not changed. If a valid value is typed, the state of the flag is changed to that value. Only one flag may be examined or altered with each Xf command. Set or reset flags by entering a value of 1 or 0.

6.3 Default Segment Values

DDT-86 internally keeps track of the current segment value, making segment specification an optional part of a DDT-86 command. DDT-86 divides the command set into two types of commands, according to which segment a command defaults if no segment value is specified in the command line.

The first type of command pertains to the code segment: A (Assemble), L (List Mnemonics) and W (Write). These commands use the internal type-1 segment value if no segment value is specified in the command.

When invoked, DDT-86 sets the type-1 segment value to 0, and changes it when one of the following actions is taken:

- When a file is loaded by an E command, DDT-86 sets the type-1 segment value to the value of the CS register.
- When a file is read by an R command, DDT-86 sets the type-1 segment value to the base segment where the file was read.
- When an X command changes the value of the CS register, DDT-86 changes the type-1 segment value to the new value of the CS register.
- When DDT-86 regains control from a user program after a G, T or U command, it sets the type-1 segment value to the value of the CS register.
- When a segment value is specified explicitly in an A or L command, DDT-86 sets the type-1 segment value to the segment value specified.

The second type of command pertains to the data segment: D (Display), F (Fill), M (Move) and S (Set). These commands use the internal type-2 segment value if no segment value is specified in the command.

When invoked, DDT-86 sets the type-2 segment value to 0, and changes it when one of the following actions is taken:

- When a file is loaded by an E command, DDT-86 sets the type-2 segment value to the value of the DS register.
- When a file is read by an R command, DDT-86 sets the type-2 segment value to the base segment where the file was read.
- When an X command changes the value of the DS register, DDT-86 changes the type-2 segment value to the new value of the DS register.
- When DDT-86 regains control from a user program after a G, T or U command, it sets the type-2 segment value to the value of the DS register.
- When a segment value is specified explicitly in an D, F, M or S command, DDT-86 sets the type-2 segment value to the segment value specified.

When evaluating programs that use identical values in the CS and DS registers, all DDT-86 commands default to the same segment value unless explicitly overridden.

Note that the G (Go) command does not fall into either group, since it defaults to the CS register.

Table 6-3 summarizes DDT-86's default segment values.

Table 6-3. DDT-86 Default Segment Values

Command	type-1	type-2
A	x	
D		x
E	u	u
F		x
G	u	u
H		
I		
L	x	
M		x
R	u	u
S		x
T	u	u
U	u	u
V		
W	x	
X	u	u

x - use this segment default if none specified; change default if specified explicitly
u - update this segment default

6.4 Assembly Language Syntax for A and L Commands

In general, the syntax of the assembly language statements used in the A and L commands is standard 8086 assembly language. Several minor exceptions are listed below.

- DDT-86 assumes that all numeric values entered are hexadecimal.

- Up to three prefixes (LOCK, repeat, segment override) may appear in one statement, but they all must precede the opcode of the statement. Alternately, a prefix may be entered on a line by itself.

- The distinction between byte and word string instructions is made as follows:

byte	word
LODSB	LODSW
STOSB	STOSW
SCASB	SCASW
MOVSB	MOVSW
CMPSB	CMPSW

- The mnemonics for near and far control transfer instructions are as follows:

short	normal	far
JMPS	JMP	JMPF
	CALL	CALLF
	RET	RETF

- If the operand of a CALLF or JMPF instruction is a 20-bit absolute address, it is entered in the form:

```
ssss:oooo
```

where ssss is the segment and oooo is the offset of the address.

- Operands that could refer to either a byte or word are ambiguous, and must be preceded either by the prefix "BYTE" or "WORD". These prefixes may be abbreviated to "BY" and "WO". For example:

```
INC     BYTE [BP]
NOT     WORD [1234]
```

  Failure to supply a prefix when needed results in an error message.

- Operands which address memory directly are enclosed in square brackets to distinguish them from immediate values. For example:

```
ADD     AX,5    ;add 5 to register AX
ADD     AX,[5]  ;add the contents of location 5 to AX
```

- The forms of register indirect memory operands are:

```
[pointer register]
[index register]
[pointer register + index register]
```

  where the pointer registers are BX and BP, and the index registers are SI and DI. Any of these forms may be preceded by a numeric offset. For example:

```
ADD     BX,[BP+SI]
ADD     BX,3[BP+SI]
ADD     BX,1D47[BP+SI]
```

6.5 DDT-86 Sample Session

In the following sample session, the user interactively debugs a simple sort program. Comments in italic type explain the steps involved.

Source file of program to test.

```
A>type sort.a86
;
;       simple sort program
;
sort:
        mov     si,0            ;initialize index
        mov     bx,offset nlist ;bx = base of list
        mov     sw,0            ;clear switch flag
comp:
        mov     al,[bx+si]      ;get byte from list
        cmp     al,1[bx+si]     ;compare with next byte
        jna     inci            ;don't switch if in order
        xchg    al,1[bx+si]     ;do first part of switch
        mov     [bx+si],al      ;do second part
        mov     sw,1            ;set switch flag
inci:
        inc     si              ;increment index
        cmp     si,count        ;end of list?
        jnz     comp            ;no, keep going
        test    sw,1            ;done - any switches?
        jnz     sort            ;yes, sort some more
done:
        jmp     done            ;get here when list ordered
;
        dseg
        org     100h            ;leave space for base page
;
nlist   db      3,8,4,6,31,6,4,1
count   equ     offset $ - offset nlist
sw      db      0
        end
```

← Fehler --- - 1

Assemble program.

```
A>asm86 sort

CP/M 8086 ASSEMBLER VER 1.1
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY. NUMBER OF ERRORS:   0
```

Type listing file generated by ASM-86.

```
A>type sort.lst
CP/M ASM86 1.1  SOURCE: SORT.A86                                    PAGE   1


                        ;
                        ;       simple sort program
                        ;
                        sort:
 0000 BE0000                    mov     si,0            ;initialize index
 0003 BB0001                    mov     bx,offset nlist ;bx = base of list
 0006 C6060801 00               mov     sw,0            ;clear switch flag
                        comp:
 000B 8A00                      mov     al,[bx+si]      ;get byte from list
 000D 3A4001                    cmp     al,1[bx+si]     ;compare with next byte
 0010 760A                      jna     inci            ;don't switch if in order
 0012 864001                    xchg    al,1[bx+si]     ;do first part of switch
 0015 8800                      mov     [bx+si],al      ;do second part
 0017 C606080101                mov     sw,1            ;set switch flag
                        inci:
 001C 46                        inc     si              ;increment index
 001D 83FE08                    cmp     si,count        ;end of list?
 0020 75E9                      jnz     comp            ;no, keep going
 0022 F606080101                test    sw,1            ;done - any switches?
 0027 75D7                      jnz     sort            ;yes, sort some more
                        done:
 0029 E9FDFF                    jmp     done            ;get here when list ordered
                        ;
                                dseg
                                org     100h            ;leave space for base page
                        ;
```

jump, if not above

```
0100 030804061F06 nlist   db      3,8,4,6,31,6,4,1
     0401
  0008            count   equ     offset $ - offset nlist
0108 00           sw      db      0
                          end


END OF ASSEMBLY. NUMBER OF ERRORS:   0
```

Type symbol table file generated by ASM-86.

```
A>type sort.sym
0000 VARIABLES
0100 NLIST      0108 SW

0000 NUMBERS
0008 COUNT

0000 LABELS
000B COMP       0029 DONE       001C INCI       0000 SORT
```

Type hex file generated by ASM-86.

```
A>type sort.h86
:0400000300000000F9
:1B000081BE0000BB0001C6060801008A003A4001760A8640018800C60608016C
:11001B81014683FE0875E9F60608010175D7E9FDFFEE
:0901008203080406 1F0604010035
:00000001FF
```

Generate CMD file from .H86 file.

```
A>gencmd sort

BYTES READ      0039
RECORDS WRITTEN 04
```

Invoke DDT-86 and load SORT.CMD.

```
A>ddt86 sort
DDT86 1.0
     START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

Display initial register values.

```
-x
           AX   BX   CX   DX   SP   BP   SI   DI   CS   DS   SS   ES   IP
--------- 0000 0000 0000 0000 119E 0000 0000 0000 047D 0480 0491 0480 0000
MOV    SI,0000
```

Disassemble the beginning of the code segment.

```
-l
047D:0000 MOV    SI,0000
047D:0003 MOV    BX,0100
047D:0006 MOV    BYTE [0108],00
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0008
047D:0020 JNZ    000B
```

Display the start of the data segment.

```
-d100,10F
0480:0100 03 08 04 06 1F 06 04 01 00 00 00 00 00 00 00 00 ................
```

```
        Disassemble the rest of the code.
-l
047D:0022 TEST   BYTE [0108],01
047D:0027 JNZ    0000
047D:0029 JMP    0029
047D:002C ADD    [BX+SI],AL
047D:002E ADD    [BX+SI],AL
047D:0030 DAS
047D:0031 ADD    [BX+SI],AL
047D:0033 ??=    6C
047D:0034 POP    ES
047D:0035 ADD    [BX],CL
047D:0037 ADD    [BX+SI],AX
047D:0039 ??=    6F

        Execute program from IP (=0) setting breakpoint at 29H.
-g,29
*047D:0029      Breakpoint encountered.

        Display sorted list.
-d100,10f
0480:0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

        Doesn't look good; reload file.
-esort
     START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F

        Trace 3 instructions.
-t3
            AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P- 0000 0100 0000 0000 119E 0000 0008 0000 0000 MOV    SI,0000
-----Z-P- 0000 0100 0000 0000 119E 0000 0000 0000 0003 MOV    BX,0100
-----Z-P- 0000 0100 0000 0000 119E 0000 0000 0000 0006 MOV    BYTE [0108],00
*047D:000B

        Trace some more.
-t3
            AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P- 0000 0100 0000 0000 119E 0000 0000 0000 000B MOV    AL,[BX+SI]
-----Z-P- 0003 0100 0000 0000 119E 0000 0000 0000 000D CMP    AL,01[BX+SI]
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 0010 JBE    001C
*047D:001C

        Display unsorted list.
-d100,10f
0480:0100 03 08 04 06 1F 06 04 01 00 00 00 00 00 00 00 00 ................

        Display next instructions to be executed.
-l
047D:001C INC    SI
047D:001D CMP    SI,0008
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
047D:0027 JNZ    0000
047D:0029 JMP    0029
047D:002C ADD    [BX+SI],AL
047D:002E ADD    [BX+SI],AL
047D:0030 DAS
047D:0031 ADD    [BX+SI],AL
047D:0033 ??=    6C
047D:0034 POP    ES

        Trace some more.
-t3
            AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 001C INC    SI
--------C 0003 0100 0000 0000 119E 0000 0001 0000 001D CMP    SI,0008
----S-APC 0003 0100 0000 0000 119E 0000 0001 0000 0020 JNZ    000B
*047D:000B
```

3 - 8
Прежде 3 ≤ 8

```
        Display instructions from current IP.
-l
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029

-t3
          AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC 0003 0100 0000 0000 119E 0000 0001 0000 000B MOV     AL,[BX+SI]
----S-APC 0008 0100 0000 0000 119E 0000 0001 0000 000D CMP     AL,01[BX+SI]
--------- 0008 0100 0000 0000 119E 0000 0001 0000 0010 JBE     001C
*047D:0012

-l
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029
047D:002C ADD     [BX+SI],AL
047D:002E ADD     [BX+SI],AL
047D:0030 DAS

        Go until switch has been performed.
-g,20
*047D:0020

        Display list.
-d100,10f
0480:0100 03 04 08 06 1F 06 04 01 01 00 00 00 00 00 00 00 ................

        Looks like 4 and 8 were switched okay.  (And toggle is true.)
-t
          AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC 0004 0100 0000 0000 119E 0000 0002 0000 0020 JNZ     000B
*047D:000B

        Display next instructions.
-l
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029

        Since switch worked, let's reload and check boundary conditions.
-esort
     START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

```
        Make it quicker by setting test length to 3.  Could also have used 547d=1e
-a1d    to patch.)
047D:001D cmp si,3
047D:0020
        Display unsorted list.
-d100
0480:0100 03 08 04 06 1F 06 04 01 00 00 00 00 00 00 00 00 ................
0480:0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0480:0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 20 ..............
        Set breakpoint when first 3 elements of list should be sorted.
-g,29
*047D:0029

-d100,10f       See if list is sorted.
0480:0100 03 04 06 08 1F 06 04 01 00 00 00 00 00 00 00 00 ................

        Interesting, the fourth element seems to have been sorted in.
-esort
     START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F

        Let's try again with some tracing.
-a1d
047D:001D cmp si,3
047D:0020 .

-t9
           AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P- 0006 0100 0000 0000 119E 0000 0003 0000 0000 MOV    SI,0000
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 0003 MOV    BX,0100
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 0006 MOV    BYTE [0108],00
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 000B MOV    AL,[BX+SI]
-----Z-P- 0003 0100 0000 0000 119E 0000 0000 0000 000D CMP    AL,01[BX+SI]
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 0010 JBE    001C
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 001C INC    SI
--------C 0003 0100 0000 0000 119E 0000 0001 0000 001D CMP    SI,0003
----S-A-C 0003 0100 0000 0000 119E 0000 0001 0000 0020 JNZ    000B
*047D:000B


-l
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0003
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
047D:0027 JNZ    0000
047D:0029 JMP    0029


-t3
           AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-A-C 0003 0100 0000 0000 119E 0000 0001 0000 000B MOV    AL,[BX+SI]
----S-A-C 0008 0100 0000 0000 119E 0000 0001 0000 000D CMP    AL,01[BX+SI]
--------- 0008 0100 0000 0000 119E 0000 0001 0000 0010 JBE    001C
*047D:0012


-l
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0003
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
```

```
-t3
          AX   BX   CX   DX   SP   BP   SI   DI   IP
--------- 0008 0100 0000 0000 119E 0000 0001 0000 0012 XCHG   AL,01[BX+SI]
--------- 0004 0100 0000 0000 119E 0000 0001 0000 0015 MOV    [BX+SI],AL
--------- 0004 0100 0000 0000 119E 0000 0001 0000 0017 MOV    BYTE [0108],01
*047D:001C

-d100,10f
0480:0100 03 04 08 06 1F 06 04 01 01 00 00 00 00 00 00 00 ................

        So far, so good.
-t3
          AX   BX   CX   DX   SP   BP   SI   DI   IP
--------- 0004 0100 0000 0000 119E 0000 0001 0000 001C INC    SI
--------- 0004 0100 0000 0000 119E 0000 0002 0000 001D CMP    SI,0003
----S-APC 0004 0100 0000 0000 119E 0000 0002 0000 0020 JNZ    000B
*047D:000B

-l
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0003
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
047D:0027 JNZ    0000
047D:0029 JMP    0029

-t3
          AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC 0004 0100 0000 0000 119E 0000 0002 0000 000B MOV    AL,[BX+SI]
----S-APC 0008 0100 0000 0000 119E 0000 0002 0000 000D CMP    AL,01[BX+SI]
--------- 0008 0100 0000 0000 119E 0000 0002 0000 0010 JBE    001C
*047D:0012

        Sure enough, it's comparing the third and fourth elements of the list.
-esort  Reload program.
    START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F

-l
047D:0000 MOV    SI,0000
047D:0003 MOV    BX,0100
047D:0006 MOV    BYTE [0108],00
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0008
047D:0020 JNZ    000B

        Patch length.
-ald
047D:001D cmp si,7
047D:0020 .
        Try it out.
-g,29
*047D:0029
```

```
        See if list is sorted.
-d100,10f
0480:0100 01 03 04 04 06 06 08 1F 00 00 00 00 00 00 00 00 ................

        Looks better;  let's install patch on test code.  To do this, we
-rsort.cmd              must read CMD file including header, so we use R
  START      END        command.
2000:0000 2000:01FF

        First 80h bytes contain header, so code starts at 80h.
-l80
2000:0080 MOV     SI,0000
2000:0083 MOV     BX,0100
2000:0086 MOV     BYTE [0108],00
2000:008B MOV     AL,[BX+SI]
2000:008D CMP     AL,01[BX+SI]
2000:0090 JBE     009C
2000:0092 XCHG    AL,01[BX+SI]
2000:0095 MOV     [BX+SI],AL
2000:0097 MOV     BYTE [0108],01
2000:009C INC     SI
2000:009D CMP     SI,0008
2000:00A0 JNZ     008B

        Install patch.
-a9d
2000:009D cmp si,7

        Write file back to disk.  (Length of file assumed to be unchanged
-wsort.cmd              since no length specified.)

        Reload file.
-esort

     START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F

        Verify that patch was installed.
-l
047D:0000 MOV     SI,0000
047D:0003 MOV     BX,0100
047D:0006 MOV     BYTE [0108],00
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0007
047D:0020 JNZ     000B

        Run it.
-g,29
*047D:0029

        Still looks good.  Ship it!
-d100,10f
0480:0100 01 03 04 04 06 06 08 1F 00 00 00 00 00 00 00 00 ................
-^C
A>
```

Appendix A
ASM-86 Invocation

Command: ASM86

Syntax: ASM86 <filename> { $ <parameters> }

where

<filename> is the 8086 assembly source file. Drive and extension are optional. The default file extension is .A86.

<parameters> are a one-letter type followed by a one-letter device from the table below.

Parameters:

form: $ Td where T = type and d = device

Table A-1. Parameter Types and Devices

Devices	Parameters				
	A	H	P	S	F
A - P	x	x	x	x	
X		x	x	x	
Y		x	x	x	
Z		x	x	x	
I					x
D					d

x = valid, d = default

Valid Parameters

Except for the F type, the default device is the the current default drive.

Table A-2. Parameter Types

A	controls location of ASSEMBLER source file
H	controls location of HEX file
P	controls location of PRINT file
S	controls location of SYMBOL file
F	controls type of hex output FORMAT

Table A-3. Device Types

A - P	Drives A - P
X	console device
Y	printer device
Z	byte bucket
I	Intel hex format
D	Digital Research hex format

Table A-4. Invocation Examples

ASM86 IO	Assemble file IO.A86, produce IO.HEX IO.LST and IO.SYM.
ASM86 IO.ASM $ AD SZ	Assemble file IO.ASM on device D, produce IO.LST and IO.HEX, no symbol file.
ASM86 IO $ PY SX	Assemble file IO.A86, produce IO.HEX, route listing directly to printer, output symbols on console.
ASM86 IO $ FD	Produce Digital Research hex format.
ASM86 IO $ FI	Produce Intel hex format.

Appendix B
Mnemonic Differences From the Intel Assembler

The CP/M 8086 assembler uses the same instruction mnemonics as the INTEL 8086 assembler except for explicitly specifying far and short jumps, calls and returns. The following table shows the four differences:

Table B-1. Mnemonic Differences

Mnemonic Function	CP/M	INTEL
Intra segment short jump:	JMPS	JMP
Inter segment jump:	JMPF	JMP
Inter segment return:	RETF	RET
Inter segment call:	CALLF	CALL

Appendix C
ASM-86 Hexadecimal Output Format

At the user's option, ASM-86 produces machine code in either Intel or Digital Research hexadecimal format. The Intel format is identical to the format defined by Intel for the 8086. The Digital Research format is nearly identical to the Intel format, but adds segment information to hexadecimal records. Output of either format can be input to GENCMD, but the Digital Research format automatically provides segment identification. A segment is the smallest unit of a program that can be relocated.

Table C-1 defines the sequence and contents of bytes in a hexadecimal record. Each hexadecimal record has one of the four formats shown in Table C-2. An example of a hexadecimal record is shown below.

```
Byte number=>  0 1 2 3 4 5 6 7 8 9 ..............n

  Contents=>   : l l a a a a t t d d d ..... c c CR LF
```

Table C-1. Hexadecimal Record Contents

Byte	Contents	Symbol
0	record mark	:
1-2	record length	l l
3-6	load address	a a a a
7-8	record type	t t
9-(n-1)	data bytes	d d.....d
n-(n+1)	check sum	c c
n+2	carriage return	CR
n+3	line feed	LF

Table C-2. Hexadecimal Record Formats

Record type	Content	Format
00	Data record	: ll aaaa DT <data...> cc
01	End-of-file	: 00 0000 01 FF
02	Extended address mark	: 02 0000 ST ssss cc
03	Start address	: 04 0000 03 ssss iiii cc

ll => record length - number of data bytes
cc => check sum - sum of all record bytes
aaaa => 16 bit address
ssss => 16 bit segment value
iiii => offset value of start address
DT => data record type
ST => segment address record type

It is in the definition of record types 00 and 02 that Digital Research's hexadecimal format differs from Intel's. Intel defines one value each for the data record type and the segment address type. Digital Research identifies each record with the segment that contains it, as shown in Table C-3.

Table C-3. Segment Record Types

Symbol	Intel's Value	Digital's Value	Meaning
DT	00		for data belonging to all 8086 segments
		81H	for data belonging to the CODE segment
		82H	for data belonging to the DATA segment
		83H	for data belonging to the STACK segment
		84H	for data belonging to the EXTRA segment
ST	02		for all segment address records
		85H	for a CODE absolute segment address
		86H	for a DATA segment address
		87H	for a STACK segment address
		88H	for a EXTRA segment address

Appendix D
Reserved Words

Table D-1. Reserved Words

Predefined Numbers				
BYTE	WORD	DWORD		
Operators				
EQ	GE	GT	LE	LT
NE	OR	AND	MOD	NOT
PTR	SEG	SHL	SHR	XOR
LAST	TYPE	LENGTH	OFFSET	
Assembler Directives				
DB	DD	DW	IF	RS
RB	RW	END	ENDM	EQU
ORG	CSEG	DSEG	ESEG	SSEG
EJECT	ENDIF	TITLE	LIST	NOLIST
INCLUDE	SIMFORM	PAGESIZE	CODEMACRO	PAGEWIDTH
Code-macro directives				
DB	DD	DW	DBIT	RELB
RELW	MODRM	SEGFIX	NOSEGFIX	
8086 Registers				
AH	AL	AX	BH	BL
BP	BX	CH	CL	CS
CX	DH	DI	DL	DS
DX	ES	SI	SP	SS
Instruction Mnemonics - See Appendix E.				

Appendix E
ASM-86 Instruction Summary

Table E-1. ASM-86 Instruction Summary

Mnemonic	Description	Section
AAA	ASCII adjust for Addition	4.3
AAD	ASCII adjust for Division	4.3
AAM	ASCII adjust for Multiplication	4.3
AAS	ASCII adjust for Subtraction	4.3
ADC	Add with Carry	4.3
ADD	Add	4.3
AND	And	4.3
CALL	Call (intra segment)	4.5
CALLF	Call (inter segment)	4.5
CBW	Convert Byte to Word	4.3
CLC	Clear Carry	4.6
CLD	Clear Direction	4.6
CLI	Clear Interrupt	4.6
CMC	Complement Carry	4.6
CMP	Compare	4.3
CMPS	Compare Byte or Word (of string)	4.4
CWD	Convert Word to Double Word	4.3
DAA	Decimal Adjust for Addition	4.3
DAS	Decimal Adjust for Subtraction	4.3
DEC	Decrement	4.3
DIV	Divide	4.3
ESC	Escape	4.6
HLT	Halt	4.6
IDIV	Integer Divide	4.3
IMUL	Integer Multiply	4.3
IN	Input Byte or Word	4.2
INC	Increment	4.3
INT	Interrupt	4.5
INTO	Interrupt on Overflow	4.5
IRET	Interrupt Return	4.5
JA	Jump on Above	4.5
JAE	Jump on Above or Equal	4.5
JB	Jump on Below	4.5
JBE	Jump on Below or Equal	4.5
JC	Jump on Carry	4.5
JCXZ	Jump on CX Zero	4.5
JE	Jump on Equal	4.5
JG	Jump on Greater	4.5
JGE	Jump on Greater or Equal	4.5
JL	Jump on Less	4.5
JLE	Jump on Less or Equal	4.5

Table E-1. (continued)

Mnemonic	Description	Section
JMP	Jump (intra segment)	4.5
JMPF	Jump (inter segment)	4.5
JMPS	Jump (8 bit displacement)	4.5
JNA	Jump on Not Above	4.5
JNAE	Jump on Not Above or Equal	4.5
JNB	Jump on Not Below	4.5
JNBE	Jump on Not Below or Equal	4.5
JNC	Jump on Not Carry	4.5
JNE	Jump on Not Equal	4.5
JNG	Jump on Not Greater	4.5
JNGE	Jump on Not Greater or Equal	4.5
JNL	Jump on Not Less	4.5
JNLE	Jump on Not Less or Equal	4.5
JNO	Jump on Not Overflow	4.5
JNP	Jump on Not Parity	4.5
JNS	Jump on Not Sign	4.5
JNZ	Jump on Not Zero	4.5
JO	Jump on Overflow	4.5
JP	Jump on Parity	4.5
JPE	Jump on Parity Even	4.5
JPO	Jump on Parity Odd	4.5
JS	Jump on Sign	4.5
JZ	Jump on Zero	4.5
LAHF	Load AH with Flags	4.2
LDS	Load Pointer into DS	4.2
LEA	Load Effective Address	4.2
LES	Load Pointer into ES	4.2
LOCK	Lock Bus	4.6
LODS	Load Byte or Word (of string)	4.4
LOOP	Loop	4.5
LOOPE	Loop While Equal	4.5
LOOPNE	Loop While Not Equal	4.5
LOOPNZ	Loop While Not Zero	4.5
LOOPZ	Loop While Zero	4.5
MOV	Move	4.2
MOVS	Move Byte or Word (of string)	4.4
MUL	Multiply	4.3
NEG	Negate	4.3
NOT	Not	4.3
OR	Or	4.3
OUT	Output Byte or Word	4.2

Table E-1. (continued)

Mnemonic	Description	Section
POP	Pop	4.2
POPF	Pop Flags	4.2
PUSH	Push	4.2
PUSHF	Push Flags	4.2
RCL	Rotate through Carry Left	4.3
RCR	Rotate through Carry Right	4.3
REP	Repeat	4.4
RET	Return (intra segment)	4.5
RETF	Return (inter segment)	4.5
ROL	Rotate Left	4.3
ROR	Rotate Right	4.3
SAHF	Store AH into Flags	4.2
SAL	Shift Arithmetic Left	4.3
SAR	Shift Arithmetic Right	4.3
SBB	Subtract with Borrow	4.3
SCAS	Scan Byte or Word (of string)	4.4
SHL	Shift Left	4.3
SHR	Shift Right	4.3
STC	Set Carry	4.6
STD	Set Direction	4.6
STI	Set Interrupt	4.6
STOS	Store Byte or Word (of string)	4.4
SUB	Subtract	4.3
TEST	Test	4.3
WAIT	Wait	4.6
XCHG	Exchange	4.2
XLAT	Translate	4.2
XOR	Exclusive Or	4.3

Appendix F
Sample Program

Listing F-1. Sample Program APPF.A86

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output          PAGE
 1


                        title 'Terminal Input/Output'
                        pagesize 50
                        pagewidth 79
                        simform
                        ;
                        ;****** Terminal I/O subroutines ********
                        ;
                        ;       The following subroutines
                        ;       are included:
                        ;
                        ;       CONSTAT   -  console status
                        ;       CONIN     -  console input
                        ;       CONOUT    -  console output
                        ;
                        ;       Each routine requires CONSOLE NUMBER
                        ;       in the BL - register
                        ;
                        ;
                        ;       *****************
                        ;       *  Jump table:  *
                        ;       *****************
                        ;
                        CSEG            ; start of code segment
                        ;
                        jmp_tab:
0000 E90600                     jmp     constat
0003 E91900                     jmp     conin
0006 E92B00                     jmp     conout
                        ;
                        ;
                        ;       ************************
                        ;       *  I/O port numbers    *
                        ;       ************************
```

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output          PAGE
 2


                        ;
                        ;         Terminal 1:
                        ;
   0010                 instat1          equ     10h     ; input status port
   0011                 indata1          equ     11h     ; input port
   0011                 outdata1         equ     11h     ; output port
   0001                 readyinmask1     equ     01h     ; input ready mask
   0002                 readyoutmask1    equ     02h     ; output ready mask
                        ;
                        ;         Terminal 2:
                        ;
   0012                 instat2          equ     12h     ; input status port
   0013                 indata2          equ     13h     ; input port
   0013                 outdata2         equ     13h     ; output port
   0004                 readyinmask2     equ     04h     ; input ready mask
   0008                 readyoutmask2    equ     08h     ; output ready mask
                        ;
                        ;
                        ;       ***********
                        ;       * CONSTAT *
                        ;       ***********
                        ;
                        ;       Entry: BL - reg = terminal no
                        ;       Exit:  AL - reg = 0 if not ready
                        ;                         0ffh if ready
                        ;
                        constat:
0009 53E83F00                   push bx ! call okterminal
                        constat1:
000D 52                         push dx
000E B600                       mov  dh,0                   ; read status port
0010 8A17                       mov  dl,instatustab [BX]
0012 EC                         in   al,dx
0013 224706                     and  al,readyinmasktab [bx]
0016 7402                       jz   constatout
0018 B0FF                       mov  al,0ffh
```

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output          PAGE
 3


                        constatout:
 001A 5A5B0AC0C3                pop dx ! pop bx ! or al,al ! ret
                        ;
                        ;
                        ;       *********
                        ;       * CONIN *
                        ;       *********
                        ;
                        ;       Entry: BL - req = terminal no
                        ;       Exit:  AL - req = read character
                        ;
 001F 53E82900          conin:  push bx ! call okterminal !
 0023 E8E7FF            coninl: call constatl                  ; test status
 0026 74FB                      jz   coninl
 0028 52                        push dx                        ; read character
 0029 B600                      mov  dh,0
 002B 8A5702                    mov  dl,indatatab [BX]
 002E EC                        in   al,dx
 002F 247F                      and  al,7fh                    ; strip parity bit
 0031 5A5BC3                    pop dx ! pop bx ! ret
                        ;
                        ;
                        ;       **********
                        ;       * CONOUT *
                        ;       **********
                        ;
                        ;       Entry:  BL - req = terminal no
                        ;               AL - req = character to print
                        ;
 0034 53E81400          conout: push bx ! call okterminal
 0038 52                        push dx
 0039 50                        push ax
 003A B600                      mov  dh,0                      ; test status
 003C 8A17                      mov  dl,instatustab [BX]
                        conoutl:
 003E EC                        in   al,dx
```

```
003F 224708                 and  al,readvoutmasktab [BX]
0042 74FA                   jz   conoutl
0044 58                     pop  ax                      ; write byte
0045 8A5704                 mov  dl,outdatatab [BX]
0048 EE                     out  dx,al
0049 5A5BC3                 pop dx ! pop bx ! ret
                    ;
                    ;
                    ;       ++++++++++++++
                    ;       + OKTERMINAL +
                    ;       ++++++++++++++
                    ;
                    ;       Entry:  BL - reg = terminal no
                    ;
                    okterminal:
004C 0ADB                   or   bl,bl
004E 740A                   jz   error
0050 80FB03                 cmp  bl,length instatustab + 1
0053 7305                   jae  error
0055 FECB                   dec  bl
0057 B700                   mov  bh,0
0059 C3                     ret
                    ;
005A 5B5BC3         error:  pop bx ! pop bx ! ret      ; do nothing
                    ;
                    ;*************** end of code segment ***************
                    ;
                    ;       ****************
                    ;       * Data segment *
                    ;       ****************
                    ;
                            dseg

                    ;       ***************************
                    ;       * Data for each terminal *
                    ;       ***************************
```

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output          PAGE
 5

                        ;
 0000 1012              instatustab     db      instat1,instat2
 0002 1113              indatatab       db      indata1,indata2
 0004 1113              outdatatab      db      outdata1,outdata2
 0006 0104              readyinmasktab  db      readyinmask1,readyinmask2
 0008 0208              readyoutmasktab db      readyoutmask1,readyoutmask2
                        ;
                        ;*************** end of file **********************
                        end

END OF ASSEMBLY. NUMBER OF ERRORS:    0
```

Appendix G
Code-Macro Definition Syntax

```
<codemacro> ::= CODEMACRO <name> [<formal$list>]
                [<list$of$macro$directives>]
                ENDM

<name> ::= IDENTIFIER

<formal$list> ::= <parameter$descr>[{,<parameter$descr>}]

<parameter$descr> ::= <form$name>:<specifier$letter>
                      <modifier$letter>[(<range>)]

<specifier$letter> ::= A | C | D | E | M | R | S | X

<modifier$letter> ::= b | w | d | sb

<range> ::= <single$range>|<double$range>

<single$range> ::= REGISTER | NUMBERB

<double$range> ::= NUMBERB,NUMBERB | NUMBERB,REGISTER |
                   REGISTER,NUMBERB | REGISTER,REGISTER

<list$of$macro$directives> ::= <macro$directive>
                               {<macro$directive>}

<macro$directive> ::= <db> | <dw> | <dd> | <segfix> |
                      <nosegfix> | <modrm> | <relb> |
                      <relw> | <dbit>

<db> ::= DB NUMBERB | DB <form$name>

<dw> ::= DW NUMBERW | DW <form$name>

<dd> ::= DD <form$name>

<segfix> ::= SEGFIX <form$name>

<nosegfix> ::= NOSEGFIX <form$name>

<modrm> ::= MODRM NUMBER7,<form$name> |
            MODRM <form$name>,<form$name>

<relb> ::= RELB <form$name>

<relw> ::= RELW <form$name>

<dbit> ::= DBIT <field$descr>{,<field$descr>}
```

```
<field$descr> ::= NUMBER15 ( NUMBERB ) |
                  NUMBER15 ( <form$name> ( NUMBERB ) )

<form$name> ::= IDENTIFIER
```

NUMBERB is 8-bits
NUMBERW is 16-bits
NUMBER7 are the values 0, 1,. . , 7
NUMBER15 are the values 0, 1,. . , 15

Appendix H
ASM-86 Error Messages

There are two types of error messages produced by ASM-86: fatal errors and diagnostics. Fatal errors occur when ASM-86 is unable to continue assembling. Diagnostics messages report problems with the syntax and semantics of the program being assembled. The following messages indicate fatal errors encountered by ASM-86 during assembly:

```
NO FILE
DISK FULL
DIRECTORY FULL
DISK READ ERROR
CANNOT CLOSE
SYMBOL TABLE OVERFLOW
PARAMETER ERROR
```

ASM-86 reports semantic and syntax errors by placing a numbered ASCII message in front of the erroneous source line. If there is more than one error in the line, only the first one is reported. Table H-1 summarizes ASM-86 diagnostic error messages.

Table H-1. ASM-86 Diagnostic Error Messages

Number	Meaning
0	ILLEGAL FIRST ITEM
1	MISSING PSEUDO INSTRUCTION
2	ILLEGAL PSEUDO INSTRUCTION
3	DOUBLE DEFINED VARIABLE
4	DOUBLE DEFINED LABEL
5	UNDEFINED INSTRUCTION
6	GARBAGE AT END OF LINE - IGNORED
7	OPERAND(S) MISMATCH INSTRUCTION
8	ILLEGAL INSTRUCTION OPERANDS
9	MISSING INSTRUCTION
10	UNDEFINED ELEMENT OF EXPRESSION
11	ILLEGAL PSEUDO OPERAND
12	NESTED "IF" ILLEGAL - "IF" IGNORED
13	ILLEGAL "IF" OPERAND - "IF" IGNORED
14	NO MATCHING "IF" FOR "ENDIF"
15	SYMBOL ILLEGALLY FORWARD REFERENCED - NEGLECTED
16	DOUBLE DEFINED SYMBOL - TREATED AS UNDEFINED
17	INSTRUCTION NOT IN CODE SEGMENT
18	FILE NAME SYNTAX ERROR
19	NESTED INCLUDE NOT ALLOWED
20	ILLEGAL EXPRESSION ELEMENT
21	MISSING TYPE INFORMATION IN OPERAND(S)
22	LABEL OUT OF RANGE
23	MISSING SEGMENT INFORMATION IN OPERAND
24	ERROR IN CODEMACROBUILDING

Appendix I
DDT-86 Error Messages

Table I-1. DDT-86 Error Messages

Error Message	Meaning
AMBIGUOUS OPERAND	An attempt was made to assemble a command with an ambiguous operand. Precede the operand with the prefix "BYTE" or "WORD".
CANNOT CLOSE	The disk file written by a W command cannot be closed.
DISK READ ERROR	The disk file specified in an R command could not be read properly.
DISK WRITE ERROR	A disk write operation could not be successfully performed during a W command, probably due to a full disk.
INSUFFICIENT MEMORY	There is not enough memory to load the file specified in an R or E command.
MEMORY REQUEST DENIED	A request for memory during an R command could not be fulfilled. Up to eight blocks of memory may be allocated at a given time.
NO FILE	The file specified in an R or E command could not be found on the disk.
NO SPACE	There is no space in the directory for the file being written by a W command.
VERIFY ERROR AT s:o	The value placed in memory by a Fill, Set, Move, or Assemble command could not be read back correctly, indicating bad RAM or attempting to write to ROM or non-existent memory at the indicated location.

Index

Release 2.2 Diskette Contents

1. System Software Diskette

132C.CMD	Utility which can be installed to enable 132 character display.
BOOTCOPY.CMD	Utility for copying boot tracks. (Version 1.6) Same as 2.1 release program.
DCOPY.CMD	Utility for copying complete diskettes. (Ver 2.6) Double-sided diskette copy implemented, plus some "cosmetic" changes.
FORMAT.CMD	Utility for formatting diskettes. (Version 2.9) Double-sided diskette format implemented, plus some "cosmetic" changes.
FIXLABEL.CMD	Utility for writing a disk label. (Version 1.1) Same as 2.1 release program.
UDCCALC.CMD	Utility which can be installed to enable the calculator functions.

2. System Configuration Diskette

Starting with this release, since it is fundemantally an internal release, this diskette is a copy of the "work" diskette used for system generation. Thus it includes more files than will appear on the external system configuration diskette. Specifically, it includes both Victor and Sirius tables, some prereleased programs (KEYGEN & CEDIT), and the control files used for system generation (BRITISH, FRENCH, GERMAN, SIRGEN, SIRSYS, VICGEN and VICSYS). The generation files are useful in two ways; (1) used as input to SYSELECT "Modify an existing system" option one can see the exact configuration of the generated system, and (2) used as input to MAKESYS and BOOTCOPY a system can be generated without needing to run SYSELECT.

ALLOC.CMD	Initializing program for CEDIT.BAS
ASM86.CMD	Digital Research Assembler.
BOOTCOPY.CMD	See above.
BRITISH.*	Output files from SYSELECT for British system.
CEDIT.BAS	PBASIC86 utility for manipulating character set files. This is a temporary program and for internal use only.
CHARGEN.SUB	Submit file used to invoke CEDIT.

COLD1/5.CMD	Portion of the operating system used to do initialization.
CPM16-06.CMD	Digital Research CP/M86 + Sirius BIOS.
DCOPY.CMD	See above.
DDT86.CMD	Digital Research Debug utility.
DEFAULT.SKL	Default skeleton used by SYSELECT.
E1/5.HDR	Header file, used in MAKESYS.
ED.CMD	Digital Research Editor.
FIXLABEL.CMD	See above.
FORMAT.CMD	See above.
FRENCH.*	Output files from SYSELECT for French system.
GENCMD.CMD	Digital Research Command generation utility.
GERMAN.*	Output files from SYSELECT for German system.
KEYGEN.BAS	Utility to update and display keyboard table entries. (This is a pre-release of the program.)
MAKESYS.CMD	Utility for collecting selected files and creating a .SYS file. (Version 1.3) Same as in 2.1 release.
PBASIC86.CMD	Microsoft runtime only BASIC86 Interpreter.
PIP.CMD	Digital Research utility.
SIRGEN.*	Output files from SYSELECT for system used for the Sirius system configuration diskette.
SIRSYS.*	Output files from SYSELECT for system used for the Sirius system utilities diskette.
SYSELEC1.CMD	Part of SYSELECT. Collects directory data.
SYSELECT.BAS	Menu driven system selection utility. (Ver 0.26)
SYSELECT.CMD	Driver program for system configuration.
SYSELECT.HLP	Help file, used by SYSELECT.BAS
VICGEN.*	Output files from SYSELECT for system used for the Victor system generation diskette.

VICSYS.*	Output files from SYSELECT for system used for the Victor system utilities diskette.
*.BAN	Banner files: SIRIUS for Sirius 1 (English), with a logo. SIRFRENC for Sirius 1 (French), with a logo. SIRGERM for Sirius 1 (German), with a logo. VICTOR for Victor 9000, with a logo. VICNOLGO for Victor 9000, without a logo.
*.CHR	Display character tables. GERM01 changed in this release: the cross hatch (#) replaced the British monetary symbol at character location Hex 23.
*.KB	Keyboard table files. Changed in this release: FRENCH01 key 12 unshifted: British monetary symbol key 12 shifted: Dollar sign. key 16 unshifted: Apostrophe (Hex 27). GERM01 key 12 unshifted: Circumflex (Hex 5e). key 12 shifted: Grave accent (Hex 60). key 24 unshifted: Cross hatch (Hex 23). key 24 shifted: Apostrophe (Hex 27).
*.LGO	Logo character sets.
*.XLT	Translation tables.

3&4. BIOS Listing Diskettes

Contains the listing files for the BIOS.

5. Utility Source and Documentation Diskette

This diskette contains source code, object code and user notes for the utility programs, as well as other documents related to this release.

*.USE files are the user notes.
*.A86 are assembly source.
*.CMD are assembly object, in command form.
*.SUB are SUBMIT files for generating the CMD files from A86
*.BSC are BASIC86 unprotected code.
*.BAS are BASIC86 protected.
*.DOC are miscellaneous documentation files.

Specifically the files on the diskette are:

132C.*	Files for 132 character display utility
ALLOC.*	Files for ALLOC.CMD, initializer for CEDIT.
ASM86.CMD	Digital Research Assembler.
BOOTCOPY.*	Files for bootcopy utility.
CEDIT.*	Files for temporary character editor.
DBLSIDE.DOC	Notes about double-sided disk use.
DCOPY.*	Disk copy utility files.
DISK.LIB & DUTIL.LIB	Include files for DCOPY & FORMAT.
DISPLAY.DOC	A corrected Display Driver Specification.
FIXLABEL.*	Fix label utility files.
FORMAT.*	Format files.
GENCMD.CMD	Digital Research utility.
KEYGEN.*	Key table editor files.
MAKESYS.*	Makesys files.
PIP.CMD	Digital Research utility.
RELEASE.DOC	This document.
SYSELEC?.*	System selection programs.
UDCCALC.*	Calculator files.
UDC.A86 & VCALC.A86	Files included by UDCCALC.

Double-sided Drive Note

Release 2.2 of the CP/M86 operating system can utilise double-sided floppy drives. If you are using a machine with double-sided drives, you can create diskettes which have roughly twice the storage capacity that was possible with earlier releases of the operating system.

A WORD OF CAUTION ABOUT USING DOUBLE-SIDED DISKETTES

The great advantage in using double-sided diskettes is that the amount of data stored on the diskette is almost twice that of a standard single-sided diskette: 1.2 megabytes compared to 610 kilobytes. A single-sided diskette can be used on a computer with either single- or double-sided drives. However, once a diskette is formatted as double-sided, **NONE** of the data on it can be accessed by a computer which has single-sided drives.

If you have a computer which has double-sided drives, you may not want to convert all of your diskettes into double-sided diskettes right away, since a majority of the computers currently in use are only single-sided and you will not be able to use your diskettes on them.

CREATING A DOUBLE-SIDED DISKETTE

You can only create a double-sided diskette on a computer which has double-sided floppy drives. To create a double-sided diskette, you need to use Version 2.8 of the FORMAT program. If your computer has double-sided drives, the FORMAT program asks the following question on the 25th line of the screen:

Format both sides of the diskette (y/n) ?

If you answer by typing a 'Y', the program creates a double-sided diskette. If you answer with a 'N', just the first side of the diskette is formatted and you have a single-sided diskette which can be used on both single- and double-sided machines.

Remember that you can only create double-sided diskettes on a computer with double-sided floppy drives.

COPYING A DOUBLE-SIDED DISKETTE

To make a copy of a double-sided diskette, you must use Version 2.5, or later, of the DCOPY program. You do not have to tell the DCOPY program whether or not the diskette to be copied is double-sided: the program automatically knows. To copy a double-sided diskette, you must have double-sided drives in your computer. If you try to DCOPY a double-sided diskette on a computer which has only single-sided drives, DCOPY prints an error message and aborts the copy.

CONVERTING SINGLE- AND DOUBLE-SIDED DISKETTES

If you have a computer that has double-sided floppy drives, you may want to convert some of your single-sided diskettes into double-sided diskettes (see the caution above). To do this, create a double-sided diskette by using the FORMAT program as described above. Then use PIP to copy the desired files from the single-sided diskette to the double-sided diskette you just created.

At times you may need to convert a double-sided diskette back into a single-sided diskette. Using the FORMAT program, create a single-sided diskette. Then use PIP to copy the desired files from the double-sided diskette to the single-sided diskette. You may need two or more diskettes to hold all of the data from a double-sided diskette.

Display Driver Specification

A2.1 Introduction

The software in the Sirius developed BIOS display interface receives ASCII characters and either displays them or uses them for display control. Characters in the ASCII control set (hex 00 through hex 1F and hex 7F) are used as control characters and are not displayed (see "ESC 8" exception below.) However, they may affect the display. Most of the control characters act by themselves. However, for some action more than one character is required to specify the action. This is accomplished by using the ASCII escape code (ESC--hex 1B) followed by one or more characters. The control characters and the escape sequences are described below.

A2.2 Control Characters

Bell (ctrl G, hex 07)	This is not really a display control character. It sends to the CODEC a series of signals to make the sound of a bell.
Backspace (ctrl H, hex 08)	Positions the cursor back one column. If at column 1, then position cursor at column 80 of previous row; unless at columnm 1, row 1 in which case position to column 80, row 1.
Horizontal Tab (ctrl I, hex 09)	Positions the cursor on the next tab stop. Tab stops are fixed and are at columns 9, 17, 25, 33, 41, 49, 57, 65, and 72 through 80. If the cursor is at column 80, it remains there.
Line Feed (ctrl J, hex 0A)	Positions the cursor down one row. If at row 24, then scroll display up 1 row. (May also be treated as a carriage return -- see ESC x9.)
Carriage Return (ctrl M, hex 0D)	Positions the cursor at column 1 of the current row. (May also be treated as a line feed -- see ESC x8.)
Shift In (ctrl N, hex 0E)	Shift to display character set 1 (G1).
Shift Out (ctrl O, hex 0F)	Shift to display character set 0 (G0).

A3.2 Escape Sequences

Escape Sequence/Function	ASCII Code Generated (Hexadecimal)	Sequence Definition
CURSOR FUNCTIONS		
Esc A	1B, 41	Moves the cursor up one line
Esc B	1B, 42	Moves the cursor down one line without changing columns.
Esc C	1B, 43	Moves the cursor forward one character position.
Esc D	1B, 44	Moves the cursor backward one character position.
Esc H	1B, 48	Sets the cursor at home position.
Esc I	1B, 49	Moves the cursor to the same horizontal position on the preceding line.
Esc n	1B, 6E	Reports the cursor position.
Esc j	1B, 6A	The display driver saves the cursor position.
Esc k	1B, 6B	Returns the cursor to the previously saved cursor position.
Esc Y[l][c]	1B, 59	Moves the cursor via direct cursor addressing, where 'l' represents the hexadecimal line number and 'c' represents the hexadecimal column number. The first line and the left column are both 20 (hex) (the smallest value of the printing characters) and increase from there. Since the lines are numbered from 1 to 19 (hex) (from top to bottom) and the columns from 1 to 50 (hex) (from left to right), you must add the proper line and column numbers to 1F. No movement is done, if l and/or c are invalid.

Escape Sequence/Function	ASCII Code Generated (Hexadecimal)	Sequence Definition
EDITING FUNCTIONS		
Esc E	1B, 45	Erases the entire screen.
Esc b	1B, 62	Erases from the start of the screen up to and including the cursor position.
Esc J	1B, 4A	Erases from the cursor position to the end of the page.
Esc l	1B, 6C	Erases entire line.
Esc o	1B, 6F	Erases the beginning of the line up to and including the cursor position.
Esc K	1B, 4B	Erases from the cursor position to the end of the line.
Esc L	1B, 4C	Inserts a blank line; the current line and all following lines are moved down one line. The cursor is moved to the beginning of the blank line.
Esc M	1B, 4D	Deletes the current line, placing the cursor at the start of the line, and moves all following lines up one line. A blank line is inserted at line 24.
Esc N	1B, 4E	Deletes cursor-position character and shifts the rest of the line one character position to the left.
Esc @	1B, 40	Enters the insert character mode, allowing insert into text on the screen. As each new character is inserted, the character at the end of the line is lost.
Esc O	1B, 4F	Exits from the insert character mode.

Escape Sequence/Function	ASCII Code Generated (Hexadecimal)	Sequence Definition
CONFIGURATION FUNCTIONS		
Esc x[Ps]	1B, 78	Sets mode(s) as follows: Ps — Mode 1 — Enable 25th line 3 — Hold Screen mode on 4 — Block cursor 5 — Cursor off 8 — Auto line feed on receipt of a carriage return 9 — Auto carriage return on receipt of a line feed A — Increase audio volume B — Increase CRT brightness C — Increase CRT contrast
Esc y[Ps]	1B, 79	Resets mode(s) as follows: Ps — Mode 1 — Disable 25th line 3 — Hold screen mode off 4 — Underscore cursor 5 — Cursor on 8 — No auto line feed 9 — No auto carriage return A — Decrease audio volume B — Decrease CRT brightness C — Decrease CRT contrast
Esc ^	1B, 5E	Toggle hold mode
Esc [	1B, 5B	Set hold mode
Esc \	1B, 5C	Clear hold mode
Esc |	1B, 7C	Activate User defined console (T.B.A.)
OPERATION MODE FUNCTIONS		
Esc p	1B, 70	Enters the reverse video mode.
Esc q	1B, 71	Exits the reverse video mode.

Escape Sequence/Function	ASCII Code Generated (Hexadecimal)	Sequence Definition
SPECIAL FUNCTIONS		
Esc }	1B, 7D	Disables the keyboard.
Esc {	1B, 7B	Enables the keyboard.
Esc v	1B, 76	Enables wrap around at the end of the line.
Esc w	1B, 77	Disables wrap around at the end of the line.
Esc z	1B, 7A	Resets terminal to power-on configuration.
Esc $	1B, 24	Transmits the character at the cursor.
Esc]	1B, 5D	Transmits the 25th line.
Esc #	1B, 23	Transmits the page.
Esc (	1B, 28	Sets high intensity.
Esc)	1B, 29	Sets low intensity.
Esc +	1B, 2B	Clears the foreground. (High intensity displayed characters)
Esc Z	1B, 5A	Identifies display as emulating VT52 (the terminal responds with an ESC \ K).
Esc 0	1B, 30	Sets the underline mode.
Esc 1	1B, 31	Resets the underline mode.
Esc 2	1B, 32	Enables cursor blink.
Esc 3	1B, 33	Disables cursor blink.
Esc 8	1B, 38	Sets the test (literally) mode for the next single character.

(Comparison Summary Continued)

		Sirius/ Victor	DEC VT52	Heathkit H19
ESC q	Reverse OFF	x		x
ESC (	High Intensity ON	x		
ESC)	High Intensity OFF	x		
ESC 0	Underline ON	x		
ESC 1	Underline OFF	x		
ESC 2	Blink Cursor ON	x		
ESC 3	Blink Cursor OFF	x		
ESC 4	Set Key Function	x		
ESC i	Display system information	x		
ESC z	Reset	x		x
ESC Z	Identify as VT52	x	x	x
ESC	Enter ASCII mode			x
ESC t	Enter keypad shift mode			x
ESC u	Exit keypad shift mode			x

Set Reset Modes

ESC x/y	1	25th line	x		x
ESC x/y	2	Key Click			x
ESC x/y	3	Hold Screen	x		x
ESC x/y	4	Block Cursor	x		x
ESC x/y	5	Cursor Display	x		x
ESC x/y	6	Kpd Shifted			x
ESC x/y	7	Alt Keypad			x
ESC x/y	8	Auto LF	x		x
ESC x/y	9	Auto CR	x		x
ESC x/y	A	Volume Increase/Decrease	x		
ESC x/y	B	Brightness " "	x		
ESC x/y	C	Contrast " "	x		

Escape Sequence/Function	ASCII Code Generated (Hexadecimal)	Sequence Definition
Esc i[n]	1B, 69	Displays the system diskette sign-on banner, as follows: (n represents the ASCII numeric character)

n	Display
0	The entire banner.
1	The company logo only.
2	The product name only.
3	Configuration information only.

A4.2 Comparison Summary

Cursor Functions	Sirius/ Victor	DEC VT52	Heathkit H19
ESC A Cusor up	x	x	x
ESC B Cusor down	x	x	x
ESC C Cusor forward	x	x	x
ESC D (BS 08H) Cusor back	x	x	x
ESC H Home	x	x	x
TAB (09H) Tab	x	x	x
LF (0AH) Line Feed	x	x	x
CR (0DH) Carriage Return	x	x	x
ESC I RVS Line Feed	x	x	x
ESC Y Address Cursor	x	x	x
ESC n Report Cursor	x		x
ESC k Restre Cursor	x		x
ESC j Save Cursor	x		x
Editing Functions			
ESC J Erase EOS	x	x	x
ESC K Erase EOL	x	x	x
ESC b Erase SOS	x		x
ESC l Erase Line	x		x
ESC o Erase SOL	x		x
ESC E Erase Screen	x		x
ESC L Insert Line	x		x
ESC M Delete Line	x		x
ESC @ Insert Mode	x		x
ESC O Exit Insert Mode	x		x
ESC N Delete Char	x		x
Special Functions			
ESC F Graphics Mode On	x	x	x
ESC G Graphics Mode Off	x	x	x
ESC = ALT. Keypad on		x	x
ESC > ALT. Keypad off		x	x
ESC $ Hold mode on	x	x	x
ESC / Hold mode off	x	x	x
ESC [Transmit 25th line	x		x
ESC # Transmit Page	x		x
Bell (07H)	x	x	x
(13H) (11H) Xon Xoff protocol		x	x
(00H) (7FH) NUL DEL Ignored	x	x	x
ESC { Enable Keyboard	x		x
ESC } Disable	x		x
ESC + Clear Foreground	x		
ESC 8 Test Mode	x		
ESC v Enable Wrap	x		x
ESC w Disable Wrap	x		x
ESC p Reverse ON	x		x

132 COLUMN SPECIFICATION

PURPOSE:

The 132 column module provides a simulated 132 column display in the 800 dot by 400 line HIRES display mode of the SIRIUS computer.

RESULT:

The normal VT52 print interface recognizes an escape sequence to enable the 132 column mode. The characters are displayed in a 5 by 7 dot matrix in a 6 by 10 cell to give the 132 column display. A standard display of 80 columns by 25 lines is also simulated with an 8 by 11 dot matrix in a 10 by 16 cell.

INSTALLATION:

The 132C program installs itself into the BIOS through a super BIOS call. The program requires approximately 50K of memory. The 800 by 400 HIRES screen requires 40K, the character sets and code are the rest. The 132c program copies itself into the lowest 64K block of memory and removes the 50K required bytes from the system permanently.

CONTROL CODES:

BELL - CTRL G (07H)
Generate a bell sound. Pass thru to VT52

BACKSPACE - CTRL H (08H)
Positions the cursor back one column. If wrap around mode is enabled and the cursor was at column 1, then position cursor at last column of previous row; unless at column 1, row 1 in which case position to last column in row 1. If in discard mode, then the cursor will not move from column 1.

HORIZONTAL TAB - CTRL I (09H)
Positions cursor the cursor forward to the next tab stop. ab stops are fixed and are at columns 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105, 113, 121, and 129. If the cursor is at the last column it remains there.

LINE FEED - CTRL J (0AH)
Moves cursor to next line same horizontal position, scrolls the screen up if a line feed occures on the bottom line. If the cursor is on bottom line + 1 then no action is taken.

RETURN - CTRL M (0DH)
Moves cursor to left most column of same line.

SHIFT OUT - CTRL O (0EH)
Switch the character cell size to 6 by 10 resulting in a display of 133 columns by 40 lines. The top of screen will be set to line 1 and the bottom of screen will be set to line 39. The bottom line + 1 will be line 40 similar to VT52's 25th line. The cursor will home.

SHIFT IN - CTRL N (0EH)
Switch the character cell size to 10 by 16 resulting in a display of 80 columns by 25 lines. The top of screen will be set to line 1 and the bottom of screen will be set to line 24. The cursor will home.

CANCEL _ CTRL X (18H)
Abort any Escape sequence in progress. Starts displaying characters as normal ASCII.

ESCAPE - CTRL [(1BH)
Start an escape sequence.

ESCAPE SEQUENCES:

TRANSMIT PAGE - ESC # (1BH,23H)
Will transmit only a RETURN (ODH) LINE FEED (OAH).

TRANSMIT CHARACTER AT CURSOR - ESC $ (1BH,24H)
Will transmit only a RETURN (ODH) LINE FEED (OAH).

SET HIGH INTENSITY - ESC ((1BH,28H)
Simulates high intensity mode by shadow printing.

SET LOW INTENSITY - ESC) (1BH,29H)
Prints normal characters.

ENTER UNDERLINE CHARACTER MODE - ESC 0 (1BH,30H)
Sets the underline mode.

EXIT UNDERLINE CHARACTER MODE - ESC 1 (1BH,31H)
Resets the underline mode.

SET KEY VALUE - ESC 4[n][lk][kc] (1BH,34H,XXH,XXH,XXH)
Set key value. Five characters are passed thru to VT52 to set new key values.

LITERAL CHARACTER - ESC 8 (1BH,38H)
Display next character literally.

ENTER INSERT CHARACTER MODE - ESC @ (1BH,40H)
Enters insert character mode, allowing insert into text on the screen. As you type in new characters, existing text to the right of the cursor shifts to the right. As each new character is inserted, the character at the end of the line is lost.

CURSOR UP - ESC A (1BH,41H)
Moves cursor up one line without changing columns. If the cursor reaches the top line, it remains there and no scrolling occurs. No action taken on bottom line + 1.

CURSOR DOWN - ESC B (1BH,42H)
Moves cursor down one line without changing columns. The cursor will not move past the bottom line and no scrolling will take place. No action taken on bottom line + 1.

CURSOR FORWARD - ESC C (1BH,43H)
Moves cursor one character position to the right. If the cursor is at the right end of the line, it will remain there.

ESCAPE SEQUENCES:

CURSOR BACKWARD - ESC D (1BH,44H)

Moves the cursor one character position to the left. If the cursor is at the start (left end) of a line, it will remain there.

CLEAR DISPLAY - ESC E (1BH,45H)

Erase the screen from the defined top line to the defined bottom line. If on bottom line + 1 the erases bottom line + 1 only. Places the cursor in the home position.

ENTER GRAPHICS MODE - ESC F (1BH,46H)

VT52 graphics characters appear in character numbers 94 to 127 of the ASCII character set.

EXIT GRAPHICS MODE - ESC G (1BH,47H)

Normal lower case characters appear in character numbers 94 to 127.

CURSOR HOME - ESC H (1BH,48H)

Moves the cursor to the first character position on the defined top line.

REVERSE LINE FEED - ESC I (1BH,49H)

Moves the cursor to the same horizontal position on the preceding line. If the cursor is on the defined top screen line, a scroll down is performed. No action taken if on bottom line + 1.

ERASE TO END OF PAGE - ESC J (1BH,4AH)

Erases all the information from the cursor (including the cursor position) to the end of the defined bottom line. If on bottom line + 1 then erase to end of line only.

ERASE TO END OF LINE - ESC K (1BH,4BH)

Erases from the cursor (including the cursor position) to the end of the line.

INSERT LINE - ESC L (1BH,4CH)

Inserts a new blank line by moving the line that the cursor is on, and all the following lines, down one line, to the defined bottom line. Then the cursor is moved to the beginning of of the new blank line. No action taken if on bottom line + 1.

DELETE LINE - ESC M (1BH,4DH)

Deletes the contents of the line that the cursor is on, places the cursor at the beginning of the line, moves all the following lines up one line, and adds a blank line at the defined bottom line. No action taken if on bottom line + 1.

ESCAPE SEQUENCES:

DELETE CHARACTER - ESC N (1BH,4EH)

Deletes the character at the cursor position and shifts any existing text that is to the right of the cursor one character position to the left.

EXIT INSERT CHARACTER MODE - ESC O (1BH,4FH)

Exits from insert character mode.

DIRECT CURSOR ADDRESSING - ESC Y[l#][c#] (1BH,59H,XXH,XXH)

Moves the cursor to a position on the screen by entering the escape code, the character which represents the line number, and the character which represents the column number. The 'l#' represents the hexadecimal line number and 'c#' represents the hexadecimal column number. The first line and the left column are both 20 (hex) (the smallest value of the printing characters) and increase from there. Since the lines are numbered from 1 up (from top to bottom) and the columns from 1 up (from left to right), You must add the proper line and column numbers to 1F (hex). If the line number entered is smaller than the defined top line, the cursor will be positioned to the top line. If the line number entered is greater than the defined bottom line + 1, the cursor will not more from it present line. If the column number is too high, the cursor will move to the end of the line.

IDENTIFY AS VT52 - ESC Z (1BH,5AH)

CONIN responds with "ESC/K" to indicate that it can perform as VT52.

TRANSMIT BOTTOM LINE - ESC] (1BH,5DH)

Will transmit only a RETURN (0DH) LINE FEED (0AH).

TOGGLE DEBUG MODE - ESC _ (1BH,5FH)

Toggles debug mode on or off. In debug mode the bottom line - 1 displays the hex codes for print stream.

ERASE BEGINNING OF DISPLAY - ESC b (1BH,62H)

Erases from the start of the screen to the cursor, and includes the cursor position.

REVERSE TAB - ESC h (1BH,68H)

Moves cursor left to next mod 8 position. Stops on left side of screen.

ESCAPE SEQUENCES:

SAVE CURSOR POSITION - ESC j (1BH,6AH)

The present cursor position is saved so the cursor can be returned here later when given the set cursor to saved position command.

SET CURSOR TO SAVED POSITION - ESC k (1BH,6BH)

Returns the cursor to the position where it was when it recieved the save cursor position command. If the line number restored is smaller than the defined top line, the cursor will be positioned to the top line. If the line number restored is greater than the defined bottom line + 1, the cursor will not more from it present line. If the column number is too high, the cursor will move to the end of the line.

ERASE ENTIRE LINE - ESC l (1BH,6CH)

Erases all the line including the cursor position.

SET SIZE - ESC m[c1][c2][c3] (1BH,6DH,XXH,XXH,XXH,)

If c1 is ASCII "1" then set cell width and height. The c2 character equals width+1FH and the c3 character equals height+1FH. Width and heigth can range from 1 to 16. The maximum number of of character columns and lines are determined by: max columns = 800/width, max lines = 400/height. After set cell size the top line is set to 1 and the bottom line is set to max lines - 1. The cursor will home after the cell size is set.

If c1 is ASCII "2" then set screen top and bottom. The c2 character equals top line+1FH and the c3 character equals bottom line+1FH. Top line and bottom line can range from 1 to max lines with bottom line always greater than or equal to top line. The cursor will home after the screen size is set.

CURSOR POSITION REPORT - ESC n (1BH,6EH,XXH,XXH)

CONIN reports the cursor position in the form of ESC Y line# olumn#.

ERASE BEGINNING OF LINE - ESC o (1BH,6FH)

Erases from the beginning of the line to the cursor, and includes the cursor position.

ENTER REVERSE VIDEO MODE - ESC p (1BH,70H)

Enters the reverse video mode so that characters are displayed as black characters on a white background.

EXIT REVERSE VIDEO MODE - ESC q (1BH,71H)

Exits the reverse video mode.

ESCAPE SEQUENCES:

WRAP AROUND AT END OF LINE - ESC v (1BH,76H)

A print to the last column of the line will position to the first column of the next line. The page scrolls up if necessary.

DISCARD AT END OF LINE - ESC w (1BH,77H)

A print to the last column of the line will not change the cursor position and overprinting occures. Therefore, only the last character received will be displayed in the last column position.

SET MODE(S) - ESC x [Ps] (1BH,78H,XXH)

Sets the following modes, where Ps equals:

1 = Enable bottom line
4 = Block cursor
5 = Cursor off
8 = Auto line feed on receipt of CR
9 = Auto CR on receipt of line feed
A = Send to VT52
B = Send to VT52
C = Send to VT52

RESET MODE(S) - ESC y [Ps] (1BH,79H,XXH)

Resets the following modes, where Ps equals:

1 = Disable bottom line
4 = Underscore cursor (ok if character height > 9)
5 = Cursor on
8 = No auto line feed
9 = No auto CR
A = Send to VT52
B = Send to VT52
C = Send to VT52

RESET TO POWER-UP CONFIGURATION - ESC z (1BH,7AH)

Reset back to 80 column mode.

KEYBOARD ENABLED - ESC { (1BH,7BH)

Enables the keyboard after it was inhibited by a keyboard disable command. Pass thru to VT52.

ENABLE 132 COLUMN DISPLAY - ESC ¦ (1BH,7CH)

Enable 132 column mode.

KEYBOARD DISABLE - ESC } (1BH,7DH)

Inhibits the output of the keyboard. Pass thru to VT52.

USING THE CALCULATOR KEYS

The keyboard unit includes a calculator-key group. The keys in this group are arranged like those on a simple hand-held calculator,and can be used to perform the four basic arithmetic functions: addition, subtraction, multiplication, and division.

Once the calculator is initiated the calculator functions are available to you at all times. For example, if you are using a word processing application program to prepare a memo, you can interrupt your work to add two numbers and then return to the memo exactly where you left off.

To initiate the calculator, run UDCCALC. This installs the calculator as part of your system. However, this uses about 5000 bytes of memory which can only be recovered by rebooting the system.

Turning the Calculator Functions On and Off

This section describes -

- Using the CALC key to turn the functions on and off.
- Using the ENTER key to turnthe functionsoff.

When the CALC key is used with the Shift key, as described in the next two sections, it is a **toggle** key: pressing it once turns the calculator functions on, and pressing it again turns them off. You can also turn the calculator functions off with the ENTER key. The two ways for turning off the functions have different results, as described later, in "Turning off the Calculator Functions."

TURNING ON THE CALCULATOR FUNCTIONS

Turn on the calculator functions by typing a shifted CALC (press he CALC key while holding down the Shift key). The **calculator-function banner** and **calculator field** will appear in the last line at the bottom of the screen. If that line contained characters when you turned on the calculator functions, they are temporarily replaced with the banner and field.

TURNING ON THE CALCULATOR FUNCTIONS

As indicated earlier, you can turn off the calculator functions either by typing a shifted CALC or by pressing the ENTER key. Both methods erase the calculator banner and field and return you to your work exactly where you left it when you turned on the calculator functions. When you use the ENTER key, the number in the calculator field is placed into your work, as if you had typed it at the keyboard instead of using the calculator functions.

Performing Arithmetic Functions

UDCCALC operates very much like many hand-held calculators. When you perform arithmetic functions, UDCCALC displays each step in the calculator field. Only one number is displayed at a time; the field accepts up to 14 digits. Each time you type a new number, the act of typing its first digit erases the previous number.

To complete a calculation, press the CALC key (without the Shift key) or the = (equal) key. The result of the calculation (for example, the sum in an addition) replaces the previous number.

The four arithmetic functions described are performed with the four **arithmetic-function keys**: the add key (+), the subtract key (-), the multiply key (X), and the divide key (÷).

Each example in the next four sections shows a series of one-line displays that would appear in the calculator field for an arithmetic function. The examples include a note beside each line to explain the steps taken to perform the functions.

ADDITION

To add numbers, type the numbers, pressing the add key after each one. (Pressing the add key gives you the sum of all the numbers typed so far.) Then press the CALC key.

For example, to add 5, 7, and 4:

5	You typed a 5.
5+	You typed a +.
7	You typed a 7.
12+	You typed a +.UDCCALC added 5 and 7.
4	You typed a 4.
16+	You typed a +. UDCCALC added 12 and 4.
16	You pressed the CALC key. UDCCALC displayed the total, and prepared to perform another series of calculations.

SUBTRACTION

To subtract one number from another, type the number that will be subtracted from, then press the plus key. Next, type the number being subtracted and press the subtract key. Then press the CALC key.

For example, to subtract 3 from 9:

9	You typed a 9.
9+	You typed a +.
3	You typed a 3.
6-	You typed a -. UDCCALC subtracted 3 from 9.

MULTIPLICATION

To multiply two or more numbers, type the numbers, pressing the multiply key after each one. Then press the CALC key.

For example, to multiply 10 by 8:

10	You typed a 10.
10	You typed a x.
8	You typed an 8.
80	You pressed the CALC key. UDCCALC displayed the product and prepared to perform another series of calculations.

DIVISION

To divide one number into another, first type the number to be divided and press the divide key. Next, type the number you want to divide by. Then press the CALC key.

For example, to divide 45 by 5:

45	You typed 45.
45	You typed ÷.
5	You typed 5.
9	You pressed the CALC key. UDCCALC displayed the quotient and prepared to perform another series of calculations.

Correcting Errors

Three keys on the keyboard **delete** (erase) errors: the CALC key, the BACKSPACE Key, and the Clear key.

To delete the last character you typed (including decimal points), press the BACKSPACE key.

To delete an entire line, type a Clear (or an ALT X); this empties the calculator field (except for a zero).

If you want to cancel a calculation completely, press the CALC key twice while holding down the Shift key. The first time you press CALC, the calculator functions turn off; the second time, they turn back on (as described earlier, in "Turning on the Calculator Functions"). You can now re-do the calculation you were working on or start another.

SUPER-BIOS REFERENCE DOCUMENT

Super-BIOS Interface Description

Calls to the Super-BIOS are made by loading the AX register with a function number and then doing a software interrupt (level 223). All parameters/results are passed to/from the Super-BIOS functions via a parameter block supplied by the caller. The parameter block is pointed to by a long pointer set by the caller in the ES:BX registers. The Super-BIOS function returns with an exception condition in the AX register: a zero value indicates that the function was succesfully completed.

The contents of the CX,DX,SI,DI and BP and flag registers return undefined. The Super-BIOS interface requires 3 words of the caller's stack.

Setup:	ax=	function number es:bx= pointer to the parameter block
Call:	INT	223
Return:	ax=	exception code return values set in parameter block

Super BIOS Functions

Function 0: Version Number

Returns a unique number used for identifing the release of the operating system.

parm1:	version number	(word/output)

Function 1: Get Zone (for diagnostic use)

Converts a track number into its zone on the disk.

parm1:	track number	(word/input)
parm2:	zone	(word/output)

Function 2: Get Sectors On Track (for diagnostic use)

Returns the number of sectors on the specified track.

parm1:	track number	(word/input)
parm2:	number of sectors	(word/output)

Function 3: Allocate Disk Job (for diagnostic use)

Returns a pointer to a disk job structure.

parm1:	job structure ptr	(long pointer/output)

Function 4: Start Disk Job (for diagnostic use)

Submit a job to the low-level floppy disk driver.

parm1:	job structure ptr	(long pointer/input)
parm2:	priority	(word/input)

Function 5: Get Drive Information (for diagnostic use)

Return a pointer to the specified drive's information structure.

parm1:	drive number	(word/input)
parm2:	DI structure ptr	(long pointer/output)

Function 6: Read Sector

Read physical sector(s) into memory buffer. Full retries are performed.

parm1:	drive number	(word/input)
parm2:	track number	(word/input)
parm3:	sector number	(word/input)
parm4:	sector count	(word/input)
parm5:	buffer address	(long pointer/input)

Function 7: Write Sector

Write physical sector(s) from memory buffer. Full retries are performed.

parm1:	drive number	(word/input)
parm2:	track number	(word/input)
parm3:	sector number	(word/input)
parm4:	sector count	(word/input)
parm5:	buffer address	(long pointer/input)

Function 8: Logical to Physical Disk Address

Convert an logical sector number into a physical track and sector number.

parm1:	drive number	(word/input)
parm2:	logical sector	(word/input)
parm3:	interleave factor	(word/input)
parm4:	track number	(word/output)
parm5:	sector number	(word/output)

Function 9: Flush Disk Buffer

Flush any updated ("dirty") disk buffers.

parm1:	drive number	(word/input)

Function 10: Display Disk Error Message

Display the disk error message for the last disk operation.

Function 11: Get Character Generator

Returns the length and the starting address of the character generator dots in memory.

parm1:	byte count	(word/output)
parm2:	base addr of dots	(word/output)

Function 12: Set Character Generator Size

Set the size of the character generator.

parm1:	byte count	(word/input)

Function 13: Get Screen Address

Return the location of screen memory.

parm1:	base of screen	(word/output)

Function 14: Get Device Vector

Get the current vector for the specified device driver.

parm1:	device number	(word/input)
parm2:	device vector	(long pointer/output)

Function 15: Set Device Vector

Set the vector for the specified device driver.

parm1:	device number	(word/input)
parm2:	device vector	(long pointer/input)

Diskette Backup - The DCOPY Program

The DCOPY program copies the contents, including the system tracks, of one diskette onto another diskette, creating a literal twin of the source or copy-from diskette. In the process, DCOPY formats the destination or copy-to diskette for CP/M-86 (eliminating the need to run the FORMAT program separately).

There are two methods - long and short - for using DCOPY. With the long method, the program prompts the user for the names of the copy-from and copy-to drives. With the short method, the user enters the program name and the copy-from and copy-to drive names in a single command line, without prompts from DCOPY.

The long DCOPY method is used to copy the contents of one or more diskettes in either drive and to make more than one copy. The short method is used to make a single copy of a diskette; the copy-from diskette must contain DCOPY. With this method, the system exits the program immediately after the copy is completed.

Several of the optional switches described for the FORMAT program can be included in the DCOPY command (see below).

USING DCOPY WITH INTERACTIVE SYSTEM PROMPTS

Use the following procedure to copy a diskette from either drive and to make multiple copies.

1. Insert a diskette with DCOPY in drive A, and enter the command:

   dcopy(cr)

   DCOPY is loaded and it displays a sign-on banner at the top of the screen.

2. Remove the system diskette from drive A if it is not the copy-from diskette.

3. At the bottom of the screen, DCOPY asks for the name of the drive containing the diskette to be copied:

   Copy from drive? (A or B; press return key to end)

   Insert the copy-from diskette in either drive and insert the copy-to diskette in the other drive. Answer the DCOPY query with the correct copy-from drive name. DCOPY assumes that the remaining drive is the copy-to drive.

4. DCOPY repeats the particulars of the copy command and asks for confirmation:

   Copy from drive f to drive t. Press space bar when ready.

   where f is the name of the copy-from drive, and t is the name of the copy-to drive. Press the Space bar to start the DCOPY process. Pressing any other key cancels the DCOPY command.

5. During the copy process, at the bottom of the screen, DCOPY displays the number of each track as it is copied.

   Near the top of the screen, DCOPY displays the message:

   Copy from drive f to drive t

   When the copy is complete, the system displays:

   Copy from drive f to drive t complete.

   and prompts with:

   Copy from drive? (A or B; press return key to end)

6. To make another copy, follow steps 3 and 4 above.

   To exit DCOPY (return to the operating system) enter Alt C or press the Return key.

The following example illustrates a sample session using DCOPY to copy one diskette. The example shows DCOPY prompts and messages consecutively, as they are displayed, but is not indicative of where on the screen the prompts and messages appear.

Example A>dcopy(cr)

```
Diskette COPY Utility - Version n.n

Copy from drive? (A or B; press return key to end) a

Copy from drive A to drive B.  Press space bar when ready.(sp)

Copy from drive A to drive B

Copy from drive A to drive B complete.

Copy from drive? (A or B; press return key to end) (cr)

A>
```

USING DCOPY WITHOUT SYSTEM PROMPTS

Use the following procedure for the short, one-command DCOPY method.

1. To copy the system diskette or any diskette containing DCOPY, insert the copy-from diskette in the default drive, insert the copy-to diskette in the other drive. Then enter the following command:

   dcopy f: to t:(cr)

   where f represents the copy-from drive name, and t represents the copy-to drive name.

2. At the bottom of the screen, DCOPY displays the number of each track as it is copied. When DCOPY has completed the copy process, the system displays the following message:

   Copy from drive f to drive t complete.

   NOTE: DCOPY's short command form can include the switches C, E, and Z, as for the FORMAT program. For example, to display the count of tracks copied and of soft errors encountered, and to copy the contents of a diskette in drive A to a diskette in drive B, enter -

   Example A>dcopy a: to b: $c(cr)

Diskette Formatting - The FORMAT Program

The FORMAT program prepares diskettes to receive data. In the process, FORMAT automatically erases any previous files on the diskette. New diskettes must be formatted before they can be used by the system.

USING FORMAT

The basic procedure for formatting a diskette follows:

1. Insert the system diskette in drive A and enter -

   format(cr)

   FORMAT is loaded and it displays a sign-on banner near the top of the screen.

2. Once FORMAT is loaded, the system diskette can be removed. Insert the diskette to be formatted in either drive.

3. At the bottom of the screen, FORMAT asks which drive contains the diskette to be formatted, as follows:

   Format drive? (A or B; press return key to end)

   Enter the letter name of the correct drive.

4. If you have double-sided disk FORMAT also asks:

   Format both sides of the diskette (y/n) ?

   Answer with a "y" if you want a double-sided diskette, else answer with an "n".

5. The system responds by displaying -

   Format drive X. Press space bar when ready.

   where X is the drive name entered in step 3.

   Check that the diskette to be formatted is in the correct drive (to avoid possible data loss) and press the Space bar.

6. The formatting procedure takes approximately one minute for a single-sided diskette. At the bottom of the screen, FORMAT displays the number of each track as it is formatted. Near the top of the screen, FORMAT displays the message -

   Format drive X

   where X is the drive name entered in step 3.

7. When formatting is complete, near the top of the screen, FORMAT displays -

   Format drive X complete.

8. FORMAT prompts to repeat the process. To format another diskette, repeat steps 2 through 5. To end FORMAT insert the system diskette in drive A, and enter Alt C or press the Return key.

The following example illustrates a sample session using FORMAT to format one diskette. The example shows FORMAT prompts and messages consecutively, as they appear on the screen, but does not indicate where on the screen the prompts and messages are displayed.

Example

```
A>format(cr)

Diskette FORMAT Utility - Version n.n

Format drive? (A or B; press return key to end) b

Format drive B.  Press space bar when ready.(sp)

Format drive B

Format drive B complete.

Format drive? (A or B; press return key to end) (cr)

A>
```

USING FORMAT SWITCHES

The user can expand the FORMAT command by including a drive name and optional "switches" that modify FORMAT program operation. These additions must appear in the command line with the FORMAT command stem; a switch value or group of switches must be preceded by a $ (dollar sign). There are four switches:

C - Display the count of the tracks copied and the number of soft errors encountered.

E - Display the locations of soft errors encountered.

Z - Display disk zone information (size of tracks and gaps).

D - Format a double-sided diskette, without asking (step 4). (This is valid only on a system with double-sided disks.)

One or more pf the to four switches may be included in the FORMAT command line, as follows:

format [drivename:] [$C|E|Z|D](cr)

The next example presents the command to format a disk in drive B and then to display (1) the number of tracks copied and (2) the number and the locations of soft errors encountered in the process.

Example

```
A>format b: $ce(cr)

Soft format error: D=1, T=01, S=12, E=4C
Soft format error: D=1, T=01, S=FF, E=41
Soft format error: D=1, T=01, S=06, E=4A

Format complete.
80 tracks formatted: 3 soft errors.

A>
```

FIXLABEL UTILITY

Some diskettes created prior to the 2.0 system release do not have a valid label. The 2.0 and subsequent systems require that a label exist on the diskette. FIXLABEL is used to write a correct label so that the diskette can be used.

An unlabeled diskette is identified if the following error message is displayed when the diskette is initially accessed:

```
Drive = 0, Track = 0, Sector = 0, Error = 22
Bdos Error On A: Bad Sector.
```

or

```
Drive = 1, Track = 0, Sector = 0, Error = 22
Bdos Error On B: Bad Sector.
```

This is a disk checksum error, caused by the bad label.

The command:

```
A>FIXLABEL
```

is used to run FIXLABEL.

The program asks for:

```
Drive A or B?
```

Enter the drive of the diskette to be fixed. (Enter RETURN to return to CP/M.) The program then fixes the label and displays:

```
Fix Label Complete.
```

It is now be possible to access the diskette.

FIXLABEL reads the existing label, prior to writing a new one. If the label is valid for a 2.0 or later system, it is not changed. The program indicates this by displaying:

```
Label is already valid.
```

Character Set Editor - User Notes

Introduction

CEDIT is an interim program, between EDOT and the new character editor, which is used to edit or create character set tables.

Files

You must have the following files to run CEDIT:

ALLOC.CMD	Machine language interface.
PBASIC86.CMD	Microsoft run-time BASIC interpreter.
CEDIT.BAS	The edit program.
*.CHR	The character set files.
CHARGEN.SUB	Submit file, optional.

Invoking

To run CEDIT do either:

```
A>SUBMIT CHARGEN(cr)
```

or

```
A>ALLOC(cr)
A>PBASIC86 CEDIT(cr)
```

CEDIT must be run on a system configured with a logo character set.

It then prompts you for the source character set files to be collected and edited (appear on the right-half of the screen). Enter the file names (.CHR is default) followed by a (cr). After the last one, enter "END".

You are then prompted for one more character set. This is the one which is to be modified and then written out. It appears on the left-half of the screen.

After this character set is read, all the character sets are displayed and the program is in "COPY" mode.

CEDIT Modes

COPY	Copies characters from the right-half of the screen to the left-half. It replaces the character under the left cursor by the character under the right cursor.
DISPLAY	Displays the dot pattern of the character under the right cursor.
EDIT	Allows changes to the dot pattern of a character.

Using CEDIT

In all modes, the number 7, 8, 9, 4, 6, 1, 2, 3 keys are used to move the cursor. The direction of the movement is ilustrated by this diagram, where the cursor is at the X and the movement is in the direction of the number pressed:

```
7  8  9
4  X  6
1  2  3
```

The number 5 key causes different actions, depending on the mode. In the COPY mode, it causes a character copy; in the DISPLAY mode, it enters the EDIT mode; and in the EDIT mode, it inverts the current dot.

COPY mode commands:

R	Activate right cursor only.
L	Activate left cursor only.
B	Activate both cursors.
E	Enter DISPLAY mode.
W	Write left character set.

DISPLAY mode commands:

Only the number keys are used in DISPLAY mode. However, any of the COPY mode commands, except E, return the program to the COPY mode and is performed.

EDIT mode commands:

These commands are the same as the old EDOT commands.

E or P	Position only mode.
T	Toggle mode.
S	Set mode.
R	Reset mode.
C	Clear character.
(cr)	Return to DISPLAY mode.

Saving the Character Set

In either COPY or DISPLAY mode type a "W". This command writes the left character set to the designated file.

You are prompted for a file name, if you respond with just a (cr) then the name of the input file is used. You are then prompted for any changes to the header record.

For now, 256 characters are always written, regardless of the set size.

HOW TO USE THE SYSTEM SELECTION PROGRAM

INTRODUCTION

SYSELECT.BAS, the system selection program, allows users to generate custom operating systems. SYSELECT generates three intermediate files which describe the system to be generated. The actual generation of the operating system is performed by the program MAKESYS.CMD, which takes the component pieces and assembles them into an operating system. The final step of system generation involves the use of BOOTCOPY.CMD to write the operating system boot tracks to the desired floppy disk.

USING SYSELECT

1. Prior to doing a system selection and generation, you may wish to DCOPY the distributed system generation diskette, and store it in a safe place.

2. Place the system generation diskette in drive A and press the reset button to boot the system. The system on this diskette is configured so it automatically start the program SYSELECT.CMD. This in turn loads BASIC with SYSELECT.BAS, afterdoing some initialization.

3. The first menu SYSELECT displays offers these three choices:

```
GENERATE A NEW OPERATING SYSTEM
MODIFY AN EXISTING OPERATING SYSTEM
HELP - DISPLAY INSTRUCTIONS
```

The first line, (GENERATE A NEW...) is highlighted by using reverse video. If this is your choice, press the RETURN key. Else press the space bar to advance to the next selection. When the desired selection is highlighted, press the RETURN key. At this point you may wish to display the instruction summary by choosing the third offer.

The second choice allows a modification to an existing configuration. The available control files are displayed for selection. The configuration specified by the control file is then displayed. A single item can be changed without going through the entire process.

The first option runs through the entire selection process to create a new configuration. If an error is made in a selection it can be fixed from the CONFIGURATION DISPLAY which allows a single item to be selected for modification. This menu is displayed after the configuration menus have been processed. The following menus are displayed:

CHARACTER SET SELECTION

The available display character sets are presented. Each set is described by the banner name, display class, a descriptive comment and the file name. The banner name is the name of the character set, including a version number, which is displayed in the banner. The display class describes the graphic subset (hex21 through hex7E) of the character set. It is provided to help the configurator avoid incompatible combinations of character sets and keyboards. (For example the International display class defines hex23 as the cross hatch '#', whereas the Bristish class defines hex23 as the British monetary sign. On the other hand, the International and VT52T display tables agree in the normal symbols and only vary for those accessed via escape sequences.) The file name is the name of the file which contains the character set.

LTERNATE CHARACTER SET

An alternate character set can be selected. Including an alternate character set allows application programs to display an entirely different character set of up to 255 characters. The user should note that including an alternate character set decreases the available memory by up to 8K bytes.

TRANSLATION TABLE SELECTION

The translation table defines pairs of characters which are to be displayed as a single character. Its primary use is for combining diagritic marks with vowels. Currently French is the only language which requires a translation table. Note that the translation table only affects the displayed characters; even though the pair of characters are displayed as a unit, they are stored and processed as two characters. A translation table must be specified; if no translation is required the table NOTRANS.XLT nust be selected.

KEYBOARD SELECTION

The keyboard table defines the codes generated, or the keyboard functions performed, when a keyboard key is pressed. It is important that the key board table selected corresponds to the actual physical keyboard which is used by your system. This menu has the same format as character set selection (see above for more information.)

PRIMARY PRINTER SELECTION

Primary printer selection sets the name of the default printer of the logical CP/M device LST: . The choices are the serial printer (TTY:) or parrallel printer (LPT:). These values can be changed at run time by the use of STAT. If a serial printer is selected then SYSELECT next displays the menu for serial port configuration, refer to Serial Port Configuration for more information on the selection.

SECONDARY PRINTER

The secondary printer selection allows your system to use a second printer in addition to your choice of primary printer. If a serial printer is chosen then the next selection menu is for configuring the Serial Port. You may note that a parrallel printer can not be selected as a primary printer and a secondary printer. This is done because the system currently supports only one parrallel port.

SERIAL PORT CONFIGURATION

The choices displayed are for setting the baud rate, stop bits, and parity of each of the two serial ports. Refer to the user manual of the device you wish to connect to each serial port for information on these settings. The choices for baud rates are 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2000, 2400, 3600 and 4800. The choices for stop bits are one, 1.5 and two. The choices for parity are even, odd and none (eg parity bit can be set by the software for transmission, and parity is not checked on incoming characters.)

LOGO SELECTION

The logo is a unique graphics set of characters, which when displayed in a particular order form a logo. Normally this is displayed as part of the banner when the operating system is loaded. Generating a system without a logo character set must be matched with a banner which does not display a logo, otherwise garbage appears instead of the logo.

BANNER SKELETON SELECTION

The banner skeleton is a string of characters which define the logo and configuration information to be displayed on the screen as the banner. They are refered to as skeletons because a customized copy of the selected banner is made by the system selection program.

START UP PROGRAM

SYSELECT collects a string of up to 127 characters. This string is used by CP/M as the initial command, after the system load. Note that for this special command CP/M does not map lower case to upper case, thus you must use upper case if you want upper case. Apart from this, the string is no different than a command typed into the system. The string, as displayed in the CONFIGURATION menu, is truncated to 20 characters. However, the entire string is loaded with the system.

CURRENT CONFIGURATION

This menu displays the configuration selected. The first selection in the upper box initiates the process of writing the intermediate files onto the disk. The second choice starts the process from the beginning. The items in the lower box display re the values of the current configuration. Any one of these values can be changed by positioning the highlight at the item to be changed and pressing RETURN. The menu for the specified selection is displayed for modification. After the modification is made, the updated CURRENT CONFIGURATION menu is displayed.

If the selection ACCEPT THE CURRENT CONFIGURATION is chosen the following menu is displayed:

WRITE THE OPERATING SYSTEM OUT

This menu displays the current intermediate files. The user chose one of these, or can enter a unique file name by selecting the USER ENTERED FILE option of the menu. After the selection is made the user is asked to confirm it. If the user answers no to this question, control returns to the CURRENT CONFIGURATION menu. If the user chooses to write to the specifed file then the system preforms this task which can take several minutes.

MAKING AN OPERATING SYSTEM

Following the use of SYSELECT the next step is to run the MAKESYS program to build the operating system according to the specifed configuration. Type MAKESYS followed by the name chosen when runnning SYSELECT. After that operation has been performed successfully type BOOTCOPY again followed by the specified name. Bootcopy allows you to write the new operating system to the disk of your choice.

Operation Notes

System generation consist of running four programs:

SYSELECT.CMD is the drver program which invokes SYSELEC1, PBASIC86(SYSELECT.BAS), MAKESYS and BOOTCOPY.

SYSELEC1.CMD collects the directory structure information for SYSELECT. It stores this information in the file SYSTEMP.$$$.

SYSELECT.BAS is used to make the system component selection. SYSELECT is a Microsoft BASIC86 program. The options currently available in SYSELECT are:

- Generate a new system, or modify an existing one.
- Select a keyboard table.
- Select a translation table.
- Select a display character set.
- Select the printer type.
- Select the serial ports options (baud rate, stop bits, and parity.)
- Select the banner file.
- Select the logo file.
- Select the start up program.

MAKESYS.CMD is used to collect the files selected and created by SYSELECT and create the system file. It is automatically started by SYSELECT.CMD. However, it can also be invoked by typing "MAKESYS xxx", where xxx is the name of the system file given to SYSELECT. The system file is created as xxx.SYS.

BOOTCOPY.CMD is used to copy the system file onto the boot tracks (see BOOTCOPY description). It is also started by SYSELECT.CMD.

System Selection File Description

The information regarding keyboards, characters sets, translation tables, banner skeletons, and the logo comes from files in the current directory. These files are detected by searching the directory for the following file name extensions: Keyboards have the extension .KB, Character sets have the extension .CHR, Banner skeletons have the extension .BAN, translation tables have the extension .XLT, and logos have the extension .LGO. The SYSELECT program expects each file to be in a particular format. Error conditions will occur if any other files use the filename extensions noted above, or if the format of any of the files is modified.

CHARACTER SET DISPLAY TABLE:

All files on the distribution disk with the file name extension ".CHR" are character set display tables. These files contain the data which is the data corresponding to the actual dot matrix displayed for each character on the console. These files also contain information regarding the character set name, version number, origin, date of origin, and display class. Most of this information is displayed by the system selection program to aid in selecting the correct character set. The format of the information in the first sector of these files is the following:

INFORMATION	LENGTH IN BYTES
File Type (K = keyboard, C = Charset)	1
Format version	1
Display Class	12
Banner Name	8
Filler (a space)	1
Banner Version	3
Filler (a space)	1
Comment	35*
Originator**	16
Date (yy/mm/dd)**	8
Length**	4
Unused**	51

* Currently only 31 bytes are displayed by SYSELECT
** Not displayed by SYSELECT

The banner name and banner version are the name and version number of the character set which are placed in the banner and displayed when the system is booted.

KEYBOARD TABLE FILE:

All files with the extension ".KB" are keyboard table files. These files contain the information regarding which code is to be sent when a key on the keyboard is pressed. These files also contain information regarding the keyboard table name, version number, origin, date of origin, and display class which is displayed by the system selection program.

TRANSLATION TABLE FILES:

Files with the extension ".XLT" are translation table files. The translation table is used to define pairs of characters which are to be displayed as a single character. For example, for the French system a circumflex and a vowel are entered, processed and stored as two characters, but must be displayed as one.

BANNER SKELETON FILE:

Files with the extension ".BAN" are banner skeleton files. The banner is the information printed on the screen when the system boots. The banner often prints the LOGO along with other information regarding configuration. The banner is a set of ASCII strings containing the neccessary escape sequences and characters to print the logo and configuration information on the screen when the system boots. SYSELECT displays the available banner files for user selection. A copy of the specified banner is made with the names of the chosen keyboards and character sets placed into the banner in the correct places. The first sector of the banner contains the location of these fields. If the first byte is not zero then SYSELECT does not customize the banner. If the data in the first sector is not "recognizable" then default locations are used when the custom banner is generated.

If a custom banner is written the first sector shall have the following format. The first byte shall be zero followed by a 0Dh 0Ah. This is followed by the length of the file in decimal with a leading and a trailing space and followed by 0Dh,0Ah. The location of the keyboard name and character set name follows in the same format as the file length. Thus if the file length is 639 characters, the keyboard name is to be placed at byte 501, and the character set name is at 541, then the first 24 bytes of the banner file would be the following (in hex):
30 0D 0A 20 36 33 39 20 0D 0A 20 35 30 32 20 0D 0A 20 35 34 31 20 0D 0A.

LOGO FILES

The logo like the character set is data corresponding to a set of special characters which are the actual set of dots representing the logo. If the logo is not of standard size then the first byte must contain its length in sectors. Currently only 16 sector logo files are supported.

OTHER FILES REQUIRED BY SYSELECT

DEFAULT.SKL

DEFAULT.SKL contains the basic information regarding the names of the CP/M, COLD, and HEADER files required by SYSELECT.

SYSTEMP.$$$

This is a temporary file generated by SYSELECT.CMD. The file contains a list of all the available files in the current directory.

FILES WHICH ARE GENERATED BY SYSELECT

*.CTL FILES

The primary output of SYSELECT is a control file which contains the specifications of the operating system to build. Currently existing control files can be modified by SYSELECT by using the "MODIFY EXISTING OPERATING SYSTEM" option of SYSELECT.

*.SPR FILES

An SPR file is generated for each operating system selected by SYSELECT. This file contains data regarding system parameters and is loaded into the operating system

*.BNR FILES

A BNR file (Banner file) is generated each time a system is selected by using SYSELECT. This file is a customized version of the banner skeleton file selected while using SYSELECT.

OPERATING SYSTEM OBJECT FILES

The disk must also contain the CP/M, HEADER, and COLD files specified by the DEFAULT.SKL file.

INSTRUCTION FILES

The file SYSELECT.HLP contains information on the use of SYSELECT. This file is used by SYSELECT when the help function is invoked.

MAKESYS UTILITY

The system build utility (MAKESYS) creates a CP/M operating system file. This is created by concatenating the files which contain the various pieces of the new operating system, such as the keyboard, character set(s), logo, CP/M, and banner.

The command:

```
A>MAKESYS control file
```

is used to create the operating system, as specified in the file control_file. The control_file is created by the SYSELECT utility. The name of the file is chosen by the SYSELECT user; it is displayed in the last SYSELECT message:

```
control_file OPERATING SYSTEM INTERMEDIATE FILES GENERATED.
```

MAKESYS first reads the file names from the control file. It then reads these files and builds the operating system. When the system is completed, MAKESYS writes it on the system file. Prior to creating the new file, it checks if there exists a file with the same name on disk. If there is one, MAKESYS requests aproval, before overwriting the file, with the message:

```
System file <sys_file_name> exists. Should I overwrite it?
```

Enter a 'y' if you want the new system file to overwrite the old one (the old system file is lost). If you enter 'n', the program return to CP/M without creating a new system file.

Example (creating brand new file):

```
A>MAKESYS STDREL2

MAKESYS - Version 1.3

Writing to system file: STDREL2.SYS


A>
```

Example (overwiting an existing file):

```
A>MAKESYS OLDONE

MAKESYS - Version 1.3

System file OLDONE.SYS exists. Should I overwrite it? y


A>
```

ERROR MESSAGES:

Can't open control file -- The control file given does not exist on this diskette. Rerun MAKESYS with the correct file name or diskette.

No memory available -- There is no memory available in the system for allocation.

Not enough memory to transfer file: <file_name> -- The system ran out of available memory. You are trying to build a new system that is too large.

Can't close system file -- You should not get this error. It indicates that the directory has been corrupted.

Can't create system file - directory is full. -- There are no directory entries for the new system file. Erase some files, or use another disk that has fewer files.

File open error: <file_name> -- The named file cannot be found on the disk. Make sure you are using a good system generation diskette.

File read error: <file_name> -- The named file has become corrupted. Redo SYSELECT. If the error persists, use your backup copy of the system generation diskette.

Can't write to system file - disk is full. -- The disk became full while writing the system file. Erase some files on the current disk or use another disk with fewer files and try again.

Can't write to system file - directory is full. -- The system tried to create another directory entry for the system file and was unable to. Erase some files on the current disk or use another disk with fewer files and try again.

BOOTCOPY UTILITY

The bootcopy utility is used to create a system (boot) diskettes by copying a "system image" to the boot tracks of the diskette. The "system image" comes from either an existing system diskette or a file created by the MAKESYS utility.

COPYING THE BOOT TRACKS FROM AN EXISTING SYSTEM DISKETTE

The command:

A>BOOTCOPY source to dest

is used to copy the boot tracks from the source diskette to the destination diskette, where source and dest are of the form "A:" or "B:".

Example:

A>BOOTCOPY A: TO B:

This copies the system tracks from the diskette in drive A to the diskette in drive B. If the source or destination drives are not specified, BOOTCOPY will ask for them.

Example:

```
A>BOOTCOPY

BOOTCOPY - Version 1.6


Source drive (A/B) ? A
Source disk in drive A and press <space>.(sp)

Reading system sectors...

Destination drive (A/B) ? B
Destination disk in drive B and press <space>.(sp)

Writing system sectors...

Bootcopy complete.

Destination drive (A/B) ?(cr)

A>
```

After completing the copy, the program asks you to specify the destination drive again. If you have more diskettes to copy the system tracks to, you can specify the drive number; if you are done, just press the <return> key to go back to the ccp.

COPYING THE BOOT TRACKS FROM A 'SYS' FILE

The command:

A>BOOTCOPY filename TO dest

can be used to copy the contents of SYS file (i.e., a file created using the MKSYS utility) to the system tracks of a diskette. If 'TO dest' is not specified in the command line, then program will ask you for the destination drive.

OVERWRITING A DISKETTE'S DIRECTORY

If the number of system tracks on the destination diskette is not large enough to completly contain the "system image" being copied to them, the directory on that diskette may be lost (since it will be overwritten). In that case, the following warning is issued:

```
WARNING:  all files will be lost.
Continue (y/n)?
```

If you do not care if the files will be lost, type a "Y"; otherwise, type a "N" and the bootcopy process will be terminated.

System Software Dokumentation

sirius COMPUTER