DIGITAL

RESEARCH

CP/M Plus

(CP/M Version 3)

Operating System

System Guide

NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or facilities exist on the distribution disk or as part of the materials and programs distributed.

Most distribution disks include a "README.DOC" file. This file explains variations from the manual which do constitute modification of the manual and the items included therewith. Be sure to read this file before using the software.

The CP/M Plus (CP/M Version 3) Operating System System Guide was prepared using the Digital Research TEX" Text Formatter and printed in the United States of America.

**

First Edition: January 1983

**

Second Edition: July 1983

***

Table of Contents

List of Tables

List of Figures

Foreword

CP/M(R) 3, also marketed as CP/M Plus(R) , is a single-console operating system for 8-bit machines that use an Intel (R) 8080, 8085, or Zilog(R) Z80(R) CPU. CP/M 3 is upward-compatible with its predecessor, CP/M 2, and offers more features and higher performance than CP/M 2. This manual describes the steps necessary to create or modify a CP/M 3 Basic Input Output System (BIOS) tailored for a specific hardware environment.

The CP/M Plus (CP/M Version 3) Operating System System Guide (hereafter cited as CP/M Plus System Guide) assumes you are familiar with systems programming in 8080 assembly language and that you have access to a CP/M 2 system. It also assumes you understand the target hardware and that you have functioning disk I/O drivers. You should be familiar with the accompanying CP/M Plus (CP/M Version 3) Operating System User's Guide (hereafter cited as CP/M Plus User's Guide) describing the operating system utilities. You should also be familiar with the CP/M Plus (CP/M Version 3) Operating system Programmer's Guide (hereafter cited as CP/M Plus Programmer's Guide), which describes the system calls use by the applications programmer to interface with the operating system. The Programmer' s Utilities Guide for the CP/M Family of Operating Systems (hereafter cited as Programmer's Utilities Guide) documents the assembling and debugging utilities.

Section 1 of this manual is an overview of the component modules of the CP/M 3 operating system. Section 2 provides an overview of the functions and data structures necessary to write an interface module between CP/M 3 and specific hardware. Section 3 contains a detailed description of these functions and data structures, followed by instructions to assemble and link the distributed modules with your customized modules. Section 4 describes the modular organization of the sample CP/M 3 BIOS on your distribution diskette.

Section 5 documents the procedure to generate and boot your CP/M 3 system. Section 6 is a sample debugging session.

The appendixes contain tables, and sample BIOS modules you can use, or study and modify. Appendix A discusses removable media drives. Appendix B discusses automatic density support. Appendix C describes how CP/M 3 differs from CP/M 2. Appendix D shows the format of the CPM3.SYS file.

Appendixes E through H are listings of the assembled source code for the four hardware-independent modules of the sample BIOS. Appendix E is the kernel module to use when creating a modular BIOS in the form of the distributed sample. Appendix F shows the System Control Block. Appendix G is a table of equates for the baud rate and mode byte for character I/O. Appendix H contains the macro definitions you can use to generate some of the CP/M 3 disk data structures. Appendix I lists the assembled source code for the six BIOS modules that depend on the Altos@ 8000-15 Computer System hardware. It also contains a sample Submit file to build a BIOS.

Appendixes J and K are tabular summaries of the public entry points and data items in the modules of the sample BIOS. Finally, Appendix L is a tabular summary of the thirty-three functions of the CP/M 3 BIOS, complete with entry parameters and returned values.

Section 1 : CP/M 3 Operating System Overview

This section is an overview of the CP/M 3 operating system, with a description of the system components and how they relate to each other. The section includes a discussion of memory configurations and supported hardware. The last portion summarizes the creation of a customized version of the CP/M 3 Basic Input Output System (BIOS).

1.1 Introduction to CP/M 3

CP/M 3 provides an environment for program development and execution on computer systems that use the Intel 8080, 8085, -or ZBO microprocessor chip. CP/M 3 provides rapid access to data and programs through a file structure that supports dynamic allocation of space for sequential and random access files.

CP/M 3 supports a maximum of sixteen logical floppy or hard disks with a storage capacity of up to 512 megabytes each. The maximum file size supported is 32 megabytes. You can configure the number of directory entries and block size to satisfy various user needs.

CP/M 3 is supplied in two versions. One version supports nonbank-switched memory; the second version supports hardware with bank-switched memory capabilities. CP/M 3 supplies additional facilities for the bank-switched system, including extended command line editing, password protection of files, and extended error messages.

The nonbanked system requires 8.5 kilobytes of memory, plus space for your customized BIOS. It can execute in a minimum of 32 kilobytes of memory.

The bank-switched system requires a minimum of two memory banks with 11 kilobytes of memory in Bank 0 and 1.5 kilobytes in common memory, plus space for your customized BIOS.

The bank-switched system provides more user memory for application programs.

CP/M 3 resides in the file CPM3.SYS, which is loaded into memory by a system loader during system initialization. The system loader resides on the first two tracks of the system disk.

CPM3.SYS contains the distributed BDOS and the customized BIOS.

The CP/M 3 operating system is distributed on two single- density, single-sided, eight-inch floppy disks. Digital Research supplies a sample BIOS that is configured for an Altos 8000-15 microcomputer system with bank-switched memory and two single- density, single-sided, eight-inch floppy disk drives.

1.2 CP/M 3 System Components

The CP/M 3 operating system consists of the following three modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the Basic Input Output System (BIOS).

The CCP is a program that provides the basic user interface to the facilities of the operating system. The CCP supplies six built- in commands: DTR, DIRS, ERASE, RENAME, TYPE, and USER. The CCP executes in the Transient Program Area (TPA), the region of memory where all application programs execute.

The CCP contains the Program Loader Module, which loads transient (applications) programs from disk into the TPA for execution.

The BDOS is the logical nucleus and file system of CP/M 3. The BDOS provides the interface between the application program and the physical input/output routines of the BIOS.

The BIOS is a hardware-dependent module that interfaces the BDOS to a particular hardware environment. The BIOS performs all physical I/O in the system. The BIOS consists of a number of routines that you must configure to support the specific hardware of the target computer system.

The BDOS and the BIOS modules cooperate to provide the CCP and other transient programs with hardware-independent access to CP/M 3 facilities. Because the BIOS is configured for different hardware environments and the BDOS remains constant, you can transfer programs that run under CP/M 3 unchanged to systems with different hardware configurations.

1.3 Communication Between Modules

The BIOS loads the CCP into the TPA at system cold and warm start. The CCP moves the Program Loader Module to the top of the TPA and uses the Program Loader Module to load transient programs.

The BDOS contains a set of functions that the CCP and applications programs call to perform disk and character input and output operations.

The BIOS contains a Jump Table with a set of 33 entry points that the BDOS calls to perform hardware-dependent primitive functions, such as peripheral device I/O. For example, CONIN is an entry point of the BIOS called by the BDOS to read the next console input character.

Similarities exist between the BDOS functions and the BIOS functions, particularly for simple device I/O. For example, when a transient program makes a console output function call to the BDOS, the BDOS makes a console output call to the BIOS. In the case of disk I/O, however, this relationship is more complex.

The BDOS might make many BIOS function calls to perform a single BDOS file I/O function. BDOS disk I/O is in terms of 128-byte logical records. BIOS disk I/O is in terms of physical sectors and tracks.

The System Control Block (SCB) is a 100-byte, decimal, CP/M 3 data structure that resides in the BDOS system component. The BDOS and the BIOS communicate through fields in the SCB. The SCB contains BDOS flags and data, CCP flags and data, and other system information, such as console characteristics and the current date and time. You can access some of the System Control Block fields from the BIOS.

Note that the SCB contains critical system parameters which reflect the current state of the operating system. If a program modif i es these parameters, the operating system can crash.

See Section 3 of this manual, and the description of BDOS Function 49 in the CP/M Plus Proqrammer's Guide for more information on the System Control Block.

Page Zero is a region of memory that acts as an interface between transient programs and the operating system. Page Zero contains critical system parameters, including the entry to the BDOS and the entry to the BIOS Warm BOOT routine. At system start-up, the BIOS initializes these two entry points in Page Zero. All linkage between transient programs and the BDOS is restricted to the indirect linkage through Page Zero. Figure 1-1 illustrates the general memory organization of CP/M 3.

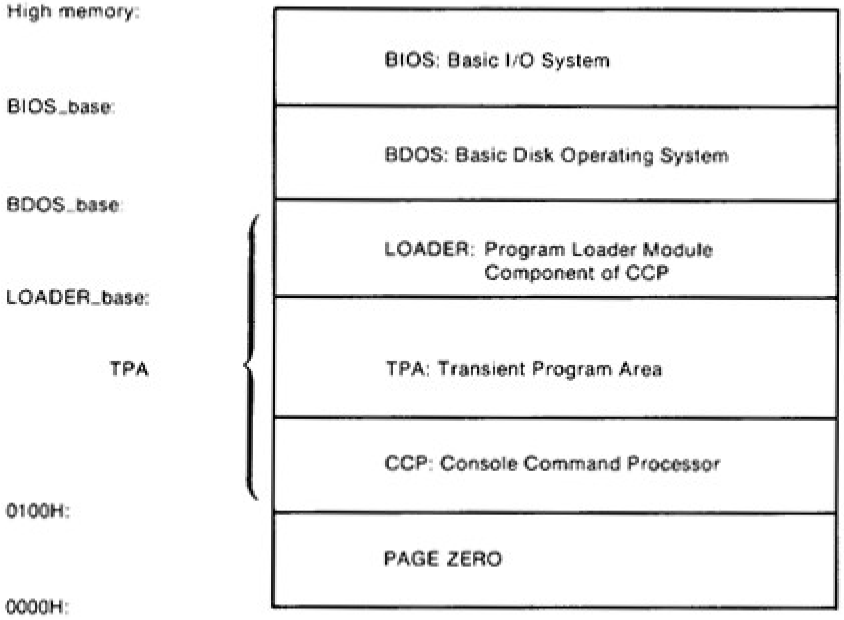

Figure 1-1 : General Memory Organization of CP/M 3

Note that all memory regions in CP/M 3 are page aligned, which means that they must begin on a page boundary. Because a page is defined as 256 (100H) bytes, a page boundary always begins at a hexadecimal address where the low-order byte of the hex address is zero.

1.4 Banked and Nonbanked Systems

CP/M 3 is supplied in two versions: one for hardware that supports banked memory, and the other for hardware with a minimum of 32 kilobytes of memory. The systems are called banked and nonbanked.

Digital Research supplies System Page Relocatable (. SPR) files for both a banked BDOS and a nonbanked BDOS. A sample banked BIOS is supplied for you to use as an example when creating a customized BIOS for your set of hardware components.

The following figure shows the memory organization for a banked system. Bank 0 and common memory are for the operating system. Bank 1 is the Transient Program Area, which contains the Page Zero region of memory. You can use additional banks to enhance operating system performance.

In banked CP/M 3 systems, CPMLDR, the system loader, loads part of the BDOS into common memory and part of the BDOS into Bank 0. CPMLDR loads the BIOS in the same manner.

Figure 1-2 shows the memory organization for the banked version of CP/M 3.

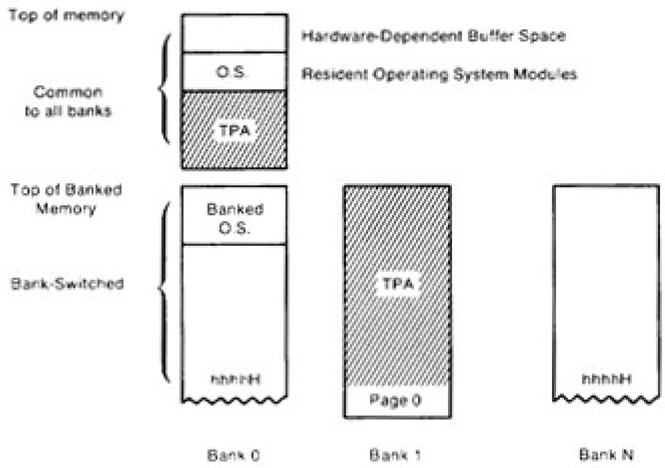

Figure 1-2. Memory organization for Banked CP/M 3 System

In this figure, the top region of memory is called common memory. Common memory is always enabled and addressable. The operating system is divided into two modules: the resident portion, which resides in common memory, and the banked portion, which resides just below common memory in Bank 0.

The shaded areas in Figure 1-2 represent the memory available to transient programs.

The clear areas are used by the operating system for disk record buffers and directory hash tables. The clear area in the common region above the operating system represents space that can be allocated for data buffers by GENCPM, the CP/M 3 system generation utility.

The size of the buffer area is determined by the specific hardware requirements of the host microcomputer system.

Bank 0, the system bank, is the bank that is enabled when CP/M 3 is cold started. Bank 1 is the transient program bank. The transient program bank must be contiguous from location zero to the top of banked memory. Common memory must also be contiguous. The other banks need not begin at location zero or have contiguous memory.

Figure 1-3 shows the CP/M 3 memory organization when the TPA bank, Bank 1, is enabled in a bank-switched system.

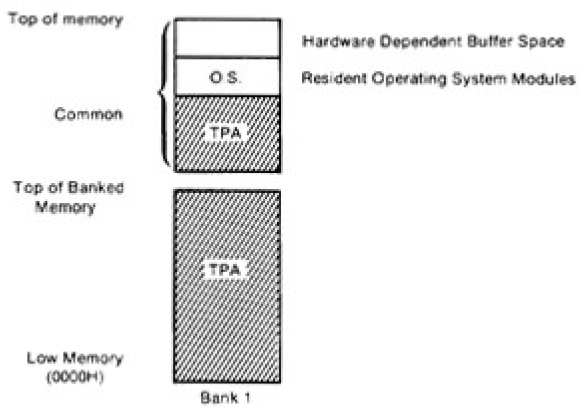

Figure 1-3. Memory Organization with Bank I Enabled in Banked System

The operating system switches to Bank 0 or other banks when performing operating system functions. In general, any bank switching performed by the operating system is transparent to the calling program.

The memory organization for the nonbanked version of CP/M 3 is much simpler, as shown in Figure 1-4:

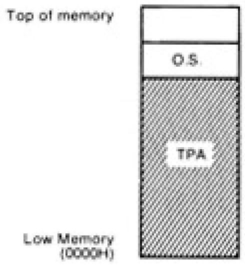

Figure 1-4. Memory Organization in Nonbanked CP/M 3 System

In the nonbanked version of CP/M 3, memory consists of a single contiguous region addressable from 0000H up to a maximum of 0FFFFH, or 64K-1. The clear area above the operating system represents space that can be allocated for data buffers and directory hash tables by the CP/M 3 system generation utility, GENCPM, or directly allocated by the BIOS. The minimum size of the buffer area is determined by the specific hardware requirements of the host microcomputer system. Again, the shaded region represents the space available for transient programs.

1.5 Memory Requirements

Table 1-1 shows typical sizes of the CP/M 3 operating system components.

Table 1-1. CP/M 3 Operating System Memory Requirements

CP/M 3 Version	Nonbanked	Banked	
		Common	Bank 0
BDOS	8.5K	1.5K	11K
	BIOS (values vary)		
Floppy system	1.5K	.75K	2K
Hard system	2.5K	1.5K	3K

The CP/M 3 banked system requires a minimum of two banks (Bank 0 and Bank 1) and can support up to 16 banks of memory. The size of the common region is often 16K, but can be as small as 4K. Common memory must be large enough to contain the required buffers and the resident (common) portion of the operating system, which means a 1.5K BDOS and the common part of your customized BIOS.

In a banked environment, CP/M 3 maintains a cache of deblocking buffers and directory records using a Least Recently Used (LRU) buffering scheme. The LRU buffer is the first to be reused when the system runs out of buffer space. The BDOS maintains separate buffer pools for directory and data record caching.

The RSX modules shown in Figure 1-5 are Resident System Extensions (RSX) that are loaded directly below the operating system when included in an application or utility program.

The Program Loader places the RSX in memory and chains BDOS calls through the RSX entry point in the RSX.

Figure 1-5 shows the memory organization in a typical bank- switched CP/M 3 system.

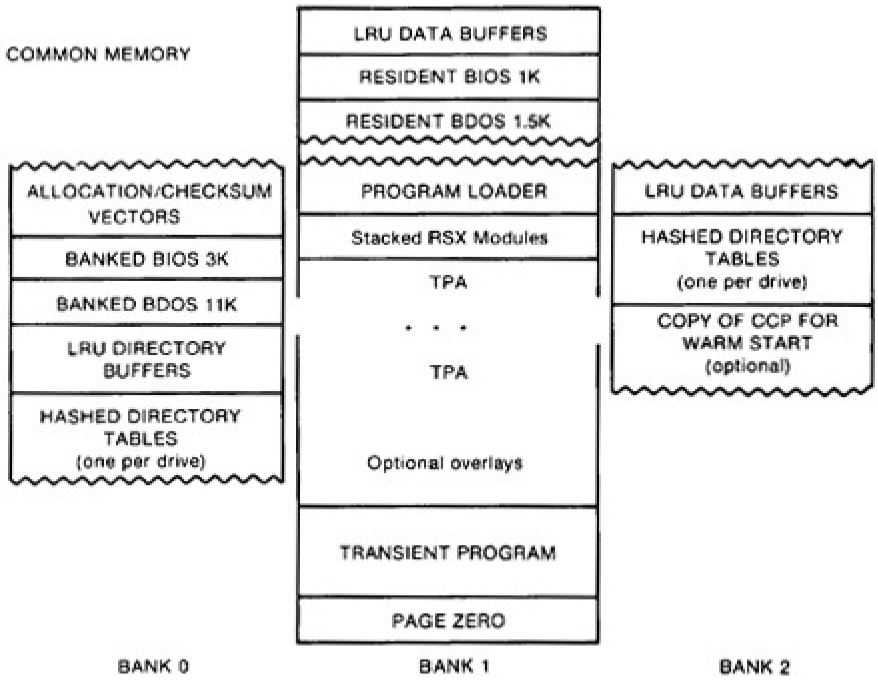

Figure 1-5. Memory Organization in Banked CP/M 3

The banked system supports a TPA of 60K or more. The banked portion of the operating system in Bank 0 requires at least 16K of memory.

In the banked system, the BDOS and the BIOS are separated into two parts: a resident portion, and a banked portion. The resident BDOS and BIOS are located in common memory.

The banked BDOS and BIOS are located in the operating system bank, referred to as Bank 0 in this manual.

The TPA extends from 100H in Bank 1 up to the bottom of the resident BDOS in common memory. The banked BIOS and BDOS reside in Bank 0 with the directory buffers.

Typically, all data buffers reside in common. Data buffers can reside in an alternate bank if the system has a DMA controller capable of transferring arbitrary blocks of data from one bank to another. Hashed directory tables (one per drive) can be placed in any bank except Bank 1 (TPA). Hashed directory tables require 4 bytes per directory entry.

Figure 1-6 shows a typical nonbanked system configuration.

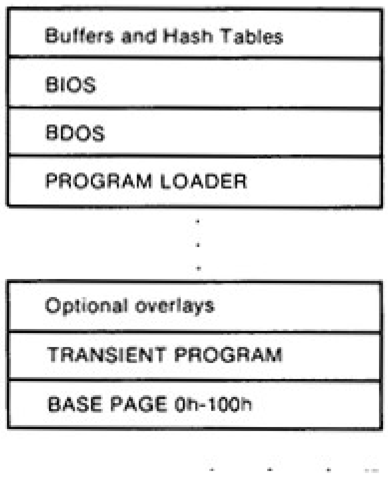

Figure 1-6. Memory organization in Nonbanked CP/M 3

The nonbanked CP/M 3 system requires 8.5K of memory plus space for the BIOS, buffers, and hash tables, allowing a TPA size of up to 52K to 54K, depending on the size of the BIOS and the number of hash tables and buffers you are using.

1.6 Disk Organization

Figure 1-7 illustrates the organization of a CP/M 3 system disk.

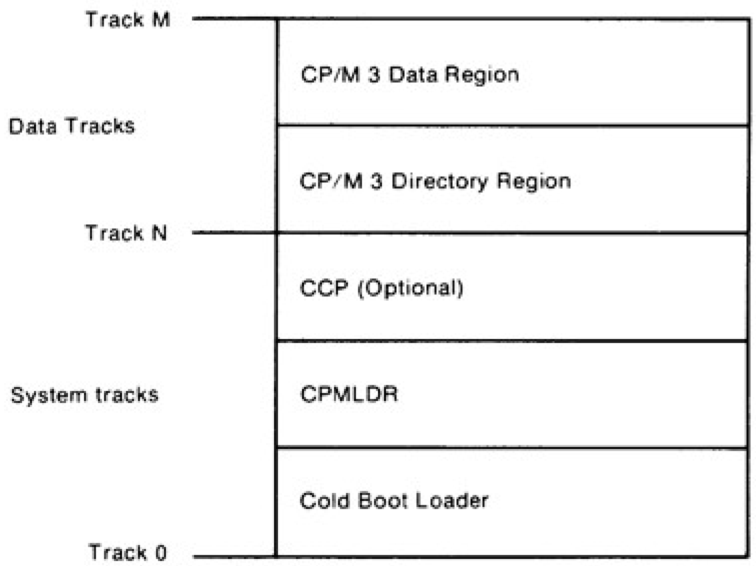

Figure 1-7. CP/M 3 System Disk Organization

In Figure 1-7, the first N tracks are the system tracks; the remaining tracks, the data tracks, are used by CP/M 3 for file storage. Note that the system tracks are used by CP/M 3 only during system cold start and warm start. All other CP/M 3 disk access is directed to the data tracks of the disk. To maintain compatibility with Digital Research products, you should use an eight-inch, single-density, IBM' 3740 formatted disk with two system tracks.

1.7 Hardware Supported

You can customize the BIOS to match any hardware environment with the following general characteristics.

1.7.1 Hardware Supported by CP/M 3 Banked System

- Intel 8080, Intel 8085, or zilog Z80 CPU or equivalent.
- A minimum of two and up to sixteen banks of memory with the top 4K-32K in common memory. Bank 1 must have contiguous memory from address 0000H to the base of common memory. A reasonable configuration consists of two banks of 48K RAM each, with the top 16K in common memory.
- one to sixteen disk drives of up to 512 megabytes capacity each.
- Some form of ASCII console device, usually a CRT.
- One to twelve additional character input and or output devices, such as printers, communications hardware, and plotters.

1.7.2 Hardware Supported by CP/M 3 Nonbanked System

- Intel 8080, Intel 8085, or Zilog Z80 CPU or equivalent.
- A minimum of 32K and up to 64K contiguous memory addressable from location zero.
- One to sixteen disk drives of up to 512 megabytes capacity each.
- Some form of ASCII console device, usually a CRT.
- One to twelve additional input and or output devices, usually including a printer.

Because most CP/M-compatible software is distributed on eight- inch, soft-sectored, single-density floppy disks, it is recommended that a CP/M 3 hardware configuration include a minimum of two disk drives, at least one of which is a single-density floppy disk drive.

1.8 Customizing CP/M 3

Digital Research supplies the BDOS files for a banked and a nonbanked version of CP/M

3. A system generation utility, GENCPM, is provided with CP/M 3 to create a version of the operating system tailored to your hardware. GENCPM combines the BDOS and your customized BIOS files to create a CPM3.SYS file, which is loaded into memory at system start-up. The CPM3.SYS file contains the BDOS and BIOS system components and information indicating where these modules reside in memory.

Digital Research supplies a CP/M 3 loader file, CPMLDR, which you can link with your customized loader BIOS and use to load the CPM3.SYS file into memory. CPMLDR is a small, self-contained version of CP/M 3 that supports only console output and sequential file input.

Consistent with CP/M 3 organization, it contains two modules: an invariant CPMLDR BDOS, and a variant CPMLDR-BIOS, which is adapted to match the host microcomputer hardware environment. The CPMLDR BIOS module can perform cold start initialization of I/O ports and similar functions. CPMLDR can display a memory map of the CP/M 3 system at start-up. This is a GENCPM option.

The following steps tell you how to create a new version of CP/M 3 tailored to your specific hardware.

1) Write and assemble a customized BIOS following the specifications described in Section 3. This software module must correspond to the exact physical characteristics of the target system, including memory and port addresses, peripheral types, and drive characteristics.
2) Use the system generation utility, GENCPM, to create the CPM3.SYS file containing the CP/M 3 distributed BDOS and your customized BIOS, as described in Section 5.
3) Write a customized loader BIOS (LDRBIOS) to reside on the system tracks as of CPMLDR. CPMLDR loads the CPM3.SYS file into memory from disk. Section 5 gives the instructions for customizing the LDRBIOS and generating CPMLDR. Link your customized LDRBIOS file with the supplied CPMLDR file.
4) Use the COPYSYS utility to put CPMLDR on the system tracks of a disk.

5) Test and debug your customized version of CP/M 3.

If you have banked memory, Digital Research recommends that you first use your customized BIOS to create a nonbanked version of the CP/M 3 operating system. You can leave your entire BIOS in common memory until you have a working system. Test all your routines in a nonbanked version of CP/M 3 before you create a banked version.

1.9 Initial Load (Cold Boot) of CP/M 3

CP/M 3 is loaded into memory as follows. Execution is initiated by a four-stage procedure. The first stage consists of loading into memory a small program, called the Cold Boot Loader, from the system tracks of the Boot disk. This load operation is typically handled by a hardware feature associated with system reset. The Cold Boot Loader is usually 128 or 256 bytes in length.

In the second stage, the Cold Boot Loader loads the memory image of the CP/M 3 system loader program, CPMLDR, from the system tracks of a disk into memory and passes control to it. For a banked system, the Cold Boot Loader loads CPMLDR into Bank 0.

A PROM loader can perform stages one and two.

In the third stage, CPMLDR reads the CPM3.SYS f ile, which contains the BDOS and customized BIOS, from the the data area of the disk into the memory addresses assigned by GENCPM. In a banked system, CPMLDR reads the common part of the BDOS and BIOS into the common part of memory, and reads the banked part of the BDOS and BIOS into the area of memory below common base in Bank 0. CPMLDR then transfers control to the Cold BOCT system initialization routine in the BIOS.

For the f inal stage, the BIOS Cold BOOT routine, BIOS Function 0, performs any remaining necessary hardware initialization, displays the sign-on message, and reads the CCP from the system tracks or from a CCP.COM file on disk into location 100H of the TPA. The Cold BOOT routine transfers control to the CCP, which then displays the system prompt.

Section 2 provides an overview of the organization of the System Control Block and the data structures and functions in the CP/M 3 BIOS.

Section 2 : CP/M 3 BIOS Overview

This section describes the organization of the CP/M 3 BIOS and the BDOS jump vector.

It provides an overview of the System Control Block, followed by a discussion of system initialization procedures, character I/O, clock support, disk I/O, and memory selects and moves.

2.1 Organization of the BIOS

The BIOS is the CP/M 3 module that contains all hardware- dependent input and output routines. To configure CP/M 3 for a particular hardware environment, use the sample BIOS supplied with this document and adapt it to the specific hardware of the target system.

Alternatively, you can modify an existing CP/M 2.2 BIOS to install CP/M 3 on your target machine. Note that an unmodified CP/M 2.2 BIOS does not work with the CP/M 3 operating system. See Appendix C for a description of the modifications necessary to convert a CP/M 2.2 BIOS to a CP/M 3 BIOS.

The BIOS is a set of routines that performs system initialization, character-oriented I/O to the console and printer devices, and physical sector I/O to the disk devices. The BIOS also contains routines that manage block moves and memory selects for systems with bank-switched memory. The BIOS supplies tables that define the layout of the disk devices and allocate buffer space which the BDOS uses to perform record blocking and deblocking. The BIOS can maintain the system time and date in the System Control Block.

Table 2-1 describes the entry points into the BIOS from the Cold Start Loader and the BDOS. Entry to the BIOS is through a jump vector. The jump vector is a set of 33 jump instructions that pass program control to the individual BIOS subroutines.

You must include all of the entry points in the BIOS jump vector in your BIOS.

However, if your system does not support some of the functions provided for in the BIOS, you can use empty subroutines for those functions. For example, if your system does not support a printer, JMP LIST can reference a subroutine consisting of only a RET instruction. Table 2-1 shows the elements of the jump vector.

Table 2-1. CP/M 3 BIOS Jump Vector

No.	Instruction	Description
0	JMP BOOT	Perform cold start initialization
1	JMP WBOOT	Perform warm start initialization
2	JMP CONST	Check for console input character ready
3	JMP CONIN	Read Console Character in
4	JMP CONOUT	Write Console Character out
5	JMP LIST	Write List Character out
6	JMP AUXOUT	Write Auxiliary Output Character
7	JMP AUXIN	Read Auxiliary Input Character
8	JMP HOME	Move to Track 00 on Selected Disk
9	JMP SELDSK	Select Disk Drive
10	JMP SETTRK	Set Track Number
11	JMP SETSEC	Set Sector Number
12	JMP SETDMA	Set DMA Address
13	JMP READ	Read Specified Sector
14	JMP WRITE	Write Specified Sector
15	JMP LISTST	Return List Status
16	JMP SECTRN	Translate Logical to Physical Sector
17	JMP CONOST	Return Output Status of Console
18	JMP AUXIST	Return Input Status of Aux. Port
19	JMP AUXOST	Return Output Status of Aux. Port
20	JMP DEVTBL	Return Address of Char. I/O Table
21	JMP DEVINI	Initialize Char. I/O Devices
22	JMP DRVTBL	Return Address of Disk Drive Table

No.	Instruction	Description
23	JMP MULTIO	Set Number of Logically Consecutive sectors to be read or written
24	JMP FLUSH	Force Physical Buffer Flushing for user-supported deblocking
25	JMP MOVE	Memory to Memory Move
26	JMP TIME	Time Set/Get signal
27	JMP SELMEM	Select Bank of memory
28	JMP SETBNK	Specify Bank for DMA Operation
29	JMP XMOVE	Set Bank When a Buffer is in a Bank other than 0 or 1
30	JMP USERF	Reserved for System Implementor
31	JMP RESERV1	Reserved for Future Use
32	JMP RESERV2	Reserved for Future Use

Each jump address in Table 2-1 corresponds to a particular subroutine that performs a specific system operation. Note that two entry points are reserved for future versions of CP/M, and one entry point is provided for OEM subroutines, accessed only by direct BIOS calls using BDOS Function 50. Table 2-2 shows the five categories of system operations and the function calls that accomplish these operations.

Table 2-2. CP/M 3 BIOS Functions

Operation	Function
System Initialization	BOOT, WBOOT, DEVTBL, DEVINI, DRVTBL
Character I/O	CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, AUXOST
Disk I/O	HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, FLUSH
Memory Selects and Moves	MOVE, SELMEM, SETBNK, XMOVE
Clock Support	TIME

You do not need to implement every function in the BIOS jump vector. However, to operate, the BDOS needs the BOOT, WBOOT, CONST, CONIN, CONOUT, HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, and FLUSH subroutines.

Implement SELMEM and SETBNK only in a banked environment. You can implement MULTIO, FLUSH, and TIME as returns with a zero in register A. DEVICE and some other utilities use the remaining entry points, but it is not necessary to fully implement them in order to debug and develop the system.

Note: include all routines but make the nonimplemented routines a RET instruction.

2.2 System Control Block

The System Control Block (SCB) is a data structure located in the BDOS. The SCB is a communications area referenced by the BDOS, the CCP, the BIOS, and other system components. The SCB contains system parameters and variables, some of which the BIOS can reference. The fields of the SCB are named, and definitions of these names are supplied as public variable and subroutine names in the SCB.ASM file contained on the distribution disk. See Section 3.1 for a discussion of the System Control Block.

2.3 System Initialization

When the BOOT and WBOOT routines of the BIOS get control, they must initialize two system parameters in Page Zero of memory, as shown in Table 2-3.

Table 2-3. Initialization of Page Zero

Location	Description
0,1,2	Set to JMP WBOOT (0000H: JMP BIOS+3). Location 1 and 2 must contain the address of WBOOT in the jump vector.
5,6,7	Set to JMP BDOS, the primary entry point to CP/M 3 for transient programs. The current address of the BDOS is maintained in the variable @MXTPA in the System Control Block. (See Section 3.1, "System Control Block," and BIOS Function 1: WBOOT)

The BOOT and WBOOT routine must load the CCP into the TPA in Bank I at location 0100H. The CCP can be loaded in two ways. If there is sufficient space on the system tracks, the CCP can be stored on the system tracks and loaded from there. If you prefer, or if there is not sufficient space on the system tracks, the BIOS Cold BOOT routine can read the CCP into memory from the file CCP.COM on disk.

If the CCP is in a COM file, use the BOOT and WBOOT routines to perform any necessary system initialization, then use the BDOS functions to OPEN and READ the CCP.COM file into the TPA. In bank-switched systems, the CCP must be read into the TPA in Bank 1.

In bank-switched systems, your Cold BOOT routine can place a copy of the CCP into a reserved area of an alternate bank after loading the CCP into the TPA in Bank 1. Then the Warm BOOT routine can copy the CCP into the TPA in Bank 1 from the alternate bank, rather than reloading the CCP from disk, thus avoiding all disk accesses during warm starts.

There is a 128-byte buffer in the resident portion of the BDOS in a banked system that can be used by BOOT and WBOOT. The address of this buffer is stored in the SCB variable @BNKBF. BOOT and WBOOT can use this buffer when copying the CCP to and from the alternate bank.

The system tracks for CP/M 3 are usually partitioned as shown in the following figure;

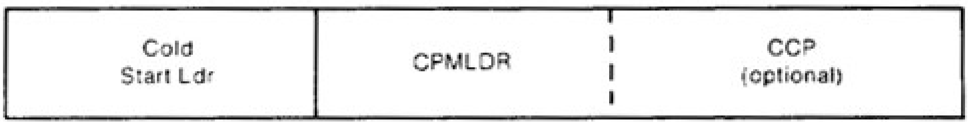

Figure 2-1. CP/M 3 System Tracks

The cold start procedure is designed so you need to initialize the system tracks only once. This is possible because the system tracks contain the system loader and need not change when you change the CP/M 3 operating system. The Cold Start Loader loads CPMLDR into a constant memory location that is chosen when the system is configured. However , CPMLDR loads the BDOS and BIOS system components into memory as specified in the CPM3.SYS file generated by GENCPM, the system generation utility. Thus, CP/M 3 allows the user to configure a new system with GENCPM and then run it without having to update the system tracks of the system disk.

2.4 Character I/O

CP/M 3 assumes that all simple character I/O operations are performed in 8-bit ASCII, upper- and lowercase, with no parity. An ASCII CRTL-Z (lAH) denotes an end-of-file condition for an input device.

Table 2-4 lists the characteristics of the logical devices.

Table 2-4. CP/M 3 Logical Device Characteristics

Device	Characteristics
CONIN, CONOUT	The interactive console that communicates with the operator, accessed by CONST, CONIN, CONOUT, and CONOUTST. Typically, the CONSOLE is a device such as a CRT or teletype, interfaced serially, but it can also be a memory-mapped video display and keyboard. The console is an input device and an output device.
LIST	The system printer, if it exists on your system. LIST is usually a hard- copy device such as a printer or teletypewriter.
AUXOUT	The auxiliary character output device, such as a modem.
AUXIN	The auxiliary character input device, such as a modem.

Note that you can define a single peripheral as the LIST, AUXOUT, and AUXIN device simultaneously. If you assign no peripheral device as the LIST, AUXOUT, or AUXIN device, the AUXOUT and LIST routines can just return, and the AUXIN routine can return with a 1AH (CTRL-Z) in register A to indicate an immediate end-of-file.

CP/M 3 supports character device I/O redirection. This means that you can direct a logical device, such as CONIN or AUXOUT, to one or more physical devices. The DEVICE utility allows you to reassign devices and display, and to change the current device configurations, as described in the CP/M Plus User's Guide. The I/O redirection facility is optional. You should not implement it until the rest of your BIOS is fully functional.

2.5 Disk I/O

The BDOS accomplishes disk I/O by making a sequence of calls to the various disk access subroutines in the BIOS. The subroutines set up the disk number to access, the track and sector on a particular disk, and the Direct Memory Access (DMA) address and bank involved in the I/O operation. After these parameters are established, the BDOS calls the READ or WRITE function to perform the actual I/O operation.

Note that the BDOS can make a single call to SELDSK to select a disk drive, follow it with a number of read or write operations to the selected disk, and then select another drive for subsequent operations.

CP/M 3 supports multiple sector read or write operations to optimize rotational latency on block disk transfers. You can implement the multiple sector I/O facility in the BIOS by using the multisector count passed to the MULTIO entry point. The BDOS calls MULTIO to read or write up to 128 sectors. For every sector number 1 to n, the BDOS calls SETDMA then calls READ or WRITE.

Table 2-5 shows the sequence of BIOS calls that the BDOS makes to read or write a physical disk sector in a nonbanked and a banked system. Table 2-6 shows the sequence of calls the BDOS makes to the BIOS to read or write multiple contiguous physical sectors in a nonbanked and banked system.

Table 2-5. BDOS Calls to BIOS in Nonbanked and Banked Systems

Nonbanked BDOS

Call	Explanation
SELDSK	Called only when disk is initially selected or reselected.
SETTRK	Called for every read or write of a physical sector.
SETSEC	Called for every read or write of a physical sector.
SETDMA	Called for every read or write of a physical sector.
READ, WRITE	Called for every read or write of a physical sector.

Banked BDOS

Call	Explanation
SELDSK	Called only when disk is initially selected or reselected.
SETTRK	Called for every read or write of a physical sector.
SETSEC	Called for every read or write of a physical sector.
SETDMA	Called for every read or write of a physical sector.
SETBNK	Called for every read or write of a physical sector.
READ, WRITE	Called for every read or write of a physical sector.

Nonbanked BDOS

Call	Explanation
SELDSK	Called only when disk is initially selected or reselected.
MULTIO	Called to inform the BIOS that the next n calls to disk READ or disk WRITE require a transfer of n contiguous physical sectors to contiguous memory.
SETTRK	Called for every read or write of a physical sector.
SETSEC	Called for every read or write of a physical sector.
SETDMA	Called for every read or write of a physical sector.
READ, WRITE	Called for every read or write of a physical sector.

Banked BDOS

Call	Explanation
SELDSK	Called only when disk is initially selected or reselected.
MULTIO	Called to inform the BIOS that the next n calls to disk READ or disk WRITE require a transfer of n contiguous physical sectors to contiguous memory.
SETTRK	Called for every read or write of a physical sector.
SETSEC	Called for every read or write of a physical sector.

SETDMA	Called for every read or write of a physical sector.
SETBNK	Called for every read or write of a physical sector.
READ, WRITE	Called for every read or write of a physical sector.

Table 2-7 shows the sequence of BDOS calls to read two contiguous physical sectors in a banked system.

Table 2-7. Reading Two Contiguous Sectors in Banked System

Call	Explanation
SELDSK	Called to initially select disk
MULTIO	With a value of 2
SETTRK	For first sector
SETSEC	For first sector
SETDMA	For first sector
SETBNK	
READ	
SETTRK	For second sector
SETSEC	For second sector
SETDMA	For second sector
SETBNK	
READ	

The CP/M 3 BDOS performs its own blocking and deblocking of logical 128-byte records. Unlike earlier versions of CP/M, the BIOS READ and WRITE routines always transfer physical sectors as specified in the Disk Parameter Block to or from the DMA buffer. The Disk Parameter Header defines one or more physical sector buffers which the BDOS uses for logical record blocking and deblocking.

In a banked environment, CP/M 3 maintains a cache of deblocking buf fers and directory records using a Least Recently Used (LRU) buffering scheme. The LRU buffer is the first to be reused when the system runs out of buffer space. The BDOS maintains separate buffer pools for directory and data record caching.

The BIOS contains the data structures to control the data and directory buffers and the hash tables. You can either assign these buffers and tables yourself in the BIOS, or allow the GENCPM utility to generate them automatically.

Hash tables greatly speed directory searching. The BDOS can use hash tables to determine the location of directory entries and therefore reduce the number of disk accesses required to read a directory entry. The hash table allows the BDOS to directly access the sector of the directory containing the desired directory entry without having to read the directory sequentially. By eliminating a sequential read of the directory records, hashing also increases the percentage of time that the desired directory record is in a buffer, eliminating the need for any physical disk accesses in these cases. Hash tables and directory caches eliminate many of the directory accesses required when accessing large files. However, in a nonbanked system, hash tables increase the size of the operating system.

When the BIOS finds an error condition, the READ and WRITE routines should perform several retries before reporting the error condition to the BDOS. Ten retries are typical. If the BIOS returns an error condition to the BDOS, the BDOS reports the error to the user in the following form:

CP/M Error on d: Disk I/O

The d: represents the drive specification of the relevant drive.

To provide better diagnostic capabilities for the user, it is often desirable to print a more explicit error message from the BIOS READ or WRITE routines before the BIOS returns an error code to the BDOS. The BIOS should interrogate the SCB Error Mode Variable to determine if it is appropriate to print a message on the console.

2.6 Memory Selects and Moves

Four BIOS functions are provided to perform memory management. The functions are MOVE, XMOVE, SELMEM, and SETBNK. The XMOVE, SELMEM, and SETBNK memory management routines are applicable to the BIOS of banked systems.

The BDOS uses the BIOS MOVE routine to perform memory-to-memory block transfers.

In a banked system, the BDOS calls XMOVE to specify the source and destination banks to be used by the MOVE routine. If you use memory that is not in the common area for data record buffers, you must implement the XMOVE routine.

The BDOS uses SELMEM when the operating system needs to execute code or access data in other than the currently selected bank.

The BDOS calls the SETBNK routine prior to calling disk READ or disk WRITE functions. The SETBNK routine must save its specified bank as the DMA bank. When the BDOS invokes a disk I/O routine, the I/O routine should save the current bank number and select the DMA bank prior to the disk READ or WRITE. After completion of the disk READ or WRITE, the disk I/O routine must reselect the current bank. Note that when the BDOS calls the disk I/O routines, Bank 0 is in context (selected).

2.7 Clock Support

If the system has a real-time clock or is capable of keeping time, possibly by counting interrupts from a counter/timer chip, then the BIOS can maintain the time of day in the System Control Block and update the time on clock interrupts. BIOS Function 26 is provided for those systems where the clock is unable to generate an interrupt.

The time of day is kept as four fields. @DATE is a binary word containing the number of days since 31 December 1977. The bytes @HOUR, @MIN, and @SEC in the System Control Block contain the hour, minute, and second in Binary Coded Decimal (BCD) format.

Section 3 : CP/M 3 BIOS Functional Specifications

This section contains a detailed description of the CP/M 3 BIOS The section first discusses the BIOS data structures and their relationships, including the System Control Block, the drive table, the Disk Parameter Header, the Disk Parameter Block, the Buffer Control Blocks, and the character I/O table. The overview of the data structures is followed by a summary of the functions in the BIOS jump vector. A detailed description of the entry values and returned values for each jump instruction in the BIOS jump vector follows the summary. The last part of this section discusses the steps to follow when assembling and linking your customized BIOS.

3.1 The System Control Block

The System Control Block (SCB) is a data structure located in the BDOS. The SCB contains flags and data used by the CCP, the BDOS, the BIOS, and other system components.

The BIOS can access specif ic data in the System Control Block through the public variables defined in the SCB.ASM file, which is supplied on the distribution disk.

Declare the variable names you want to reference in the SCB as externals in your BIOS.ASM source file. Then link your BIOS with the SCB.REL module.

In the SCB.ASM file, the high-order byte of the various SCB addresses is defined as 0FEH. The linker marks absolute external equates as page relocatable when generating a System Page Relocatable (SPR) format file. GENCPM recognizes page relocatable addresses of 0FExxH as references to the System Control Block in the BDOS. GENCPM changes these addresses to point to the actual SCB in the BDOS when it is relocating the system.

Do not perform assembly-time arithmetic on any references to the external labels of the SCB. The result of the arithmetic could alter the page value to something other than 0FEH.

Listing 3-1 shows the SCB.ASM file. The listing shows the f ield names of the System Control Block. A @ before a name indicates that it is a data item. A ? preceding a name indicates that it is the label of an instruction. In the listing, r/w means Read-Write, and r/o means Read-Only. The BIOS can modify a Read- Write variable, but must not modify a Read-Only variable. Table 3-1 describes each item in the System Control Block in detail.

```
title 'System Control Block Definition for CP/M3 BIOS'
public @civec, @covec, @aivec, @aovec, @lovec, @bnkbf public @crdma, @crdsk, @vinfo,
@resel, @fx, @usrcd public @mltio, @ermde, @erdsk, @media, @bflgs public @date, @hour,
@min, @sec, ?erjmp, @mxtpa scb$base equ
0FE00H ; Base of the SCB
@CIVEC equ scb$base+22h     ; Console Input Redirection Vector (word, r/w)
@COVEC equ scb$base+24h     ; Console Output Redirection Vector (word, r/w)
@AIVEC equ scb$base+26h     ; Auxiliary Input Redirection Vector (word, r/w)
@AOVEC equ scb$base+28h     ; Auxiliary Output Redirection Vector (word, r/w)
@LOVEC equ scb$base+2Ah     ; List Output Redirection Vector (word, r/w)
@BNKBF equ scb$base+35h     ; Address of 128 Byte Buffer for Banked BIOS (word, r/o)
@CRDMA equ scb$base+3Ch     ; Current DMA Address (word, r/o)
@CRDSK equ scb$base+3Eh     ; Current Disk (byte, r/o)
@VINFO equ scb$base+3Fh     ; BDOS Variable "INFO" (word, r/o)
@RESEL equ scb$base+41h     ; FCB Flag (byte, r/o)
@FX equ scb$base+43h        ; BDOS Function for Error Messages (byte, r/o)
@USRCD equ scb$base+44h     ; Current User Code (byte, r/o)
@MLTIO equ scb$base+4Ah     ; Current Multisector Count (byte,r/w)
@ERMDE equ scb$base+4Bh     ; BDOS Error Mode (byte, r/o)
```

```
@ERDSK equ scb$base+51h    ; BDOS Error Disk (byte, r/o)
@MEDIA equ scb$base+54h    ; Set by BIOS to indicate open door (byte,r/w)
@BFLGS equ scb$base+57h    ; BDOS Message Size Flag (byte,r/o)
@DATE equ scb$base+58h     ; Date in Days Since 1 Jan 78 (word, r/w)
@HOUR equ scb$base+5Ah     ; Hour in BCD (byte, r/w)
@MIN equ scb$base+5Bh      ; Minute in BCD (byte, r/w)
@SEC equ scb$base+5Ch      ; Second in BCD (byte, r/w)
?ERJMP equ scb$base+5Fh    ; BDOS Error Message Jump (3 bytes, r/w)
@MXTPA equ scb$base+62h    ; Top of User TPA ; (address at 6,7)(word, r/o)
end
```

Listing 3-1. SCB.ASM File

The following table describes in detail each of the fields of the System Control Block.

Table 3-1. System Control Block Fields

Field	Meaning
@CIVEC, @COVEC, @AIVEC, @AOVEC, @LOVEC (Read-Write Variables)	These fields are the 16 bit I/O redirection vectors for the five logical devices: console input, console output, auxiliary input, auxiliary output, and the list device. (See Section 3.4.2, "Character I/O Functions.")
@BNKBF (Read-Only Variable)	@BNKBF contains the address of a 128 byte buffer in the resident portion of the BDOS in a banked system. This buffer is available for use during BOOT and WBOOT only. You can use it to transfer a copy of the CCP from an image in an alternate bank if the system does not support interbank moves.
@CRDMA, @FX, @USRCD, @ERDSK (Read-Only Variables)	These variables contain the current DMA address, the BDOS function number, the current user code, and the disk code of the drive on which the last error occurred. They can be displayed when a BDOS error is intercepted by the BIOS. See ?ERJMP.
@CRDSK (Read-Only Variable)	@CRDSK is the current default drive, set by BDOS Function 14.
@VINFO, @RESEL (Read-Only Variables)	If @RESEL is equal to OFFH then @VINFO contains the address of a valid FCB. If @RESEL is not equal to OFFH, then @VINFO is undefined. You can use @VINFO to display the filespec when the BIOS intercepts a BDOS error.
@MLTIO (Read-Write Variable)	@MLTIO contains the current multisector count. The BIOS can change the multisector count directly, or through BDOS Function 44. The value of the multisector count can range from 1 to 128.
@ERMDE (Read-Only Variable)	@ERMDE contains the current BDOS error mode. 0FFH indicates the BDOS is returning error codes to the application program without displaying any error messages. 0FEH indicates the BDOS is both displaying and returning errors. Any other value indicates the BDOS is displaying errors without notifying the application program.
@MEDIA (Read-Write Variable)	@MEDIA is global system flag indicating that a drive door has been opened. The BIOS routine that detects the open drive door sets this flag to 0FFH. The BIOS routine also sets the MEDIA byte in the Disk Parameter Header

Field	Meaning
	associated with the open-door drive to 0FFH.
@BFLGS **(Read-Only Variable)**	The BDOS in CP/M 3 produces two kinds of error messages: short error messages and extended error messages. Short error messages display one or two lines of text. Long error messages display a third line of text containing the filename, filetype, and BDOS Function Number involved in the error. In banked systems, GENCPM sets this flag in the System Control Block to indicate whether the BIOS displays short or extended error messages. Your error message handler should check this byte in the System Control Block. If the high- order bit, bit 7, is set to 0, the BDOS displays short error messages. if the high- order bit is set to 1, the BDOS displays the extended three-line error messages. For example, the BDOS displays the following error message if the BIOS returns an error from READ and the BDOS is displaying long error messages. CP/M Error on d: Disk I/O BDOS Function = nn File = filename.typ In the above error message, Function nn and filename.typ represent BDOS function number and file specification involved, respectively.
@DATE **(Read-Write Variable)**	The number of days since 31 December 1977, expressed as a 16-bit unsigned integer, low byte first. A real-time clock interrupt can update the @DATE field to indicate the current date.
@HOUR, @MIN, @SEC **(Read-Write Variables)**	These 2-digit Binary Coded Decimal (BCD) fields indicate the current hour, minute, and second if updated by a real-time clock interrupt.
?ERJMP **(Read-Write Code Label)**	The BDOS calls the error message subroutine through this jump instruction. Register C contains an error code as follows: 1 Permanent Error 2 Read Only Disk 3 Read Only File 4 Select Error 7 Password Error 8 File Exists 9 ? in Filename Error code 1 above results in the BDOS message Disk I/O. The ?ERJMP vector allows the BIOS to intercept the BDOS error messages so you can display them in a foreign language. Note that this vector is not branched to if the application program is expecting return codes on physical

Field	Meaning
	errors. Refer to the CP/M Plus Programmer's Guide for more information. ?ERJMP is set to point to the default (English) error message routine contained in the BDOS. The BOOT routine can modify the address at ?ERJMP+L to point to an alternate message routine. Your error message handler can refer to @FX, @VINFO (if @RESEL is equal to OFFH), @CRDMA, @CRDSK, and @USRCD to print additional error information. Your error handler should return to the BDOS with a RET instruction after printing the appropriate message.
@MXTPA **(Read-Only Variable)**	@MXTPA contains the address of the current BDOS entry point. This is also the address of the top of the TPA. The BOOT and WBOOT routines of the BIOS must use this address to initialize the BDOS entry JMP instruction at location 005H, during system initialization. Each time a RSX is loaded, @MXTPA is adjusted by the system to reflect the change in the available User Memory (TPA).

3.2 Character I/O Data Structures

TheBIOS data structure CHRTBL is a character table describing the physical I/O devices.

CHRTBL contains 6-byte physical device names and the characteristics of each physical device.

These characteristics include a mode byte, and the current baud rate, if any, of the device. The DEVICE utility references the physical devices through the names and attributes contained in your CHRTBL. DEVICE can also display the physical names and characteristics in your CHRTBL.

The mode byte specifies whether the device is an input or output device, whether it has a selectable baud rate, whether it is a serial device, and if XON/XOFF protocol is enabled.

Listing 3-2 shows a sample character device table that the DEVICE utility uses to set and display I/O direction.

```
; sample character device table chrtbl
db 'CRT '                               ; console VDT
db mb$in$out+mb$serial+mb$soft$baud
db baud$9600 db 'LPT '                  ; system serial printer
db mb$output+mb$serial+mb$soft$baud+mb$xon
db baud$9600 db 'TI810 '                ; alternate printer
db mb$output+mb$serial+mb$soft$baud
db baud$9600 db 'MODEM '                ; 300 baud modem port
db mb$in$out+mb$serial+mb$soft$baud
db baud$300 db 'VAX '                   ; interface to VAX 11/780
db mb$in$out+mb$serial+mb$soft$baud
db baud$9600 db 'DIABLO'                ; Diablo 630 daisy wheel printer
db mb$output+mb$serial+mb$soft$baud+mb$xon$xoff
db baud$1200 db 'CEN '                  ; Centronics type parallel printer
db mb$output
db baud$none
db 0                                    ; table terminator
```

Listing 3-2. Sample Character Device Table

Listing 3-3 shows the equates for the fields contained in the sample character device table. Many systems do not support all of these baud rates.

```
; equates for mode byte fields
mb$input        equ 0000$0001b            ; device may do input
mb$output       equ 0000$0010b            ; device may do output
mb$in$out       equ mb$input+mb$output    ; dev may do both
mb$soft$baud    equ 0000$0100b            ; software selectable baud rates
mb$serial       equ 0000$1000b            ; device may use protocol
mb$xon$xoff     equ 0001$0000b            ; XON/XOFF protocol
; equates for baud rate byte
baud$none       equ 0                     ; no baud rate associated with device
baud$50         equ 1                     ; 50 baud
baud$75         equ 2                     ; 75 baud
baud$110        equ 3                     ; 110 baud
baud$134        equ 4                     ; 134.5 baud
baud$150        equ 5                     ; 150 baud
baud$300        equ 6                     ; 300 baud
baud$600        equ 7                     ; 600 baud
baud$1200       equ 8                     ; 1200 baud
baud$1800       equ 9                     ; 1800 baud
baud$2400       equ 10                    ; 2400 baud
baud$3600       equ 11                    ; 3600 baud
baud$4800       equ 12                    ; 4800 baud
baud$7200       equ 13                    ; 7200 baud
baud$9600       equ 14                    ; 9600 baud
baud$19200      equ 15                    ; 19.2k baud
```

Listing 3-3. Equates for Mode Byte Bit Fields

3.3 BIOS Disk Data Structures

The BIOS includes tables that describe the particular characteristics of the disk subsystem used with CP/M 3. This section describes the elements of these tables.

In general, each disk drive has an associated Disk Parameter Header (DPH) that contains information about the disk drive and provides a scratche>ad area for certain BDOS operations.

One of the elements of this Disk Parameter Header is a pointer to the Disk Parameter Block (DPB), which contains the actual disk description.

In the banked system, only the Disk Parameter Block must reside in common memory.

The DPHS, checksum vectors, allocation vectors, Buffer Control Blocks, and Directory Buffers can reside in common memory or Bank 0. The hash tables can reside in common memory or any bank except Bank 1. The data buffers can reside in banked memory if you implement the XMOVE function.

Figure 3-1 shows the relationships between the drive table, the Disk Parameter Header, and the Data and Directory Buffer Control Block fields and their respective data structures and buffers.

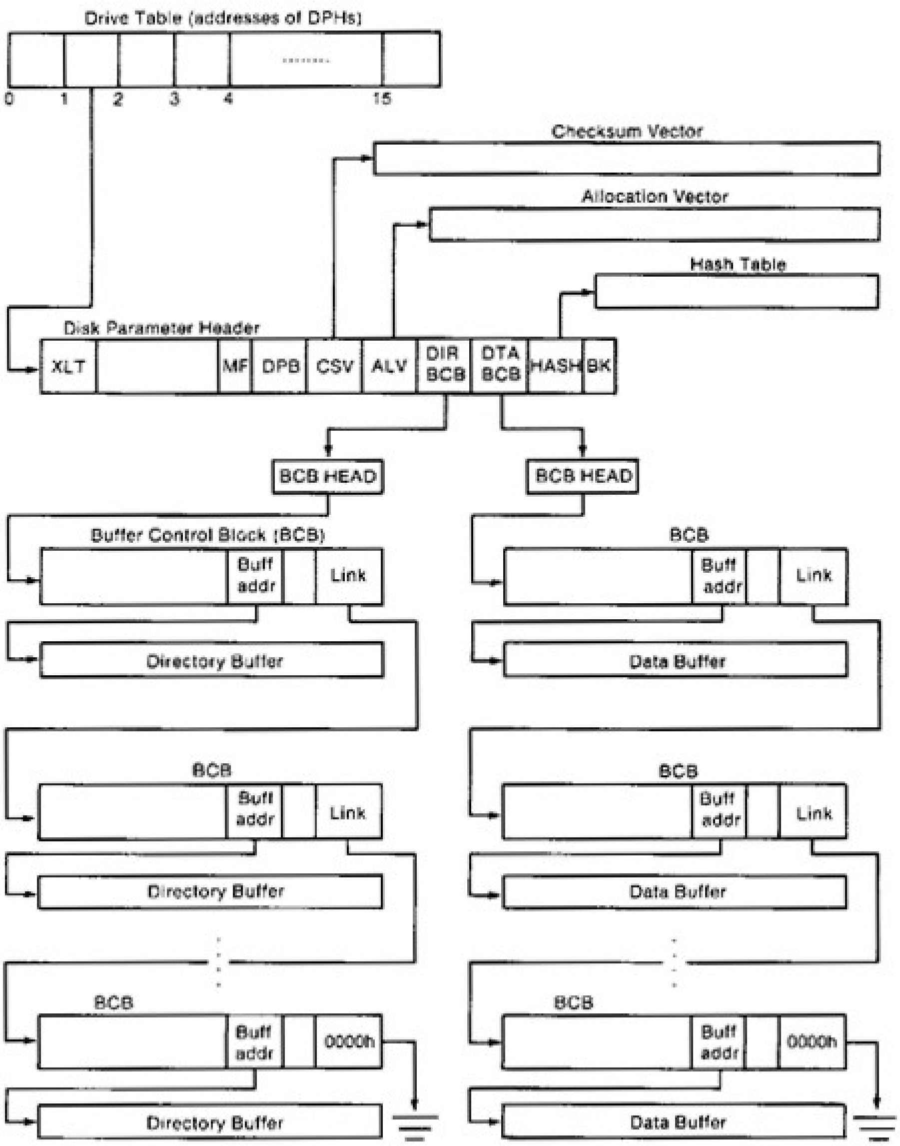

Figure 3-1. Disk Data Structures in a Banked System

3.3.1 Drive Table

The drive table consists of 16 words containing the addresses of the Disk Parameter Headers for each logical drive name, A through P, and takes the general form: drivetable dw dpho dw dphl dw dph2 dw dphf

If a logical drive does not exist in your system, the corresponding entry in the drive table must be zero.

The GENCPM utility accesses the drive table to locate the various disk parameter data structures, so that it can determine which system configuration to use, and optionally allocate the various buffers itself. You must supply a drive table if you want GENCPM to do this allocation.

If certain addresses in the Disk Parameter Headers referenced by this drive table are set to 0FFFEH, GENCPM allocates the appropriate data structures and updates the DPH. You can supply the drive table even if you have performed your own memory allocation. See the BIOS DRVTBL function described in Section 3.4.1.

3.3.2 Disk Parameter Header

In Figure 3-2, which shows the format of the Disk Parameter Header, b refers to bits.

XLT	-0-	MF	DPB	CSV	ALV	DIRBCB	DTABCB	HASH	HBANK
16b	72b	8b	16b	16b	16b	16b	16b	16b	8b

Figure 3-2. Disk Parameter Header Format

Table 3-2 describes the fields of the Disk Parameter Header.

Table 3-2. Disk Parameter Header Fields

Field	Comments
XLT	Set the XLT field to the address of the logical to hysical sector translation table. If there is no sector translation and the logical and physical sector numbers are the same, set XLT to 0000H. Disk drives with identical sector skew factors can share the same translation table. XLT is the value passed to SECTRN in registers DE. Usually the translation table consists of one byte per physical sector. Generally, it is advisable to keep the number of physical sectors per logical track to a reasonable value to prevent the translation table from becoming too large. In the case of disks with multiple heads, you can compute the head number from the track address rather than the sector address.
-0-	These 72 bits (9 bytes) of zeroes are the scratch area the BDOS uses to maintain various parameters associated with the drive.
MF	MF is the Media Flag. The BDOS resets MF to zero when the drive is logged in. The BIOS can set this flag and @MEDIA in the SCB to 0FFH if it detects that a drive door has been opened. If the flag is set to 0FFH, the BDOS checks for a media change prior to performing the next BDOS file operation on that drive. If the BDOS determines that the drive contains a new volume, the BDOS performs a login on that drive, and resets the MF flag to 00H. Note that the BDOS checks this flag only when a system call is made, and not during an operation. Usually, this flag is used only by systems that support door-open interrupts.
DPB	Set the DPB f ield to the address of a Disk Parameter Block that describes the characteristics of the disk drive. Several Disk Parameter Headers can address the same Disk Parameter Block if their drive characteristics are identical. (The Disk Parameter Block is described in Section 3.3.3.)
CSV	CSV is the address of a scratchpad area used to detect changed disks. This address must be different for each removable media Disk Parameter Header. There must be one byte for every 4 directory entries (or 128 bytes of directory). In other words, length(CSV) = (DRM/4)+l. (See Table 3-3 for an explanation of the DRM field.) If the drive is permanently mounted, set the CKS variable in the DPB to 8000H and set CSV to 0000H. This way, no storage is reserved for a checksum vector. The checksum vector may be located in common memory or in Bank 0. Set CSV to 0FFFEH for GENCPM to set up the checksum vector.
ALV	ALV is the address of the scratchpad area called the allocation vector, which the BDOS uses to keep disk storage allocation information.

	This area must be unique for each drive. The allocation vector usually requires 2 bits for each block on the drive. Thus, length(ALV) = (DSM/4) + 2. (See Table 3-3 for an explanation of the DSM field.) In the nonbanked version of CP/M 3, you can optionally specify that GENCPM reserve only one bit in the allocation vector per block on the drive. In this case, length(ALV) = (DSM/8) + The GENCPM option to use single-bit allocation vectors is provided in the nonbanked version of CP/M 3 because additional memory is required by the double-bit allocation vector. This option applies to all drives on the system. With double-bit allocation vectors, CP/M 3 automatically frees, at every system warm start, all file blocks that are not permanently recorded in the directory. Note that file space allocated to a f ile is not permanently recorded in a directory unless the file is closed. Therefore, the allocation vectors in memory can indicate that space is allocated although directory records indicate that space is free for allocation. With single-bit allocation vectors, CP/M 3 requires that a drive be reset before this space can be reclaimed. Because it increases performance, CP/M 3 does not reset disks at system warm start. Thus, with single-bit allocation vectors, if you do not reset the disk system, DIR and SHOW can report an inaccurate amount of free space. With single-bit ALV allocation vectors, the user must type a CTRL-C at the system prompt to reset the disk system to ensure accurate reporting of free space. Set ALV to 0FFFEH for GENCPM to automatically assign space for the allocation vector, single- or double-bit, during system generation. In the nonbanked system, GENCPM prompts for the type of allocation vector. In the banked system, the allocation vector is always double-bit and can reside in common memory or Bank 0. When GENCPM automatically assigns space for the allocation vector (ALV = 0FFFEH) , it places the allocation vector in Bank 0.
DIRBCB	Set DIRBCB to the address of a single directory Buffer Control Block (BCB) in an unbanked system. Set DIRBCB to the address of a BCB list head in a banked system. Set DIRBCB to 0FFFEH for GENCPM to set up the DIRBCB field. The BDOS uses directory buffers for all accesses of the disk directory. Several DPHs can refer to the same directory BCB or BCB list head; or, each DPH can reference an independent BCB or BCB list head. Section 3.3.4 describes the format of the Buffer Control Block.
DTABCB	Set DTABCB to the address of a single data BCB in an unbanked system. Set DTABCB to the address of a data BCB list head in a banked system. Set DTABCB to 0FFFEH for GENCPM to set up the DTABCB field. The BDOS uses data buffers to hold physical sectors so that it can block and deblock logical 128-byte records. If the physical record size of the media associated with a DPH is 128 bytes, you can set the DTABCB field of the DPH to 0FFFFH, because in this case, the BDOS does not use a data buffer.
HASH	HASH contains the address of the optional directory hashing table associated with a DPH.

	Set HASH to 0FFFFH to disable directory hashing. Set RASH to 0FFFEH to make directory hashing on the drive a GENCPM option. Each DPH using hashing must reference a unique hash table. If a hash table is supplied, it must be 4*(DRM+l) bytes long, where DRM is one less than the length of the directory. In other words, the hash table must contain four bytes for each directory entry of the disk.
HBANK	Set HBANK to the bank number of the hash table. HBANK is not used in unbanked systems and should be set to zero. The hash tables can be contained in the system bank, common memory, or any alternate bank except Bank 1, because hash tables cannot be located in the Transient Program Area. GENCPM automatically sets HBANK when HASH is set to 0FFFEH.

3.3.3 Disk Parameter Block

Figure 3-3 shows the format of the Disk Parameter Block, where b refers to bits.

SPT	BSH	BLM	EXM	DSM	DRM	ALO	AL1	CKS	OFF	PSH	PHM
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b	8b	8b

Figure 3-3. Disk Parameter Block Format

Table 3-3 describes the fields of the Disk Parameter Block.

Table 3-3. Disk Parameter Block Fields

Field	Comments
SPT	Set SPT to the total number of 128-byte logical records per track.
BSH	Data allocation block shift factor. The value of BSH is determined by the data block allocation size.
BLM	Block mask. The value of BLM is determined by the data block allocation size.
EXM	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. DSM is one less than the total number of blocks on the drive.
DRM	Total number of directory entries minus one that can be stored on this drive. The directory requires 32 bytes per entry.
ALO, AL1	Determine reserved directory blocks. See Figure 3-4 for more information.
CKS	The size of the directory check vector, @DRM/4)+l. Set bit 15 of CKS to 1 if the drive is permanently mounted. Set CKS to 8000H to indicate that the drive is permanently mounted and directory checksumming is not required. Note: full directory checksumming is required on removable media to support the automatic login feature of CP/M 3.
OFF	The number of reserved tracks at the beginning of the logical disk. 0FF is the track on which the directory starts.
PSH	Specifies the physical record shift factor.

PHM	Specifies the physical record mask.

CP/M allocates disk space in a unit called a block. Blocks are also called allocation units, or clusters. BLS is the number of bytes in a block. The block size can be 1024, 2048, 4096, 8192, or 16384 (decimal) bytes.

A large block size decreases the size of the allocation vectors but can result in wasted disk space. A smaller block size increases the size of the allocation vectors because there are more blocks on the same size disk.

There is a restriction on the block size. If the block size is 1024, there cannot be more than 255 blocks present on a logical drive. In other words, if the disk is larger than 256K, it is necessary to use at least 2048 byte blocks.

The value of BLS is not a field in the Disk Parameter Block; rather, it is derived from the values of BSH and BLM as given in Table 3-4.

Table 3-4. BSH and BLM Values

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

The block mask, BLM, equals one less than the number of 128- byte records in an allocation unit, (BLS/128 - 1), or (2**BSH)-1.

The value of the Block Shift Factor, BSH, is determined by the data block allocation size. The Block Shift Factor (BSH) equals the logarithm base two of the block size in 128-byte records, or LOG2 (BLS/128) , where LOG2 represents the binary logarithm function.

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 Table 3-5.

Table 3-5. Maximum EXK Values

BLS	EXM values	
	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 EXM is one less than the maximum number of 16K extents per FCB.

Set EXM to zero if you want media compatibility with an extended CP/M 1.4 system.

This only applies to double-density CP/M 1.4 systems, with disk sizes greater than 256K bytes.

It is preferable to copy double-density 1.4 disks to single-density, then reformat them and recreate them with the CP/M 3 system, because CP/M 3 uses directory entries more effectively than CP/M 1.4.

DSM is one less than the total number of blocks on the drive. DSM must be less than or equal to 7FFFH. If the disk uses 1024 byte blocks (BSH=3, BLM=7), DSM must be less than or equal to 00FFH. The product BLS*(DSM+l) is the total number of bytes the drive holds and must be within the capacity of the physical disk. It does not include the reserved operating system tracks.

The DRM entry is one less than the total number of 32-byte directory entries, and is a 16-bit value. DRM must be less than or equal to (BLS/32 * 16) - 1. DRM determines the values of AL0 and ALI. The two fields AL0 and ALI can together be considered a string of 16 bits, as shown in Figure 3-4.

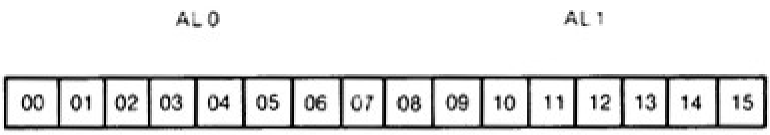

Figure 3-4. ALO and ALI

Position 00 corresponds to the high-order bit of the byte labeled AL0, and position 15 corresponds to the low-order bit of the byte labeled ALI. Each bit position reserves a data block for a number of directory entries, thus allowing a maximum of 16 data blocks to be assigned for directory entries. Bits are assigned starting at 00 and filled to the right until position 15.

AL0 and ALI overlay the first two bytes of the allocation vector for the associated drive. Table 3-6 shows DRM maximums for the various block sizes.

Table 3-6. BLS and Number of Directory Entries

BLS	Directory Entries	Maximum DRM
1,024	32 * reserved blocks	511
2,048	64 * reserved blocks	1,023
4,096	128 * reserved blocks	2,047
8,192	256 * reserved blocks	4,095
16,384	512 * reserved blocks	8,191

If DRM = 127 (128 directory entries) , and BLS = 1024, 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 = OF0H and AL1 = 00H. The maximum directory allocation is 16 blocks where the block size is determined by BSH and BLM.

The OFF field determines the number of tracks that are skipped at the beginning of the physical disk. It can be used as a mechanism for skipping reserved operating system tracks, which on system disks contain the Cold Boot Loader, CPMLDR, and possibly the CCP. It is also used to partition a large disk into smaller segmented sections.

PSH and PHM determine the physical sector size of the disk. All disk I/O is in terms of the physical sector size. Set PSH and PSM to zero if the BIOS is blocking and deblocking instead of the BDOS.

PSH specifies the physical record shift factor, ranging from 0 to 5, corresponding to physical record sizes of 128, 256, 512, 1K, 2K, or 4K bytes. It is equal to the logarithm base two of the physical record size divided by 128, or LOG2(sector-size/128). See Table 3-7 for PSH values.

PHM specifies the physical record mask, ranging from 0 to 31, corresponding to physical record sizes of 128, 256, 512, 1K, 2K, or 4K bytes. It is equal to one less than the sector size divided by 128, or, (sector-size/128)-1. See Table 3-7 for PHM values.

Table 3-7. PSH and PHN Values

Sector size	PSH	PHM
128	0	0
256	1	1
512	2	3
1,024	3	7
2,048	4	15
4,096	5	31

3.3.4 Buffer Control Block

A Buffer Control Block (BCB) locates physical record buffers for the BDOS. The BDOS uses the BCB to manage the physical record buffers during processing. More than one Disk Parameter Header can specify the same BCB. The GENCPM utility can create the Buffer Control Block.

Note that the BANK and LINK fields of the Buffer Control Block are present only in the banked system. Therefore, the Buffer Control Block is twelve bytes long in the nonbanked system, and fifteen bytes long in the banked system. Note also that only the DRV, BUFFAD, BANK, and LINK fields need to contain initial values. In Figure 3-5, which shows the form of the Buffer Control Block, b refers to bits.

DRV	REC#	WFLG	00	TRACK	SECTOR	BUFFAD	BANK	LINK
8b	24b	8b	8b	16b	16b	16b	8b	16b

Figure 3-5. Buffer Control Block Format

Table 3-8 describes the fields of each Buffer Control Block.

Table 3-8. Buffer Control Block Fields

Field	Comment
DRV	Identifies the disk drive associated with the record contained in the buffer located at address BUFFAD. If you do not use GENCPM to allocate buffers, you must set the DRV field to 0FFH.
REC#	Identifies the record position of the current contents of the buffer located at address BUFFAD. REC# consists of the absolute sector number of the record where the first record of the directory is zero.
WFLG	Set by the BDOS to OFFH to indicate that the buffer contains new data that has not yet

	been written to disk. When the data is written, the BDOS sets the WFLG to zero to indicate the buffer is no longer dirty.
00	Scratch byte used by BDOS.
TRACK	Contains the physical track location of the contents of the buffer.
SECTOR	Contains the physical sector location of the contents of the buffer.
BUFFAD	Specifies the address of the buffer associated with this BCB.
BANK	Contains the bank number of the buffer associated with this BCB. This field is only present in banked systems.
LINK	Contains the address of the next BCB in a linked list, or zero if this is the last BCB in the linked list. The LINK field is present only in banked systems.

The BDOS distinguishes between two kinds of buffers: data buffers referenced by DTABCB, and directory buffers referenced by DIRBCB. In a banked system, the DIRBCB and DTABCB fields of a Disk Parameter Header each contain the address of a BCB list head rather than the address of an actual BCB. A BCB list head is a word containing the address of the first BCB in a linked list. If several DPHs reference the same BCB list, they must reference the same BCB list head. Each BCB has a LINK field that contains the address of the next BCB in the list, or zero if it is the last BCB.

In banked systems, the one-byte BANK field indicates the bank in which the data buffers are located. The BANK field of directory BCBs must be zero because directory buffers must be located in Bank 0, usually below the banked BDOS module, or in common memory. The BANK field is for systems that support direct memory-to-memory transfers from one bank to another. (See the BIOS XMOVE entry point in section 3.4.4.)

The BCD data structures in a banked system must reside in Bank 0 or in common memory.

The buffers of data BCBs can be located in any bank except Bank I (the Transient Program Area).

For banked systems that do not support interbank block moves through XMOVE, the BANK field must be set to 0 and the data buffers must reside in common memory. The directory buffers can be in Bank 0 even if the system does not support bank-to-bank moves.

In the nonbanked system, the DPH, DIRBCB, and DTABCB can point to the same BCB if the DPH defines a fixed media device. For devices with removable media, the DPH DIRBCB and the DPH DTABCB must reference different BCBS. In banked systems, the DPH DIRBCB and DTABCB must point to separate list heads.

In general, you can enhance the performance of CP/M 3 by allocating more BCBS, but the enhancement reduces the amount of TPA memory in nonbanked systems.

If you set the DPH DIRBCB or the DPH DTABCB fields to 0FFFEH, the GENCPM utility creates BCBS, allocates physical record buffers, and sets these f ields to the address of the BCBS.

This allows you to write device drivers without regard to buffer requirements.

3.3.5 Data Structure Macro Definitions

Several macro definitions are supplied with CP/M 3 to simplify the creation of some of the data structures in the BIOS. These macros are defined in the library file CPM3.LIB on the distribution disk.

To reference these macros in your BIOS, include the following statement:

MACLIB CPM3

DTBL Macro

Use the DTBL macro to generate the drive table, DRVTBL. It has one parameter, a list of the DPHs in your system. The list is enclosed in angle brackets.

The form of the DTBL macro call is label: DTBL <DPHA,DPHB,...,DPHP> where DPHA is the address of the DPH for drive A, DPHB is the address of the DPH for drive B, up to drive P. For example, DRVTBL: DTBL <ACSHDO,FDSDO,FDSD1>

This example generates the drive table for a three-drive system. The DTBL macro always generates a sixteen-word table, even if you supply fewer DPH names.

The unused entries are set to zero to indicate the corresponding drives do not exist.

DPH Macro

The DPH macro routine generates a Disk Parameter Header (DPH) . It requires two parameters: the address of the skew table for this drive, and the address of the Disk Parameter Block (DPB) . Two parameters are optional: the maximum size of the checksum vector, and the maximum size of the allocation vector. If you omit the maximum size of the checksum vector and the maximum size of the allocation vector from the DPH macro invocation, the corresponding fields of the Disk Parameter Header are set to 0FFFEH so that GENCPM automatically allocates the vectors.

The form of the DPH macro call is label: DPH ?trans,?dpb [,?csize] [,?asize] where:

?trans is the address of the translation vector for this drive;
?dpb is the address of the DPB for this drive;
?csize is the maximum size in bytes of the checksum vector;
?asize is the maximum size in bytes of the allocation vector.

The following example, which includes all four parameters, shows a typical DPH macro invocation for a standard single-density disk drive:

```
FDSDO: DPH SKEW6,DPB$SD,16,31
```

SKEW Macro

The SKEW macro generates a skew table and requires the following parameters: the number of physical sectors per track, the skew factor, and the first sector number on each track (usually 0 or 1).

The form of the SKEW macro call is label: SKEW ?secs,?skf,?fsc where:

?secs is the number of physical sectors per track;
?skf is the sector skew factor;
?fsc is the first sector number on each track.

The following macro invocation generates the skew table for a standard single-density disk drive.

```
SKEW6: SKEW 26,6,1
```

DPB Macro

The DPB macro generates a Disk Parameter Block specifying the characteristics of a drive type. It requires six parameters: the physical sector size in bytes, the number of physical sectors per track, the total number of tracks on the drive, the size of an allocation unit in bytes, the number of directory entries desired, and the number of system tracks to reserve at the beginning of the drive. There is an optional seventh parameter that defines the CKS field in the DPB. If this parameter is missing, CKS is calculated from the directory entries parameter.

The form of the DPB macro call is label: DPB ?psize,?pspt,?trks,?bls,?ndirs,?off[,?ncks] where:

?psize is the physical sector size in bytes;
?pspt is the number of physical sectors per track;
?trks is the number of tracks on the drive;
?bls is the allocation unit size in bytes;
?ndirs is the number of directory entries;
?off is the number of tracks to reserve;
?ncks is the number of checked directory entries.

The following example shows the parameters for a standard single-density disk drive:

```
DPB$SD: DPB 128,26,77,1024,64,2
```

The DPB macro can be used only when the disk drive is under eight megabytes. DPBs for larger disk drives must be constructed by hand.

3.4 BIOS Subroutine Entry Points

This section describes the entry parameters, returned values, and exact responsibilities of each BIOS entry point in the BIOS jump vector. The routines are arranged by function.

Section 3.4.1 describes system initialization. Section 3.4.2 presents the character I/O functions, followed by Section 3.4.3, discussing the disk I/O functions. Section 3.4.4 discusses the BIOS memory select and move functions. The last section, 3.4.5, discusses the BIOS clock support function. Table 3-9 shows the BIOS entry points the BDOS calls to perform each of the four categories of system functions.

Table 3-9. Functional Organization of BIOS Entry Points

Operation	Function
System Initialization	BOOT, WBOOT, DEVTBL, DEVINI, DRVTBL,
Character I/O	CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, AUXOST
Disk I/O	HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, FLUSH
Memory Selects and Moves	MOVE, XMOVE, SELMEM, SETBNK
Clock Support	TIME

Table 3-10 is a summary showing the CP/M 3 BIOS function numbers, jump instruction names, and the entry and return parameters of each jump instruction in the table, arranged according to the BIOS function number.

Table 3-10. CP/M 3 BIOS Function Jump Table Sunmary

No.	Function	Input	Output

No.	Function	Input	Output
0	BOOT	None	None
1	WBOOT	None	None
2	CONST	None	A=0FFH if ready A=00H if not ready
3	CONIN	None	A=Console Character
4	CONOUT	C=Con Char	None
5	LIST	C=Char	None
6	AUXOUT	C=Char	None
7	AUXIN	None	A=Char
8	HOME	None	None
9	SELDSK	C=Drive 0-15 HL=DPH addr E=Init Sel Flag	HL=0000H if invalid drive
10	SETTRK	BC=Track No	None
11	SETSEC	BC=Sector No	None
12	SETDMA	BC=.DMA	None
13	READ	None	A=00H if no Error A=01H if Non-recov Err A=0FFH if media changed
14	WRITE	C=Deblk Code	A=00H if no Error A=01H if Physical Error A=02H if Disk is R/O A=0FFH if media changed
15	LISTST	None	A=00H if not ready A=0FFH if ready
16	SECTRN	BC=Log Sect No DE=Trans Tbl Adr	HL=Physical Sector Number
17	CONOST	None	A=00H if not ready A=0FFH if ready
18	AUXIST	None	A=0H if not ready A=0FFH if ready
19	AUXOST	None	A=00H if not ready A=0FFH if ready
20	DEVTBL	None	HL=Chrtbl addr
21	DEVINI	C=Dev No 0-15	None
22	DRVTBL	None	HL=Drv Tbl addr HL=0FFFFH HL=0FFFEH

No.	Function	Input	Output
23	MULTIO	C=Mult Sec Cnt	None
24	FLUSH	None	A=00H if no err A=01H if phys err A=02H if disk R/O
25	MOVE	BC=Count DE=Source Adr HL=Dest Adr	HL & DE point to next bytes following MOVE
26	TIME	C=Get/Set Flag	None
27	SELMEM	A=Mem Bank	None
28	SETBNK	A=Mem Bank	None
29	XMOVE	B=Dest Bank C=Source Bank	None
30	USERF	Reserved for System Implementor	
31	RESERV1	Reserved for Future Use	
32	RESERV2	Reserved for Future Use	

3.4.1 System Initialization Functions

This section defines the BIOS system initialization routines BOOT, WBOOT, DEVTBL, DEVINI, and DRVTBL.

BIOS Function 0: BOOT

Get Control from Cold Start Loader and Initialize System

Entry Parameters: None

Returned Values: None

The BOOT entry point gets control from the Cold Start Loader in Bank 0 and is responsible for basic system initialization. Any remaining hardware initialization that is not done by the boot ROMS, the Cold Boot Loader, or the LDRBIOS should be performed by the BOOT routine.

The BOOT routine must perform the system initialization outlined in Section 2.3, "System Initialization." This includes initializing Page Zero jumps and loading the CCP.

BOOT usually prints a sign-on message, but this can be omitted. Control is then transferred to the CCP in the TPA at 0100H.

To initialize Page Zero, the BOOT routine must place a jump at location 0000H to BIOS base + 3, the BIOS warm start entry point. The BOOT routine must also place a jump instruction at location 0005H to the address contained in the System Control Block variable, @MXTPA.

The BOOT routine must establish its own stack area if it calls any BDOS or BIOS routines. In a banked system, the stack is in Bank 0 when the Cold BOOT routine is entered.

The stack must be placed in common memory.

BIOS Function 1: WBOOT

Get Control When a Warm Start Occurs

Entry Parameters: None

Returned Values: None

The WBOOT entry point is entered when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H or attempts to return to the CCP. The WBOOT routine must perform the system initialization outlined in BIOS Function 0, including initializing Page zero jumps and loading the CCP.

When your WBOOT routine is complete, it must transfer control to the CCP at location 0100H in the TPA.

Note that the CCP does not reset the disk system at warm start. The CCP resets the disk system when a CTRL-C is pressed following the system prompt.

Note also that the BIOS stack must be in common memory to make BDOS function calls.

Only the BOOT and WBOOT routines can perform BDOS function calls.

If the WBOOT routine is reading the CCP from a file, it must set the multisector I/O count, @MLTIO in the System Control Block, to the number of 128-byte records to be read in one operation before reading CCP.COM. You can directly set @MLTIO in the SCB, or you can call BDOS Function 44 to set the multisector count in the SCS.

If blocking/deblocking is done in the BIOS instead of in the BDOS, the WBOOT routine must discard all pending buffers.

BIOS Function 20: DEVTBL

Return Address of Character I/O Table

Entry Parameters: None

Returned Values:

```
HL=address of Chrtbl
```

The DEVTBL and DEVINI entry points allow you to support device assignment with a flexible, yet completely optional system. It replaces the IOBYTE facility of CP/M 2.2. Note that the CHRTBL must be in common in banked systems.

BIOS Function 21: DEVINI

Initialize Character I/O Device

Entry Parameters:

```
C=device number, 0-15
```

Returned Values: None

The DEVINI routine initializes the physical character device specified in register C to the baud rate contained in the appropriate entry of the CHRTBL. It need only be supplied if I/O redirection has been implemented and is referenced only by the DEVICE utility supplied with CP/M 3.

BIOS Function 22: DRVTBL

Return Address of Disk Drive Table

Entry Parameters: None

Returned Values:

`HL=Address of Drive Table of Disk Parameter Headers (DPH);`
Hashing can utilized if specified by the DPHs Referenced by this DRVTBL.

`HL=0FFFFH if no Drive Table; GENCPM does not set up buffers.`

Hashing is supported.

`HL=0FFFEH if no Drive Table; GENCPM does not set up buffers.`

Hashing is not supported.

The first instruction of this subroutine must be an LXI H,<address> where <address> is one of the above returned values. The GENCPM utility accesses the address in this instruction to locate the drive table and the disk parameter data structures to determine which system configuration to use.

If you plan to do your own blocking/deblocking, the first instruction of the DRVTBL routine must be the following: lxi h,0FFFEh

You must also set the PSH and PSM fields of the associated Disk Parameter Block to zero.

3.4.2 Character I/O Functions

This section defines the CP/M 3 character I/O routines CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, and AUXOST.

CP/M 3 assumes all simple character I/O operations are performed in eight-bit ASCII, upper and lowercase, with no parity. ANASCII CTRL-Z (lAH) denotes an end-of-file condition for an input device.

In CP/M 3, you can direct each of the five logical character devices to any combination of up to twelve physical devices. Each of the five logical devices has a 16-bit vector in the System Control Block (SCB) . Each bit of the vector represents a physical device where bit 15 corresponds to device zero, and bit 4 is device eleven. Bits 0 through 3 are reserved for future system use.

You can use the public names defined in the supplied SCB.ASM file to reference the I/O redirection bit vectors. The names are shown in Table 3-11.

Table 3-11. I/O Redirection Bit Vectors in SCB

Name	Logical Device
@CIVEC	Console Input
@COVEC	Console Output

@AIVEC	Auxiliary Input
@AOVEC	Auxiliary Output
@LOVEC	List Output

You should send an output character to all of the devices whose corresponding bit is set.

An input character should be read from the first ready device whose corresponding bit is set.

An input status routine should return true if any selected device is ready. An output status routine should return true only if all selected devices are ready.

BIOS Function 2: CONST

Sample the Status of the Console Input Device

Entry Parameters: None

Returned value:

```
A=0FFH if a console character is ready to read
A=00H if no console character is ready to read
```

Read the status of the currently assigned console device and return 0FFH in register A if a character is ready to read, and 00H in register A if no console characters are ready.

BIOS Function 3: CONIN

Read a Character from the Console

Entry Parameters: None

Returned Values:

```
A=Console Character
```

Read the next console character into register A with no parity. If no console character is ready, wait until a character is available'before returning.

BIOS Function 4: CONOUT

Output Character to Console

Entry Parameters:

```
C=Console Character
```

Returned Values: None

Send the character in register C to the console output device. The character is in ASCII with no parity.

Character I/O Functions

BIOS Function 5: LIST

Output Character to List Device

Entry Parameters:

```
C=Character
```

Returned Values: None

Send the character from register C to the listing device. The character is in ASCII with no parity.

BIOS Function 6: AUXOUT

Output a Character to the Auxiliary Output Device

Entry Parameters:

```
C=Character
```

Returned Values: None

Send the character from register C to the currently assigned AUXOUT device. The character is in ASCII with no parity.

BIOS Function 7: AUXIN

Read a Character from the Auxiliary Input Device

Entry Parameters: None

Returned Values:

```
A=Character
```

Read the next character from the currently assigned AUXIN device into register A with no parity. A returned ASCII CTRL-Z (lAH) reports an end-of-file.

BIOS Function 15: LISTST

Return the Ready Status of the List Device

Entry Parameters: None

Returned Values:

```
A=00H if list device is not ready to accept a character
A=0FFH if list device is ready to accept a character
```

The BIOS LISTST function returns the ready status of the list device.

BIOS Function 17: CONOST

Return Output Status of Console

Entry Parameters: None

Returned Values:

```
A=0FFH if ready
A=00H if not ready
```

The CONOST routine checks the status of the console. CONOST returns an OFFH if the console is ready to display another character. This entry point allows for full polled handshaking communications support.

BIOS Function 18: AUXIST

Return Input Status of Auxiliary Port

Entry Parameters: None

Returned Values:

```
A=0FFH if ready
A=00H if not ready
```

The AUXIST routine checks the input status of the auxiliary port. This entry point allows full polled handshaking for communications support using an auxiliary port.

BIOS Function 19: AUXOST

Return Output Status of Auxiliary Port

Entry Parameters: None

Returned Values:

```
A=0FFH if ready
A=00H if not ready
```

The AUXOST routine checks the output status of the auxiliary port. This routine allows full polled handshaking for communications support using an auxiliary port.

3.4.3 Disk I/O Functions

This section defines the CP/M 3 BIOS disk I/O routines HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, and FLUSH.

BIOS Function 8: HOME

Select Track 00 of the Specified Drive

Entry Parameters: None

Returned Values: None

Return the disk head of the currently selected disk to the track 00 position. Usually, you can translate the HOME call into a call on SETTRK with a parameter of 0.

BIOS Function 9: SELDSK

Select the Specified Disk Drive

Entry Parameters:

```
C=Disk Drive (0-15)
E=Initial Select Flag
```

Returned Values:

```
HL=Address of Disk Parameter
Header (DPH) if drive exists
HL=0000H if drive does not exist
```

Select the disk drive specified in register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so on to 15 for drive P. On each disk select, SELDSK must return in HL the base address of a 25-byte area called the Disk Parameter Header. If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator.

On entry to SELDSK, you can determine if it is the first time the specified disk is selected. Bit 0, the least significant bit in register E, is set to 0 if the drive has not been previously selected. This information is of interest in systems that read configuration information from the disk to set up a dynamic disk definition table.

When the BDOS calls SELDSK with bit 0 in register E set to 1, SELDSK must return the same Disk Parameter Header address as it returned on the initial call to the drive. SELDSK can only return a 00H indicating an unsuccessful select on the initial select call.

SELDSK must return the address of the Disk Parameter Header on each call. Postpone the actual physical disk select operation until a READ or WRITE is performed, unless I/O is required for automatic density sensing.

BIOS Function 10: SETTRK

Set Specified Track Number

Entry Parameters:

```
BC=Track Number
```

Returned Values: None

Register BC contains the track number for a subsequent disk access on the currently selected drive. Normally, the track number is saved until the next READ or WRITE occurs.

BIOS Function 11: SETSEC

Set Specified Sector Number

Entry Parameters:

```
BC=Sector Number
```

Returned Values: None

Register BC contains the sector number for the subsequent disk access on the currently selected drive. This number is the value returned by SECTRN. Usually, you delay actual sector selection until a READ or WRITE operation occurs.

BIOS Function 12: SETDMA

Set Address for Subsequent Disk I/O

Entry Parameters:

```
BC=Direct Memory Access Address
```

Returned Values: None

Register BC contains the DMA (Direct Memory Access) address for the subsequent READ or WRITE operation. For example, if B = 00H and C = 80H when the BDOS calls SETDMA, then the subsequent read operation reads its data starting at 80H, or the subsequent write operation gets its data from 80H, until the next call to SETDMA occurs.

BIOS Function 13: READ

Read a Sector from the Specified Drive

Entry Parameters: None

Returned Values:

```
A=000H if no errors occurred
A=001H if nonrecoverable error condition occurred
A=0FFH if media has changed
```

Assume the BDOS has selected the drive, set the track, set the sector, and specified the DMA address. The READ subroutine attempts to read one sector based upon these parameters, then returns one of the error codes in register A as described above.

If the value in register A is 0, then CP/M 3 assumes that the disk operation completed properly. If an error occurs, the BIOS should attempt several retries to see if the error is recoverable before returning the error code.

If an error occurs in a system that supports automatic density selection, the system should verify the density of the drive. If the density has changed, return a OFFH in the accumulator.

This causes the BDOS to terminate the current operation and relog in the disk.

BIOS Function 14: WRITE

Write a Sector to the Specified Disk

Entry Parameters:

```
C=Deblocking Codes
```

Returned Values:

```
A=00H if no error occurred
A=001H if physical error occurred
A=002H if disk is Read-Only
A=0FFH if media has changed
```

Write the data from the currently selected DMA address to the currently selected drive, track, and sector. Upon each call to WRITE, the BDOS provides the following information in register C:

0 = deferred write
1 = nondeferred write
2 = deferred write to the first sector of a new data block

This information is provided for those BIOS implementations that do blocking/deblocking in the BIOS instead of the BDOS.

As in READ, the BIOS should attempt several retries before reporting an error.

If an error occurs in a system that supports automatic density selection, the system should verify the density of the drive. If the density has changed, return a 0FFH in the accumulator. This causes the BDOS to terminate the current operation and relog in the disk.

BIOS Function 16: SECTRN

Translate Sector Number Given Translate Table

Entry Parameters:

```
BC=Logical Sector Number
DE=Translate Table Address
```

Returned Values:

```
HL=Physical Sector Number
```

SECTRN performs logical sequential sector address to physical sector translation to improve the overall response of CP/M 3. Digital Research ships standard CP/M disk with a skew factor of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs on a slow system to process their buffers without missing the next sector. In computer systems that use fast processors, memory, and disk subsystems, you can change the skew factor to improve overall response.

Typically, most disk systems perform well with a skew of every other physical sector. You should maintain support of single-density, IBM 3740 compatible disks using a skew factor of 6 in your CP/M 3 system to allow information transfer to and from other CP/M users.

SECTRN receives a logical sector number in BC, and a translate table address in DE. The logical sector number is relative to zero. The translate table address is obtained from the Disk Parameter Block for the currently selected disk. The sector number is used as an index into the translate table, with the resulting physical sector number returned in HL. For standard, single-density, eight- inch disk systems, the tables and indexing code are provided in the sample BIOS and need not be changed.

Certain drive types either do not need skewing or perform the skewing externally from the system software. In this case, the skew table address in the DPH can be set to zero, and the SECTRN routine can check for the zero in DE and return with the physical sector set to the logical sector.

BIOS Function 23: MULTIO

Set Count of Consecutive Sectors for READ or WRITE

Entry Parameters:

```
C=Multisector Count
```

Returned Values: None

To transfer logically consecutive disk sectors to or from contiguous memory locations, the BDOS issues a MULTIO call, followed by a series of READ or WRITE calls. This allows the BIOS to transfer multiple sectors in a single disk operation. The maximum value of the sector count is dependent on the physical sector size, ranging from 128 with 128-byte sectors, to 4 with 4096-byte sectors. Thus, the BIOS can transfer up to 16K directly to or from the TPA with a single operation.

The BIOS can directly transfer all of the specified sectors to or from the DMA buffer in one operation and then count down the remaining calls to READ or WRITE.

If the disk format uses a skew table to minimize rotational latency when single records are transferred, it is more difficult to optimize transfer time for multisector transfers.

One way of utilizing the multisector count with a skewed disk format is to place the sector numbers and associated DMA addresses into a table until either the residual multisector count reaches zero, or the track number changes. Then you can sort the saved requests by physical sector to allow all of the required sectors on the track to be read in one rotation. Each sector must be transferred to or from its proper DMA address.

When an error occurs during a multisector transfer, you can either reset the multiple sector counters in the BIOS and return the error immediately, or you can save the error status and return it to the BDOS on the last READ or WRITE call of the MULTIO operation.

BIOS Function 24: FLUSH

Force Physical Buffer Flushing for User-supported Deblocking

Entry Parameters: None

Returned Values:

```
A=00H if no error occurred
A=001H if physical error occurred
A=002H if disk is Read-Only
```

The flush buffers entry point allows the system to force physical sector buffer flushing when your BIOS is performing its own record blocking and deblocking.

The BDOS calls the FLUSH routine to ensure that no dirty buffers remain in memory. The BIOS should immediately write any buffers that contain unwritten data.

Normally, the FLUSH function is superfluous, because the BDOS supports blocking/deblocking internally. It is required, however, for those systems that support blocking/deblocking in the BIOS, as many CP/M 2.2 systems do.

Note: if you do not implement FLUSH, the routine must return a zero in register A. You can accomplish this with the following instructions: xra a ret

3.4.4 Memory Select and Move Functions

This section defines the memory management functions MOVE, XMOVE, SELMEM, and SETBNK.

BIOS Function 25: MOVE

Memory-to-Memory Block Move

Entry Parameters:

```
HL=Destination address
DE=Source address
```

BC=Count

Returned Values:

```
HL and DE must point to next bytes following move operation
```

The BDOS calls the MOVE routine to perform memory to memory block moves to allow use of the Z80 LDIR instruction or special DMA hardware, if available. Note that the arguments in HL and DE are reversed from the Z8O machine instruction, necessitating the use of XCHG instructions on either side of the LDIR. The BDOS uses this routine for all large memory copy operations. On return, the HL and DE registers are expected to point to the next bytes following the move.

Usually, the BDOS expects MOVE to transfer data within the currently selected bank or common memory. However, if the BDOS calls the XMOVE entry point before calling MOVE, the MOVE routine must perform an interbank transfer.

BIOS Function 27: SELMEM

Select Memory Bank

Entry Parameters:

```
A=Memory Bank
```

Returned Values; None

The SELMEM entry point is only present in banked systems. The banked version of the CP/M 3 BDOS calls SELMEM to select the current memory bank for further instruction execution or buffer references. You must preserve or restore all registers other than the accumulator, A, upon exit.

BIOS Function 28: SETBNK

Specify Bank for DMA Operation

Entry Parameters:

```
A=Memory Bank
```

Returned Values: None

SETBNK only occurs in the banked version of CP/M 3. SETBNK specifies the bank that the subsequent disk READ or WRITE routine must use for memory transfers. The BDOS always makes a call to SETBNK to identify the DMA bank before performing a READ or WRITE call. Note that the BDOS does not reference banks other than 0 or 1 unless another bank is specified by the BANK field of a Data Buffer Control Block (BCB).

BIOS Function 29: XMOVE

Set Banks for Following MOVE

Entry Parameters:

```
B=destination bank
C=source bank
```

Returned Values: None

XMOVE is provided for banked systems that support memory-to- memory DMA transfers over the entire extended address range. Systems with this feature can have their data buffers located in an alternate bank instead of in common memory, as is usually required. An XMOVE call affects only the following MOVE call. All subsequent MOVE calls apply to the memory selected by the latest call to SELMEM. After a call to the XMOVE function, the following call to the MOVE function is not more than 128 bytes of data. If you do not implement XMOVE, the first instruction must be a RET instruction.

3.4.5 Clock Support Function

This section defines the clock support function TIME.

BIOS Function 26: TIME

Get and Set Time

Entry Parameters:

```
C=Time Get/Set Flag
```

Returned values: None

The BDOS calls the TIME function to indicate to the BIOS whether it has just set the Time and Date fields in the SCB, or whether the BDOS is about to get the Time and Date from the SCB. On entry to the TIME function, a zero in register C indicates that the BIOS should update the Time and Date fields in the SCB. A OFFH in register C indicates that the BDOS has just set the Time and Date in the SCB and the BIOS should update its clock. Upon exit, you must restore register pairs HL and DE to their entry values.

This entry point is for systems that must interrogate the clock to determine the time.

Systems in which the clock is capable of generating an interrupt should use an interrupt service routine to set the Time and Date fields on a regular basis.

3.5 Banking Considerations

This section discusses considerations for separating your BIOS into resident and banked modules. You can place part of your customized BIOS in common memory, and part of it in Bank 0. However, the following data structures and routines must remain in common memory:

- the BIOS stack
- the BIOS jump vector
- Disk Parameter Blocks
- memory management routines
- the CHRTBL data structure
- all character I/O routines
- portions of the disk I/O routines

You can place portions of the disk I/O routines in the system bank, Bank 0. In a banked environment, if the disk I/O hardware supports DMA transfers to and from banks other than the currently selected bank, the disk I/O drivers can reside in Bank 0. If the system has a DMA controller that supports block moves from memory to memory between banks, CP/M 3 also allows you to place the blocking and deblocking buffers in any bank other than Bank 1, instead of common memory.

If your disk controller supports data transfers only into the currently selected bank, then the code that initiates and performs a data transfer must reside in common memory. In this case, the disk I/O transfer routines must select the DMA bank, perform the transfer, then reselect Bank 0.

The routine in common memory performs the following procedure:

1) Selects the DMA bank that SETBNK saved.
2) Performs physical I/O.
3) Reselects Bank 0.
4) Returns to the calling READ or WRITE routine in Bank 0.

Note that Bank 0 is in context (selected) when the BDOS calls the system initialization functions BOOT and DRVTBL; the disk I/O routines HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, SECTRN, MULTIO, and FLUSH; and the memory management routines KMOVE and SETBNK.

Bank 0 or Bank 1 is in context when the BDOS calls the system initialization routines WBOOT, DEVTBL, and DEVINI; the character I/O routines CONST, CONIN, CONOUT, LIST, AUXOUT, AUXIN, LISTST, CONOST, AUXIST, and AUXOST, the memory select and move routines MOVE and SELMEM, and the clock support routine TIME.

You can place a portion of the character I/O routines in Bank 0 if you place the following procedure in common memory.

1) Swap stacks to a local stack in common.
2) Save the current bank.
3) Select Bank 0.
4) Call the appropriate character I/O routine.
5) Reselect the saved bank.
6) Restore the stack.

3.6 Assembling and Linking Your BIOS

This section assumes you have developed a BIOS3.ASM or BNKBIOS3.ASM file appropriate to your specific hardware environment. Use the Digital Research Relocatable Macro Assembler RMAC to assemble the BIOS. Use the Digital Research Linker LINK-8 OTM to create the BIOS3.SPR and BNKBIOS3.SPR files. The SPR files are part of the input to the GENCPM program.

In a banked environment, your CP/M 3 BIOS can consist of two segments: a banked segment and a common segment. This allows you to minimize common memory usage to maximize the size of the TPA. To prepare a banked BIOS, place code and data that must reside in common in the CSEG segment, and code and data that can reside in the system bank in the DSEG segment. When you link the BIOS, LINK-80 creates the BNKBIOS3.SPR file with all the CSEG code and data first, then the DSEG code and data.

After assembling the BIOS with RMAC, link your BNKBIOS using LINK-80 with the

[B] option. The [B] option aligns the DSEG on a page boundary, and places the length of the CSEG into the BNKBIOS3.SPR header page.

Use the following procedure to prepare a BIOS3.SPR or BNKBIOS3.SPR file from your customized BIOS.

1) Assemble your BIOS3.ASM or BNKBIOS3.ASM file with the relocatable assembler RMAC.COM to produce a relocatable file of type REL. Assemble

SCB.ASM to produce the relocatable file SCB.REL.

Assembling the Nonbanked BIOS:

```
A>RMAC BIOS3
```

Assembling the Banked BIOS:

```
A>RMAC BNKBIOS3
```

2) Link the BIOS3.REL or BNKBIOS3.REL file and the SCB.REL file with LINK-80 to produce the BIOS3.SPR or BNKBIOS3.SPR file. The [OS] option with LINK causes the output of a System Page Relocatable (SPR) file.

Linking the Nonbanked BIOS:

```
A>LINK BIOS3[OS]=BIOS3,SCB
```

Linking the Banked BIOS:

```
A>LINK BNKBIOS3[B]=BNKBIOS3.SCB
```

The preceding examples show command lines for linking a banked and nonbanked BIOS.

In these examples, the BIOS3.REL and BNKBIOS3.REL are the files of your assembled BIOS.

SCB.REL contains the definitions of the System Control Block variables. The [B] option implies the [OS] option.

Section 4 : CP/M 3 Sample BIOS Modules

This section discusses the modular organization of the example CP/M 3 BIOS on your distribution disk. For previous CP/M operating systems, it was necessary to generate all input/output drivers from a single assembler source file. Such a file is difficult to maintain when the BIOS supports several peripherals. As a result, Digital Research is distributing the BIOS for CP/M 3 in several small modules.

The organization of the BIOS into separate modules allows you to write or modify any I/O driver independently of the other modules. For example, you can easily add another disk I/O driver for a new controller with minimum impact on the other parts of the BIOS.

4.1 Functional Sumary of BIOS Modules

The modules of the BIOS are BIOSKRNL.ASM, SCB.ASM, BOOT.ASM, MOVE.ASM, CHARIO.ASM, DRVTBL.ASM, and a disk I/O module for each supported disk controller in the configuration.

BIOSKRNL.ASM is the kernel, root, or supervisor module of the BIOS. The SCB.ASM module contains references to locations in the System Control Block. You can customize the other modules to support any hardware configuration. To customize your system, add or modify external modules other than the kernel and the SCE.ASM module.

Digital Research supplies the BIOSKRNL.ASM module. This module is the fixed, invariant portion of the BIOS, aiid the interface from the BDOS to all BIOS functions. It is supplied in source form for reference only, and you should not modify it except for the equate statement described in the following paragraph.

You must be sure the equate statement (banked equ true) at the start of the BIOSKRNL.ASM source file is correct for your system configuration. Digital Research distributes the BIOSKRNL.ASM file for a banked system. If you are creating a BIOS for a nonbanked system, change the equate statement to the following: banked equ false and reassemble with RMAC. This is the only change you should make to the BIOSKRNL.ASM file.

Table 4-1 summarizes the modules in the CP/M 3 BIOS.

Table 4-1. CP/M 3 BIOS Module Function Summary

Module	Function
BIOSKRNL.ASM	Performs basic system initialization, and dispatches character and disk I/O.
SCB.ASM module	Contains the public definitions of the various fields in the System Control Block. The BIOS can reference the public variables.
BOOT.ASM module	Performs system initialization other than character and disk I/O. BOOT loads the CCP for cold starts and reloads it for warm starts. CHARIO.ASM module Performs all character device initialization, input, output, and status polling. CHARIO contains the character device characteristics table.
DRVTBL.ASM module	Points to the data structures for each configured disk drive. The drive table determines which physical disk unit is associated with which logical drive. The data structure for each disk drive is called an Extended Disk Parameter Header (XDPH).

Disk I/O modules	Initialize disk controllers and execute READ and WRITE code for disk controllers. You must provide an XDPH for each supported unit, and a separate disk I/O module for each controller in the system. To add another disk controller for which a prewritten module exists, add its XDPH names to the DRVTBL and link in the new module.
MOVE.ASM module	Performs memory-to-memory moves and bank selects.

4.2 Conventions Used in BIOS Modules

The Digital Research RMAC relocating assembler and LINK-80 linkage editor allow a module to reference a symbol contained in another module by name. This is called an external reference. The Microsoft relocatable object module format that RMAC and LINK use allows six-character names for externally defined symbols. External names must be declared PUBLIC in the module in which they are defined. The external names must be declared EXTRN in any modules that reference them.

The modular BIOS defines a number of external names for specific purposes. Some of these are defined as public in the root module, BIOSKRNL.ASM. Others are declared external in the root and must be defined by the system implementor. Section 4.4 contains a table summarizing all predefined external symbols used by the modular BIOS.

External names can refer to either code or data. All predefined external names in the modular BIOS prefixed with a @ character refer to data items. All external names prefixed with a ? character refer to a code label. To prevent conflicts with future extensions, user-defined external names should not contain these characters.

4.3 Interactions of Modules

The root module of the BIOS, BIOSKRNL.ASM, handles all BDOS calls, performs interfacing functions, and simplifies the individual modules you need to create.

4.3.1 Initial Boot

BIOSKRNL.ASM initializes all configured devices in the following order:

1) BIOSKRNL calls ?CINIT in the CHARIO module for each of the 16 character devices and initializes the devices.
2) BIOSKRNL invokes the INIT entry point of each XDPH in the FD1797SD module.
3) BIOSKRNL calls the ?INIT entry of the BOOT module to initialize other system hardware, such as memory controllers, interrupts, and clocks. It prints a sign-on message specific to the system, if desired.
4) BIOSKRNL calls ?LDCCP in the BOOT module to load the CCP into the TPA.
5) The BIOSKRNL module sets up Page Zero of the TPA with the appropriate jump vectors, and passes control to the CCP.

4.3.2 Character I/O Operation

The CHARIO module performs all physical character I/O. This module contains both the character device table (@CTBL) and the routines for character input, output, initialization, and status polling. The character device table, @CTBL, contains the ASCII name of each device, mode information, and the current baud rate of serial devices.

To support logical to physical redirection of character devices, CP/M 3 supplies a 16-bit assignment vector for each logical device. The bits in these vectors correspond to the physical devices. The character I/O interface routines in BIOSKRNL handle all device assignment, calling the appropriate character I/O routines with the correct device number. The BIOSKRNL module also handles XON/XOFF processing on output devices where it is enabled.

You can use the DEVICE utility to assign several physical devices to a logical device. The BIOSKRNL root module polls the assigned physical devices, and either reads a character from the first ready input device that is selected, or sends the character to all of the selected output devices as they become ready.

4.3.3 Disk I/O Operation

The BIOSKRNL module handles all BIOS calls associated with disk I/O. It initializes global variables with the parameters for each operation, then invokes the READ or WRITE routine for a particular controller. The SELDSK routine in the BIOSKRNL calls the LOGIN routine for a controller when the BDOS initiates a drive login. This allows disk density or media type to be automatically determined.

The DRVTBL module contains the sixteen-word drive table, @DTBL. The order of the entries in @DTBL determines the logical to physical drive assignment. Each word in @DTBL contains the address of a DPH, which is part of an XDPH, as shown in Table 4-10. The word contains a zero if the drive does not exist. The XDPH contains the addresses of the INIT, LOGIN, READ, and WRITE entry points of the I/O driver for a particular controller. When the actual drivers are called, globally accessible variables contain the various parameters of the operation, such as the track and sector.

4.4 Predefined Variables and Subroutines

The modules of the BIOS define public variables which other modules can reference.

Table 4-2 contains a summary of each public symbol and the module that defines it.

Table 4-2. Public Symbols in CP/M 3 BIOS

Symbol	Function and Use	Defined in Module
@ADRV	Byte, Absolute drive code	BIOSKRNL
@CBNK	Byte, Current CPU bank	BIOSKRNL
@CNT	Byte, Multisector count	BIOSKRNL
@CTBL	Table, Character device table	CHARIO
@DBNK	Byte, Bank for disk I/O	BIOSKRNL
@DMA	Word, DMA address	BIOSKRNL
@DTBL	Table, Drive table	DRVTBL
@RDRV	Byte, Relative drive code (UNIT)	BIOSKRNL
@SECT	Word, Sector address	BIOSKRNL
@TRK	Word, Track number	BIOSKRNL
?BANK	Bank select	MOVE
?CI	Character device input	CHARIO
?CINIT	Character device initialization	CHARIO
?CIST	Character device input status	CHARIO

?CO	Character device output	CHARIO
?COST	Character device output status	CHARIO
?INIT	General initialization	BOOT
?LDCCP	Load CCP for cold start	BOOT
?MOVE	Move memory to memory	MOVE
?PDEC	Print decimal number	BIOSKRNL
?PDERR	Print BIOS disk error header	BIOSKRNL
?PMSG	Print message	BIOSKRNL
?RLCCP	Reload CCP for warm start	BOOT
?XMOVE	Set banks for extended move	MOVE
?TIME	Set or Get time	BOOT

The System Control Block defines public variables that other modules can reference.

The System Control Block variables @CIVEC, @COVEC, @AIVEC, @AOVEC, and @LOVEC are referenced by BIOSKR,-NL.ASM. The variable @BNKBF can be used by ?LDCCP and ?RLCCP to implement interbank block moves. The public variable names

@ERMDE, @FX, @RESEL, @VINFO, @CRDSK, @USRCD, and @CRDMA are used for error routines which intercept BDOS errors. The publics @DATE, @HOUR, @MIN, and @SEC can be updated by an interrupt-driven real-time clock. @MXTPA contains the current BDOS entry point.

Disk I/O operation parameters are passed in the following global variables, as shown in Table 4-3.

Table 4-3. Global Variables in BIOSKRNL.ASM

Variable	Meaning
@ADRV	Byte; contains the absolute drive code (0 through F for A through P) that CP/M is referencing for READ and WRITE operations. The SELDSK routine in the BIOSKRNL module obtains this value from the BDOS and places it in @DRV. The absolute drive code is used to print error messages.
@RDRV	Byte; contains the relative drive code for READ and WRITE operations.

The relative drive code is the UNIT number of the controller in a given disk I/O module. BIOSKRNL obtains the unit number from the XDPH.

This is the actual drive code a driver should send to the controller.

Several utility subroutines are defined in the BIOSKRNL.ASM module, as shown in Table 4-4.

Table 4-4. Public Utility Subroutines in BIOSKRNL.ASK utility I meaning

@TRK	Word; contains the starting track for READ and WRITE.
@SECT	Word; contains the starting sector for READ and WRITE.
@DMA	Word; contains the starting disk transfer address.

@DBNK	Byte; contains the bank of the DMA buffer.
@CNT	Byte; contains the physical sector count for the operations that follow.
@CBNK	Byte; contains the current bank for code execution.

?PMSG Print string starting at <HL>, stop at null (0). ?PDEC Print binary number in decimal from HL. ?PDERR Print disk error message header using current disk parameters:

<CR><LF>BIOS Error on d:, T- nn, S-nn.

All BIOS entry points in the jump vector are declared as public for general reference by other BIOS modules, as shown in Table 4-5.

Table 4-5. Public Names in the BIOS Jump Vector

Public Name	Function
?BOOT	Cold boot entry
?WBOOT	Warm boot entry
?CONST	Console input status
?CONIN	Console input
?CONO	Console output
?LIST	List output
?AUXO	Auxiliary output
?AUXI	Auxiliary input
?HOME	Home disk drive
?SLDSK	Select disk drive
?ST12RK	Set track
?STSEC	Set sector
?STDMA	Set DMA address
?READ	Read record
?WRITE	Write record
?LISTS	List status
?SCTRN	Translate sector
?CONOS	Console output status
?AUXIS	Auxiliary input status
?AUXOS	Auxiliary output status
?DVTBL	Return character device table address
?DEVIN	Initialize character device
?DRTBL	Return disk drive table address

Public Name	Function
?MLTIO	Set multiple sector count
?FLUSH	Flush deblocking buffers (not implemented)
?MOV	Move memory block
?TIM	Signal set or get time from clock
?BNKSL	Set bank for further execution
?STBNK	Set bank for DMA
?XMOV	Set banks for next move

4.5 BOOT Module

The BOOT module performs general system initialization, and loads and reloads the CCP. Table 4-6 shows the entry points of the BOOT module.

Table 4-6. BOOT Module Entry Points

Module	Meaning
?INIT	The BIOSKRNL module calls ?INIT during cold start to perform hardware initialization other than character and disk I/O. Typically, this hardware can include time-of-day clocks, interrupt systems, and special I/O ports used for bank selection.
?LDCCP	BIOSKRNL calls ?LDCCP during cold start to load the CCP into the TPA. The CCP can be loaded either from the system tracks of the boot device or from a file, at the discretion of the system implementor. In a banked system, you can place a copy of the CCP in a reserved area of another bank to increase the performance of the ?RLCCP routine.
?RLCCP	BIOSKRNL calls ?RLCCP during warm start to reload the CCP into the TPA. In a banked system, the CCP can be copied from an alternate bank to eliminate any disk access. Otherwise, the CCP should be loaded from either the system tracks of the boot device or from a file.

4.6 Character I/O

The CHARIO module handles all character device interfacing. The CHARIO module contains the character device definition table @CTBL, the character input routine ?CI , the character output routine ?CO, the character input status routine ?CIST, the character output status routine ?COST, and the character device initialization routine ?CINIT.

The BIOS root module, BIOSKRNL.ASM, handles all character I/O redirection. This module determines the appropriate devices to perform operations and executes the actual operation by calling ?CI, ?CO, ?CIST, and ?COST with the proper device number(s).

@CTBL is the external name for the structure CHRTBL described in Section 3 of this manual. @CTBL contains an 8-byte entry for each physical device def-ined by this BIOS. The table is terminated by a zero byte after the last entry.

The first field of the character device table, @CTBL, is the 6- byte device name. This device name should be all upper-case, left- justified, and padded with ASCII spaces (20H).

The second field of @CTBL is 1 byte containing bits that Indicate the type of device and its current mode, as shown in Table 4-7.

Table 4-7. Mode Bits

Mode Bits	Meaning
00000001	Input device (such as a keyboard)
00000010	output device (such as a printer)
00000011	Input/output device (such as a terminal or modem)
00000100	Device has software-selectable baud rates
00001000	Device may use XON protocol
00010000	XON/XOFF protocol enabled

The third field of @CTBL is 1 byte and contains the current baud rate for serial devices. The high-order nibble of this field is reserved for future use and should be set to zero.

The low-order four bits contain the current baud rate as shown in Table 4-8. Many systems do not support all of these baud rates.

Table 4-8. Baud Rates for Serial Devices

Decimal	Binary	Baud Rate
0	0000	none
1	0001	50
2	0010	75
3	0011	110
4	010 0	134. 5
5	0101	150
6	0110	300
7	0111	600
8	1000	1200
9	1001	1800
10	1010	2400
11	1011	3600
12	1100	4800
13	1101	7200
14	1110	9600
15	1111	19200

Table 4-9 shows the entry points to the routines in the CHARIO module. The BIOSKRNL module calls these routines to perform machine-dependent character I/O.

Table 4-9. Character Device Labels

Label	Meaning
?Ci **Character Device Input**	?CI is called with a device number in register B. It should wait for the next available input character , then return the character in register A. The character should be in 8-bit ASCII with no parity.
?CO **Character Device Output**	?CO is called with a device number in register B and a character in register C. It should wait until the device is ready to accept another character and then send the character. The character is in 8-bit ASCII with no parity.
?CIST **Character Device Input Status**	?CIST is called with a device number in register B. It should return with register A set to zero if the device specified has no input character ready; and should return with A set to 0FFH if the device specified has an input character ready to be read.
?COST **Character Device Output Status**	?COST is called with a device number in register B. It should return with register A set to zero if the device specified cannot accept a character immediately, and should return with A set to 0FFH if the device is ready to accept a character.
?CINIT **Character Device Initialization**	?CINIT is called for each of the 16 character devices, and initializes the devices. Register C contains the device number. The ?CINIT routine initializes the physical character device specified in register C to the baud rate contained in the appropriate entry of the CHRTBL. You only need to supply this routine if I/O redirection has been implemented. It is referenced only by the DEVICE utility supplied with CP/M 3.

4.7 Disk I/O

The separation of the disk I/O section of the BIOS into several modules allows you to support each particular disk controller independently from the rest of the system. A manufacturer can supply the code for a controller in object module form, and you can link it into any existing modular BIOS to function with other controllers in the system.

The data structure called the Extended Disk Parameter Header, or XDPH, contains all the necessary information about a disk drive. BIOSKRNL.ASM locates the XDPH for a particular logical drive using the Drive Table. The XDPH contains the addresses of the READ, WRITE, initialization, and login routines. The XDPH also contains the relative unit number of the drive on the controller, the current media type, and the Disk Parameter Header (DPH) that the BDOS requires. Section 3 of this manual describes the Disk Parameter Header.

The code to read and write from a particular drive is independent of the actual CP/M logical drive assignment, and works with the relative unit number of the drive on the controller.

The position of the XDPH entry in the DRVTBL determines the actual CP/M 3 drive code.

4.7.1 Disk I/O Structure

The BIOS requires a DRVTBL module to locate the disk driver. it also requires a disk module for each controller that is supported.

The drive table module, DRVTBL, contains the addresses of each XDPH defined in the system. Each XDPH referenced in the DRVTBL must be declared external to link the table with the actual disk modules.

The XDPHs are the only public entry points in the disk I/O modules. The root module references the XDPHs to locate the actual I/O driver code to perform sector READS and WRITES. When the READ and WRITE routines are called, the parameters controlling the READ or WRITE operation are contained in a series of global variables that are declared public in the root module.

4.7.2 Drive Table Module (DRVTBL)

The drive table module, DRVTBL, def ines the CP/M absolute drive codes associated with the physical disks.

The DRVTBL module contains one public label, @DTBL. @DTBL is a 16-word table containing the addresses of up to 16 XDPH'S. Each XDPH name must be declared external in the DRVTBL. The first entry corresponds to drive A, and the last to drive P. You must set an entry to 0 if the corresponding drive is undefined. Selecting an undefined drive causes a BDOS SELECT error.

4.7.3 Extended Disk Parameter Headers (XDPHS)

An Extended Disk Parameter Header (XDPH) consists of a prefix and a regular Disk Parameter Header as described in Section 3. The label of a XDPH references the start of the DPH. The fields of the prefix are located at relative offsets from the XDPH label.

The XDPHs for each unit of a controller are the only entry points in a particular disk drive module. They contain both the DPH for the drive and the addresses of the various action routines for that drive, including READ, WRITE, and initialization. Figure 4-1 shows the format of the Extended Disk Parameter Header.

Figure 4-1. XDPH Format

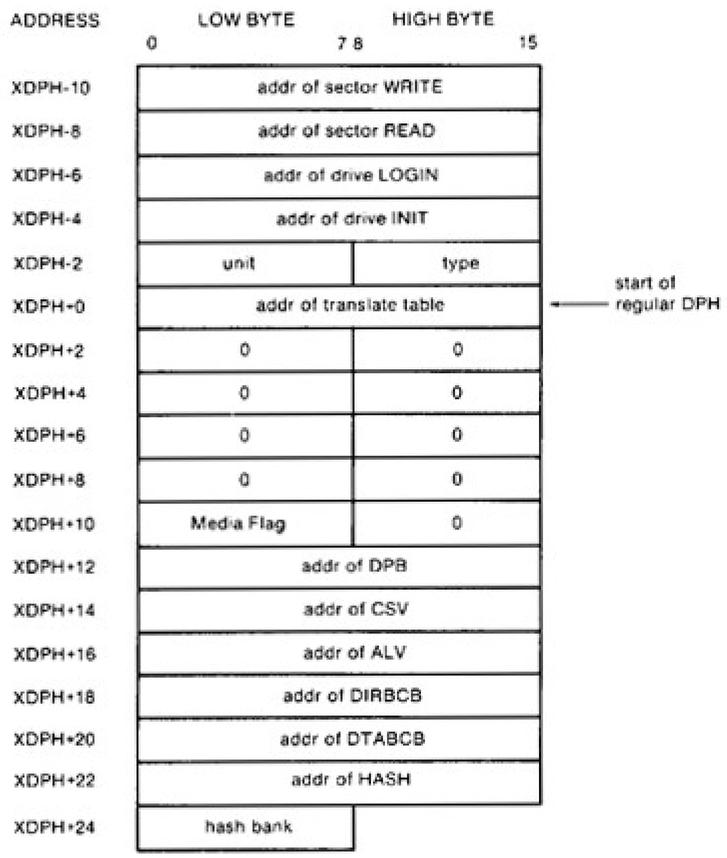

Table 4-10 describes the fields of each Extended Disk Parameter Header.

Table 4-10. Fields of Each XDPH

Field	Meaning
WRITE	The WRITE word contains the address of the sector WRITE routine for the drive.
READ	The READ word contains the address of the sector READ routine for the drive.
LOGIN	The LOGIN word contains the address of the LOGIN routine for the drive.
INIT	The INIT word contains the address of the f irst-time initialization code for the drive.
UNIT	The UNIT byte contains the drive code relative to the disk controller. This is the value placed in @RDRV prior to calling the READ, WRITE, and LOGIN entry points of the drive.
TYPE	The TYPE byte is unused by the BIOS root, and is reserved for the driver to keep the current density or media type to support multiple-format disk subsystems. regular DPH The remaining fields of the XDPH comprise a standard DPH, as discussed in Section 3 of this manual.

4.7.4 Subroutine Entry Points

The pointers contained in the XDPH reference the actual code entry points to a disk driver module. These routines are not declared public. Only the XDPH itself is public. The BIOS root references the XDPHs only through the @DTBL. Table 4-11 shows the BIOS subroutine entry points.

Table 4-11. Subroutine Entry Points

Entry Point	Meaning
WRITE	When the WRITE routine is called, the address of the XDPH is passed in registers DE. The parameters for the WRITE operation are contained in the public variables @ADRV, @RDRV, @TRK, @SECT, @DMA, and @DBNK. The WRITE routine should return an error code in register A. The code 00 means a successful operation, 01 means a permanent error occurred, and 02 means the drive is write-protected if that feature is supported.
READ	When the READ routine is called, the address of the XDPH is contained in registers DE. The parameters for the READ operation are contained in the public variables @ADRV, @RDRV, @TRK, @SECT, @DMA, and
@DBNK	The READ routine should return an error code in register A. A code of 00 means a successful operation and 01 means a permanent error occurred.
LOGIN	The LOGIN routine is called before the BDOS logs into the drive, and allows the automatic determination of density. The LOGIN routine can alter the various parameters in the DPH, including the translate table address (TRANS) and the Disk Parameter Block (DPB) . The LOGIN routine can also set the TYPE byte. On single media type systems, the LOGIN routine can simply return. When LOGIN is called, the registers DE point to the XDPH for this drive.
INIT	The BOOT entry of the BIOSKRNL module calls each INIT routine during cold start and prior to any other disk accesses. INIT can perform any necessary hardware initialization, such as setting up the controller and interrupt vectors, if any.

4.7.5 Error Handling and Recovery

The READ and WRITE routines should perform several retries of an operation that produces an error. If the error is related to a seek operation or a record not found condition, the retry routine can home or restore the drive, and then seek the correct track. The exact sequence of events is hardware-dependent.

When a nonrecoverable error occurst the READ or WRITE routines can print an error message informing the operator of the details of the error. The BIOSKRNL module supplies a subroutine, ?PDERR, to print a standard BIOS error message header. This routine prints the following message:

BIOS Err on D: T-nn S-nn

The D: is the selected drive, and T-nn and S-nn display the track and sector number for the operation. The READ and WRITE routines should print the exact cause of the error after this message, such as Not Ready, or Write Protect. The driver can then ask the operator if additional retries are desired, and return an error code to the BDOS if they are not.

However, if the @ERMDE byte in the System Control Block indicates the BDOS is returning error codes to the application program without printing error messages, the BIOS should simply return an error without any message.

4.7.6 Multiple Sector I/O

The root module global variable @CNT contains the multisector count. Refer to Sections 2.5 and 3.4.3 for a discussion of the considerations regarding multirecord I/O.

4.8 MOVE Module

The MOVE Module performs memory-to-memory block moves and controls bank selection. The ?MOVE and ?XMOVE entry points correspond directly to the MOVE and XMOVE jump vector routines documented in Section 3. Table 4-12 shows the entry points for the MOVE module.

Table 4-12. Move Module Entry Points

Entry Point	Meaning
?MOVE	Memory-to-memory move ?MOVE is called with the source address for the move in register DE, the destination address in register HL, and the byte count in register BC. If ?XMOVE has been called since the last call to ?MOVE, an interbank move must be performed. On return, registers HL and DE must point to the next bytes after the MOVE. This routine can use special DMA hardware for the interbank move capability, and can use the Z8O LDIR instruction for intrabank moves.
?XMOVE	Set banks for one following ?MOVE ?XMOVE is called with the destination bank in register B and the source bank in register C. Interbank moves are only invoked if the DPHs specify deblocking buffers in alternate banks. ?XMOVE only applies to one call to ?MOVE. (Not implemented in the example.)
?BANK	Set bank for execution ?BANK is called with the bank address in register A. This bank address has already been stored in @CBNK for future reference. All registers except A must be maintained upon return.

4.9 Linking Modules into the BIOS

The following lines are examples of typical link commands to build a modular BIOS ready for system generation with GENCPK:

```
LINK BNKBIOS3[b]=BNKBIOS,SCB,BOOT,CHARIO,MOVE,DRVTBL,<disk-modules>
LINK BIOS3[os]=BIOS,SCB,BOOT,CHARIO,MOVE,DRVTBL,<disk-modules>
```

Section 5 : System Generation

This section describes the use of the GENCPM utility to create a memory image CPM3.SYS file containing the elements of the CP/M 3 operating system. This section also describes customizing the LDRBIOS portion of the CPMLDR program, and the operation of CPMLDR to read the CPM3.SYS file into memory.

Finally, this section describes the procedure to follow to boot CP/M 3.

In the nonbanked system, GENCPM creates the CPM3.SYS file from the BDOS3.SPR and your customized BIOS3.SPR files. In the banked system, GENCPM creates the CPM3.

SYS f i le from the RESBDOS3. SPR f i le, the BNKBDOS3.SPR file, and your customized BNKBIOS3.SPR file.

If your BIOS contains a segment that can reside in banked memory, GENCPM separates the code and data in BNKBIOS3.SPR into a banked portion which resides in Bank 0 just below common memory, and a resident portion which resides in common memory.

GENCPM relocates the system modules, and can allocate physical record buffers, allocation vectors, checksum vectors, and hash tables as requested in the BIOS data structures. It also relocates references to the System Control Block, as described on page 27. GENCPM accepts its command input from a file, GENCPM.DAT, or interactively from the console.

5.1 GENCPM Utility

Syntax:

```
GENCPM [AUTO | AUTO DISPLAY}
```

Purpose:

GENCPM creates a memory image CPM3.SYS file, containing the CP/M 3 BDOS and customized BIOS. The GENCPM utility performs late resolution of intermodule references between system modules. GENCPM can accept its command input interactively from the console or from a file GENCPM.DAT.

In the nonbanked system, GENCPM creates a CPM3.SYS file from the BDOS3.SPR and BIOS3.SPR files. In the banked system, GENCPM creates the CPM3.SYS file from the RESBDOS3.SPR, the BNKBDOS3.SPR and the BNKBIOS3.SPR files. Remember to back up your CPM3.SYS file before executing GENCPM, because GENCPM deletes any existing CPM3.SYS file before it generates a new system.

Input Files:

Banked System	**Nonbanked System**
BNKBIOS3.SPR	BIOS3.SPR
RESBDOS3.SPR	BDOS3.SPR
BNKBDOS3.SPR	
optionally GENCPM.DAT	

Output File:

CPM3.SYS optionally GENCPM.DAT

GENCpm determines the location of the system modules in memory and, optionally, the number of physical record buffers allocated to the system. GENCPM can specify the location of hash tables requested by the Disk Parameter Headers (DPHS) in the BIOS. GENCPM can allocate all required disk buffer space and create all the required Buffer Control Blocks (BCBs).

GENCPM can also create checksum vectors and allocation vectors.

GENCPM can get its input from a file GENCPM.DAT. The values in the file replace the default values of GENCPM. If you enter the AUTO parameter in the command line GENCPM gets its input from the file GENCPM.DAT and generates a new system displaying only its sign- on and sign-off messages on the console. If AUTO is specified and a GENCPM.DAT file does not exist on the current drive, GENCPM reverts to manual generation.

If you enter the AUTO DISPLAY parameter in the command line, GENCPM automatically generates a new system and displays all questions on the console. If AUTO DISPLAY is specified and a GENCPM.DAT file does not exist on the current drive, GENCPM reverts to manual generation. If GENCPM is running in AUTO mode and an error occurs, it reverts to manual generation and starts from the beginning.

The GENCPM.DAT file is an ASCII file of variable names and their associated values. In the'following discussion, a variable name in the GENCPM.DAT file is referred to as a Question Variable. A line in the GENCPM.DAT file takes the following general form:

Question Variable = value I ? I ?value <CR><LF> value = #decimal value or hexadecimal value or drive letter (A - P) or Yes, No, Y, or N

You can specify a default value by following a question mark with the appropriate value, for example ?A or ?25 or ?Y. The question mark tells GENCPM to stop and prompt the user for input, then continue automatically. At a ?value entry, GENCPM displays the default value and stops for verification.

The following pages display GENCPM questions. The items in parentheses are the default values. The Question Variable associated with the question is shown below the explanation of the answers to the questions.

5.1.1 : Program Questions:

Use GENCPM.DAT for defaults (Y) ?

Enter Y - GENCPM gets its default values from the file GENCPM.DAT.

Enter N - GENCPM uses the built-in default values.

No Question Variable is associated with this question

Create a new GENCPM.DAT file (N) ?

Enter N - GENCPM does not create a new GENCPM.DAT fil,

Enter Y - After GENCPM generates the new CPM3.SYS file it creates a new GENCPM.DAT file containing the default values.

Question Variable: CRDATAF

Display Load Table at Cold Boot (Y) ?

Enter Y - On Cold Boot the system displays the load table containing the filename, filetype, hex starting address, length of system modules, and the TPA size.

Enter N - System displays only the TPA size on cold boot.

Question Variable: PRTMSG

Number of console columns (#80) ?

Enter the number of columns (characters-per-line) for your console.

A character in the last column must not force a new line for console editing in CP/M 3. If your terminal forces a new line automatically, decrement the column count by one.

Question Variable: PAGWID

Number of lines per console page (#24) ?

Enter the number of the lines per screen for your console.

Question Variable: PAGLEN

Backspace echoes erased character (N) ?

Enter N - Backspace (Ctrl-H, 08H) moves back one column and erases the previous character.

Enter Y - Backspace moves forward one column and displays the previous character.

Question Variable: BACKSPC

Rubout echoes erased character (Y) ?

Enter Y - Rubout (7FH) moves forward one column and displays the previous character.

Enter N - Rubout moves back one column and erases the previous character.

Question Variable: RUBOUT

Initial default drive (A:) ?

Enter the drive code the prompt is to display at cold boot.

Question Variable: BOOTDRV

Top page of memory (FF) ?

Enter the page address that is to be the top of the operating system. OFFH is the top of a 64K system.

Question Variable: MEMTOP

Bank-switched memory (Y) ?

Enter Y - GENCPM uses the banked system files.

Enter N - GENCPM uses the nonbanked system files.

Question Variable: BNKSWT

Common memory base page (CO) ?

This question is displayed only if you answered Y to the previous question. Enter the page address of the start of common memory.

Question Variable: COMBAS

Long error messages (Y) ?

This question is displayed only if you answered Y to bank- switched memory.

Enter Y - CP/M 3 error messages contain the BDOS function number and the name of the file on which the operation was attempted.

Enter N - CP/M 3 error messages do not display the function number or file.

Question Variable: LERROR

Double allocation vectors (Y) ?

This question is displayed only if you answered N to bank- switched memory.

For more information about double allocation vectors, see the definition of the Disk Parameter Header ALV field in Section 3.

Enter Y - GENCPM creates double-bit allocation vectors for each drive.

Enter N - GENCPM creates single-bit allocation vectors for each drive.

Question Variable: DBLALV

Accept new system definition (Y) ?

Enter Y GENCPM proceeds to the next set of questions.

Enter N GENCPM repeats the previous questions and displays your previous input in the default parentheses. You can modify your answers.

No Question Variable is associated with this question.

Number of memory segments (#3) ?

GENCPM displays this question if you answered Y to bank- switched memory.

Enter the number of memory segments in the system. Do not count common memory or memory in Bank 1, the TPA bank, as a memory segment. A maximum of 16 (0 - 15) memory segments are allowed. The memory segments define to GENCPM the memory available for buffer and hash table allocation.

Do not include the part of Bank 0 that is reserved for the operating system.

Question Variable: NUMSEGS

CP/M 3 Base,size,bank (8E,32,00)

Enter memory segment table:

Base,size,bank (00,8E,00) ?

Base,size,bank (00,CO,02) ?

Base,size,bank (00,CO,03) ?

Enter the base page, the length, and the bank of the memory segment.

Question Variable: MEMSEGO# where 0 to F hex

Accept new memory segment table entries (Y) ?

Enter Y GENCPM displays the next group of questions.

Enter N GENCPM displays the memory segment table definition questions again.

No Question Variable is associated with this question.

5.1.2 : Setting up directory hash tables:

Enable hashing for drive d: (Y)

GENCPM displays this question if there is a Drive Table and if the DPHs for a given drive have an 0FFFEH in the hash table address field of the DPH. The question is asked for every drive d: defined in the BIOS.

Enter Y - Space is allocated for the Hash Table. The address and bank of the Hash Table is entered into the DPH.

Enter N - No space is allocated for a Hash Table for that drive.

Question Variable: HASHDRVD where d = drives A-P.

5.1.3 : Setting up Blocking/Deblocking buffers:

GENCPM displays the next set of questions if either or both the DTABCB field or the DIRBCB field contain 0FFFEH.

Number of directory buffers for drive d: (#l) ? 10

This question appears only if you are generating a banked system. Enter the number of directory buffers to allocate for the specified drive. In a banked system, directory buffers are allocated only inside Bank 0. In a nonbanked system, one directory buffer is allocated above the BIOS.

Question Variable: NDIRRECD where d = drives A-P.

Number of data buffers for drive d: (#l) ? 1

This question appears only if you are generating a Banked system. Enter the number of data buffers to allocate for the specified drive. In a banked system, data buffers can only be allocated outside Bank 1, and in common. You can only allocate data buffers in alternate banks if your BIOS supports interbank moves.

In a nonbanked system, data buffers are allocated above the BIOS.

Question Variable: NDTARECD where d = drives A-P.

Share buffer(s) with which drive (A:) ?

This question appears only if you answered zero to either of the above questions.

Enter the drive letter (A-P) of the drive with which you want this drive to share a buffer.

Question Variable: ODIRDRVD for directory records where d = drives A-P.

Question Variable: ODTADRVD for data records where d drives A-P.

Allocate buffers outside of Commom (N) ?

This question appears if the BIOS XMOVE routine is implemented.

Answer Y - GENCPM allocates data buffers outside of common and Bank 0.

Answer N - GENCPM allocates data buffers in common.

Question Variable: ALTBNKSD where d = drives A-P.

Overlay Directory buffer for drive d: (Y) ?

This question appears only if you are generating a nonbanked system.

Enter Y this drive shares a directory buffer with another drive.

Enter N GENCPM allocates an additional directory buffer above the BIOS.

Question Variable: OVLYDIRD where d = drives A-P.

Overlay Data buffer for drive d: (Y) ?

This question appears only if you are generating a nonbanked system.

Enter Y - this drive shares a data buffer with another drive.

Enter N - GENCPM allocates an additional data buffer above the BIOS.

Question Variable: OVLYDTAD for directory records where d = drives A-P.

Accept new buffer definitions (Y) ?

Enter Y GENCPM creates the CPM3.SYS file and terminates.

Enter N GENCPM redisplays all of the buffer definition questions.

No Question Variable is associated with this question.

Examples:

The following section contains examples of two system generation sessions. If no entry follows a program question, assume RETURN was entered to select the default value in parentheses. Entries different from the default appear after the question mark.

EXAMPLE OF CONTENTS OF GENCPM.DAT FILE

```
combas = c0 <CR> lerror = ? <CR> numsegs
3 <CR> memseg00
00,80,00 <CR> memseg0l
0d,b3,02 <CR> memseg0f
?00,c0,10 <CR> hashdrva y <CR> hashdrvd n <CR> ndirreca
20 <CR> ndtarecf
10 <CR>
```

EXAMPLE OF SYSTEM GENERATION WITH BANKED MEMORY

```
A>GENCPM
CP/M 3.0 System Generation Copyright (C) 1982, Digital Research
Default entries are shown in (parens).
```

```
Default base is Hex, precede entry with # for decimal
Use GENCPM.DAT for defaults (Y) ?
Create a new GENCPM.DAT file (N) ?
Display Load Map at Cold Boot (Y) ?
Number of console columns (#80) ?
Number of lines in console page (#24) ?
Backspace echoes erased character (N) ?
Rubout echoes erased character (N) ?
Initial default drive (A:) ?
Top page of memory (FF) ?
Bank switched memory (Y) ?
Common memory base page (CO) ?
Long error messages (Y) ?
Accept new system definition (Y) ?
Setting up Allocation vector for drive A:
Setting up Checksum vector for drive A:
Setting up Allocation vector for drive B:
Setting up Checksum vector for drive B:
Setting up Allocation vector for drive C:
Setting up Checksum vector for drive C:
Setting up Allocation vector for drive D:
Setting up Checksum vector for drive D:
*** Bank 1 and Common are not included ***
*** in the memory segment table.
Number of memory segments (#3) ?
CP/M 3 Base,size,bank (8B,35,00)
Enter memory segment table:
Base,size,bank (00,8B,00)?
Base,size,bank (OD,B3,02)?
Base,size,bank (00,CO,03)?
CP/M 3 Sys SBOOH 3500H Bank 00
Memseg No. 00 0000H BBOOH Bank 00
Memseg No. 01 ODOOH B300H Bank 02
Memseg No. 02 0000H C000H Bank 03
Accept new memory segment table entries (Y) ?
Setting up directory hash tables:
Enable hashing for drive A: (Y) ?
Enable hashing for drive B: (Y) ?
Enable hashing for drive C: (Y) ?
Enable hashing for drive D: (Y) ?
9 5
Setting up Blocking/Deblocking buffers:
The physical record size is 0200H:
Available space in 256 byte pages:
TPA = 00F4H, Bank 0 = 00BBH, Other banks = 0166H
Number of directory buffers for drive A: (#32) ?
Available space in 256 byte pages:
TPA = 00F4H, Bank 0 = 0049H, Other banks = 0166H
Number of data buffers for drive A: (#2) ?
Allocate buffers outside of Common (N) ?
Available space in 256 byte pages:
TPA = 00F0H, Bank 0 = 0049H, Other banks = 0166H
Number of directory buffers for drive B; (#32) ?
```

```
Available space in 256 byte pages:
TPA = 00F0H, Bank 0 = 0007H, Other banks = 0166H
Number of data buffers for drive B: (#0) ?
Share buffer(s) with which drive (A:) ?
The physical record size is 0080H:
Available space in 256 byte pages:
TPA = 00F0H, Bank 0 = 0007H, Other banks = 0166H
Number of directory buffers for drive C: (410) ?
Available space in 256 byte pages:
TPA = 00F0H, Bank 0 = 0001H, Other banks = 0166H
Number of directory buffers for drive D: (#0) ?
Share buffer(s) with which drive (C:) ?
Available space in 256 byte pages:
TPA = 00F0H, Bank 0 = 0001H, Other banks = 0166H
Accept new buffer definitions (Y) ?
BNKBIOS3 SPR F600H 0600H
BNKBIOS3 SPR BI00H 0F00H
RESBDOS3 SPR F000H 0600H
BNKBDOS3 SPR 8700H 2A00H
*** CP/M 3.0 SYSTEM GENERATION DONE
```

In the preceding example GENCPM displays the resident portion of BNKBIOS3.SPR first, followed by the banked portion.

EXAMPLE OF SYSTEM GENERATION WITH NONBANKED MEMORY

```
A>GENCPM
CP/M 3.0 System Generation
Copyright (C) 1982, Digital Research
Default entries are shown in (parens).
Default base is Hex, precede entry with for decimal
Use GENCPM.DAT for defaults (Y) ?
Create a new GENCPM.DAT file (N) ?
Display Load Map at Cold Boot (Y) ?
Number of console columns (#80) ?
Number of lines in console page (#24) ?
Backspace echoes erased character (N) ?
Rubout echoes erased character (N) ?
Initial default drive (A:) ?
Top page of memory (FF) ?
Bank switched memory (Y) ? N
Double allocation vectors (Y) ?
Accept new system definition (Y) ?
Setting up Blocking/Deblocking buffers:
The physical record size is 0200H:
Available space in 256 byte pages:
TPA = 00D8H
Directory buffer required and allocated for drive A:
Available space in 256 byte pages:
TPA = 00D5H
Overlay Data buffer for drive A: (Y) ?
Available space in 256 byte pages:
TPA = 00D5H
```

```
Overlay Directory buffer for drive B: (Y) ?
Share buffer(s) with which drive (A:) ?
Available space in 256 byte pages:
TPA = 00D5H
Overlay Data buffer for drive B: (Y) ?
Share buffer(s) with which drive (A:) ?
The physical record size is 00B0H:
Available space in 256 byte pages:
TPA = 0005H
Overlay Directory buffer for drive C: (Y) ?
Share buffer(s) with which drive (A:) ?
Available space in 256 byte pages:
TPA = 00D5H
Overlay Directory buffer for drive D: (Y) ?
Share buffer(s) with which drive (C;) ?
Available space in 256 byte pages:
TPA = 00D5H
Accept new buffer definitions (Y) ?
BIOS3 SPR F300H 0B00H
BDOS3 SPR D600H 1D00H
*** CP/M 3.0 SYSTEM GENERATION DONE
A>
```

5.2 Customizing the CPMLDR

The CPMLDR resides on the system tracks of a CP/M 3 system disk, and loads the CPM3.SYS file into memory to cold start the system. CPMLDR contains the LDRBDOS supplied by Digital Research, and must contain your customized LDRBIOS.

The system tracks for CP/M 3 contain the customized Cold Start Loader, CPMLDR with the customized LDRBIOS, and possibly the CCP.

The COPYSYS utility places the Cold Start Loader, the CPMLDR, and optionally the CCP on the system tracks, as shown in Table 5-1.

Table 5-1. Sample CP/M 3 System Track Organization

Track	Sector	Page	Memory Address	CP/M 3 Module Name
00	01		Boot Address	Cold Start Loader
00	02	00	0100H	CPMLDR
.	.	.	.	and
00	21	09	0A80H	LDRBDOS
00	22	10	0B00H	LDRBIOS
.	.	.	.	and
00	26	12	0D00H	
01	01	12	0D80H	
.	.	.	.	
01	26	25	1A00H	CCP

Typically the Cold Start Loader is loaded into memory from Track 0, Sector 1 of the system tracks when the reset button is depressed. The Cold Start Loader then loads CPMLDR from the system tracks into memory.

Alternatively, if you are starting from an existing CP/M 2 system, you can run CPMLDR.COM as a transient program. CP/M 2 loads CPMLDR.COM into memory at location

100H. CPMLDR then reads the CPM3.SYS file from User 0 on drive A and loads it into memory.

Use the following procedure to create a customized CPMLDR.COM file, including your customized LDRBIOS:

1) Prepare a LDRBIOS.ASM file.

2) Assemble the LDRBIOS file with RMAC to produce a LDRBIOS.REL file.

3) Link the supplied CPMLDR.REL file with the LDRBIOS.REL file you created to produce a CPMLDR.COM file.

A>LINK CPMLDR[Ll00]=CPNLDR,LDRBIOS

Replace the address 100 with the load address to which your boot loader loads CPMLDR.COM. You must include a bias of 100H bytes for buffer space when you determine the load address.

The CPMLDR requires a customized LDRBIOS to perform disk input and console output.

The LDRBIOS is essentially a nonbanked BIOS. The LDRBIOS has the same JMP vector as the regular CP/M 3 BIOS. The LDRBIOS is called only to perform disk reads (READ) from one drive, console output (CONOUT) for sign-on messages, and minimal system initialization.

The CPMLDR calls the BOOT entry point at the beginning of the LDRBIOS to allow it to perform any necessary hardware initialization. The BOOT entry point should return to CPMLDR instead of loading and branching to the CCP, as a BIOS normally does. Note that interrupts are not disabled when the LDRBIOS BOOT routine is called.

Test your LDRBIOS completely to ensure that it properly performs console character output and disk reads. Check that the proper tracks and sectors are addressed on all reads and that data is transferred to the proper memory locations.

You should assemble the LDRBIOS.ASM file with a relocatable origin of 0000H.

Assemble the LDRBIOS with RMAC to produce a LDRBIOS.REL file. Link the LDRBIOS.REL file with the CPMLDR.REL file supplied by Digital Research to create a CPMLDR.COM .file. Use the L option in LINK to specify the load origin (address) to which the boot loader on track 0 sector 1 loads the CPMLDR.COM file.

Unnecessary BIOS functions can be deleted from the LDRBIOS to conserve space. There is one absolute restriction on the length of the LDRBIOS; it cannot extend above the base of the banked portion of CP/M 3. (GENCPM lists the base address of CP/M 3 in its load map.) If you plan to boot CP/M 3 from standard, single-density, eight-inch floppy disks, your CPMLDR must not be longer than 1980H to place the CPMLDR.COM file on two system tracks with the boot sector. If the CCP resides on the system tracks with the Cold Start Loader and CPMLDR, the combined lengths must not exceed 1980H.

5.3 CPKLDR Utility

Syntax:

CPMLDR

Purpose:

CPMLDR loads the CP/M 3 system file CPM3.SYS into Bank 0 and transfers control to the BOOT routine in the customized BIOS. You can specify in GENCPM for CPMLDR to display a load table containing the names and addresses of the system modules.

The CPM3.SYS file contains the CP/M 3 BDOS and customized BIOS. The file CPM3.SYS must be on drive A in USER 0. You can execute CPMLDR under SID or DDT to help debug the BIOS. A $B in the default File Control Block (FCB) causes CPMLDR to execute a RST 7 (SID breakpoint) just before jumping to the CP/M 3 Cold Boot BIOS entry point.

Input File:

CPM3.SYS

Examples:

```
A>CPMLDR
CP/M V3.0 Loader
Copyright (C) 1982, Digital Research
BNKBIOS3 SPR F600H 0A00H
BNKBIOS3 SPR BB00H 0500H
RESBDOS3 SPR F100H 0500H
BNKBDOS3 SPR 9A00H 2100H
60K TPA
A>
```

In the preceding example, CPMLDR displays its name and version number, the Digital Research copyright message, and a four-column load table containing the filename, filetype, hex starting address, and length of the system modules. CPMLDR completes its sign-on message by indicating the size of the Transient Program Area (TPA) in kilobytes. The CCP then displays the system prompt, A>.

5.4 Booting CP/M 3

The CP/M 3 cold start operation loads the CCP, BDOS, and BIOS modules into their proper locations in memory and passes control to the cold start entry point (BIOS Function 0: BOOT) in the BIOS. Typically, a PROM-based loader initiates a cold start by loading sector 0 on track I of the system tracks into memory and jumping to it. This first sector contains the Cold Start Loader. The Cold Start Loader loads the CPMLDR.COM program into memory and jumps to it. CPMLDR loads the CPM3.SYS file into memory and jumps to the +BIOS cold start entry point.

To boot the CP/M 3 system, use the following procedure:

1) Create the CPM3.SYS file.

2) Copy the CPM3.SYS file to the boot drive.

3) Create a CPMLDR.COM for your machine.

4) Place the CPMLDR.COM file on your system tracks using SYSGEN with CP/M 2 or COPYSYS with CP/M 3. The boot loader must place the CPMLDR.Com file at the address at which it originated. If CPMLDR has been linked to load at 100H, you can run CPMLDR under CP/M 2.

The COPYSYS utility handles initialization of the system tracks. The source of COPYSYS is included with the standard CP/M 3 system because you need to customize COPYSYS to support nonstandard system disk formats. COPYSYS copies the Cold Start Loader, the CPMLDR.COM file, and optionally the CCP to the system tracks. Refer to the COPYSYS.ASM source file on the distribution disk.

Section 6 : Debugging the BIOS

This section describes a sample debugging session for a nonbanked CP/M 3 BIOS. You must create and debug your nonbanked system first, then bring up the banked system.

Note that your system probably displays addresses that differ from the addresses in the following example.

You can use SID, Digital Research' s Symbolic Debugger Program, running under CP/M 2.2, to help debug your customized BIOS. The following steps outline a sample debugging session.

1) Determine the amount of memory available to CP/M 3 when the debugger and CP/M 2.2 are in memory. To do this, load the debugger under CP/M 2.2 and list the jump instruction at location 0005H. In the following example of a 64K system, C500 is the base address of the debugger, and also the maximum top of memory that you can specify in GENCPM for your customized CP/M 3 system.

```
A>SID
CP/M 3 SID - Version 3.0
#L5
0005 JMP C500
```

2) Running under CP/M 2.2, use GENCPM to generate a CPM3.SYS file, which specifies a top of memory that is less than the base address of the debugger, as determined by the previous step. Allow at least 256K bytes for a patch area. In this example, you can specify C3 to GENCPM as the top of memory for your CP/M 3 system.

```
A>GENCPM
Top page of memory (FF)? C3
```

3) Now you have created a system small enough to debug under SID. Use SID to load the CPMLDR.COM file, as shown in the following example:

```
A>SID CP14LDR.COM
CP/M 3 SID - Version 3.0
NEXT MSZE PC END
0E80 0EB0 0100 D4FF
```

4) `#I$B`

5) Transfer control to CPMLDR using the G command:

```
#G
```

At this point, the screen clears and the following information appears:

```
CP/M V3.0 LOADER
Copyright (c) 1982, Digital Research
BIOS3 SPR AA00 0B00
BDOS3 SPR 8B00 1F00
34K TPA
01A9
```

6) With the CP/M 3 system in the proper location, you can set passpoints in your BIOS. Use the L command with the address specified as the beginning of the BIOS by the CPMLDR load table as shown in step 5 above. This L command causes SID to display the BIOS jump vector which begins at that address. The jump vector indicates the beginning address of each subroutine in the table. For example, the first jump instruction in the example below is to the Cold Boot subroutine.

```
#LAA00
```

The output from your BIOS might look like this:

```
JMP AA68
JMP AA8E
JMP ABA4
JMP ABAF
JMP ABCA
```

7) Now set a passpoint in the Cold BOOT routine. Use the P command with an address to set a passpoint at that address.

```
#PAA68
```

8) Continue with the CPMLDR.COM program by entering the G command, followed by the address of Cold Boot, the first entry in the BIOS jump vector.

```
#GAA00
```

9) In response to the G command, the CPMLDR transfers control to the CP/M 3 operating system. If you set a passpoint in the Cold BOOT routine, the program stops executing, control transfers to SID, and you can begin tracing the BOOT routine.

10) When you know the BOOT routine is functioning correctly, enter passpoints for the other routines you want to trace, and begin tracing step by step to determine the location of problems.

Refer to the Digital Research Symbolic Instruction Debugger User's Guide (SID) in the Programmer's Utilities Guide for the CP/M Family of Operating Systems for a discussion of all the SID commands.

Appendix A

Removable Media Considerations

All disk drives under CP/M 3 are classified as either permanent or removable. In general, removable drives support media changes; permanent drives do not. Setting the high-order bit in the CKS field in a drive's Disk Parameter Block (DPB) marks the drive as a permanent drive.

The BDOS file system distinguishes between permanent and removable drives. If a drive is permanent, the BDOS always accepts the contents of physical record buffers as valid. In addition, it also accepts the results of hash table searches on the drive.

On removable drives, the status of physical record buffers is more complicated.

Because of the potential for media change, the BDOS must discard directory buffers before performing most directory related BDOS function calls. This is required because the BDOS detects media changes by reading directory records. When it reads a directory record, the BDOS computes a checksum for the record, and compares the checksum to the currently stored value in the drive's checksum vector . If the checksum values do not match, the BDOS assumes the media has changed. Thus, the BDOS can only detect a media change by an actual directory READ operation.

A similar situation occurs with directory hashing on removable drives. Because the directory hash table is a memory-resident table, the BDOS must verify all unsuccessful hash table searches on removable drives by accessing the directory.

The net result of these actions is that there is a significant performance penalty associated with removable drives as compared to permanent drives. In addition, the protection provided by classifying a drive as removable is not total. Media changes are only detected during directory operations. If the media is changed on a drive during BDOS WRITE operations, the new disk can be damaged.

The BIOS media flag facility gives you another option for supporting drives with removable media. However, to use this option, the disk controller must be capable of generating an interrupt when the drive door is opened. If your hardware provides this support, you can improve the handling of removable media by implementing the following procedure:

1) Mark the drive as a permanent drive and set the DPB CKS parameter to th'e total number of directory entries, divided by four. For example, set the CKS field for a disk with 96 directory entries to 8018H.

2) Implement an interrupt service routine that sets the @MEDIA flag in the System Control Block and the DPH MEDIA byte for the drive that signaled the door open condition.

By using the media flag facility, you gain the performance advantage associated with permanent drives on drives that support removable media. The BDOS checks the System Control Block @MEDIA flag on entry for all disk-related function calls. If the flag has not been set, it implies that no disks on the system have been changed. If the flag is set, the BDOS checks the DPH MEDIA flag of each currently logged-in disk. If the DPH MEDIA flag of a drive is set, the BDOS reads the entire directory on the drive to determine whether the drive has had a media change before performing any other operations on the drive. In addition, it temporarily classifies any permanent disk with the DPH MEDIA flag set as a removable drive. Thus, the BDOS discards all directory physical record buffers when a drive door is opened to force all directory READ operations to access the disk.

To summarize, using the BIOS MEDIA flag with removable drives offers two important benefits. First, because a removable drive can be classified as permanent, performance is enhanced. Second, because the BDOS immediately checks the entire directory before performing any disk-related function an the drive if the drive's DPH MEDIA flag is set, disk integrity is enhanced.

Appendix B : Auto-density Support

Auto-density support refers to the capability of CP/M 3 to support different types of media on a single drive. For example, some floppy-disk drives accept single-sided and double-sided disks in both single-density and double-density formats. Auto-density support requires that the BIOS be able to determine the current density when SELDSK is called and to subsequently be able to detect a change in disk format when the READ or WRITE routines are called.

To support multiple disk formats, the drivers BIOS driver must include a Disk Parameter Block (DPB) for each type of disk or include code to generate the proper DPB parameters dynamically. In addition, the BIOS driver must determine the proper format of the disk when the SELDSK entry point is called with register E bit 0 equal to 0 (initial SELDSK calls). If the BIOS driver cannot determine the format, it can return 0000H in register pair HL to indicate the select was not successful. Otherwise, it must update the Disk Parameter Header (DPH) to address a DPB that describes the current media, and return the address of the DPH to the BDOS.

Note: all subsequent SELDSK calls with register E bit 0 equal to 1, the BIOS driver must continue to return the address of the DPH returned in the initial SELDSK call. The value 0000H is only a legal return value for initial SELDSK calls.

After a driver's SELDSK routine has determined the format of a disk, the driver's READ and WRITE routines assume this is the correct format until an error is detected. If an error is detected and the driver determines that the media has been changed to another format, it must return the value OFFH in register A and set the media flag in the System Control Block. This signals the BDOS that the media has changed and the next BIOS call to the drive will be an initial SELDSK call. Do not modify the drivers DPH or DPB until the initial SELDSK call is made.

Note that the BDOS can detect a change in media and will make an initial SELDSK call, even though the BIOS READ and WRITE routines have not detected a disk format change. However, the SELDSK routine must always determine the format on initial calls.

A drive's Disk Parameter Header (DPH) has associated with it several uninitialized data areas: the allocation vector, the checksum vector, the directory hash table, and physical record buffers. The size of these areas is determined by DPB parameters. If space for these areas is explicitly allocated in the BIOS, the DPB that requires the most space determines the amount of memory to allocate. If the BIOS defers the allocation of these areas to GENCPM, the DPH must be initialized to the DPB with the largest space requirements. If one DPB is not largest in all of the above categories, a false one must be constructed so that GENCPM allocates sufficient space for each data area.

Appendix C : Modifing a CP/M 2 BIOS

If you are modifying an existing CP/M 2.2 BIOS, you must note the following changes.

- The BIOS jump vector is expanded from 17 entry points in CP/M 2.2 to 33 entry points in CP/M 3. You must implement the necessary additional routines.
- The Disk Parameter Header and Disk Parameter Block data structures are expanded.

See Section 3 of this manual, "CP/M 3 BIOS Functional Specifications," for details of the BIOS data structures and subroutines. The following table shows all CP/M 3 BIOS functions with the changes necessary to support CP/M 3.

Table C-1. CP/M 3 BIOS Functions

Function	Meaning
BIOS Function 00: BOOT	The address for the JMP at location 5 must be obtained from @MXTPA in the System Control Block.
BIOS Function 01: WBOOT	The address for the JMP at location 5 must be obtained from @MXTPA in the System Control Block. The CCP can be reloaded from a file.
BIOS Function 02: CONST	Can be implemented unchanged.
BIOS Function 03: CONIN	Can be implemented unchanged. Do not mask the high-order bit.
BIOS Function 04: CONOUT	Can be implemented unchanged.
BIOS Function 05: LIST	Can be implemented unchanged.
BIOS Function 06: AUXOUT	Called PUNCH in CP/M 2. Can be implemented unchanged.
BIOS Function 07: AUXIN	Called READER in CP/M 2. Can be implemented unchanged. Do not mask the high-order bit.
BIOS Function 08: HOME	No change.
BIOS Function 09: SELDSK	Can not return a select error when SELDSK is called with bit 0 in register E equal to 1.
BIOS Function 10: SETTRK	No change.
BIOS Function 11: SETSEC	Sectors are physical sectors, not logical 128-byte sectors.
BIOS Function 12: SETDMA	Now called for every READ or WRITE operation. The DMA buffer can now be greater than 128 bytes.
BIOS Function 13: READ	READ operations are in terms of physical sectors. READ can return a 0FFH error code if it detects that the disk format has changed.
BIOS Function 14: WRITE	WRITE operations are in terms of physical sectors. If write detects that the disk is Read-Only, it can return error code 2. WRITE can return a 0FFH error code if it detects that the disk format has changed.
BIOS Function 15: LISTST	Can be implemented unchanged.
BIOS Function 16: SECTRN	Sectors are physical sectors, not logical 128-byte sectors.

The following is a list of new BIOS functions:

BIOS Function 17:	CONOST
BIOS Function 18:	AUXIST
BIOS Function 19:	AUXOST
BIOS Function 20:	DEVTBL
BIOS Function 21:	DEVINI
BIOS Function 22:	DRVTBL
BIOS Function 23:	MULTIO

BIOS Function 24:	**FLUSH**
BIOS Function 25:	**MOVE**
BIOS Function 26:	**TIME**
BIOS Function 27:	**SELMEM**
BIOS Function 28:	**SETBNK**
BIOS Function 29:	**XMOVE**
BIOS Function 30:	**USERF**
BIOS Function 31:	**RESERV1**
BIOS Function 32:	**RESERV2**

Appendix D : CPM3.SYS File Format

Table D-1. CPM3.SYS File Format

Record	Contents
0	Header Record (128 bytes)
1	Print Record (128 bytes)
2-n	CP/M 3 operating system in reverse order, top down.

Table D-2. Header Record Definition

Byte	Contents
0	Top page plus one, at which the resident portion of CP/M 3 is to be loaded top down.
1	Length in pages (256 bytes) of the resident portion of CP/M 3.
2	Top page plus one, at which the banked portion of CP/M 3 is to be loaded top down.
3	Length in pages (256 bytes) of the banked portion of CP/M 3.
4-5	Address of CP/M 3 Cold Boot entry point.
6-15	Reserved.
16-51	Copyright Message.
52	Reserved.
53-58	Serial Number.
59-127	Reserved.

The Print Record is the CP/M 3 Load Table in ASCII, terminated by a dollar sign ($).

Appendix E : Root Module of Relocatable BIOS for CP/M 3

All the listings in Appendixes E through I are assembled with , the cP/M Relocating Macro Assembler, and cross-referenced XREF , an assembly language cross-reference program used with . listings are output from the XREF program. The assembly sources are on your distribution disk as ASM files.

```
title 'Root module of relocatable BIOS for CP/M 3.0'

; version 1.0 15 Sept 82

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

 FFFF  = banked equu true


 ;      Copyright (C), 1982
 ;      Digital Research, Inc
 ;      P.O. Box 579
 ;      Pacific Grove, CA 93950


 ;      This is the invariant portion of the modular BIOS and is
 ;      distributed as source for informational purposes only.
 ;      All desired modifications should be performed by
 ;      adding or changing externally defined modules.
 ;      This allows producing "standard" I/O modules that
 ;      can be combined to support a particular system
 ;      configuration.

        000d  = cr equ 13
        000A  =     lf equ 10
        0007  = bell equ 7
        0011  = ctlQ equ 'Q'-'@'
        0013  = ctlS equ 'S'-'@'

        0100  = ccp equ 0100h          ; CCP gets loaded the TPA

 cseg  ; GENCPM puts CSEG stuff in common memory


        ; variables in system data page

 extrn @covec,@civec,@aovec.@aivec,@lovec ; I/O redirection vectors
 extrn @mxtpa                           ; addr of system entry point
 extrn @bnkbf                           ; 128 byte scratch buffer

        ; initialization

 extrn ?init                            ; general initialization and signon
```

```
extrn ?ldccp,?rlccp                 ; load & reload CCP for BOOT & WBOOT
      ; user defined character I/O routines
extrn 7ci,?co,?cist,?cost           ; each take device in <B>
extrn ?cinlt                        ; (re)initialize device in <C>
extrn @ctbl                         ; physical character device table
      ; disk communication data items
extrn @dtbl                         ; table of pointers to XDPHs
public @adrv,@rdrv,@trk,@sect       ; parameters for disk I/O
public @dma,@dbnk,@cnt              ;     "     "     "
      ; memory control
public @cbnk                        ; current bank
extrn ?xmove,?move                  ; select move bank, and block move
extrn ?bank                         ; select CPU bank
      ; clock support
extrn ?time                         ; siqnal time operation
      ; general utility routines
public ?pmsg,?pdec                  ; print message, print number from 0 to 65535
public ?pderr                       ; print BIOS disk error message header
maclib modebaud                     ; define mode bits
      ; External names for BIOS entry points
public ??boot,?wboot,?const,?conin,?cono,?list,?auxo,?auxi
public ?home,?sldsk,?sttrk ,?stsec,?stdma,7read,?write
public ?lists,?sctrn
public ?conos,?auxis,?auxos,?dvtbl,?devin,?drtbl
public 7mltio,?flush,?mov,7tim,?bnksl,7stbnk,?xmov
      ; BIOS Jump vector
      ' All BIOS routines are invoked by calling these
      ; entry points.
      ?boot: jmp boot               ; initial entry on cold start
      ?wboot. jmp wboot             ; reentry on program exit, warm start
      ?const. jmp const             ; return console input Status
      ?conin: jmp conin             ; return console input character
      ?cono: jmp conout             ; send console output character
      ?list: jmp list               ; send list output character
```

```
        ?auxo: jmp auxout              ; send auxilliary output character
        ?auxi: jmp auxin               ; return auxilliary input character

        ?home: jmp home                ; met disks to logical home
        ?sldsk: jmp seldek             ; select disk drive, return disk parm info
        ?sttrk: jmp settrk             ; Set disk track
        ?stsec: jmp setsec             ; set disk sector
        ?stdma: jmp setdma             ; set disk I/O memory address
        ?read: jmp read                ; read physical block(s)
        ?write: jmp write              ; write physical block(s)

        ?lists: jmp listat             ; return list device Status
        ?sctrn: jmp sectrn             ; translate logical to physical sector

        ?conos: jmp conost             ; return console output status
        ?auxis: jmp auxibt             ; return aux input status
        ?auxoS: jmp auxost             ; return aux output status
        ?dvtbl: jmp devtbl             ; return address of device def table
        ?devin: jmp ?cinit             ; change baud rate of device

        ?drtbl: jmp getdrv             ; return address of disk drive table
        ?mltio: jmp multio             ; Set multiple record count for disk I/O
        ?flush: jmp flush              ; flush BIOS maintained disk caching

        ?mov: jmp ?move                ; block move memory to memory
        ?tim: jmp ?time                ; Signal Time and date operation
        ?bnksl: jmp bnksel             ; select bank for code execution and DMA
        ?stbnk: jmp setbnk             ; select different bank for disk I/O DMA
        ?xmov:jmp ?xmove               ; set source and destination banks for one
        jmp   0                        ; reserved for system implementor
        jmp   0                        ; reserved for future expansion
        jmp   0                        ; reserved for future expansion


        ; BOOT
        ;     Initialentry point for SyStem startup.

dseg          ; this part can be banked

boot:
        lxi sp,boot$stack
        mvi c,15                       ; initialize all 16 character devices
c$init$loop:
        push b
        call ?cinit
        pop b
        dcr c
        jp c$init$loop

        call ?init                     ; perform any additiunal system initialization
                                       ; and print signon message

        lxi b,16*256+0
```

```
        lxi h,@dtbl                    ; Init all 16 logical disk drives
d$init$loop:
        push b                         ; save remaining count and abs drive
        mov e,m
        inx h
        mov d,m
        inx b                          ; grab @drv entry
        mov a,e
        ora d
        jz d$init$next                 ; if null, no drive
        push h                         ; save @drv pointer
        xchg                           ; XDPH address in HL
        dcx h
        dcx h
        mov a,m
        sta @RDRV                      ; get relative drive code
        mov a,c
        sta @ADRV                      ; get absolute drive code
        dcx h                          ; point to init pointer
        mov d,m
        dcx h
        mov e,m                        ; get init pointer
        xchg
        call ipchl                     ; call init routine
        pop h                          ; recover @drv pointer
d$init$next:
        pop b                          ; recover counter and drive #
        inr c
        dcr b
        jnz d$init$loop                ; and loop for each drive
        jmp boot$1

cseg        ; following in resident memory

boot$1:
        call set$jumps
        call ?ldccp                    ; fetch CCP for first time
        jmp ccp


        ; WBOOT
        ;     Entry for system restarts.

wboot:
        lxi sp,boot$stack
        call set$jumps                 ; initialize page zero
        call ?rlccp                    ; reload CCP
        jmp ccp                        ; then reset jmp vectors and exit to ccp


set$jumps:

if banked
```

```
        mvi a,1
        call ?bnksl
endif

        mvi a,JMP
        sta 0
        sta 5                       ; met up jumps in page zero
        lxi h,?wboot
        shld 1                      ; BIOS warm start entry
        lhld @MXTPA
        shld 6                      ; BDOS system call entry
        ret


        ds 64
        boot$stack equ $


        ; DEVTBL
        ;     Return address of character device table

devtbl:
        lxi h,@ctbl
        ret


        ; GETDRV
        ;     Return address of drive table

getdrv:
        lxi h,@dtbl
        ret



        ; CONOUT
        ;     Console Output. Send character in <C>
        ;      to all selected devices

conout;

        lhld @covec                 ; fetch console output bit vector
        jmp out$scan


        ; AUXOUT
        ;     Auxiliary Output. Send character in <C>
        :     to all selected devices

auxout:
        lhld @aovec                 ; fetch aux output bit vector
        jmp out$Scan

```

```

        ; LIST
        ;       List Output. Send character in <C>
        ;       to all selected devices.

list:
        lhld @lovec                     ; fetch list output bit vector

out$scan:
        mvi b,                          ; Start with device 0
co$next:
        dad h                           ; shift out next bit
        jnc not$out$device
        push h                          ; save the vector
        push b                          ; save the count and character
not$out$ready:
        call coster
        ora a
        jz not$out$ready
        pop b
        push b                          ; restore and resave the character and device
        call ?co                        ; if device selected, print it
        pop b                           ; recover count and character
        pop h                           ; recover the rest of the vector
not$out$device:
        inr b                           ; next device number
        mov a,h
        ora 1                           ; see if any devices left
        jnz co$next                     ; and go find them. . .
        ret


        ; CONOST
        ;       Console Output Status. Return true if
        ;       all selected console output devices
        ;       are ready.

conost:
        lhld @covec                     ; get console output bit vector
        jmp ost$scan


        ; AUXOST
        ;       Auxiliary Output Status. Return true if
        ;       all selected auxiliary output devices
        ;       are ready.

auxost:
        lhld @aovec                     ; get aux output bit vector
        jmp ost$scan


        ; LISTST
```

```
        ,     List Output Status. Return true if
        ;     all selected list output devices
        ;     are ready.

listst:
        lhld @lovec                 ; get list output bit vector

ost$scan:
        mvi b,0                     ; start with device 0
cos$next:
        dad h                       ; check next bit
        push h                      ; save the vector
        push b                      ; save the count
        mvi a,0FFh                  ; assume device ready
        cc coster                   ; check status for this device
        pop b                       ; recover count
        pop h                       ; recover bit vector
        ora a                       ; see if device ready
        rz                          ; if any not ready, return false
        dcr b                       ; drop device number
        mov a,b
        ora 1                       ; see if any more selected devices
        jnz cos$next
        ori 0FFh                    ; all selected were ready, return true
        ret

coster:     ; check for output device ready. including optional
        ; xon/xorf support
        mov l,b
        mvi h,0                     ; make device code 16 bits
        push h                      ; save it in stack
        dad h
        dad h
        dad h                       ; offset into device characteristics tbl
        lxi d,@ctbl+6
        dad d                       ; make address of mode byte
        mov a,m
        ani mb$xonxoff
        pop h                       ; recover console number in HL
        jz    ?cost                 ; not a xon device, go get output status direct
        lxi d,xofflist
        dad d                       ; make pointer to proper xon/xoff flag
        call cisti                  ; see if this keyboard has character
        mov a,m
        cnz cii                     ; get flag or read key if mny
        cpi ctlq
        jnz not$q                   ; if its a ctl-Q,
        mvi a,0FFh                  ; set the flag ready
not$g:
        cpi ctls
        jnz not$s                   ; if its a ctl-S,
        mvi a,00h                   ; clear the flag
not$s:
```

```
        mov m,a                         ; save the flag
        call costl                      ; get the actual output status,
        ana m                           ; and mask with ctl-Q/ctl-S flag
        ret                             ; return this am the status

cistl:      ; get input status with BC and HL saved
        push b
        push h
        call ?cist
        pop h
        pop b
        ora a
        ret

costl       ; get output status, saving BC & HL
        push b
        push h
        call ?cost
        pop h
        pop b
        ora a
        ret

cil:        ; get input, saving BC & HL
        push b
        push h
        call ?ci
        pop h
        pop b
        ret


        ; CONST
        ;     Console Input Status. Return true if
        ;      any selected console input device
        ;      has an available character.

const
        lhld @civec                     ; get console input hit vector
        jmp ist$scan


        ; AUXIST
        ;     Auxiliary Input   Status. Return true if
        ;      any selected auxiliary input device
        ;      has an available character.

auxist:
        lhld @aivec                     ; get aux input bit vector

ist$scan:
        mvi b,0                         ; start with device 0
cis$next:
```

```
        dad h                       ; check next hit
        mvi a,0                     , assume device not ready
        cc cistl                    ; check status for this device
        ora a
        rnz                         ; if any ready, return true
        inr b                       ; next device number
        mov a,h
        ora 1                       ; see if any more selected devices
        jnz cis$next
        xra a                       ; all selected were not ready. return false
        ret


        ; CONIN
        ;     Console Input.Return character from first
        ;     ready console input device.

conin:
        lhld @Civec
        jmp in$scan


        ; AUXIN
        ;     Auxiliary Input. Return character from first
        ;     ready auxiliary input device.

auxin:
        lhld @aivec

in$scan:
        push h                      ; save bit vector
        mvi b,0
ci$next:
        dad h                       ; shift out next bit
        mvi a,0                     ; insure 00 a (nonexistant device not ready)
        cc cisti                    ; see if the device has a character
        ora a
        jnz ci$rdy                  ; this device has a character
        dcr b                       ; else, next device
        mov a,h
        ora 1                       ; see if any more devices
        jnz ci$next                 ; go look at them
        pop h                       ; recover bit vector
        jmp in$scan                 ; loop til we find a character

ci$rdy:
        pop h                       ; discard extra stack
        jmp ?ci


        ;     Utility Subroutines


```

```
ipchl:     ; vectored CALL point
         pchl


         ?pmsg:      ; print message @HL up to a null
         ; saves BC & DE
         push b
         push d
pmsg$loop:
         mov a,m
         ora a
         jz pmsg$exit
         mov c,a
         push h
         call ?cono
         pop h
         inx h
         jmp pmsg$loop
pmsg$exit:
         pop d
         pop h
         ret

         ?pdec:      ; print binary number 0-65535 from HL
         lxi b,table10
         lxi d,-10000
next:
         mvi a, '0'-1
pdecl:
         push h
         inr a
         dad d
         jnc stoploop
         inx sp
         inx sp
         jmp pdecl
stoploop:
         push d
         push b
         mov c,a
         call ?cono
         pop b
         pop d
nextdigit:
         pop h
         ldax b
         mov e,a
         inx b
         ldax b
         mov d,a
         inx b
         mov a,e
         ora d
```

```
        jnz next
        ret

tabel10:
        dw      -1000,-100,-10,-1,0

        ?pderr:
        lxi h,drive$msg
        call ?pmsg                      ; error header
        lda @adrv
        adi 'A'
        mov c,a
        call ?cono                      ; drive code
        lxi h,track$msg
        call ?pmsg                      ; track header
        lhld @trk
        call ?pdec                      ; track number
        lxi h,sector$msg
        call ?pmsg                      ; sector header
        lhld @sect
        call ?pdec                      ; sector number
        ret


        : BNKSEL
        ;       Bank Select. Select CPU bank for further execution.

bnksel
        sta @cbnk                       ; remember current bank
        jmp ?bank                       ; and go exit through users
        ; physical bank select routine


        xofflist db -1,-1,-1,-1,-1,-1,-1,-1        ; ctl-s clears to zero
              db    -1,-1,-1,-1,-1,-1,-1,-1



dseg          ; following resides in banked memory



        ; Disk I/O interface routines


        ; SELDSK
        ;       Select Disk Drive. Drive code in <C>.
        ;       Invoke login procedure for drive
        ;       if this is first select. Return
        ;       address of disk parameter header
        ;       in HL

seldsk:
```

```
        mov a,c
        sta @adrv                    ; save drive select code
        mov l,c
        mvi h,0
        dad h                        ; create index from drive code
        lxi h,@dtbl
        dad b                        ; get pointer to dispatch table
        mov a,m
        inx h
        mov h,m
        mov l,a                      ; point at disk descriptor
        ora h
        rz                           ; if no entry in table, no disk
        mov a,e
        ani 1
        jnz not$first$select         ; examine login bit
        push h
        xchg                         ; put pointer in stack & DE
        lxi h,-2
        dad d
        mov a,m
        sta @RDRV                    ; get relative drive
        lxi h,-6
        dad d                        ; find LOGIN addr
        mov a,m
        inx h
        mov h,m
        mov l,a                      ; get addr of LOGIN routine
        call ipchl                   ; call LOGIN
        pop h                        ; recover DPH pointer
not$first$select:
        ret


        ;     HOME
        ;     Home selected drive. Treated as SETTRK(0).

home:
        lxi b,0                      ; same as set track zero


        ; SETTRK
        ;     Set Track. Saves track address from BC
        ;     in @TRE for further operations.

settrk:
        mov l,c
        mov h,b
        shld @trk
        ret


        ; SETSEC
```

```
        ;       Set Sector. Saves sector number from BC
        ;       in @sect for further operations.

setsec
        mov l,c
        mov h,b
        shld @sect
        ret


        ; SETDMA
        ;       Set Disk Memory Address. Saves DMA address
        ;       from BC in @DMA and sets @DBNK to @CBNK
        ;       so that further disk operations take place
        ;       in current bank.

setdma
        mov l,c
        mov h,b
        shld @dma

        lda @cbnk                     ; default DMA bank is current hank
        ; fall through to set DMA bank

        ; SETBNK
        ;       Set DiBk Memory Bank. Saves bank number
        ;       in @DBNK for future disk dBtB
        ;       transfers.

setbnk
        sta @dbnk
        ret


        ; SECTRN
        ;       Sector Translate. Indexes skew table in DE
        ;       with sector in BC. Returns physical Sector
        ;       in HL If no skew table (DE=0) then
        ;       returns physical=logical.

sectrn:
        mov l,c
        mov h,b
        mov a,d
        ora e
        rz
        xchg
        dad b
        mov l,m
        mvi h,0
        ret
```

```
        ; READ
        ;     Read physical record from currently selected drive,
        ;     Finds address of proper read routine from
        ;     extended disk parameter header (XDPH) .

read:
        lhld @adrv
        mvi h,0
        dad h                          ; get drive code and double it
        lxi d,@dtbl
        dad d                          ; make address of table entry
        mov a,m
        inx h
        mov h,m
        mov l,a                        ; fetch table entry
        push h                         ; save address of table
        lxi d,-8
        dad d                          ; point to read routine ddress
        jmp rw$common                  ; use common code


        ; WRITE
        ;     Write physical sector from currently selected drive.
        ;     Finds address of proper write routine from
        ;     extended disk parameter header (XDPH) .

write:
        lhld @adrv
        mvi h,0
        dad h                          ; get drive code and double it
        lxi d,@dtbl
        dad d                          ; make address of table entry
        mov a,B
        inx h
        mov h,m
        mov l,a                        ; fetch able entry
        push h                         ; save address of table
        lxi d,-10
        dad d                          ; point to write routine address

rw$common:
        mov a,m
        inx h
        mov h,m
        mov l,a                        ; get address of routine
        pop d                          ; recover address of table
        dcx d
        dcx d                          ; point to relative drive
        ldax d
        sta @rdrv                      ; get relative drive code and post it
        inx d
        inx d                          ; point to DPN again
        pchl                           ; leap to driver
```

```


        ; MULTIO
        ;      Set multiple sector count. Saves passed count in
        ;      @CNT

multio
        sta @cnt
        ret


        ; FLUSH
        ;      BIDS deblocking buffer flush. Not implemented.

flush:
        xra a
        ret                           ; return with no error



        ; error message components
        drive$msg db cr,lf,bell,'BIOS Error on ',0
        track$msg db ' T-',0
        sector$msg db ', S-',0


        ; disk communication data items

        @adrv ds     1                ; currently melected dimk drive
        @rdrv ds     1                ; controller relative disk drive
        @trk ds      2                ; current track number
        @eect ds     2                ; Current Sector number
        @dma ds      2                ; Current DMA address
        @cnt db      0                ; record count for multisectortransfer
        @dbnk db     0                ; bank for DMA operations


cseg        ; common memory

        @cbnk db     0                ; bank fOr processor operations



023C end
AUXIN                     0198  99    397#
AUXIST                    017D  113   367#
AUXOST                    010C  114   277#
AUXOUT                    00E0  98    230#
BANKED                    FFFF  8#    186
BAUD110                   0003
BAUD12000                 0008
BAUD134                   0004
BAUD150                   0005
```

```
BAUD1800           00009
BAUD19200          000F
BAUD2400           000A
BAUD300            0006
BAUD3600           000B
BAUD4800           000C
BAUD50             0001
BAUD600            0007
BAUD7200           000D
BAUD75             0002
BAUD9600           000E
BAUDNONE           0000
BELL               0007  27#   632
BNKSEL             0225  124   475#
BOOT               0000  91    138#
BOOT1              0063  164   168#
BOOTSTACK          00D2  139   178   1981
CCP                0100  31#   171   181
CI1                016F  319   345#
CINEXT             019E  403#  411
CINITLOOP          0005  141#  143
CIRDY              01B2  408   415#
CISNEXT            0182  372#  379
CIST1              015D  318   331#  375   406
CONEXT             00EB  244#  258
CONIN              0192  95    388#
CONOST             0106  112   267#
CONOUT             00DA  96    220#
CONST              0177  94    357#
COSNEXT            0117  292#  304
COST1              0166  327   3381
COSTER             012C  250   297
3081
CR                 000D  25#   632
CTLQ               0011  281   320
CTLS               0013  291   323
DEVTBL             00D2  115   2041
DINITLOOP          0017  1491  163
DINITNEXT          0036  152   1611
DRIVEMSG           00]D1       463   6321
FALSE              0000  6#
FLUSH              00CF  120   6261
GETDRV             00D6  118   211#
HOME               006E  101   5201
INSCAN             019B  390   4001  413
IPCHL              *01B6       159   4231  511
ISTSCAN            0180  359   3701
LF                 000A  26#   632
LIST               00E6  97    2391
LISTST             0112  109   2879
MBINOUT            0003
MBINPUT            0001
MBOUTPUT           0002
```

```
MBSERIAL            0008
MBSOFTBAUD          0004
MBXONXOFF           0010  314
MULTIO              00CB  119   6191
NEXT                01D1  443#  456
NEXTDIGIT           01E6  452#
NOTFIRSTSELECT      006D  506   513#
NOTOUTDEVICE        00FF  246   255#
NOTOUTREADY         00F1  249#  250
NOTQ                0150  320   322#
HOTS                0157  323   325#
OSTSCAN             0115  269   279   290#
OUTSCAN             00E9  223   232   242#
PDECL               01D3  445#  447
PMSGEXIT            01C8  432   436#
PMSGLOOP            01B9  431#  435
READ                0094  106   SBS#
RWCOMMON            00BD  591   606#
SECTORMSG           00E8  46?   634#
SECTRN              0089  110   573#
SELDSK              003F  102   500#
SETBNK              0085  125   562#
SETDMA              007D  105   550#
SETJUMPS            0078  169   179   184#
SETSEC              0077  104   538#
SETTRK              0071  103   528#
STOPLOOP            01DE  446   448#
TABLE10             01F3  442   459#
TRACKMSG            00E3  465   633#
TRUE                FFFF  S#    6     8
WBOOT               006C  92    177#
WRITE               00AA  107   599#
XOFFLIST            022B  317   481#
?AUXI               0015  79    99#
?AUXIS              0036  82    113#
?AUXO               0012  79    98#
?AUXOS              0039  82    114#
?BANK               0000  63    477
?BNKSL              0051  83    124#  187
?BOOT               0000  79    91#
?CI                 0000  49    347   417
?CINIT              0000  50    116   142
?CIST               0000  49    333
?CO                 0000  49    252
?CONIN              0009  79    95#
?CONO               000C  79    96#   434   450   464
?CONOS              0033  82    112#
?CONST              0006  79    941
?COST               0000  49    316   340
?DEVIN              003F  82    116#
?DRTBL              0042  82    118#
?DVTBL              003C  82    115#
?FLUSH              0048  83    1201
```

?HOME	0018	80	1011					
?INIT	0000	44	145					
?LDCCP	0000	45]	170					
?LIST	000F	79	97#					
?LISTS	002D	81	109#					
?MLTIO	0045	83	119#					
?MOV	004B	83	1221					
?MOVE	0000	62	122					
?PDEC	0ICB	71	4411	466	468			
?PDERR	0IFD	72	4621					
?PMSG	01B7	71	4271	463	465]	467		
?READ	0027	80	106#					
?RLCCP	0000	45	180					
?SCTRN	0030	81	1101					
?SLDSK	001B	80	1021					
?STBNK	0054	83	125#					
?STDMA	0024	80	1051					
?STSEC	0021	80	1041					
?STTRK	001E	80	1031					
?TIM	004E	83	1231					
?TIME	0000	67	123					
?WBOOT	0003	79	921	192				
?WRITE	002A	80	1071					
?XMOV	0057	83	1261					
?XMOVE	0000	62	126					
@ADRV	00ED	56	156	464	501	586	600	639#
@AIVEC	0000	38	368	398				
@AOVEC	0000	38	231	278				
@BNKBF	0000	40						
@CBNK	023B	61	476	554	650#			
@CIVEC	0000	38	358	389				
@CNT	00F5	57	620	6441				
@COVEC	0000	38	222	268				
@CTBL	0000	51	205	313				
@DBNK	00F6	57	563	645#				
@DMA	00F3	57	552	643#				
@DTBL	0000	55	148	212	503	587	601	
@LOVEC	0000	38	240	288				
@MXT'PA	0000	39	193					
@RDRV	00EE	56	155	508	610	640#		
@SECT	00F1	56	468	540	642#			
@TRK	00EF	56	466	530	641#			

Appendix F : SCB Definition for CP/M 3 BIOS

The SCB.ASM module contains the public definitions of the fields in the System Control Block. The BIOS can reference public variables.

```
title 'System Control Block Definition for CP/M3 BIOS'

public @civec, @covec, @aivec, @aovec, @lovec, @bnkbf
public @cradma, @crdsk, @vinfo, @resel, @fx, @usrcd
public @mltio, @ermde, @erdsk, @media, @bflgs
public @date, @hour,@min, @sec, ?erjmp, @mxtpa


 FE00 = srb*base equ     0FE00H       , Baseof the SCB

FE22  =      @CIVEC equ scb$base+22h ; Console Input Redirection
                                     ; vector (word. r/w)
FE24  =      @COVEC equ scb$base+24h ; Console Output Redirection
                                     ; Vector (word, r/w)
FE26  =      @AIVEC equ scb$base+26h ; Auxiliary Input Redirection
                                     ; Vector (word, r/w)
FE2B  =      @AOVEC equ scb$base+28h ; Auxiliary Output Redirection
                                     ; Vector (word. r/w)
FE2A  =      @LOVEC equ scb$base+2Ah ; List Output Redirection
                                     ; Vector (word, r/w)
FE35  =      @BNKBF equ scb$base+35h ; Address of 128 Byte Buffer
                                     ; for Banked BIOS (word, r/o)
FE3C  =      @CRDMA equ scb$base+3Ch ; Current DMA Address
                                     ; (word. r/o)
FE3E  =      @CRDSK equ scb$base+3Eh ; Current Disk (byte. r/o)
FE3F  =      @VINFO equ scb$base+3Fh ; BDOS Variable "INFO"
                                     ; (word, r/o)
FE41  =      @RESEL equ scb$base+41h ; FC0 Flag (byte, r/o)
FE43  =      @FX equ scb$base+43h    ; BDOS Function )br Error
                                     ; Message (byte, r/o)
FE44  =      @USRCD equ scb$base+44h ; current User Code (byte, r/o)
FE4A  =      @MLTIO equ scb$base+4Ah ; Current Multi-Sector Count
                                     ; (byte. r/w)
FE4B  =      @ERMOE equ scb$base+4Bh ; BDOS Error Mode (byte. r/o)
FE51  =      @BROSK equ scb$base+51h ; BDOS Error Disk (byte.r/o)
FE54  =      @MED!A equ scb$base+54h ; Set by BIOS to indicate
                                     ; open door (byte. r/w)
FE57  =      @BFLOS equ scb$base+57h ; BDOS Message Size Flag (byte. r/o)
FE58  =      @iDATE equ scb$base+58h ; Date in Days Since I Jan 78
                                     ; (word, r/w)
FE5A =       @HOUR equ scb$base+5Ah  ; Hour in BCD (byte. r/w)
FE5B =       @MIN equ scb$base+58h   ; Minute in BCD (byte. r/w)
FE5c  =      @SEC equ scb$base+5Ch   ; Second in BCID (byte. r/w)
FE5F  =      @ERJMP equ scb$base+5Fh ; BDOS Error "Message Jump
                                     ; (word. r/w)
FE62  =      @MXTPA equ scb$base+62h ; Top of User TPA
                                     ; (address at 6, 7)(word, r/o)
```

```
4B 0000 end
```

Appendix G : Equates for Mode Byte Bit Fields

```
; equates for mode byte bit fields
mb$input       equ    0000$0001b        ; device may do input
mb$output      equ    0000$0010b        ; device may do output
mb$in$out      equ    mb$input+mb$output
mb$soft$baud   equ    0000$0100b        ; software selectable baud rates
mb$serial      equ    0000$1000b        ; device may use protocol
mb$xon$xoff    equ    0001$0000b        ; XON/XOFF protocol enabled
baud$none      equ    0                 ; no baud rate associated with device
baud$50        equ    1                 ; S0 baud
baud$75        equ    2                 ; 75 baud
baud$110       equ    3                 ; 110] baud
baud$134       equ    4                 ; 134.5 baud
baud$150       equ    5                 ; 150] baud
baud$300       equ    6                 ; 300 baud
baud$600       equ    7                 ; 600 baud
baud$1200      equ    8                 ; 1200 baud
baud$1800      equ    9                 ; 1900 baud
baud$2400      equ    10                ; 2400 baud
baud$3600      equ    11                ; 3600 baud
baud$4800      equ    12                ; 4800 baud
baud$7200      equ    13                ; 7200 baud
baud$9600      equ    14                ; 9600 baud
baud$19200     equ    15                ; 19.2k baud
```

Listing G-1. Equates for Node Byte Fields: NODEBAUD.LIB

Appendix H : Macro Definitions for CP/M 3 BIOS Data Structures

```
; dtbl   <dph0,dph1,...>    - drive table
; dph translate$table,      - disk parameter header
; disk$Parameter$block,
; checksum$size (optional)
; alloc$size (optional)
; skew sectors              - skew table
; skew$factor
; first$sector$number
; dpb physical$sector$size - disk parameter block
; physical$sectors$per$track
; number$tracks
; block$size
; number$dir$entries
; track$offset
; checksum$vec$size (optional)
; Drive Table. Contains 16 one word entries.
dtbl     macro ?list
   Local ?n
   ?n     set 0
          irp   ?drv,<?list>
   ?n     set ?n+1
                dw    ?drv
          endm

          if ?n > 16
             .'Too many drives. Max 16 allowed'
             Exitm
          Endif

          if ?n < 16
                rept (16-?n)
                      dw 0
                endm
          endif
   endm

dph      macro ?trans,?dpb,?csize,?asize
   local ?csv,?alv
          dw ?trans                 ; translate table address
          db 0,0,0,0,0,0,0,0,0      ; BDOS Scratch area
          db 0                      ; media flag
          dw ?dpb                   ; disk parameter block
   if not nul ?csize
          dw ?csv                   ; checksum vector
   else
          dw 0FFFEh                 ; checksum vector allocated by GENCPM
   endif
   if not nul ?asize
```

```
		dw ?alv			; allocation vector
	else
		dw 0FFFEh		; alloc vector allocated by GENCPM
	endif
		dw 0fffeh,0fffeh,0fffeh ; dirbeb, dtabcb, hash alloc'd by GENCPM
		db 0			; hash bank
	if not nul ?csize
		?csv ds ?csize		; checksum vector
	endif
	if not nul ?asize
		?alv ds ?asize		; allocation vector
	endif
endm

dpb macro ?psize,?pspt,?trks,?bls,?ndirs,?off,?ncks
	local ?spt,?bsh,?blm,?exu,?dsm,?drm,?al0,?all,?cks,?psh,?psm
	local ?n
;; physical sector mask and physical sector shift
	?psh	set 0
	?n	set ?psize/128
	?psm	Set ?n-1
		rept 8
		?n	set ?n/2
			if ?n = 0
			exitm
			endif
		?psh	set ?psh + 1
		endm
	?spt	set ?pspt*(?psize/128)
	?bsh	set 3
	?n	set ?bls/1024
		rept 8
		?n	set ?n/2
			if ?n - 0
			exits
			endif
		?bsh	set ?bsh + 1
		endm
	?blm	set ?bls/128-1
	?size set (?trks-?Off)*?spt
	?dsm	set ?size/(?bls/128)-1
	?exm	set ?bls/1024
		if ?dsm > 255
			if ?bls - 1024
			.'Error, can''t have this size disk with 1k block size'
			exitm
			endif
		?exm	set ?exm/2
		endif
	?exm	set ?exm-1
	?all	set 0
	?n	set	(?ndirs*32+?bls-1)/?bls
		rept ?n
```

```
            ?all  set (?all shr 1) or 8000h
            endm
      ?al0  set high ?all
      ?all  set low ?a11
      ?drm  set ?ndirs-1
      if not nul ?ncks
            ?cks  set ?ncks
      else
            ?cks  set ?ndirs/4
      endif
            dw ?spt                 ; 128 byte records per track
            db ?bsh,?blm            ; block shift and mask
            db ?exm                 ; extent mask
            dw ?dsm                 ; maximum block number
            dw ?drm                 ; maximum directory entry number
            db ?al0,?all            ; alloc vector for directory
            dw ?cks                 ; checksum size
            dw ?0ff                 ; offset for system tracks
            db ?psh,?psm            ; physical sector size shift and mask
         endm

   ;
   gcd macro ?m,?n
         ;; greatest common divisor of m,n
         ;; produces value gcdn as result
         ;; (used in sector translate table generation)
      ?gcdm set ?m      ;;variable for m
      ?gcdn set ?n      ;;vsrisble for n
      ?gcdr set 0       ;;variable for r
            rept 65535
            ?gcdx set ?gcdm/?gcdn
            ?gcdr set ?gcdm - ?gcdx*?gcdn
                  if ?gcdr = 0
                  exitm
                  endif
            ?gcdm set ?gcdn
            ?gcdn set ?gcdr
            endm
      endm


   skew macro ?secs,?skf,?fsc
      ;; generate the translate table
      ?nxtsec     set 0 ;;next sector to fill
      ?nxtbas     set 0 ;;moves by one on overflow
      gcd %?secs,?skf
      ;; ?gcdn - gcd(?secs,skew)
      ?neltst     set ?secs/?gcdn
      ;; neltst is number of elements to generate
      ;; before we overlap previous elements
      ?nelts      set ?neltst ;;counter
                  rept ?secs  ;;once for each sector
                  db ?nxtsec+?fsc
```

```
                ?nxtsec     set ?nxtsec+?skf
                      if ?nxtsec >= ?secs
                      ?nxtsec     set ?nxtsec-?secs
                      endif
                ?nelts      set ?nelts-1
                      if ?nelts = 0
                      ?nxtbas     set ?nxtbas+l
                      ?nxtsec     set ?nxtbas
                      ?nelts      set ?neltst
                      endif
                endm
endm
```

Appendix I : ACS 8000-15 BIOS Modules

I.1 Boot Loader Module for CP/M 3

The BOOT.ASM module performs system initialization other than and disk I/O. BOOT loads the CCP for cold starts and it for warm starts. Note that the device drivers in the Research sample BIOS initialize devices for a polled, and an interrupt-driven, environment.

```
title 'Boot loadar module for Cp/M 3.0'

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

 FFFF  = banked equ true

public ?init,?ldccp,?rlccp,?time
extrn  ?pmsg,?conin
extrn @civec,@covec,@aivec,@aovec,@lovec
extrn @cbnk,?bnksl

maclib ports
maclib z80

0005 = bdos equ 5

if banked
      0001 = tpa$bank equ 1
else
      tpa$bank equ 0
endif

dseg  ; init done from banked memory

?init:
      lxi h,08000h
      shld @civec
      shld @covec                  ; assign console to CRT:
      lxi h,04000h
      shld @lovec                  ; assign printer to LPT:
      lxi h,02000h
      shld @aivec
      shld @aovec                  ; assign AUX to CRTl:
      lxi h,init$table
      call out$blocks              ; set up misc hardware
      lxi h,signon$msg
      call ?pmsg                   ; print signon message
      ret

out$blocks:
      mov a,m
      ora a
      rz
```

```
        mov b,a
        inx h
        mov c,m
        inx h
        outir
        DB    0EDH,0B3H
        jmp out$blocks


cseg    ; boot loading most be done from resident memory

;       This version of the boot loader loads the CCP from a file
;       called CCP.COM on the system drive (A:).


?ldccp:
; First time, load the A:CCP,COM file into TPA
        xra a
        Sta ccp$fcb+15                  ; zero extent
        lxi h,0
        shld fcb$nr                     ; start at beginning of file
        lxi d,ccp$fcb
        call open                       ; open file containing CCP
        inr a
        jz no$CCP                       ; error if no file. . '
        lxi d,0100h
        call setdma                     ; Start of TPA
        lxi d,128
        call setmulti                   ; allow up to 16k bytes
        lxi d,ccp$fcb
        call read                       ; load the thing
; now,
; copy CCP to bank 0 for reloading
        lxi h;0100h
        lxi b,0C80h                     ; clone 3.125K, just in case
        lda   @cbnk
        push psw                        ; save current bank
 ld$1:
        mvi a,tpa$bank
        call ?bnks                      ; select TPA
        mov a,m
        push psw                        ; get a byte
        mvi a,2
        call ?bnksl                     ; select extra bank
        pop psw
        mov m,s                         ; save the byte
        inx h
        dcx b                           ; bump pointer, drop count
        mov a,b
        ora c                           ; test for done
        jnz ld$1
        pop psw
        call ?bnksl                     ; restore original bank
```

```
        ret

no$CCP:     ; here 1f we couldn't find the file
        lxi h,ccp$msg
        call ?pms                    ; report this. . .
        call ?conin                  ; get a response
        jmp   ?ldccp                 ; and try again


?rlccp:
        lxi h,0100h
        lxi b,0C80h                  ; clone 3.125K
 rl$1:
        mvi a,2
        call ?bnksl                  ; select extra bank
        mov a,m
        push psw                     ; get a byte
        mvi a,tpa$bank
        call ?bnksl                  ; select TPA
        pop psw
        mov m,a                      ; save the byte
        inx h
        dcx b                        ; bump pointer, drop count
        mov a,b
        ora c                        ; test for done
        jnz rl$1
        ret

; No external clock.
?time:
        ret

; CP/M BDOS Function Interfaces

open:
        mvi c,15
        jmp bdos                     ; open file control block

setdma:
        mvi c,26
        jmp bdos                     ; set data transfer address

setmulti:
        mvi c,44
        jmp bdos                     ; set record count

read:
        mvi c,20
        jmp bdos                     ; read records


        signon$msg  db   13,10,13,10,'CP/M Version 1.0, sample BIOS',13,10,0

```

```
111     ccp$msg     db    13,10,'BIOS Err on A: No CCP.COM file',0
112
113
114     ccp$fcb     db    1,'CCP ','COM',0,0,0,0
115     ds    16
116     fcb$nr      db    0,0,0
117
118     init$table  db    3,p$zpio$3a,0CFh,0FFh,07h     ; set up config port
119                 db    3,p$zpio$3b,0CFh,000h,07h     ;set up bank port
120                 db    1,p$bank$select,0             ; select bank 0
121                 db    0            ; end of init$table
122
123     end
BANKED         FFFF  6#    18
BC             0000
BDOS           0005  16#   97    100   103   106
CCPFCB         00CC  50    52    56    114#
CCPMSG         00AB  73 111#
BE             0002
FALSE          0000  4#
FCBNR          00BC  51 116#
EL             0004
INITTABLE      00EF  30    119#
IX             0004
IY             0004
1,01           0030  61#   68
NOCCP          004A  53    72#
OPEN           0073  52    96#
OUTBLOCKS00    25    30    34#   39
PBANKSELECT    0025  120
PBAUDCON1      000C
PBAUDCON2      0030
PBAUDCON34     0031
PBAUDLPT1      000E
PBAUDLPT2      0032
PBOOT          0014
PCENTDATA      0011
PCENTSTAT      0010
PCON2DATA      002C
PCON2STAT      002D
PCON3DATA      002E
PCON3STAT      0021
PCON4DATA      002A
PCON4STAT      002B
PCONFIGURATION 0024
PCRTDAT'A      001C
PCRTSTAT       001D
PFDCMND        0004
PFDDATA        0007
PFDINT         0008
PFDMISC        0009
PFDSBCTOR      000E
PFDSTAT        0004
```

```
PFDTRACK     0005
PINDEX       0001
PLPT2DATA    0028
PLPT2STAT    0029
PLP'TDATA    001E
PLPTSTAT     0011
PRTC         0033
PSELECT      0008
PWD1797      0004
PZCTCl       000C
PZCTC2       0030
PZDART       00IC
PZDNA        0000
PZPIO1       0008
PZPI01A      000A
PZPIO1B      000B
PZPIO2       0010
PZPIO2A      0012
PZPIO2B      0013
PZPIO3       0024
PZPIO3A      002E  118
PZPIO3B      0027  119
PZSIO1       0028
PZSIO2       002C
RIAD         0082  56    105#
RL1          005C  80#   87
SETDMA       0078  54    99#
SETMULTI     007D  55    102#
SIGNONMSG    0087  31    109#
TPABANK      0001  19#   21#   62    83
TRUE         FFFF  3#    4     6
?BNKSL       0000  11    62    64    69    81    83
?CONIN       0000  9     74
?INIT        0000  8     26#
?LDCCP       0000  8     484   75
?PMSG        0000  9     31    73
?RLCCP       0056  8     78#
?TINE        0072  8     914
@AIVEC       0000  10    29
@AOVEC       0000  10    29
@CBNK        0000  11    60
@CIVEC       0000  10    27
@COVEC       0000  10    27
@LOVEC       0000  10    28
```

I.2 : Character I/O Handler for Z80 Chip--based System

The CHARIO.ASM module performs all character device , input, output, and status polling. CHARIO contains character device characteristics table.

```
title 'Character I/O handler for z80 chip based system'

  ; Character I/O for the Modular CP/M 3 BIOS
```

```

; limitations:

; haud rates 19200;7200,3600,1800 and 134
; are approximations.

;      9600 is the maximum baud rate that is likely
; to work,
; haud rates 50, 75, and 110 are not supported


public      ?cinit,?ci,?co,?cist,?cost
public      @ctbl

maclib Z80                      ; define Z80 op codes
maclib ports                    ; define port addresses
maclib modebaud                 ; define mode bits and baud equates

0006 = max$devices equ 6

cseg

?cinit:
        mov a,c
        cpi max$devlces
        jz cent$init            ; init parallel printer
        rnc                     ; invalid device
        mov 1,c
        mvi h,0                 ; make 16 bits from device number
        push h                  ; save device in stack
        dad h
        dad h
        dad h                   ; *8
        lxi d,@ctbl+7
        dad d
        mov 1,m                 ; get baud rate
        mov a,l
        cpi baud$600            ; see if baud > 300
        mvi a,44h
        jnc hi$speed            ; if > 600, use *16 mode
        mvi a,0C4h              ; else, use *64 mode
hi$speed:
        sta sio$reg$4
        mvi h,0
        lxi d,speed$table
        dad d                   ; point to counter entry
        mov a,m
        sta speed               ; get and save ctc count
        pop h                   ; recover
        lxi d,data$ports
        dad d                   ; point at SIO port address
        mov a,m
        inr a
```

```
        sta sio$port                    ; get and save port
        lxi d,baud$ports-data$ports
        dad d                           ; offset to baud rate port
        mov a,B
        sta ctc$port                    ; get and save
        lxi h,serlal$init$tbl
        jmp stream$out

cent$init:
        lxi h,pio$init$tbl

stream$out:
        mov a,m
        ora a
        rz
        mov b,a
        inx h
        mov c,m
        inx h
        outir
        DB    0EDH,0B3H
        jmp stream$out


?ci:  ; character input

        mov a,b
        cpi 6
        jnc null$input                  ; can't read from centronics
cil:
        call ?cist
        jz cii                          ; wait for character ready
        dcr c
        inp a                           ; get data
        DB    0EDH,A*8+40H
        ani 7Fh                         ; mask parity
        ret

null$1nput:
        mvi a,lAh                       ; return a ctl-Z for no device
        ret

?ciat:      ; character input Status

        mov a,b
        cpi 6
        jnc null$status                 ; can't read from centronics
        mov l,b
        mvi h,0                         ; make device number 16 bits
        lxi d,data$ports
        dad                             ; make pointer to port address
        mov c,m
        inr c                           ; get SIO status port
```

```
      inp a                        ; read from status port
      DB    0EDH,A*8+40H
      ani 1                        ; isolate RxRdy
      rz                           ; return with zero
      ori 0FFh
      ret

null$status:
xra a
      ret

?co:  ; character output
      mov a,b
      cpi 6
      jz centronics$out
      jnc null$Output
      mov a,c
      push psw                     ; save character from <C>
      push b                       ; save device number
co$spin:
      call ?cost
      jz co$spin                   ; wait for TxEmpty
      pop h
      mov l,h
      mvi h,0                      ; get device number in HL
      lxI d,data$ports
      dad d                        ; make address of port address
      mov c,m                      ; get port address
      pop psw
      outp a                       ; send data
      DB    0EDH,A*8+41H
null$output:
      ret

centronics$out:
      in p$centstat
      ani 20h
      jnz csntronics$out
      mov a,c
      out p$centdata               ; give printer data
      in p$centstat
      ori 1
      out p$centstat               ; set strobe
      ani 7Eh
      out p$centstat               ; clear strobe
      ret

      ?cost:      ; character output status
      mov a,b
      cpi 6
      jz cent$stat
      jnc null$status
      mov l,b
```

```
        mvi h,0
        lxi d,data$ports
        dad d
        mov c,m
        inr c
        inp a                       ; get input status
        DB    0EDH,A*8+40H
        ani 4
        rz                          ; test transmitter empty
        ori 0FFh
        ret                         ; return true if ready


cent$stat:
        in p$centstat
        cas
        ani 20h
        rz
        ori 0FFh
        ret

baud$ports:       ; CTC ports by physical device number
        db p$baud$conl , p$baud$lptl , p$baud$con2, p$baud$con 34
        db p$baud$con34,p$baud$lpt2

data$ports:       ; serial base ports by physical device number
        db p$crt$data,p$lpt$data,p$con2data,p$con3data
        db p$con4data,p$lpt2data


        @ctbl db 'CRT '             ; device 0, CRT port 0
        db mb$in$out+mb$serial+mb$softbaud
        db baud$9600
        db 'LPT '                   ; device 1, LPT port 0
        db mb$in$out+mb$serial+mb$softbaud+mb$xonxoff
        db baud$9600
        db 'CRT1 '                  ; device 2, CRT port 1
        db mb$in$out+mb$serlal+mb$softbaud
        db baud$9600
        db 'CRT2 '                  ; device 3, CRT port 2
        db mb$'in$out+mb$serial+mb$softbaud
        db baud$9600
        db 'CRT3 '                  ; device 4, CRT port 3
        db 'mb$in$out+mb$serial+mb$softbaud
        db baud$9600
        db 'VAX '                   ; device 5, LPT port 1 used for VAX interface
        db mb$in$out+mb$serial+mb$softbaud
        db baud$9600
        db 'CEN '                   ; device 6, Centronics parallel printer
        db mb$output
        db baud$none
        db 0  ; table terminator
```

```
        Speed$table db 0,255,255,255,233,208,104,208,104,69,52,35,26,17,13,7
    serial$init$tbl
        db     2                        ; two bytes to CTC
        ctc$port ds I                   ; port addresS of CTC
        db     47h                      ; CTC mode byte
        Speed ds     1                  ; baud multiplier
        db     7                        ; 7 bytes to SIO
        Sio$port ds 1                   ; port address of SIO
        db     18h,3,0Elh,4
        sio$reg$4 ds      1
        db     5,0EAh
        db     0                        ; terminator
        pio$init$tbl db   2,p$zpio$2b,0Fh,07h
        db     3,p$zpio$2a,0CFh,0F8h,07h
        db 0
        end
BAUDII0        0003
BAUD1200       0008
BAUD134        0004
BAUD150        0005
BAUD1800       0009
BAUD19200      000F
BAUD2400       000A
BAUD300        0006
BAUD3600       000B
BAUD4800       000C
BAUD50         0001
BAUD600        0007  34
BAUD7200       0000
BAUD75         0002
BAUD9600       000E  140   143   146   149   152   155
BAUDNONE       0000  158
BAUDPORTS      0006  44    129#
BC             0000
CENTINIT       0042  28    49#
CENTRONICSOUT  00911 91    105#  106
CENTSTAT       00C0  113   124#
CII            0057  63#   64
COSPIN         008B  95#   96
CTCPORT        012C  45    166#
DATAPORTS      00DC  42    44    78    98    116   133#
DE             0002
HISPEED        001D  35    37#
HL             0004
IX             0004
IY             0004
MAXDEVICES     0006  23#   28
MBINOUT        0003  139   142   145   148   151   154
MBINPUT        0001
```

MBOUTPUT	0002	157					
MBSERIAL	000B	139	142	145	148	151	154
MBSOFTBAUD	0004	139	142	145	148	151	154
MBXONXOFF	0010	142					
NULLINPUT	0063	62	70#				
NULLOUTPUT	0090	92	102#				
NULLSTATUS	0070	76	87#	114			
PBANKSELECT	0025						
PBAUDCON1	000C	130					
PBAUDCON2	0030	130					
PBAUDCON34	0031	130	131				
PBAUDLPT1	000E	130					
PBAUDLPT2	0032	131					
PBOOT	0014						
PCENTDATA	0011	107					
PCENTSTAT	0010	106	108	108	109	125	
PCON2DATA	002C	134					
PCON2STAT	002D						
PCON3DATA	002E	134					
PCON3STAT	002F						
PCON4DATA	002A	135					
PCON4STAT	002B						
PCONFIGURATION	0024						
PCRT'DATA	001C	134					
PCRTSTAT	001D						
PFDCMBD	0004						
PFDDATA	0007						
PFDINT	0008						
PFDMISC	0009						
PFDSECTOR	0006						
PFDSTAT	0004						
PFDTRACK	0005						
PINDEX	000F						
PIOINITTBL	0139	50	176#				
PLPT2DATA	0028	135					
PLPT2ST'AT	0029						
PLPTDATA	001E	134					
PLPTSTAT	001F						
PRTC	0033						
PSELECT	0008						
PWD1797	0004						
PZCTC1	000C						
PZCTC2	0030						
PZDART	001C						
PZDMA	0000						
PZPIO1	0008						
PZPIO1A	000A						
PZPIO1B	000B						
PZPIO2	0010						
PZPIO2A	0012	177					
PZPIO2B	0013	176					
PZPIO3	0024						
PZPIO3A	0026						

```
PZPIO3B        0027
PZSIO1         0028
PZSIO2         002C
SERIALINITTBL  012B  46    164#
SIOPORT        0130  43    170#
SIOREG4        0135  38    172#
SPEED          012E  40    168#
SPEEDTABLE     011B  39    162#
STREAMOUT      0045  47    52#   57
?CI            0051  16    60#
?CINIT         0000  16    27#
?CIST          0066  16    64    74#
?CO            007F  16    90#
?COST          0083  16    96    112#
@CTBL          00E2  17    33    138#
```

I.3 : Drive Table

The DRVTBL.ASM module points to the data structures for each configured disk drive. The drive table determines which physical disk unit is associated with which logical drive. The data structure for each disk drive is called an Extended Disk Parameter

Header (XDPH) .

```
public @dtbl
extin fdsd0,fdsdl
cseg
      @dtbl dw fdsd0,fdsdl
      dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0 ; drives C-P non-existent
 end
FDSDO     0000  2     6
FDSDl     0000  2     6
@DTBL     0000  1     6#
```

Listing I-3. Drive Table

I.4 Z80 DMA single-density Disk Handler

The FDl797SD module initializes the disk controllers for thedisks described in the Disk Parameter Headers and Disk Parameter Blocks contained in this module. FDl797SD is written for hardware that supports Direct Memory Access (DMA) .

```
title 'wd1797 w/ Z80 DMA Single density diskette handler'
 ;      CP/M-80 Version 3 -- Modular BIOS
 ;      Disk I/O Module for wd1797 based diskette systems
 ;      Initial version 0.01,
 Single density floppy only.   - jrp, 4 Aug
```

```

dseg

; Disk drive dispatc))ing tables for linked BIOS

public fdsd0,fdsdl

; Variables containing parameters passed by BDOS

extrn @adrv,@rdrv
extrn @dma,@trk,@sect
extrn @dbnk

; System Control Block variables

extrn @ermde                    ; BDOS error mode

; Utility routines in standard BIOS

extrn ?wboot                    ; warm hoot vector
extrn ?pmsg                     ; printmessage @HL up to 00, saves BC & DE
extrn ?pdec                     ; print binary number in A from 0 to 99,
extrn ?pderr                    ; print BIOS disk error header
extrn ?conin,?cono              ; con in and out
extrn ?const                    ; get console status


; Port Address Equates

saclib ports

; CP/M 3 Disk derinition macros

maclib cpm3

; Z80 macro library instruction definitions

maclib z80

; common control characters

000b = cr equ 13
000A = lf equ 10
0007 = bell equ 7


; Extended  Disk Parameter Headers (XPDNS)

      dw fd$write
      dw fd$read
      dw fd$login
      dw fd$init0
      db    0,0                 ; relative drive zero
```

```
        fdsd0 dph trans,dpbsd,16,31
        DW TRANS                      ; TRANSLATE TABLE ADDRESS
        DB 0,0,0,0,0,0,0,0,0          ; BDOS SCRATCH AREA
        DB 0                          ; MEDIA FLAG
        DW DPBSD                      ; DISK PARAMETER BLOCK
        DW ??0001                     ; CHECKSUM VECTOR
        DW ??0002                     ; ALLOCATION VECTOR
        DW 0FFFEH,0FFFEH,0FFFEH       ; DIRBCB, DTABCB, HASH  ALLOC'D BY GENCPM
        DB 0                          ; HASH BANK
        DS    16                      ; CHECKSUM VECTOR
        DS    31                      ; ALLOCATION VECTOR

        dw fd$write
        dw fd$read
        dw fd$login
        dw fd$initl
        db    1,0                     ; relative drive one
        fdsdl dph trans,dpbsd,16,31
        DW TRANS                      ; TRANSLATE TABLE ADDRESS
        DB 0,0,0,0,0,0,0,0,0          ; BDOS SCRATCH AREA
        DB 0                          ; MEDIA FLAG
        DW DPBSD                      ; DISK PARAMETER BLOCK
        DW ??0003                     ; CHECKSUM VECTOR
        DW ??0004                     ; ALLOCATION VECTOR
        DW 0FFFEH,0FFFEH,0FFFEH       ; DIRBCB, DTABCB, HASH  ALLOC'D BY GENCPM
        DB 0                          ; HASH BANK
        DS    16                      ; CHECKSUM VECT0R
        DS    31                      ; ALLOCATION VECTOR
   cseg  ; DPB must be resident

        dpbsd dpb 128,26,77,1024,64,2
        DW    ??0005                  ; 128 BYTE REC0RDS PER TRACK
        DB    ??0006,??000            ; BLOCK SHIFT' AND MASK
        DB    ??0008                  ; EXTENT MASK
        DW    ??0009                  ; MAXIMUM BLOCK NUMBER
        DW    ??0010                  ; MAXIMUM DIRECTORY ENTRY NUMBER
        DB    ??0011,??0012           ; ALLOC VECTOR FOR DIRECTORY
        DW    ??0013                  ; CHECKSUM SIzE
        DW    2                       ; OFFSET FOR SYSTEM TRACKS
        DB    ??0014,??0015           ; PHYSICAL SECTOR SIZE SHIFT AND MASK

    dseg ; rest is banked

    trans skew 26,6,1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
```

```
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1
        DB    ?NXTSEC+1



        ; Disk I/O routines for standardized BIOS interface

        ; Initialization entry point.

        ; called for first time initialization.


fd$init0
        lxi h,init$table
fd$init$next:
        mov a,m
        ora a
        rz
        mov b,a
        inx h
        mov c,m
        inx h
outir
        DB    0EDH,0B3H
        jmp fd$init$next

fd$initl:  ; all initialization done by drive 0
        ret

        init$table db 4,p$zpio$1A
        db 11001111b, 11000010b, 00010111b,11111111b
        db 4,p$zpio$lB
        db 11001111b, 1101110lb, 00010111b,11111111b
        db 0


fd$login
        ; This entry is called when a logical drive is about to
```

```
        ; be logged into for the purpose of density determination.

        ; It may adjust the parameters contained in the disk
        ; parameter header pointed at by <DE)

        00DB C9 ret ; we have nothing to do in
        ; simple single density only environment.


        ; disk READ and WRITE entry points.

        ; these entries are called with the following arguments:

        ; relative drive number in @rdrv (8 bits)
        ; absolute drive number in @adrv (8 bits)
        ; disk transfer address in @dma (16 bitS)
        ; disk transfer bank in @dbnk (8 bits)
        ; disk track address in @trk (16 bits)
        ; disk sector address in @sect (16 bits)
        ; pointer to XDPH in DE

        ; they transfer the appropriate data, perform retries
        ; if necessary, then return an error code in A

fd$read:
        lxi h,read$msg                  ; point at " Read "
        mvi a,88h
        mvi b,01 h                      ; 1797 read + Z80DMA direction
        jmp rw$common

fd$write
        lxi h,write$msg                 ; point at " Write "
        mvi a,0A8h
        mvi b,05h                       ; 1797 write + Z80DMA direction
        ; jmp wr$common                 ; fall through

rw$common: ; seek to correct track (if necessary),
        ; initialize DMA controller,
        ; and issue 1797 command.

        shld operation$name             ; save message for errors
        sta disk$command                ; save 1797 command
        mov a,b
        sta zdma$direction              ; save Z80DMA direction code
        lhld @dma
        shld zdma$dma                   ; get and save DMA address
        lda @rdrv
        mov 1,a
        mvi h, 0                        ; get controller-relative disk drive
        lxi d,select$table
        dad d                           ; point to select mask for drive
        mov a,m
        sta select$mask                 ; get select mask and save it
```

```
        out pSelect                  ; select drive
more$retries:
        mvi c,10                     ; allow 10 retries
retry$operation:
        push b                       ; save retry counter

        lda select$mask
        lxi h,old$select
        cmp m
        mov m,a
        jnz new$track                ; if not same drive as last, seek

        lda @trk
        lxi h,old$track
        csp m
        mov m,a

        jnz new$track                ; if not same track, then seek

        in p$fdmisc
        ani 2
        jnz same$track               ; head still loaded, we are OK

new$track: ; or drive or unloaded head means we should ...
        call check$seek              ;... read address and seek if wrong track

        lxi b,16667                  ; 100 ms / (24 t states*250 ns)
spin$loop:                           ; wait for head/seek settling
        dcx b
        mov a,b
        ora c
        jnz spin$loop

same$track
        lda @trk
        out p$fdtrack                ; give 1797 track
        lda @sect
        Out p$fdsector               ; and sector

        lxi h,dma$block              ; point to dma command block
        lxi b,dmab$length*256 + p$zdma      ; command block length and port address
outir      ; send commands to Z80 DMA
        DB    0EDH,0B3H

        in p$bankselect              ; get old value of bank select port
        ani 3Fh
        mov b,a                      ; mask off DMA bank and save
        lda @dbnk
        rrc
        rrc                          ; get DMA bank to 2 hi-order bits
        ani 0C0h
        ora b                        ; merge with other bank stuff
        out p$bankselect             ; and select the correct DMA bank
```

```

        lda disk$command                ; get 1797 command
        call exec$command               ; start it then wait for IREQ and read status
        sta disk$status                 ; save status for error messages

        pop b                           ; recover retry counter
        ora a
        rz                              ; check status and return to BDOS if no error

        ani 0001$0000b                  ; see if record not found error
        cnz check$seek                  ; if a record not found, we might need to seek

        dcr c
        jnz retry$operation

        ; suppress error message if BDOS is returning errors to application. . .

        lda @ermde
        cpi 0FFh
        jz hard$error

        ; Had permanent error, print message like:

        ; BIOS Err on d: T-nn, S-mm, <operation> <type>, Retry ?

        call ?pderr ; print message header

        lhld operation$name
        call ?pmsg                      ; last function

        ; then, messages for all indicated error bits

        lda disk$status                 ; get Status byte from last error
        lxi h,error$table               ; point at table of message addresses
    errml:
        mov e,m
        Inx h
        mov d,m
        inx h                           ; get next message address
        add a
        push psw                        ; shift left and push residual bits with status
        xchg
        cc ?pmsg
        xchg                            ; print message, saving table pointer
        pop psw
        jnz errml                       ; if any more bits left, continue

        lxi h,error$msg
        call ?pmsg                      ; print <BEL>, Retry (Y/N) ? "
        call u$conin$echo               ; get operator response
        cpi 'Y'
        jz more$retries                 ; Yes, then retry 10 more times
    hard$error:                         ; otherwise,
```

```
        mvi a,1
        ret                             ; return hard error to BDOS
cancel:     ; here to abort job
        jmp ?wboot                      ; leap directly to warmstart vector
        ; subroutine to seek if on wrong track
        ; called both to set up new track or drive
check$seek:
        push b                          ; save error counter
        call read$id                    ; try to read ID, put track in <B>
        jz id$ok                        ; if OK, we're DE
        call step$out                   ; else step towards Trk 0
        call read$id                    ; and try again
        jz id$ok                        ; if OK, we're OK
        call restore                    ; else, restore the drive
        mvi b,0                         ; and make like we are at track
id$ok:
        mov a,b
        Out p$fdtrack                   ; send current track to track port
        lda @trk
        cmp b
        pop b
        rz                              ; lf its desired track, we are done
        out p$fddata                    ; else, desired track to data port
        mvi a,00011010b                 ; seek wi 10 ms. steps
        jmp exec$command
step$out:
        mvi a,01101010b                 ; step out once at 10 ms.
        jmp exec$command
restore:
        mvi a,00001011b                 ; restore at 15 ms
        ; jmp exec$command
exec$command:     ; issue 1797 command, and wait for IREQ
                  ; return Status
        out p$fdcmnd                    ; send 1797 command
wait$IREQ:                              ; spin til IREQ
        in p$fdint
        ani 40h
        jz wait$IREQ
        in p$fdstat                     ; get 1797 Status and clear IREQ
        ret
read$id:
        lxi h,read$id$block             ; set up DMA controller
```

```
        lxi b,length$id$dmab*256 + p$zdma ; for READ ADDRESS operation
outir
        DB    0EDH,0B3H
        mvi a,11000100b             ; issue     1797 read address command
        call exec$command           ; wait for IREQ and read status
        ani 10011101b               ; mask status
        lxi h;id$buffer
        mov b,m                     ; get actual track number in <B>
        ret                         ; and return with z flag true for OK
u$conin$echo:    ; get console input, echo it, and shift to upper case
        call ?const
        ora a
        jz u$cl                     ; see if any char already struck
        call ?conin
        jmp u$conin$echo            ; yes, eat it and try again
u$cl:
        call ?conin
        push psw
        mov c,a
        call ?cono
        pop psw
        cpi 'a'
        rc
        sui 'a'-'A'                 ; make upper case
        ret
        disk$command ds   1         ; current wd1797 command
        select$mask ds    1         ; current drive select code
        old$select ds     1         ; laat drive selected
        old$track ds      1         ; last track seeked to
        diik$status ds    1         ; last error status code for messages
        select$table db   00010000b,00100000b ; for now use drives C and  D
        ; error message components
        read$msg db ', Read',0
        write$msg db ', Write',0
        operation$name dw read$msg
        ; table of pointers to error message strings
        ; first entry is for bit 7 of 1797 status byte
        error$table dw b7$msg
        dw b6$msg
        dw b5$msg
        dw b4$msg
```

```
dw b3$msg
dw b2$msg
dw b1$msg
dw b0$msg

b7$msg db   ' Not ready,',0
b6$msg db   ' protect,',0
b5$msg db   ' Fault,',0
b4$msg db   ' Record not found,',0
b3$msg db   ' CRC,',0
b2$msg db   ' Lost dsta,',0
b1$msg db   ' DREQ,',0
b0$msg db   ' Busy,',0

error$msg db      ' Retry (Y/N) ? ',0



; command string for Z80DMA device for normal operation

dma$block db      0C3h         ; reset DMA channel
db     14h                     ; channel A is incrementing memory
db     28h                     ; channel B is fixed port address
db     8Ah                     ; RDY is high, CE/ only, stop on E0B
db     79h                     ; program all of ch. A, xfer B->A (temp)
zdma$dma ds 2                  ; starting DMA address
dw     128-1                   ; 128 byte sectors in SD
db     85h                     ; xfer byte at a time, ch B is 8 bit address
db p$fddata                    ; ch B port address (1797 data port)
db     0CFh                    ; load B as source register
db     05h                     ; ;'fer A->B
db     0CFh                    ; load A as source register
zdma$direction ds 1            ; either A->B or B->A
db     0CFh                    ; load final source register
db     87h                     ; enable DMA channel
dmab$length equ   $-dma$block



read$id$block db  0C3h         ; reset DMA channel
db     14h                     ; channel A is incrementing memory
db     28h                     ; channel B is fixed port address
db     8Ah                     ; RDY is high, CE/ only, stop on EOB
db     7Dh                     ; program all of ch. A, xfer A->B (temp)
dw id$buffer                   ; starting DMA address
dw     6-1                     ; Read ID always xfers 6 bytes
db     85h                     ; byte xfer, ch B is 8 bit address
db p$fddata                    ; ch B port address (1797 data port)
db     0CFh                    ; load dest (currently source) register
db     01h                     ; xfer B->A
db     0CFh                    ; load source register
db     87h                     ; enable DMA channel
length$id$dmab equ $-read$id$block
```

```
431
432 cseg ; easier to put ID buffer in common
433
434       id$buffer ds      6           ; buffer to hold ID field
435                                     ; track
436                                     ; Side
437                                     ; Sector
438                                     ; length
439                                     ; CRC 1
440                                     ; CRC 2
441
442 end

B0MSG           0283  381   390#
B1MSG           027C  380   389#
B2MSG           0270  379   388#
B3MSG           026A  378   3871
B4MSG           0257  377   386#
B5MSG           024F  376   3851
B6MSG           0245  375   384#
B7MSG           0239  374   383#
BC              0000
BELL            0007  52#
CANCEL          01A6  289#
CHECKSEEK       01A9  226   257   296#
CR              000D  50#
DE              0002
DISKC0MMAMD     0211  203   249   354#
DISKSTATUS      0215  251   275   359#
DMABLENGTH      0011  239   4131
DMABLOCK        029A  238   398#  413
DPBSD           0000  62    66    79    83    93#
ERRM1           0186  277#  281
ERRORMSG        028A  283   392#
ERRORTABLE      0229  276   374#
EXECCOMMAND     01D5  250   310   316   323#  337
FDINITD         00BE  60    143#
FDINIT1         00CD  77    152#
FDIMITNEXT      00C1  145#  150
FDLOGIN         00DB  59    76    1621
FDREAD          00DC  58    75    188#
FDSD0           000A  14    621
FDSD1           005C  14    791
FDWRITE         00E6  57    74    193#
HARDERROR       01A3  263   286#
HL              0004
IDBUFFER        0011  339   422   4341
IDOK            01BE  299   302   3051
INITTABLE       00CE  144   1551
IX              0004
IY              0004
LEMGTMIDDMAB    000F  333   430#
LF              000A  511
```

MORERETRIES	010D	2101	285	
NEWTRACK	012D	217	221	225#
OLDSELECT	0213	215	356#	
OLDTRACK	0214	219	3571	
OPERATIOMMAME	0227	202	271	3691
PBANkSELECT	0025	243	247	
PBAUDCON1	000C			
PBAUDCON2	0030			
PBAUDCON34	0031			
PBAUDLPT1	000E			
PBAUDLPT2	0032			
PBOOT	0014			
PCENTDATA	0011			
PCENTSTAT	0010			
PCON2DATA	002C			
PCON2STAT	002D			
PCON3DATA	002E			
PCON3STAT	002F			
PCON4DATA	002A			
PCON4STAT	002B			
PCONFIGURATION	0024			
PCRTDATA	001C			
PCRTSTAT	001D			
PFDCMND	0004	325		
PFDDATA	0007	308	406	425
PFDINT	0008	327		
PFDMISC	0009	223		
PFDSECTOR	0006	236		
PFDSTAT	0004	329		
PFDTRACK	0005	235	306	
PINDEX	000?			
PLPT2DATA	0028			
PLPT2STAT	0029			
PLPTDATA	001E			
PLPTSTAT	001F			
PRTC	0033			
PSELECT	0008	209		
PWD1797	0004			
PZCTCl	000C			
PZCTC2	0030			
PZDART	001C			
PZDMA	0000	239	333	
PZPIO1	0008			
PzPIOlA	000A	155		
PZPIO1B	000B	157		
PZPIO2	0010			
PzPIO2A	0012			
PZPIO2B	0013			
PZPIO3	0024			
PZPI03A	0026			
PzPIO3B	0027			
PZSIO1	0028			
PZSIO2	002C			

```
READID         01E1  298   301   331#
READIDBLOCK    02AB  332   417#  430
READMSG        0218  189   366#  369
RESTORE        01D3  303   318#
RETRYOPERATION 010F  212#  259
RWCOMMON       C0ED  191   198#
SAMETRACK      0139  223   234#
SELECTMASK     0212  208   215   355#
SELECTTABLE    0216  207   361#
SPINLOOP       0133  229#  232
STEPOUT        01CE  300   314#
TRANS          00A4  62    63    79    80    106#
UCI            0202  344   346#
UCONINECHO     01F5  284   343#  345
WAITIREQ       0107  326*  327
WRITEMSG       021F  194   367*
ZDMADIRECTION 02A8   204   410#
ZDMADMA        029F  205   403*
?CONIN         0000  32    345   347
?CONO          0000  32    348
?CONST         0000  33    344
?PDEC          0000  30
?PDERR         0000  31    269
?PMSG          0000  29    271   280   283
?WBOOT         0000  28    290
@ADRV          0000  18
@DBNK          0000  20    245
@DMA           0000  19    205
@ERMDE         0000  24    263
@RDRV          0000  18    206
@SECT          0000  19    236
@TRK           0000  19    219   235   307
```

I.5 : Bank and Move Module for CP/M 3 Linked BIOS

The MOVE.ASM module performs memory-to-memory moves and bankselects .

```
title 'bank & move module for CP/M3 linked BIOS'

cseg

public ?move,?xmove,?bank
extrn @cbnk

maclib z80
maclib ports

?xmove:                             ; ALTOS can't perform interbank moves
        ret

?move:
        xchg                        ; we are passed source in DE and dest in HL
        ldir                        ; use Z80 block move instruction
```

```
17        DB    0EDH,0B0H
18        xchq                           ; need next addresses in same regs
19        ret
20
21 ; by exiting through bank select
22 ?bank:
23        push b                         ; save register b for temp
24        ral
24a       ral
24b       ral
24c       ani 1Bh                        ; isolate bank in proper bit position
25        mov b,a                        ; save in reg B
26        in p$bankselect                ; get old memory control byte
27        ani 0E7h
27A       ora b                          ; mask out old and merge in new
28        out p$bankselect               ; put new memory control byte
29        pop b                          ; restore register b
30        ret
31
32 ;      128 bytes at a time
33
34 end
BC             0000
DE             0002
HL             0004
IX             0004
IY             0004
PBANKSELECT    0025  26    28
PBAUDCON1      000C
PBAUDCON2      0030
PBAUDCON34     0031
PBAUDLPT1      000E
PBAUDLPT2      0032
PBOOT          0014
PCENTDATA      0011
PCENTSTAT      0010
PCON2DATA      002C
PCON2STAT      002D
PCON3DATA      002E
PCON3STAT      002F
PCON4DATA      002A
PCON4STAT      002B
PCONFIGURATION 0024
PORTDATA       001C
PCRTSTAT       001D
PFDCMND        0004
PFDDATA        0007
PFDINT         000B
PFDMISC        0009
PFDSECTOR      0006
PFDSTAT        0004
PFDTRACK       0005
PINDEX         000F
```

```
PLPT2DATA      0028
PLPT2STAT      0029
PLPTDATA       001E
PLPTSTAT       001F
PRTC           0033
PSELECT        000B
PWD1797        0004
PZCTCl         000C
PZCTC2         0030
PZDART         001C
PZDMA          0000
PZPIO1         0008
PZPIOlA        000A
PZPIO1B        000B
PZPIO2         0010
PZPIO2A        0012
PZPIO2B        0013
PZPIO3         0024
PZPI03A        0026
PZPIO3B        0027
PZSIO1         002B
PZSIO2         002C
?BANK          0006  5     22#
?MOVE          0001  5     14#
?XMOVE         0000  5     11#
@CBNK          0000  6
```

I.6 : I/o Port Addresses for Z80 Chip-based System: PORTS.LIB

This listing is the PORTS.LIB file on your distributiondiskette. It contains the port addresses for the Z80 chip-basedsystem with a Western Digital 1797 Floppy Disk Controller.

I/O Port addresses for Z80 chip set based system with wd1797 FDC

```
; chip bases
p$zdma         equ   0
p$wd1797       equ   4
p$zpio1        equ   8
p$zctcl        equ   12
P$zpio2        equ   16
p$boot         equ   20
; OUT disables boot EPROM
p$zdart        equ   28
; console 1 and printer 1
p$zpio3        equ   36
p$zsio1        equ   40
p$zsio2        equ   44
p$zctc2        equ   48
; diskette controller chip ports p$fdcmnd equ p$wd1797+0 p$fdstat equ p$wd1797+0
p$fdtrack      equ   p$wd1797+1
p$fdsector     equ   p$wd1797+2
p$fddata       equ   p$wd1797+3
; parallel I/O 1
```

```
p$select        equ   p$zpiol+0
p$fdint         equ   p$zpiol+0
p$fdmisc        equ   p$zpiol+1
p$zpiola        equ   p$zpiol+2
p$zpiolb        equ   p$zpiol+3
; counter timer chip 1
p$baudconl      equ   p$zctcl+0
p$baudlptl      equ   p$zctcl+2
p$index         equ   p$zctcl+3
; parallel I/O 2, Centronics printer interface
p$cent$stat     equ   p$zpio2+0
p$cent$data     equ   p$zpio2+1
p$zpio2a        equ   p$zpio2+2
p$zpio2b        equ   p$zpio2+3
; dual asynch rcvr/xstr, console and serial printer ports
p$crt$data      equ   p$zdart+0
p$crt$stat      equ   p$zdart+1
p$lpt$data      equ   p$zdart+2
p$lpt$stat      equ   p$zdart+3
; Third Parallel I/0 device
p$configuration       equ   p$zpio3+0
p$bankselect    equ   p$zplo3+1
p$zpio3a        equ   p$zpio3+2
p$zpio3b        equ   p$zpio3+3
; Serial I/0 device 1, printer 2 and console 4
p$lpt2data      equ   p$zsiol+0
p$lpt2stat      equ   p$zsiol+1
p$con4data      equ   p$zsiol+2
p$con4stat      equ   p$zsio1+3
; Serial I/0 device 2, console 2 and 3
p$con2data      equ   p$zsio2+0
p$con2stat      equ   p$zsio2+1
p$con3data      equ   p$zsio2+2
p$con3stat      equ   p$zsio2+3
; second Counter Timer Circuit
p$baudcon2      equ   p$zctc2+0
p$baudcon34     equ   p$zctc2+1
p$baudlpt2      equ   p$zctc2+2
p$rtc           equ   p$zctc2+3
PCRTSTAT        001D
PFDCMND         0004
pFDDATA         0007
PFDINT          0008
PFDMISC         0009
PFDSECTOR       0006
PFDSTAT         0004
PFDTRACK        0005
PINDEX          000F
PLPT2DATA       0028
PLPT2STAT       0029
PLPTDATA        001E
PLPTSTAT        001F
PRTC            0033
```

```
PSELECT        0008
PWD1797        0004
PZCTC1         000C
PZCTC2         0030
PZDART         001C
PZDMA          0000
PZPIO1         0008
PZPIO1A        000A
PZPIO1B        000B
PZPIO2         0010
PZPIO2A        0012
PZPIO2B        0013
PZPI03         0024
PZPI03A        0026
PZPIO3B        0027
PZSIO1         002B
PZSIO2         002C
?BANK          0006  5     22#
?MOVE          0001  5     14#
?XMOVE         0000  5     11#
@CBNK          0000  6
```

I.7 : Sample Submit File for ASC 8000-15 System

Digital Research used this SUBMIT file to build the sample BIOS.

```
;Submit file to build sample BIOS for ACS 8000-15 single-density system rmac bioskrnl
rmac boot rmac move rmac chario rmac drvtbl rmac fd1797sd rmac scb link
bnkbios3[b,q]=bioskrnl,boot,move,chario,drvtbl,fd17975d,scb gencpm
```

Listing 1-7. Sample Submit File for ASC 8000-15 System

Appendix J : Public Entry Points for CP/M 3 Sample BIOS Modules

Table J-1 : Public Entrypoints for CP/M 3 Sample BIOS Modules

Module Name	Public Entry Point	Function	Input Parameter	Return Value
BIOSERNL	?PMSG	Print Message	HL points to msg	none
	?PDEC	Print Decimal	HL=number	none
	?PDERR	Print BIOS Disk Err Msg Header	none	none
CHARIO	?CINIT	Char Dev Init	Dev Parms in @CTBL C=Phys Dev #	none
	?CIST	Char Inp Dev St	B=Phys Dev#	A=00 if no input A=0FFH if char avail.
	?COST	Char Out Dev St	B=Phys Dev#	A=00 if output busy A=0FFH if output ready
	?CI	Char Dev Input	B=Phys Dev#	A=next available input char
	?CO	Char Dev Output	B=Phys Dev# C=Input Char	None
MOVE	?MOVE	Memory to Memory Move	BC=byte count DE=start source adr HL=start dest adr	DE,HL point to next bytes after move
	?xMOVE	Set Banks for Extended Move	B=Dest Bank C=Source Bank	BC,DE,HL are unchanged
	?BANK	Select Bank	A=Bank Number	All unchanged
BOOT	?INIT	System Init	none	none
	?LDCCP	Load CCP	none	none
	?RLCCP	Reload CCP	none	none
	?TIME	Get/Set Time	C=000H if get C=0FFH if set	none

Listing J-1. Public Entry Points for cP/M 3 Sample BIOS Modules

Appendix K : Public Data Items in CP/M 3 Sample BIOS Modules

Table K-1. Public Data Items

Module Name	Public Data	Description
BIOSKRNL	@ADRV	Absolute Logical Drive Code
	@RDRV	Relative logical drive code (UNIT)
	@TRK	Track Number
	@SECT	Sector Address
	@DMA	DMA Address
	@DBNK	Bank for Disk I/O
	@CNT	Multi-sector Count
	@CBNK	Current CPU Bank
CHARIO	@CTBL	Character Device Table
DRVTBL	@DTBL	Drive Table

Appendix L : CP/M 3 BIOS Function Summary

Table L-1. BIOS Function Jump Table Summary

No.	Function	Input	Output
0	BOOT	None	None
1	WBOOT	None	None
2	CONST	None	A=0FFH if ready A=00H if not ready
3	CONIN	None	A=Con Char
4	CONOUT	C=Con Char	None
5	LIST	C=Char	None
6	AUXOUT	C=Char	None
7	AUXIN	None	A=Char
8	HOME	None	None
9	SELDSK	C=Drive 0-15 E=Init Sel Flag	HL=DPH addr HL=000H if invalid dr.
10	SETTRK	BC=Track No	None
11	SETSEC	BC=Sector No	None
12	SETDMA	BC=.DMA	None
13	READ	None	A=00H if no Err A=01H if Non-recov Err A=0FFH if media changed
14	WRITE	C=Deblk Codes	A=00H if no Err A=01H if Phys Err A=02H if Dsk is R/O A=0FFH if media changed
15	LISTST	None	A=00H if not ready A=0FFH if ready
16	SECTRN	BC=Log Sect No DE=Trans Tbl Adr	HL=Phys Sect No
17	CONOST	None	A=00H if not ready A=0FFH if ready
18	AUXIST	None	A=00H if not ready A=0FFH if ready
19	AUXOST	None	A=00H if not ready A=0FFH if ready
20	DEVTBL	None	HL=Chrtbl addr
21	DEVINI	C=Dev No 0-15	None
22	DRVTBL	None	HL=Drv Tbl addr HL=0FFFFH HL=0FFFEH
23	MULTIO	C=Mult Sec Cnt	None
24	FLUSH	None	A=000H if no err A=00IH if phys err A=002H if disk R/O
25	MOVE	HL=Dest Adr DE=Source Adr bytes following MOVE BC=Count	HL & DE point to next
26	TIME	C=Get/Set Flag	None
27	SELMEM	A=Mem Bank	None
28	SETBNK	A=Mem Bank	None

No.	Function	Input	Output
29	XMOVE	B=Dest Bank C=Source Bank	None
30	USERF	Reserved for System Implementor	
31	RESERV1	Reserved for Future Use	
32	RESERV2	Reserved for Future Use	

Revised, reformatted, and restored from original Ami Pro and PS sources by PCPete (PCPete@audiography.com.au) 10MAR2011 (Rev 1.00)