CP/M-68K TM
Operating System
System Guide

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

The CP/M-68K 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

Foreword

CP/M-68K" is a single-user general purpose operating system. it is designed for use with any disk-based computer using a Motorola, MC68000 or compatible processor. CP/M-68K is modular in design, and can be modified to suit the needs of a particular installation.

The hardware interface for a particular hardware environment is supported by the OEM or CP/M-68K distributor. Digital Research supports the user interface to CP/M-68K as documented in the CP/M-68K Operating System User's Guide. Digital Research does not support any additions or modifications made to CP/M-68K by the OEM or distributor.

Purpose and Audience

This manual is intended to provide the information needed by a systems programmer in adapting CP/M-68K to a particular hardware environment. A substantial degree of programming expertise is assumed on the part of the reader, and it is not expected that typical users of CP/M-68K will need or want to read this manual.

Prerequisites and Related Publications

In addition to this manual, the reader should be familiar with the architecture of the Motorola MC68000 as described in the Motorola 16-Bit Microprocessor User's Manual (third edition), the CP/M-68K User's and Programmer's Guides, and, of course, the details of the hardware environment where CP/M-68K is to be implemented.

How This Book is Organized

Section 1 presents an overview of CP/M-68K and describes its major components. Section 2 discusses the adaptation of CP/M-68K for your specific hardware system. Section 3 discusses bootstrap procedures and related information. Section 4 describes each BIOS function including entry parameters and return values. Section 5 describes the process of creating a BIOS for a custom hardware interface. Section 6 discusses how to get CP/M working for the first time on a new hardware environment. Section 7 describes a procedure for causing a command to be automatically executed on cold boot. Section 8 describes the PUTBOOT utility, which is useful in generating a bootable disk.

Appendix A describes the contents of the CP/M-68K distribution disks. Appendixes B, C, and D are listings of various BIOSes. Appendix E contains a listing of the PUTBOOT utility program. Appendix F describes the Motorola S-record representation for programs.

Table of Contents

Table of Contents
(continued)

Appendixes

Tables and Figures

Tables

Figures

Section 1
System Overview

1.1 Introduction

CP/M-68K is a single-user, general purpose operating system for microcomputers based on the Motorola MC68000 or equivalent microprocessor chip. It is designed to be adaptable to almost any hardware environment, and can be readily customized for particular hardware systems.

CP/M-68K is equivalent to other CP/M systems with changes dictated by the 68000 architecture. In particular, CP/M-68K supports the very large address space of the 68000 family. The CP/M-68K file system is upwardly compatible with CP/M-80 version 2.2 and CP/M-86 Version 1.1. The CP/M-68K file structure allows files of up to 32 megabytes per file. CP/M-68K supports from one to sixteen disk drives with as many as 512 megabytes per drive.

The entire CP/M-68K operating system resides in memory at all times, and is not reloaded at a warm start. CP/M-68K can be configured to reside in any portion of memory above the 68000 exception vector area (OH to 3FFH). The remainder of the address space is available for applications programs, and is called the transient program area, TPA.

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

Table 1-1. CP/M-68K Terms

Term	Meaning
nibble	4-bit half-byte
byte	8-bit value
word	16-bit value
longword	32-bit value
address	32-bit identifier of a storage location
offset	a value defining an address in storage; a fixed displacement from some other address

Table 1-1. (continued)

Term	Meaning
text segment	program section containing machine instructions
data segment	program section containing initialized data
block storage segment (bss)	program section containing uninitialized data
absolute	describes a program which must reside at a fixed memory address.
relocatable	describes a program which includes relocation information so it can be loaded into memory at any address

The CP/M-68K programming model is described in detail in the CP/M-68K Operating System Programmer's Guide. To summarize that model briefly, CP/M-68K supports four segments within a program: text, data, block storage segment (bss) , and stack. When a program is loaded, CP/M-68K allocates space for all four segments in the TPA, and loads the text and data segments. A transient program may manage free memory using values stored by CP/M-68K in its base page.

```
            USER
              ^
              |
              V
        User Interface
            (CCP)
              ^
              |
              V
    Programming Interface
           (BDOS)
              ^
              |
              V
     Hardware Interface
           (BIOS)
              ^
              |
              V
    HARDWARE ENVIRONMENT
```

Figure 1-1. CP/M-68K Interfaces

1.2 CP/M-68K Organization

CP/M-68K comprises three system modules: the Console Command Processor (CCP) the Basic Disk Operating System (BDOS) and the Basic Input/Output System (BIOS). These modules are linked together to form the operating system. They are discussed individually in this section.

1.3 Memory Layout

The CP/M-68K operating system can reside anywhere in memory except in the interrupt vector area (0H to 3FFH) . The location of CP/M-68K is defined during system generation. Usually, the CP/M-68K operating system is placed at the top end (high address) of available memory, and the TPA runs from 400H to the base of the operating system. It is possible, however, to have other organizations for memory. For example, CP/M-68K could go in the low part of memory with the TPA above it. CP/M-68K could even be placed in the middle of available memory.

However, because the TPA must be one contiguous piece, part of memory would be unavailable for transient programs in this case. Usually this is wasteful, but such an organization might be useful if an area of memory is to be used for a bit-mapped graphics device, for example, or if there are ROM-resident routines. The BIOS and specialized application programs might know this memory exists, but it is not part of the TPA.

```
                          Top of Memory -------|
                                               |
CCP & BDOS & BIOS         CP/M-68K -           |
                                               |
User Stack                ---------------------|
                                               |
Free Memory                                    |
                                               |
bss                       TPA                  |--------|
                                               |        |
Data                                           |        |
                                               |    user pgm
Text                                           |        |
                          00500H               |        |
Base Page                                      |        |
                          00400H               |        |
Interrupt Vectors         ---------------------|--------|
                          00000H
```

Figure 1-2. Typical CP/M-68K Memory Layout

1.4 Console Command Processor (CCP)

The Console Command Processor, (CCP) provides the user interface to CP/M-68K. It uses the BDOS to read user commands and load programs, and provides several built-in user commands. It also provides parsing of command lines entered at the console.

1.5 Basic Disk Operating System (BDOS)

The Basic Disk Operating System (BDOS) provides operating system services to applications programs and to the CCP. These include character I/O, disk file I/O (the BDOS disk I/O operations comprise the CP/M-68K file system), program loading, and others.

1.6 Basic I/O System (BIOS)

The Basic Input Output System (BIOS) is the interface between CP/M-68K and its hardware environment. All physical input and output is done by the BIOS. It includes all physical device drivers, tables defining disk characteristics, and other hardware specific functions and tables. The CCP and BOOS do not change for different hardware environments because all hardware dependencies have been concentrated in the BIOS. Each hardware configuration needs its own BIOS. Section 4 describes the BIOS functions in detail. Section 5 discusses how to write a custom BIOS. Sample BIOSes are presented in the appendixes.

1.7 I/O Devices

CP/M-68K recognizes two basic types of I/O devices: character devices and disk drives. Character devices are serial devices that handle one character at a time. Disk devices handle data in units of 128 bytes, called sectors, and provide a large number of sectors which can be accessed in random, nonsequential, order. In fact, real systems might have devices with characteristics different from these. It is the BIOS's responsibility to resolve differences between the logical device models and the actual physical devices.

1.7.1 Character Devices

Character devices are input output devices which accept or supply streams of ASCII characters to the computer. Typical character devices are consoles, printers, and modems. In CP/M-68K operations on character devices are done one character at a time. A character input device sends ASCII CTRL-Z (lAH) to indicate end-of- file.

1.7.2 Character Devices

Disk devices are used for file storage. They are organized into sectors and tracks. Each sector contains 128 bytes of data. (If sector sizes other than 128 bytes are used on the actual disk, then the BIOS must do a logical-to-physical mapping to simulate 128- byte sectors to the rest of the system.) All disk I/O in CP/M-68K is done in one-sector units. A track is a group of sectors. The number of sectors on a track is a constant depending on the particular device. (The characteristics of a disk device are specified in the Disk Parameter Block for that device. See Section 5.) To locate a particular sector, the disk, track number, and sector number must all be specified.

1.8 System Generation and Cold Start Operation

Generating a CP/M-68K system is done by linking together the CCP, BDOS, and BIOS to create a file called CPM.SYS, which is the operating system. Section 2 discusses how to create CPM.SYS. CPM.SYS is brought into memory by a bootstrap loader which will typically reside on the first two tracks of a system disk. (The term system disk as used here simply means a disk with the file CPM.SYS and a bootstrap loader.) Creation of a bootstrap loader is discussed in Section 3.

End of Section 1

Section 2
System Generation

2.1 Overview

This section describes how to build a custom version of CP/M- 68K by combining your BIOS with the CCP and BDOS supplied by Digital Research to obtain a CP/M-68K operating system suitable for your specific hardware system. Section 5 describes how to create a BIOS.

In this section, we assume that you have access to an already configured and executable CP/M-68K system. If you do not, you should first read Section 6, which discusses how you can make your first CP/M-68K system work.

A CP/M-68K operating system is generated by using the linker, L068, to link together the system modules (CCP, BDOS, and BIOS) . Then the RELOC utility is used to bind the system to an absolute memory location. The resulting file is the configured operating system. It is named CPM.SYS.

2.2 Creating CPM.SYS

The CCP and BDOS for CP/M-68K are distributed in a library file named CPMLIB. You must link your BIOS with CPMLIB using the following command:

A>LO68 -R -UCPM -0 CPM.REL CPMLIB BIOS.0

where BIOS.0 is the compiled or assembled BIOS. This creates CPM.REL, which is a relocatable version of your system. The cold boot loader, however, can load only an absolute version of the system, so you must now create CPM.SYS, an absolute version of your system. If you want your system to reside at the top of memory, first find the size of the system with the following command:

A>SIZE68 CPM.REL

This gives you the total size of the system in both decimal and hex byte counts. Subtract this number from the highest memory address in your system and add one to get the highest possible address at which CPM.REL can be relocated. Assuming that the result is aaaaaa, type this command:

A>RELOC -Baaaaaa CPH.REL CPK.SYS

The result is the CPM.SYS file, relocated to load at memory address aaaaaa. If you want CPM.SYS to reside at some other memory address, such as immediately above the exception vector area, you can use RELOC to place the system at that address.

When you perform the relocation, verify that the resulting system does not overlap the TPA as defined in the BIOS. The boundaries of the system are determined by taking the relocation address of CPM.SYS as the base, and adding the size of the system (use SIZE68 on CPM.SYS) to get the upper bound. This address range must not overlap the TPA that the BIOS defines in the Memory Region Table.

2.3 Relocating Utilities

Once you have built CPM.SYS, it is advisable to relocate the operating system utilities for your TPA using the RELOC utility. RELOC is described in the CP/M-68K Operating System Programmer's Guide. This results in the utilities being absolute, rather than relocatable, but they will occupy half the disk space and load into memory twice as fast in their new form. You should also keep the relocatable versions backed up in case you ever need to use them in a different TPA.

End of Section 2

Section 3
Bootstrap Procedures

3.1 Bootstrapping Overview

Bootstrap loading is the process of bringing the CP/M-68K operating system into memory and passing control to it. Bootstrap loading is necessarily hardware dependent, and it is not possible to discuss all possible variations in this manual. However, the manual presents a model of bootstrapping that is applicable to most systems.

The model of bootstrapping which we present assumes that the CP/M-68K operating system is to be loaded into memory from a disk in which the first few tracks (typically the first two) are reserved for the operating system and bootstrap routines, while the remainder of the disk contains the file structure, consisting of a directory and disk files. (The topic of disk organization and parameters is discussed in Section 5.) In our model, the CP/M-68K operating system resides in a disk file named CPM.SYS (described in Section 2), and the system tracks contain a bootstrap loader program (CPMLDR.SYS) which knows how to read CPM.SYS into memory and transfer control to it.

Most systems have a boot procedure similar to the following:

1) When you press reset, or execute a boot command from a monitor ROM, the hardware loads one or more sectors beginning at track 0, sector 1, into memory at a predetermined address, and then jumps to that address.

2) The code that came from track 0, sector 1, and is now executing, is typically a small bootstrap routine that loads the rest of the sectors on the system tracks (containing CPMLDR) into another predetermined address in memory, and then jumps to that address. Note that if your hardware is smart enough, steps 1 and 2 can be combined into one step.

3) The code loaded in step 2, which is now executing, is the CP/M Cold Boot Loader, CPMLDR, which is an abbreviated version of CP/M-68K itself. CPMLDR now finds the file CPM.SYS, loads it, and jumps to it. A copy of CPM.SYS is now in memory, executing. This completes the bootstrapping process.

In order to create a CP/M-68K diskette that can be booted, you need to know how to create CPM.SYS (see Section 2.2), how to create the Cold Boot Loader, CPMLDR, and how to put CPMLDR onto your system tracks. You must also understand your hardware enough to be able to design a method for bringing CPMLDR into memory and executing it.

3.2 Creating the Cold Boot Loader

CPMLDR is a miniature version of CP/M-68K. It contains stripped versions of the BOOS and BIOS, with only those functions which are needed to open the CPM.SYS file and read it into memory. CPMLDR will exist in at least two forms; one form is the information in the system tracks, the other is a file named CPMLDR.SYS which is created by the linker. The term CPMLDR is used to refer to either of these forms, but CPMLDR.SYS only refers to the file.

CPMLDR.SYS is generated using a procedure similar to that used in generating CPM.SYS. That is, a loader BIOS is linked with a loader system library, named LDRLIB, to produce CPMLDR.SYS. Additional modules may be linked in as required by your hardware. The resulting file is then loaded onto the system tracks using a utility program named PUTBOOT.

3.2.1 Writing a Loader BIOS

The loader BIOS is very similar to your ordinary BIOS; it just has fewer functions, and the entry convention is slightly different. The differences are itemized below.

1) Only one disk needs to be supported. The loader system selects only drive A. If you want to boot from a drive other than A, your loader BIOS should be written to select that other drive when it receives a request to select drive A.

2) The loader BIOS is not called through a trap; the loader BDOS calls an entry point named _bios instead. The parameters are still passed in registers, just as in the normal BIOS. Thus, your Function 0 does not need to initialize a trap, the code that in a normal BIOS would be the Trap 3 handler should have the label _bios, and you exit from your loader BIOS with an RTS instruction instead of an RTE.

3) Only the following BIOS functions need to be implemented:

0 (Init) Called just once, should initialize hardware as necessary, no return value necessary. Note that Function 0 is called via _bios with the function number equal to 0. You do not need a separate _init entry point.

4 (Conout) Used to print error messages during boot. If you do not want error messages, this function should just be an rts.

9 (Seldsk) Called just once, to select drive A.

10 (Settrk)

11 (Setsec)

12 (Setdma)

13 (Read)

16 (Sectran)

18 (Get MRT) Not used now, but may be used in future releases.

22 (Set exception)

4) You do not need to include an allocation vector or a check vector, and the Disk Parameter Header values that point to these can be anything. However, you still need a Disk Parameter Header, Disk Parameter Block, and directory buffer.

It is possible to use the same source code for both your normal BIOS and your loader BIOS if you use conditional compilation or assembly to distinguish the two. We have done this in our example BIOS for the EXORmacs"

3.2.2 Building CPMLDR.SYS

Once you have written and compiled (or assembled) a loader BIOS, you can build CPMLDR.SYS in a manner very similar to building CPM.SYS. There is one additional complication here: the result of this step is placed on the system tracks. So, if you need a small prebooter to bring in the bulk of CPMLDR, the prebooter must also be included in the link you are about to do. The details of what must be done are hardware dependent, but the following example should help to clarify the concepts involved.

Suppose that your hardware reads track 0, sector 1, into memory at location 400H when reset is pressed, then jump to 400H. Then your boot disk must have a small program in that sector that can load the rest of the system tracks into memory and execute the code that they contain. Suppose that you have written such a program, assembled it, and the assembler output is in BOOT.O. Also assume that your loader BIOS object code is in the file LDRBIOS.O. Then the following command links together the code that must go on the system tracks.

```
A>lo68 -s -T400 -uldr -o cpmldr.sys boot.o ldrlib ldrbios.o
```

Once you have created CPMLDR.SYS in this way, you can use the PUTBOOT utility to place it on the system tracks. PUTBOOT is described in Section 8. The command to place CPMLDR on the system tracks of drive A is:

```
A>putboot cpmldr.sys a
```

PUTBOOT leads the file CPMLDR.SYS, strips off the 28-byte command file header, and puts the result on the specified drive. You can now boot from this disk, assuming that CPM.SYS is on the disk.

End of Section 3

Section 4
BIOS Functions

4..l Introduction

All CP/M-68K hardware dependencies are concentrated in subroutines that are collectively referred to as the Basic I/O System (BIOS) . A CP/M-68K system implementor can tailor CP/M-68K to fit nearly any 68000 operating environment. This section describes each BIOS function: its calling conventions, parameters, and the actions it must perform. The discussion of Disk Definition Tables is treated separately in Section 5.

When the BDOS calls a BIOS function, it places the function number in register DO.W, and function parameters in registers Dl and D2. It then executes a TRAP 3 instruction. DO.W is always needed to specify the function, but each function has its own requirements for other parameters, which are described in the section describing the particular function. The BIOS returns results, if any, in register D0. The size of the result depends on the particular function.

Note: the BIOS does not need to preserve the contents of registers. That is, any register contents which were valid on entry to the BIOS may be destroyed by the BIOS on exit. The BDOS does not depend on the BIOS to preserve the contents of data or address registers. Of course, if the BIOS uses interrupts to service I/O, the interrupt handlers will need to preserve registers.

Usually, user applications do not need to make direct use of BIOS functions. However, when access to the BIOS is required by user software, it should use the BDOS Direct BIOS Function, Call 50, instead of calling the BIOS with a TRAP 3 instruction. This rule ensures that applications remain compatible with future systems.

The Disk Parameter Header (DPH) and Disk Parameter Block (DPB) formats have changed slightly from previous CP/M versions to accommodate the 68000's 32-bit addresses. The formats are described in Section 5.

Table 4-1. BIOS Register Usage

Entry Parameters:

D0.W = function code
D1.x = first parameter
D2.x = second parameter

Return Values:

D0.B = byte values (8 bits)
D0.W = word values (16 bits)
D0.L = longword values (32 bits)

The decimal BIOS function numbers and the functions they correspond to are listed in Table 4-2.

Table 4-2. BIOS Functions

Number	Function
0	Initialization (called for cold boot)
1	Warm Boot (called for warm start)
2	Console Status (check for console character ready)
3	Read Console Character In
4	Write Console Character Out
5	List (write listing character out)
6	Auxiliary Output (write character to auxiliary output device)
7	Auxiliary Input (read from auxiliary input)
8	Home (move to track 00)
9	Select Disk Drive
10	Set Track Number
11	Set Sector Number
12	Set DMA Address
13	Read Selected Sector
14	Write Selected Sector
15	Return List Status
16	Sector Translate
18	Get Memory Region Table Address
19	Get I/O Mapping Byte
20	Set I/O Mapping Byte
21	Flush Buffers
22	Set Exception Handler Address

FUNCTION 0: INITIALIZATION

Entry Parameters:
Register D0.W: 00H

Returned Value:
Register D0.W: User/Disk Numbers

This routine is entered on cold boot and must initialize the BIOS. Function 0 is unique, in that it is not entered with a TRAP 3 instruction. Instead, the BIOS has a global label, -init, which is the entry to this routine. On cold boot, Function 0 is called by a jsr init. When initialization is done, exit is through an rts instruction. Function 0 is responsible for initializing hardware if necessary, initializing BIOS internal variables (such as IOBYTE) as needed, setting up register D0 as described below, setting the Trap 3 vector to point to the main BIOS entry point, and then exiting with an rts.

Function 0 returns a longword value. The CCP uses this value to set the initial user number and the initial default disk drive. The least significant byte of DO is the disk number (0 for drive A, 1 for drive B, and so on). The next most significant byte is the user number. The high-order bytes should be zero.

The entry point to this function must be named init and must be declared global. This function is called only once from the system at system initialization.

Following is an example of skeletal code:

```
.globl  _init                          ;bios init entry point

_init:
*       do  any initialization here
        move.1      # traphndl,$8c     ;set trap 3 handler
        clr.1       d0                 ;login drive A, user 0
        rts
```

FUNCTION 1: WARM BOOT

Entry Parameters:
  Register D0.W: 01H

Returned Value: None

This function is called whenever a program terminates. Some reinitialization of the hardware or software might occur. When this function completes, it jumps directly to the entry point of the CCP, named _ccp. Note that -ccp must be declared as a global.

Following is an example of skeletal code for this BIOS function:

```
.globl      -ccp

wboot:
*   do any reinitialization here if necessary
        jmp     -ccp
```

FUNCTION 2: CONSOLE STATUS

Entry Parameters:		
	Register D0.W:	02H
Returned Value:		
	Register D0.W:	00FFH if ready
	Register D0.W:	0000H if not ready

This function returns the status of the currently assigned console device. It returns 00FFH in register D0 when a character is ready to be read, or 0000H in register D0 when no console characters are ready.

FUNCTION 3: READ CONSOLE CHARACTER

Entry Parameters:	
Register D0.W:	03H
Returned Value:	
Register D0.W:	Character

This function reads the next console character into register D0.W. If no console character is ready, it waits until a character is typed before returning.

FUNCTION 4: WRITE CONSOLE CHARACTER

Entry Parameters:	
Register D0.W:	04H
Register Dl.W:	Character
Returned Value:	None

This function sends the character from register Dl to the console output device. The character is in ASCII. You might want to include a delay or filler characters for a line-feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 Terminal). You can also filter out control characters which have undesirable effects on the console device.

FUNCTION 5: LIST CHARACTER OUTPUT

Entry Parameters:

Register D0.W:	05H
Register Dl.W:	Character
Returned Value:	None

This function sends an ASCII character from register Dl to the currently assigned listing device. If your list device requires some communication protocol, it must be handled here.

FUNCTION 6: AUXILIARY OUTPUT

Entry Parameters:
Register D0.W: 06H
Register DI.W: Character

Returned Value:
Register D0.W: Character

This function sends an ASCII character from register Dl to the currently assigned auxiliary output device.

FUNCTION 7: AUXILIARY INPUT

Entry Parameters,
Register D0.W: 07H

Returned Value:
Register D0.W: Character

This function reads the next character from the currently assigned auxiliary input device into register D0. It reports an end-of-file condition by returning an ASCII CTRL-Z (lAH).

FUNCTION 8: HOME

Entry Parameters:	
Register D0.W:	08H
Returned Value:	None

This function returns the disk head of the currently selected disk to the track 00 position. If your controller does not have a special feature for finding track 00, you can translate the call to a SETTRK function with a parameter of 0.

FUNCTION 9: SELECT DISK DRIVE

Entry Parameters:	
Register D0.W:	09H
Register DI.B:	Disk Drive
Register D2.B:	Logged in Flag
Returned Value:	
Register D0.L:	Address of Selected Drivels DPH

This function selects the disk drive specified in register D1 for further operations. Register D1 contains 0 for drive A, 1 for drive B, up to 15 for drive P.

On each disk select, this function returns the address of the selected drive's Disk Parameter Header in register D0.L. See Section 5 for a discussion of the Disk Parameter Header.

If there is an attempt to select a nonexistent drive, this function returns 00000000H in register D0.L as an error indicator. Although the function must return the header address on each call, it may be advisable to postpone the actual physical disk select operation until an I/0 function (seek, read, or write) is performed. Disk select operations can occur without a subsequent disk operation. Thus, doing a physical select each time this function is called may be wasteful of time.

On entry to the Select Disk Drive function, if the least significant bit in register D2 is zero, the disk is not currently logged in. If the disk drive is capable of handling varying media (such as single and double-sided disks, single- and double-density, and so on), the BIOS should check the type of media currently installed and set up the Disk Parameter Block accordingly at this time.

FUNCTION 10: SET TRACK NUMBER

Entry Parameters:	
Register D0.W:	0AH
Register DI.W:	Disk track number
Returned Value:	None

This function specifies in register D0.W the disk track number for use in subsequent disk accesses. The track number remains valid until either another Function 10 or a Function 8 (Home) is performed.

You can choose to physically seek to the selected track at this time, or delay the physical seek until the next read or write actually occurs.

The track number can range from 0 to the maximum track number supported by the physical drive. However, the maximum track number is limited to 65535 by the fact that it is being passed as a 16-bit quantity. Standard floppy disks have tracks numbered from 0 to 76.

FUNCTION 11: SET SECTOR NUMBER

Entry Parameters:		
	Register D0.W:	0BH
	Register Dl.W: S	sector Number
Returned Value:		None

This function specifies in register Dl.W the sector number for subsequent disk accesses. This number remains in effect until either another Function 11 is performed.

The function selects actual (unskewed) sector numbers. if skewing is appropriate, it will have previously been done by a call to Function 16. You can send this information to the controller at this point or delay sector selection until a read or write operation occurs.

FUNCTION 12: SET DMA ADDRESS

Entry Parameters:

Register D0.W:	0CH
Register Dl.L:	DMA Address
Returned Value:	None

This function contains the DMA (disk memory access) address in register DI for subsequent read or write operations. Note that the controller need not actually support DMA (direct memory access). The BIOS will use the 128-byte area starting at the selected DMA address for the memory buffer during the following read or write operations. This function can be called with either an even or an odd address for a DMA buffer.

FUNCTION 13: READ SECTOR

Entry Parameters:
Register D0.W: 0DH

Returned Value:
Register D0.W: 0 if no error
Register D0.W: 1 if physical error

After the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the read function uses these parameters to read one sector and returns the error code in register D0.

Currently, CP/M-68K responds only to a zero or nonzero return code value. Thus, if the value in register D0 is zero, CP/M-68K assumes that the disk operation completed properly. If an error occurs however, the BIOS should attempt at least ten retries to see if the error is recoverable.

FUNCTION 14: WRITE SECTOR

Entry Parameters:

Register D0.W:		0EH
Register Dl.W:		0=normal write
		1=write to a directory sector
		2=write to first sector of new block

Returned Value:

Register D0.W:		0=no error
		1=physical error

This function is used to write 128 bytes of data from the currently selected DMA buffer to the currently selected sector, track, and disk. The value in register DI.W indicates whether the write is an ordinary write operation or whether the there are special considerations.

If register D1.W=0, this is an ordinary write operation. if D1.W=1, this is a write to a directory sector, and the write should be physically completed immediately. If DI.W=2, this is a write to the first sector of a newly allocated block of the disk. The significance of this value is discussed in Section 5 under Disk Buffering.

FUNCTION 15: RETURN LIST STATUS

Entry Parameters:
  Register D0.W: 0FH

Returned Value:
  Register D0: 00FFH=device ready
  Register D0: 0000H=device not ready

This function returns the status of the list device. Register D0 contains either 0000H when the list device is not ready to accept a character or 00FFH when a character can be sent to the list device.

FUNCTION 16: SECTOR TRANSLATE

Entry Parameters:

Register D0.W:	10H
Register D1.W:	Logical Sector Number
Register D2.L:	Address of Translate Table

Returned Value:

Register D0.W:	Physical Sector Number

This function performs logical-to-physical sector translation, as discussed in Section 5.2.2. The Sector Translate function receives a logical sector number from register Dl.W. The logical sector number can range from 0 to the number of sectors per track-1. Sector Translate also receives the address of the translate table in register D2.L. The logical sector number is used as an index into the translate table. The resulting physical sector number is returned in D0.W.

If register D2.L = 00000000H, implying that there is no translate table, register D1 is copied to register D0 before returning. Note that other algorithms are possible; in particular, is is common to increment the logical sector number in order to convert the logical range of 0 to n-1 into the physical range of 1 to n. Sector Translate is always called by the BIOS, whether the translate table address in the Disk Parameter Header is zero or nonzero.

FUNCTION 18: GET ADDRESS OF MEMORY REGION TABLE

Entry Parameters:
Register D0.W: 12H

Returned Value:
Register D0.L: Memory Region Table Address

This function returns the address of the Memory Region Table (MRT) in register D0. For compatibility with other CP/M systems, CP/M-68K maintains a Memory Region Table. However, it contains only one region, the Transient Program Area (TPA) . The f or mat of the MRT is shown below:

Count =l	(16 bits)
Base address of first region	(32 bits)
Length of first region	(32 bits)

Figure 4-1. Memory Region Table Format

The memory region table must begin on an even address, and must be implemented.

FUNCTION 19: GET I/O BYTE

Entry Parameters:
Register D0.W: 13H

Returned Value:
Register D0.W: I/O Byte Current Value

This function returns the current value of the Logical, to physical input/output device byte (I/O byte) in register DO.W. This 8-bit value associates physical devices with CP/M-68K's four logical devices as noted below. Note that even though this is a byte value, we are using word references. The upper byte should be zero.

Peripheral devices other than disks are seen by CP/M-68K as logical devices, and are assigned to physical devices within the BDOS. Device characteristics are defined in Table 4-3 below.

Table 4-3. CP/M-68K Logical Device Characteristics

Device Name	Characteristics
CONSOLE	The interactive console that you use to communicate with the system is accessed through functions 2, 3 and 4. Typically, the console is a CRT or other terminal device.
LIST	The listing device is a hard-copy device, usually a printer.
AUXILIARY OUTPUT	An optional serial output device.
AUXILIARY INPUT	An optional serial input device.

Note that a single peripheral can be assigned as the LIST, AUXILIARY INPUT, and AUXILIARY OUTPUT device simultaneously. if no peripheral device is assigned as the LIST, AUXILIARY INPUT, or AUXILIARY OUTPUT device, your BIOS should give an appropriate error message so that the system does not hang if the device is accessed by PIP or some other transient program. Alternatively, the AUXILIARY OUTPUT and LIST functions can simply do nothing except return to the caller, and the AUXILIARY INPUT function can return with a 1AH (CTRL-Z) in register D0.W to indicate immediate end-of- file.

The I/O byte is split into four 2-bit fields called CONSOLE, AUXILIARY INPUT, AUXILIARY OUTPUT, and LIST, as shown in Figure 4-2.

	most significant		least significant	
I/O Byte	LIST	AUXILIARY OUTPUT	AUXILIARY INPUT	CONSOLE
bits	7, 6	5,4	3,2	1,0

Figure 4-3. I/O Byte

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given in Table 4-4.

Table 4-4. I/O Byte Field Definitions

CONSOLE field (bits 1,0)

Bit	Definition
0	console is assigned to the console printer (TTY:)
1	console is assigned to the CRT device (CRT;)
2	batch mode: use the AUXILIARY INPUT as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
3	user defined console device (UC1:)

AUXILIARY INPUT field (bits 3,2)

Bit	Definition
0	AUXILIARY INPUT is the Teletype device (TTY:)
1	AUXILIARY INPUT is the high-speed reader device (PTR:)
2	user defined reader #1 (URI:)
3	user defined reader #2 (UR2:)

Table 4-4. (continued)

AUXILIARY OUTPUT field (bits 5,4)

Bit	Definition
0	AUXILIARY OUTPUT is the Teletype device (TTY:)
1	AUXILIARY OUTPUT is the high-speed punch device (PTP:)
2	user defined punch #1 (UP1:)
3	user defined punch #2 (UP2:)

LIST field (bits 7,6)

Bit	Definition
0	LIST is the Teletype device (TTY:)
1	LIST is the CRT device (CRT:)
2	LIST is the line printer device (LPT:)
3	user defined list device (UL1:)

Note that the implementation of the I/O byte is optional, and affects only the organization of your BIOS. No CP/M-68K utilities use the I/O byte except for PIP, which allows access to the physical devices, and STAT, which allows logical-physical assignments to be made and displayed. It is a good idea to first implement and test your BIOS without the IOBYTE functions, then add the I/O byte function.

FUNCTION 20: SET I/O BYTE

Entry Parameters:	
Register D0.W:	14H
Register D1.W:	Desired
Returned Value:	None

This function uses the value in register D1 to set the value of the I/O byte that is stored in the BIOS. See Table 4-4 for the I/O byte field definitions Note that even though this is a byte value, we are using word references. The upper byte should be zero.

FUNCTION 21 FLUSH BUFFERS

Entry Parameters:

Register D0.W:	15H

Returned Value:

Register D0.W:	0000H=successful write
Register D0.W:	FFFFH=unsuccessful write

This function forces the contents of any disk buffers that have been modified to be written. That is, after this function has been performed, all disk writes have been physically completed. After the buffers are written, this function returns a zero in register D0.W. However, if the buffers cannot be written or an error occurs, the function returns a value of FFFFH in register D0.W.

FUNCTION 22: SET EXCEPTION HANDLE ADDRESS

Entry Parameters:

Register D0.W:	16H
Register Dl.W:	Exception Vector Number
Register D2.L:	Exception Vector Address

Returned Value:

Register D0.L:	Previous Vector Contents

This function sets the exception vector indicated in register Dl-W to the value specified in register D2.L. The previous vector value is returned in register DO.L. Unlike the BDOS Set Exception Vector Function (61), this BIOS function sets any exception vector. Note that register Dl.W contains the exception vector number. Thus, to set exception #2, bus error, this register contains a 2, and the vector value goes to memory locations 08H to OBH.

End of Section 4

Section 5
Creating a BIOS

5.1 Overview

The BIOS provides a standard interface to the physical input/output devices in your system. The BIOS interface is defined by the functions described in Section 4. Those functions, taken together, constitute a model of the hardware environment. Each BIOS is responsible for mapping that model onto the real hardware.

In addition, the BIOS contains disk definition tables which define the characteristics of the disk devices which are present, and provides some storage for use by the BOOS in maintaining disk directory information.

Section 4 describes the functions which must be performed by the BIOS, and the external interface to those functions. This Section contains additional information describing the structure and significance of the disk definition tables and information about sector blocking and deblocking. Careful choices of disk parameters and disk buffering methods are necessary if you are to achieve the best possible performance from CP/M-68K. Therefore, this section should be read thoroughly before writing a custom BIOS.

CP/M-68K, as distributed by Digital Research, is configured to run on the Motorola EXORmacs development system with Universal Disk . The sample BIOS in Appendix D is the BIOS used in the distributed system, and is written in C language. A sample BIOS for an Empirical Research Group (ERG) 68000 based microcomputer with Tarbell floppy disk controller is also included in Appendix B, and is written in assembly language. These examples should assist the reader in understanding how to construct his own BIOS.

5.2 Disk Definition Tables

As in other CP/M systems, CP/M-68K uses a set of tables to define disk device characteristics. This section describes each of these tables and discusses choices of certain parameters.

5.2.1 Disk Parameter Header

Each disk drive has an associated 26-byte Disk Parameter Header (DPH) which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. Each drive must, have its own unique DPH. The format of a Disk Parameter Header is shown in Figure 5-1.

XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
32b	16b	16b	16b	32b	32b	32b	32b

Figure 5-1. Disk Parameter Header

Each element of the DPH is either a word (16-bit) or longword (32-bit) value. The meanings of the Disk Parameter Header (DPH) elements are given in Table 5-1.

Table 5-1. Disk Parameter Header Elements

Element	Description
XLT	Address of the logical-to-physical sector translation table, if used for this particular drive, or the value 0 if there is no translation table for this drive (i.e, the physical and logical sector numbers are the same) . Disk drives with identical sector translation may share the same translate table. The sector translation table is described in Section 5.2.2.
0000	Three scratchpad words for use within the BDOS.
DIRBUF	Address of a 128-byte scratchpad area for directory operations within BDOS. All DPHs address the same scratchpad area.
DPB	Address of a disk parameter block for this drive. Drives with identical disk characteristics may address the same disk parameter block.

Table 5-1. (continued)

Element	Description
CSV	Address of a checksum vector. The BDOS uses this area to maintain a vector of directory checksums for the disk. These checksums are used in detecting when the disk in a drive has been changed. If the disk is not removable, then it is not necessary to have a checksum vector. Each DPH must point to a unique checksum vector. The checksum vector should contain 1 byte for every four directory entries (or 128 bytes of directory). In other words: length (CSV) = (DRM+l) / 4. (DRM is discussed in Section 5.2.3.)
ALV	Address of a scratchpad area used by the BDOS to keep disk storage allocation information. The area must be different for each DPH. There must be 1 bit for each allocation block on the drive, requiring the following: length (ALV) =- (DSM/8) + 1. (DSM is discussed below.)

5.2.2 Sector Translate Table

Sector translation in CP/M-68K is a method of logically renumbering the sectors on each disk track to improve disk I/O performance. A frequent situation is that a program needs to access disk sectors sequentially. However, in reading sectors sequentially, most programs lose a full disk revolution between sectors because there is not enough time between adjacent sectors to begin a new disk operation. To alleviate this problem, the traditional CP/M solution is to create a logical sector numbering scheme in which logically sequential sectors are physically separated. Thus, between two logically contiguous sectors, there is a several sector rotational delay.
The sector translate table defines the logical-to-physical mapping in use for a particular drive, if a mapping is used.

Sector translate tables are used only within the BIOS. Thus the table may have any convenient format. (Although the BDOS is aware of the sector translate table, its only interaction with the table is to get the address of the sector translate table from the DPH and to pass that address to the Sector Translate Function of the BIOS.) The most common form for a sector translate table is an n-byte (or n-word) array of physical sector numbers, where n is the number of sectors per disk track. Indexing into the table with the logical sector number yields the corresponding physical sector number.

Although you may choose any convenient logical-to-physical mapping, there is a nearly universal mapping used in the CP/M community for single-sided, single-density, 8-inch diskettes. That mapping is shown in Figure 5-2. Because your choice of mapping affects diskette compatibility between different systems, the mapping of Figure 5-2 is strongly recommended.

Logical Sector	0	1	2	3	4	5	6	7	8	9	10	11	12
Physical Sector	1	7	13	19	25	5	11	17	23	3	9	15	21
Logical Sector	13	14	15	16	17	18	19	20	21	22	23	24	25
Physical Sector	2	8	14	20	26	6	12	18	24	4	10	16	22

Figure 5-2. Sample Sector Translate Table

5.2.3 Disk Parameter Block

A Disk Parameter Block (DPB) defines several characteristics associated with a particular disk drive. Among them are the size of the drive, the number of sectors per track, the amount of directory space, and others.

A Disk Parameter Block can be used in one or more DPH's if the disks are identical in definition. A discussion of the fields of the DPB follows the format description. The format of the DPB is shown in Figure 5-3.

SPT	BSH	BLM	EXM	0	DSM	DRM	Reserved	CKS	OFF
16b	8b	8b	8b	8b	16b	16b	16b	16b	6b

Figure 5-3. Disk Parameter Block

Each field is a word (16 bit) or a byte (8 bit) value. The description of each field is given in Table 5-2.

Table 5-2. Disk Parameter Block Fields

Field	Definition
SPT	Number of 128-byte logical sectors per track.
BSH	The block shift factor, determined by the data block allocation size, as shown in Table 5-3.

Table 5-2. (continued)

Field	Definition
BLM	The block mask which is determined by the data block allocation size, as shown in Table 5-3.
EXM	The extent mask, determined by the data block allocation size and the number of disk blocks, as shown in Table 5-4.
0	Reserved byte.
DSM	Determines the total storage capacity of the disk drive and is the number of the last block, counting from 0. That is, the disk contains DSM+1 blocks.
DRM	Determines the total number of directory entries which can be stored on this drive. DRM is the number of the last directory entry, counting from 0. That is, the disk contains DRM+L directory entries. Each directory entry requires 32 bytes, and for maximum efficiency, the value of DRM should be chosen so that the directory entries exactly fill an integral number of allocation units.
CKS	The size of the directory check vector, which is zero if the disk is permanently mounted, or length (CSV) = (DRM) / 4 + I for removable media.
OFF	The number of reserved tracks at the beginning of a logical disk. This is the number of the track on which the directory begins.

To choose appropriate values for the Disk Parameter Block elements, you must understand how disk space is organized in CP/M-68K. A CP/M-68K disk has two major areas: the boot or system tracks, and the file system tracks. The boot tracks are usually used to hold a machine-dependent bootstrap loader for the operating system. They consist of tracks 0 to 0FF-1. Zero is a legal value for 0FF, and in that case, there are no boot tracks. The usual value of 0FF for 8-inch floppy disks is two.

The tracks after the boot tracks (beginning with track number 0FF) are used for the disk directory and disk files. Disk space in this area is grouped into units called allocation units or blocks. The block size for a particular disk is a constant, called BLS. BLS may take on any one of these values: 1024, 2048, 4096, 8192, or 16384 bytes. No other values for BLS are allowed. (Note that BLS does not appear explicitly in any BIOS table. However, it determines the values of a number of other parameters.) The DSM field in the Disk Parameter Block is one less than the number of blocks on the disk. Space is allocated to a file or to the directory in whole blocks. No fraction of a block can be allocated. block size

The choice of BLS is very important, because it effects the efficiency of disk space utilization, and because for any disk size there is a minimum value of BLS that allows the entire disk to be used. Each block on the disk has a block number ranging from 0 to DSM. The largest block number allowed is 32767. Therefore, the largest number of bytes that can be addressed in the file system space is 32768 * BLS. Because the largest allowable value for BLS is 16384, the biggest disk that can be accessed by CP/M-68K is 16384*32768 = 512 Mbytes.

Each directory entry may contain either 8 block numbers (if DSM >= 256) or 16 block numbers (if DSM < 256). Each file needs enough directory entries to hold the block numbers of all blocks allocated to the f i le. Thus a large value for BLS implies that fewer directory entries are needed. Since fewer directory entries are used, the directory search time is decreased.

The disadvantage of a large value for BLS is that since files are allocated BLS bytes at a time, there is potentially a large unused portion of a block at the end of the file. If there are many small files on a disk, the waste can be very significant.

The BSH and BLM parameters in the DPB are functions of BLS. Once you have chosen BLS, you should use Table 5-3 to determine BSH and BIM. The EXM parameter of the DPB is a function of BLS and DSM. You should use Table 5-4 to find the value of EXM for your disk.

Table 5-3. BSH and BLM Values

BLS	BSH	BLM
1024	3	7
2048	4	15
4096	5	31
8192	6	63
16384	7	127

Table 5-4. EXM Values

BLS	DSM <= 255	DSM > 255
1024	0	N/A
2048	1	0
4096	3	1
8192	7	3
16384	15	7

The DRM entry in the DPB is one less than the total number of directory entries. DRM should be chosen large enough so that you do not run out of directory entries before running out of disk space. It is not possible to give an exact rule for determining DRM, since the number of directory entries needed will depend on the number and sizes of the files present on the disk.

The CKS entry in the DPB is the number of bytes in the CSV (checksum vector) which was pointed to by the DPH. If the disk is not removable, a checksum vector is not needed, and this value may be zero.

5.3 Disk Blocking

When the BDOS does a disk read or write operation using the BIOS, the unit of information read or written is a 128-byte sector. This may or may not correspond to the actual physical sector size of the disk. If not, the BIOS must implement a method of representing the 128-byte sectors used by CP/M-68K on the actual device. Usually if the physical sectors are not 128 bytes long, they will be some multiple of 128 bytes. Thus, one physical sector can hold some integer number of 128-byte CP/M sectors. In this case, any disk I/O will actually consist of transferring several CP/M sectors at once.

It might also be desirable to do disk I/O in units of several 128-byte sectors in order to increase disk throughput by decreasing rotational latency. (Rotational latency is the average time it takes for the desired position on a disk to rotate around to the read/write head. Generally this averages 1/2 disk revolution per transfer.) Since a great deal of disk I/O is sequential, rotational latency can be greatly reduced by reading several sectors at a time, and saving them for future use.

In both the cases above, the point of interest is that physical I/O occurs in units larger than the expected sector size of 128 bytes. Some of the problems in doing disk I/O in this manner are discussed below.

5.3.1 A Simple Approach

This section presents a simple approach to handling a physical sector size larger than the logical sector size. The method discussed in this section is not recommended for use in a real BIOS. Rather, it is given as a starting point for refinements discussed in the following sections. Its simplicity also makes it a logical choice for a first BIOS on new hardware. However, the disk throughput that you can achieve with this method is poor, and the refinements discussed later give dramatic improvements.

Probably the easiest method for handling a physical sector size which is a multiple of 128 bytes is to have a single buffer the size of the physical sector internal to the BIOS. Then, when a disk read is to be done, the physical sector containing the desired 128-byte logical sector is read into the buffer, and the appropriate 128 bytes are copied to the DMA address. Writing is a little more complicated. You only want to put data into a 128-byte portion of the physical sector, but you can only write a whole physical sector. Therefore, you must first read the physical sector into the BIOS's buffer; copy the 128 bytes of output data into the proper 128-byte piece of the physical sector in the buffer; and finally write the entire physical sector back to disk.

Note: this operation involves two rotational latency delays in addition to the time needed to copy the 128 bytes of data. In fact, the second rotational wait is probably nearly a full disk revolution, since the copying is usually much faster than a disk revolution.

5.3.2 Some Refinements

There are some easy things that can be done to the algorithm of Section 5.2.1 to improve its performance. The first is based on the fact that disk accesses are usually done sequentially. Thus, if data from a certain physical sector is needed, it is likely that another piece of that sector will be needed on the next disk operation. To take advantage of this fact, the BIOS can keep information with its physical sector buffer as to which disk, track, and physical sector (if any) is represented in the buffer. Then, when reading, the BIOS need only do physical disk reads when the information needed is not in the buffer.

On writes, the BIOS still needs to preread the physical sector for the same reasons discussed in Section 5.2.1, but once the physical sector is in the buffer, subsequent writes into that physical sector do not require additional prereads. An additional saving of disk accesses can be gained by not writing the sector to the disk until absolutely necessary. The conditions under which the physical sector must be written are discussed in Section 5.3.4.

5.3.3 Track Buffering

Track buffering is a special case of disk buffering where the I/O is done a full track at a time. When sufficient memory for several full track buffers is available, this method is quite good. The method is essentially the same as discussed in Section 5.3.2, but there are some interesting features. First, transferring an entire track is much more efficient than transferring a single sector. The rotational latency is incurred only once for the entire track, whereas if the track is transferred one sector at a time, the rotational latency occurs once per sector. On a typical diskette with 26 sectors per track, rotating at 6 revolutions per second, the difference in rotational latency per track is about 2 seconds versus a twelfth of a second. Of course, in applications where the disk is accessed purely randomly, there is no advantage because there is a low probability that more than one sector will be used from a given track. However, such applications are extremely rare.

5.3.4 LRU Replacement

With any method of disk buffering using more than one buffer, it is necessary to have some algorithm for managing the buffers. That is, when should buffers be filled, and when should they be written back to disk. The first question is simple, a buffer should be filled when there is a request for a disk sector that is not presently in memory. The second issue, when to write a buffer back to disk, is more complicated.

Generally, it is desirable to defer writing a buffer until it becomes necessary. Thus, several transfers can be done to a buffer for the cost of only one disk access, two accesses if the buffer had to be preread. However, there are several reasons why buffers must be written. The following list describes the reasons:

1) A BIOS Write operation with mode=l (write to directory sector) . To maintain the integrity of CP/M-68K's file system, it is very important that directory information on the disk is kept up to date. Therefore, all directory writes should be performed immediately.

2) A BIOS Flush Buffers operation. This BIOS function is explicitly intended to force all disk buffers to be written. After performing a Flush Buffers, it is safe to remove a disk from its drive.

3) A disk buffer is needed, but all buffers are full. Therefore some buffer must be emptied to make it available for reuse.

4) A Warm Boot occurs. This is similar to number 2 above.

Case three above is the only one in which the BIOS writer has any discretion as to which buffer should be written. Probably the best strategy is to write out the buffer which has been least recently used. The fact that an area of disk has not been accessed for some time is a fairly good indication that it will not be needed again soon.

5.3.5 The New Block Flag

As explained in Section 5.2.2, the BDOS allocates disk space to files in blocks of BLS bytes. When such a block is first allocated to a file, the information previously in that block need not be preserved. To enable the BIOS to take advantage of this fact, the BDOS uses a special parameter in calling the BIOS Write Function. If register Dl.W contains the value 2 on a BIOS Write call, then the write being done is to the first sector of a newly allocated disk block. Therefore, the BIOS need not preread any sector of that block. If the BIOS does disk buffering in units of BLS bytes, it can simply mark any free buffer as corresponding to the disk address specified in this write, because the contents of the newly allocated block are not important. If the BIOS uses a buffer size other than BLS, then the algorithm for taking full advantage of this information is more complicated.

This information is extremely valuable in reducing disk delays. Consider the case where one file is read sequentially and copied to a newly created file. Without the information about newly allocated disk blocks, every physical write would require a preread. With the information, no physical write requires a preread. Thus, the number of physical disk operations is reduced by one third. 0

End of Section 5

Section 6
Installing and Adapting
the Distributed BIOS and CP/M-68K

6.1 Overview

The process of bringing up your first running CP/M-68K system is either trivial or involved, depending on your hardware environment. Digital Research supplies CP/M-68K in a form suitable for booting on a Motorola EXORmacs development system. If you have an EXORmacs, you can read Section 6.1 which tells how to load the distributed system. Similarly, you can buy or lease some other machine which already runs CP/M-68K. If you do not have an EXORmacs, you can use the S-record files supplied with your distribution disks to bring up your first CP/M-68K system. This process is discussed in Section 6.2.

6.2 Booting on an EXORmacs

The CP/M-68K disk set distributed by Digital Research includes disks boot and run CP/M-68K on the Motorola EXORmacs. You can use the distribution system boot disk without modification if you have a Motorola EXORmacs system and the following configuration:

1) 128K memory (minimum)
2) a Universal Disk Controller (UDC) or Floppy Disk Controller (FDC)
3) a single-density, IBM 3740 compatible floppy disk drive
4) an EXORter Hi

To load CP/M-68K, do the following:

1) Place the disk in the first floppy drive (#FD04 with the UDC or #FDOO with the FDC).
2) Press SYSTEM RESET (front panel) and RETURN (this brings in MACSbugT.M.) .
3) Type "BO 4" if you are using the UDC, "BO 0" if you are using the FDC, and RETURN. CP/M-68K boots and begins running.

6.3 Bringing Up CP/M-68K Using the S-record Files

The CP/M-68K distribution disks contain two copies of the CP/M- 68K operating system in Motorola S-record form, for use in getting your first CP/M-68K system running. S-records (described in detail in Appendix F) are a simple ASCII representation for absolute programs. The two S-record systems contain the CCP and BDOS, but no BIOS. One of the S-record systems resides at locations 400H and up, the other is configured to occupy the top of a 128K memory space. (The exact bounds of the S-record systems may vary from release to release. There will be release notes and/or a file named README describing the exact characteristics of the S-record systems distributed on your disks.) To bring up CP/M-68K using the S-record files, you need:

1) some method of down-loading absolute data into your target system

2) a computer capable of reading the distribution disks (a CP/M-based computer that supports standard CP/M 8-inch diskettes)

3) a BIOS for your target computer

Given the above items, you (--an use the following procedure to bring a working version of CP/M-68K into your target system:

1) You must patch one location in the S-record system to link it to your BIOS's init entry point. This location will be specified in release notes and/or in a README file on your distribution disks. The patch simply consists of inserting the address of the init entry in your BIOS at one long word location in the -record system. This patching can be done either before or after down-loading the system, whichever is more convenient.

2) Your BIOS needs the address of the cop entry point in the S-record system. This can be obtained from the release notes and/or the README file.

3) Down-load the S-record system into the memory of your target computer.

4) Down-load your BIOS into the memory of your target computer.

5) Begin executing instructions at the first location of the down-loaded S-record system.

Now that you have a working version of CP/M-68K, you can use the tools provided with the distribution system for further development.

End of Section 6

Section 7
Cold Boot Automatic Command Execution

7.1 Overview

The Cold Boot Automatic Command Execution feature of CP/M-68K allows you to configure CP/M-68K so that the CCP will automatically execute a predetermined command line on cold boot. This feature can be used to start up turn-key systems, or to perform other desired operations.

7.2 Setting up Cold Boot Automatic Command Execution

The CBACE feature uses two global symbols: autost, and _usercmd. These are both defined in the CCP, which uses them on cold boot to determine whether this feature is enabled. If you want to have a CCP command automatically executed on cold boot, you should include code in your BIOS's _init routine (which is called at cold boot) to do the following:

1) The byte at _autost must be set to the value 01H.

2) The command line to be executed must be placed in memory at _usercmd and subsequent locations. The command must be terminated with a NULL (OOH) byte, and may not exceed 128 bytes in length. All alphabetic characters in the command line should be upper-case.

Once you write a BIOS that performs these two functions, you can build it into a CPM.SYS file as described in Section 2. This system, when booted, will execute the command you have built into it.

End of Section 7

Section 8
The PUTBOOT Utility

8.1 PUTBOOT Operation

The PUTBOOT utility is used to copy information (usually a bootstrap loader system) onto the system tracks of a disk. Although PUTBOOT can copy any file to the system tracks, usually the file being written is a program (the bootstrap system).

8.2 Invoking PUTBOOT

Invoke PUTBOOT with a command of the form:

PUTBOOT [-H] <filename> <drive>

where

- o -H is an optional flag discussed below;
- o <filename> is the name of the tile to be written to the system tracks;
- o <drive> is the drive specifier for the drive to which <filename> is to be written (letter in the range A-P.)

PUTBOOT writes the specified file to the system tracks of the specified drive. Sector skewing is not used; the file is written to the system tracks in physical sector number order.

Because the file that is written is normally in command file format, PUTBOOT contains special logic to strip off the first 28 bytes of the file whenever the file begins with the number 601AH, the magic number used in command files. If, by chance, the file to be written begins with 601AH, but should not have its first 28 bytes discarded, the -H flag should be specified in the PUTBOOT command line. This flag tells PUTBOOT to write the file verbatim to the system tracks.

PUTBOOT uses BDOS calls to read <filename> , and used BIOS calls to write <filename> to the system tracks. It refers to the OFF and SPT parameters in the Disk Parameter Block to determine how large the system track space is. The source and command files for PUTBOOT are supplied on the distribution disks for CP/M-68K.

End of Section 8

Appendix A
Contents of Distribution Disks

This appendix briefly describes the contents of the disks that contain CP/M-68K as distributed by Digital Research.

Table A-1. Distribution Disk Contents

File	Contents
AR68.REL	Relocatable version of the archiver/librarian.
AS68INIT	Initialization file for assembler--see AS68 documentation in the CP/M-68K Operating System Programer's Guide.
AS68.REL	Relocatable version of the assembler.
ASM.SUB	Submit file to assemble an assembly program with file type S, put the object code in filename.0, and a listing file in filename.PRN.
BIOS.0	Object file of BIOS for EXORmacs.
BIOS.C	C language source for the EXORmacs BIOS as distributed with CP/M-68K.
BIOSA.0	Object file for assembly portion of EXORmacs BIOS.
BIOSA.S	Source for the assembly language portion of the EXORmacs BIOS as distributed with CP/M-68K.
BIOSTYPS.H	Include file for use with BIOS.C.
BOOTER.0	object for EXORmacs bootstrap.
BOOTER.S	Assembly boot code for the EXORmacs.
C.SUB	Submit file to do a C compilation. Invokes all three passes of the C compiler as well as the assembler. You can compile a C program with the line: A>C filename.
C068.REL	Relocatable version of the C parser.
C168.REL	Relocatable version of the C code generator.

Table A-1. (continued)

File	Contents
CLIB	The C run-time library.
CLINK.SUB	Submit file for linking C object programs with the C run-time library.
CP68.REL	Relocatable version of the C preprocessor.
CPM.H	Include file with C definitions for CP/M-68K. See the C Programminq Guide for CP/M-68K for details.
CPM.REL	Relocatable version of CPM.SYS.
CPM.SYS	CP/M-68K operating system file for the EXORmacs.
CPMLIB	Library of object files for CP/M-68K. See Section 2.
CPMLDR.SYS	The bootstrap loader for the EXORmacs. A copy of this was written to the system tracks using PUTBOOT.
CTYPE.H	Same as above.
DDT.REL	Relocatable version of the preloader for DDT. (Loads DDT1 into the high end of the TPA.)
DDT1.68K	This is the real DDT that gets loaded into the top of the TPA. It is relocatable even though the file type is .68K, because it must be relocated to the top of the TPA each time it is used.
DUMP.REL	Relocatable version of the DUMP utility.
ED.REL	Relocatable version of the ED utility.
ELDBIOS.S	Assembly language source for the ERG sample loader BIOS.
ERGBIOS.S	Assembly language source for the ERG sample BIOS.
ERRNO.H	Same as above.
FORMAT.REL	Relocatable disk formatter for the Motorola EXORmacs.

Table A-1. (continued)

File	Contents
FORMAT.S	Assembly language source for the FORMAT utility.
INIT.REL	Relocatable version of the INIT utility.
INIT.S	Assembly language source for the INIT utility.
LCPM.SUB	Submit file to create CPM.REL for EXORmacs.
LDBIOS.0	Object file of loader BIOS for EXORmacs.
LDBIOSA.0	Object file for assembly portion of EXORmacs loader BIOS.
LDBIOSA.S	Source for the assembly language portion of the EXORmacs loader BIOS as distributed with CP/M-68K.
LDRLIB	Library of object files for creating a Bootstrap Loader. See Section 3.
L068.REL	Relocatable version of the linker.
LOADBIOS.H	Include file for use with BIOS.C, to make it into a loader BIOS.
LOADBIOS.SUB	Submit file to create loader BIOS for EXORmacs.
MAKELDR.SUB	Submit file to create CPMLDR.SYS on EXORmacs.
NORMBIOS.H	Include file for use with BIOS.C, to make it into a normal. BIOS
NORMBIOS.SUB	Submit file to create normal BIOS for EXORmacs.
NM68.REL	Relocatable version of the symbol table dump utility.
PIP.REL	Relocatable version of the PIP utility.
PORTAB.H	Same as above.
PUTBOOT.REL	Relocatable version of the PUTBOOT utility.

Table A-1. (continued)

File	Contents
PUTBOOT.S	Assembly language source for the PUTBOOT utility.
README.TXT	ASCII file containing information relevant to this shipment of CP/M-68K. This file might not be present.
RELCPM.SUB	Submit file to relocate CPM.REL into CPM.SYS.
RELOC.REL	Relocatable version of the command file relocation utility.
RELOCX.SUB b	This file is included on each disk that contains REL command files. (x is the number of the distribution disk containing the files) . It is a submit file which will relocate the REL files for the target system.
S.0	Startup routine for use with C programs-- must be first object file linked.
SENDC68.REL	Relocatable version of the S-record creation utility.
SETJMP.H	Same as above.
SIGNAL.H	Same as above.
SIZE68.REL	Relocatable version of the SIZE68 utility.
SR128K.SYS	S-record version of CP/M-68K. This version has no BIOS, and is provided for use in porting CP/M-68K to new hardware.
SR40O.SYS	S-record version of CP/M-68K. This version has no BIOS, and is provided for use in porting CP/M-68K to new hardware.
STAT.REL	Relocatable version of the STAT utility.
STDIO.H	Include file with standard I/O definitions for use with C programs. See the C Programming Guide for CP/M-68K for details.

End of Appendix A

Appendix B
Sample BIOS Written in Assembly Language

```
                            ***************************************************************
                            *                                                             *
                            *                   CP/M-68K BIOS                             *
                            *         Basic Input/Output Subsystem                        *
                            *         For ERG 68000 with Tarbell floppy disk controller   *
                            *                                                             *
                            ***************************************************************

                                    .globl  _init                   * bios initialization entry point
                                    .globl  _ccp                    * ccp entry point

00000000 23FC0000000E0000008C _init:  move.l  #traphndl,$8c         * set up trap #3 handler
0000000A 4280                       clr.l   d0                      * log on disk A, user 0
0000000C 4E75                       rts

                            traphndl:
0000000E 0C400017                   cmpi    #nfuncs,d0
00000012 6408                       bcc     trapng
00000014 E548                       lsl     #2,d0                   * multiply bios function by 4
00000016 207B0006                   movea.l 6(pc,d0),a0             * get handler address
0000001A 4E90                       jsr     (a0)                    * call handler
                            trapng:
0000001C 4E73                       rte

                            biosbase:
0000001E 00000000                   .dc.l   _init
00000022 0000007A                   .dc.l   wboot
00000026 00000080                   .dc.l   constat
0000002A 00000094                   .dc.l   conin
0000002E 000000A8                   .dc.l   conout
00000032 000000BC                   .dc.l   lstout
00000036 000000BE                   .dc.l   pun
0000003A 000000C0                   .dc.l   rdr
0000003E 000000C8                   .dc.l   home
00000042 000000D0                   .dc.l   seldsk
00000046 000000F8                   .dc.l   settrk
0000004A 00000100                   .dc.l   setsec
0000004E 00000114                   .dc.l   setdma
00000052 0000011C                   .dc.l   read
00000056 0000015E                   .dc.l   write
0000005A 000000C2                   .dc.l   listst
```

Listing B-1. Sample Assembly Language BIOS

```
0000005E 00000108             .dc.l  sectran
00000062 00000114             .dc.l  setdma
00000066 0000029C             .dc.l  getseg
0000006A 000002A4             .dc.l  getiob
0000006E 000002A6             .dc.l  setiob
00000072 00000298             .dc.l  flush
00000076 000002A8             .dc.l  setexc

                      nfuncs=(*-biosbase)/4

0000007A 4EF900000000         wboot:  jmp    _ccp

00000080 103900FFFF01         constat: move.b $ffff01,d0        * get status byte
00000086 02400002             andi.w  #2,d0                    * data available bit on?
0000008A 6704                 beq     noton                    * branch if not
0000008C 7001                 moveq.l #$1,d0                   * set result to true
0000008E 4E75                 rts

00000090 4280         noton:  clr.l   d0                       * set result to false
00000092 4E75                 rts

00000094 61EA         conin:  bsr     constat                  * see if key pressed
00000096 4A40                 tst     d0
00000098 67FA                 beq     conin                    * wait until key pressed
0000009A 103900FFFF00         move.b  $ffff00,d0               * get key
000000A0 C0BC0000007F         and.l   #$7f,d0                  * clear all but low 7 bits
000000A6 4E75                 rts

000000A8 103900FFFF01conout: move.b  $ffff01,d0              * get status
000000AE C03C0001             and.b   #$1,d0                   * check for transmitter buffer empty
000000B2 67F4                 beq     conout                   * wait until our port has aged...
000000B4 13C100FFFF00         move.b  d1,$ffff00               * and output it
000000BA 4E75                 rts                              * and exit

000000BC 4E75         lstout: rts

000000BE 4E75         pun:     rts

000000C0 4E75         rdr:    rts

000000C2 103C00FF listst:     move.b  #$ff,d0
000000C6 4E75                 rts

                      *
                      * Disk Handlers for Tarbell 1793 floppy disk controller
                      *
                      maxdsk  = 2                              * this BIOS supports 2 floppy drives
                      dphlen  = 26                             * length of disk parameter header

                      iobase  = $00fffff8                      * Tarbell floppy disk port base address
                      dcmd    = iobase                         * output port for command
```

Listing B-1. (continued)

```
                                dstat   = iobase                * input status port
                                dtrk    = iobase+1              * disk track port
                                dsect   = iobase+2              * disk sector port
                                ddata   = iobase+3              * disk data port
                                dwait   = iobase+4              * input port to wait for op finished
                                dcntrl  = iobase+4              * output control port for drive selection


000000C8 423900000002   home:   clr.b   track
000000CE 4E75                   rts

                        seldsk:
                        *       select disk given by register d1.b
000000D0 7000                   moveq   #0,d0
000000D2 B23C0002               cmp.b   #maxdsk,d1              * valid drive number?
000000D6 6A1E                   bpl     selrtn                  * if no, return 0 in d0
000000D8 13C100000000           move.b  d1,seldrv               * else, save drive number
000000DE E909                   lsl.b   #4,d1
000000E0 13C10000000A           move.b  d1,selcode              * select code is 00 for drv 0, $10 for drv 1
000000E6 103900000000           move.b  seldrv,d0
000000EC C0FC001A               mulu    #dphlen,d0
000000F0 D0BC00000016           add.l   #dph0,d0                * point d0 at correct dph
000000F6 4E75           selrtn: rts

000000F8 13C100000002 settrk: move.b  d1,track
000000FE 4E75                   rts

00000100 13C10000000  setsec: move.b  d1,sector
00000106 4E75                   rts

                        sectran:
                        *       translate sector in d1 with translate table pointed to by d2
                        *       result in d0
00000108 2042                   movea.l d2,a0
0000010A 48C1                   ext.l   d1
0000010C 10301000               move.b  #0(a0,d1),d0
00000110 48C0                   ext.l   d0
00000112 4E75                   rts

                        setdma:
00000114 23C100000006           move.l  d1,dma
0000011A 4E75                   rts

                        read:
                        * Read one sector from requested disk, track, sector to dma address
                        * Retry if necessary, return in d0 00 if ok, else non-zero
0000011C 13FC000A0000000B move.b  #10,errcnt            * set up retry counter
                        rretry:
00000124 61000076               bsr     setup
00000128 00430088               ori     #$88,d3                 * OR read command with head load bit
0000012C 13C300FFFFF8           move.b  d3,dcmd                 * output it to FDC
```

Listing B-1.   (continued)

```
00000132 0839000700FFFFFC rloop:  btst    #7,dwait
0000013A 6708                     beq     rdone           * if end of read, exit
0000013C 10F900FFFFFB             move.b  ddata,(a0)+     * else, move next byte of data
00000142 60EE                     bra     rloop
                          rdone:
00000144 61000146                 bsr     rstatus         * get FDC status
00000148 6604                     bne     rerror
0000014A 4280                     clr.l   d0
0000014C 4E75                     rts
0000014E 610000B0         rerror: bsr     errchk          * go to error handler
00000152 53390000000B             subq.b  #1,errcnt
00000158 66CA                     bne     rretry
0000015A 70FF                     move.l  #$ffffffff,d0
0000015C 4E75                     rts

                          write:
                          * Write one sector to requested disk, track, sector from dma address
                          * Retry if necessary, return in d0 00 if ok, else non-zero
0000015E 13FC000A0000000B         move.b  #10,errcnt      * set up retry counter
                          wretry:
00000166 6134                     bsr     setup
00000168 004300A8                 ori     #$a8,d3         * OR write command with head load bit
0000016C 13C300FFFFF8             move.b  d3,dcmd         * output it to FDC
00000172 0839000700FFFFFC wloop:  btst    #7,dwait
0000017A 6708                     beq     wdone           * if end of read, exit
0000017C 13D800FFFFFB             move.b  (a0)+,ddata     * else, move next byte of data
00000182 60EE                     bra     wloop
                          wdone:
00000184 61000106                 bsr     rstatus         * get FDC status
00000188 6604                     bne     werror
0000018A 4280                     clr.l   d0
0000018C 4E75                     rts
0000018E 6170             werror: bsr     errchk          * go to error handler
00000190 53390000000B             subq.b  #1,errcnt
00000196 66CE                     bne     wretry
00000198 70FF                     move.l  #$ffffffff,d0
0000019A 4E75                     rts

                          setup:
                          * common read and write setup code
                          * select disk, set track, set sector were all deferred until now
0000019C 13FC00D000FFFFF8         move.b  #$d0,dcmd       * clear controller, get status
000001A4 163900000001             move.b  curdrv,d3
000001AA B63900000000             cmp.b   seldrv,d3
000001B0 661A                     bne     newdrive        * if drive not selected, do it
000001B2 163900000002             move.b  track,d3
000001B8 B63900000003             cmp.b   oldtrk,d3
000001BE 6620                     bne     newtrk          * if not on right track, do it
000001C0 4283                     clr.l   d3              * if head already loaded, no head load delay
000001C2 0839000500FFFFF8         btst    #5,dstat        * if head unloaded, treat as new disk
000001CA 6618                     bne     sexit
```

Listing B-1.   (continued)

```
                                       newdrive:
000001CC 13F90000000A00FFFFFC  move.b  selcode,dcntrl      * select the drive
000001D6 13F90000000000000001  move.b  seldrv,curdrv
                                       newtrk:
000001E0 6126                  bsr     chkseek             * seek to correct track if required
000001E2 7604                  moveq   #4,d3               * force head load delay
                                       sexit:
000001E4 13F90000000400FFFFFA  move.b  sector,dsect        * set up sector number
000001EE 13F90000000200FFFFF9  move.b  track,dtrk          * set up track number
000001F8 207900000006          move.l  dma,a0              * dma address to a0
000001FE 4E75                  rts

                                       errchk:
00000200 08070004              btst.b  #4,d7
00000204 6602                  bne     chkseek             * if record not found error, reseek
00000206 4E75                  rts

                                       chkseek:
                                       *      check for correct track, seek if necessary
00000208 615C                  bsr     readid              * find out what track we're on
0000020A 671E                  beq     chks1               * if read id ok, skip restore code
                                       restore:
                                       *      home the drive and reseek to correct track
0000020C 13FC000B00FFFFF8      move.b  #$0B,dcmd           * restore command to command port
                                       rstwait:
00000214 0839000700FFFFFC      btst    #7,dwait
0000021C 66F6                  bne     rstwait             * loop until restore completed
0000021E 0839000200FFFFF8      btst    #2,dstat
00000226 67E4                  beq     restore             * if not at track 0, try again
00000228 4283                  clr.l   d3                  * track number returned in d3 from readid
                                       chks1:
0000022A 13C300FFFFF9          move.b  d3,dtrk             * update track register in FDC
00000230 13F90000000200000003  move.b  track,oldtrk        * update oldtrk
0000023A B63900000002          cmp.b   track,d3            * are we at right track?
00000240 6722                  beq     chkdone             * if yes, exit
00000242 13F90000000200FFFFFB  move.b  track,ddata         * else, put desired track in data reg of FDC
0000024C 13FC001800FFFFF8      move.b  #$18,dcmd           *   and issue a seek command
00000254 0839000700FFFFFC chks2: btst  #7,dwait
0000025C 66F6                  bne     chks2               * loop until seek complete
0000025E 163900FFFFF8          move.b  dstat,d3            * read status to clear FDC
                                       chkdone:
00000264 4E75                  rts

                                       readid:
                                       *      read track id, return track number in d3
00000266 13FC00C400FFFFF8      move.b  #$c4,dcmd           * issue read id command
0000026E 1E3900FFFFFC          move.b  dwait,d7            * wait for intrq
00000274 163900FFFFFB          move.b  ddata,d3            * track byte to d3
                                       rid2:
0000027A 0839000700FFFFFC      btst    #7,dwait
00000282 6708                  beq     rstatus             * wait for intrq
```

Listing B-1. (continued)

```
00000284 1E3900FFFFFB      move.b  ddata,d7          * read another byte
0000028A 60EE              bra     rid2              * and loop
                   rstatus:
0000028C 1E3900FFFFF8      move.b  dstat,d7
00000292 0207009D          andi.b  #$9d,d7           * set condition codes
00000296 4E75              rts


                 flush:
00000298 4280              clr.l   d0                * return successful
0000029A 4E75              rts

                   getseg:
0000029C 203C0000000C      move.l  #memrgn,d0        * return address of mem region table
000002A2 4E75              rts

                   getiob:
000002A4 4E75              rts

                   setiob:
000002A6 4E75              rts

                   setexc:
000002A8 0281000000FF      andi.l  #$ff,d1           * do only for exceptions 0 - 255
000002AE E549              lsl     #2,d1             * multiply exception nmbr by 4
000002B0 2041              movea.l d1,a0
000002B2 2010              move.l  (a0),d0           * return old vector value
000002B4 2082              move.l  d2,(a0)           * insert new vector
000002B6 4E75      noset:  rts


00000000                   .data

00000000 FF        seldrv:  .dc.b  $ff               * drive requested by seldsk
00000001 FF        curdrv:  .dc.b  $ff               * currently selected drive

00000002 00        track:   .dc.b  0                 * track requested by settrk
00000003 00        oldtrk:  .dc.b  0                 * track we were on

00000004 0000      sector:  .dc.w  0
00000006 00000000 dma:      .dc.l  0
0000000A 00        selcode: .dc.b  0                 * drive select code

0000000B 0A        errcnt:  .dc.b  10                * retry counter

0000000C 0001      memrgn:.dc.w  1                   * 1 memory region
0000000E 00000800           .dc.l  $800              * starts at 800 hex
00000012 00017800           .dc.l  $17800            * goes until 18000 hex


                 * disk parameter headers
```

Listing B-1.   (continued)

```

00000016 0000005A  dph0:   .dc.l   xlt
0000001A 0000              .dc.w   0              * dummy
0000001C 0000              .dc.w   0
0000001E 0000              .dc.w   0
00000020 00000000          .dc.l   dirbuf         * ptr to directory buffer
00000024 0000004A          .dc.l   dpb            * ptr to disk parameter block
00000028 00000080          .dc.l   ckv0           * ptr to check vector
0000002C 000000A0          .dc.l   alv0           * ptr to allocation vector

00000030 0000005A  dph1:   .dc.l   xlt
00000034 0000              .dc.w   0              * dummy
00000036 0000              .dc.w   0
00000038 0000              .dc.w   0
0000003A 00000000          .dc.l   dirbuf         * ptr to directory buffer
0000003E 0000004A          .dc.l   dpb            * ptr to disk parameter block
00000042 00000090          .dc.l   ckv1           * ptr to check vector
00000046 000000C0          .dc.l   alv1           * ptr to allocation vector

                   * disk parameter block

0000004A 001A      dpb:    .dc.w   26             * sectors per track
0000004C 03                .dc.b   3              * block shift
0000004D 07                .dc.b   7              * block mask
0000004E 00                .dc.b   0              * extent mask
0000004F 00                .dc.b   0              * dummy fill
00000050 00F2              .dc.w   242            * disk size
00000052 003F              .dc.w   63             * 64 directory entries
00000054 C000              .dc.w   $c000          * directory mask
00000056 0010              .dc.w   16             * directory check size
00000058 0002              .dc.w   2              * track offset

                   * sector translate table

0000005A 01070D13  xlt:    .dc.b   1, 7,13,19
0000005E 19050B11          .dc.b   25, 5,11,17
00000062 1703090F          .dc.b   23, 3, 9,15
00000066 1502080E          .dc.b   21, 2, 8,14
0000006A 141A060C          .dc.b   20,26, 6,12
0000006E 1218040A          .dc.b   18,24, 4,10
00000072 1016              .dc.b   16,22


00000000                   .bss

00000000           dirbuf: .ds.b   128            * directory buffer

00000080           ckv0:   .ds.b   16             * check vector
00000090           ckv1:   .ds.b   16

000000A0           alv0:   .ds.b   32             * allocation vector
```

Listing B-1.   (continued)

```
000000C0          alv1:   .ds.b  32

000000E0                  .end
```

S y m b o l   T a b l e

```
_ccp      ******** EXT   _init    00000000 TEXT  alv0     000000A0 BSS   alv1     000000C0 BSS
biosbase  0000001E TEXT  chkdone  00000264 TEXT  chks1    0000022A TEXT  chks2    00000254 TEXT
chkseek   00000208 TEXT  ckv0     00000080 BSS   ckv1     00000090 BSS   conin    00000094 TEXT
conout    000000A8 TEXT  constat  00000080 TEXT  curdrv   00000001 DATA  dcmd     00FFFFF8 ABS
dcntrl    00FFFFFC ABS   ddata    00FFFFFB ABS   dirbuf   00000000 BSS   dma      00000006 DATA
dpb       0000004A DATA  dph0     00000016 DATA  dph1     00000030 DATA  dphlen   0000001A ABS
dsect     00FFFFFA ABS   dstat    00FFFFF8 ABS   dtrk     00FFFFF9 ABS   dwait    00FFFFFC ABS
errchk    00000200 TEXT  errcnt   0000000B DATA  flush    00000298 TEXT  getiob   000002A4 TEXT
getseg    0000029C TEXT  home     000000C8 TEXT  iobase   00FFFFF8 ABS   listst   000000C2 TEXT
lstout    000000BC TEXT  maxdsk   00000002 ABS   memrgn   0000000C DATA  newdrive 000001CC TEXT
newtrk    000001E0 TEXT  nfuncs   00000017 ABS   noset    000002B6 TEXT  noton    00000090 TEXT
oldtrk    00000003 DATA  pun      000000BE TEXT  rdone    00000144 TEXT  rdr      000000C0 TEXT
read      0000011C TEXT  readid   00000266 TEXT  rerror   0000014E TEXT  restore  0000020C TEXT
rid2      0000027A TEXT  rloop    00000132 TEXT  rretry   00000124 TEXT  rstatus  0000028C TEXT
rstwait   00000214 TEXT  sector   00000004 DATA  sectran  00000108 TEXT  selcode  0000000A DATA
seldrv    00000000 DATA  seldsk   000000D0 TEXT  selrtn   000000F6 TEXT  setdma   00000114 TEXT
setexc    000002A8 TEXT  setiob   000002A6 TEXT  setsec   00000100 TEXT  settrk   000000F8 TEXT
setup     0000019C TEXT  sexit    000001E4 TEXT  track    00000002 DATA  traphndl 0000000E TEXT
trapng    0000001C TEXT  wboot    0000007A TEXT  wdone    00000184 TEXT  werror   0000018E TEXT
wloop     00000172 TEXT  wretry   00000166 TEXT  write    0000015E TEXT  xlt      0000005A DATA
```

Listing B-1. (continued)

End of Appendix B

Appendix C
Sample Loader BI0S Written in Assembly Language

```
C P / M   6 8 0 0 0   A s s e m b l e r        Revision 02.01      Page   1
Source File: eldbios.s

                         ***************************************************************
                         *                                                             *
                         *                  CP/M-68K Loader BIOS                       *
                         *        Basic Input/Output Subsystem                         *
                         *        For ERG 68000 with Tarbell floppy disk controller    *
                         *                                                             *
                         ***************************************************************


                                  .globl  _bios                 * declare external entry point


                         _bios:
   00000000 0C400017              cmpi    #nfuncs,d0
   00000004 6C08                  bge     nogood
   00000006 E548                  lsl     #2,d0                 * multiply bios function by 4
   00000008 207B0006              movea.l 6(pc,d0),a0           * get handler address
   0000000C 4E90                  jsr     (a0)                  * call handler
                         nogood:
   0000000E 4E75                  rts

                         biosbase:
   00000010 0000000E              .dc.l  nogood
   00000014 0000000E              .dc.l  nogood
   00000018 0000006C              .dc.l  constat
   0000001C 00000080              .dc.l  conin
   00000020 00000094              .dc.l  conout
   00000024 0000000E              .dc.l  nogood
   00000028 0000000E              .dc.l  nogood
   0000002C 0000000E              .dc.l  nogood
   00000030 000000A8              .dc.l  home
   00000034 000000B0              .dc.l  seldsk
   00000038 000000C4              .dc.l  settrk
   0000003C 000000CC              .dc.l  setsec
   00000040 000000E0              .dc.l  setdma
   00000044 000000E8              .dc.l  read
   00000048 0000000E              .dc.l  nogood
   0000004C 0000000E              .dc.l  nogood
   00000050 000000D4              .dc.l  sectran
   00000054 000000E0              .dc.l  setdma
   00000058 0000000E              .dc.l  nogood
   0000005C 0000000E              .dc.l  nogood
```

Listing C-1. Sample BIOS Loader

```
00000060 0000000E            .dc.l  nogood
00000064 0000000E            .dc.l  nogood
00000068 00000222            .dc.l  setexc
                     nfuncs=(*-biosbase)/4
0000006C 103900FFFF01 constat: move.b $ffff01,d0        * get status byte
00000072 02400002            andi.w  #2,d0              * data available bit on?
00000076 6704                beq     noton              * branch if not
00000078 7001                moveq.l #$1,d0             * set result to true
0000007A 4E75                rts
0000007C 4280        noton:  clr.l   d0                 * set result to false
0000007E 4E75                rts
00000080 61EA        conin:  bsr     constat            * see if key pressed
00000082 4A40                tst     d0
00000084 67FA                beq     conin              * wait until key pressed
00000086 103900FFFF00        move.b  $ffff00,d0         * get key
0000008C C0BC0000007F        and.l   #$7f,d0            * clear all but low 7 bits
00000092 4E75                rts
00000094 103900FFFF01conout: move.b  $ffff01,d0         * get status
0000009A C03C0001            and.b   #$1,d0             * check for transmitter buffer empty
0000009E 67F4                beq     conout             * wait until our port has aged...
000000A0 13C100FFFF00        move.b  d1,$ffff00         * and output it
000000A6 4E75                rts                        * and exit
                     *
                     * Disk Handlers for Tarbell 1793 floppy disk controller
                     *
                     maxdsk  = 2                        * this BIOS supports 2 floppy drives
                     dphlen  = 26                       * length of disk parameter header
                     iobase  = $00ffff8                 * Tarbell floppy disk port base address
                     dcmd    = iobase                   * output port for command
                     dstat   = iobase                   * input status port
                     dtrk    = iobase+1                 * disk track port
                     dsect   = iobase+2                 * disk sector port
                     ddata   = iobase+3                 * disk data port
                     dwait   = iobase+4                 * input port to wait for op finished
                     dcntrl  = iobase+4                 * output control port for drive selection
000000A8 423900000002 home:  clr.b   track
000000AE 4E75                rts
                     seldsk:
                     *       select disk A
000000B0 423900000000        clr.b   seldrv             * select drive
```

Listing C-1.   (continued)

```
000000B6 42390000000A        clr.b   selcode          * select code is 00 for drv 0, $10 for drv 1
000000BC 203C0000000C        move.l  #dph0,d0
000000C2 4E75        selrtn: rts

000000C4 13C100000002 settrk: move.b  d1,track
000000CA 4E75                rts

000000CC 13C100000004 setsec: move.b  d1,sector
000000D2 4E75                rts

                     sectran:
                     *       translate sector in d1 with translate table pointed to by d2
                     *       result in d0
000000D4 2042                movea.l d2,a0
000000D6 48C1                ext.l   d1
000000D8 10301000            move.b  #0(a0,d1),d0
000000DC 48C0                ext.l   d0
000000DE 4E75                rts

                     setdma:
000000E0 23C100000006        move.l  d1,dma
000000E6 4E75                rts

                     read:
                     * Read one sector from requested disk, track, sector to dma address
                     * Retry if necessary, return in d0 00 if ok, else non-zero
000000E8 13FC000A0000000B  move.b  #10,errcnt     * set up retry counter
                     rretry:
000000F0 6134                bsr     setup
000000F2 00430088            ori     #$88,d3          * OR read command with head load bit
000000F6 13C300FFFFF8        move.b  d3,dcmd          * output it to FDC
000000FC 0839000700FFFFFC rloop: btst  #7,dwait
00000104 6708                beq     rdone            * if end of read, exit
00000106 10F900FFFFFB        move.b  ddata,(a0)+      * else, move next byte of data
0000010C 60EE                bra     rloop
                     rdone:
0000010E 61000106            bsr     rstatus          * get FDC status
00000112 6604                bne     rerror
00000114 4280                clr.l   d0
00000116 4E75                rts
00000118 6170        rerror: bsr     errchk           * go to error handler
0000011A 53390000000B        subq.b  #1,errcnt
00000120 66CE                bne     rretry
00000122 70FF                move.l  #$ffffffff,d0
00000124 4E75                rts


                     setup:
                     * common read and write setup code
                     * select disk, set track, set sector were all deferred until now
00000126 13FC00D000FFFFF8  move.b  #$d0,dcmd      * clear controller, get status
0000012E 163900000001        move.b  curdrv,d3
```

Listing C-1. (continued)

```
00000134 B63900000000       cmp.b   seldrv,d3
0000013A 661A               bne     newdrive      * if drive not selected, do it
0000013C 163900000002       move.b  track,d3
00000142 B63900000003       cmp.b   oldtrk,d3
00000148 6620               bne     newtrk        * if not on right track, do it
0000014A 4283               clr.l   d3            * if head already loaded, no head load delay
0000014C 0839000500FFFFF8   btst    #5,dstat      * if head unloaded, treat as new disk
00000154 6618               bne     sexit
                    newdrive:
00000156 13F90000000A00FFFFFC  move.b  selcode,dcntrl  * select the drive
00000160 13F9000000000000001   move.b  seldrv,curdrv
                    newtrk:
0000016A 6126               bsr     chkseek       * seek to correct track if required
0000016C 7604               moveq   #4,d3         * force head load delay
                    sexit:
0000016E 13F90000000400FFFFFA  move.b  sector,dsect   * set up sector number
00000178 13F90000000200FFFFF9  move.b  track,dtrk     * set up track number
00000182 207900000006       move.l  dma,a0        * dma address to a0
00000188 4E75               rts

                    errchk:
0000018A 08070004           btst.b  #4,d7
0000018E 6602               bne     chkseek       * if record not found error, reseek
00000190 4E75               rts

                    chkseek:
                    *       check for correct track, seek if necessary
00000192 615C               bsr     readid        * find out what track we're on
00000194 671E               beq     chks1         * if read id ok, skip restore code
                    restore:
                    *       home the drive and reseek to correct track
00000196 13FC000B00FFFFF8   move.b  #$0B,dcmd     * restore command to command port
                    rstwait:
0000019E 0839000700FFFFFC   btst    #7,dwait
000001A6 66F6               bne     rstwait       * loop until restore completed
000001A8 0839000200FFFFF8   btst    #2,dstat
000001B0 67E4               beq     restore       * if not at track 0, try again
000001B2 4283               clr.l   d3            * track number returned in d3 from readid
                    chks1:
000001B4 13C300FFFFF9       move.b  d3,dtrk       * update track register in FDC
000001BA 13F9000000020000003 move.b  track,oldtrk  * update oldtrk
000001C4 B63900000002       cmp.b   track,d3      * are we at right track?
000001CA 6722               beq     chkdone       * if yes, exit
000001CC 13F90000000200FFFFFB move.b  track,ddata   * else, put desired track in data reg of FDC
000001D6 13FC001800FFFFF8   move.b  #$18,dcmd     *    and issue a seek command
000001DE 0839000700FFFFFC chks2:  btst    #7,dwait
000001E6 66F6               bne     chks2         * loop until seek complete
000001E8 163900FFFFF8       move.b  dstat,d3      * read status to clear FDC
                    chkdone:
000001EE 4E75               rts

                    readid:
```

Listing C-1.   (continued)

```
                                *      read track id, return track number in d3
000001F0 13FC00C400FFFFF8  move.b  #$c4,dcmd          * issue read id command
000001F8 1E3900FFFFFC      move.b  dwait,d7           * wait for intrq
000001FE 163900FFFFFB      move.b  ddata,d3           * track byte to d3
                  rid2:
00000204 0839000700FFFFFC  btst    #7,dwait
0000020C 6708              beq     rstatus            * wait for intrq
0000020E 1E3900FFFFFB      move.b  ddata,d7           * read another byte
00000214 60EE              bra     rid2               * and loop
                  rstatus:
00000216 1E3900FFFFF8      move.b  dstat,d7
0000021C 0207009D          andi.b  #$9d,d7            * set condition codes
00000220 4E75              rts
                  setexc:
00000222 0281000000FF      andi.l  #$ff,d1            * do only for exceptions 0 - 255
00000228 E549              lsl     #2,d1              * multiply exception number by 4
0000022A 2041              movea.l d1,a0
0000022C 2010              move.l  (a0),d0            * return old vector value
0000022E 2082              move.l  d2,(a0)            * insert new vector
00000230 4E75              rts
00000000                   .data
00000000 FF       seldrv:  .dc.b   $ff                * drive requested by seldsk
00000001 FF       curdrv:  .dc.b   $ff                * currently selected drive
00000002 00       track:   .dc.b   0                  * track requested by settrk
00000003 00       oldtrk:  .dc.b   0                  * track we were on
00000004 0000     sector:  .dc.w   0
00000006 00000000 dma:     .dc.l   0
0000000A 00       selcode: .dc.b   0                  * drive select code
0000000B 0A       errcnt:  .dc.b   10                 * retry counter
                  * disk parameter headers
0000000C 00000036 dph0:    .dc.l   xlt
00000010 0000              .dc.w   0                  * dummy
00000012 0000              .dc.w   0
00000014 0000              .dc.w   0
00000016 00000000          .dc.l   dirbuf             * ptr to directory buffer
0000001A 00000026          .dc.l   dpb                * ptr to disk parameter block
0000001E 00000000          .dc.l   0                  * ptr to check vector
00000022 00000000          .dc.l   0                  * ptr to allocation vector
                  * disk parameter block
```

Listing C-1.   (continued)

```

00000026 001A            dpb:    .dc.w   26          * sectors per track
00000028 03                      .dc.b   3           * block shift
00000029 07                      .dc.b   7           * block mask
0000002A 00                      .dc.b   0           * extent mask
0000002B 00                      .dc.b   0           * dummy fill
0000002C 00F2                    .dc.w   242         * disk size
0000002E 003F                    .dc.w   63          * 64 directory entries
00000030 C000                    .dc.w   $c000       * directory mask
00000032 0010                    .dc.w   16          * directory check size
00000034 0002                    .dc.w   2           * track offset

                         * sector translate table

00000036 01070D13 xlt:           .dc.b    1, 7,13,19
0000003A 19050B11                .dc.b   25, 5,11,17
0000003E 1703090F                .dc.b   23, 3, 9,15
00000042 1502080E                .dc.b   21, 2, 8,14
00000046 141A060C                .dc.b   20,26, 6,12
0000004A 1218040A                .dc.b   18,24, 4,10
0000004E 1016                    .dc.b   16,22


00000000                         .bss

00000000                 dirbuf:  .ds.b   128     * directory buffer


00000080                         .end
```

S y m b o l   T a b l e

```
_bios    00000000 TEXT  biosbase 00000010 TEXT  chkdone  000001EE TEXT  chks1    000001B4 TEXT
chks2    000001DE TEXT  chkseek  00000192 TEXT  conin    00000080 TEXT  conout   00000094 TEXT
constat  0000006C TEXT  curdrv   00000001 DATA  dcmd     00FFFFF8 ABS   dcntrl   00FFFFFC ABS
ddata    00FFFFFB ABS   dirbuf   00000000 BSS   dma      00000006 DATA  dpb      00000026 DATA
dph0     0000000C DATA  dphlen   0000001A ABS   dsect    00FFFFFA ABS   dstat    00FFFFF8 ABS
dtrk     00FFFFF9 ABS   dwait    00FFFFFC ABS   errchk   0000018A TEXT  errcnt   0000000B DATA
home     000000A8 TEXT  iobase   00FFFFF8 ABS   maxdsk   00000002 ABS   newdrive 00000156 TEXT
newtrk   0000016A TEXT  nfuncs   00000017 ABS   nogood   0000000E TEXT  noton    0000007C TEXT
oldtrk   00000003 DATA  rdone    0000010E TEXT  read     000000E8 TEXT  readid   000001F0 TEXT
rerror   00000118 TEXT  restore  00000196 TEXT  rid2     00000204 TEXT  rloop    000000FC TEXT
rretry   000000F0 TEXT  rstatus  00000216 TEXT  rstwait  0000019E TEXT  sector   00000004 DATA
sectran  000000D4 TEXT  selcode  0000000A DATA  seldrv   00000000 DATA  seldsk   000000B0 TEXT
selrtn   000000C2 TEXT  setdma   000000E0 TEXT  setexc   00000222 TEXT  setsec   000000CC TEXT
settrk   000000C4 TEXT  setup    00000126 TEXT  sexit    0000016E TEXT  track    00000002 DATA
xlt      00000036 DATA
```

Listing C-1.   (continued)

End of Appendix C

Appendix D
EXORmacs BIOS Written in C

This Appendix contains several files in addition to the C BIOSproper. First, the C BIOS includes conditional compilation to make it into either a loader BIOS or a normal BIOS, and there is an include file for each possibility. One of these include files should be renamed BIOSTYPE.H before compiling the BIOS. The choice of which file is used as BIOSTYPE.H determines whether a normal or loader BIOS is compiled. Both the normal and the loader BIOSes need assembly language interfaces, and they are not the same. Both assembly interface modules are given. Finally, there is an include file that defines some standard variable types.

Listing D-1. EXORmacs BIOS.C File

```
/*=========================================================================*/
/*/---------------------------------------------------------------------------*/
/*|                                                                         |*/
/*|             CP/M-68K(tm) BIOS for the EXORMACS                          |*/
/*|             Copyright 1983, Digital Research.                           |*/
/*|                                                                         |*/
/*|                  Modified 9/ 7/82 wbt                                   |*/
/*|                          10/ 5/82 wbt                                   |*/
/*|                          12/15/82 wbt                                   |*/
/*|                          12/22/82 wbt                                   |*/
/*|                          1/28/83 wbt                                    |*/
/*|                          2/05/84 sw        V1.2                         |*/
/*|                                                                         |*/
/*\---------------------------------------------------------------------------/*/
/*=========================================================================*/

#include "biostype.h"   /* defines LOADER : 0-> normal bios, 1->loader bios     */
                        /* also defines CTLTYPE 0 -> Universal Disk Cntrlr      */
                        /*                      1 -> Floppy Disk Controller     */
                        /* MEMDSK: 0 -> no memory disk                          */
                        /*         4 -> 384K memory disk                        */

#include "biostyps.h"   /* defines portable variable types */

char copyright[] = "Copyright 1983, Digital Research";

struct memb { BYTE byte; };     /* use for peeking and poking memory */
struct memw { WORD word; };
struct meml { LONG lword;};


/**************************************************************************/
/*      I/O Device Definitions                                            */
/**************************************************************************/
```

```
/****************************************************************************/
/*      Define the two serial ports on the DEBUG board                      */
/****************************************************************************/

/* Port Addresses */

#define PORT1 0xFFEE011        /* console port */
#define PORT2 0xFFEE015        /* debug port   */

/* Port Offsets */

#define PORTCTRL 0      /* Control Register */
#define PORTSTAT 0      /* Status  Register */
#define PORTRDR  2      /* Read  Data Register */
#define PORTTDR  2      /* Write Data Register */

/* Port Control Functions */

#define PORTRSET 3      /* Port Reset */
#define PORTINIT 0x11   /* Port Initialize */

/* Port Status Values */

#define PORTRDRF 1      /* Read  Data Register Full */
#define PORTTDRE 2      /* Write Data Register Empty */


/****************************************************************************/
/* Define Disk I/O Addresses and Related Constants                          */
/****************************************************************************/

#define DSKIPC          0xFF0000        /* IPC Base Address */

#define DSKINTV         0x3FC           /* Address of Disk Interrupt Vector */

#define INTTOIPC        0xD             /* offsets in mem mapped io area */
#define RSTTOIPC        0xF
#define MSGTOIPC        0x101
#define ACKTOIPC        0x103
#define  PKTTOIPC       0x105
#define  MSGFMIPC       0x181
#define ACKFMIPC        0x183
#define PKTFMIPC        0x185

#define DSKREAD         0x10            /* disk commands */
#define DSKWRITE        0x20

/* Some characters used in disk controller packets */

#define  STX    0x02
#define ETX     0x03
#define  ACK    0x06
#define NAK     0x15
```

```
#define PKTSTX        0x0           /* offsets within a disk packet */
#define PKTID         0x1
#define PKTSZ         0x2
#define PKTDEV        0x3
#define PKTCHCOM      0x4
#define PKTSTCOM      0x5
#define  PKTSTVAL     0x6
#define PKTSTPRM      0x8
#define  STPKTSZ      0xf


/************************************************************************/
/* BIOS  Table Definitions                                              */
/************************************************************************/

/* Disk Parameter Block Structure */

struct dpb
{
          WORD  spt;
          BYTE  bsh;
          BYTE  blm;
          BYTE  exm;
          BYTE  dpbjunk;
          WORD  dsm;
          WORD  drm;
          BYTE  al0;
          BYTE  al1;
          WORD  cks;
          WORD  off;
};


/* Disk Parameter Header Structure */

struct dph
{
          BYTE   *xltp;
          WORD    dphscr[3];
          BYTE   *dirbufp;
struct    dpb    *dpbp;
          BYTE   *csvp;
          BYTE   *alvp;
};



/************************************************************************/
/*        Directory Buffer for use by the BDOS                          */
/************************************************************************/

BYTE dirbuf[128];

#if ! LOADER
```

```
/*************************************************************************/
/*      CSV's                                                            */
/*************************************************************************/

BYTE    csv0[16];
BYTE    csv1[16];
#if ! CTLTYPE

BYTE    csv2[256];
BYTE    csv3[256];

#endif

#if     MEMDSK
BYTE    csv4[16];
#endif

/*************************************************************************/
/*      ALV's                                                            */
/*************************************************************************/

BYTE    alv0[32];       /* (dsm0 / 8) + 1 */
BYTE    alv1[32];       /* (dsm1 / 8) + 1 */

#if ! CTLTYPE

BYTE    alv2[412];      /* (dsm2 / 8) + 1 */
BYTE    alv3[412];      /* (dsm2 / 8) + 1 */

#endif

#if     MEMDSK
BYTE    alv4[48];       /* (dsm4 / 8) + 1 */
#endif

#endif

/*************************************************************************/
/*      Disk Parameter Blocks                                            */
/*************************************************************************/

/* The following dpb definitions express the intent of the writer,       */
/* unfortunately, due to a compiler bug, these lines cannot be used.     */
/* Therefore, the obscure code following them has been inserted.         */
/*sw With release 1.2, the structure init bug disappeared, so... */
/*************    spt, bsh, blm, exm, jnk,   dsm,  drm,  al0, al1, cks, off */

struct dpb dpb0 = { 26,   3,   7,   0,   0,   242,   63,    0,   0,  16,   2};

#if ! CTLTYPE
struct dpb dpb2 = { 32,   5,  31,   1,   0,  3288, 1023,    0,  0, 256,   4};
#endif
```

```
#if   MEMDSK
struct dpb dpb3 = { 32,   4,  15,   0,   0,   191,   63,    0,   0,  0,   0};
#endif

/************************************************************************/
/* Sector Translate Table for Floppy Disks                              */
/************************************************************************/

BYTE    xlt[26] = {  1,  7, 13, 19, 25,  5, 11, 17, 23,  3,  9, 15, 21,
                     2,  8, 14, 20, 26,  6, 12, 18, 24,  4, 10, 16, 22 };


/************************************************************************/
/* Disk Parameter Headers                                               */
/*                                                                      */
/* Four disks are defined : dsk a: diskno=0, (Motorola's #fd04)         */
/* if CTLTYPE = 0         : dsk b: diskno=1, (Motorola's #fd05)         */
/*                        : dsk c: diskno=2, (Motorola's #hd00)         */
/*                        : dsk d: diskno=3, (Motorola's #hd01)         */
/*                                                                      */
/* Two disks are defined  : dsk a: diskno=0, (Motorola's #fd00)         */
/* if CTLTYPE = 1         : dsk b: diskno=1, (Motorola's #fd01)         */
/*                                                                      */
/************************************************************************/

#if ! LOADER

/* Disk Parameter Headers */
struct dph dphtab[] =

                { {&xlt, 0, 0, 0, &dirbuf, &dpb0, &csv0, &alv0}, /*dsk a*/
                  {&xlt, 0, 0, 0, &dirbuf, &dpb0, &csv1, &alv1}, /*dsk b*/
#if ! CTLTYPE
                  {  0L, 0, 0, 0, &dirbuf, &dpb2, &csv2, &alv2}, /*dsk c*/
                  {  0L, 0, 0, 0, &dirbuf, &dpb2, &csv3, &alv3}, /*dsk d*/
#endif

#if   MEMDSK
                  {  0L, 0, 0, 0, &dirbuf, &dpb3, &csv4, &alv4}  /*dsk e*/
                };
#endif

#else

#if ! CTLTYPE
struct dph dphtab[4] =
#else
struct dph dphtab[2] =
#endif
                { {&xlt, 0, 0, 0, &dirbuf, &dpb0,    0L,    0L}, /*dsk a*/
                  {&xlt, 0, 0, 0, &dirbuf, &dpb0,    0L,    0L}, /*dsk b*/
#if ! CTLTYPE
                  {  0L, 0, 0, 0, &dirbuf, &dpb2,    0L,    0L}, /*dsk c*/
                  {  0L, 0, 0, 0, &dirbuf, &dpb2,    0L,    0L}, /*dsk d*/
```

```
#endif
                    };
#endif

/****************************************************************************/
/*      Memory Region Table                                                 */
/****************************************************************************/
```


Listing D-1. (continued)

```
struct mrt {    WORD count;
                LONG tpalow;
                LONG tpalen;
           }
        memtab;                         /* Initialized in BIOSA.S  */


#if  MEMDSK
BYTE *memdsk;                           /* Initialized in BIOSA.S  */
#endif

#if ! LOADER

/****************************************************************************/
/*      IOBYTE                                                              */
/****************************************************************************/

WORD iobyte;    /* The I/O Byte is defined, but not used */

#endif


/****************************************************************************/
/*      Currently Selected Disk Stuff                                       */
/****************************************************************************/

WORD settrk, setsec, setdsk;       /* Currently set track, sector, disk */
BYTE *setdma;                      /* Currently set dma address */



/****************************************************************************/
/*      Track Buffering Definitions and Variables                           */
/****************************************************************************/

#if ! LOADER

#define  NUMTB3 /* Number of track buffers -- must be at least 3       */
                /* for the algorithms in this BIOS to work properly  */

/* Define the track buffer structure */
```

```
struct   tbstr {
                struct  tbstr  *nextbuf;        /* form linked list for LRU        */
                        BYTE   buf[32*128];     /* big enough for 1/4 hd trk       */
                        WORD   dsk;             /* disk for this buffer            */
                        WORD   trk;             /* track for this buffer           */
                        BYTE   valid;           /* buffer valid flag               */
                        BYTE   dirty;           /* true if a BIOS write has        */
                                                /* put data in this buffer,        */
                                                /* but the buffer hasn't been      */
                                                /* flushed yet.                    */
};

struct tbstr *firstbuf;         /* head of linked list of track buffers */
struct tbstr *lastbuf;  /* tail of ditto */

struct tbstr tbuf[NUMTB];/* array of track buffers */

#else

/* the loader bios uses only 1 track buffer */

BYTE buf1trk[32*128];   /* big enough for 1/4 hd trk */
BYTE bufvalid;
WORD buftrk;

#endif


/************************************************************************/
/*      Disk I/O Packets for the UDC and other Disk I/O Variables       */
/************************************************************************/

/* Home disk packet */

struct hmpkst {
                BYTE    a1;
                BYTE    a2;
                BYTE    a3;
                BYTE    dskno;
                BYTE    com1;
                BYTE    com2;
                BYTE    a6;
                BYTE    a7;
           }
        hmpack = { 2,0, 7,0, 0,0, 3,0 };  /*sw Init by bytes now... */
/*      hmpack = { 512, 1792, 0,  768 };*/ /* kludge init by words */


/* Read/write disk packet */

struct rwpkst {
                BYTE    stxchr;
                BYTE    pktid;
                BYTE    pktsize;
```

```
                    BYTE    dskno;
                    BYTE    chcmd;
                    BYTE    devcmd;
                    WORD  numblks;
                    WORD  blksize;
                    LONG   iobf;
                    WORD  cksum;
                    LONG   lsect;
                    BYTE    etxchr;
                    BYTE    rwpad;
               };
 struct rwpkst rwpack = { 2,0, 21,0,  16,1, 13, 256, 0L,   0, 0L,   3,0 };
/*struct rwpkst rwpack = { 512, 5376, 4097, 13, 256, 0, 0, 0, 0, 0, 768 };*/

#if ! LOADER

/* format disk packet */

struct fmtpkst {
                    BYTE    fmtstx;
                    BYTE    fmtid;
                    BYTE    fmtsize;
                    BYTE    fmtdskno;
                    BYTE    fmtchcmd;
                    BYTE    fmtdvcmd;
                    BYTE    fmtetx;
                    BYTE    fmtpad;
               };

/*struct fmtpkst fmtpack = { 512, 1792, 0x4002, 0x0300 };*/
 struct fmtpkst fmtpack = { 2,0,  7,0,   64,2,    3,0 };

#endif

/************************************************************************/
/*         Define the number of disks supported and other disk stuff            */
/************************************************************************/

#if ! CTLTYPE
#define NUMDSKS 4                          /* number of disks defined */
#else
#define NUMDSKS 2
#endif
#if MEMDSK
#define NUMDSKS 5
#endif

#define MAXDSK  (NUMDSKS-1)            /* maximum disk number    */

#if ! CTLTYPE
BYTE cnvdsk[NUMDSKS] = { 4, 5, 0, 1 };  /* convert CP/M dsk# to EXORmacs */
BYTE rcnvdsk[6]                 = { 2, 3, 0, 0, 0, 1 }; /* and vice versa */
#else
BYTE cnvdsk[NUMDSKS] = { 0, 1 };
```

```
BYTE rcnvdsk[2]             = { 0, 1 };
#endif

/* defines for IPC and disk states */

#define IDLE     0
#define ACTIVE 1

WORD ipcstate;  /* current IPC state */
WORD actvdsk;  /* disk number of currently active disk, if any */
LONG intcount;  /* count of interrupts needing to be processed  */

struct dskst {
                WORD  state;     /* from defines above       */
                BYTE   ready;    /* 0 => not ready */
                BYTE   change;  /* 0 => no change          */
             }
         dskstate[NUMDSKS];


/************************************************************************/
/*     Generic Serial Port I/O Procedures                              */
/************************************************************************/

/************************************************************************/
/*        Port initialization                                          */
/************************************************************************/

portinit(port)
REG BYTE *port;
{
     *(port + PORTCTRL) = PORTRSET;  /* reset the port */
     *(port + PORTCTRL) = PORTINIT;
}



/************************************************************************/
/*        Generic serial port status input status                      */
/************************************************************************/


portstat(port)
REG BYTE *port;
{
     if ( *(port + PORTSTAT) & PORTRDRF) return(0xff);  /* input ready */
        else                              return(0x00);  /* not ready        */
}


/************************************************************************/
/*        Generic serial port input                                    */
/************************************************************************/
```

```
BYTE portin(port)
REG BYTE *port;
{
	while ( ! portstat(port)) ;		/* wait for input   */
	return ( *(port + PORTRDR));		/* got some, return it	*/
}


/************************************************************************/
/*	Generic serial port output						*/
/************************************************************************/

portout(port, ch)

REG BYTE *port;
REG BYTE ch;
{
	while ( ! (*(port + PORTSTAT) & PORTTDRE) ) ; /* wait for ok to send */
	*(port + PORTTDR) = ch;			/* then send character */
}


/************************************************************************/
/*	Error procedure for BIOS						*/
/************************************************************************/

#if ! LOADER

bioserr(errmsg)
REG BYTE *errmsg;
{
	printstr("\n\rBIOS ERROR -- ");
	printstr(errmsg);
	printstr(".\n\r");
}

printstr(s)	/* used by bioserr */
REG BYTE *s;
{
	while (*s) {portout(PORT1,*s); s += 1; };
}

#else

bioserr() /* minimal error procedure for loader BIOS */
{
		l : goto l;
}

#endif

/************************************************************************/
/*	Disk I/O Procedures							*/
/************************************************************************/
```

```
EXTERN dskia();                /* external interrupt handler -- calls dskic     */
EXTERN setimask();      /* use to set interrupt mask -- returns old mask */

dskic()
{
        /* Disk Interrupt Handler -- C Language Portion */

        REG BYTE workbyte;
        BYTE    stpkt[STPKTSZ];

        workbyte = (DSKIPC + ACKFMIPC)->byte;
        if ( (workbyte == ACK) || (workbyte == NAK) )
        {
                if ( ipcstate == ACTIVE ) intcount += 1;
                else (DSKIPC + ACKFMIPC)->byte = 0;     /* ??? */
        }

        workbyte = (DSKIPC + MSGFMIPC)->byte;
        if ( workbyte & 0x80 )
        {
                getstpkt(stpkt);

                if ( stpkt[PKTID] == 0xFF )
                {
                        /* unsolicited */

                        unsolst(stpkt);
                        sendack();
                }
                else
                {
                        /* solicited */

                        if ( ipcstate == ACTIVE ) intcount += 1;
                        else sendack();
                }
        }


} /* end of dskic */

/************************************************************************/
/*      Read status packet from IPC                                     */
/************************************************************************/

getstpkt(stpktp)
REG BYTE *stpktp;
{
        REG BYTE *p, *q;
        REG WORD i;

        p = stpktp;
        q = (DSKIPC + PKTFMIPC);
```

```
        for ( i = STPKTSZ; i; i -= 1 )
        {
                *p = *q;
                 p += 1;
                 q += 2;
        }
}

/************************************************************************/
/*      Handle Unsolicited Status from IPC                              */
/************************************************************************/

unsolst(stpktp)
REG BYTE *stpktp;

{
        REG WORD dev;
        REG WORD ready;
        REG struct dskst *dsp;

        dev = rcnvdsk[ (stpktp+PKTDEV)->byte ];
        ready = ((stpktp+PKTSTPRM)->byte & 0x80) == 0x0;
        dsp = & dskstate[dev];
        if (  ( ready  && !(dsp->ready) ) ||
              (!ready) &&  (dsp->ready)     ) dsp->change = 1;
        dsp->ready = ready;
#if ! LOADER
        if ( ! ready ) setinvld(dev); /* Disk is not ready, mark buffers */
#endif
}


#if ! LOADER

/************************************************************************/
/*      Mark all buffers for a disk as not valid                        */
/************************************************************************/

setinvld(dsk)
REG WORD dsk;
{
        REG struct tbstr *tbp;

        tbp = firstbuf;
        while ( tbp )
        {
                if ( tbp->dsk == dsk ) tbp->valid = 0;
                tbp = tbp->nextbuf;
        }
}

#endif
```

```
/************************************************************************/
/*      Wait for an ACK from the IPC                                     */
/************************************************************************/

waitack()
{
        REG WORD imsave;
        REG BYTE work;

        while (1)
        {
                while ( ! intcount ) ; /* wait */

                imsave = setimask(7);
                intcount -= 1;
                work = (DSKIPC + ACKFMIPC)->byte;
                if ( (work == ACK) || (work == NAK) )
                {
                        (DSKIPC + ACKFMIPC)->byte = 0;
                        setimask(imsave);
                        return(work == ACK);
                }
                setimask(imsave);
        }
}


/************************************************************************/
/*      Acknowledge a message from the IPC                               */
/************************************************************************/

sendack()
{
        (DSKIPC + MSGFMIPC)->byte = 0;          /* clear message flag */
        (DSKIPC + ACKTOIPC)->byte = ACK;        /* send ACK   */
        (DSKIPC + INTTOIPC)->byte = 0;/* interrupt IPC      */
}


/************************************************************************/
/*      Send a packet to the IPC                                         */
/************************************************************************/

sendpkt(pktadr, pktsize)
REG BYTE *pktadr;
REG WORD  pktsize;
{
        REG BYTE *iopackp;
        REG WORD  imsave;

        while ( (DSKIPC+MSGTOIPC)->byte ); /* wait til ready */
        (DSKIPC+ACKFMIPC)->byte = 0;
        (DSKIPC+MSGFMIPC)->byte = 0;
```

```
        iopackp = (DSKIPC+PKTTOIPC);
        do {*iopackp = *pktadr++; iopackp += 2; pktsize -= 1;} while(pktsize);
        (DSKIPC+MSGTOIPC)->byte = 0x80;
        imsave = setimask(7);
        dskstate[actvdsk].state = ACTIVE;
        ipcstate = ACTIVE;
        intcount = 0L;
        (DSKIPC+INTTOIPC)->byte = 0;
        setimask(imsave);
        waitack();
}

/************************************************************************/
/*      Wait for a Disk Operation to Finish                             */
/************************************************************************/
WORD dskwait(dsk, stcom, stval)
REG WORD dsk;
BYTE    stcom;
WORD    stval;
{
        REG WORD imsave;
        BYTE stpkt[STPKTSZ];

        imsave = setimask(7);
        while ( (! intcount) &&
                dskstate[dsk].ready && (! dskstate[dsk].change) )
        {
                setimask(imsave); imsave = setimask(7);
        }

        if ( intcount )
        {
                intcount -= 1;
                if ( ( (DSKIPC + MSGFMIPC)->byte & 0x80 ) == 0x80 )
                {
                        getstpkt(stpkt);
                        setimask(imsave);
                        if ( (stpkt[PKTSTCOM] == stcom) &&
                           ( (stpkt+PKTSTVAL)->word == stval ) ) return (1);
                        else                                     return (0);
                }
        }
        setimask(imsave);
        return(0);
}

/************************************************************************/
/*      Do a Disk Read or Write                                         */
/************************************************************************/

dskxfer(dsk, trk, bufp, cmd)
REG WORD  dsk, trk, cmd;
REG BYTE *bufp;
{
```

```
        /* build packet */

        REG WORD sectcnt;
        REG WORD result;

#if CTLTYPE
        LONG bytecnt;   /* only needed for FDC */
        WORD cheksum;
#endif

        rwpack.dskno = cnvdsk[dsk];
        rwpack.iobf  = bufp;
        sectcnt = (dphtab[dsk].dpbp)->spt;
        rwpack.lsect = trk * (sectcnt >> 1);
        rwpack.chcmd = cmd;
        rwpack.numblks = (sectcnt >> 1);

#if CTLTYPE
      cheksum = 0;              /* FDC needs checksum */
        bytecnt = ((LONG)sectcnt) << 7;
        while ( bytecnt-- ) cheksum += (~(*bufp++)) & 0xff;
        rwpack.cksum = cheksum;
#endif

        actvdsk = dsk;
        dskstate[dsk].change = 0;
        sendpkt(&rwpack, 21);
        result = dskwait(dsk, 0x70, 0x0);
        sendack();
        dskstate[dsk].state = IDLE;
        ipcstate = IDLE;
        return(result);
}


#if ! LOADER

/************************************************************************/
/*      Write one disk buffer                                           */
/************************************************************************/

flush1(tbp)
struct tbstr *tbp;
{
        REG WORD ok;

        if ( tbp->valid && tbp->dirty )
                ok = dskxfer(tbp->dsk, tbp->trk, tbp->buf, DSKWRITE);
        else ok = 1;

        tbp->dirty = 0;         /* even if error, mark not dirty */
        tbp->valid &= ok;       /* otherwise system has trouble  */
                                /* continuing.                   */
        return(ok);
```

```
}

/************************************************************************/
/*      Write all disk buffers                                          */
/************************************************************************/

flush()
{
        REG struct tbstr *tbp;
        REG WORD ok;

        ok = 1;
        tbp = firstbuf;
        while (tbp)
        {
                if ( ! flush1(tbp) ) ok = 0;
                tbp = tbp->nextbuf;
        }
        return(ok);
}



/************************************************************************/
/*      Fill the indicated disk buffer with the current track and sector */
/************************************************************************/

fill(tbp)
REG struct tbstr *tbp;
{
        REG WORD ok;

        if ( tbp->valid && tbp->dirty ) ok = flush1(tbp);
        else ok = 1;

        if (ok) ok = dskxfer(setdsk, settrk, tbp->buf, DSKREAD);

        tbp->valid = ok;
        tbp->dirty = 0;
        tbp->trk = settrk;
        tbp->dsk = setdsk;

        return(ok);
}


/************************************************************************/
/*      Return the address of a track buffer structure containing the   */
/*      currently set track of the currently set disk.                  */
/************************************************************************/

struct tbstr *gettrk()
{
        REG struct tbstr *tbp;
```

```
    REG struct tbstr *ltbp;
    REG struct tbstr *mtbp;
    REG WORD imsave;

    /* Check for disk on-line -- if not, return error */

    imsave = setimask(7);
    if ( ! dskstate[setdsk].ready )
    {
        setimask(imsave);
        tbp = 0L;
        return (tbp);
    }

    /* Search through buffers to see if the required stuff  */
    /* is already in a buffer                                */

     tbp = firstbuf;
    ltbp = 0;
    mtbp = 0;
    while (tbp)
    {
        if ( (tbp->valid) && (tbp->dsk == setdsk)
            && (tbp->trk == settrk) )
        {
            if (ltbp)  /* found it -- rearrange LRU links */
            {
                ltbp->nextbuf = tbp->nextbuf;
                tbp->nextbuf  = firstbuf;
                firstbuf      = tbp;
            }
            setimask(imsave);
            return ( tbp );
        }
        else
        {
            mtbp = ltbp;        /* move along to next buffer */
            ltbp = tbp;
            tbp  = tbp->nextbuf;
        }
    }

    /* The stuff we need is not in a buffer, we must make a buffer */
    /* available, and fill it with the desired track              */

    if (mtbp) mtbp->nextbuf = 0;        /* detach lru buffer */
    ltbp->nextbuf = firstbuf;
    firstbuf = ltbp;
    setimask(imsave);
    if (flush1(ltbp) && fill(ltbp)) mtbp = ltbp;   /* success */
    else                            mtbp = 0L ;    /* failure */
    return (mtbp);
}
```

```
/************************************************************************/
/*      Bios READ Function -- read one sector                           */
/************************************************************************/

read()
{
        REG BYTE        *p;
        REG BYTE        *q;
        REG WORD        i;
        REG struct tbstr *tbp;

#if     MEMDSK
   if(setdsk != MEMDSK)
   {
#endif
        tbp = gettrk();              /* locate track buffer with sector */

        if ( ! tbp ) return(1); /* failure */
```


Listing D-1. (continued)

```
        /* locate sector in buffer and copy contents to user area */

        p = (tbp->buf) + (setsec << 7);      /* multiply by shifting */
#if     MEMDSK
   }
   else
        p = memdsk + (((LONG)(settrk) << 12L) + ((LONG)setsec << 7L));
#endif
        q = setdma;
        i = 128;
        do {*q++ = *p++; i -= 1;} while (i); /* this generates good code */
        return(0);
}


/************************************************************************/
/*      BIOS WRITE Function -- write one sector                         */
/************************************************************************/

write(mode)
BYTE mode;
{
        REG BYTE        *p;
        REG BYTE        *q;
        REG WORD        i;
        REG struct tbstr *tbp;

        /* locate track buffer containing sector to be written */
#if     MEMDSK
   if(setdsk != MEMDSK)
   {
```

```
#endif
        tbp = gettrk();
        if ( ! tbp ) return (1); /* failure */

        /* locate desired sector and do copy the data from the user area */

        p = (tbp->buf) + (setsec << 7);         /* multiply by shifting */
#if     MEMDSK
    } else
    {
        p = memdsk + (((LONG)(settrk) << 12L) + ((LONG)setsec << 7L));
        q = setdma;
        i = 128;
        do {*p++ = *q++; i -= 1;} while (i); /* this generates good code */
        return(0);
     }
#endif
        q = setdma;
        i = 128;
        do {*p++ = *q++; i -= 1;} while (i); /* this generates good code */

        tbp->dirty = 1;     /* the buffer is now "dirty" */

        /* The track must be written if this is a directory write */

        if ( mode == 1 ){if ( flush1(tbp) ) return(0); else return(1);}
        else return(0);
}

#else

/**************************************************************************/
/*      Read and Write functions for the Loader BIOS                      */
/**************************************************************************/

read()
{
        REG BYTE *p;
        REG BYTE *q;
        REG WORD  i;

        if ( ( (! bufvalid) || (buftrk != settrk) ) &&
            ( ! dskxfer(setdsk, settrk, buf1trk, DSKREAD) ) ) {return(1);}
        bufvalid = 1;
        buftrk = settrk;
        p = buf1trk + (setsec << 7);
        q = setdma;
        i = 128;
        do { *q++ = *p++; i-=1; } while(i);
        return(0);
}

#endif
```

```
/************************************************************************/
/*      BIOS Sector Translate Function                                  */
/************************************************************************/

WORD sectran(s, xp)
REG WORD  s;
REG BYTE *xp;
{
        if (xp) return (WORD)xp[s];
        else    return (s+1);
}


/************************************************************************/
/*      BIOS Set Exception Vector Function                              */
/************************************************************************/

LONG setxvect(vnum, vval)
WORD vnum;
LONG vval;
{
        REG LONG  oldval;
        REG BYTE *vloc;
```


Listing D-1. (continued)

```
        vloc = ( (long)vnum ) << 2;
        oldval = vloc->lword;
        vloc->lword = vval;

        return(oldval);

}


/************************************************************************/
/*      BIOS Select Disk Function                                       */
/************************************************************************/

LONG slctdsk(dsk, logged)
REG BYTE dsk;
  BYTE logged;
{
        REG struct dph   *dphp;
        REG BYTE      st1, st2;
        BYTE    stpkt[STPKTSZ];

        setdsk = dsk;       /* Record the selected disk number */

#if ! LOADER
```

```
        /* Special Code to disable drive C. On the EXORmacs, drive C          */
        /* is the non-removable hard disk.  Including this code lets     */
        /* you save your non-removable disk for non-CP/M use.                 */

        if ( (dsk > MAXDSK) || ( dsk == 2 ) )
        {
                printstr("\n\rBIOS ERROR -- DISK ");
                portout(PORT1, 'A'+dsk);
                printstr(" NOT SUPPORTED\n\r");
                return(0L);
        }
#endif

        dphp = &dphtab[dsk];

#if     MEMDSK
        if (setdsk == MEMDSK)
                return(dphp);
#endif

        if ( ! (logged & 0x1) )
        {

                hmpack.dskno = cnvdsk[setdsk];
                hmpack.com1  = 0x30;
                hmpack.com2  = 0x02;
                actvdsk = dsk;
                dskstate[dsk].change = 0;
                sendpkt(&hmpack, 7);
                if ( ! dskwait(dsk, 0x72, 0x0) )
                {
                        sendack();
                        ipcstate = IDLE;
                        return ( 0L );
                }
                getstpkt(stpkt);    /* determine disk type and size */
                sendack();
                ipcstate = IDLE;
                st1 = stpkt[PKTSTPRM];
                st2 = stpkt[PKTSTPRM+1];

                if ( st1 & 0x80 )  /* not ready / ready */
                {
                        dskstate[dsk].ready = 0;
                        return(0L);
                }
                else
                        dskstate[dsk].ready = 1;

                switch ( st1 & 7 )
                {
                  case 1 :           /* floppy disk      */

                                dphp->dpbp = &dpb0;
```

```
                                break;

#if ! CTLTYPE
                case 2 :        /* hard disk        */

                                dphp->dpbp = &dpb2;
                                break;
#endif

                default :       bioserr("Invalid Disk Status");
                                dphp = 0L;
                                break;
                }
        }
        return(dphp);
}


#if ! LOADER
/************************************************************/
/*                                                          */
/*      This function is included as an undocumented,      */
/*      unsupported method for EXORmacs users to format    */
/*      disks.  It is not a part of CP/M-68K proper, and   */
/*      is only included here for convenience, since the   */
/*      Motorola disk controller is somewhat complex to    */
/*      program, and the BIOS contains supporting routines. */
/*                                                          */
/************************************************************/

format(dsk)
REG WORD dsk;
{
        REG WORD retval;

        if ( ! slctdsk( (BYTE)dsk, (BYTE) 1 ) ) return;

#if     MEMDSK
        if (setdsk == MEMDSK) return;
#endif

        fmtpack.dskno = cnvdsk[setdsk];
        actvdsk = setdsk;
        dskstate[setdsk].change = 0;
        sendpkt(&fmtpack, 7);
        if ( ! dskwait(setdsk, 0x70, 0x0) ) retval = 0;
        else                                retval = 1;
        sendack();
        ipcstate = IDLE;
        return(retval);
}

#endif
```

```
/************************************************************************/
/*                                                                      */
/*      Bios initialization.  Must be done before any regular BIOS      */
/*      calls are performed.                                            */
/*                                                                      */
/************************************************************************/

biosinit()
{
        initprts();
        initdsks();
}

initprts()
{
    portinit(PORT1);
    portinit(PORT2);
}

initdsks()
{
        REG WORD i;
        REG WORD imsave;

#if ! LOADER
        for ( i = 0; i < NUMTB; ++i )
        {
                tbuf[i].valid = 0;
                tbuf[i].dirty = 0;
                if ( (i+1) < NUMTB ) tbuf[i].nextbuf = &tbuf[i+1];
          else          tbuf[i].nextbuf = 0;
        }
        firstbuf = &tbuf[0];
        lastbuf  = &tbuf[NUMTB-1];
#else
        bufvalid = 0;
#endif

        for ( i = 0; i <= MAXDSK; i += 1)
        {
                dskstate[i].state  = IDLE;
                dskstate[i].ready  = 1;
                dskstate[i].change = 0;
        }

        imsave = setimask(7);  /* turn off interrupts */
        intcount = 0;
        ipcstate = IDLE;
        setimask(imsave);      /* turn on interrupts */
}
```

```
/************************************************************************/
/*                                                                      */
/*      BIOS MAIN ENTRY -- Branch out to the various functions.         */
/*                                                                      */
/************************************************************************/

LONG cbios(d0, d1, d2)
REG WORD        d0;
REG LONG        d1, d2;
{

        switch(d0)
        {
                case 0:   biosinit();                        /* INIT          */
                          break;

#if ! LOADER
                case 1:   flush();                 /* WBOOT       */
                          initdsks();
                          wboot();
                   /* break; */
#endif
                case 2:   return(portstat(PORT1));  /* CONST       */
                   /* break; */

                case 3: return(portin(PORT1));               /* CONIN         */
                   /* break; */

                case 4: portout(PORT1, (char)d1);  /* CONOUT      */
                          break;

                case 5:   ;                                  /* LIST          */
                case 6: portout(PORT2, (char)d1);            /* PUNCH        */
                          break;

                case 7:   return(portin(PORT2));             /* READER        */
                   /* break; */

                case 8:   settrk = 0;                        /* HOME                  */
                          break;

                case 9:
                          return(slctdsk((char)d1, (char)d2)); /* SELDSK      */
                   /* break; */

                case 10: settrk = (int)d1;         /* SETTRK      */
                          break;

                case 11: setsec = ((int)d1-1);               /* SETSEC        */
                          break;

                case 12: setdma = d1;                        /* SETDMA        */
                          break;
```

```
                    case 13: return(read());              /* READ                    */
                        /* break; */
#if ! LOADER
                    case 14: return(write((char)d1));     /* WRITE          */
                        /* break; */

                    case 15: if ( *(BYTE *)(PORT2 + PORTSTAT) & PORTTDRE )
                                        return ( 0x0ff );
                                  else     return ( 0x000 );
                        /* break; */
#endif

                    case 16: return(sectran((int)d1, d2));          /* SECTRAN     */
                        /* break; */
#if ! LOADER
                    case 18: return(&memtab);                       /* GMRTA       */
                        /* break; */

                    case 19: return(iobyte);              /* GETIOB         */
                        /* break; */

                    case 20: iobyte = (int)d1;            /* SETIOB         */
                             break;

                    case 21: if (flush()) return(0L);     /* FLUSH          */
                             else           return(0xffffL);
                        /* break; */
#endif
                    case 22: return(setxvect((int)d1,d2));          /* SETXVECT    */
                        /* break; */
#if ! LOADER

                    /*********************************************************/
                    /*      This function is not part of a standard BIOS.            */
                    /*          It is included only for convenience, and will        */
                    /*          not be supported in any way, nor will it             */
                    /*          necessarily be included in future versions of        */
                    /*          CP/M-68K                                             */
                    /*********************************************************/
                    case 63: return( ! format((int)d1) ); /* Disk Formatter */
                        /* break; */
#endif

                    default: return(0L);
                             break;

          } /* end switch */


} /* END OF BIOS */


/* End of C Bios */
```

Listing D-2. EXORmacs BIOSTYPS.H File

```
/*  @(#)biostyps.h  1.1     */
/*************************************************/
/*                                               */
/*      Portable type definitions for use        */
/*      with the C BIOS according to             */
/*      CP/M-68K (tm) standard usage.            */
/*                                               */
/*************************************************/

#define LONG    long
#define ULONG   unsigned long
#define WORD    short int
#define UWORD   unsigned short
#define BYTE    char
#define UBYTE   unsigned char
#define VOID

#define REG     register
#define LOCAL   auto
#define MLOCAL          static
#define GLOBAL          extern
#define EXTERN          extern

/***********************************************/
```

Listing D-3. EXORmacs NORMBIOS.H File

```
#define LOADER  0
#define CTLTYPE 0
#define MEMDSK          4
```

Listing D-4. EXORmacs LOADBIOS.H File

```
#define LOADER  1
#define CTLTYPE 0
#define MEMDSK  0
```

Listing D-5. EXORmacs BIOSA.S File

```
        .text
*
*       Global Code addresses
*
        .globl  _init
        .globl  _biosinit
```

```
        .globl  _flush
        .globl  _wboot
        .globl  _cbios
        .globl  _dskia
        .globl  _dskic
        .globl  _setimask
        .globl  _ccp
        .globl  cpm             *       Lowest addr of CP/M
        .globl  _end            *       Highest addr of CP/M
*
*       Global data addresses
*
        .globl  _memtab         *       memory region table
        .globl  _dpb3           *       RAM disk dpb address
        .globl  _memdsk         *       -> First memory disk location
*
*       Vector Addresses
*
dskint: .equ    $3fc            *       UDC Interrupt vector
trap3:  .equ    $8c             *       Trap 3 vector
buserr: .equ    $8              *       Bus error vector
*
*
_init:  lea     entry,a0
        move.l  a0,trap3
        lea     _dskia,a0
        move.l  a0,dskint
*
*       Auto-Size TPA
*
        lea     _memtab,a0      *       a0 -> Memory region table
        move.w  #1,(a0)+        *       1 region
        move.l  #$400,(a0)+     *       TPA starts at 400
        move.l  #cpm-$408,(a0)+ *       Ends where CP/M begins
*
*       Auto-Size RAM disk
*
        move.l  buserr,-(sp)    *       Push bus err vector
        lea     _end,a0         *       a0 -> Last location in CP/M
        add.l   #cpm,a0         *       Linker doesn't reloc this!!
        move.l  a0,_memdsk      *       -> first location in RAM disk
        move.l  #quit,buserr    *       set up vector -> ourselves
loop:
        tst.w   (a0)+           *       Find
        bra     loop            *               End of memory
quit:
        add.l   #14,a7          *    Clear buserr gorp
        move.l  (a7)+,buserr    *       Pop buserr vector
        sub.l   #_end,a0        *       a0 = # bytes in RAM disk
        sub.l   #cpm,a0         *       Relocation bug
        move.l  a0,d0           *       Into D reg for shift
        move.l  #11,d1          *       Load shift count
        lsr.l   d1,d0           *       Divide by 2048
        move.w  d0,_dpb3+6      *       Load DRM field of dpb
```

```
        move    #$2000,sr
        jsr     _biosinit
        clr.l   d0
        rts
*
_wboot: clr.l   d0
        jmp     _ccp
*
entry:  move.l  d2,-(a7)
        move.l  d1,-(a7)
        move.w  d0,-(a7)
        jsr     _cbios
        add     #10,a7
        rte
*
_dskia: link    a6,#0
        movem.l d0-d7/a0-a5,-(a7)
        jsr     _dskic
        movem.l (a7)+,d0-d7/a0-a5
        unlk    a6
        rte
*
_setimask: move sr,d0
        lsr     #8,d0
        and.l   #7,d0
        move    sr,d1
        ror.w   #8,d1
        and.w   #$fff8,d1
        add.w   4(a7),d1
        ror.w   #8,d1
        move    d1,sr
        rts
        .data
        .globl  BIOSDATA
BIOSDATA: .dc.l         0,0

        .end
```

Listing D-6. EXORmacs LDBIOS.S File

```
        .text
        .globl  _bios
        .globl  _biosinit
        .globl  _cbios
        .globl  _dskia
        .globl  _dskic
        .globl  _setimask
*
*
*
*
_bios:  link    a6,#0
        move.l  d2,-(a7)
```

```
        move.l  d1,-(a7)
        move.w  d0,-(a7)
        move    #$2000,sr
        lea     _dskia,a0
        move.l  a0,$3fc
        jsr     _cbios
        unlk    a6
        rts
*
_dskia: link    a6,#0
        movem.ld0-d7/a0-a5,-(a7)
        jsr     _dskic
        movem.l(a7)+,d0-d7/a0-a5
        unlk    a6
        rte
*
_setimask: move sr,d0
        lsr     #8,d0
        and.l   #7,d0
        move    sr,d1
        ror.w   #8,d1
        and.w   #$fff8,d1
        add.w   4(a7),d1
        ror.w   #8,d1
        move    d1,sr
        rts
*
        .end
```

Listing D-7. EXORmacs BOOTER.S File

```
************************************
*  Information to go on the 256 byte     *
*    boot sector of an ExorMacs           *
************************************

        .text
        .dc.l   $4000                   * starting stack pointer
        .dc.l   start                   * starting program counter
        dc.w    1                       * garbage
        .dc.w   1                       * length of SAT
        .dc.l   2                       * secondary directory start
        .dc.l   0                       * primary directory PSN list start
        .dc.l   0                       * start of boot loader
        .dc.w   26                      * length of boot loader
        .dc.l   $0                      * boot execution address
        .dc.l   $0                      * boot load address
        .dc.b   '9/30'                  * generation date
        .dc.b   'CP/M-68K of 9/30/82 '  * volume descriptor
        .dc.b   '0020'                  * version/revision
        .dc.w   $0a484                  * checksum (god help us)
        .dc.l   $0f1e2d3c               * diagnostic test pattern
```

```
        .dc.l   $4b5a6978
        .dc.l   $8796a5b4
        .dc.l   $c3d2e1f0
        .dc.l   $0f1e2d3c               * diagnostic test pattern
        .dc.l   $4b5a6978
        .dc.l   $8796a5b4
        .dc.l   $c3d2e1f0
        .dc.l   $4f8f0f07               * diagnostic test pattern
        .dc.l   $0b0d0e06
        .dc.l   $0a0c0408
        .dc.l   $04020100
        .dc.l   00, 00, 00, 00          * diagnostic test pattern
        .dc.l   0                       * diagnostic test area directory
        .dc.l   0                       * start of dump area
        .dc.w   0                       * length of dump area
        .dc.l   0                       * start of sector lockout table
        .dc.w   0                       * length of sector lockout table
        .dc.l   0,0,0,0,0,0,0           * unused, reserved
        .dc.l   0,0,0,0,0,0
        .dc.l   0,0,0,0,0,0,0
        .dc.l   0,0,0,0,0,0
        .dc.b   'EXORMACS'              * let's hear it for Motorola
*
*       end of volume id
*
*       begin boot info proper
*
        .ds.b   $300            * skip over exception vectors
        .even
start:  move    #$2700,sr
        move.l  #$8,a0
move.w #253,d0
exlp:   move.l  #expdef,(a0)+
        dbf     d0,exlp
        jmp     grunt
expdef: rte
grunt:  move    #$2000,sr
        .end
```

Appendix E
Putboot Utilility Assembly Language Source

```
                    ********************************************************
                    *                                                      *
                    *     Program to Write Boot Tracks for CP/M-68K (tm)   *
                    *                                                      *
                    *     Copyright Digital Research 1982                  *
                    *                                                      *
                    ********************************************************
                    *
                    *
                    *
                    prntstr =    9          BDOS Functions
                    dseldsk =    14
                    open    =    15
                    readseq =    20
                    dsetdma =    26
                    *
                    seldsk  =    9          BIOS Functions
                    settrk  =    10
                    setsec  =    11
                    isetdma =    12
                    write   =    14
                    sectran =    16
                    flush   =    21
                    *
                    bufcnt  =    $80
                    bufsize =    $80*bufcnt
                    *
00000000                    .text
                    *
00000000 4E560000 start:  link    a6,#0
00000004 206E0008         move.l  8(a6),a0           base page address
00000008 43E8005C         lea     $5c(a0),a1
0000000C 23C900004080     move.l  a1,fcb
00000012 423900004094     clr.b   hflag
00000018 D0FC0081         add     #$81,a0            first character of command tail
0000001C 0C180020 scan:   cmpi.b  #$20,(a0)+         skip over blanks
00000020 67FA             beq     scan
00000022 5388             sub.l   #1,a0
00000024 4A10     scan1:  tst.b   (a0)
00000026 670001A4         beq     erxit
0000002A 0C18002D         cmpi.b  #$2d,(a0)+         check for -H flag
0000002E 6626             bne     nohyph
00000030 0C180048         cmpi.b  #$48,(a0)+
00000034 66000196         bne     erxit
00000038 4A3900004094     tst.b   hflag
0000003E 6600018C         bne     erxit
00000042 13FC00FF00004094 move.b  #$ff,hflag
```

```
0000004A 04B90000002400004080 sub.l   #$24,fcb              change to 2nd default fcb
00000054 60C6                 bra     scan
00000056 0C100020  nohyph:    cmpi.b  #$20,(a0)
0000005A 66C8                 bne     scan1
0000005C 0C180020  scan2:     cmpi.b  #$20,(a0)+
00000060 67FA                 beq     scan2
00000062 0C200061             cmpi.b  #$61,-(a0)            get disk letter
00000066 6D04                 blt     upper                 upshift
00000068 04500020             sub     #$20,(a0)
0000006C 0C100041  upper:     cmpi.b  #$41,(a0)             compare with range A - P
00000070 6D00015A             blt     erxit
00000074 0C100050             cmpi.b  #$50,(a0)
00000078 6E000152             bgt     erxit
0000007C 1010                 move.b  (a0),d0
0000007E 4880                 ext.w   d0                    put disk letter into range 0 - 15
00000080 907C0041             sub.w   #$41,d0
00000084 33C00000408A         move.w  d0,dsk
                   *
                   *      open file to copy
                   *
0000008A 303C000F             move.w  #open,d0
0000008E 223900004080         move.l  fcb,d1
00000094 4E42                 trap    #2
00000096 0C4000FF             cmpi.w  #$00ff,d0
0000009A 660C                 bne     openok
0000009C 223C00000034         move.l  #opnfl,d1
000000A2 4EF9000001D2         jmp     erx
000000A8 207900004080 openok: move.l  fcb,a0
000000AE 42280020             clr.b   32(a0)
                   *
                   *      read
                   *
000000B2 243C00000000         move.l  #buf,d2
000000B8 42790000408E         clr.w   count
000000BE 303C001A  rloop:     move.w  #dsetdma,d0
000000C2 2202                 move.l  d2,d1
000000C4 4E42                 trap    #2
000000C6 303C0014             move.w  #readseq,d0
000000CA 223900004080         move.l  fcb,d1
000000D0 4E42                 trap    #2
000000D2 4A40                 tst.w   d0
000000D4 661A                 bne     wrtout
000000D6 D4BC00000080         add.l   #128,d2
000000DC 52790000408E         add.w   #1,count
000000E2 0C7900800000408E cmpi.w #bufcnt,count
000000EA 6E0000FE             bgt     bufoflx
000000EE 60CE                 bra     rloop
                   *
                   *      write
                   *
000000F0 303C0009 wrtout:     move.w  #seldsk,d0            select the disk
000000F4 32390000408A         move.w  dsk,d1
000000FA 4202                 clr.b   d2
000000FC 4E43                 trap    #3
```

```
000000FE 4A80              tst.l   d0                  check for select error
00000100 670000D8          beq     selerx
00000104 2040              move.l  d0,a0
00000106 2068000E          move.l  14(a0),a0           get DPB address
0000010A 33D000004084      move.w  (a0),spt            get sectors per track
00000110 33E8000E0000408C move.w  14(a0),off           get offset
00000118 427900004088      clr.w   trk                 start at trk 0
0000011E 33FC000100004086  move.w  #1,sect             start at sector 1
00000126 41F900000000      lea     buf,a0
0000012C 4A3900004094      tst.b   hflag
00000132 660C              bne     wrt1
00000134 0C50601A          cmpi.w  #$601a,(a0)
00000138 6606              bne     wrt1
0000013A D1FC0000001C      add.l   #28,a0
00000140 23C800004090      wrt1:   move.l  a0,bufp
                           *
00000146 4A790000408E      wloop:  tst.w   count
0000014C 6774              beq     exit
0000014E 323900004086      move.w  sect,d1             check for end-of-track
00000154 B27900004084      cmp.w   spt,d1
0000015A 6F1E              ble     sok
0000015C 33FC000100004086 move.w  #1,sect              advance to new track
00000164 303900004088      move.w  trk,d0
0000016A 5240              add.w   #1,d0
0000016C 33C000004088      move.w  d0,trk
00000172 B0790000408C      cmp.w   off,d0
00000178 6C78              bge     oflex
0000017A 303C000A  sok:    move.w  #settrk,d0          set the track
0000017E 323900004088      move.w  trk,d1
00000184 4E43              trap    #3
00000186 323900004086      move.w  sect,d1             set sector
0000018C 303C000B          move.w  #setsec,d0
00000190 4E43              trap    #3
00000192 303C000C          move.w  #isetdma,d0         set up dma address for write
00000196 223900004090      move.l  bufp,d1
0000019C 4E43              trap    #3
0000019E 303C000E          move.w  #write,d0           and write
000001A2 4241              clr.w   d1
000001A4 4E43              trap    #3
000001A6 4A40              tst.w   d0                  check for write error
000001A8 6638              bne     wrterx
000001AA 527900004086      add     #1,sect             increment sector number
000001B0 53790000408E      sub     #1,count
000001B6 06B90000008000004090  add.l  #128,bufp
000001C0 6084              bra     wloop
                           *
000001C2 303C0015  exit:   move.w  #flush,d0           exit location - flush bios buffers
000001C6 4E43              trap    #3
000001C8 4E5E              unlk    a6
000001CA 4E75              rts                         and exit to CCP
                           *
000001CC 223C00000000 erxit:  move.l  #erstr,d1        miscellaneous errors
000001D2 303C0009 erx:     move.w  #prntstr,d0         print error message and exit
000001D6 4E42              trap    #2
```

```
000001D8 60E8                  bra     exit
                         *
000001DA 223C00000017 selerx: move.l  #selstr,d1          disk select error
000001E0 60F0                  bra     erx
000001E2 223C00000026  wrterx: move.l  #wrtstr,d1         disk write error
000001E8 60E8                  bra     erx
000001EA 223C0000004E bufoflx: move.l #bufofl,d1          buffer overflow
000001F0 60E0                  bra     erx
000001F2 223C00000060  oflex:  move.l  #trkofl,d1
000001F8 60D8                  bra     erx
                         *
                         *
00000000                       .bss
                         *
                               .even
                         *
00000000                 buf:      .ds.b   bufsize+128
                         *
00004080                 fcb:      .ds.l   1              fcb address
00004084                 spt:      .ds.w   1              sectors per track
00004086                 sect:     .ds.w   1              current sector
00004088                 trk:      .ds.w   1              current track
0000408A                 dsk:      .ds.w   1              selected disk
0000408C                 off:      .ds.w   1              1st track of non-boot area
0000408E                 count:    .ds.w   1
00004090                 bufp:     .ds.l   1
00004094                 hflag:    .ds.b   1
                         *
00004096                           .data
00000000
                         *
00000000 496E76616C696420      erstr:  .dc.b   'Invalid Command Line',13,10,'$'
00000008 436F6D6D616E6420
00000010 4C696E650D0A24
00000017 53656C6563742045      selstr: .dc.b   'Select Error',13,10,'$'
0000001F 72726F720D0A24
00000026 5772697465204572      wrtstr: .dc.b   'Write Error',13,10,'$'
0000002E 726F720D0A24
00000034 43616E6E6F74204F      opnfl:  .dc.b   'Cannot Open Source File',13,10,'$'
0000003C 70656E20536F7572
00000044 63652046696C650D
0000004C 0A24
0000004E 427566666572204F      bufofl: .dc.b   'Buffer Overflow',13,10,'$'
00000056 766572666C6F770D
0000005E 0A24
00000060 546F6F204D756368      trkofl: .dc.b   'Too Much Data for System Tracks',13,10,'$'
00000068 204461746120666F
00000070 722053797374656D
00000078 20547261636B730D
00000080 0A24
                         *
                         *
00000082                       .end
```

Symbol Table

buf	00000000	BSS	bufcnt	00000080	ABS	bufofl	0000004E	DATA	bufoflx	000001EA	TEXT
bufp	00004090	BSS	bufsize	00004000	ABS	count	0000408E	BSS	dseldsk	0000000E	ABS
dsetdma	0000001A	ABS	dsk	0000408A	BSS	erstr	00000000	DATA	erx	000001D2	TEXT
erxit	000001CC	TEXT	exit	000001C2	TEXT	fcb	00004080	BSS	flush	00000015	ABS
hflag	00004094	BSS	isetdma	0000000C	ABS	nohyph	00000056	TEXT	off	0000408C	BSS
oflex	000001F2	TEXT	open	0000000F	ABS	openok	000000A8	TEXT	opnfl	00000034	DATA
prntstr	00000009	ABS	readseq	00000014	ABS	rloop	000000BE	TEXT	scan	0000001C	TEXT
scan1	00000024	TEXT	scan2	0000005C	TEXT	sect	00004086	BSS	sectran	00000010	ABS
seldsk	00000009	ABS	selerx	000001DA	TEXT	selstr	00000017	DATA	setsec	0000000B	ABS
settrk	0000000A	ABS	sok	0000017A	TEXT	spt	00004084	BSS	start	00000000	TEXT
trk	00004088	BSS	trkofl	00000060	DATA	upper	0000006C	TEXT	wloop	00000146	TEXT
write	0000000E	ABS	wrt1	00000140	TEXT	wrterx	000001E2	TEXT	wrtout	000000F0	TEXT
wrtstr	00000026	DATA									

End of Appendix E

Appendix F
Motorola S-Records

F.1 S-record Format

The Motorola S-record format is a method of representing binary memory images in an ASCII form. The primary use of S-records is to provide a convenient form for transporting programs between computers. Since most computers have means of reading and writing ASCII information, the format is widely applicable. The SENDC68 utility provided with CP/M-68K may be used to convert programs into S-record form.

An S-record file consists of a sequence of S-records of various types. The entire content of an S-record is ASCII. When a hexadecimal number needs to be represented in an S-record it is represented by the ASCII characters for the hexadecimal digits comprising the number. Each S-record contains five fields as follows:

Field:	S	type	length	address	data	checksum
Characters:	1	1	2	2, 4 or 6	variable	2

Figure F-1. S-record Fields

The field contents are as follows:

Table F-1. S-record Field Contents

Field	Contents
S	The. ASCII Character IS'. This signals the beginning of the S-record.
type	A digit, between 0 and 9, represented in ASCII, with the exceptions that 4 and 6 are not allowed. Type is explained in detail below.

Table F-1. (continued)

Field	Contents
length	The number of character pairs in the record, excluding the first three fields. (That is, one half the number of characters total in the address, data, and checksum fields.) This field has two hexadecimal digits, representing a one byte quantity.
address	The address at which the data portion of the record is to reside in memory. The data goes at this address and successively higher numbered addresses. The length of this field is determined by the record type.
data	The actual data to be loaded into memory, with each byte of data represented as a pair of hexadecimal digits, in ASCII.
checksum	A checksum computed over the length, address, and data fields. The checksum is computed by adding the values of all the character pairs (each character pair represents a one-byte quantity) in these fields, taking the one's complement of the result, and finally taking the least significant byte. This byte is then represented as two ASCII hexadecimal digits.

F.2 S-record Types

There are eight types of S-records. They can be divided into two categories: records containing actual data, and records used to define and delimit groups of data-containing records. Types 1, 2, and 3 are in the first category, and the rest of the types are in the second category. Each of the S-record types is described individually below.

Table F-2. S-record Types

Type	Meaning
0	This type is a header record used at the beginning of a group of S-records. The data field may contain any desired identifying information. The address field is two bytes (four S-record characters) long, and is normally zero.
1	This type of record contains normal data. The address field is two bytes long (four S-record characters).
2	Similar to Type 1, but with a 3-byte (six S-record characters) address field.
3	Similar to Type 1, but with a 4-byte (eight S- record characters) address field.
5	This record type indicates the number of Type 1, 2, and 3 records in a group of S-records. The count is placed in the address field. The data field is empty (no characters).
7	This record signals the end of a block of type 3 S-records. If desired, the address field is 4 bytes long (8 characters), and may be used to contain an address to which to pass control. The data field is empty.
8	This is similar to type 7 except that it ends a block of type 2 S-records, and its address field is 3 bytes (6 characters) long.
9	This is similar to type 7 except that it ends a block of type I S-records, and its address field is 2 bytes (4 characters) long.

S-records are produced by the SENDC68 utility program (described in the CP/M-68K Operating System Programmer's Guide).

End of Appendix F

Appendix G
CP/M-68K Error Messages

This appendix lists the error messages returned by the internal components of CP/M-68K: BDOS, BIOS, and CCP, and by the CP/M-68K system utility, PUTBOOT. The BIOS error messages listed here are specific to the EXORmacs BIOS distributed by Digital Research. BIOSes for other hardware might have different error messages which should be documented by the hardware vendor.

The error messages are listed in Table G-1 in alphabetic order with explanations and suggested user responses.

Table G-1. CP/M-68K Error Messages

Message Meaning

bad relocation information bits

CCP. This message is a result of a BDOS Program Load Function (59) error. It indicates that the file specified in the command line is not a valid executable command file, or that the file has been corrupted. Ensure that the file is a command file. The CP/M-68K Operating System Programmer's Guide describes the format of a command file. If the file has been corrupted, reassemble or recompile the source file, and relink it before you reenter the command line.

BIOS ERROR -- DISK X NOT SUPPORTED

BIOS. The disk drive indicated by the variable "X" is not supported by the BIOS. The BDOS supports a maximum of 16 drives, lettered A through P. Check the documentation provided by the manufacturer for your particular system configuration to find out which of the BDOS drives your BIOS implements. Specify the correct drive code and reenter the command line.

Table G-1. (continued)

Message	Meaning
BIOS ERROR -- Invalid Disk Status	BDOS. The disk controller returned unexpected or incomprehensible information to the BIOS. Retry the operation. If the error persists, check the hardware. If the error does not come from the hardware, it is caused by an error in the internal logic of the BIOS. Contact the place you purchased your system for assistance. You should provide the information below. 1) Indicate which version of the operating system you are using. 2) Describe your system's hardware configuration. 3) Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. if possible, you should also provide a disk with a copy of the program.
Buffer Overflow	PUTBOOT. The bootstrap file will not fit in the PUTBOOT bootstrap buffer. PUTBOOT contains an internal buffer of approximately 16K bytes into which it reads the bootstrap file. Either make the bootstrap file smaller so that it will fit into the buffer, or change the size of the PUTBOOT buffer. The PUTBOOT source code is supplied with the system distributed by DRI. Equate bufsize (located near the front of the PUTBOOT source code) to the required dimension in Hexidecimals. Reassemble and relink the source code before you reenter the PUTBOOT command line.
Cannot Open Source File	PUTBOOT. PUTBOOT cannot locate the source file. Ensure that you specify the correct drive code and filename before you reenter the PUTBOOT command line.

Table G-1. (continued)

Message	Meaning
CP/M Disk change error on drive x	BDOS. The disk in the drive indicated by the variable x is not the same disk the system logged in previously. When the disk was replaced you did not enter: a CTRL-C to log in the current disk. Therefore, when you attempted to write to, erase, or rename a file on the current disk, the BDOS set the drive status to read-only and warm booted the system. The current disk in the drive was not overwritten. The drive status was returned to read-write when the system was warm booted. Each time a disk is changed, you must type a CTRL-C to log in the new disk.
CP/M Disk file error: filename is read-only. Do you want to: Change it to read/write (C), or Abort (A)?	BDOS. You attempted to write to, erase, or rename a file whose status is read-only. Specify one of the options enclosed in parentheses. If you specify the C option, the BDOS changes the status of the file to read- write and continues the operation. The read-only protection previously assigned to the file is lost. If you specify the A option or a CTRL-C, the program terminates and CP/M-68K returns the system prompt.
CP/M Disk read error on drive x Do you want to: Abort (A), Retry (R), or Continue with bad data (C)?	BDOS. This message indicates a hardware error. Specify one of the options enclosed in parentheses. Each option is described below.

Option	Action
A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt. (Meaning continued on next page.)

Table G-1. (continued)

Message	Meaning

CP/M Disk read error on drive x (continued)

Option	Action
R	Retries operation. If the retry fails, the system reprompts with the option message.
C	Ignores error and continues program execution. Be careful if you use this option. Program execution should not be continued for some types of programs. For example, if you are updating a data base and receive this error but continue program execution, you can corrupt the index fields and the entire data base. For other programs, continuing program execution is recommended. For example, when you transfer a long text file and receive an error because one sector is bad, you can continue transferring the file. After the file is transferred, review the file, and add the data that was not transferred due to the bad sector.

CP/M Disk write error on drive x
Do you want to: Abort (A), Retry (R),
or Continue with bad data (C)?

BDOS. This message indicates a hardware error. Specify one of the options enclosed in parentheses. Each option is described below.

Option	Action
A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.
R	Retries operation. If the retry fails, the system reprompts with the option message (Meaning continued on next page.)

Table G-1. (continued)

Message	Meaning

CP/M Disk write error on drive x (continued)

Option	Action
C	Ignores error and continues program execution. Be careful if you use this option. Program execution should not be continued for some types of programs. For example, if you are updating a data base and receive this error but continue program execution, you can corrupt the index fields and the entire data base, For other programs, continuing program execution is recommended. For example, when you transfer a long text file and receive an error because one sector is bad, you can continue transferring the file. After the file is transferred, review the file, and add the data that was not transferred due to the bad sector.

CP/M Disk select error on drive x
Do you want to: Abort (A), Retry (R)

BDOS. There is no disk in the drive or the disk is not inserted correctly. Ensure that the disk is securely inserted in the drive. If you enter the R option, the system retries the operation. If you enter the A option or CTRL-C the program terminates and CPM-68K returns the system prompt.

CP/M Disk select error on drive x

BDOS. The disk selected in the command line is outside the range A through P. CP/M-68K can support up to 16 drives, lettered A through P. Check the documentation provided by the manufacturer to find out which drives your particular system configuration supports. Specify the correct drive code and reenter the command line.

Table G-1. (continued)

Message	Meaning
File already exists	CCP. This error occurs during a REN command. The name specified in the command line as the new filename already exists. Use the ERA command to delete the existing file if you wish to replace it with the new file. If not, select another filename and reenter the REN command line.
insufficient memory or bad file header	CCP. This error could result from one of three causes: 1) The file is not a valid executable command file. Ensure that you are requesting the correct file. This error can occur when you enter the filename before you enter the command for a utility. Check the appropriate section of the CP/M-68K Operating System Programmer's Guide or the CP/M-68K Operating System User's Guide for the correct command syntax before you reenter the command line. If you are trying to run a program when this error occurs, the program f i le may have been corrupted. Reassemble or recompile the source file and relink it before you reenter the command line. 2) The program is too large for the available memory. Add more memory boards to the system configuration, or rewrite the program to use less memory. 3) The program is linked to an absolute location in memory that cannot be used. The program must be made relocatable, or linked to a usable memory location. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the memory space that is available for loading programs.

Table G-1. (continued)

Message	Meaning
Invalid Command Line	PUTBOOT. Either the command line syntax is incorrect, or you have selected a disk drive code outside the range A through P. Refer to the section in this manual on the PUTBOOT utility for a full description of the command line syntax. The CP/M-68K BDOS supports 16 drives, lettered A through P. The BIOS may or may not support all 16 drives. Check the documentation provided by the manufacturer for your particular system configuration to find out which drives your BIOS supports. Specify a valid drive code before reentering the PUTBOOT command line.
No file	CCP. The filename specified in the command line does not exist. Ensure that you use the correct filename and reenter the command line.
No wildcard filenames	CCP. The command specified in the command line does not accept wildcards in file specifications. Retype the command line using a specific filename.
Program Load Error	CCP. This message indicates an undefined failure of the BDOS Program Load Function (59). Reboot the system and try again. If the error persists, then it is caused by an error in the internal logic of the BDOS. Contact the place you purchased your system for assistance. You should provide the information below. 1) Indicate which version of the operating system you are using. 2) Describe your system's hardware configuration. (Meaning continued on next page.)

Table G-1. (continued)

Message	Meaning
	3)Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.
read error on program load	
	CCP. This message indicates a premature end-of-file. The file is smaller than the header information indicates. Either the file header has been corrupted or the file was only partially written. Reassemble or recompile the source file, and relink it before you reenter the command line.
Select Error	
	PUTBOOT. This error is returned from the BIOS select disk function. The drive specified in the command line is either not supported by the BIOS, or is not physically accessible. Check the documentation provided by the manufacturer to find out which drives your BIOS supports. This error is also returned if a BIOS supported drive is not supported by your system configuration. Specify a valid drive and reenter the PUTBOOT command line.
SUB file not found	
	CCP. The file requested either does not exist, or does not have a filetype of SUB. Ensure that you are requesting the correct file. Refer to the section on SUBMIT in the CP/M-68K Operating System User's Guide for information on creating and using submit files.
Syntax: REN newfile=oldfile	
	CCP. The syntax of the REN command line is incorrect. The correct syntax is given in the error message. Enter the REN command followed by a space, then the new filename, followed immediately by an equals sign (=) and the name of the file you want to rename.

Table G-1. (continued)

Message	Meaning
Too many arguments: argument?	CCP. The command line contains too many arguments. The extraneous arguments are indicated by the variable argument. Refer to the CP/M-68K Operating System User's Guide for the correct syntax for the command. Specify only as many arguments as the command syntax allows and reenter the command line. Use a second command line for the remaining arguments, if appropriate.
Too Much Data for System Tracks	PUTBOOT. The bootstrap file is too large for the space reserved for it on the disk. Either make the bootstrap file smaller, or redefine the number of tracks reserved on the disk for the file. The number of tracks reserved for the bootstrap file is controlled by the OFF parameter in the disk parameter block in the BIOS. This error can also be caused by a bootstrap file that contains a symbol table and relocation bits. To find out if the bootstrap program will fit on the system tracks without the symbol table and relocation bits, use the SIZE68 Utility to display the amount of space the bootstrap program occupies. The first and second items returned by the SIZE68 Utility are the amount of space occupied by the text and data, respectively. The third item returned is the amount of space occupied by the BSS. The sum of the first two items, or the total minus the third item, will give you the amount of space required for the bootstrap program on the system tracks. Compare the amount of space your bootstrap program requires to the amount of space allocated by the OFF parameter. Because the symbol table and relocation bits are at the end of the file, the bootstrap program may have been entirely written to the system tracks and you can ignore this message. Or, you can run RELOC on the bootstrap file to remove the symbol table and relocation bits from the bootstrap file and reenter the PUTBOOT command line.

Table G-1. (continued)

Message	Meaning
User # range is [0-15]	CCP. The user number specified in the command line is not supported by the BIOS. The valid range is enclosed in the square brackets in the error message. Specify a user number between 0 and 15 (decimal) when you reenter the command line.
Write Error	PUTBOOT. Either the disk to which PUTBOOT is writing is damaged or there is a hardware error. Insert a new disk and reenter the PUTBOOT command line. If the error persists, check for a hardware error.

End of Appendix G

Index