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. Appendices 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

1 System Overview
   1.1 Introduction ............................................. 1
   1.2 CP/M-68K Organization ................................. 3
   1.3 Memory Layout ........................................... 3
   1.4 Console Command Processor ............................ 4
   1.5 Basic Disk Operating System (BDOS) .................. 5
   1.6 Basic I/O System (BIOS) ............................... 5
   1.7 I/O Devices .............................................. 5
      1.7.1 Character Devices .................................. 5
      1.7.2 Character Devices .................................. 5
   1.8 System Generation and Cold Start Operation ........ 6

2 System Generation
   2.1 Overview .................................................. 7
   2.2 Creating CPM.SYS ........................................ 7
   2.3 Relocating Utilities .................................... 8

3 Bootstrap Procedures
   3.1 Bootstrapping Overview ................................ 9
   3.2 Creating the Cold Boot Loader ....................... 10
      3.2.1 Writing a Loader BIOS .......................... 10
      3.2.2 Building CPMLDR.SYS ............................ 11

4 BIOS Functions
   4.1 Introduction ............................................ 13
5 Creating a BIOS

5.1 Overview ....................................................... 39
5.2 Disk Definition Tables ................................. 39
  5.2.1 Disk Parameter Header ................................. 40
  5.2.2 Sector Translate Table ................................. 41
  5.2.3 Disk Parameter Block ................................. 42
5.3 Disk Blocking Guide ................................. 45
  5.3.1 A Simple Approach ................................. 46
  5.3.2 Some Refinements ................................. 46
  5.3.3 Track Buffering ................................. 47
  5.3.4 LRU Replacement ................................. 47
  5.3.5 The New Block Flag ................................. 48

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

6.1 Overview ....................................................... 49
6.2 Booting on an EXORmacs ................................. 49
6.3 Bringing up CP/M-68K Using S-record Files .......... 50

7 Cold Boot Automatic Command Execution

7.1 Overview ....................................................... 51
7.2 Setting up Cold Boot Automatic Command Execution ... 51

8 The PUTBOOT Utility

8.1 PUTBOOT Operation ........................................ 53
8.2 Invoking PUTBOOT ........................................ 53
Appendixes

A  Contents of Distribution Disks  ..........  55

B  Sample BIOS Written in Assembly Language  ..........  59

C  Sample Loader BIOS Written in Assembly Language  ..........  67

D  EXORMacs BIOS Written in C  ...........  73

E  PUTBOOT Utility Assembly Language Source  ...........  101

F  The Motorola S-record Format  ...........  107
   F.1  S-record Format  ...........  107
   F.2  S-record Types  ...........  108

G  CP/M-68K Error Messages  ...........  109
Tables and Figures

Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1.</td>
<td>CP/M-68K Terms</td>
<td>1</td>
</tr>
<tr>
<td>4-1.</td>
<td>BIOS Register Usage</td>
<td>14</td>
</tr>
<tr>
<td>4-2.</td>
<td>BIOS Functions</td>
<td>14</td>
</tr>
<tr>
<td>4-3.</td>
<td>CP/M-68K Logical Device Characteristics</td>
<td>33</td>
</tr>
<tr>
<td>4-4.</td>
<td>I/O Byte Field Definitions</td>
<td>34</td>
</tr>
<tr>
<td>5-1.</td>
<td>Disk Parameter Header Elements</td>
<td>40</td>
</tr>
<tr>
<td>5-2.</td>
<td>Disk Parameter Block Fields</td>
<td>42</td>
</tr>
<tr>
<td>5-3.</td>
<td>BSH and BLM Values</td>
<td>44</td>
</tr>
<tr>
<td>5-4.</td>
<td>EXM Values</td>
<td>45</td>
</tr>
<tr>
<td>A-1.</td>
<td>Distribution Disk Contents</td>
<td>55</td>
</tr>
<tr>
<td>F-1.</td>
<td>S-Record Field Contents</td>
<td>107</td>
</tr>
<tr>
<td>F-2.</td>
<td>S-Record Types</td>
<td>109</td>
</tr>
<tr>
<td>G-1.</td>
<td>CP/M-68K Error Messages</td>
<td>109</td>
</tr>
</tbody>
</table>

Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1.</td>
<td>CP/M-68K Interfaces</td>
<td>3</td>
</tr>
<tr>
<td>1-2.</td>
<td>Typical CP/M-68K Memory Layout</td>
<td>4</td>
</tr>
<tr>
<td>4-1.</td>
<td>Memory Region Table Format</td>
<td>32</td>
</tr>
<tr>
<td>4-2.</td>
<td>I/O Byte Fields</td>
<td>34</td>
</tr>
<tr>
<td>5-1.</td>
<td>Disk Parameter Header</td>
<td>40</td>
</tr>
<tr>
<td>5-2.</td>
<td>Sample Sector Translate Table</td>
<td>42</td>
</tr>
<tr>
<td>5-3.</td>
<td>Disk Parameter Block</td>
<td>42</td>
</tr>
<tr>
<td>F-1.</td>
<td>S-Reference Fields</td>
<td>107</td>
</tr>
</tbody>
</table>
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\textsuperscript{TM} version 2.2 and CP/M-86\textsuperscript{®} 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>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nibble</td>
<td>4-bit half-byte</td>
</tr>
<tr>
<td>byte</td>
<td>8-bit value</td>
</tr>
<tr>
<td>word</td>
<td>16-bit value</td>
</tr>
<tr>
<td>longword</td>
<td>32-bit value</td>
</tr>
<tr>
<td>address</td>
<td>32-bit identifier of a storage location</td>
</tr>
<tr>
<td>offset</td>
<td>a value defining an address in storage; a fixed displacement from some other address</td>
</tr>
</tbody>
</table>
### Table 1-1. (continued)

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

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.
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.

<table>
<thead>
<tr>
<th>Interrupt Vectors</th>
<th>Base Page</th>
<th>Text</th>
<th>bss</th>
<th>Data</th>
<th>Free Memory</th>
<th>User Stack</th>
<th>CCP &amp; BDOS &amp; BIOS</th>
<th>Top of Memory</th>
<th>CP/M 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000H</td>
<td>00400H</td>
<td>00500H</td>
<td>TPA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CP/M 68K</td>
</tr>
</tbody>
</table>

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 BDOS 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
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, LO68, 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 -O CPM.REL CPMLIB BIOS.O
```

where BIOS.O 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 CPM.REL CPM.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 BDOS 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)
3.2 Creating the Cold Boot Loader

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 LDRLIBS.O. Then the following command links together the code that must go on the system tracks.

A>lo68 -s -T400 -uld -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 reads 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

4.1 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 D0.W, and function parameters in registers D1 and D2. It then executes a TRAP 3 instruction. D0.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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0.W = function code</td>
</tr>
<tr>
<td>D1.x = first parameter</td>
</tr>
<tr>
<td>D2.x = second parameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0.B = byte values (8 bits)</td>
</tr>
<tr>
<td>D0.W = word values (16 bits)</td>
</tr>
<tr>
<td>D0.L = longword values (32 bits)</td>
</tr>
</tbody>
</table>

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

Table 4-2. BIOS Functions

<table>
<thead>
<tr>
<th>Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initialization (called for cold boot)</td>
</tr>
<tr>
<td>1</td>
<td>Warm Boot (called for warm start)</td>
</tr>
<tr>
<td>2</td>
<td>Console Status (check for console character ready)</td>
</tr>
<tr>
<td>3</td>
<td>Read Console Character In</td>
</tr>
<tr>
<td>4</td>
<td>Write Console Character Out</td>
</tr>
<tr>
<td>5</td>
<td>List (write listing character out)</td>
</tr>
<tr>
<td>6</td>
<td>Auxiliary Output (write character to auxiliary output device)</td>
</tr>
<tr>
<td>7</td>
<td>Auxiliary Input (read from auxiliary input)</td>
</tr>
<tr>
<td>8</td>
<td>Home (move to track 00)</td>
</tr>
<tr>
<td>9</td>
<td>Select Disk Drive</td>
</tr>
<tr>
<td>10</td>
<td>Set Track Number</td>
</tr>
<tr>
<td>11</td>
<td>Set Sector Number</td>
</tr>
<tr>
<td>12</td>
<td>Set DMA Address</td>
</tr>
<tr>
<td>13</td>
<td>Read Selected Sector</td>
</tr>
<tr>
<td>14</td>
<td>Write Selected Sector</td>
</tr>
<tr>
<td>15</td>
<td>Return List Status</td>
</tr>
<tr>
<td>16</td>
<td>Sector Translate</td>
</tr>
<tr>
<td>17</td>
<td>Get Memory Region Table Address</td>
</tr>
<tr>
<td>18</td>
<td>Get I/O Mapping Byte</td>
</tr>
<tr>
<td>19</td>
<td>Set I/O Mapping Byte</td>
</tr>
<tr>
<td>20</td>
<td>Flush Buffers</td>
</tr>
<tr>
<td>21</td>
<td>Set Exception Handler Address</td>
</tr>
</tbody>
</table>
FUNCTION 0: INITIALIZATION

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register DO.W: 00H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register DO.W: User/Disk Numbers</td>
</tr>
</tbody>
</table>

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 D0 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.l #traphndl,$8c ;set trap 3 handler
    clr.l d0 ;login drive A, user 0
    rts
```
FUNCTION 1: WARM BOOT

Entry Parameters:  
Register DO.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

<table>
<thead>
<tr>
<th>Function 2: Console Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry Parameters:</strong></td>
</tr>
<tr>
<td>Register D0.W: 02H</td>
</tr>
<tr>
<td><strong>Returned Value:</strong></td>
</tr>
<tr>
<td>Register D0.W: 00FFH if ready</td>
</tr>
<tr>
<td>Register D0.W: 0000H if not ready</td>
</tr>
</tbody>
</table>

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: | Returned Value: |
| Register D0.W: 03H | 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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 04H</td>
</tr>
<tr>
<td>Register D1.W: Character</td>
</tr>
<tr>
<td>Returned Value: None</td>
</tr>
</tbody>
</table>

This function sends the character from register D1 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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register DO.W: 05H</td>
<td>None</td>
</tr>
<tr>
<td>Register DL.W: Character</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 06H</td>
<td>Register D0.W: Character</td>
</tr>
<tr>
<td>Register D1.W: Character</td>
<td></td>
</tr>
</tbody>
</table>

This function sends an ASCII character from register D1 to the currently assigned auxiliary output device.
<table>
<thead>
<tr>
<th>Function 7: Auxiliary Input</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry Parameters:</strong></td>
</tr>
<tr>
<td>Register D0.W: 07H</td>
</tr>
<tr>
<td><strong>Returned Value:</strong></td>
</tr>
<tr>
<td>Register D0.W: Character</td>
</tr>
</tbody>
</table>

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 (1AH).
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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 09H</td>
</tr>
<tr>
<td>Register D1.B: Disk Drive</td>
</tr>
<tr>
<td>Register D2.B: Logged in Flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.L: Address of Selected Drive's DPH</td>
</tr>
</tbody>
</table>

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/O 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 D1.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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 0BH</td>
</tr>
<tr>
<td>Register D1.W: Sector Number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
</tbody>
</table>

This function specifies in register D1.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 D1.L: DMA Address

Returned Value: None

This function contains the DMA (disk memory access) address in register D1 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: ODH |

| 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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 0EH</td>
<td>Register D0.W: 0=no error</td>
</tr>
<tr>
<td>Register D1.W: 0=normal write</td>
<td>1=physical error</td>
</tr>
<tr>
<td>1=write to a directory sector</td>
<td></td>
</tr>
<tr>
<td>2=write to first sector of new block</td>
<td></td>
</tr>
</tbody>
</table>

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 D1.W indicates whether the write is an ordinary write operation or whether 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 D1.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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: OFH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0: 00FFH = device ready</td>
</tr>
<tr>
<td>Register D0: 0000H = device not ready</td>
</tr>
</tbody>
</table>

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.
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 D1.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, it 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 BDOS, 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 format of the MRT is shown below:

<table>
<thead>
<tr>
<th>Entry Count = 1</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base address of first region</td>
<td>32 bits</td>
</tr>
<tr>
<td>Length of first region</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 13H</td>
<td>Register D0.W: I/O Byte Current Value</td>
</tr>
</tbody>
</table>

This function returns the current value of the logical to physical input/output device byte (I/O byte) in register D0.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 BIOS. Device characteristics are defined in Table 4-3 below.

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

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSOLE</td>
<td>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.</td>
</tr>
<tr>
<td>LIST</td>
<td>The listing device is a hard-copy device, usually a printer.</td>
</tr>
<tr>
<td>AUXILIARY OUTPUT</td>
<td>An optional serial output device.</td>
</tr>
<tr>
<td>AUXILIARY INPUT</td>
<td>An optional serial input device.</td>
</tr>
</tbody>
</table>

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 lAH (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.

<table>
<thead>
<tr>
<th>I/O Byte bits</th>
<th>LIST</th>
<th>AUXILIARY OUTPUT</th>
<th>AUXILIARY INPUT</th>
<th>CONSOLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,6</td>
<td>5,4</td>
<td>3,2</td>
<td>1,0</td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>CONSOLE field (bits 1,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AUXILIARY INPUT field (bits 3,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
Table 4-4. (continued)

<table>
<thead>
<tr>
<th>AUXILIARY OUTPUT field (bits 5,4)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AUXILIARY OUTPUT is the Teletype device (TTY:)</td>
</tr>
<tr>
<td>1</td>
<td>AUXILIARY OUTPUT is the high-speed punch device (PTP:)</td>
</tr>
<tr>
<td>2</td>
<td>user defined punch #1 (UP1:)</td>
</tr>
<tr>
<td>3</td>
<td>user defined punch #2 (UP2:)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIST field (bits 7,6)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LIST is the Teletype device (TTY:)</td>
</tr>
<tr>
<td>1</td>
<td>LIST is the CRT device (CRT:)</td>
</tr>
<tr>
<td>2</td>
<td>LIST is the line printer device (LPT:)</td>
</tr>
<tr>
<td>3</td>
<td>user defined list device (UL1:)</td>
</tr>
</tbody>
</table>

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.
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

<table>
<thead>
<tr>
<th>Entry Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 15H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register D0.W: 0000H = successful write</td>
</tr>
<tr>
<td>Register D0.W: FFFFH = unsuccessful write</td>
</tr>
</tbody>
</table>

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.
This function sets the exception vector indicated in register D1.W to the value specified in register D2.L. The previous vector value is returned in register D0.L. Unlike the BDOS Set Exception Vector Function (61), this BIOS function sets any exception vector. Note that register D1.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 0BH.
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 BDOS 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 Controller. 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.

<table>
<thead>
<tr>
<th>XLT</th>
<th>0000</th>
<th>0000</th>
<th>0000</th>
<th>DIRBUF</th>
<th>DPB</th>
<th>CSV</th>
<th>ALV</th>
</tr>
</thead>
<tbody>
<tr>
<td>32b</td>
<td>16b</td>
<td>16b</td>
<td>16b</td>
<td>32b</td>
<td>32b</td>
<td>32b</td>
<td>32b</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLT</td>
<td>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.</td>
</tr>
<tr>
<td>0000</td>
<td>Three scratchpad words for use within the BDOS.</td>
</tr>
<tr>
<td>DIRBUF</td>
<td>Address of a 128-byte scratchpad area for directory operations within BDOS. All DPHs address the same scratchpad area.</td>
</tr>
<tr>
<td>DPB</td>
<td>Address of a disk parameter block for this drive. Drives with identical disk characteristics may address the same disk parameter block.</td>
</tr>
</tbody>
</table>
Table 5-1. (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV</td>
<td>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+1) / 4. (DRM is discussed in Section 5.2.3.)</td>
</tr>
<tr>
<td>ALV</td>
<td>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.)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Logical Sector</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Sector</td>
<td>1</td>
<td>7</td>
<td>13</td>
<td>19</td>
<td>25</td>
<td>5</td>
<td>11</td>
<td>17</td>
<td>23</td>
<td>3</td>
<td>9</td>
<td>15</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical Sector</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Sector</td>
<td>2</td>
<td>8</td>
<td>14</td>
<td>20</td>
<td>26</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>4</td>
<td>10</td>
<td>16</td>
<td>22</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>SPT</th>
<th>BSH</th>
<th>BLM</th>
<th>EXM</th>
<th>0</th>
<th>DSM</th>
<th>DRM</th>
<th>Reserved</th>
<th>CKS</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>16b</td>
<td>8b</td>
<td>8b</td>
<td>8b</td>
<td>8b</td>
<td>16b</td>
<td>16b</td>
<td>16b</td>
<td>16b</td>
<td>16b</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPT</td>
<td>Number of 128-byte logical sectors per track.</td>
</tr>
<tr>
<td>BSH</td>
<td>The block shift factor, determined by the data block allocation size, as shown in Table 5-3.</td>
</tr>
</tbody>
</table>
Table 5-2. (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLM</td>
<td>The block mask which is determined by the data block allocation size, as shown in Table 5-3.</td>
</tr>
<tr>
<td>EXM</td>
<td>The extent mask, determined by the data block allocation size and the number of disk blocks, as shown in Table 5-4.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved byte.</td>
</tr>
<tr>
<td>DSM</td>
<td>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.</td>
</tr>
<tr>
<td>DRM</td>
<td>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+1 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.</td>
</tr>
<tr>
<td>CKS</td>
<td>The size of the directory check vector, which is zero if the disk is permanently mounted, or length (CSV) = (DRM) / 4 + 1 for removable media.</td>
</tr>
<tr>
<td>OFF</td>
<td>The number of reserved tracks at the beginning of a logical disk. This is the number of the track on which the directory begins.</td>
</tr>
</tbody>
</table>

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 OFF-1. Zero is a legal value for OFF, and in that case, there are no boot tracks. The usual value of OFF for 8-inch floppy disks is two.

The tracks after the boot tracks (beginning with track number OFF) 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 $32768 \times BLS$. Because the largest allowable value for BLS is 16384, the biggest disk that can be accessed by CP/M-68K is $16384 \times 32768 = 512$ Mbytes.

Each directory entry may contain either 8 block numbers (if DSM $\geq 256$) or 16 block numbers (if DSM $< 256$). Each file needs enough directory entries to hold the block numbers of all blocks allocated to the file. 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 BLM. 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>
<thead>
<tr>
<th>BLS</th>
<th>BSH</th>
<th>BLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>2048</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>4096</td>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>8192</td>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>16384</td>
<td>7</td>
<td>127</td>
</tr>
</tbody>
</table>
Table 5-4. EXM Values

<table>
<thead>
<tr>
<th>BLS</th>
<th>DSM &lt;= 255</th>
<th>DSM &gt; 255</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>2048</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8192</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>16384</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

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=1 (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.

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 EXORterm™.

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

1) Place the disk in the first floppy drive (#FD04 with the UDC or #FD00 with the FDC).
2) Press SYSTEM RESET (front panel) and RETURN (this brings in MACSbug™).
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 can 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 S-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 _ccp 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 (00H) 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

- -H is an optional flag discussed below;
- <filename> is the name of the file to be written to the system tracks;
- <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

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR68.REL</td>
<td>Relocatable version of the archiver/librarian.</td>
</tr>
<tr>
<td>AS68INIT</td>
<td>Initialization file for assembler—see AS68 documentation in the CP/M-68K Operating System Programmer's Guide.</td>
</tr>
<tr>
<td>AS68.REL</td>
<td>Relocatable version of the assembler.</td>
</tr>
<tr>
<td>ASM.SUB</td>
<td>Submit file to assemble an assembly program with file type .S, put the object code in filename.O, and a listing file in filename.PRN.</td>
</tr>
<tr>
<td>BIOS.O</td>
<td>Object file of BIOS for EXORmacs.</td>
</tr>
<tr>
<td>BIOS.C</td>
<td>C language source for the EXORmacs BIOS as distributed with CP/M-68K.</td>
</tr>
<tr>
<td>BIOSA.O</td>
<td>Object file for assembly portion of EXORmacs BIOS.</td>
</tr>
<tr>
<td>BIOSA.S</td>
<td>Source for the assembly language portion of the EXORmacs BIOS as distributed with CP/M-68K.</td>
</tr>
<tr>
<td>BIOSTYPS.H</td>
<td>Include file for use with BIOS.C.</td>
</tr>
<tr>
<td>BOOTER.O</td>
<td>Object for EXORmacs bootstrap.</td>
</tr>
<tr>
<td>BOOTER.S</td>
<td>Assembly boot code for the EXORmacs.</td>
</tr>
<tr>
<td>C.SUB</td>
<td>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&gt;C filename.</td>
</tr>
<tr>
<td>C068.REL</td>
<td>Relocatable version of the C parser.</td>
</tr>
<tr>
<td>C168.REL</td>
<td>Relocatable version of the C code generator.</td>
</tr>
<tr>
<td>File</td>
<td>Contents</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CLIB</td>
<td>The C run-time library.</td>
</tr>
<tr>
<td>CLINK.SUB</td>
<td>Submit file for linking C object programs with the C run-time library.</td>
</tr>
<tr>
<td>CP68.REL</td>
<td>Relocatable version of the C preprocessor.</td>
</tr>
<tr>
<td>CPM.H</td>
<td>Include file with C definitions for CP/M-68K. See the C Programming Guide for CP/M-68K for details.</td>
</tr>
<tr>
<td>CPM.REL</td>
<td>Relocatable version of CPM.SYS.</td>
</tr>
<tr>
<td>CPM.SYS</td>
<td>CP/M-68K operating system file for the EXORmacs.</td>
</tr>
<tr>
<td>CPMLIB</td>
<td>Library of object files for CP/M-68K. See Section 2.</td>
</tr>
<tr>
<td>CPMLDR.SYS</td>
<td>The bootstrap loader for the EXORmacs. A copy of this was written to the system tracks using PUTBOOT.</td>
</tr>
<tr>
<td>CTYPE.H</td>
<td>Same as above.</td>
</tr>
<tr>
<td>DDT.REL</td>
<td>Relocatable version of the preloader for DDT*. (Loads DDT1 into the high end of the TPA.)</td>
</tr>
<tr>
<td>DDT1.68K</td>
<td>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.</td>
</tr>
<tr>
<td>DUMP.REL</td>
<td>Relocatable version of the DUMP utility.</td>
</tr>
<tr>
<td>ED.REL</td>
<td>Relocatable version of the ED utility.</td>
</tr>
<tr>
<td>ELDBIOS.S</td>
<td>Assembly language source for the ERG sample loader BIOS.</td>
</tr>
<tr>
<td>ERGBIOS.S</td>
<td>Assembly language source for the ERG sample BIOS.</td>
</tr>
<tr>
<td>ERRNO.H</td>
<td>Same as above.</td>
</tr>
<tr>
<td>FORMAT.REL</td>
<td>Relocatable disk formatter for the Motorola EXORmacs.</td>
</tr>
</tbody>
</table>

56
<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT.S</td>
<td>Assembly language source for the FORMAT utility.</td>
</tr>
<tr>
<td>INIT.REL</td>
<td>Relocatable version of the INIT utility.</td>
</tr>
<tr>
<td>INIT.S</td>
<td>Assembly language source for the INIT utility.</td>
</tr>
<tr>
<td>LCPM.SUB</td>
<td>Submit file to create CPM.REL for EXORmacs.</td>
</tr>
<tr>
<td>LDBIOS.O</td>
<td>Object file of loader BIOS for EXORmacs.</td>
</tr>
<tr>
<td>LDBIOSA.O</td>
<td>Object file for assembly portion of EXORmacs loader BIOS.</td>
</tr>
<tr>
<td>LDBIOSA.S</td>
<td>Source for the assembly language portion of the EXORmacs loader BIOS as distributed with CP/M-68K.</td>
</tr>
<tr>
<td>LDRLIB</td>
<td>Library of object files for creating a Bootstrap Loader. See Section 3.</td>
</tr>
<tr>
<td>LO68.REL</td>
<td>Relocatable version of the linker.</td>
</tr>
<tr>
<td>LOADBIOS.H</td>
<td>Include file for use with BIOS.C, to make it into a loader BIOS.</td>
</tr>
<tr>
<td>LOADBIOS.SUB</td>
<td>Submit file to create loader BIOS for EXORmacs.</td>
</tr>
<tr>
<td>MAKELDR.SUB</td>
<td>Submit file to create CPMLDR.SYS on EXORmacs.</td>
</tr>
<tr>
<td>NORMBIOS.H</td>
<td>Include file for use with BIOS.C, to make it into a normal. BIOS</td>
</tr>
<tr>
<td>NORMBIOS.SUB</td>
<td>Submit file to create normal BIOS for EXORmacs.</td>
</tr>
<tr>
<td>NM68.REL</td>
<td>Relocatable version of the symbol table dump utility.</td>
</tr>
<tr>
<td>PIP.REL</td>
<td>Relocatable version of the PIP utility.</td>
</tr>
<tr>
<td>PORTAB.H</td>
<td>Same as above.</td>
</tr>
<tr>
<td>PUTBOOT.REL</td>
<td>Relocatable version of the PUTBOOT utility.</td>
</tr>
<tr>
<td>File</td>
<td>Contents</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PUTBOOT.S</td>
<td>Assembly language source for the PUTBOOT utility.</td>
</tr>
<tr>
<td>README.TXT</td>
<td>ASCII file containing information relevant to this shipment of CP/M-68K. This file might not be present.</td>
</tr>
<tr>
<td>RELCPM.SUB</td>
<td>Submit file to relocate CPM.REL into CPM.SYS.</td>
</tr>
<tr>
<td>RELOC.REL</td>
<td>Relocatable version of the command file relocation utility.</td>
</tr>
<tr>
<td>RELOCx.SUB b</td>
<td>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.</td>
</tr>
<tr>
<td>S.O</td>
<td>Startup routine for use with C programs--must be first object file linked.</td>
</tr>
<tr>
<td>SENC68.REL</td>
<td>Relocatable version of the S-record creation utility.</td>
</tr>
<tr>
<td>SETJMP.H</td>
<td>Same as above.</td>
</tr>
<tr>
<td>SIGNAL.H</td>
<td>Same as above.</td>
</tr>
<tr>
<td>SIZE68.REL</td>
<td>Relocatable version of the SIZE68 utility.</td>
</tr>
<tr>
<td>SR128K.SYS</td>
<td>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.</td>
</tr>
<tr>
<td>SR400.SYS</td>
<td>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.</td>
</tr>
<tr>
<td>STAT.REL</td>
<td>Relocatable version of the STAT utility.</td>
</tr>
<tr>
<td>STDIO.H</td>
<td>Include file with standard I/O definitions for use with C programs. See the C Programming Guide for CP/M-68K for details.</td>
</tr>
</tbody>
</table>

End of Appendix A
Appendix B
Sample BIOS Written in Assembly Language

Listing B-1. Sample Assembly Language BIOS
**CP/M-68K System Guide**

**B Sample Assembly Language BIOS**

42 0000005E 00000108
43 00000062 00000114
44 00000066 0000029C
45 0000006A 000002A4
46 0000006E 000002A6
47 00000072 00000298
48 00000076 000002A8
49
50
51
52 0000007A 4EP900000000
53
54 00000080 103900FFFF00
55 00000086 02400002

**Source File:** aergbios.s

56 0000008A 6704
57 0000008C 7001
58 0000008E 4E75
59
60 00000090 4280
61 00000092 4E75
62
63 00000094 61EA
64 00000096 4A40
65 00000098 67FA
66 0000029A 103900FFFF00
67 00000290 C0BC0000007F
68 00000294 4E75
69 70 00000298 103900FFFF01
71 0000029A C01C0001
72 00000298 6704
73 00000294 1C1200FFFF00
74 0000029E 4E75
75
76 0000029C 4E75
77
78 000002BE 4E75
79
80 000002C0 4E75
81
82 000002C2 103000FF
83 000002C6 4E75
84
85
86
87
88
89
90
91
92

**Listing B-1. (continued)**
Listing B-1. (continued)
Listing B-1. (continued)
newdrive:
move.b selcode,dcntrol * select the drive
move.b seldrv,curdrv

newtrk:
move.b sector,dsect  * set up sector number
move.b track,dtrak  * set up track number
move.1 dma,4u     * dma address to 4u

exit:
b r chkselk   * force head load delay

errchk:
btst $4,d7   * record not found error, reseek

chkseek:
* check for correct track, seek if necessary
btst chksel  * find out what track we're on
beq chksel   * if read id ok, skip restore code

restor e:
* home the drive and seek to correct track
move.b $50b,dcmd   * restore command to command port

rstwait:
btst $7,dwait   * loop until restore completed
beq restore    * if not at track 0, try again
clr.1 d3      * track number returned in d3 from readid

chks2:
move.b d3,dtrak   * update track register in FOC
move.b track,oldtrak  * are we at right track?
beq chksdone   * if yes, exit

chksdone:
move.b track,ddata  * else, put desired track in data reg of FOC
btst $17,dwait   * and issue a seek command
beq rstatus    * wait for intrq

readid:
* read track id, return track number in d3
move.b $50b,dcmd   * issue read id command
move.b dwait,d7    * wait for intrq
move.b ddata,d3    * track byte to d3

Listing B-1. (continued)
Listing B-1. (continued)
Listing B-1. (continued)
Listing B-1. (continued)

End of Appendix B
Appendix C
Sample Loader BIOS Written in Assembly Language

Listing C-1. Sample BIOS Loader
43 00000060 00000000 .dc.l nogood
44 00000064 00000000 .dc.l nogood
45 00000068 00000222 .dc.l setexc
46 47
48
49 0000006C 103900FFFFOl constat: move.b $fff01,00
50 00000072 02400002 andi.w $2,00
51 00000076 6704 beq noton
52 00000078 7001 moveq.l $1,00
53 0000007A 00000000 rts
54 0000007C 4280
55 0000007E 4E75
56 00000080 61EA
57 00000082 4A40 conin: bsr constat
58 00000084 67F4 tst 00
59 00000086 103900FFFF00 move.b $fff00,00
60 00000088 E7F4 andi.l $7f,00
61 00000090 4E75 rts
62 00000092 00000000 conout: move.b $fff01,00
63 00000096 103900FFFF00 move.b $fff00,00
64 00000098 61EA
65 0000009A 67FA beq conout
66 0000009C 4280
67 0000009E 4E75
68 000000A0 103900FFFF00 move.b $fff00,00
69 000000A2 61EA
70 000000A4 4280
71 000000A6 4E75
72
73
74
75
76 maxdsk = 2 * this BIOS supports 2 floppy drives
77 dphlen = 26 * length of disk parameter header
78
79 iobase = $0fffff8 * Tarbell floppy disk port base address
80 dcmd = iobase * output port for command
81 dstat = iobase * input status port
82 dtrk = iobase+1 * disk track port
83 dsect = iobase+2 * disk sector port
84 ddata = iobase+3 * disk data port
85 dwait = iobase+4 * input port to wait for op finished
86 dcntrl = iobase+4 * output control port for drive selection
87
88 89 000000A8 423900000002 home: clr.b track
90 000000AA 4875 rts
91
92
93
94 000000B0 423900000000 * select drive A
95 clr.b seldrv

Listing C-1. (continued)
Listing C-1. (continued)
Listing C-1. (continued)

C Sample Loader BIOS

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>147</td>
<td>00000134</td>
<td>B63900000000</td>
</tr>
<tr>
<td>148</td>
<td>00000138</td>
<td>B661A</td>
</tr>
<tr>
<td>149</td>
<td>0000013C</td>
<td>163900000002</td>
</tr>
<tr>
<td>150</td>
<td>00000142</td>
<td>B63900000003</td>
</tr>
<tr>
<td>151</td>
<td>00000148</td>
<td>6620</td>
</tr>
<tr>
<td>152</td>
<td>0000014A</td>
<td>4283</td>
</tr>
<tr>
<td>153</td>
<td>0000014C</td>
<td>0839000500FFFFF8</td>
</tr>
<tr>
<td>154</td>
<td>00000154</td>
<td>6618</td>
</tr>
<tr>
<td>155</td>
<td>00000156</td>
<td>13F90000000000000000</td>
</tr>
<tr>
<td>156</td>
<td>0000015A</td>
<td>6618</td>
</tr>
<tr>
<td>157</td>
<td>0000015C</td>
<td>163900000002</td>
</tr>
<tr>
<td>158</td>
<td>00000160</td>
<td>6620</td>
</tr>
<tr>
<td>159</td>
<td>00000162</td>
<td>13F9000000200FFFFF9</td>
</tr>
<tr>
<td>160</td>
<td>00000164</td>
<td>207300000006</td>
</tr>
<tr>
<td>161</td>
<td>00000166</td>
<td>4E75</td>
</tr>
<tr>
<td>162</td>
<td>00000168</td>
<td>13FC000900000000</td>
</tr>
<tr>
<td>163</td>
<td>0000016A</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>164</td>
<td>0000016C</td>
<td>7604</td>
</tr>
<tr>
<td>165</td>
<td>0000016E</td>
<td>13F90000000200FFFFF9</td>
</tr>
<tr>
<td>166</td>
<td>00000170</td>
<td>13FC000900000000</td>
</tr>
<tr>
<td>167</td>
<td>00000172</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>168</td>
<td>00000174</td>
<td>6126</td>
</tr>
<tr>
<td>169</td>
<td>00000176</td>
<td>163900000002</td>
</tr>
<tr>
<td>170</td>
<td>00000178</td>
<td>0000015A</td>
</tr>
<tr>
<td>171</td>
<td>0000017A</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>172</td>
<td>0000017C</td>
<td>163900000002</td>
</tr>
<tr>
<td>173</td>
<td>0000017E</td>
<td>13FC000900000000</td>
</tr>
<tr>
<td>174</td>
<td>00000180</td>
<td>6126</td>
</tr>
<tr>
<td>175</td>
<td>00000182</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>176</td>
<td>00000184</td>
<td>7604</td>
</tr>
<tr>
<td>177</td>
<td>00000186</td>
<td>13F90000000200FFFFF9</td>
</tr>
<tr>
<td>178</td>
<td>00000188</td>
<td>13FC000900000000</td>
</tr>
<tr>
<td>179</td>
<td>0000018A</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>180</td>
<td>0000018C</td>
<td>7604</td>
</tr>
<tr>
<td>181</td>
<td>0000018E</td>
<td>13F90000000200FFFFF9</td>
</tr>
<tr>
<td>182</td>
<td>00000190</td>
<td>163900000002</td>
</tr>
<tr>
<td>183</td>
<td>00000192</td>
<td>163900000002</td>
</tr>
<tr>
<td>184</td>
<td>00000194</td>
<td>6126</td>
</tr>
<tr>
<td>185</td>
<td>00000196</td>
<td>13FC000900000000</td>
</tr>
<tr>
<td>186</td>
<td>00000198</td>
<td>6126</td>
</tr>
<tr>
<td>187</td>
<td>0000019A</td>
<td>13FC000900000000</td>
</tr>
<tr>
<td>188</td>
<td>0000019C</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>189</td>
<td>0000019E</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>190</td>
<td>000001A0</td>
<td>163900000002</td>
</tr>
<tr>
<td>191</td>
<td>000001A2</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>192</td>
<td>000001A4</td>
<td>6126</td>
</tr>
<tr>
<td>193</td>
<td>000001A6</td>
<td>163900000002</td>
</tr>
<tr>
<td>194</td>
<td>000001A8</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>195</td>
<td>000001AA</td>
<td>163900000002</td>
</tr>
<tr>
<td>196</td>
<td>000001AC</td>
<td>0839000000000000</td>
</tr>
<tr>
<td>197</td>
<td>000001AEE</td>
<td>6126</td>
</tr>
<tr>
<td>198</td>
<td>000001B0</td>
<td>163900000002</td>
</tr>
</tbody>
</table>

errchk:
btst $4, d7
bne chseek

chkseek:
check for correct track, seek if necessary

chkdone:
chkdone

readid:
read status to clear FDC
Listing C-1. (continued)
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 BIOS proper. 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.

BIOS.C

This is the main text of the C language BIOS for the EXORMacs.

ivecure D-1. EXORMacs BIOS Written in C


```c
#define NAK 0x15
#define PXTSTX 0x0 /* offsets within a disk packet */
#define PXTID 0x1
#define PXTSZ 0x2
#define PXTDEV 0x3
#define PXTCHCOM 0x4
#define PXTSTCOM 0x5
#define PXTSTVAL 0x6
#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 a10;
    BYTE a11;
    WORD cks;
    WORD off;
};

/* Disk Parameter Header Structure */
struct dph
{
    BYTE *xltp;
    WORD dphscr[3];
    BYTE *dirbufp;
    struct dpb *dpbp;
    BYTE *csvp;
    BYTE *alvp;
};

/**********************************************/
/* Direction Buffer for use by the BIOS */
**********************************************/
BYTE dirbuf[128];

Listing D-1. (continued)

```
## Sector Translate Table for Floppy Disks

Listing D-1. (continued)
Disk Parameter Headers

Four disks are defined:
- dsk a: diskno=0, (Motorola's fd04)
- dsk b: diskno=1, (Motorola's fd05)
- dsk c: diskno=2, (Motorola's h001)
- dsk d: diskno=3, (Motorola's h001)

Listing D-1. (continued)
CP/M-68K System Guide

Listing D-1. (continued)
struct hmpkst {
    BYTE a1;
    BYTE a2;
    BYTE a3;
    BYTE dskno;
    BYTE com1;
    BYTE com2;
    BYTE a6;
    BYTE a7;
}

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 numbklks;
    WORD blksize;
    LONG iobsf;
    WORD cksnum;
    [LONG lsect];
    BYTE etxchr;
    BYTE rwpad;
};

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 fmtblkno;
    BYTE fmtchcmd;
    BYTE fmtdevcmd;
    BYTE ffmtetx;
    BYTE fmtetx;
    BYTE fmtpad;
};

struct fmtpkst fmtpack = { 512, 1792, 0x4002, 0x0300 };
#endif

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

Listing D-l. (continued)
```c
BYTE portin(port);
REG BYTE *port;
{
    while (!portstat(port)); /* wait for input */
    return (*({port + PORTTHD})); /* got some, return it */
}

********************************************************************************

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

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

********************************************************************************

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

#endif

bioserr(errmsg);
REG BYTE *errmsg;
{
    printf("nBIOS ERROR -- ");
    printf(errmsg);
    printf(".
");

    printf(errmsg); /* used by bioserr */
    REG BYTE *s;
    {
        while (*s) {portout(PORT1,*s); s += 1; }
    }
    else
bioserr(); /* minimal error procedure for loader BIOS */
    
}endif

Listing D-1. (continued)
EXTERN dskia(); /* external interrupt handler -- calls dskic */
EXTERN setmask(); /* 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 ++; // ??? */
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 ++; else sendack();
    }
}
/* end of dskic */

/* Read status packet from IPC */
getstpkt(stpkt)
REG BYTE *stpktp;
{
    REG BYTE *p, *q;
    REG WORD i;
}

Listing D-1. (continued)
p = stpckt;
q = (DSKIPC + PKTPMIPC);
for ( i = STPKTSZ; i; i --)
{
    p -= q;
    p += 1;
    q += 2;
}

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

unsolst(stpckt)
REG BYTE *stpckt;
{
    REG WORD dev;
    REG WORD ready;
    REG struct dskst *dsp;
    dev = cncvdk[ (stpckt+PKTDEV)->byte ];
    ready = ((stpckt+PKTSPRM)->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

Listing D-1. (continued)
iopackp = (DSKIPC+PKTTOIPC);
do {*iopackp = *pktadr++; iopackp += 2; pktsize -= 1;} while(pktsize);
(DSKIPC+MSGTOIPC)->byte = 0x80;
imsave = setimask(7);
dskstate[wavdsk].state = ACTIVE;
prestate = 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 + MSGTOIPC)->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;

Listing D-1. (continued)
/* build packet */

REG WORD sectcnt;
REG WORD result;

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

    rwpack.dskno = cnvdsk[dsk];
    rwpack.trk = bufp;
    sectcnt = (dphstab[dsk].dphbp) -> spt;
    rwpack.lsect = trk * (sectcnt >> 1);
    rwpack.ckcmd = cmd;
    rwpack.numbiks = (sectcnt >> 1);

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

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

if ! LOADER

#define

flushl(tbp)
struct tbstr *tbp;
{
    REG WORD ok;
    if ( (tbp->valid & bp->dirty) ok = dskxfer(tbp->dsk, bp->trk, bp->buf, DSKWRITE);
        else ok = 1;
    bp->dirty = 0; /* even if error, mark not dirty */
    bp->valid & ok; /* otherwise system has trouble */
        /* continuing. */
    return(ok);
}

Listing D-1. (continued)
Listing D-1. (continued)
REG WORD imsav;

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

imsav = setmask(7);
if (1 dskstate[setdsk].ready )
{
    setmask(imsav);
    tblp = 0L;
    return (tblp);
}

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

tbp = firstbuf;
ltbp = 0;
mtbp = 0;
while (tblp)
{
    if ( (tblp->valid) && (tblp->dsk == setdsk)
        && (tblp->trk == settrk) )
    {
        if (ltbp) /* found it -- rearrange LRU links */
        {
            ltbp->nextbuf = tblp->nextbuf;
            tblp->nextbuf = firstbuf;
            firstbuf = tblp;
        }
        setmask(imsav);
        return (tblp);
    }
    else
    {
        mtbp = tblp; /* move along to next buffer */
        ltbp = tblp;
        tblp = tblp->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;
setmask(imsav);
if (flushl(ltbp) && fill(ltbp)) mtbp = ltbp; /* success */
else
    mtbp = 0L; /* failure */
return (mtbp);

Listing D-1. (continued)
/*************** READ Function -- read one sector *****************************/
read()
{
    REG BYTE *p;
    REG BYTE *q;
    REG WORD i;
    REG struct tbstr *tbp;
    tbp = gettrk(); /* locate track buffer with sector */
    if ( ! tbp ) return(1); /* failure */
    /* locate sector in buffer and copy contents to user area */
    p = (tbp->buf) + (setsec << 7); /* multiply by shifting */
    q = setdma;
    i = 128;
    do {*p++ = *q++; i -= 1;} while (i); /* this generates good code */
    return(0);
}

/*** WRITE Function -- write one sector *******************/
write(mode)
{
    REG BYTE *p;
    REG BYTE *q;
    REG WORD i;
    REG struct tbstr *tbp;
    /* locate track buffer containing sector to be written */
    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 */
    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 ( flushl(tbp) ) return(0); else return(1);}
    else return(0);
}

Listing D-1. (continued)
Listing D-1. (continued)
return(oldval);

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

LONG slctdsk(dsk, logged)
REG BYTE dsk;
BYTE logged;
{
    REG struct dph *dphp;
    REG BYTE stl, 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. */
        if ( (dsk > MAXDSK) || (dsk == 2) )
            {
                printstr("nrBIOS ERROR -- DISK ");
                portout(PORT1, 'A'+dsk);
                printstr(" NOT SUPPORTEDnr");
                return(OL);
            }
    endif

    dphp = &dphtab[dsk];
    if! (logged & 0x1) 
    {
        hmpack.dskno = cvndsk[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 (OL);
            }
    }
    getstpkt(stpkt); /* determine disk type and size */
    sendack();
    ipcstate = IDLE;
    stl = stpkt[PKTSTPRM];
    st2 = stpkt[PKTSTPRM+1];

Listing D-1. (continued)
if ( stl & 0x80 ) /* not ready / ready */
    
    dskstate[dsk].ready = 0;
    return(0L);
} else
    
    dskstate[dsk].ready = 1;

switch ( stl & 7 )
{

    case 1 : /* floppy disk */
        
        dpb->dpbp = &dpb0;
        break;
    case 2 : /* hard disk */
        
        dpb->dpbp = &dpb2;
        break;
    default : bioserr("Invalid Disk Status");
        
        dpb = OL;
        break;
}

return(dpbh);

/****** 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;
    
    fmtpack.dskno = cvdsk[setdsk];
    actvdsk = setdsk;
    dskstate[setdsk].change = 0;
    sendpkt(&fmtpack, 7);
    if ( ! dskwait(setdsk, 0x70, 0x0) ) retval = 0;
    else
    
        retval = 1;

Listing D-1. (continued)
sendack();
ipcstate = IDLE;
return(setval);
}
#endif
	
																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
tubuf[i].valid = 0;
tubuf[i].dirty = 0;
if ( (i+1) < NUMTB )
tubuf[i].nextbuf = &tbuf[i+1];
else

tubuf[i].nextbuf = 0;

firstbuf = &tbuf[0];
lastbuf = &tbuf[NUMTB-1];

bufvalid = 0;

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

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

Listing D-1. (continued)
CP/M-68K System Guide

setimask(imsave); /* turn on interrupts */
}

******************************************************************************

******************************************************************************

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

LONG cbios(d0, dl, d2)
REG WORD d0;
REG LONG dl, d2;
{
    switch(d0)
    {
        case 0: biosinit(); /* INIT */
            break;
        #if ! LOADER
        case 1: flush(); /* WBOOT */
            initdsk();
            wbooth();
            /* break; */
        #endif
        case 2: return(portstat(PORT1)); /* CONST */
            /* break; */
        case 3: return(portin(PORT1)); /* CONIN */
            /* break; */
        case 4: portout(PORT1, (char) dl); /* CONOUT */
            break;
        case 5:;
        case 6: portout(PORT2, (char) dl); /* LIST */
            break;
        case 7: return(portin(PORT2)); /* PUNCH */
            /* break; */
        case 8: settrak = 0;
            break;
        case 9: return(glctdsk((char) dl, (char) d2)); /* SELDSK */
            /* break; */
        case 10: settrak = (int) dl;
            /* SETTRK */
            break;
        case 11: setsec = ((int) dl-1);
            /* SETSEC */
            break;
    }

Listing D-1. (continued)
Listing D-1. (continued)
/* End of C Bios */

NORMBIOS.H
This should be renamed "BIOSTYPE.H" if you are compiling a normal BIOS.

#define LOADER 0
#define CTLTYPE 0

LOADBIOS.H
This should be renamed "BIOSTYPE.H" if you are compiling a loader BIOS.

#define LOADER 1
#define CTLTYPE 0

BIOSA.H
This is the assembly language interface needed by the normal BIOS.

.text

Listing D-1. (continued)
Listing D-1. (continued)
This is the assembly language interface used by the loader BIOS.

```
.text
.globl _bios
.globl _biosinit
.globl _cbios
.globl _dskia
.globl _dskic
.globl _setimask

* *
* *
* _bios: link a6,#0
move.l d2,-(a7)
mov.l d1,-(a7)
mov.w d0,-(a7)
mov $2000, sr
lea _dskia,a0
mov.l a0,$3fc
jnr _cbios
unlk a6
rts

* _dskia: link a6,#0
movem.l d0-d7/a0-a5,-(a7)
jas _dskic
movem.l (a7)+,d0-d7/a0-a5
unlk a6
rte

* _setimask: movel sr,d0
lar $8,d0
and.l #7,d0
movel sr,dl
ror.w $8,dl
and.w $fff8,dl
add.w 4(a7),dl
ror.w $8,dl
movel dl, sr
rts
* 
.end
```

Listing D-1. (continued)
These type definitions are needed by the C BIOS.

`/*******************************/
 */
 /* Portable type definitions for use */
 /* with the C BIOS according to */
 /* CP/M-80K (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-1. (continued)

End of Appendix D
Appendix E
Putboot Utility Assembly Language Source

Program to Write Boot Tracks for CP/M-68K (tm)
Copyright Digital Research 1982

* ...

Listing E-1. PUTBOOT Assembly Language Source
Listing E-1. (continued)
Listing E-1. (continued)
Listing E-1. (continued)
Listing E-1. (continued)

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:  

<table>
<thead>
<tr>
<th>Field</th>
<th>S</th>
<th>type</th>
<th>length</th>
<th>address</th>
<th>data</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters:</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2, 4 or 6</td>
<td>variable</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure F-1. S-record Fields

The field contents are as follows:

Table F-1. S-record Field Contents

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>The ASCII Character 'S'. This signals the beginning of the S-record.</td>
</tr>
<tr>
<td>type</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table F-1. (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>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.</td>
</tr>
<tr>
<td>address</td>
<td>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.</td>
</tr>
<tr>
<td>data</td>
<td>The actual data to be loaded into memory, with each byte of data represented as a pair of hexadecimal digits, in ASCII.</td>
</tr>
<tr>
<td>checksum</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>1</td>
<td>This type of record contains normal data. The address field is two bytes long (four S-record characters).</td>
</tr>
<tr>
<td>2</td>
<td>Similar to Type 1, but with a 3-byte (six S-record characters) address field.</td>
</tr>
<tr>
<td>3</td>
<td>Similar to Type 1, but with a 4-byte (eight S-record characters) address field.</td>
</tr>
<tr>
<td>5</td>
<td>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).</td>
</tr>
<tr>
<td>7</td>
<td>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.</td>
</tr>
<tr>
<td>8</td>
<td>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.</td>
</tr>
<tr>
<td>9</td>
<td>This is similar to type 7 except that it ends a block of type 1 S-records, and its address field is 2 bytes (4 characters) long.</td>
</tr>
</tbody>
</table>

S-records are produced by the SENDC68 utility program (described in the CP/M-68K Operating System Programmer's Guide).
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>
<thead>
<tr>
<th>Message Meaning</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad relocation information bits</td>
<td>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.</td>
</tr>
<tr>
<td>BIOS ERROR -- DISK X NOT SUPPORTED</td>
<td>BIOS. The disk drive indicated by the variable &quot;X&quot; 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.</td>
</tr>
</tbody>
</table>

107
Table G-1. (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOS ERROR -- Invalid Disk Status</td>
<td>BIOS. 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.</td>
</tr>
<tr>
<td></td>
<td>1) Indicate which version of the operating system you are using.</td>
</tr>
<tr>
<td></td>
<td>2) Describe your system's hardware configuration.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>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.</td>
</tr>
<tr>
<td>Cannot Open Source File</td>
<td>PUTBOOT. PUTBOOT cannot locate the source file. Ensure that you specify the correct drive code and filename before you reenter the PUTBOOT command line.</td>
</tr>
</tbody>
</table>
### Table G-1. (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP/M Disk change error on drive x</td>
<td>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.</td>
</tr>
<tr>
<td>CP/M Disk file error: filename is read-only. Do you want to: Change it to read/write (C), or Abort (A)?</td>
<td>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.</td>
</tr>
<tr>
<td>CP/M Disk read error on drive x Do you want to: Abort (A), Retry (R), or Continue with bad data (C)?</td>
<td>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.)</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>CP/M Disk read error on drive x (continued)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Retries operation. If the retry fails, the system reprompts with the option message.</td>
</tr>
<tr>
<td>C</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A or CTRL-C</td>
<td>Terminates the operation and CP/M-68K returns the system prompt.</td>
</tr>
<tr>
<td>R</td>
<td>Retries operation. If the retry fails, the system reprompts with the option message (Meaning continued on next page.)</td>
</tr>
</tbody>
</table>
Table G-1. (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CP/M Disk write error on drive x (continued)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Action</strong></td>
</tr>
<tr>
<td>C</td>
<td>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.</td>
</tr>
<tr>
<td><strong>CP/M Disk select error on drive x</strong></td>
<td></td>
</tr>
<tr>
<td>Do you want to: Abort (A), Retry (R)</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><strong>CP/M Disk select error on drive x</strong></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>File already exists</td>
<td>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.</td>
</tr>
<tr>
<td>insufficient memory</td>
<td>CCP. This error could result from one of three causes:</td>
</tr>
<tr>
<td>or bad file header</td>
<td>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 file may have been corrupted. Reassemble or recompile the source file and relink it before you reenter the command line.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
Table G-1. (continued)

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid Command Line</td>
<td>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.</td>
</tr>
<tr>
<td>No file</td>
<td>CCP. The filename specified in the command line does not exist. Ensure that you use the correct filename and reenter the command line.</td>
</tr>
<tr>
<td>No wildcard filenames</td>
<td>CCP. The command specified in the command line does not accept wildcards in file specifications. Retype the command line using a specific filename.</td>
</tr>
<tr>
<td>Program Load Error</td>
<td>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.)</td>
</tr>
<tr>
<td>Message</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>read error on program load</td>
<td></td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too many arguments: argument?</td>
<td>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.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>User # range is [0-15]</td>
<td>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.</td>
</tr>
<tr>
<td>Write Error</td>
<td>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.</td>
</tr>
</tbody>
</table>
Index

-H flag, 53
0000, 40
_autost, 51
_ccp, 16
_ccp entry point, 50
_init, 15
_init entry point, 50
_usercmd, 51

A
absolute, 2
absolute data
down-loading, 50
address, 1
address space, 1
algorithms, 31
allocation vector, 11
ALV, 41
applications programs, 5
ASCII character, 5, 20
ASCII CTRL-Z (lAH), 22
AUXILIARY INPUT device, 33
AUXILIARY OUTPUT device, 33

B
base page, 2
BDOS, 3, 5, 6, 7, 50
BDOS Direct BIOS Function
Call 50, 13
BDOS function 61 Set Exception
Vector, 38
BIOS, 3, 5, 6, 10, 13
BIOS
compiled, 7
creating, 39
BIOS flush buffers operation, 47
BIOS function 0, 15
BIOS function 0
Initialization, 15
BIOS function 2 Console
Status, 17
BIOS function 3 Read Console
Character, 18
BIOS function 4 Write Console
Character, 19
BIOS function 5 List Character
Output, 20

BIOS function 6 Auxiliary
Output, 21
BIOS function 7 Auxiliary
Input, 22
BIOS function 8 Home, 23
BIOS function 9 Select Disk
Drive, 24
BIOS function 10 Set Track
Number, 25
BIOS function 11 Set Sector
Number, 26
BIOS function 12 Set DMA
Address, 27
BIOS function 13 Read Sector, 28
BIOS function 14 Write Sector, 29
BIOS function 15 Return List
Status, 30
BIOS function 16 Sector
Translate, 31
BIOS function 18 Get Address
of MRT, 32
BIOS function 19 Get I/O Byte, 33
BIOS function 20 Set I/O Byte, 36
BIOS function 21 Flush
Buffers, 37
BIOS function 22 Set Exception
Handler Address, 38
BIOS function I Warm Boot, 16
BIOS function
called by BDOS, 13
Home (8), 25
BIOS interface, 39
BIOS internal variables, 15
BIOS register usage, 14
BIOS write operation, 47
BLM, 43
Block Mask, 43
block number
largest allowed, 44
Block Shift Factor, 42
block storage, 2
BLS, 44
BLS bytes, 48
boot disk, 11, 49
boot tracks, 43
boot
warm, 47
bootstrap loader, 6
  machine dependent, 43
bootstrap procedure, 9
bootstrapping loading, 9
BSH, 42
bss, 2
buffer
  writing to disk, 47
built-in user commands, 4
byte, 1
byte (8 bit) value, 42

C
C language, 39
carriage return, 19
CBASE feature, 51
CCP, 3, 4, 6, 7, 50
CCP entry point, 16
character devices, 5
checksum vector, 41
CKS, 43
Cold Boot Automatic Command
  Execution, 51
Cold Boot Loader, 7
Cold Boot Loader
  creating, 10
cold start, 6
communication protocol, 20
configuration requirements, 49
Conout, 10
CONSOLE device, 33
CP/M-68K
  customizing, 7
generating, 7
  installing, 49
  loading, 49
  logical device
    characteristics, 33
    system modules, 3
CP/M-68K configuration, 39
CP/M-68K file structure, 1
CP/M-68K programming model, 2
CPM.REL, 7
CPM.SYS
  creating, 7
CPM.SYS, 6, 9
CPM.SYS file, 51
CPMLDR, 9
CPMLDR.SYS, 10
  building, 11
CPMLIB, 7
CSV, 41
CTRL-Z (1AH), 5
D
data segment, 2
device models
  logical, 5
DIRBUF, 40
directory buffer, 11
directory check vector, 43
disk, 6
disk access
  sequential, 46
disk buffers
  writing, 37
disk definition tables, 39
disk devices, 6
disk drive
  total storage capacity, 43
disk head, 23
Disk Parameter Block (DPB), 11, 13, 24, 42, 43
Disk Parameter Block fields, 42
Disk Parameter Header (DPH), 11, 13, 24, 31, 40
Disk Parameter Header elements, 40, 41
disk select operation, 24
disk throughput, 46
disk writes, 37
DMA address, 27
DMA buffer, 29
DPB, 40
DRM, 43
DSM, 43, 44
E
end-of-file, 5
end-of-file condition, 22
error indicator, 24
ESM, 44
exception vector area, 1, 38
EXORmacs, 49
Extent Mask, 43
F
FDC, 49
file storage, 6
file system tracks, 43
Function 0, 10
G

Get MRT, 11
graphics device
   bit-mapped, 4

I

I/O byte, 34
I/O byte field definitions, 34
I/O character, 5
I/O devices
   character, 5
   disk drives, 5
   disk file, 5
Init, 10
interface
   hardware, 5
interrupt vector area, 3

J

jsr _init, 15

L

L068 command, 7
LDRLIB, 10
line-feed, 19
list device, 20
LIST device, 33
Loader BIOS
   writing, 10
loader system library, 10
logical sector numbering, 41
longword (32-bit) value, 40
longword value, 1, 15
LRU buffers, 48

M

MACSbug, 49
mapping
   logical to physical, 41
maximum track number
   65535, 25
memory location
   absolute, 7
Memory Region Table, 32
mopping
   logical-to-physical, 6
Motorola MC68000, 1

N

nibble, 1

O

OFF parameter, 43, 53
offset, 1
output device
   auxiliary, 21

P

parsing
   command lines, 4
physical sector, 46
PIP, 35
PUTBOOT utility, 10, 11, 53

R

Read, 11
read/write head, 45
README file, 50
register contents
   destroyed by BIOS, 13
RELOC utility, 7
relocatable, 2
reserved tracks
   number of, 43
return code value, 28
rotational latency, 41, 45, 47
RTE, 10
rts instruction, 15

S

S-record files, 49
S-record systems, 50
S-records
   bringing up CP/M-68K, 50
   longword location, 50
scratchpad area, 40
scratchpad words, 40
sector, 5
sector numbers
   unskewed, 26
sector skewing, 53
sector translate table, 41
sectors
   128-byte, 5, 45
Sectran, 11
Seldsk, 10
Set exception, 11
Setdma, 11
Setsec, 11
Settrk, 11
SETTRK function, 23
SIZE68 command, 7, 8
SPT, 42
SPT parameter, 53
STAT, 35
system disk, 6
system generation, 6
T
text segment, 2
TPA, 1
track, 6
track 00 position, 23
transient program, 2
translate table, 31
Trap 3 handler, 10
TRAP 3 instruction, 13
Trap 3 vector, 15
trap initialization, 10
turn-key systems, 51
U
UDC, 49
user interface, 4
W
warm boot, 47
word, 1
word (16-bit) value, 40, 42
word references, 36
X
XLT, 40
Reader Comment Card

We welcome your comments and suggestions. They help us provide you with better product documentation.

Date __________ First Edition: January 1983

1. What sections of this manual are especially helpful?

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

2. What suggestions do you have for improving this manual? What information is missing or incomplete? Where are examples needed?

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

3. Did you find errors in this manual? (Specify section and page number.)

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

CP/M-68K™ Operating System System Guide

COMMENTS AND SUGGESTIONS BECOME THE PROPERTY OF DIGITAL RESEARCH.
The following listings are omitted from Appendix D of the CP/M-68K™ Operating System System Guide. Insert them in your System Guide after the following pages.

**PAGE 73**

Insert page 73a after page 73 in Appendix D.

**PAGE 78**

Insert page 78a after page 78 in Appendix D.

**PAGE 81**

Insert page 81a after page 81 in Appendix D.
/*******************************************************************************/

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

/* Port Addresses */
#define PORT1 0xFEE011 /* console port */
#define PORT2 0xFEE015 /* debug port */

/* Port Offsets */
#define PORTCTRL 0 /* Control Register */
#define PORTSTAT 0 /* Status Register */
#define PORTMR 2 /* Read Data Register */
#define PORTWR 2 /* Write Data Register */

/* Port Control Functions */
#define PORTSET 1 /* Port Reset */
#define PORTINIT 0x11 /* Port Initialize */

/* Port Status Values */
#define PORTDRF 1 /* Read Data Register Full */
#define PORTER 2 /* Write Data Register Empty */

/******************************************************************************/

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

#define DSKRTEC 0xFFE000 /* IPC Base Address */
#define DSKINTV 0x3FC /* Address of Disk Interrupt Vector */
#define INTDIP 0x00 /* offsets in mem mapped io area */
#define MSGTOIPC 0x101
#define MSGTOIPC 0x103
#define MSGTOIPC 0x105
#define MSGTOIPC 0x101
#define MSGTOIPC 0x103
#define MSGTOIPC 0x105
#define DSKREAD 0x10 /* disk commands */
#define DSKWRITE 0x20

/* Some characters used in disk controller packets */
#define STX 0x02
#define ETX 0x03
#define ACK 0x06

Listing D-1. (continued)
define NUMDSK 4   /* number of disks defined */
define MAXDSK (NUMDSK-1)   /* maximum disk number */

BYTE convdisk[NUMDSK] = { 4, 5, 0, 1 };   /* convert CP/M disk to EXORmacs */
BYTE convdisk[6] = { 2, 3, 0, 0, 0, 1 };   /* and vice versa */

/* defines for I/O and disk states */
define IDLE 0
#define ACTIVE 1

WORD inactive;   /* current I/O state */
WORD actdisk;    /* disk number of currently active disk, if any */
LONG intcount;   /* count of interrupts needing to be processed */

struct disk {
  WORD state;   /* from defines above */
  BYTE ready;   /* 0 => not ready */
  BYTE change;  /* 0 => no change */
}

diskstate[NUMDSK];

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

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

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

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

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

Listing D-1. (continued)
All Information Presented Here is Proprietary to Digital Research
NOTICE TO USER – PLEASE READ THIS NOTICE CAREFULLY. DO NOT OPEN THE DISKETTE PACKAGE UNTIL YOU HAVE READ THIS LICENSE AGREEMENT.

OPENING THE DISKETTE PACKAGE INDICATES YOUR AGREEMENT TO BE BOUND BY THESE TERMS AND CONDITIONS. IF YOU DO NOT ACCEPT THESE TERMS AND CONDITIONS, YOU MUST PROMPTLY RETURN THE PACKAGE UNOPENED TO THE PLACE OF ACQUISITION AND YOUR MONEY WILL BE REFUNDED.

1. DEFINITIONS

- In this License Agreement, the terms:

1. DRI means DIGITAL RESEARCH (CALIFORNIA) INC., P.O. Box 579, Pacific Grove, California 93950, owner of the copyright in, or authorized licensor of, the program.

2. Machine means the single microcomputer on which you use the program. Multiple CPU systems require additional licenses.

3. Program means the set of programs, documentation and related materials in this package, together with all ancillary updates and enhancements supplied by DRI to you regardless of the form in which you may subsequently use it, and regardless of any modification which you make to it.

4. AUTHOR means any third party author and owner of the copyright in this program.

You assume responsibility for the selection of the program to achieve your intended results, and for the installation, use and results obtained from the program.

2. LICENSE

You may:

1. Use the program on a single machine;

2. Copy the program into any machine readable or printed form for backup or modification purposes in support of your use of the program on a single machine. You may make up to three (3) copies of the program for such purposes. (Certain programs, however, may include mechanisms to limit or inhibit copying. They are marked "copy protected.") Copying of documentation and other printed materials is prohibited;

3. Modify the program and/or merge it into another program for your use on the single machine (Any portion of this program merged into another program will continue to be subject to the terms and conditions of this Agreement); and,
4. Transfer the program and license to another party if you notify DRI of name and address of the other party and the other party agrees to a) accept the terms and conditions of this Agreement, b) sign and forward to DRI a copy of the registration card and c) pay the then current transfer fee. If you transfer the program, you must at the same time either transfer all copies, including the original, whether in printed or machine readable form to the same party or destroy any copies not transferred; this includes all modifications and portions of the program contained or merged into other programs.

You must reproduce and include the copyright notice on any copy, modification or portion merged into another program.

YOU MAY NOT USE, COPY, MODIFY, TRANSFER, OR OTHERWISE MAKE AVAILABLE TO ANY THIRD PARTY, THE PROGRAM, OR ANY COPY, MODIFICATION OR MERGED PORTION, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE AGREEMENT.

IF YOU TRANSFER POSSESSION OF ANY COPY, MODIFICATION OR MERGED PORTION OF THE PROGRAM TO ANOTHER PARTY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

3. TERM

The license is effective until terminated. You may terminate it at any other time by destroying the program together with all copies, modifications and merged portions in any form. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the program together with all copies, modifications and merged portions in any form.

4. LIMITED WARRANTY

THE PROGRAM IS PROVIDED "AS IS". NEITHER DRI NOR AUTHOR MAKE ANY WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT DRI OR AUTHOR) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY BY STATE OR JURISDICTION.

Neither DRI nor AUTHOR warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, DRI warrants the diskette(s) on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.
END USER PROGRAM LICENSE AGREEMENT

5. LIMITATIONS OF REMEDIES

DRI's entire liability and your exclusive remedy shall be:

1. The replacement of any diskette not meeting DRI's "Limited Warranty" and which is returned to DRI or your place of acquisition with a copy of your receipt, or

2. If DRI or the place of acquisition is unable to deliver a replacement diskette which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT SHALL DRI OR AUTHOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, EVEN IF DRI OR AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES AND JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

6. REGISTRATION CARD

DRI may from time to time update its programs. Updates will be provided to you only if a properly signed registration card is on file at DRI's main office or an authorized registration card recipient. DRI is not obligated to make any program updates, or to supply any such updates to you.

7. GENERAL

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of California, except as to matters of copyright where this Agreement shall be governed by U.S. federal law. You and DRI hereby agree to the non-exclusive jurisdiction of the courts in the State of California.

Should you have any questions concerning this Agreement, you may contact DRI by writing to Digital Research Inc., P.O. Box 579, Pacific Grove, California 93950.

THIS AGREEMENT CANNOT AND SHALL NOT BE MODIFIED BY PURCHASE ORDERS, ADVERTISING OR OTHER REPRESENTATIONS BY ANYONE, AND MAY ONLY BE MODIFIED BY A WRITTEN AMENDMENT EXECUTED BY YOU AND AN AUTHORIZED OFFICER OF DRI.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN YOU AND DRI WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY COMMUNICATIONS BETWEEN YOU AND DRI RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.