

CP/M-68K™
Operating System
System Guide

COPYRIGHT

Copyright © 1983 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California, 93950.

DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

TRADEMARKS

CP/M and CP/M-86 are registered trademarks of Digital Research. CP/M-80, CP/M-68K, DDT, and MP/M are trademarks of Digital Research. Motorola MC68000 is a registered trademark of Motorola, Incorporated. EXORMacs, EXORterm, and MACSbug are trademarks of Motorola, Inc. VAX/VMS is a trademark of Digital Equipment Corporation. UNIX is a trademark of Bell Laboratories. TI Silent 700 Terminal is a registered trademark of Texas Instruments, Incorporated.

The CP/M-68K Operating System System Guide was prepared using the Digital Research TEX Text Formatter and printed in the United States of America.

* First Edition: January 1983 *

Foreword

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

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

Purpose and Audience

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

Prerequisites and Related Publications

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

How This Book is Organized

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

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

Table of Contents

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
-----	------------------------	----

Table of Contents (continued)

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	105
F	The Motorola S-record Format	110
	F.1 S-record Format	110
	F.2 S-record Types	111
G	CP/M-68K Error Messages	113
H	VME/10 BIOS Written in C	123

Tables, Figures, and Listings

Tables

1-1.	CP/M-68K Terms	1
4-1.	BIOS Register Usage	14
4-2.	BIOS Functions	14
4-3.	CP/M-68K Logical Device Characteristics	33
4-4.	I/O Byte Field Definitions	34
5-1.	Disk Parameter Header Elements	40
5-2.	Disk Parameter Block Fields	42
5-3.	BSH and BLM Values	44
5-4.	EXM Values	45
A-1.	Distribution Disk Contents	55
F-1.	S-Record Field Contents	110
F-2.	S-Record Types	112
G-1.	CP/M-68K Error Messages	113

Figures

1-1.	CP/M-68K Interfaces	3
1-2.	Typical CP/M-68K Memory Layout	4
4-1.	Memory Region Table Format	32
4-2.	I/O Byte Fields	34
5-1.	Disk Parameter Header	40
5-2.	Sample Sector Translate Table	42
5-3.	Disk Parameter Block	42
F-1.	G-Reference Fields	107

Listings

B-1.	Sample Assembly Language BIOS	59
C-1.	Sample BIOS Loader	67
D-1.	EXORMacs BIOS.C File	73
D-2.	EXORMacs BIOSTYPS.H File	99
D-3.	EXORMacs NORMBIOS.H File	99
D-4.	EXORMacs LOADBIO.S File	99
D-5.	EXORMacs BIOSA.S File	100
D-6.	EXORMacs LDBIO.S File	102
D-7.	EXORMacs BOOTER.S File	103

Tables, Figures, and Listings (continued)

Listings (continued)

E-1.	PUTBOOT Assembly Language Source	105
H-1.	VME/10 BIOS.C File	123
H-2.	VME/10 VT52.C File	177
H-3.	VME/10 BIOSTYPS.H File	185
H-4.	VME/10 NORMBIOS.H File	186
H-5.	VME/10 LOADBIOS.H File	187
H-6.	VME/10 BIOSA.S File	188
H-7.	VME/10 LDBIOSA.S File	191
H-8.	VME/10 BOOTER.S File	193

Section 1

System Overview

1.1 Introduction

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

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

The entire CP/M-68K operating system resides in memory at all times, and is not reloaded at a warm start. CP/M-68K can be configured to reside in any portion of memory above the 68000 exception vector area (0H to 3FFH). The remainder of the address space is available for applications programs, and is called the transient program area, TPA.

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

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

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

Table 1-1. (continued)

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

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

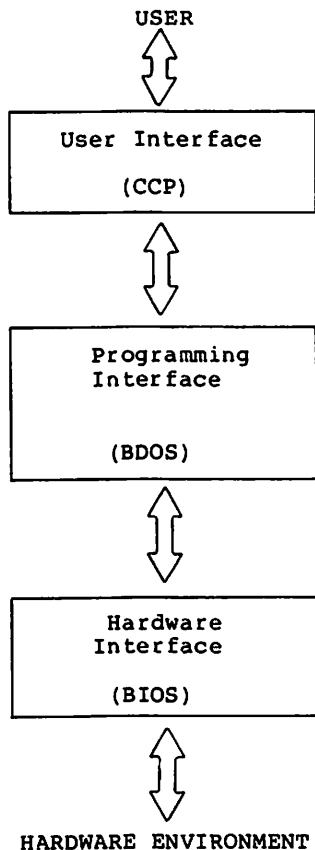


Figure 1-1. CP/M-68K Interfaces

1.2 CP/M-68K Organization

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

1.3 Memory Layout

The CP/M-68K operating system can reside anywhere in memory except in the interrupt vector area (0H to 3FFH). The location of CP/M-68K is defined during system generation. Usually, the CP/M-68K operating system is placed at the top end (high address) of available memory, and the TPA runs from 400H to the base of the

operating system. It is possible, however, to have other organizations for memory. For example, CP/M-68K could go in the low part of memory with the TPA above it. CP/M-68K could even be placed in the middle of available memory.

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

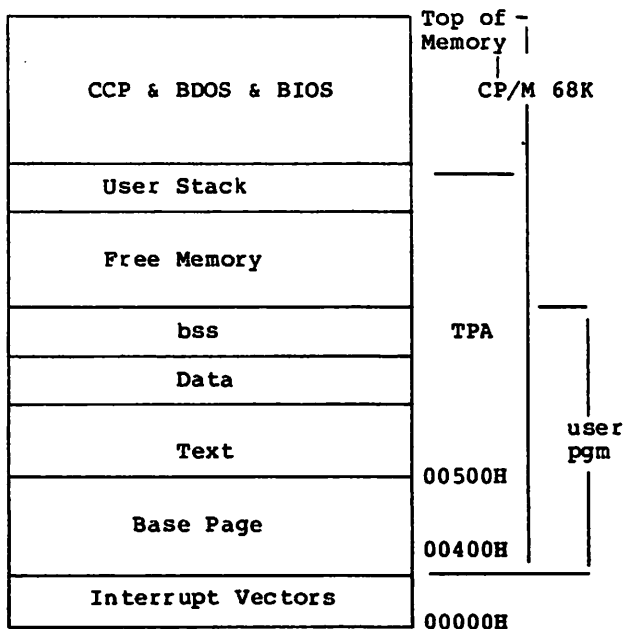


Figure 1-2. Typical CP/M-68K Memory Layout

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 (1AH) to indicate end-of-file.

1.7.2 Character Devices

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

Section 5.) To locate a particular sector, the disk, track number, and sector number must all be specified.

1.8 System Generation and Cold Start Operation

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

End of Section 1

Section 2

System Generation

2.1 Overview

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

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

A CP/M-68K operating system is generated by using the linker, 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)

- 11 (Setsec)
 - 12 (Setdma)
 - 13 (Read)
 - 16 (Sectran)
 - 18 (Get MRT) Not used now, but may be used in future releases.
 - 22 (Set exception)
- 4) You do not need to include an allocation vector or a check vector, and the Disk Parameter Header values that point to these can be anything. However, you still need a Disk Parameter Header, Disk Parameter Block, and directory buffer.

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

3.2.2 Building CPMLDR.SYS

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

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

```
A>lo68 -s -T400 -uldr -o cpmlldr.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 cpmlldr.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

Entry Parameters:	
D0.W	= function code
D1.x	= first parameter
D2.x	= second parameter
Return Values:	
D0.B	= byte values (8 bits)
D0.W	= word values (16 bits)
D0.L	= longword values (32 bits)

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

Table 4-2. BIOS Functions

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

FUNCTION 0: INITIALIZATION
Entry Parameters: Register D0.W: 00H
Returned Value: Register D0.W: User/Disk Numbers

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

Function 0 returns a longword value. The CCP uses this value to set the initial user number and the initial default disk drive. The least significant byte of 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 D0.W: 01H
Returned Value: None

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

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

```
        .globl    _ccp

wboot:
* do any reinitialization here if necessary
        jmp      _ccp
```


FUNCTION 2: CONSOLE STATUS	
Entry Parameters:	Register D0.W: 02H
Returned Value:	Register D0.W: 00FFH if ready Register D0.W: 0000H if not ready

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

FUNCTION 3: READ CONSOLE CHARACTER
Entry Parameters: Register D0.W: 03H
Returned Value: Register D0.W: Character

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

FUNCTION 4: WRITE CONSOLE CHARACTER	
Entry Parameters:	
Register D0.W: 04H	
Register D1.W: Character	
Returned	Value: None

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	
Entry Parameters:	
Register D0.W: 05H	
Register D1.W: Character	
Returned	Value: None

This function sends an ASCII character from register D1 to the currently assigned listing device. If your list device requires some communication protocol, it must be handled here.

FUNCTION 6: AUXILIARY OUTPUT	
Entry Parameters:	
Register D0.W:	06H
Register D1.W:	Character
Returned Value:	
Register D0.W:	Character

This function sends an ASCII character from register D1 to the currently assigned auxiliary output device.

FUNCTION 7: AUXILIARY INPUT
Entry Parameters: Register D0.W: 07H
Returned Value: Register D0.W: Character

This function reads the next character from the currently assigned auxiliary input device into register D0. It reports an end-of-file condition by returning an ASCII CTRL-Z (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	
Entry Parameters:	
Register D0.W:	09H
Register D1.B:	Disk Drive
Register D2.B:	Logged in Flag
Returned Value:	
Register D0.L:	Address of Selected Drive's DPH

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

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

If there is an attempt to select a nonexistent drive, this function returns 00000000H in register D0.L as an error indicator. Although the function must return the header address on each call, it may be advisable to postpone the actual physical disk select operation until an I/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 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: 0DH
Returned Value:	Register D0.W: 0 if no error Register D0.W: 1 if physical error

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

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

FUNCTION 14: WRITE SECTOR	
Entry Parameters:	
Register D0.W:	0EH
Register D1.W:	0=normal write
	1=write to a directory sector
	2=write to first sector of new block
Returned Value:	
Register D0.W:	0=no error
	1=physical error

This function is used to write 128 bytes of data from the currently selected DMA buffer to the currently selected sector, track, and disk. The value in register 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	
Entry Parameters:	Register D0.W: 0FH
Returned Value:	Register D0: 00FFH=device ready Register D0: 0000H=device not ready

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

FUNCTION 16: SECTOR TRANSLATE	
Entry Parameters:	
Register D0.W: 10H	
Register D1.W: Logical Sector Number	
Register D2.L: Address of Translate Table	
Returned	Value: Register D0.W: Physical Sector Number

This function performs logical-to-physical sector translation, as described in Section 5.2.2. Function 16 receives a logical sector number from register D1.W. The logical sector number can range from 0 to the number of sectors per track minus 1. Function 16 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 the value in register D2.L is equal to 00000000H, indicating that there is no translate table, register D1 is copied to register D0 before the function returns. Note that other algorithms are possible. It is common to increment the logical sector number to convert the logical sector range (0 to n-1) into the physical range of 1 to n.

Function 16 is always called by the BDOS, whether the translate table address in the Disk Parameter Header is zero or non-zero.

FUNCTION 18: GET ADDRESS OF MEMORY REGION TABLE	
Entry Parameters: Register D0.W: 12H	
Returned	Value: Register D0.W: Memory Region Table Address

This function returns the address of the Memory Region Table (MRT) in register D0. CP/M-68K maintains the MRT for compatibility with other CP/M systems. However, the CP/M-68K MRT contains only one region, the Transient Program Area (TPA). The MRT is required for operation and must begin on an even address. The format of the MRT is shown in Figure 4-1.

Entry Count = 1	16 bits
Base address of first region	32 bits
Length of first region	32 bits

Figure 4-1. Memory Region Table Format

FUNCTION 19: GET I/O BYTE	
Entry Parameters: Register D0.W: 13H	
Returned Value: Register D0.W: I/O Byte Current Value	

This function returns the current value of the logical to physical input/output device byte (I/O byte) in register 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

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

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

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

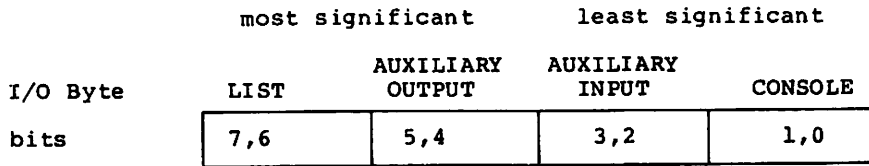


Figure 4-3. I/O Byte

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

Table 4-4. I/O Byte Field Definitions

CONSOLE field (bits 1,0)	
Bit	Definition
0	console is assigned to the console printer (TTY:)
1	console is assigned to the CRT device (CRT:)
2	batch mode: use the AUXILIARY INPUT as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
3	user defined console device (UC1:)
AUXILIARY INPUT field (bits 3,2)	
Bit	Definition
0	AUXILIARY INPUT is the Teletype device (TTY:)
1	AUXILIARY INPUT is the high-speed reader device (PTR:)
2	user defined reader #1 (UR1:)
3	user defined reader #2 (UR2:)

Table 4-4. (continued)

AUXILIARY OUTPUT field (bits 5,4)	
Bit	Definition
0	AUXILIARY OUTPUT is the Teletype device (TTY:)
1	AUXILIARY OUTPUT is the high-speed punch device (PTP:)
2	user defined punch #1 (UP1:)
3	user defined punch #2 (UP2:)
LIST field (bits 7,6)	
Bit	Definition
0	LIST is the Teletype device (TTY:)
1	LIST is the CRT device (CRT:)
2	LIST is the line printer device (LPT:)
3	user defined list device (UL1:)

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

FUNCTION 20: SET I/O BYTE
Entry Parameters: Register D0.W: 14H Register D1.W: Desired
Returned Value: None

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

FUNCTION 21: FLUSH BUFFERS	
Entry Parameters:	Register D0.W: 15H
Returned Value:	Register D0.W: 0000H=successful write Register D0.W: FFFFH=unsuccessful write

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

FUNCTION 22: SET EXCEPTION HANDLER ADDRESS	
Entry Parameters:	
Register D0.W:	16H
Register D1.W:	Exception Vector Number
Register D2.L:	Exception Vector Address
Returned Value:	
Register D0.L:	Previous Vector Contents

This function sets the exception vector indicated in register 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.

End of Section 4

Section 5

Creating a BIOS

5.1 Overview

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

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

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

Figure 5-1. Disk Parameter Header

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

Table 5-1. Disk Parameter Header Elements

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

Table 5-1. (continued)

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

5.2.2 Sector Translate Table

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

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

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

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

Figure 5-2. Sample Sector Translate Table

5.2.3 Disk Parameter Block

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

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

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

Figure 5-3. Disk Parameter Block

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

Table 5-2. Disk Parameter Block Fields

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

Table 5-2. (continued)

Field	Definition
BLM	The block mask which is determined by the data block allocation size, as shown in Table 5-3.
EXM	The extent mask, determined by the data block allocation size and the number of disk blocks, as shown in Table 5-4.
0	Reserved byte.
DSM	Determines the total storage capacity of the disk drive and is the number of the last block, counting from 0. That is, the disk contains DSM+1 blocks.
DRM	Determines the total number of directory entries which can be stored on this drive. DRM is the number of the last directory entry, counting from 0. That is, the disk contains DRM+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.
CKS	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.
OFF	The number of reserved tracks at the beginning of a logical disk. This is the number of the track on which the directory begins.

To choose appropriate values for the Disk Parameter Block elements, you must understand how disk space is organized in CP/M-68K. A CP/M-68K disk has two major areas: the boot or system tracks, and the file system tracks. The boot tracks are usually used to hold a machine-dependent bootstrap loader for the operating system. They consist of tracks 0 to 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 $32768 * \text{BLS}$. Because the largest allowable value for BLS is 16384, the biggest disk that can be accessed by CP/M-68K is $16384 * 32768 = 512 \text{ Mbytes}$.

Each directory entry may contain either 8 block numbers (if DSM ≥ 256) or 16 block numbers (if DSM < 256). Each file needs enough directory entries to hold the block numbers of all blocks allocated to the 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 5-3. BSH and BLM Values

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

Table 5-4. EXM Values

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

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

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

5.3 Disk Blocking

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

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

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

5.3.1 A Simple Approach

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

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

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

5.3.2 Some Refinements

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

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

5.3.3 Track Buffering

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

5.3.4 LRU Replacement

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

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

- 1) A BIOS Write operation with mode=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 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 VME/10..development system. If you have a VME/10, read Section 6.2, which describes how to load CP/M-68K from the distribution disks.

If you do not have a VME/10, you can load CP/M-68K using the S-record files supplied on your distribution disks. Section 6.2 describes this procedure.

6.2 Booting CP/M-68K on a VME/10

The CP/M-68K product disks distributed by Digital Research include a boot disk to run CP/M-68K on the Motorola VME/10. If you have a VME/10 and the following specifications, you can use the boot disk as distributed. No modification is required.

- a minimum 384K of Random Access Memory
- a Winchester Disk Controller (WDC)

Use the following steps to load CP/M-68K from the boot disk.

- 1) Place the boot disk in the first floppy disk drive (#F002).
- 2) Press SYSTEM RESET (front panel) and RETURN. This brings in TENbug.
- 3) Type BO 2 and press RETURN. CP/M-68K should boot and begin to run.

6.3 Booting CP/M-68K Using S-record Files

Your CP/M-68K distribution disks contain two copies of CP/M-68K in the Motorola S-record format. The S-record format is a method of representing binary memory images using ASCII characters. Refer to Appendix F for a detailed explanation of the S-record format. Both copies of CP/M-68K in S-record format contain the CCP and BDOS, but no BIOS. You must already have a custom designed BIOS for your computer. One of the S-record systems, CPM400.SK, is designed to occupy locations 400H and up. The other, CPM15000.SR, is configured to occupy the top of a 128K memory space. The following items are required to boot CP/M-68K using the S-record files:

- A method of down-loading absolute data to your target computer system.
- A computer capable of reading the distribution disks. (A CP/M based computer that supports standard CP/M 8-inch diskettes.)
- A BIOS custom designed for your target computer system.

Do the following steps to load CP/M-68K using the S-record files:

- 1) Choose one of the S-record files to work with, CPM400.SR or CPM15000.SR. Examine the corresponding symbol table file, CPM400.MAP or CPM15000.MAP, to find the symbol `_init`. The symbol `_init` has a dual significance. First, it is the address at which the S-record system expects to find your custom BIOS `_init` entry point. Second, it marks the first free memory location following the S-record system. That is, your BIOS may occupy memory locations from `_init` on up.
- 2) Build your custom BIOS, using the `_init` value found in step 1.
- 3) Down-load the S-record system to your target computer memory.
- 4) Down-load the custom BIOS to your target computer memory.
- 5) You can now execute instructions at the first location of the down-loaded system.

With a working CP/M-68K system, you can use the programming tools provided with the operating 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

The following table describes all the files on the CP/M-68K product disks distributed by Digital Research.

Table A-1. Distribution Disk Contents

File	Contents
AR68.REL	Relocatable version of the archiver/librarian.
AS68INIT	Initialization file for assembler. (See AS68 in <u>CP/M-68K Operating System Programmer's Guide.</u>)
AS68.REL	Relocatable version of the assembler.
ASM.SUB	Submit file to assemble programs. Source program must have a .S filetype. Assembler creates an object file with filetype .O and a listing file with filetype .PRN.
BIOS.O	Object file of BIOS for VME/10.
BIOS.C	C language source file for VME/10 BIOS as distributed with CP/M-68K.
BIOSA.O	Object file for assembly portion of the VME/10 BIOS.
BIOSA.S	Source file for the assembly portion of the VME/10 BIOS as distributed with CP/M-68K.
BIOSTYPS.H	Include file for use with BIOS.C.
BOOTER.O	Object file for VME/10 bootstrap.
BOOTER.S	Assembly code for VME/10 bootstrap.
C.SUB	Submit file to use C compiler. The file invokes all three compiler passes plus the assembler. Use the following general command to compile a C program. A>C filename
C068.REL	Relocatable version of the C parser.
C168.REL	Relocatable version of the C code generator.
CLIB	The C run-time library.

Table A-1. (continued)

File	Contents
CLINK.SUB	Submit file for linking C object programs with the C run-time library.
CP68.REL	Relocatable version of the C preprocessor.
CPM.H	C program include file containing C definitions for CP/M-68K. Refer to the <u>C Language Programming Guide for CP/M-68K</u> .
CPM.REL	Relocatable version of CP/M-68K system.
CPM.SYS	CP/M-68K operating system file for the VME/10.
CPMLIB	Library of object files for CP/M-68K. See Section 2.
CPMLDR.SYS	The bootstrap loader for the VME/10. The PUTBOOT utility writes this file on the system tracks.
CTYPE.H	C program include file containing ASCII character classification routines. Refer to the <u>C Language Programming Guide for CP/M-68K</u> .
DDT.REL	Relocatable version of the preloader for DDT™ (Loads DDT1 into the high end of the TPA.)
DDT1.68K	Debugger program that is loaded by DDT.REL. It must be relocated to the top of the TPA every time it is used. It is a relocatable file although the filetype is .68K.
DUMP.REL	Relocatable version of the dump utility.
ED.REL	Relocatable version of the ED™ utility.
ELDBIOS.S	Assembly language source for the ERG sample BIOS loader.
ERGBIOS.S	Assembly language source for the ERG sample BIOS.
ERRNO.H	C program include file containing macro definitions for the C perror function. Refer to the <u>C Language Programming Guide for CP/M-68K</u> .
FIND.REL	FIND utility that locates and prints all occurrences of a specified string in one or more specified files.

Table A-1. (continue)

File	Contents
FORMAT.REL	Relocatable disk formatter for the VME/10.
FORMAT.S	Assembly source file for the FORMAT utility.
INIT.REL	Relocatable version of the INIT utility.
INIT.S	Assembly source file for the INIT utility.
LCPM.SUB	Submit file to create CPM.REL for VME/10.
LDBIOS.O	Object file of BIOS loader for VME/10.
LDBIOSA.S	Source for the assembly portion of the VME/10 BIOS loader as distributed with CP/M-68K.
LDRLIB	Library of object files for creating a bootstrap loader. (See Section 3.)
LINK68.REL	Relocatable version of the LINK68 linker.
LO68.REL	Relocatable version of the LO68 linker.
LOADBIOS.H	Include file for use with BIOS.C, to create BIOS loader.
LOADBIOS.SUB	Submit file to create BIOS loader for VME/10.
LOADR.O	Overlay loader object file for LINK68.
MAKELDR.SUB	Submit file to create CPMLDR.SYS on VME/10.
NORMBIOS.H	Include file for use with BIOS.C to create a normal BIOS.
NORMBIOS.SUB	Submit file to create normal BIOS for VME/10.
NM68.REL	Relocatable version of the symbol table dump utility.
OVLHDLR.O	Overlay handler object file for LINK68.
PIP.REL	Relocatable version of the PIP utility.
PORTAB.H	C program include file containing portability macro definitions. Refer to the <u>C Language Programming Guide for CP/M-68K</u> .
PUTBOOT.REL	Relocatable version of the PUTBOOT utility.
PUTBOOT.S	Assembly source file for the PUTBOOT utility.

Table A-1. (continued)

File	Contents
README.TXT	Text file containing information relevant to this shipment of CP/M-68K. Some versions of CP/M-68K do not have this file.
RELCPM.SUB	Submit file to relocate CPM.REL into CPM.SYS.
RELOC.REL	Relocatable version of the command file relocation utility.
RELOCx.SUB	Submit file to relocate .REL files in the target system. Each distribution disk that contains .REL files has one of these files. The x stands for the distribution disk number.
S.O	Start-up routine for C programs. Must be the first object file linked in a C program.
SEND68.REL	Relocatable version of the S-record creation utility.
SETJMP.H	C program include file for the nonlocal jump routine. Refer to the <u>C Language Programming Guide for CP/M-68K</u> .
SIGNAL.H	C program include file for the C signal function. Refer to the <u>C Language Programming Guide for CP/M-68K</u> .
SIZE68.REL	Relocatable version of the SIZE68 utility.
CPM15000.SR	S-record version of CP/M-68K for porting the operating system to new hardware environments. This version is configured to occupy the top of a 128K memory space. It does not contain a BIOS.
CPM400.SR	S-record version of CP/M-68K for porting the operating system to new hardware environments. This version is configured to occupy locations 400H and up. It does not contain a BIOS.
STAT.REL	Relocatable version of the STAT utility.
STDIO.H	C program include file containing standard I/O macro definitions. Refer to the <u>C Language Programming Guide for CP/M-68K</u> .

End of Appendix A

Appendix B

Sample BIOS Written in Assembly Language

CP/M 68000 Assembler
Source File: a:ergbios.s

Revision 02.01

Page 1

```

1
2
3
4
5
6
7
8
9
10
11
12 00000000 23FC0000000E0000008C _init: move.l #trapnd1,S8c
13 0000000A 4280                clr.l d0
14 0000000C 4E75                rts
15
16
17 0000000E 0C400017        trapnd1: cmpi    #nfuncs,d0
18 00000012 6408                bcc    trapng
19 00000014 E548                lsl     #2,d0
20 00000016 207B0006        movea.l 6(pc,d0),a0
21 0000001A 4E90                jsr     (a0)
22
23 0000001C 4E73        trapng: rte
24
25
26 0000001E 00000000        biosbase: .dc.l _init
27 00000022 0000007A        .dc.l wboot
28 00000026 00000080        .dc.l constat
29 0000002A 00000094        .dc.l conin
30 0000002E 000000A8        .dc.l conout
31 00000032 000000BC        .dc.l listout
32 00000036 000000BE        .dc.l pun
33 0000003A 000000C0        .dc.l rdr
34 0000003E 000000C8        .dc.l home
35 00000042 000000D0        .dc.l seldsk
36 00000046 000000F8        .dc.l settrk
37 0000004A 00000100        .dc.l setsec
38 0000004E 00000114        .dc.l setdma
39 00000052 0000011C        .dc.l read
40 00000056 0000015E        .dc.l write
41 0000005A 000000C2        .dc.l listst

```

Listing B-1. Sample Assembly Language BIOS

```

42 0000005E 00000108      .dc.l  sectran
43 00000062 00000114      .dc.l  setdma
44 00000066 0000029C      .dc.l  getseg
45 0000006A 000002A4      .dc.l  getiob
46 0000006E 000002A6      .dc.l  setiob
47 00000072 00000298      .dc.l  flush
48 00000076 000002A8      .dc.l  setexc
49
50      nfuncs=(*-biosbase)/4
51
52 0000007A 4EF900000000    wboot:  jmp     _ccp
53
54 00000080 103900FFFF01    constat: move.b $ffff01,d0      * get status byte
55 00000086 02400002      andi.w $2,d0      * data available bit on?
C P / M   6 8 0 0 0   A s s e m b l e r      Revision 02.01      Page   2
Source File: aiergbios.s

56 0000008A 6704          beq     noton      * branch if not
57 0000008C 7001      moveq.l $51,d0      * set result to true
58 0000008E 4E75          rts
59
60 00000090 4280          noton:  clr.l  d0      * set result to false
61 00000092 4E75          rts
62
63 00000094 61EA          conin:  bsr     constat      * see if key pressed
64 00000096 4A40          tst     d0
65 00000098 67FA          beq     conin      * wait until key pressed
66 0000009A 103900FFFF00    move.b $ffff00,d0      * get key
67 0000009C C0BC0000007F    and.l  $57f,d0      * clear all but low 7 bits
68 0000009E 4E75          rts
69
70 000000A9 103900FFFF01    conout: move.b $ffff01,d0      * get status
71 000000AB C03C0001      and.b  $51,d0      * check for transmitter buffer empty
72 000000B2 67F4          beq     conout      * wait until our port has aged...
73 000000B4 13C100FFFF00    move.b dl,$ffff00      * and output it
74 000000B6 4E75          rts      * and exit
75
76 000000BC 4E75          lstout: rts
77
78 000000BE 4E75          pun:   rts
79
80 000000C0 4E75          rdc:   rts
81
82 000000C2 103C00FF      listst: move.b $fff,d0
83 000000C6 4E75          rts
84
85
86
87
88      * Disk Handlers for Tarbell 1793 floppy disk controller
89
90      maxdisk = 2      * this BIOS supports 2 floppy drives
91      dphlen = 26      * length of disk parameter header
92
93      iobase = $00ffff8      * Tarbell floppy disk port base address
94      dcmd = iobase      * output port for command

```

Listing B-1. (continued)


```

93      dstat = iobase      * input status port
94      dtrk  = iobase+1    * disk track port
95      dsect = iobase+2    * disk sector port
96      ddata = iobase+3    * disk data port
97      dwait = iobase+4    * input port to wait for op finished
98      dcntrl = iobase+4   * output control port for drive selection
99
100
101 000000C8 423900000002   home: ctr.b   track
102 000000CE 4E75          rts
103
104      seldsk:
105      *      select disk given by register dl.b
106 000000D0 7000          moveq  #0,d0
107 000000D2 B23C0002      cmp.b  #maxdsk,dl      * valid drive number?
108 000000D6 6A1E          bpl  selrtn      * if no, return 0 in d0
109 000000D8 13C100000000  move.b  dl,seldrv      * else, save drive number
110 000000DE E909          lsl.b  #4,dl
111 000000E0 13C10000000A  move.b  dl,selcode      * select code is 00 for drv 0, $10 for drv 1
112 000000E6 103900000000  move.b  seldrv,d0
113 000000EC C0FC001A      mulu  #dphlen,d0
114 000000F0 D0AC00000016  add.l  #dph0,d0      * point d0 at correct dph
115 000000F6 4E75          selrtn: rts
116
117 000000F8 13C1000000C02 settrk: move.b  dl,track
118 000000FE 4E75          rts
119
120 0000100 13C100000004   setsec: move.b  dl,sector
121 0000106 4E75          rts
122
123      sectran:
124      *      translate sector in dl with translate table pointed to by d2
125      *      result in d0
126 0000108 2042          movea.l d2,a0
127 000010A 48C1          ext.l  d1
128 000010C 10301000      move.b  #0(a0,d1),d0
129 0000110 48C0          ext.l  d0
130 0000112 4E75          rts
131
132      setdma:
133 0000114 23C100000006  move.l  d1,dma
134 000011A 4E75          rts
135
136      read:
137      * Read one sector from requested disk, track, sector to dma address
138      * Retry if necessary, return in d0 00 if ok, else non-zero
139 000011C 13FC000A0000000B move.b  #10,errcnt      * set up retry counter
140
141      rretry:
142      bsr  setup
143      ori  #$88,d3      * OR read command with head load bit
144      move.b d3,dcmd      * output it to FDC

```

CP/M 68000 Assembler
Source File: a:erghbios.s

Revision 02.01

Page 3

Listing B-1. (continued)

```

144 00000132 0839000700FFFFC      rloop:  btst    $7,dwait
145 0000013A 6708                  beq      rdone      * if end of read, exit
146 0000013C 10F900FFFFFB          move.b   ddata,(a0)+ * else, move next byte of data
147 00000142 60EE                  bra      rloop
148
149 00000144 61000146                  rdone:  bsr      rstatus    * get FDC status
150 00000148 6604                  bne      rerror
151 0000014A 4280                  clr.l    d0
152 0000014C 4E75                  rts
153 0000014E 610000B0          rerror:  bsr      errchk    * go to error handler
154 00000152 533900000000B      subq.b   $1,errcnt
155 00000158 66CA                  bne      rretry
156 0000015A 70FF                  move.l   $5fffffff,d0
157 0000015C 4E75                  rts
158
159
160
161
162 0000015E 13FC000A0000000B      write:  * Write one sector to requested disk, track, sector from dma address
163
164 00000166 6114                  * Retry if necessary, return in d0 00 if ok, else non-zero
165 00000168 004300A8          move.b   $10,errcnt  * set up retry counter
166
167 0000016E 6114          wretry:  bsr      setup
168 00000168 004300A8          ori      $5a8,d3      * OR write command with head load bit
169
170 0000017C 13D800FFFFFB          Revision 02.01      Page 4
171
172 00000184 61000106          C P / M 6 8 0 0 0 A s s e m b l e r
173 00000188 6604          Source File: a:ergbios.s
174 0000018A 4280
175 0000018C 4E75
176 0000018E 6170
177 00000190 533900000000B      wloop:  move.b   d3,dcmd    * output it to FDC
178 00000196 66C8                  btst     $7,dwait
179 00000198 70FF                  beq      wdone      * if end of read, exit
180 0000019A 4E75                  move.b   (a0)+,ddata * else, move next byte of data
181
182
183
184
185 0000019C 13FC00D000FFFFF8          wdone:  bsr      rstatus    * get FDC status
186 000001A4 1639000000001          bne      werror
187 000001AA B63900000000          clr.l    d0
188 000001B0 661A                  rts
189 000001B2 1639000000002          werror:  bsr      errchk    * go to error handler
190 000001B8 B639000000003      subq.b   $1,errcnt
191 000001BE 6620                  bne      wretry
192 000001C0 4283                  move.l   $5fffffff,d0
193 000001C2 08390000500FFFFF8      rts
194 000001CA 6618

```

* common read and write setup code
 * select disk, set track, set sector were all deferred until now
 move.b \$5d0,dcmd * Clear controller, get status
 move.b curdrv,d3
 cmp.b seldrv,d3
 bne newdrive * if drive not selected, do it
 move.b track,d3
 cmp.b oldtrk,d3
 bne newtrk * if not on right track, do it
 clr.l d3 * if head already loaded, no head load delay
 btst \$5,dstat * if head unloaded, treat as new disk
 bne sext

Listing B-1. (continued)

```

195
196 000001CC 13F90000000A00FFFFFFC newdrive:
197 000001DE 13F90000000000000001 move.b selcode,dctrl * select the drive
198 move.b seldrv,curdrv
199 000001E0 6126 newtrk: bsr chkseek * seek to correct track if required
200 000001E2 7604 moveq #4,d3 * force head load delay
201
202 000001E4 13F90000000400FFFFFFFA sexit: move.b sector,dsect * set up sector number
203 000001EE 13F90000000200FFFFFFF9 move.b track,dtrk * set up track number
204 000001F8 207900000006 move.l dma,a0 * dma address to a0
205 000001FE 4E75 rts
206
207 errchk:
208 00000200 08070004 btst #4,d7
209 00000204 6602 bne chkseek * if record not found error, reseek
210 00000206 4E75 rts
211
212 chkseek:
213 * check for correct track, seek if necessary
214 00000208 615C bsr readid * find out what track we're on
215 0000020A 671E beq chksl * if read id ok, skip restore code
216
217 restore:
218 * home the drive and reseek to correct track
219 0000020C 13FC000B00FFFFFFF8 move.b #$0B,dcmd * restore command to command port
220 00000214 0839000700FFFFFFFC rstwait: btst #7,dwait
CP/M 68000 Assembler Revision 02.01 Page 5
Source File: a:ergbios.s
221 0000021C 66F6 bne rstwait * loop until restore completed
222 0000021E 0839000200FFFFFFF8 btst #2,dstat
223 00000226 67E4 beq restore * if not at track 0, try again
224 00000228 4283 clr.l d3 * track number: returned in d3 from readid
225
226 0000022A 13C300FFFFFFF9 chksl: move.b d3,dtrk * update track register in FDC
227 00000230 13F90000000200000003 move.b track,oldtrk * update oldtrk
228 0000023A B63900000002 cmp.b track,d3 * are we at right track?
229 00000240 6722 beq chkdne * if yes, exit
230 00000242 13F90000000200FFFFFFF8 move.b track,ddata * else, put desired track in data reg of FDC
231 0000024C 13FC001800FFFFFFF8 move.b #$18,dcmd * and issue a seek command
232 00000254 0839000700FFFFFFFC chks2: btst #7,dwait
233 0000025C 66F6 bne chks2 * loop until seek complete
234 0000025E 163900FFFFFFF8 move.b dstat,d3 * read status to clear FDC
235
236 00000264 4E75 chkdne: rts
237
238 readid:
239 * read track id, return track number in d3
240 00000266 13FC00C400FFFFFFF8 move.b #$C4,dcmd * issue read id command
241 0000026E 1E3900FFFFFFFC move.b dwait,d7 * wait for intrq
242 00000274 163900FFFFFFFB move.b ddata,d3 * track byte to d3
243
244 0000027A 0839000700FFFFFFFC rid2: btst #7,dwait
245 00000282 6708 beq rstatus * wait for intrq

```

Listing B-1. (continued)

```

246 00000284 1E3900FFFFB      move.b  ddata,d7      * read another byte
247 0000028A 60EE              bra      rid2          * and loop
248
249 0000028C 1E3900FFFFB      rstatus: move.b  dstat,d7
250 00000292 0207009D      andi.b  $59d,d7      * set condition codes
251 00000296 4E75              rts
252
253
254
255 00000298 4280      flush:  clr.l  d0      * return successful
256 0000029A 4E75              rts
257
258
259 0000029C 203C0000000C      getseg:  move.l  $memrgh,d0      * return address of mem region table
260 000002A2 4E75              rts
261
262
263 000002A4 4E75      getiob:  rts
264
265
266 000002A6 4E75      setiob:  rts
267
268
269 000002A8 0281000000FF      setexc:  andi.l  $5ff,d1      * do only for exceptions 0 - 255
270 000002AE E549              lsl      #2,d1          * multiply exception nmbr by 4
271 000002B0 2041              movea.l  d1,a0
272 000002B2 2010              move.l  (a0),d0          * return old vector value
273 000002B4 2082              move.l  d2,(a0)          * insert new vector
274 000002B6 4E75      noset:  rts
275

```

CP/M 68000 Assembler Revision 02.01 Page 6
Source File: a:ergbios.s

```

276
277 00000000      .data
278
279 00000000 FF      seldrv: .dc.b  $ff      * drive requested by seldsk
280 00000001 FF      curdrv: .dc.b  $ff      * currently selected drive
281
282 00000002 00      track: .dc.b  0      * track requested by settrk
283 00000003 00      oldtrk: .dc.b  0      * track we were on
284
285 00000004 0000      sector: .dc.w  0
286 00000006 00000000      dma: .dc.l  0
287 0000000A 00      selcode: .dc.b  0      * drive select code
288
289 0000000B 0A      errcnt: .dc.b  10      * retry counter
290
291 0000000C 0001      memrgh: .dc.w  1      * 1 memory region
292 0000000E 00000400      .dc.l  $400      * starts at 400 hex
293 00000012 00017C00      .dc.l  $17c00      * goes until 18000 hex
294
295
296      * disk parameter headers

```

Listing B-1. (continued)

```

297
298 00000016 0000005A      dph0: .dc.l  xlt
299 0000001A 0000      .dc.w  0      * dummy
300 0000001C 0000      .dc.w  0
301 0000001E 0000      .dc.w  0
302 00000020 00000000      .dc.l  dirbuf  * ptr to directory buffer
303 00000024 0000004A      .dc.l  dpb      * ptr to disk parameter block
304 00000028 00000080      .dc.l  ckv0      * ptr to check vector
305 0000002C 000000A0      .dc.l  alv0      * ptr to allocation vector
306
307 00000030 0000005A      dph1: .dc.l  xlt
308 00000034 0000      .dc.w  0      * dummy
309 00000036 0000      .dc.w  0
310 00000038 0000      .dc.w  0
311 0000003A 00000000      .dc.l  dirbuf  * ptr to directory buffer
312 0000003E 0000004A      .dc.l  dpb      * ptr to disk parameter block
313 00000042 00000090      .dc.l  ckv1      * ptr to check vector
314 00000046 000000C0      .dc.l  alv1      * ptr to allocation vector
315
316      * disk parameter block
317
318 0000004A 001A      dpb: .dc.w  26      * sectors per track
319 0000004C 03      .dc.b  3      * block shift
320 0000004D 07      .dc.b  7      * block mask
321 0000004E 00      .dc.b  0      * extent mask
322 0000004F 00      .dc.b  0      * dummy fill
323 00000050 00F2      .dc.w  242      * disk size
324 00000052 003F      .dc.w  63      * 64 directory entries
325 00000054 C000      .dc.w  $C000      * directory mask
326 00000056 0010      .dc.w  16      * directory check size
327 00000058 0002      .dc.w  2      * track offset
328
329      * sector translate table
330
C P / M   6 8 0 0 0   A s s e m b l e r           Revision 02.01           Page   7
Source File: a:erqbios.s

331 0000005A 01070D13      xlt: .dc.b  1, 7,13,19
332 0000005E 19050B11      .dc.b  25, 5,11,17
333 00000062 1703090F      .dc.b  23, 3, 9,15
334 00000066 1502080E      .dc.b  21, 2, 8,14
335 0000006A 141A060C      .dc.b  20,26, 6,12
336 0000006E 1218040A      .dc.b  18,24, 4,10
337 00000072 1016      .dc.b  16,22
338
339
340 00000000      .bss
341
342 00000000      dirbuf: .ds.b  128      * directory buffer
343
344 00000080      ckv0: .ds.b  16      * check vector
345 00000090      ckv1: .ds.b  16
346
347 000000A0      alv0: .ds.b  32      * allocation vector

```

Listing B-1. (continued)

```

348 000000C0          alvl: .ds.b 32
349
350 000000E0          .end
CP/M 68000 Assembler      Revision 02.01      Page 8
Source File: a:ergbios.s
Symbol Table

```

ccp	***** EXT	_init	00000000 TEXT	alv0	000000A0 BSS	alvl	000000C0 BSS
biosbase	0000001E TEXT	chkdone	00000264 TEXT	chksl	0000022A TEXT	chks2	00000254 TEXT
chkseek	00000208 TEXT	ckv0	00000080 BSS	ckv1	00000090 BSS	conin	00000094 TEXT
conout	000000A8 TEXT	constat	00000080 TEXT	curdrv	00000001 DATA	dcmd	00FFFFFF8 ABS
dentrl	00FFFFFFC ABS	ddata	00FFFFFFB ABS	dirouf	00000000 BSS	dms	00000006 DATA
dpb	0000004A DATA	dph0	00000016 DATA	dph1	00000030 DATA	dphlen	0000001A ABS
dssect	00FFFFFFA ABS	dstat	00FFFFFF8 ABS	drk	00FFFFFF9 ABS	dwait	00FFFFFFC ABS
errchk	00000200 TEXT	errcnt	0000000B DATA	flush	00000298 TEXT	getiob	000002A4 TEXT
getseg	0000029C TEXT	home	000000C8 TEXT	ibase	00FFFFFF8 ABS	listst	000000C2 TEXT
lstout	0000008C TEXT	maxdsk	00000002 ABS	memrgn	0000000C DATA	newdrive	000001CC TEXT
newtrk	000001E0 TEXT	nfuncs	00000017 ABS	noset	000002B6 TEXT	noton	00000090 TEXT
oldtrk	00000003 DATA	pun	000000BE TEXT	rdone	00000144 TEXT	rdr	000000C0 TEXT
read	0000011C TEXT	readid	00000266 TEXT	rerror	0000014E TEXT	restore	0000020C TEXT
rid2	0000027A TEXT	rloop	00000132 TEXT	rretry	00000124 TEXT	rstatus	0000028C TEXT
stwait	00000214 TEXT	sector	00000004 DATA	sectran	00000108 TEXT	selcode	0000000A DATA
seldrv	00000000 DATA	seldsk	000000D0 TEXT	seltrn	000000F6 TEXT	setdma	00000114 TEXT
setexc	000002A8 TEXT	setiob	000002A6 TEXT	setsec	00000100 TEXT	settrk	000000F8 TEXT
setup	0000019C TEXT	sexit	000001E4 TEXT	track	00000002 DATA	traphndl	0000000E TEXT
trapng	0000001C TEXT	wboot	0000007A TEXT	wdone	00000184 TEXT	werror	0000018E TEXT
wloop	00000172 TEXT	wretry	00000166 TEXT	write	0000015E TEXT	xlt	0000005A DATA

Listing B-1. (continued)

End of Appendix B

Appendix C

Sample Loader BIOS Written in Assembly Language

CP/M 68000 Assembler
Source File: seldbios.s

Revision 02.01

Page 1

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14 00000000 0C400017
15 00000004 6C08
16 00000006 E548
17 00000008 207B0006
18 0000000C 4E90
19
20 0000000E 4E75
21
22
23 00000010 0000000E
24 00000014 0000000E
25 00000018 0000006C
26 0000001C 00000080
27 00000020 00000094
28 00000024 0000000E
29 00000028 0000000E
30 0000002C 0000000E
31 00000030 000000A8
32 00000034 000000B0
33 00000038 000000C4
34 0000003C 000000CC
35 00000040 000000E0
36 00000044 000000E8
37 00000048 0000000E
38 0000004C 0000000E
39 00000050 000000D4
40 00000054 000000E0
41 00000058 0000000E
42 0000005C 0000000E

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

        .globl _bios          * declare external entry point

_bios:
        cmpi    $nfuncs,d0
        bge     nogood
        lsl     #2,d0          * multiply bios function by 4
        movea.l 6(pc,d0),a0    * get handler address
        jsr     (a0)          * call handler

nogood:
        rts

biosbase:
        .dc.l   nogood
        .dc.l   nogood
        .dc.l   constat
        .dc.l   conin
        .dc.l   conout
        .dc.l   nogood
        .dc.l   nogood
        .dc.l   nogood
        .dc.l   home
        .dc.l   seldsk
        .dc.l   settck
        .dc.l   setsec
        .dc.l   setdma
        .dc.l   read
        .dc.l   nogood
        .dc.l   nogood
        .dc.l   sectran
        .dc.l   setdma
        .dc.l   nogood
        .dc.l   nogood

```

Listing C-1. Sample BIOS Loader

```

43 00000060 0000000E      .dc.l  nogood
44 00000064 0000000E      .dc.l  nogood
45 00000068 00000222      .dc.l  setexc
46
47      nfuncs=(*-biosbase)/4
48
49
50 0000006C 103903FFFF01    constat: move.b $ffff01,d0      * get status byte
51 00000072 02400002      andi.w  #2,d0      * data available bit on?
52 00000076 6704          beq  noton      * branch if not
53 00000078 7001          moveq.l #51,d0      * set result to true
54 0000007A 4E75          rts
55
C P / M   6 8 0 0 0   A s s e m b l e r           Revision 02.01           Page   2
Source File: a:eldbios.s

56 0000007C 4280          noton: clr.l  d0      * set result to false
57 0000007E 4E75          rts
58
59 00000080 61EA          conin: bsr  constat      * see if key pressed
60 00000082 4A40          tst  d0
61 00000084 67FA          beq  conin      * wait until key pressed
62 00000086 103903FFFF00    move.b  $ffff00,d0      * get key
63 0000008C C0BC0000007F    and.l  #57f,d0      * clear all but low 7 bits
64 00000092 4E75          rts
65
66 030G0094 10390CFFFF01    conout: move.b  $ffff01,d0      * get status
67 0500009A C03C0001      and.b  #51,d0      * check for transmitter buffer empty
68 0000009E 67F4          beq  conout      * wait until our port has aged...
69 000000A0 13C100FFFF00    move.b  d1,$ffff00      * and output it
70 C00000A6 4E75          rts      * and exit
71
72
73
74      * Disk Handlers for Tarbell 1793 floppy disk controller
75      *
76      maxdsk = 2      * this BIOS supports 2 floppy drives
77      dphlen = 26      * length of disk parameter header
78
79      iobase = $00ffff8      * 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      dsct = iobase+2      * disk sector port
84      ddata = iobase+3      * disk data port
85      dwait = iobase+4      * input port to wait for op finished
86      dcntl = iobase+4      * output control port for drive selection
87
88
89 000000A8 423900000002    home:  clr.b  track
90 000000AE 4E75          rts
91
92      seldsk:
93      *      select disk A
94 000000B0 423900000000    clr.b  seldrv      * select drive A

```

Listing C-1. (continued)


```

95 000000B6 42390000000A      clr.b    selcode      * select code is 00 for drv 0, $10 for drv 1
96 000000BC 203C0000000C      move.l   $dph0,d0
97 000000C2 4E75              selrtn:  rts
98
99 000000C4 13C100000002      settrk:  move.b   d1,track
100 000000CA 4E75              rts
101
102 000000CC 13C100000004      setsec:  move.b   d1,sector
103 000000D2 4E75              rts
104
105      sectran:
106      *      translate sector in d1 with translate table pointed to by d2
107      *      result in d0
108      move.l   d2,a0
109 000000D6 48C1      ext.l    d1
110 000000D8 10301000      move.b   $0(a0,d1),d0
C P / M 6 8 0 0 0   A s s e m b l e r      Revision 02.01      Page 3
Source File: a:eldbios.s

111 000000DC 48C0      ext.l    d0
112 000000DE 4E75      rts
113
114      setdma:
115 000000E0 23C100000006      move.l   d1,dma
116 000000E6 4E75      rts
117
118      read:
119      * Read one sector from requested disk, track, sector to dma address
120      * Retry if necessary, return in d0 00 if ok, else non-zero
121 000000E6 13FC000A0000000B      move.b   $10,errcnt      * set up retry counter
122
123 000000F0 6134      bsr      setup
124 000000F2 004300B8      ori      $588,d3      * OR read command with head load bit
125 000000F6 13C300FFFFF8      move.b   d3,dcmd      * output it to FDC
126 000000FC 08390000700FFFFFC      bsr      $7,dwait
127 00000104 6708      beq      rdone      * if end of read, exit
128 00000106 10F900FFFFFB      move.b   ddata,(a0)+      * else, move next byte of data
129 0000010C 60E2      bra      rloop
130
131 0000010E 61000106      rdone:   bsr      rstatus      * get FDC status
132 00000112 6604      bne      rerror
133 00000114 4280      clr.l    d0
134 00000116 4E75      rts
135 00000118 6170      error:  bsr      errchk      * go to error handler
136 0000011A 53390000000B      subq.b   $1,errcnt
137 00000120 66CE      bne      rretry
138 00000122 70FF      move.l   $FFFFFFF,d0
139 00000124 4E75      rts
140
141      setup:
142      * common read and write setup code
143      * select disk, set track, set sector were all deferred until now
144      move.b   $5d0,dcmd      * clear controller, get status
145 00000126 13FC00D000FFFFF8      move.b   curdrv,d3
146 0000012E 163900000001

```

Listing C-1. (continued)

```

147 00000134 B63900000000    cmp.b    seldrv,d3
148 0000013A 661A             bne      newdrive      * if drive not selected, do it
149 0000013C 163900000000    move.b    track,d3
150 00000142 B63900000000    cmp.b    oldtrk,d3
151 00000148 6620             one      newtrk      * if not on right track, do it
152 0000014A 4283             clr.l    d3          * if head already loaded, no head load delay
153 0000014C 0839000050GFFFFF8      btst     $5,dstat    * if head unloaded, treat as new disk
154 00000154 6618             bne      sextit
155
newdrive:
156 00000156 13F90000000A00FFFFFC    move.b    selcode,dctrl    * select the drive
157 00000160 13F90000000000000001    move.b    seldrv,curdrv
158
newtrk:
159 0000016A 6126             bsr      chkseek      * seek to correct track if required
160 0000016C 7604             moveq    #4,d3        * force head load delay
161
sextit:
162 0000016E 13F900000000400FFFFFA    move.b    sector,dsect    * set up sector number
163 00000178 13F900000000200FFFFF9    move.b    track,dtrk      * set up track number
164 00000182 2079000000006         move.l    dma,a0          * dma address to a0
165 00000188 4E75             rts
C P / M 6 8 0 0 0 A s s e m b l e r      Revision 02.01      Page 4
Source File: a:eldbios.s

166
167
168 0000018A 08070004          errchk:    btst     #4,d7
169 0000018E 6602          bne      chkseek      * if record not found error, reseek
170 00000190 4E75          rts
171
chkseek:
172
173      * check for correct track, seek if necessary
174 00000192 615C          bsr      readid      * find out what track we're on
175 00000194 671E          beq      chksl       * if read id ok, skip restore code
176
restore:
177      * home the drive and reseek to correct track
178 00000196 13FC000800FFFFF8    move.b    $50B,dcmd      * restore command to command port
179
rstwait:
180 0000019E 08390000700FFFFFC    btst     $7,dwait      * loop until restore completed
181 000001A6 66F6          bne      rstwait
182 000001A8 08390000200FFFFF8    btst     $2,dstat
183 000001B0 67E4          beq      restore      * if not at track 0, try again
184 000001B2 4283          clr.l    d3          * track number returned in d3 from readid
185
chksl:
186 000001B4 13C300FFFFF9    move.b    d3,dtrk      * update track register in FDC
187 000001BA 13F900000000200000003    move.b    track,oldtrk  * update oldtrk
188 000001C4 B639000000002    cmp.b    track,d3      * are we at right track?
189 000001CA 6722          beq      chkdne        * if yes, exit
190 000001CC 13F900000000200FFFFFB    move.b    track,ddata    * eluc, put desired track in data reg of FDC
191 000001D6 13FC001800FFFFF8    move.b    $518,dcmd      * and issue a seek command
192 000001DE 08390000700FFFFFC    btst     $7,dwait
193 000001E6 66F6          bne      chks2         * loop until seek complete
194 000001E8 163900FFFFF8    move.b    dstat,d3      * read status to clear FDC
195
chkdne:
196 000001EE 4E75          rts
197
readid:
198
199

```

Listing C-1. (continued)

```

199
200 000001F0 13FC00C400FFFFF8      * read track id, return track number in d3
201 000001F8 1E3900FFFFF8      move.b $5c4,dcmd      * issue read id command
202 000001FE 163900FFFFF8      move.b dwait,d7      * wait for intrq
203                                move.b ddata,d3      * track byte to d3
204 00000204 0a39000700FFFFF8
205 0000020C 6708
206 0000020E 1E3900FFFFF8
207 00000214 60E2
208                                rid2:
209 00000216 1E3900FFFFF8      btest $7,dwait      * wait for intrq
210 0000021C 0207009D      beq rstatus      * read another byte
211 00000220 4E75      move.b ddata,d7      * and loop
212                                rstatus:
213                                move.b dstat,d7
214                                andi.b $59d,d7      * set condition codes
215                                rts
216                                setexc:
217 00000222 0281000000FF      andi.l $5ff,d1      * do only for exceptions 0 - 255
218 00000228 E549      lsl $2,d1      * multiply exception number by 4
219 0000022A 2041      movea.l d1,a0
220 0000022C 2010      move.l (a0),d0
221 0000022E 2082      move.l d2,(a0)
222 00000230 4E75      rts      * return old vector value
223                                * insert new vector
C P / M 6 8 0 0 0 A s s e m b l e r      Revision 02.01      Page 5
Source File: a:elddbios.s

```

```

221
222
223 00000000      .data
224
225 00000000 FF      seldrv: .dc.b $ff      * drive requested by seldsk
226 00000001 FF      curdrv: .dc.b $ff      * currently selected drive
227
228 00000002 00      track: .dc.b 0      * track requested by settck
229 00000003 00      oldtrk: .dc.b 0      * track we were on
230
231 00000004 0000      sector: .dc.w 0
232 00000006 00000000      dma: .dc.l 0
233 0000000A 00      selcode: .dc.b 0      * drive select code
234
235 0000000B 0A      errcnt: .dc.b 10      * retry counter
236
237
238
239
240 0000000C 00000036      * disk parameter headers
241 00000010 0000      dph0: .dc.l xlt
242 00000012 0000      .dc.w 0      * dummy
243 00000014 0000      .dc.w 0
244 00000016 00000000      .dc.w 0
245 0000001A 00000026      .dc.l dirbuf      * ptr to directory buffer
246 0000001E 00000000      .dc.l dpb      * ptr to disk parameter block
247 00000022 00000000      .dc.l 0      * ptr to check vector
248                                .dc.l 0      * ptr to allocation vector
249
250      * disk parameter block

```

Listing C-1. (continued)

```

251
252 00000026 001A      dpb:  .dc.w  26      * sectors per track
253 00000028 03        .dc.b  3      * block shift
254 00000029 07        .dc.b  7      * block mask
255 0000002A 00        .dc.b  0      * extent mask
256 0000002B 00        .dc.b  0      * dummy fill
257 0000002C 00F2      .dc.w  242     * disk size
258 0000002E 003F      .dc.w  63      * 64 directory entries
259 00000030 C000      .dc.w  5C000   * directory mask
260 00000032 0010      .dc.w  16      * directory check size
261 00000034 0002      .dc.w  2      * track offset

```

* sector translate table

```

262
263
264
265 00000036 01070D13  xlt:  .dc.b  1, 7, 13, 19
266 0000003A 19050B11  .dc.b  25, 5, 11, 17
267 0000003E 1703090F  .dc.b  23, 3, 9, 15
268 00000042 1502080E  .dc.b  21, 2, 8, 14
269 00000046 141A060C  .dc.b  20, 26, 6, 12
270 0000004A 1218040A  .dc.b  18, 24, 4, 10
271 0000004E 1016      .dc.b  16, 22

```

```

272
273
274 00000000      .bss
275

```

CP/M 68000 Assembler Revision 02.01 Page 6
Source File: a:eldbios.s

```

276 00000000      dirbuf: .ds.b  128    * directory buffer
277
278
279 00000080      .end

```

CP/M 68000 Assembler Revision 02.01 Page 7
Source File: a:eldbios.s

Symbol Table

_bios	00000000 TEXT	biosbase	00000010 TEXT	chkdone	000001EE TEXT	chksl	000001B4 TEXT
Chks2	000001DE TEXT	chkseek	00000192 TEXT	conin	00000080 TEXT	conout	00000094 TEXT
constat	0000006C TEXT	curdrv	00000001 DATA	dcmd	00FFFFFF8 ABS	dcntrl	00FFFFFFC ABS
ddata	00FFFFFFB ABS	dirbuf	00000000 BSS	dma	00000006 DATA	dpb	00000026 DATA
dph0	0000000C DATA	dphlen	0000001A ABS	dsect	00FFFFFFA ABS	dstat	00FFFFFF8 ABS
dtrk	00FFFFFF9 ABS	dwait	00FFFFFFC ABS	errchk	0000018A TEXT	errcnt	0000000B DATA
home	000000A8 TEXT	iobase	00FFFFFF8 ABS	maxdsk	00000002 ABS	newdrive	00000156 TEXT
newtrk	0000016A TEXT	nfuncs	00000017 ABS	nogood	0000000E TEXT	noton	0000007C TEXT
oldtrk	00000003 DATA	rdone	0000010E TEXT	read	000000E8 TEXT	readid	000001F0 TEXT
rerror	00000118 TEXT	restore	00000196 TEXT	rid2	00000204 TEXT	rloop	000000FC TEXT
rretry	000000F0 TEXT	rstatus	00000216 TEXT	rstwait	0000019E TEXT	sector	00000004 DATA
sectran	000000D4 TEXT	selcode	0000000A DATA	seldrv	00000000 DATA	seldsk	000000B0 TEXT
selrtn	000000C2 TEXT	setdma	000000E0 TEXT	setexc	00000222 TEXT	setsec	000000CC TEXT
settrk	000000C4 TEXT	setup	00000126 TEXT	sexit	0000016E TEXT	track	00000002 DATA
xlt	00000036 DATA						

Listing C-1. (continued)

End of Appendix C

Appendix D

EXORmacs BIOS Written in C

This appendix contains several files in addition to the C 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.

Listing D-1. EXORmacs BIOS.C File

```

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

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

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

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

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

/*=====*/
/*      I/O Device Definitions
/*=====*/

```

Listing D-1. (continued)

```

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

/* Port Addresses */

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

/* Port Offsets */

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

/* Port Control Functions */

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

/* Port Status Values */

#define PORTDRF 1 /* Read Data Register Full */
#define PORTTDR 2 /* Write Data Register Empty */

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

#define DSKIPC 0xFF0000 /* IPC Base Address */
#define DSKINTV 0x3FC /* Address of Disk Interrupt Vector */

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

#define DSKREAD 0x10 /* disk commands */
#define DSKWRITE 0x20
/* Some characters used in disk controller packets */
#define STX 0x02
#define ETX 0x03

#define ACK 0x06
#define NAK 0x15
#define PKTSTX 0x0 /* offsets within a disk packet */
#define PKTID 0x1
#define PKTSZ 0x2

```

Listing D-1. (continued)

```

#define PKTDEV          0x3
#define PKTCHCOM        0x4
#define PKTSTCOM        0x5
#define PKTSTVAL        0x6
#define PKTSTPRM        0x8
#define STPKTSZ         0xf

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

/* Disk Parameter Block Structure */

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

/* Disk Parameter Header Structure */
struct dph
{
    BYTE    *x1tp;
    WORD    dphscr[3];
    BYTE    *dirbutp;
    struct dpb    *dpbp;
    BYTE    *csvp;
    BYTE    *alvp;
};
/*****
/* Directory Buffer for use by the BDOS */
*****/

BYTE dirbuf[128];

#if ! LOADER

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

BYTE    csv0[16];
BYTE    csv1[16];

```

Listing D-1. (continued)

```

#if ! CTLTYPE
BYTE    csv2[256];
BYTE    csv3[256];

#endif

#if    MEMDSK
BYTE    csv4[16];
#endif

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

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

#if ! CTLTYPE

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

#endif

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

#endif

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

/* The following apb definitions express the intent of the writer,
/* unfortunately, due to a compiler bug, these lines cannot be used.
/* Therefore, the obscure code following them has been inserted.

/*sw With release 1.2, the structure init bug disappeared, so...
/*****      spt, bsh, blm, exn, jnk, dsm, arm, al0, all, cks, off */
struct dpb apb0 = { 46,   3,   7,   0,   0,  242,  63,   0,   0, 16,  4};

#if ! CTLTYPE
struct dpb apb2 = { 32,   5,  31,   1,   0, 3288, 1023,   0,   0, 256,  4};
#endif

#if    MEMDSK
struct dpb apb3 = { 32,   4,  15,   0,   0,  191,  63,   0,   0,  0,  0};
#endif

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

```


Listing D-1. (continued)

```

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

/*****
/* Disk Parameter Headers
/*
/* Four disks are defined : dsk a: diskno=0, (Motorola's #fd04)
/* if CTLTYPE = 0      : dsk b: diskno=1, (Motorola's #fd05)
/*                      : dsk c: diskno=2, (Motorola's #hd00)
/*                      : dsk d: diskno=3, (Motorola's #ha01)
/*
/* Two disks are defined : dsk a: diskno=0, (Motorola's #fd00)
/* if CTLTYPE = 1      : ask b: diskno=1, (Motorola's #fd01)
/*
*****/

#if 1 LOADER

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

    { {&xlt, 0, 0, 0, &dirbuf, &dpb0, &csv0, &alv0}, /*dsk a*/
      {&xlt, 0, 0, 0, &dirbuf, &dpb0, &csv1, &alv1}, /*dsk b*/

#if 1 CTLTYPE
    { 0L, 0, 0, 0, &dirbuf, &dpb2, &csv2, &alv2}, /*dsk c*/
    { 0L, 0, 0, 0, &dirbuf, &apb2, &csv3, &alv3}, /*dsk d*/
#endif

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

#endif
#else

#if 1 CTLTYPE
struct dph dphtab[4] =
#else
struct dph dphtab[2] =

#endif

    { {&xlt, 0, 0, 0, &dirbuf, &dpb0, 0L, 0L}, /*dsk a*/
      {&xlt, 0, 0, 0, &dirbuf, &apb0, 0L, 0L}, /*dsk b*/

#if 1 CTLTYPE
    { 0L, 0, 0, 0, &dirbuf, &dpb2, 0L, 0L}, /*dsk c*/
    { 0L, 0, 0, 0, &dirbuf, &apb2, 0L, 0L}, /*ask d*/
#endif

    };

#endif

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

```

Listing D-1. (continued)

```

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

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

#if ! LOADER

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

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

#endif

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

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

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

#if ! LOADER

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

/* Define the track buffer structure */

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

```

Listing D-1. (continued)

```

struct tbstr *firstouf; /* head of linked list of track buffers */
struct tbstr *lastouf; /* tail of ditto */

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

#else

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

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

#endif

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

/* Home disk packet */

struct hmpkst {
    BYTE    a1;
    BYTE    a2;
    BYTE    a3;
    BYTE    askno;
    BYTE    com1;
    BYTE    com2;
    BYTE    a6;
    BYTE    a7;
}

    hmpack = { 2,0, 7,0, 0,0, 3,0 }; /*sw Init by bytes now... */
/*      hmpack = { 512, 1792, 0, 768 }; */ /* kludge init by words */

/* Read/write disk packet */

struct rwpkst {
    BYTE    stxchr;
    BYTE    pktid;
    BYTE    pktsize;
    BYTE    askno;
    BYTE    chcm;
    BYTE    devcmd;
    WORD    numdks;
    WORD    blksize;
    LONG    lodi;
    WORD    cksum;
    LONG    lsect;
    BYTE    etxchr;
    BYTE    rwpad;
};

    struct rwpkst rwpack = { 2,0, 21,0, 16,1, 13, 256, 0L, 0, 0L, 3,0 };
/*struct rwpkst rwpack = { 512, 5376, 4097, 13, 256, 0, 0, 0, 0, 0, 768 };*/

```

Listing D-1. (continued)

```

#if ! LOADER
/* format disk packet */
struct ftmpkst {
    BYTE    ftmpstx;
    BYTE    ftmpid;
    BYTE    ftmpsize;
    BYTE    ftmpaskno;
    BYTE    ftmpcncmd;
    BYTE    ftmpdvcmd;
    BYTE    ftmpetx;
    BYTE    ftmpad;
};

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

#endif

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

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

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

#if ! CTLTYPE
BYTE cnvask[NUMDSKS] = { 4, 5, 0, 1 }; /* convert CP/M dsk# to EXORMacs */
BYTE rcnvask[6]      = { 2, 3, 0, 0, 0, 1 }; /* and vice versa */
#else
BYTE cnvask[NUMDSKS] = { 0, 1 };
BYTE rcnvask[2]      = { 0, 1 };
#endif
/* defines for IPC and disk states */

#define IDLE 0
#define ACTIVE 1

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

```

Listing D-1. (continued)

```

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

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

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

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

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

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

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

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

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

portout(port, ch)

```

Listing D-1. (continued)

```

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

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

#if ! LOADER

bioserr(errmsg)
REG BYTE *errmsg;
{
    printstr("nrBIOS ERROR -- ");
    printstr(errmsg);
    printstr(".nr");
}

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

#else

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

#endif

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

EXTERN dskia(); /* external interrupt handler -- calls dskic */
EXTERN setimask(); /* use to set interrupt mask -- returns old mask */
dskic()
{
    /* Disk Interrupt Handler -- C Language Portion */

    REG BYTE workbyte;
    BYTE stpkt[STPKTSZ];
    workbyte = (DSKIPC + ACKFMIPC)->byte;
    if ( (workbyte == ACK) || (workbyte == NAK) )

```

Listing D-1. (continued)

```

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

workbyte = (DSKIPC + MSGFMIPC)->byte;
if ( workbyte & 0x80 )
{
    getstpkt(stpkt);
    if ( stpkt[PKTID] == 0xFF )
    {
        /* unsolicited */

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

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

} /* end of dskic */

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

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

    p = stpkt;
    q = (DSKIPC + PKTFMIPC);

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

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

unsolst(stpkt)
REG BYTE *stpkt;

```

Listing D-1. (continued)

```

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

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

#if 1 LOADER
/*****
/*      Mark all buffers for a disk as not valid      */
*****/

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

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

#endif

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

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

    while (1)
    {
        while ( ! intcount ) ; /* wait */
        msave = setimask(7);
        intcount -= 1;
        work = (DSKIPC + ACKFMIPC)->byte;
        if ( (work == ACK) || (work == NAK) )

```


Listing D-1. (continued)

```

        {
            (DSKIPC + ACKFMIPC)->byte = 0;
            setimask(imsave);
            return(work == ACK);
        }
        setimask(imsave);
    }
}

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

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

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

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

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

/*****
/*      Wait for a Disk Operation to Finish
*****/

WORD dskwait(dsk, stcom, stval)
REG WORD dsk;
BYTE stcom;

```

Listing D-1. (continued)

```

WORD    stval:
{
    REG WORD imsave;
    BYTE stpkt(STPKTSZ);

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

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

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

dskxfer(dsk, trk, bufp, cmd)
REG WORD dsk, trk, cmd;
REG BYTE *bufp;
{
    /* build packet */

    REG WORD sectcnt;
    REG WORD result;

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

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

    #if CTLTYPE

```

Listing D-1. (continued)

```

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;
ipcstate = IDLE;
return(result);
}

#if ! LOADER
/***** Write one disk buffer *****/
/* Write one disk buffer */
/*****

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

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

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

    return(ok);
}

```

Listing D-1. (continued)

```

}

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

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

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

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

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

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

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

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

    return(ok);
}

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

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

```

Listing D-1. (continued)

```
REG struct tbstr *mtbp;
REG WORD msave;

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

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

/* Search through buffers to see if the required stuff
/* is already in a buffer */
;
tbp = firstbuf;
ltbp = 0;
mtbp = 0;
```

Listing D-1. (continued)

```

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

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

if (mtbp) mtbp->nextbuf = 0;      /* detach lru buffer */
ltbp->nextbuf = firstbuf;
firstbuf = ltbp;
setimask(imsave);
if (flush1(ltbp) && fill1(ltbp)) mtbp = ltbp;      /* success */
else                               mtbp = 0L;      /* failure */
return (mtbp);
}

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

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

    #if MEMDSK
    if(setdsk != MEMDSK)
    {
        #endif
        tbp = gettrk();          /* locate track buffer with sector */
        if ( ! tbp ) return(1); /* failure */

```

Listing D-1. (continued)

```

/* locate sector in buffer and copy contents to user area */
p = (tbp->buf) + (setsec << 7); /* multiply by shifting */
MEMDSK
#endif
else
p = memdsk + (((LONG)(settrk) << 12L) + ((LONG)setsec << 7L));
#endif
q = setdma;
i = 128;
do { *q++ = *p++; i -= 1; } while (i); /* this generates good code */
return(0);
}

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

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

    /* locate track buffer containing sector to be written */
    MEMDSK
    if (setdsk != MEMDSK)
    {
        #endif
        tbp = gettrk();
        if ( ! tbp ) return (1); /* failure */

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

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

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

```

Listing D-1. (continued)

```

/* The track must be written if this is a directory write */
if ( mode == 1 ) { if ( flush1(tbp) ) return(0); else return(1); }
else return(0);
}

#else

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

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

#endif

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

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

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

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

```


Listing D-1. (continued)

```

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

        return(oldval);
    }

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

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

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

#if : LOADER

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

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

    dphp = &dphtab[dsk];

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

    if ( ! (logged & 0x1) )
    {
        hmpack.dskno = cnvdsk[setdsk];
        hmpack.com1 = 0x30;
        hmpack.com2 = 0x02;
        actvdsk = dsk;
        dskstate[dsk].change = 0;
        sendpkt(&hmpack, 7);
    }

```

Listing D-1. (continued)

```

if ( ! dskwait(dsk, 0x72, 0x0) )
{
    sendack();
    ipcstate = IDLE;
    return ( 0L );
}
getstpkt(stpkt);          /* determine disk type and size */
sendack();
ipcstate = IDLE;
st1 = stpkt[PKTSTPRM];
st2 = stpkt[PKTSTPRM+1];

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

switch ( st1 & 7 )
{
    case 1 :                /* floppy disk */
        dphp->dphp = &dphp0;
        break;

    case 2 :                /* hard disk */
        dphp->dphp = &dphp2;
        break;

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

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

```

Listing D-1. (continued)

```

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

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

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

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

#endif

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

biosinit()
{
    initprts();
    initdsk();
}

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

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

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

```

Listing D-1. (continued)

```

        else                                tbuf[1].nextbuf = 0;
    }
    firstbuf = &tbuf[0];
    lastbuf = &tbuf[NUMTB-1];
#else
    bufvalid = 0;
#endif

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

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

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

LONG cbios(d0, d1, d2)
REG WORD    d0;
REG LONG    d1, d2;
{
    switch(d0)
    {
        case 0: biosinit();                /* INIT      */
            break;

#ifdef ! LOADER
        case 1: flush();                    /* WBOOT     */
            initdsk();
            wboot();
            /* break; */

#endif

        case 2: return(portstat(PORT1));    /* CONST     */
            /* break; */

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

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

        case 5: :                          /* LIST      */

```

Listing D-1. (continued)

```

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

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

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

        case 9:
#if LOADER
        /*      dl = 3; */                      /* Always HD   */
#endif
                return(selctdisk((char)d1, (char)d2)); /* SELDSK     */
                /* break; */

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

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

        case 12: setdma = dl;                  /* SETDMA     */
                break;

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

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

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

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

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

        case 21: if (!flush()) return(OL);     /* FLUSH      */
                    else return(0xffffL);
                /* break; */
#endif

        case 22: return(setxvect((int)d1,d2)); /* SETXVECT   */
                /* break; */
#if ! LOADER

```

Listing D-1. (continued)

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

#endif

default: return(OL);
break;

} /* end switch */

} /* END OF BIOS */

/* End of C Bios */
```

Listing D-2. EXORmatx BIOSTYPS.H File

```

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

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

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

/*****/

```

Listing D-3. EXORmacs NORMBIOS.H File

```

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

```

Listing D-4. EXORmacs LOADBIOS.H File

```

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

```

Listing D-5. EXORmacs BIOSA.S File:

```

*      .text
*
*      Global Code addresses
*
*      .globl  _init
*      .globl  _biosinit
*      .globl  _flusn
*      .globl  _wboot
*      .globl  _cbios
*      .globl  _dskia
*      .globl  _dskic
*      .globl  _setimask
*      .globl  _ccp
*      .globl  _cpm
*      .globl  _end
*
*      Global data addresses
*
*      .globl  _memtab
*      .globl  _dpb3
*      .globl  _memdisk
*
*      Vector Addresses
*
dskint: .equ    $3fc
trap3:  .equ    $8c
buserr: .equ    $8
*
*      _init: lea    entry,a0
*             move.l a0,trap3
*             lea    dskia,a0
*             move.l a0,dskint
*
*      Auto-Size TPA
*
*      lea    _memtab,a0
*      move.w #1,(a0)+
*      move.l #$400,(a0)+
*      move.l #cpm-$40d,(a0)+
*
*      Auto-Size RAM disk
*
*      move.l buserr,-(sp)
*      lea    _end,a0
*      add.l  #cpm,a0
*      move.l a0,_memdisk
*      move.l #quit,buserr
*
loop:   tst.w  (a0)+
*
quit:   bra   loop
*
*      UDC Interrupt vector
*      Trap 3 vector
*      Bus error vector
*
*      a0 -> Memory region table
*      1 region
*      TPA starts at 400
*      Ends where CP/M begins
*
*      Push bus err vector
*      a0 -> Last location in CP/M
*      Linker doesn't reloc this!!
*      -> first location in RAM disk
*      set up vector -> ourselves
*
*      Find
*      End of memory

```


Listing D-5. (continued)

```

        add.l    #14,a7
        move.l   (a7)+,buserr
        sub.l    #_end,a0
        sub.l    #cpm,a0
        move.l   a0,d0
        move.l   #11,d1
        lsr.l    d1,d0
        move.w   d0,_dpb3+6
        *
        move     $$2000,sr
        jsr      _biosinit
        clr.l    d0
        rts
    *
_wboot: clr.l    d0
        jmp      _ccp
    *
entry:  move.l   d2,-(a7)
        move.l   d1,-(a7)
        move.w   d0,-(a7)
        jsr      _cbios
        add      #10,a7
        rte
    *
_dskia: link     a6,#0
        movem.l  d0-d7/a0-a5,-(a7)
        jsr      _dskic
        movem.l  (a7)+,d0-d7/a0-a5
        unlk     a6
        rte
    *
_setimask: move  sr,d0
        lsr      #8,d0
        and.l    #7,d0
        move     sr,d1
        ror.w    #8,d1
        and.w    #$fff8,d1
        add.w    4(a7),d1
        ror.w    #8,d1
        move     d1,sr
        rts
        .end

```

```

*      Clear buserr gorp
*      Pop buserr vector
*      a0 = # bytes in RAM disk
*      Relocation bug
*      Into D reg for shift
*      Load shift count
*      Divide by 2048
*      Load DRM field of dpb

```

Listing D-6. EXORmacs LDBIOS.S File

```

        .text
        .globl  _bios
        .globl  _biosinit
        .globl  _cbios
        .globl  _dskia
        .globl  _dskic
        .globl  _setimask
*
*
*
*
_bios:  link     a6,#0
        move.l   d2,-(a7)
        move.l   d1,-(a7)
        move.w   d0,-(a7)
        move     #$2000,sr
        lea      _dskia,a0
        move.l   a0,$3fc
        jsr      _cbios
        unlk     a6
        rts
*
_dskia: link     a6,#0
        movem.l  d0-d7/a0-a5,-(a7)
        jsr      _dskic
        movem.l  (a7)+,a0-d7/a0-a5
        unlk     a6
        rte
*
_setimask: move  sr,d0
        lsr      #8,d0
        and.l    #7,d0
        move     sr,d1
        ror.w    #8,d1
        and.w    #$fff8,d1
        add.w    4(a7),d1
        ror.w    #8,d1
        move     d1,sr
        rts
*
        .end

```

Listing D-7. EXORMacs BOOTER.S File

```

*****
* Information to go on the 256 byte *
* boot sector of an ExorMacs *
*****
,
    .text
    .dc.l    $4000          * starting stack pointer
    .dc.l    start         * starting program counter
    .dc.w    1              * garbage
    .dc.w    1              * length of SAT
    .dc.l    2              * secondary directory start
    .dc.l    0              * primary directory PSN list start
    .dc.l    0              * start of boot loader
    .dc.w    26             * length of boot loader
    .dc.l    $0             * boot execution address
    .dc.l    $0             * boot load address
    .dc.b    '9/30'         * generation date
    .dc.b    'CP/M-68K of 9/30/82 ' * volume descriptor
    .dc.b    '0020'         * version/revision
    .dc.w    $0a484         * checksum (god help us)
    .dc.l    $0fle2d3c      * diagnostic test pattern
    .dc.l    $4b5a6978
    .dc.l    $8796a5b4
    .dc.l    $c3d2elf0
    .dc.l    $0fle2d3c      * diagnostic test pattern
    .dc.l    $4b5a6978
    .dc.l    $8796a5b4
    .dc.l    $c3d2elf0
    .dc.l    $4f8f0f07      * diagnostic test pattern
    .dc.l    $0b0d0e06
    .dc.l    $0a0c0408
    .dc.l    $04020100
    .dc.l    00, 00, 00, 00 * diagnostic test pattern
    .dc.l    0              * diagnostic test area directory
    .dc.l    0              * start of dump area
    .dc.w    0              * length of dump area
    .dc.l    0              * start of sector lockout table
    .dc.w    0              * length of sector lockout table
    .dc.l    0,0,0,0,0,0,0 * unused, reserved
    .dc.l    0,0,0,0,0,0
    .dc.l    0,0,0,0,0,0
    .dc.l    0,0,0,0,0,0
    .dc.l    0,0,0,0,0,0
    .dc.b    'EXORMACS'     * let's hear it for Motorola
*
*
*
*
    .ds.b    $300          * skip over exception vectors
    .even
start: move    $$2700,sr
    move.l   $$8,a0

```

Listing D-7. (continued)

```
        move.w  #253,d0
exlp:   move.l  #expdef,(a0)+
        dbf     d0,exlp
        jmp     grunt
expdef: rte
grunt:  move    #$2000,sr
        .end
```

Appendix E

Putboot Utility Assembly Language Source

Listing E-1. PUTBOOT Assembly Language Source

```
CP/M 68000 Assembler
Source File: putboot.s
```

Revision 02.01

Page 1

```

1      *
2      *
3      *       Program to Write Boot Tracks for CP/M-68K (tm)   *
4      *
5      *       Copyright Digital Research 1982                  *
6      *
7      *.....*
8      *
9      *
10     *
11     printstr =          9           BDOS Functions
12     dseldsk =         14
13     open =            15
14     readseq =         20
15     dsetdma =         26
16     *
17     *
18     *
19     *
20     *
21     *
22     *
23     *
24     *
25     *
26     *
27     *
28     *
29     *
30     *
31     *
32     *
33     *
34     *
35     *
36     *
37     *
38     *
39     *
40     *
41     *
42     *
43     *
44     *
45     *
46     *
47     *
48     *
49     *
50     *
51     *
52     *
53     *
54     *
55     *
56     *
57     *
58     *
59     *
60     *
61     *
62     *
63     *
64     *
65     *
66     *
67     *
68     *
69     *
70     *
71     *
72     *
73     *
74     *
75     *
76     *
77     *
78     *
79     *
80     *
81     *
82     *
83     *
84     *
85     *
86     *
87     *
88     *
89     *
90     *
91     *
92     *
93     *
94     *
95     *
96     *
97     *
98     *
99     *
100    *
101    *
102    *
103    *
104    *
105    *
106    *
107    *
108    *
109    *
110    *
111    *
112    *
113    *
114    *
115    *
116    *
117    *
118    *
119    *
120    *
121    *
122    *
123    *
124    *
125    *
126    *
127    *
128    *
129    *
130    *
131    *
132    *
133    *
134    *
135    *
136    *
137    *
138    *
139    *
140    *
141    *
142    *
143    *
144    *
145    *
146    *
147    *
148    *
149    *
150    *
151    *
152    *
153    *
154    *
155    *
156    *
157    *
158    *
159    *
160    *
161    *
162    *
163    *
164    *
165    *
166    *
167    *
168    *
169    *
170    *
171    *
172    *
173    *
174    *
175    *
176    *
177    *
178    *
179    *
180    *
181    *
182    *
183    *
184    *
185    *
186    *
187    *
188    *
189    *
190    *
191    *
192    *
193    *
194    *
195    *
196    *
197    *
198    *
199    *
200    *
201    *
202    *
203    *
204    *
205    *
206    *
207    *
208    *
209    *
210    *
211    *
212    *
213    *
214    *
215    *
216    *
217    *
218    *
219    *
220    *
221    *
222    *
223    *
224    *
225    *
226    *
227    *
228    *
229    *
230    *
231    *
232    *
233    *
234    *
235    *
236    *
237    *
238    *
239    *
240    *
241    *
242    *
243    *
244    *
245    *
246    *
247    *
248    *
249    *
250    *
251    *
252    *
253    *
254    *
255    *
256    *
257    *
258    *
259    *
260    *
261    *
262    *
263    *
264    *
265    *
266    *
267    *
268    *
269    *
270    *
271    *
272    *
273    *
274    *
275    *
276    *
277    *
278    *
279    *
280    *
281    *
282    *
283    *
284    *
285    *
286    *
287    *
288    *
289    *
290    *
291    *
292    *
293    *
294    *
295    *
296    *
297    *
298    *
299    *
300    *
301    *
302    *
303    *
304    *
305    *
306    *
307    *
308    *
309    *
310    *
311    *
312    *
313    *
314    *
315    *
316    *
317    *
318    *
319    *
320    *
321    *
322    *
323    *
324    *
325    *
326    *
327    *
328    *
329    *
330    *
331    *
332    *
333    *
334    *
335    *
336    *
337    *
338    *
339    *
340    *
341    *
342    *
343    *
344    *
345    *
346    *
347    *
348    *
349    *
350    *
351    *
352    *
353    *
354    *
355    *
356    *
357    *
358    *
359    *
360    *
361    *
362    *
363    *
364    *
365    *
366    *
367    *
368    *
369    *
370    *
371    *
372    *
373    *
374    *
375    *
376    *
377    *
378    *
379    *
380    *
381    *
382    *
383    *
384    *
385    *
386    *
387    *
388    *
389    *
390    *
391    *
392    *
393    *
394    *
395    *
396    *
397    *
398    *
399    *
400    *
401    *
402    *
403    *
404    *
405    *
406    *
407    *
408    *
409    *
410    *
411    *
412    *
413    *
414    *
415    *
416    *
417    *
418    *
419    *
420    *
421    *
422    *
423    *
424    *
425    *
426    *
427    *
428    *
429    *
430    *
431    *
432    *
433    *
434    *
435    *
436    *
437    *
438    *
439    *
440    *
441    *
442    *
443    *
444    *
445    *
446    *
447    *
448    *
449    *
450    *
451    *
452    *
453    *
454    *
455    *
456    *
457    *
458    *
459    *
460    *
461    *
462    *
463    *
464    *
465    *
466    *
467    *
468    *
469    *
470    *
471    *
472    *
473    *
474    *
475    *
476    *
477    *
478    *
479    *
480    *
481    *
482    *
483    *
484    *
485    *
486    *
487    *
488    *
489    *
490    *
491    *
492    *
493    *
494    *
495    *
496    *
497    *
498    *
499    *
500    *
501    *
502    *
503    *
504    *
505    *
506    *
507    *
508    *
509    *
510    *
511    *
512    *
513    *
514    *
515    *
516    *
517    *
518    *
519    *
520    *
521    *
522    *
523    *
524    *
525    *
526    *
527    *
528    *
529    *
530    *
531    *
532    *
533    *
534    *
535    *
536    *
537    *
538    *
539    *
540    *
541    *
542    *
543    *
544    *
545    *
546    *
547    *
548    *
549    *
550    *
551    *
552    *
553    *
554    *
555    *
556    *
557    *
558    *
559    *
560    *
561    *
562    *
563    *
564    *
565    *
566    *
567    *
568    *
569    *
570    *
571    *
572    *
573    *
574    *
575    *
576    *
577    *
578    *
579    *
580    *
581    *
582    *
583    *
584    *
585    *
586    *
587    *
588    *
589    *
590    *
591    *
592    *
593    *
594    *
595    *
596    *
597    *
598    *
599    *
600    *
601    *
602    *
603    *
604    *
605    *
606    *
607    *
608    *
609    *
610    *
611    *
612    *
613    *
614    *
615    *
616    *
617    *
618    *
619    *
620    *
621    *
622    *
623    *
624    *
625    *
626    *
627    *
628    *
629    *
630    *
631    *
632    *
633    *
634    *
635    *
636    *
637    *
638    *
639    *
640    *
641    *
642    *
643    *
644    *
645    *
646    *
647    *
648    *
649    *
650    *
651    *
652    *
653    *
654    *
655    *
656    *
657    *
658    *
659    *
660    *
661    *
662    *
663    *
664    *
665    *
666    *
667    *
668    *
669    *
670    *
671    *
672    *
673    *
674    *
675    *
676    *
677    *
678    *
679    *
680    *
681    *
682    *
683    *
684    *
685    *
686    *
687    *
688    *
689    *
69
```

Listing E-1. (continued)

```

65      *
66      *      open file to copy
67      *
68      0000008A 303C000F      move.w  $open,d0
69      0000008E 223900004080  move.l  fcb,d1
70      00000094 4E42          trap      #2
71      00000096 0C4000FF      cmpi.w  #$50ff,d0
72      0000009A 660C          bne      openok
73      0000009C 223C00000034  move.l  topnfl,d1
74      000000A2 4EF9000001D2  jmp      err
75      000000A8 207900004080  openok: move.l  fcb,a0
76      000000AE 422H0020      clr.b   32(a0)
77      *
78      *      read
79      *
80      000000B2 243C00000000  move.l  sbuf,d2
81      000000B8 42790000408E  clr.w   count
82      000000BE 303C001A      rloop:  move.w  rdsetdma,a0
83      000000C2 2202          move.l  d2,d1
84      000000C4 4E42          trap      #2
85      000000C6 303C0014      move.w  $readseq,d0
86      000000CA 223900004080  move.l  fcb,d1
87      000000D0 4E42          trap      #2
88      000000D2 4A40          tst.w   d0
89      000000D4 661A          bne      wrtout
90      000000D6 D45C00000080  add.l   $128,d2
91      000000DC 52790000408E  add.w   $1,count
92      000000E2 0C7900000000408E  cmpi.w  $outcnt,count
93      000000E4 6E0000FE      bgt      outofix
94      000000E6 60CE          ora      rloop
95      *
96      *      write
97      *
98      000000F0 303C0009      wrtout: move.w  $seidsk,d0      select the disk
99      000000F4 22390000408A  move.w   dsk,d1
100     000000FA 4202          clr.b   d2
101     000000FC 4E43          trap      #3
102     000000FE 4A80          tst.l   d0      check for select error
103     00000100 670000D8      beq      selerrx
104     00000104 2040          move.l   d0,a0
105     00000106 0668000E      move.l  14(a0),a0      get DPd address
106     0000010A 33D000004084      move.w  (a0),spt      get sectors per track
107     00000110 33E8000000408C      move.w  14(a0),off    get offset
108     00000118 227900004088      clr.w   trk          start at trk 0
109     0000011K 33FC000100004086      move.w  $1,sect      start at sector 1
110     00000126 11F900000000      lea     buf,a0
C P / M 6 8 0 0 0 A s s e m b l e r      Revision 02.01      Page 3
Source File: putboot.s

111     0000012C 4A3900004094      tst.b   hflag
112     00000132 660C          bne      wrtl
113     00000134 0C50601A      cmpi.w  #$601a,(a0)
114     00000138 6606          bne      wrtl
115     0000013A D1FC0000001C      add.l   $18,a0
116     00000140 23C800004090      wrtl:  move.l  a0,bufp
117     *
118     00000146 4A790000408E      wloop:  tst.w   count
119     0000014C 6774          beq      exit
120     0000014E 223900004086      move.w  sect,d1      check for end-of-track
121     00000154 827900004084      cmp.w   spt,d1
122     0000015A 6F1E          blt      sok
123     0000015C 33FC000100004086      move.w  $1,sect      advance to new track
124     00000164 303900004088      move.w  trk,d0
125     0000016A 5240          add.w   $1,d0
126     0000016C 33C000004088      move.w  d0,trk
127     00000172 80790000408C      cmp.w   off,d0
128     00000178 6C78          bge      ofix
129     0000017A 303C000A      sok:  move.w  $settrk,d0      set the track
130     0000017E 223900004088      move.w  trk,d1
131     00000184 4E43          trap      #3
132     00000186 223900004086      move.w  sect,d1      set sector
133     0000018C 303C0008      move.w  $setsec,d0
134     00000190 4E43          trap      #3
135     00000192 303C000C      move.w  $isetdma,d0      set up dma address for write
136     00000196 223900004090      move.l  bufp,d1
137     0000019C 4E43          trap      #3
138     0000019E 303C000E      move.w  $write,d0      and write
139     000001A2 4241          clr.w   d1
140     000001A4 4E43          trap      #3
141     000001A6 4A40          tst.w   d0      check for write error

```

Listing E-1. (continued)

```

142 000001A8 6638          bne      wrtrrx          increment sector number
143 000001AA 527900004086        add      $1,sect
144 000001B0 53790000408E        sub      $1,count
145 000001B6 06B90000008000004090      add.l    $128,bufp
146 000001C0 6084          bra      wloop
147
148 000001C2 303C0015        exit:    move.w  $flush,d0      exit location - flush bios buffers
149 000001C6 4E43          trap      $3
150 000001C8 4E5E          unlk     a6
151 000001CA 4E75          rts          and exit to CCP
152
153 000001CC 223C00000000      crxit:   move.l    $erstr,d1      miscellaneous errors
154 000001D2 303C0009      crx:    move.w    $prntstr,d0     print error message and exit
155 000001D6 4E42          trap      $2
156 000001D8 60E8          bra      exit
157
158 000001DA 223C000000017    selerr: move.l    $selstr,d1      disk select error
159 000001E0 60F0          bra      err
160 000001E2 223C000000026    wrtrrx: move.l    $wrtstr,d1      disk write error
161 000001E8 60E8          bra      err
162 000001EA 223C00000004E    bufoflx: move.l   $bufofl,d1     buffer overflow
163 000001F0 60E0          bra      err
164 000001F2 223C000000060    oflex:  move.l    $trkofl,d1
165 000001F8 60D8          bra      err
C P / M 6 8 0 0 0   A s s e m b l e r          Revision 02.01          Page 4
Source File: putboot.s

166
167
168 00000000          .bss
169
170          .even
171
172 00000000      buf:    .ds.b    bufsize+128
173
174 00004080      fcb:    .ds.l    1          fcb address
175 00004084      spt:    .ds.w    1          sectors per track
176 00004086      sect:   .ds.w    1          current sector
177 00004088      trk:    .ds.w    1          current track
178 0000408A      dsk:    .ds.w    1          selected disk
179 0000408C      off:    .ds.w    1          1st track of non-boot area
180 0000408E      count:   .ds.w    1
181 00004090      bufp:    .ds.l    1
182 00004094      hflag:   .ds.b    1
183
184 00004096          .data
185
186 00000000 496E76616C696420      erstr:   .dc.b    'Invalid Command Line',13,10,'$'
187 00000008 436F6D6416E6420
188 00000010 4C696550D0A24
189 00000017 53656C6563742045      selstr:  .dc.b    'Select Error',13,10,'$'
190 0000001F 72726F72D0A24
191 00000026 5772697465204572      wrtstr:  .dc.b    'Write Error',13,10,'$'
192 0000002E 746F72D0A24
193 00000034 43616E656F74204F      opnfl:   .dc.b    'Cannot Open Source File',13,10,'$'
194 0000003C 70656E20536F7572
195 00000044 63652046696C65D0
196 0000004C 0A24
197 0000004E 427566666572204F      bufofl:  .dc.b    'Buffer Overflow',13,10,'$'

```

Listing E-1. (continued)

```

190 00000056 76657266C6777UD
190 0000005E 0A24
191 00000060 546FbF204D756366      trk01: .uc.b  'Too Much Data for System Tracks'.13.10.'5'
191 00000068 20446.74612066bF
191 00000070 72205379737466bC
191 00000078 20547261636673UD
191 00000080 0A24
192
193
194 00000082                      .end
C P / M 6 8 0 0 0  A s s e m b l e r      Revision 02.01      Page 5
Source File: putboot.s

```

Symbol Table

buf	00000000 BSS	bufcnt	00000080 ABS	buf011	0000004E DATA	buf01x	000001EA TEXT
buip	00004090 BSS	bufsize	00004000 ABS	count	0000408E BSS	iseidsk	0000000E ABS
dsbtdma	0000001A ABS	ask	0000406A BSS	erstr	00000000 DATA	erx	000001D1 TEXT
exit	000001CC TEXT	exit	000001C2 TEXT	fc0	00004080 BSS	ilush	00000015 ABS
hziag	00004094 BSS	isetdma	0000000C ABS	nonyph	00000030 TEXT	oif	0000406C BSS
oflex	000001F4 TEXT	open	000000JF ABS	open0x	000000A0 TEXT	opnfl	00000034 DATA
prntstr	00000009 ABS	realseq	00000014 ABS	rloop	0000006E TEXT	scan	0000001C TEXT
scan1	00000024 TEXT	scan2	0000003C TEXT	sect	00004060 BSS	sectran	00000010 ABS
seldsk	00000009 ABS	se1erx	000001DA TEXT	se1str	00000017 DATA	setsec	0000000B ABS
settrk	0000000A ABS	sok	0000017A TEXT	spt	00004064 BSS	start	00000000 TEXT
trk	00004086 BSS	trk01	00000060 DATA	upper	0000006C TEXT	w_cop	00000140 TEXT
write	0000000E ABS	wrt1	00000140 TEXT	wrt0x	000001E2 TEXT	wrtout	000001F0 TEXT
wrtstr	00000016 DATA						

End of Appendix E

Appendix F Motorola S-Records

F.1 S-record Format

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

An S-record file consists of a sequence of S-records of various types. The entire content of an S-record is ASCII. When a hexadecimal number needs to be represented in an S-record it is represented by the ASCII characters for the hexadecimal digits comprising the number. Each S-record contains five fields as shown in Figure F-1.

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

Figure F-1. S-record Fields

The field contents are as follows:

Table F-1. S-record Field Contents

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

Table F-1. (continued)

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

F.2 S-record Types

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

Table F-2. S-record Types

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

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

End of Appendix F

Appendix G

CP/M-68K Error Messages

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

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

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

Message	Meaning
bad relocation information bits	<p>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. <u>The CP/M-68K Operating System Programmer's Guide</u> 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.</p>
BIOS ERROR -- DISK X NOT SUPPORTED	<p>BIOS. The disk drive indicated by the variable X is not supported by the BIOS. The BDOS supports a maximum of 16 drives, lettered A through P. Check the documentation provided by the manufacturer for your particular system configuration to find out which of the BDOS drives your BIOS implements. Specify the correct drive code and reenter the command line.</p>

Table G-1. (continued)

Message	Meaning
BIOS ERROR -- Invalid Disk Status	<p>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.</p> <ol style="list-style-type: none">1) Indicate which version of the operating system you are using.2) Describe your system's hardware configuration.3) Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.
Buffer Overflow	<p>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.</p>
Cannot Open Source File	<p>PUTBOOT. PUTBOOT cannot locate the source file. Ensure that you specify the correct drive code and filename before you reenter the PUTBOOT command line.</p>

Table G-1. (continued)

Message	Meaning				
CP/M Disk change error on drive x	<p>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.</p>				
CP/M Disk file error: filename is read-only. Do you want to: Change it to read/write (C), or Abort (A)?	<p>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.</p> <p>If you specify the A option or a CTRL-C, the program terminates and CPM-68K returns the system prompt.</p>				
CP/M Disk read error on drive x Do you want to: Abort (A), Retry (R), or Continue with bad data (C)?	<p>BDOS. This message indicates a hardware error. Specify one of the options enclosed in parentheses. Each option is described below.</p> <table> <tr> <th>Option</th><th>Action</th></tr> <tr> <td>A or CTRL-C</td><td>Terminates the operation and CP/M-68K returns the system prompt.</td></tr> </table>	Option	Action	A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.
Option	Action				
A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.				

Table G-1. (continued)

Message	Meaning						
	<table> <tr> <th>Option</th><th>Action</th></tr> <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> </table>	Option	Action	R	Retries operation. If the retry fails, the system reprompts with the option message.	C	Ignores error and continues program execution. Be careful if you use this option. Program execution should not be continued for some types of programs. For example, if you are updating a data base and receive this error but continue program execution, you can corrupt the index fields and the entire data base. For other programs, continuing program execution is recommended. For example, when you transfer a long text file and receive an error because one sector is bad, you can continue transferring the file. After the file is transferred, review the file, and add the data that was not transferred due to the bad sector.
Option	Action						
R	Retries operation. If the retry fails, the system reprompts with the option message.						
C	Ignores error and continues program execution. Be careful if you use this option. Program execution should not be continued for some types of programs. For example, if you are updating a data base and receive this error but continue program execution, you can corrupt the index fields and the entire data base. For other programs, continuing program execution is recommended. For example, when you transfer a long text file and receive an error because one sector is bad, you can continue transferring the file. After the file is transferred, review the file, and add the data that was not transferred due to the bad sector.						
<p>CP/M Disk write error on drive x Do you want to: Abort (A), Retry (R), or Continue with bad data (C):</p> <p>BDOS. This message indicates a hardware error. Specify one of the options enclosed in parentheses. Each option is described below.</p> <table> <tr> <th>Option</th><th>Action</th></tr> <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.</td></tr> </table>		Option	Action	A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.	R	Retries operation. If the retry fails, the system reprompts with the option message.
Option	Action						
A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.						
R	Retries operation. If the retry fails, the system reprompts with the option message.						

Table G-1. (continued)

Message	Meaning				
	<table> <tr> <th>Option</th><th>Action</th></tr> <tr> <td>C</td><td> <p> 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.</p> </td></tr> </table>	Option	Action	C	<p> 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.</p>
Option	Action				
C	<p> 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.</p>				
CP/M Disk select error on drive x Do you want to: Abort (A), Retry (R)	<p>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.</p>				
CP/M Disk select error on drive x	<p>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.</p>				

Table G-1. (continued)

Message	Meaning
File already exists	<p>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.</p>
insufficient memory or bad file header	<p>CCP. This error could result from one of three causes:</p> <ol style="list-style-type: none">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 <u>CP/M-68K Operating System Programmer's Guide</u> or the <u>CP/M-68K Operating System User's Guide</u> 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.2) The program is too large for the available memory. Add more memory boards to the system configuration, or rewrite the program to use less memory.3) The program is linked to an absolute location in memory that cannot be used. The program must be made relocatable, or linked to a usable memory location. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the memory space that is available for loading programs.

Table G-1. (continued)

Message	Meaning
Invalid Command Line	<p>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.</p>
No file	<p>CCP. The filename specified in the command line does not exist. Ensure that you use the correct filename and reenter the command line.</p>
No wildcard filenames	<p>CCP. The command specified in the command line does not accept wildcards in file specifications. Retype the command line using a specific filename.</p>
Program Load Error	<p>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.</p> <ol style="list-style-type: none">1) Indicate which version of the operating system you are using.2) Describe your system's hardware configuration.

Table G-1. (continued)

Message	Meaning
Program Load Error (continued)	<p>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.</p>
read error on program load	<p>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.</p>
Select Error	<p>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.</p>
SUB file not found	<p>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.</p>

Table G-1. (continued)

Message	Meaning
Syntax: REN newfile=oldfile	<p>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.</p>
Too many arguments: argument?	<p>CCP. The command line contains too many arguments. The extraneous arguments are indicated by the variable argument. Refer to the <u>CP/M-68K Operating System User's Guide</u> 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.</p>
Too Much Data for System Tracks	<p>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.</p> <p>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.</p>

Table G-1. (continued)

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

End of Appendix G

Appendix H

VME/10 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. The file VT52.C contains code to emulate a VT52 terminal. This file constitutes part of a normal BIOS for the VME/10. It should be compiled and linked with the other BIOS files and CPMLIB when building a system for the VME/10. Finally, there is an include file that defines some standard variable types.

Listing H-1. VME/10 BIOS.C File

```

/*      @(#)bios.c      2.10      */
/*=====*/
/*-----*/
/*      CP/M-68K(tm) BIOS for the MOTOROLA VME/10      */
/*      Copyright 1984, Motorola Inc.                  */
/*      Created 1/27/83 lrj, jek, bjp                  */
/*-----*/
/*=====*/

#include "biostype.h" /* defines LOADER : 0-> normal bios, 1->loader bios */
/* MEMDSK: 0 -> no memory disk */
/*      4 -> memory disk(sized at boot, drive E:)*
/* DISKB: 0 -> no disk B: (second floppy)
/*      1 -> disk B: present
/* DISKC: 5 -> 5 megabyte hard disk for disk C:
/*      15 -> 15 megabyte hard disk for disk C:
/* DISKD: 0 -> no disk D: (second hard disk)
/*      5 -> 5 megabyte hard disk for disk D:
/*      15 -> 15 megabyte hard disk for disk D:

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

char copyright[] = "Copyright 1984, Motorola Inc.";
char biosrev[] = "BIOS 2.10";

```

Listing H-1. (continued)

```

#define NULL      (0L)

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

/*      for VME-10 defined separately with each module below      */

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

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

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

#if 1 LOADER

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

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

/*
 *      used to interface cons_stat() with cons_in().  in_char
 *      is a one character buffer.
 */

char  in_char = '0';
char  char_avail = 0;

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

/*
 *      VME/10 Console interface
 */

/* defines used for VME/10 Control Register #2
 * initial values for CR2:
 *
 *      bit 0 - enable MMU interface
 *      bit 1 - disable keyboard transmit interrupt
 *      bit 2 - enable BCLR line
 *      bit 3 - ?
 *      bit 4 - disable keyboard RESET

```

```

*      bit 5 - disable RAM writes by other devices
*      bit 6 - enable SYSFAIL interrupts
*      bit 7 - enable keyboard receive interrupt
*/
#define CR2_INIT          0x0d
#define CR2_ADDR          0xf19f09 /* address of control register 2 */
extern VOID      keybd_init();      /* vme/10 keyboard initialization */
VOID c_init() /* INITIALIZE THE SYSTEM CONSOLE */
{
    *((char *)CR2_ADDR) = CR2_INIT; /* initialize Control Register 2;
                                     ostensibly to disable transmit
                                     interrupts from the keyboard */
    keybd_init(); /* initialize the keyboard */
}

```

```

.....
/*      VME-10 keyboard input      */
.....

```

```

KEYBOARD -- ROUTINES FOR HANDLING THE VME/10 KEYBOARD.

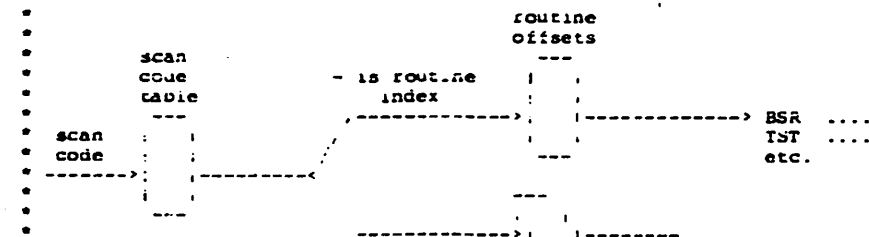
ENVIRONMENT:  This is part of the VME/10 keyboard/screen driver.

FUNCTION:  These routines provide initialization and interrupt
service for the VME/10 keyboard, which is connected to the
processor board through a 2661 half-duplex serial port.

NOTES:  This module is very much table-driven, so pains have been
taken to reduce the size of the tables.  You will find byte and
word offsets instead of longword addresses, and where feasible
functions are used instead of tables to generate characters.

```

/* DIAGRAM DEPICTING THE ROLES OF THE VARIOUS TABLES



Listing H-1. (continued)

```

*
*      + is group      | |      |      char
*      index           | |      |      table
*
*
*      parameter      group
*      [-----]      offsets
*      [-----]      [-----]
*
*
* When a scan code is received, it is used as an index into the SCAN CODE
* TABLE to get a group/routine index and a parameter.
*
* If the group/routine index is positive, then it is a group index. The
* corresponding offset is taken from the GROUP TABLE, giving the address
* of the character table currently in effect for that group. The parameter
* is used as an index into the character table to retrieve the character.
* If the entry in the character table is 0, no character is generated.
*/

VOID      mode_chg();

typedef struct      /* entry in the scan code table */
{
    char      grp_rout_x;      /* group/routine index */
    char      arg;      /* a parameter */
} sc_tbl;

/* MODE TYPES */

#define BLANK      0x01      /* blank locking mode key is down */
#define CTRL      0x02      /* CTRL is down. */
#define SHIFT      0x04      /* one or both of shift keys is down */
#define PAD      0x08      /* FUNC/PAD is down (PAD). */
#define ALPHA_LOCK      0x10      /* ALPHA LOCK is down. */
#define ALT      0x20      /* ALT is down. */
#define ALOCK_OR_SHIFT      0x14      /* either lock of shift are down */

/* values generated by function keys. */

#define FNKEY_F1      0xa0      /* Value to generate for func. key 1 */
#define FNKEY_S1      0xb0      /* Value for shifted func. key 1. */

#define STATUS_REG      0xf19f85      /* Address of VMEC1 status register. */
#define KBD_UNLOCK      0x10      /* status of the lock on the front
panel, which we interpret as a
keyboard lock: 0 if keyboard
locked; 1 if unlocked */

#define LEFT      0x01      /* left shift key is down */
#define RIGHT      0x02      /* right shift key is down */

static char      modes;      /* value of modes reflect the status

```

Listing H-1. (continued)

```

static char      shift_keys;          /* (up | down) of mode keys on */
/* Bits in this byte reflect status */
/* (up * or down) of two SHIFT keys. */

/* Declarations for the 2661 (EPCI) connecting us to the keyboard. */

#define EPCI_ADDR      0xf1a031      /* Base address of the 2661. */
typedef struct          /* map of the 2661 registers */
{
    char    data;          /* data register */
    char    fill12, status; /* status register */
    char    fill14, mode;   /* mode register */
    char    fill16, command; /* command register */
} epci_map;
/* Bit definitions for the 2661 status register. */

#define CHAR_AVAIL      0x02          /* a received char is in DATA reg. */
#define PARITY          0x08          /* received char had a parity error. */
#define OVERRUN         0x010        /* receive buffer was overrun. */
#define FRAMING         0x020        /* received char had a framing error. */
#define ANY_ERROR       0x038        /* Mask selecting all 3 errors. */

/* Initialization values for the 2661 control registers. */

#define MODE1_INIT      0x5e          /* Initialization for mode reg 1 */
#define MODE2_INIT      0x7b          /* Initialization for mode reg 2 */
#define COMM_INIT       0x15          /* Initialization for command reg */

/* Commands for keyboard. In each, bits 0-3 = $0 are keyboard's ID. */

#define SELECT_KB        0x80          /* SELECT command. */
#define READ_KB          0xa0          /* READ command. */
#define AGAIN_KB         0xd0          /* AGAIN command; this causes last
                                        scan code sent to be repeated. */

/* Declarations for control register 2 on the VME/10 processor board. */

typedef struct          /* description of control register 2 */
{
    char    cr2;
} cr2_map;
#define CR2_ADDR         0xf19f09      /* Location of control register 2. */
#define KB_RESET         0x10          /* when low, holds keyboard in reset. */
#define KB_ENABLE        0x80          /* when high, enables receipt
                                        interrupts from the 2661 which
                                        connects us to the keyboard. */

/* status values returned by keybd_read */

#define NOCHAR           0             /* no character was returned */
#define BREAK            1             /* break key was struck */
#define KYBD_LOCKED      2             /* key on front panel in lock pos. */

```

Listing H-1. (continued)

```

#define IN_CHAR      3          /* a real live character available */
#define WAIT_COUNT   2          /* Initial loop value for timing loop
                                in keyboard reset. */

#define AT_CHAR      '@'
#define TWO_CHAR     '2'

/*****

ROUTINE AND GROUP OFFSET TABLES

FUNCTION: Each entry in the scan code table contains a byte value
          called the group/routine index. If it is negative it is a routine
          index and is used as the index in a switch statement.
          If it is positive, it is a group index and is
          an offset into GRP_TBL. These tables contain word offsets to
          routines or tables in this module. The whole purpose for these guys
          is to allow each scan code table entry to be just 2 bytes long.

*****/
/*
 * If the group/routine index is negative, then it is a routine index. The
 * index is one of the enumerated types "rtn_index" and is used as the
 * switch index in a switch statement. The parameter may or may not be
 * used by the routine.
 *
 * Also note that rtn_index starts at 1 so that there is no entry #0 in
 * the routine table; this is necessary since negative 0 is 0 and that index
 * would be indistinguishable from the positive 0 index in the group table.
 *
 * Note: the NONLOCKABLE enumerated type. Routine indexes which
 * are above that type will not be executed while the keyboard lock on the front
 * below that point are called regardless of the keyboard lock's status.
 */

#define TWO_AT      1
#define ESC_RESET   2
#define BREAK_CLEAR 3
#define GEN_ALPHA   4
#define FNKEY       5
#define NONLOCKABLE 5
#define SET_SHIFT   6
#define CLR_SHIFT   7
#define SET_MODE    8
#define CLR_MODE    9
#define IGNORE      10

typedef int rtn_index;

/*****

```

Listing H-1. (continued)

```

ALPHA('N')          /* 3C -- N key. */
ALPHA('M')          /* 3D -- M key. */
GROUP(1,20)         /* 3E -- , < key. */
RTN(CLR_MODE,CTRL) /* 3F -- CTRL key released. */
RTN(IGNORE,0)       /* 40 -- Ignore it. */
RTN(FNKEY,7)        /* 41 -- Function key 7. */
RTN(FNKEY,8)        /* 42 -- Function key 8. */
GROUP(1,9)          /* 43 -- 0 ) key. */
GROUP(1,10)         /* 44 -- - _ key. */
GROUP(1,11)         /* 45 -- = + key. */
ALPHA('O')          /* 46 -- O key. */
ALPHA('P')          /* 47 -- P key. */
GROUP(1,14)         /* 48 -- [ { key. */
ALPHA('L')          /* 49 -- L key. */
GROUP(1,17)         /* 4A -- : ; key. */
GROUP(1,18)         /* 4B -- ' " key. */
GROUP(1,21)         /* 4C -- . */
GROUP(1,22)         /* 4D -- / ? key. */
RTN(IGNORE,0)       /* 4E -- Ignore it. */
RTN(CLR_MODE,ALT)   /* 4F -- ALT key released. */
RTN(IGNORE,0)       /* 50 -- Ignore it. */
RTN(FNKEY,9)        /* 51 -- Function key 9. */
RTN(FNKEY,10)       /* 52 -- Function key 10. */
RTN(FNKEY,11)       /* 53 -- Function key 11. */
RTN(FNKEY,12)       /* 54 -- Function key 12. */

GROUP(1,12)         /* 55 -- DEL key. */
RTN(BREAK CLEAR, 0) /* 56 -- BREAK CLEAR key. */
GROUP(1,15)         /* 57 -- ] } key. */
GROUP(1,16)         /* 58 -- | key. */
GROUP(0,1)          /* 59 -- Home key. */
GROUP(1,19)         /* 5A -- Return key. */
GROUP(0,3)          /* 5B -- Backarrow key. */
RTN(IGNORE,0)       /* 5C -- SEL key. */
GROUP(0,5)          /* 5D -- Backtab key. */
GROUP(0,6)          /* 5E -- Downarrow key. */
RTN(CLR_MODE,BLANK) /* 5F -- lock mode key released. */
RTN(CLR_MODE,PAD)   /* 60 -- PAD key released. */
RTN(FNKEY,13)       /* 61 -- Function key 13. */
RTN(FNKEY,14)       /* 62 -- Function key 14. */
RTN(FNKEY,15)       /* 63 -- Function key 15. */
RTN(FNKEY,16)       /* 64 -- Function key 16. */
RTN(ESC RESET, 0)  /* 65 -- ESC RESET key. */
GROUP(2,10)         /* 66 -- A DCHR key. */
GROUP(0,2)          /* 67 -- Uparrow key. */
RTN(IGNORE,0)       /* 68 -- CLR TAB SET key. */
GROUP(2,7)          /* 69 -- 7 EOF key. */
GROUP(0,4)          /* 6A -- Rightarrow key. */
GROUP(2,4)          /* 6B -- 4 key on hexpad. */
GROUP(0,7)          /* 6C -- Tab key on cursor pad. */
GROUP(2,1)          /* 6D -- 1 ICHR key. */
GROUP(2,0)          /* 6E -- 0 on hexpad. */
RTN(IGNORE,0)       /* 6F -- Clacker mode key release */
RTN(IGNORE,0)       /* 70 -- Ignore it. */
GROUP(2,11)         /* 71 -- B DLINE key. */

```

Listing H-1. (continued)

```

GROUP(2,12)      /* 72 -- C on hexpad.      */
GROUP(2,13)      /* 73 -- D on hexpad.      */
GROUP(2,8 )      /* 74 -- 8 EOL key.       */
GROUP(2,9 )      /* 75 -- 9 EOP key.       */
GROUP(2,14)      /* 76 -- E EAU key.       */
GROUP(2,5 )      /* 77 -- 5 on hexpad.     */

GROUP(2,6 )      /* 78 -- 6 on hexpad.     */
GROUP(2,15)      /* 79 -- F on hexpad.     */
GROUP(2,2 )      /* 7A -- 2 ILINE key.     */
GROUP(2,3 )      /* 7B -- 3 on hexpad.     */
GROUP(2,16)      /* 7C -- TEST , key.     */
GROUP(2,17)      /* 7D -- HELP .          */
GROUP(0,0 )      /* 7E -- ENTER key.      */
RTN(IGNORE,0)    /* 7F -- Ignore it.      */
};

/*
 * NOTE: DUE TO A COMPILER BUG WE ARE UNABLE TO INITIALIZE
 * CHAR VARIABLES AS MEMBERS OF A STRUCTURE, SO WE HAVE DECLARED
 * THE TABLES AS CONTIGUOUS CHAR ARRAYS AND ASSIGN THE ORIGINAL
 * STRUCTURE ARRAY AS A PTR TO THE CHAR ARRAY OF INITIALIZED
 * DATA. THIS WAS FELT TO BE THE MOST DIRECT SOLUTION.
 */

static sc_tbl  *scan_code_tbl = (sc_tbl *)c_scan_code_tbl;

/*****

GROUP 0 CHARACTER TABLE -- KEYS UNAFFECTED BY MODE KEYS

FUNCTION: This table contains the characters to return for scan
codes which fall into group 0. Use the parameter obtained from
the scan code table as an index into this table. A value of 0
will cause no character to result.

Table TBL.___ is used regardless of any mode keys.

*****/

static char  tbl___[] = {
    0x0d,      /* element # 0 -- CR (from the ENTER key). */
    0xc0,      /* 1 -- Home. */
    0x0b,      /* 2 -- VT (from the uparrow key). */
    0x08,      /* 3 -- BS (from the leftarrow key). */
    0x0c,      /* 4 -- FF (from the rightarrow key). */
    0xdb,      /* 5 -- Backtab. */
    0x0a,      /* 6 -- LF (from the downarrow key). */
    0x09,      /* 7 -- HT (from the tab key on the */
};

/*****

GROUP 1 CHARACTER TABLES -- KEYS AFFECTED BY SHIFT AND CTRL KEYS.

```

Listing H-1. (continued)

FUNCTION: These tables contain the characters to return for scan codes which fall into group 1. Use the parameter obtained from the scan code table as an index into these tables. A value of 0 will cause no character to result.

Table TBL.NN_ is used when we're in neither SHIFT nor CTRL .
Table TBL.XY_ is used when we're in CTRL mode (SHIFTed or).

Table TBL.YN_ is used when we're in SHIFT but not CTRL mode.

```

*****/

static char    tblnn_[] = {      /* element #          */
    '\0',      /* 0.                  */
    '1',      /* 1.                  */
    '3',      /* 2.                  */
    '4',      /* 3.                  */
    '5',      /* 4.                  */
    '6',      /* 5.                  */
    '7',      /* 6.                  */
    '8',      /* 7.                  */
    '9',      /* 8.                  */
    '0',      /* 9.                  */
    '-',      /* 10.                 */
    '=',      /* 11.                 */
    0x7F,     /* 12 -- DEL.          */
    0x09,     /* 13 -- HT (tab key on keyboard). */
    '\t',     /* 14.                 */
    '\n',     /* 15.                 */
    '\r',     /* 16.                 */
    '\f',     /* 17.                 */
    0x27,     /* 18 -- Apostrophe.   */
    0x0D,     /* 19 -- CR (from return key). */
    '\0',     /* 20.                 */
    '\0',     /* 21.                 */
    '\0',     /* 22.                 */
    '\0',     /* 23.                 */
};

static char    tblxy_[] = {      /* element #          */
    0,        /* 0.                  */
    0,        /* 1.                  */
    0,        /* 2.                  */
    0,        /* 3.                  */
    0,        /* 4.                  */
    0x1e,     /* 5 -- CTRL ~.        */
    0,        /* 6.                  */
    0,        /* 7.                  */
    0,        /* 8.                  */
    0,        /* 9.                  */
    0x1f,     /* 10 -- CTRL __.      */
    0,        /* 11.                 */
    0,        /* 12.                 */
};

```

Listing H-1. (continued)

```

0 , /* 13. */
0x1b , /* 14 -- CTRL [. */
0x1d , /* 15 -- CTRL ]. */
0x1c , /* 16 -- CTRL . */
0 , /* 17. */
0 , /* 18. */
0 , /* 19. */
0 , /* 20. */
0 , /* 21.

0 , /* 22.
0 , /* 23.
    };

static char  tblyn_[] = {
    /* element #
    /* -----
    /* 0.
    /* 1.
    /* 2.
    /* 3.
    /* 4.
    /* 5.
    /* 6.
    /* 7.
    /* 8.
    /* 9.
    /* 10.
    /* 11.
    /* 12 -- DEL.
    /* 13 -- HT (tab key on main keyboard)*/
    /* 14.
    /* 15.
    /* 16.
    /* 17.
    /* 18.
    /* 19 -- CR (from return key).
    /* 20.
    /* 21.
    /* 22.
    /* 23.
    };
/*****

```

GROUP 2 CHARACTER TABLES -- KEYS AFFECTED ONLY BY FUNC/PAD KEY.

FUNCTION: These tables contain the characters to return for scan codes which fall into group 2. Use the parameter obtained from the scan code table as an index into these tables. A value of 0 will cause no character to result.

Table TBL.__N is used when we're not in PAD mode.

Table TBL.__Y is used when we are in PAD mode.

These tables contain characters associated with the hexpad. Note,

Listing H-1. (continued)

however, that the ENTER key is not in these tables.

...../

```
static char    tbl_n[j] = {
/* element #
/* -----
/* 0.
/* 1. -- ICHR.
0xD0,
/* 2. -- ILINE.
0xD6,
/* 3.
0,
/* 4.
0,
/* 5.
0,
/* 6.
0,
/* 7.
0xD5,
/* 8. -- EOL.
0xD4,
/* 9. -- EOP.
0xD1,
/* 10. -- DCHR.
0xD7,
/* 11. -- DLINE.
0,
/* 12. -- PMODE.
0,
/* 13.
0,
/* 14.
0,
/* 15.
0,
/* 16.
0,
/* 17.
};
```

```
static char    tbl_y[i] = {
/* element #
/* -----
/* 0.
/* 1.
/* 2.
/* 3.
/* 4.
/* 5.
/* 6.
/* 7.
/* 8.
/* 9.
/* 10.
/* 11.
/* 12.
/* 13.
/* 14.
/* 15.
/* 16.
/* 17.
'0',
'1',
'2',
'3',
'4',
'5',
'6',
'7',
'8',
'9',
'A',
'B',
'C',
'D',
'E',
'F',
',',
'.'
};
```

};

```
/*
* grp_tbl_ptr is used when the group/routine index for a scan code is
* positive ( >=0 ). As mode keys are pressed, the address in the table
* may get replaced.
*/
```


Listing H-1. (continued)

```

static char    *grp_tbl_ptr[3] = {
    tbl_,      /* Group 0: no mode keys affect these.*/
    tbinn_,    /* Group 1: SHIFT and CTRL affect */
    tbl_n,     /* Group 2: PAD affects these. */
};

/* table of offsets for group 1. One is
   selected based on CTRL and SHIFT keys*/
static char    *grpl_off[] = {
    tbinn_,    /* no SHIFT, no CTRL */
    tblxy_,    /* no SHIFT, CTRL */
    tblyn_,    /* SHIFT, no CTRL */
    tblxy_,    /* SHIFT, CTRL */
};

/* delay for a multiple of 4 usecs; which is the time it takes to
 * traverse the loop once on MC68010
 *
 * since the CP/M compiler doesn't support blocked variables
 * we need to declare a register int in the function
 * itself
 */
#define DELAY(n)    delayvar=n; do; while(--delayvar);

/*****

    keybd_init -- INITIALIZE THE VME/10 KEYBOARD

FUNCTION: This routine, which is XDEFed, resets the keyboard,
configures the 2661 so we can communicate with the keyboard,
selects the keyboard, and disables keyboard interrupts.

Currently, however, we are not checking the self-test status,
and therefore there is no possibility of an error return. The
hooks are left in to facilitate later enhancements.

*****/

keybd_init()
{
    register int    delayvar;    /* req'd for DELAY macro!!! */
    register epci_map *epci_base;
    register cr2_map *cr2_base;
    register char    cr2_state;
    register char    cr;    /* to hold contents of control reg. */

    epci_base = (epci_map *)EPCI_ADDR;
    cr2_base = (cr2_map *)CR2_ADDR;

    /* reset the keyboard */

    cr2_base->cr2 &= (char) (~KB_ENABLE); /* disable keyboard interrupts */

```

Listing H-1. (continued)

```

cr2_base->cr2 &= (~KB_RESET);          /* set RESET line to keyboard */
DELAY(2);                             /* need to wait about 5 usecs */
cr2_base->cr2 |= KB_RESET;
DELAY(100000);                         /* allow keyboard to get it's act
                                     together; about 1/2 second */
/* Initialize the 2661. Because 2661 mode register is really two registers
   and the one you're writing to depends upon sequence of prior writes, we
   must do this with interrupts disabled. */

cr = epci_base->command;                /* read command register, which causes
                                     the mode reg. to point to reg 1. */
epci_base->mode = MODE1_INIT;           /* initialize mode register 1 */
epci_base->mode = MODE2_INIT;           /* initialize mode register 2 */
epci_base->command = COMM_INIT;         /* initialize the command register */

/* Select the keyboard. */
epci_base->data = SELECT_KB;            /* send command to select keyboard */

for (;!(epci_base->status & CHAR_AVAIL);) /* wait for reception of char */
    ;

reg = epci_base->data;                  /* Read response; if it's not an ACK (0)
                                     ignore the status since it is
                                     possible that the self-test failed
                                     'cause someone was holding down a key

                                     send us first scan code */
}

/*****

      keybd_read -- ROUTINE TO GET A CHAR FROM THE KEYBOARD

      FUNCTION: This routine is called when a we want to read a code
                 from the 2661. It gets the code from the 2661, sends
                 another READ to the keyboard, and either JMPs to a routine or
                 converts the scan code into a char, depending on the scan code.
*****/

char  keybd_read(status)
int   *status;          /* used to indicate a brk was struck */
{
    register epci_map   *epci_base; /* pointer to the keyboard registers */

    epci_base = (epci_map *)EPCI_ADDR;

```

Listing H-1. (continued)

```

/* Get scan code and check for errors. */
/*
 *   if we just make a mode change, we immediately
 *   try again to read a character.  we don't pass any
 *   SHIFT codes and such up.
 */
if (!(epci_base->status & ANY_ERROR)) /* NO errors from the keyboard */
{
    scn_code = epci_base->data; /* get scan code from the keyboard */
    /* save arg from the scan table entry for later use */
    scn_tbl_arg = scan_code_tbl[scn_code].arg;
    if (scn_code != 0) /* Verify that bit 8 is 0 */
    {
        epci_base->data = READ_KB; /* send another READ to the keyboard */
        group/routine index and parameter. */

        /* returned to the caller */
        if (group_x < 0) /* negative group_x indicates a switch
                        routine is invoked */
        {
            /* ROUTINE TO INTERPRET SCAN CODE */
            if (group_x > (char)NONLOCKABLE) /* routine affected by front
            panel lock */
            {
                if ((*((char *)STATUS_REG) & KBD_UNLOCK)
                    rtn_x = IGNORE; /* if Lock is on then ignore keystroke*/
            }
            else
            {
                rtn_x = (rtn_index)(group_x); /* negate to get switch index */
                rtn_x = -rtn_x; /* compiler BUG work around */
            }
        }
        switch ((int)rtn_x)
        {
            case TWO_AT :
                /* TWO_AT -- HANDLE THE 2 @ KEY.

                This code is entered when the 2 @ key is pressed. Its purpose
                is to generate the appropriate character code depending on the
                state of the SHIFT and CTRL mode keys. The reason this key
                can't be handled by the tables is that if CTRL is pressed, a NUL
                is reserved to mean NO CHARACTER.

                if (!(modes & CTRL)) /* is the CTRL key down? */
                {
                    /* No, so look further */
                    if (modes & SHIFT) /* is one of SHIFT keys down */
                        rv = '@'; /* return @ character */
                    else
                        rv = '2'; /* return 2 character */
                }

```

Listing H-1. (continued)

```

    }
    *status = IN_CHAR;
    break;

case ESC_RESET :
/*  ESC_RESET -- HANDLE ESC RESET KEY.

This code is entered when the ESC RESET key is pressed.
Its purpose is to generate the appropriate character code
depending on the state of the SHIFT mode keys. */

    if (modes & SHIFT)          /* is a SHIFT key down */
        /*reset_screen()*/;    /* NOT IMPLEMENTED YET */
    else
    {
        rv = 0x1b;              /* character is an esc */
        *status = IN_CHAR;
    }
    break;

case BREAK_CLEAR :
/*  BREAK_CLEAR -- HANDLE BREAK CLEAR KEY

This code is entered when the BREAK CLEAR key is pressed.
Its purpose is to generate the appropriate character code
depending on the state of the SHIFT mode keys. */

    if (modes & SHIFT)
        /*clear_screen()*/;    /* NOT IMPLEMENTED YET */
    else
        *status = BREAK;        /* tell caller break was hit */

case GEN_ALPHA :
/*  GEN_ALPHA -- HANDLE ALPHABETIC KEYS.

This code is entered when an alphabetic key is pressed.
Its purpose is to generate the appropriate character code
depending on the state of the SHIFT, ALPHA LOCK, and CTRL
mode keys. */

    if (modes & CTRL)
        rv = scn_tbl_arg - ('A' - 1);
    else
    {
        if ((modes & ALOCK_OR_SHIFT)) /* is a shift key down? */
            rv = scn_tbl_arg + ('a' - 'A'); /* lower case letter */
        else
            rv = scn_tbl_arg; /* vanilla upper case letter */

        *status = IN_CHAR;
    }
    break;

```

Listing H-1. (continued)

```

case FNKEY :
/* FNKEY -- HANDLE FUNCTION KEYS.

This code is entered when a function key is pressed.
Its purpose is to generate the appropriate character code
depending on the state of the SHIFT mode keys. */

if (modes & SHIFT)
    rv = scn_tbl_arg + FNKEY_S1 - 1;
else
    rv = scn_tbl_arg + FNKEY_F1 - 1;
*status = IN_CHAR;
break;

case SET_SHIFT :
/* SET_SHIFT AND CLR_SHIFT -- SET OR CLEAR 1 OF THE 2 SHIFT KEYS.

Use these routines when a shift key is pressed or released
to update the SHIFT_KEYS byte and make a mode change for
SHIFT if necessary. */

shift_keys |= scn_tbl_arg; /* which shift keys was hit */
mode_chg();
break;

case CLR_SHIFT :

shift_keys &= (~scn_tbl_arg); /* clear previous key */
if (shift_keys == 0) /* both shift keys released */

    modes &= (~SHIFT); /* clear the shift mode */
    mode_chg();
}
break;

case SET_MODE :
/* SET_MODE AND CLR_MODE -- SET OR CLEAR A MODE (E.G., SHIFT).

Use these routines when a mode key is pressed or released
to update the MODES byte and set the appropriate table
offsets in the GRP_TBL. */

modes |= scn_tbl_arg;
mode_chg();
break;

case CLR_MODE :

modes &= (~scn_tbl_arg); /* Clear the specified mode */

```

Listing H-1. (continued)

```

        break;

    case IGNORE :

        break;
    } /* end of switch ((int)rtn_x)*/

    return(rv);
}
else
/* It's not a routine, it's a group, so use the parameter to get
   the char from the table currently in effect for that group. */
{
    /* CHARACTER DESIGNATED BY SCAN CODE */
    if ((* (char *)STATUS_REG) & KBD_UNLOCK) /* is keyboard locked */
    {
        rv = (grp_tbl_ptr[group_x])[scn_tbl_arg];
        *status = IN_CHAR;
        return(rv);
    }
    else /* if keyboard is locked, ignore code */
    {
        *status = KYBD_LOCKED; /* tell caller the keyboard is locked */
        return(0);
    }
}

/* Some kind of error was detected in getting the scan code--try again. */
epc1_base->data; /* read char if haven't done it yet */
epc1_base->data = AGAIN_KB; /* tell keyboard to send last scan code
                             again; just ignore the present one */
return(0);
}

VOID mode_chg()
{
    int i; /* used as index into grpl_off */

    i = (modes & 0xb) >> 1; /* use SHIFT and CTRL bits in mode as
                             an index to reference grpl_off */

    grp_tbl_ptr[i] = grpl_off[i];

    if (modes & PAD; /* is PAD key down */
        grp_tbl_ptr[2] = tbl_y; /* get address of table for Pad chars */
    else /* get address of table for no PAD */
        grp_tbl_ptr[2] = tbl_n;

    return;
}

```

Listing H-1. (continued)

```

/*****
/*      VME-10 input status
*****/

cons_stat()
{
    register char  ret;
    int           status;
    register epci_map *epci_base;

    /*
     *      first check to see if a character is already available
     *
     *      in one character buffer and hasn't been picked up.
     */
    if( char_avail )
        return(0xff);

    /*
     *      no char was already available so check if there
     *      is something from the keyboard.
     */

    epci_base = (epci_map *)EPCI_ADDR;
    if ( !(epci_base->status & CHAR_AVAIL) )
        return(0x00);

    ret = keybd_read(&status);

    /*
     *      there are four status returns from keybd_read()
     *
     *      BREAK - a break character was returned
     *              return 0x00.
     *      IN_CHAR - we have a character returned
     *              return 0xff.
     *      NOCHAR - there is no character currently
     *              available from the keyboard.
     *      KYBD_LOCKED - there is no character available
     *              as the keyboard lock is on.
     */

    if( status == NOCHAR )
        return(0x00);
    else if( status == KYBD_LOCKED )
        return(0x00);
    else if( status == BREAK )
        return(0x00);
    else if( status == IN_CHAR )
    {
        in_char = ret;
        char_avail = 0xff;
        return(0xff);
    }
}

```

Listing H-1. (continued)

```

    }

/*
 *   end of keyboard code
 */

/*****
 *
 *   BIOS Interface to keyboard functions
 *
 *****/

/*****
 *
 *   VME-10 Keyboard Input
 *
 *****/

BYTE   cons_in()
{
    while( ! cons_stat() )           /* wait for input */
        ;

    char_avail = 0;
    return( in_char );
}
#endif /* ! LOADER */

/*****
 *
 *   VME-10 Console output
 *
 *****/

/*   assembly language routine "cons_out" in bios.a.s */

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

#if ! LOADER

bioserr(errmsg)
REG BYTE *errmsg;
{
    printstr("nrBIOS ERROR -- ");
    printstr(errmsg);
    printstr(".nr");
}

printstr(s)           /* used by bioserr */
REG BYTE *s;
{
    while (*s)
        cons_out(*s++);
}

```


Listing H-1. (continued)

```

#else
bioserr()      /* minimal error procedure for loader BIOS */
{
    1 : goto 1;
}

#endif

/*****
/*      Disk I/O Procedures and Definitions      */
*****/

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

#define DSKCNTL ((struct dio *) 0xflc0dl)      /* controller address */

#define LSCTSZ 128          /* logical sector size - 128 bytes */
#define DCXFER 128         /* bytes per disk controller transfer request */

#define FDRM 127            /* floppy disk directory maximum */
#define FDSM96 313         /* floppy disk storage maximum - 96 tpi */
#define FDSM48 153         /* floppy disk storage maximum - 48 tpi */

#if DISKC == 5
#define HDRMC 1023          /* hard disk C: directory maximum */
#define HDSMC 1215         /* hard disk C: storage maximum - 5 MB */
#define CMCYL 305          /* hard disk C: maximum cylinder */
#define CMHD 1             /* hard disk C: maximum head */
#endif

#if DISKC == 15
#define HDRMC 2047          /* hard disk C: directory maximum */
#define HDSMC 3655         /* hard disk C: storage maximum - 15 MB */
#define CMCYL 305          /* hard disk C: maximum cylinder */
#define CMHD 5             /* hard disk C: maximum head */
#endif

#if DISKD == 5
#define HDRMD 1023          /* hard disk D: directory maximum */
#define HDSMD 1215         /* hard disk D: storage maximum - 5 MB */
#define DMCYL 305          /* hard disk D: maximum cylinder */
#define DMHD 1             /* hard disk D: maximum head */
#endif

#if DISKD == 15
#define HDRMD 2047          /* hard disk D: directory maximum */
#define HDSMD 3655         /* hard disk D: storage maximum - 15 MB */
#define DMCYL 305          /* hard disk D: maximum cylinder */
#define DMHD 5             /* hard disk D: maximum head */
#endif

#define FCSVSIZE ((FDRM / 4) + 1)      /* floppy csv size */

```

Listing B-1. (continued)

```

#define FALVSIZE      ((FDSM96 / 8) + 1)      /* floppy alv size */
#define HCSVSIZEC     ((HDRMC / 4) + 1)       /* hard C: csv size */
#define HALVSIZEC     ((HDSMC / 8) + 1)       /* hard C: alv size */
#define HCSVSIZEZ     ((HDRMD / 4) + 1)       /* hard D: csv size */
#define HALVSIZEZ     ((HDSMD / 8) + 1)       /* hard D: alv size */

#define TKBUFSZ (32 * 128)                    /* number of bytes in track buffer */
                                           /* at least as large as a floppy track */

#define DSKSTAT        0x00                    /* commands */
#define DSKRECAL       0x01
#define DSKDFRMT       0x04                    /* format disk */
#define DSKTFRMT       0x06                    /* format track */
#define DSKREAD        0x08

#define DSKSCAN        0x09
#define DSKWRITE       0x0a
#define DSKCONFIG      0xc0

#define CTL512DD       0x34                    /* default command control byte: 48 tpi */
                                           /* IBM sn, DD, 512 bps, DS, no ci, blk drq */
#define CTL128SD       0x04                    /* command control byte: 48 tpi */
                                           /* IBM sn, SD, 128 bps, DS, no ci, blk drq */

#define BUSY           0x80                    /* status bits */
#define DRQ             0x08                    /* data request - can xfer DCXFER bytes */
#define RST            0x80                    /* sense status bit - reset */
#define RDY            0x08                    /* sense status bit - drive ready */
#define NOERR          0x00                    /* sense error code - no error */
#define IDNTFND        0x06                    /* sense error code - ID header not found */
#define CORRDER        0x13                    /* sense error code - correctable read error */

#define TPI96          0x80
#define TPI48          0

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

/* default number of logical sectors per physical sector for each drive */
WORD lperp[NUMDSKS] = { 512/LSCTS, 512/LSCTS, 256/LSCTS, 256/LSCTS };

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

/* Disk Parameter Block Structure */

struct dpb
{
    WORD    spt;

```

Listing H-1. (continued)

```

        BYTE    osh;
        BYTE    bin;
        BYTE    exm;
        BYTE    dpbjunk;
        WORD    dsm;
        WORD    drn;
        BYTE    a10;
        BYTE    all;
        WORD    cks;
        WORD    off;
};

/* Disk Parameter Header Structure */-
struct dph
{
        BYTE    *x1tp;
        WORD    dphscr[3];
        BYTE    *dirbufp;
struct    dpb    *dpbp;
        BYTE    *csvp;
        BYTE    *alvp;
};

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

BYTE dirbuf[LSCTSZ];

#if ! LOADER

/...../
/*      CSV's      */
/...../

BYTE    csv0[FCSVSIZE];
#if DISKB
BYTE    csv1[FCSVSIZE];
#endif
BYTE    csv2[HCSVSIZE];
#if DISKD
BYTE    csv3[HCSVSIZE];
#endif

#if MEMDSK
BYTE    csv4[16];
#endif

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

```

Listing H-1. (continued)

```

BYTE    alv0[FALVSIZE];
#if DISKB
BYTE    alv1[FALVSIZE];
#endif
BYTE    alv2[HALVSIZEC];
#if DISKD
BYTE    alv3[HALVSIZEZD];
#endif

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

#endif

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

/*****      spt, bsh, blm, exm, jnk,   dsm,   drn, */
/*      al0, all, cks, off */

#if MEMDSK
struct dpb dpb3 =          /* memory disk */
{ 32, 4, 15, 0, 0, 191, 63,
  0, 0, 0, 0 };
#endif

struct dpb dpb96 =          /* 96 tpi floppy disk - BLS = 2048 */
{ 32, 4, 15, 0, 0, FDSM96, FDRM,
  0xc0, 0, FCSVSIZE, 2 };
struct dpb dpb48 =          /* 48 tpi floppy disk - BLS = 2048 */
{ 32, 4, 15, 1, 0, FDSM48, FDRM,
  0xc0, 0, FCSVSIZE, 2 };

struct dpb dpbwdc =          /* winchester disk C: - BLS = 4096 */
{ 32, 5, 31, 1, 0, HDSMC, HDRMC,
  0xff, 0, HCSVSIZEC, 4 };

#if DISKD
struct dpb dpbwdd =          /* winchester disk D: - BLS = 4096 */
{ 32, 5, 31, 1, 0, HDSMD, HDRMD,
  0xff, 0, HCSVSIZEZD, 4 };
#endif

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

/*      No translation for 5-1/4" floppy disks */

/*****

```

Listing H-1. (continued)

```

/* Disk Parameter Headers */
/*
/* Four disks are defined : dsk a: diskno=0, (Motorola's #fd02)
/*                               : dsk b: diskno=1, (Motorola's #fd03)
/*                               : dsk c: diskno=2, (Motorola's #hd00)
/*                               : dsk d: diskno=3, (Motorola's #hd01)
/*
/*****

#if ! LOADER

/* Disk Parameter Headers */
struct dph dphtab[] =
{
    { NULL, 0, 0, 0, dirbuf, &dpb96, csv0, alv0}, /* dsk a */
#if DISKB
    { NULL, 0, 0, 0, dirbuf, &dpb96, csv1, alv1}, /* dsk b */
#else
    { NULL, 0, 0, 0, dirbuf, &dpb96, NULL, NULL}, /* dsk b */
#endif
    { NULL, 0, 0, 0, dirbuf, &dpbwdc, csv2, alv2}, /* dsk c */
#if DISKD
    { NULL, 0, 0, 0, dirbuf, &dpbwdd, csv3, alv3}, /* dsk d */
#else
    { NULL, 0, 0, 0, dirbuf, NULL, NULL, NULL}, /* dsk d */
#endif
}

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

#else

struct dph dphtab[4] =
{
    { NULL, 0, 0, 0, dirbuf, &dpb96, NULL, NULL}, /* dsk a */
    { NULL, 0, 0, 0, dirbuf, &dpb96, NULL, NULL}, /* dsk b */
    { NULL, 0, 0, 0, dirbuf, &dpbwdc, NULL, NULL}, /* dsk c */
    { NULL, 0, 0, 0, dirbuf, &dpbwdd, NULL, NULL}, /* dsk d */
}
#endif
/*****
/*      Currently Selected Disk Stuff
/*****

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

/* 48/96 tpi flags for floppies: 0x80-> 96 tpi, 0-> 48 tpi */
WORD tkflag[NUMDSKS];

```

Listing H-1. (continued)

```

/* last psn referenced on each drive - used to speed up determination */
/* of floppy diskette type when logging a new diskette */
LONG   lstpsn[NUMDSKS];

/* flag indicating configuration has been done */
WORD   config = 0;

#if LOADER
/* disk that booter was loaded from - set by booter - load cpm.sys from it */
WORD   bootdisk = 2;          /* disk number is controller lun (first fd) */
#endif

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

#if LOADER

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

/* Define the track buffer structure */

struct tbstr {
    struct tbstr *nextbuf; /* form linked list for LRU */
    BYTE   buf[TKBUFSZ]; /* at least fd trk */
    WORD   disk; /* disk for this buffer */
    WORD   trk; /* track for this buffer */
    BYTE   valid; /* buffer valid flag */
    BYTE   dirty; /* true if a BIOS write has
                  /* put data in this buffer,
                  /* put the buffer hasn't been
                  /* flushed yet.
};
struct tbstr *firstbuf; /* head of linked list of track buffers */
struct tbstr *lastbuf; /* tail of ditto */

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

#else

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

BYTE bufptr[TKBUFSZ]; /* at least fd trk */
BYTE bufvalid;
WORD buftrk;

#endif

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

```

Listing H-1. (continued)

```

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

BYTE cnvdsk[NUMDSKS] = { 2, 3, 0, 1 }; /* convert CP/M dsk# to Motorola */
BYTE rcnvdsk[NUMDSKS] = { 2, 3, 0, 1 }; /* and vice versa */

/*****/
/*      Disk I/O Packets and Variables      */
/*****/

struct dio /* disk controller registers */
{
    BYTE  cmdsns; /* command/sense byte */
    BYTE  diofl1; /* fill byte */
    BYTE  intstt; /* interrupt/status byte */

    BYTE  diofl2; /* fill byte */
    BYTE  rst; /* reset */
    BYTE  diofl3; /* fill byte */
    BYTE  ntusd; /* not used */
    BYTE  diofl4; /* fill byte */
    BYTE  data; /* data */
};

struct snsstr /* sense packet */
{
    BYTE  ercode; /* error code */
    BYTE  lun; /* CP/M logical unit number */
    BYTE  status; /* status includes controller lun */
    WORD  pcylnm; /* physical cylinder number */
    BYTE  headnm; /* head number */
    BYTE  sectnm; /* sector number */
    BYTE  n; /* number sectors left to process */
    BYTE  snsbt6; /* sense packet byte 6 */
    BYTE  snsbt7; /* sense packet byte 7 */
    BYTE  snsbt8; /* sense packet byte 8 */
    BYTE  snsbt9; /* sense packet byte 9 */
};

struct snsstr sns; /* last sense packet read form disk */

/*****/
/*      Send disk command packet      */
/*****/

sndcmd(dsk, psn, n, ctl, cmd)
REG WORD dsk, n, ctl, cmd;
REG LONG psn;
{
    /* write the packet to the controller */
    /* the DSKCNTL references must NOT be reordered */

    /* update last psn referenced */

```

Listing H-1. (continued)

```

switch (cmd)
{
case DSKREAD:
case DSKWRITE:
case DSKSCAN:
case DSKTFRMT:
    lstpsn[dsk] = psn;
    break;
case DSKRECAL:
    lstpsn[dsk] = 0;
    break;
}

/* correction for reading or writing track 0 */
/* track 0 is sd, 128 bytes/sector, 16 sectors */
/* correction assumes reads and writes are done on a track basis */
if ( (dsk < 2) && ((cmd == DSKREAD) || (cmd == DSKWRITE)) )
    if ( psn == 0 )
    {
        ctl = CTL128SD;
        n = 16;
    }

DSKCNTL->cmdsns = cmd; /* command - byte 0 */
/* following line assumes psn <= 21 bits long */
DSKCNTL->cmdsns = (cnvdsk[dsk] << 5) | (psn >> 16); /* byte 1 */
DSKCNTL->cmdsns = (psn >> 8); /* byte 2 */
DSKCNTL->cmdsns = psn; /* byte 3 */
DSKCNTL->cmdsns = n; /* byte 4 */
DSKCNTL->cmdsns = ctl | tkflg[dsk]; /* byte 5 */
}

/*****
/*      Send disk configuration packet
*****/

sndcnf(dsk, mxhd, mxcl, prcmp)
REG WORD dsk, mxhd, mxcl, prcmp;
{
    WORD zero;

    zero = 0; /* so clr.b won't be generated for byte 5 */

    /* write the configuration packet to the controller */
    /* the DSKCNTL references must NOT be reordered */

    DSKCNTL->cmdsns = DSKCONFIG; /* command - byte 0 */
    DSKCNTL->cmdsns = (cnvdsk[dsk] << 5); /* byte 1 */
    /* following line assumes mxcl <= 13 bits long */
    DSKCNTL->cmdsns = (mxhd << 5) | (mxcl >> 8); /* byte 2 */
    DSKCNTL->cmdsns = mxcl; /* byte 3 */
    DSKCNTL->cmdsns = prcmp; /* byte 4 */
    DSKCNTL->cmdsns = zero; /* byte 5 */
}

```


Listing H-1. (continued)

```

/* performs automatic recalibration - set last psn to 0 */
lastpsn[dsb] = 0;
}

/*****
/*      Get disk sense
*****/

gtsns()
{
    /* read the sense block from the controller */
    /* the DSKCNTL references must NOT be reordered */

    while ( DSKCNTL->intstt & BUSY )
        ; /* wait while controller busy */

    sns.ercode = DSKCNTL->cmdsns;
    sns.status = DSKCNTL->cmdsns;
    sns.lun = rcnvdsb[(sns.status >> 5) & 0x3];

    sns.pcylnm = DSKCNTL->cmdsns;
    sns.pcylnm = (sns.pcylnm << 8) + DSKCNTL->cmdsns;
    sns.headnm = DSKCNTL->cmdsns;
    sns.sectnm = sns.headnm & 0x1f;
    sns.headnm = sns.headnm >> 5;
    sns.n = DSKCNTL->cmdsns;
    sns.snsbt6 = DSKCNTL->cmdsns;
    sns.snsbt7 = DSKCNTL->cmdsns;
    sns.snsbt8 = DSKCNTL->cmdsns;
    sns.snsbt9 = DSKCNTL->cmdsns;
}

#if NO_ASM_SUPPORT

/*****
/*      Disk read data transfer
*****/

rddat(bp)
REG BYTE *bp;
{
    /* This routine should be written in assembly language later. */

    REG WORD cnt;
    for ( cnt = DCXFER; cnt; cnt-- )
        *bp++ = DSKCNTL->data;
}

/*****
/*      Disk write data transfer
*****/

wrdat(bp)
REG BYTE *bp;

```

Listing H-1. (continued)

```

{
    /* This routine should be written in assembly language later. */
    REG WORD cnt;
    for ( cnt = DCXFER; cnt; cnt-- )
        DSKCNTL->data = *bp++;
}

#endif /* NO_ASM_SUPPORT */

/*****
/* Translate track number to physical sector number */
*****/

LONG tk2psn(dsk, trk)
REG WORD dsk, trk;
{
    REG struct dpb *pp;
    REG WORD ttrks;

    pp = dphtab[dsk].dpbp;
    if ( dsk >= 2 )
        return(trk*((pp->spt)/lperp[dsk]));
    ttrks = 80;
    if ( pp == &dpb96 ) ttrks = 160;
    if ( trk < ttrks/2 )
        return(trk*((pp->spt)/lperp[dsk])*2);
    return(((ttrks-trk)*2-1)*(pp->spt/lperp[dsk]));
}

/*****
/* Disk Read with error correction */
*****/

WORD rddsk(dsk, psn, psent, bufp)
REG WORD dsk, psent;
REG LONG psn;
REG BYTE *bufp;
{
    LONG erofst; /* offset from bufp of location to correct */

    sndcmd(dsk, psn, psent, CTL512DD, DSKREAD);
    while ( 1 )
    {
        while ( DSKCNTL->intstt & BUSY )
            if ( DSKCNTL->intstt & DRQ )
            {
                rddat(bufp);
                bufp += DCXFER;
                psent--;
                psn++;
            }
        gtsns(); /* check for error */
        if ( sns.ercode != CORRDER )
            return (sns.ercode);
    }
}

```

Listing H-1. (continued)

```

else
{
    /* correct the data - winchester only */
    erofst = (sns.snsbt6 << 8) + sns.snsbt7 - 256;
    bufp[erofst] ^= sns.snsbt8;
    bufp[erofst+1] ^= sns.snsbt9;
    if ( pscnt ) /* more to read - reissue command */
        sndcmd(dsk, psn, pscnt, CTL512DD, DSKREAD);
    else return (0); /* done - no error to report */

    /* Should probably check for consistent correctable
    error (snsbt6-9 same twice) then write corrected
    sector back to disk. Refer to Winchester Disk
    Controller User's Manual section 4.3 Disk Error
    Recovery page 4-15. */
}

}

/*****
/*      Disk Transfer      */
*****/

dskxfer(dsk, trk, bufp, cmd)
REG WORD dsk, trk, cmd;
REG BYTE *bufp;
{
    WORD rcnt; /* retry count */
    LONG psn; /* physical sector number */
    WORD pscnt; /* physical sector count */
    BYTE *bp; /* buffer pointer for retries */
    WORD error; /* error flag */

    /* set up */
    psn = tk2psn(dsk, trk);
    pscnt = (dphtab[dsk].dpbp->spt)/lperp[dsk]; /* # phys sctrs */
    bp = bufp; /* save buffer addr */
    rcnt = 10; /* retry count */

    do /* error retry loop */
    {
        /* handle command */
        switch (cmd)
        {
            case DSKREAD:
                error = rddsk(dsk, psn, pscnt, bufp);
                break;
            case DSKWRITE:
                sndcmd(dsk, psn, pscnt, CTL512DD, cmd);
                while ( DSKCNTL->intstt & BUSY )
                    if ( DSKCNTL->intstt & DRQ )
                    {
                        wrdat(buftp);
                        bufp += DCXFER;
                    }
        }
    }
}

```

Listing H-1. (continued)

```

        gtsns();
        error = sns.ercode;
        break;
    default:
        sndcmd(dsk, psn, pscnt, CTL512DD, cmd);
        gtsns();
        error = sns.ercode;
        break;
    }
    bufp = bp; /* restore buffer addr */
} while (error && --rcnt);

/* return pass/fail indication */
if (error) return(0); /* failure */
else      return(1); /* success */
}

#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

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

struct dph *slctdsk(dsk, logged)
REG BYTE dsk;
BYTE logged;
{
    REG struct dph *dphp;
    REG LONG psn;

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

#if ! LOADER
    if ( (dsk > MAXDSK)
#if ! DISKB
        || (dsk == 1)
#endif
    )
#endif
    #if ! DISKB
    #endif

```

Listing H-1. (continued)

```

|| (dsk == 3)
#endif
    )
    {
        printstr("nrBIOS ERROR -- DISK ");
        cons_out('A'+dsk);
        printstr(" NOT SUPPORTEDnr");
        return(OL);
    }
#endif

dphp = &dphtab[dsk];

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

    if ( ! (logged & 0x1) )
    {
        /* determine disk type and size */
        switch ( dsk )
        {
            case 0:
            case 1:
                /* floppy disk */
                setinvld(dsk);

                /* assume 96 tpi */
                if ( tkflg[dsk] == TPI48 )
                {
                    lstpsn[dsk] *= 2; /* correct to 96 */
                    tkflg[dsk] = TPI96;
                }

                /* lstpsn assumes 16 sct/cyl */
                psn = lstpsn[dsk] / 16;
                if ( psn == 0 ) psn = 1; /* skip track 0 */
                psn = psn * 16; /* sector to test */

                /* scan a sector at an odd cyl to check */
                /* track density */
                andcmd(dsk, psn, 1, CTL512DD, DSKSCAN);
                gtsns();
                switch (sns.rcode)
                {
                    case NOERR:
                        dphp->dpbp = &dpb96;
                        break;
                    case IDNTFND:
                        dphp->dpbp = &dpb48;
                        tkflg[dsk] = TPI48;
                        break;
                }
            }
        }
    }
#endif

```

Listing H-1. (continued)

```

                                default: /* other error */
                                    dphp = NULL;
                                    break;
                                }
                                break;
case 2:
case 3:
                                /* hard disk */
                                break;
default:
                                printstr("nrBIOS ERROR -- DISK ");
                                cons_out('A'+dsk);
                                printstr(" NOT SUPORTEDnr");
                                return(NULL);
#endif
    }
    return(dphp);
}

/*****
/*      Home Disk
*****/

homedsk()
{
    settirk = 0;
    snocmd(setdisk, (LONG)0, 0, CTL512DD, DSKRECAL);
    while ( DSKCNTL->intstt & BUSY ) /* wait while controller busy */
        ; /* assume no errors */
}

/*****
/*      Disk Initialization
*****/

initvdsks()
{
    REG WORD i;

    if ( config ) return; /* only init once */

    /* turn off controller interrupts */
    DSKCNTL->intstt = 0;

    /* set up initial disk assumptions */
    for ( i = 0; i < NUMDSKS; i++ )
    {
        tkflg[i] = TPI96;
        lstpsn[i] = 0;
    }

    /* configure controller for disks */

```

Listing H-1. (continued)

```

        sndcnf(0, 1, 79, 40);          /* a: fd, 96 tpi, ds */
        while ( DSKCNTL->intstt & BUSY ) /* wait while controller busy */
            ;                          /* assume no errors */
#if DISKB
        sndcnf(1, 1, 79, 40);          /* b: fd, 96 tpi, ds */
        while ( DSKCNTL->intstt & BUSY ) /* wait while controller busy */
            ;                          /* assume no errors */
#endif
        sndcnf(2, CMHD, CMCYL, 255);    /* c: wd */
        while ( DSKCNTL->intstt & BUSY ) /* wait while controller busy */
            ;                          /* assume no errors */
#if DISKD
        sndcnf(3, DMHD, DMCYL, 255);    /* d: wd */
        while ( DSKCNTL->intstt & BUSY ) /* wait while controller busy */
            ;                          /* assume no errors */
#endif
        config = 1;                    /* set configuration flag */
    }

#if ! LOADER
/*****
/*      Write one disk buffer
*****/

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

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

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

    return(ok);
}

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

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

    ok = 1;
    tbp = firstbuf;
    while (tbp)
    {

```

Listing H-1. (continued)

```

        if ( ! flushl(tbp) ) ok = 0;
        tbp = tbp->nextbuf;
    }
    return(ok);
}

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

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

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

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

    tbp->valid = ok;
    tbp->dirty = 0;

    tbp->trk = settrk;
    tbp->dsk = setdsk;

    return(ok);
}

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

struct tbstr *gettrk()
{
    REG struct tbstr *tbp;
    REG struct tbstr *ltbp;
    REG struct tbstr *mtbp;

    /* Does not check for disk on-line. Doing so causes floppy */
    /* disk I/O to be extremely slow. We will catch the disk */
    /* problem the next time we try to read or write the disk. */

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

    tbp = firstbuf;
    ltbp = 0;
    mtbp = 0;

    while (tbp)

```


Listing H-1. (continued)

```

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

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

if (mtbp) mtbp->nextbuf = 0;    /* detach lru buffer */
ltpb->nextbuf = firstbuf;

firstbuf = ltpb;
if (flushl(ltpb) && fill(ltpb)) mtbp = ltpb;    /* success */
else                               mtbp = 0L;    /* failure */
return (mtbp);
}

```

```

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

```

```

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

    #if MEMDSK
    if(setdsk != MEMDSK)
    {
        #endif
        tbp = gettrk();          /* locate track buffer with sector */
        if ( ! tbp ) return(l); /* failure */
        /* locate sector in buffer and copy contents to user area */

```

Listing H-1. (continued)

```

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

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

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

    /* locate track buffer containing sector to be written */
    if (MEMDSK
        if (setdsk != MEMDSK)
    )
    {
        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 */
        MEMDSK
    }
    else
    {
        p = memdsk + (((LONG)(settrk) << 12L) + ((LONG)setsec << 7L));
        q = setdma;
        i = 128;
        do { *p++ = *q++; i -= 1; } while (i); /* this generates good code */
        return(0);
    }
}

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 ( flush(tbp) ) return(0); else return(1); }

```

Listing H-1. (continued)

```

        else return(0);
    }

#endif

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

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

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

#endif

/*.....*/
/*      BIOS Sector Translate Function      */
/*.....*/

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

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

/* exception vector base address */
LONG      *vbase = ((LONG *)0);

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

```

Listing H-1. (continued)

```

        vloc = &vbase[vnum];
        oldval = *vloc;

/* TENBUG used the illegal instruction, trace, TRAP #15 and user
   vector 78 (Abort button on console) so don't allow these to be
   clobbered while debugging. Also we use TRAP #15 to do screen
   output. */

    if (
#ifdef DEBUG
        (vnum != 4) && /* illegal instruction */
        (vnum != 9) && /* trace vector */
    #endif
        (vnum != 47) && /* TRAP #15 vector */
        (vnum != 78)) /* user trap (trap # 78) */

        *vloc = vval;

    return(oldval);
}

#ifdef LOADER
/*****
/*
/* This function is included as an undocumented,
/* unsupported method for VME/10 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;
{
    if ( ! slctdsk( (BYTE)dsk, (BYTE) 1 ) ) return(0);

#ifdef MEMDSK
    if (setdsk == MEMDSK) return(1);
#endif

    tkflg[dsk] = TPI96;
    sndcmd(dsk, (LONG)0, 0, CTL512DD, DSKDFRMT);
    gtsns();
    if ( sns.ercode ) return(0);
    sndcmd(dsk, (LONG)0, 0, CTL128SD, DSKTFRMT);
    gtsns();
    if ( sns.ercode ) return(0);
    return(1);
}

```

Listing H-1. (continued)

```

#endif

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

biosinit()
{
#if ! LOADER
    c_init();
    m400init();

#if    MVME410
    m410_init();
#endif

#endif

    initdsk();
}

initdsk()
{
    REG WORD i;

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

    initvdsks();
}

#if    ! LOADER
/*****
/*      Driver for MVME400 Dual RS-232C Serial Port Module
*****/

*      WARNING:  DO NOT OPTIMIZE THIS DRIVER !!!!!!!
*      It contains redundant stores to the serial controller
*      that are necessary for proper operation.

```

Listing H-1. (continued)

```

*/

/* 7201 control register 0 operations */

#define SELREG1      1
#define SELREG2      2
#define SELREG3      3
#define SELREG4      4
#define SELREG5      5
#define SELREG6      6
#define SELREG7      7
#define ABRTSDLC      8
#define REXSTINT      0x10
#define CHANRST       0x18
#define INTNXTRC      0x20
#define RSTTXINT      0x28
#define ERRRST        0x30
#define EOINT         0x38
#define RSTRXCRC      0x40
#define RSTTXCRC      0x80
#define RSTTXUR       0xC0

/* 7201 control register 1 operations */

#define EXINTEN       1
#define TXINTEN       2

#define STATAFV       4
#define RXINTDS       0
#define RXINT1        8
#define RXINTALP      0x10
#define RXINTANP      0x18
#define WAITRXTX      0x20
#define WAITEN        0x80
#define INTDSMSK      0xE4

/* 7201 control register 2A operations (2B is int vector) */

#define BOTHINT       0
#define ADMABINT      1
#define BOTHDMA       2
#define PRIAGB        0
#define PRIrgT        4
#define M8085M        0
#define M8085S        8
#define M8086         0x10
#define NONVEC        0
#define INTVEC        0x20
#define RTSBP10       0
#define SYNCBP10      0x80

/* 7201 control register 3 operations */

#define RXENABLE      1

```

Listing H-1. (continued)

```

#define SYNLDIR          2
#define ADIRSRCH         4
#define RXCRCEN          8
#define ENTHUNT          0x10
#define AUTOENA          0x20
#define RX5BITS          0
#define RX7BITS          0x40
#define RX6BITS          0x80
#define RX8BITS          0xC0
#define RXS2MSK          0x3F

/* 7201 control register 4 operations */

#define PARENAB          1
#define EVENPAR          2
#define ODDPAR           0
#define SYNCMODE         0
#define SBIT1            4
#define SBIT1P5          8
#define SBIT2            0xC
#define SBITMSK          0xF3
#define SYN8BIT          0
#define SYN16BIT         0x10
#define SDLCMODE         0x20
#define EXTSYNC          0x30
#define CLKX1            0
#define CLKX16           0x40
#define CLKX32           0x80

#define CLKX64           0xC0

/* 7201 control register 5 operations */

#define TXCRCEN          1
#define RTS              2
#define CRC16            4
#define CRCCCITT         0
#define TXENABLE         8
#define SENDBRK          0x10
#define TX5BITS          0
#define TX7BITS          0x20
#define TX6BITS          0x40
#define TX8BITS          0x60
#define TXS2MSK          0x9F
#define DTR              0x80

/* 7201 control register 6 = sync bits 0-7 */
/* 7201 control register 7 = sync bits 8-15 */

/* 7201 status register 0 */

#define RXCHAR           1          /*Receive character available */
#define INTPNDNG         2          /*Transmit register is empty */
#define TXBUFEMP         4
#define DCD              8

```

Listing H-1. (continued)

```

#define SYNCHUNT      0x10
#define CTS           0x20
#define TXUNDER       0x40
#define BRKABRT       0x80

/* 7201 status register 1 */

#define PARERR        0x10
#define RXOVRUN       0x20
#define CRCPRMER      0x40
#define EOFFRAME      0x80

/* 7201 status register 2 = int vector */

#define FALSE         0
#define TRUE          1

/* Macro to WRITE information to the 7201's control registers */
#define WRITE(y,x,z,w)  *y = x;                *y = aux_state[z].w

#define M400_1         0 /* table indices for the 2 auxillary ports */
#define M400_2         1
#define AUX            M400_1 /* Auxiliary Serial Device */
#define LST            M400_2 /* List Device (line printer) */

/* baud rate control values
0 for 0 ,      0 for 50 ,      1 for 75 ,      2 for 110 ,
3 for 134 ,    4 for 150 ,    4 for 200 ,    5 for 300 ,
6 for 600 ,    7 for 1200 ,    8 for 1800 ,    10 for 2400 ,
12 for 4800 ,  14 for 9600 ,    15 for EXTA ,    15 for EXTB
EXTA & EXTB = 19.2K */

typedef struct
{
    BYTE    cr1;
    BYTE    cr2;
    BYTE    cr3;
    BYTE    cr4;
    BYTE    cr5;
    BYTE    baud;
    BYTE    char_size;

    /*baud rate for the port */
    /*size of character: 0x20 = 7
                        0x40 = 6
                        0x60 = 8 */
} mstate;

/*following is the actual variable which holds the state
of the MVME400 board. It is referenced in the code below
using the pointer variable "aux state" defined above.
The array dimension expression forces even byte alignment
at the end of each element of mstate(of which there are
2). All of this foolishness is necessary because the
C compiler will not initialize char variables. */

```


Listing B-1. (continued)

```

BYTE      init_state[] = {
(0),                      /* A cr1 */
(BOTHINT|PRIRGT|M8086|RTSBP10), /* A cr2 */
(AUTOENA|RX8BITS|RXENABLE), /* A cr3 */
(SBIT1|CLKX16),           /* A cr4 */
(TXENABLE|TX8BITS|RTS|DTR), /* A cr5 */
(14),                     /* A baud rate = 9600 */
(TX8BITS),                /* A character size */
(0),                       /* Dummy fill char */
(STATAPV),                /* B cr1 */
(0),                       /* B cr2 */
(AUTOENA|RX8BITS|RXENABLE), /* B cr3 */
(SBIT1|CLKX16),           /* B cr4 */
(TXENABLE|TX8BITS|RTS|DTR), /* B cr5 */
(7),                       /* A baud rate = 1200 */
(TX8BITS),                /* A character size */
0                           /* Dummy fill char */
};

/* State of the MVME400 board; one set of information
   for each of the ports on the board */

mstate     *aux_state = (mstate *)init_state;

/*
 * Structure of MVME400 hardware registers.
 * Assumes an odd starting address.
 */
typedef struct
{
    BYTE      m4_piaad;      /* pia a data */
    BYTE      m4_fill10;     /* fill */
    BYTE      m4_piaac;      /* pia a control */
    BYTE      m4_fill11;     /* fill */
    BYTE      m4_piabd;      /* pia b data */
    BYTE      m4_fill12;     /* fill */
    BYTE      m4_piabc;      /* pia b control */
    BYTE      m4_fill13;     /* fill */
    BYTE      m4_7201d[3];    /* 7201 a data */
    /* BYTE      m4_fill14;     fill */
    /* BYTE      m4_72bd;       7201 b data */
    BYTE      m4_fill15;     /* fill */
    /* BYTE      m4_7201c[3];    7201 a control */
    /* BYTE      m4_fill16;     fill */
    /* BYTE      m4_72bc;       7201 b control */
} m4_map;

/* MVME400 BASE ADDRESS */

#define M400_ADDR      ((m4_map *)0xf1c1c1)

/* Macro to dereference VME/10 control register #6 */

#define IOCHANRG      (*(BYTE *)0xf19f11)

```

Listing H-1. (continued)

```

#define CHN3EN          0x20

/*****
** Initialize both ports on the mvme400 card
*****/

m400init()
{
    REG_BYTE          *caddr, *caddrb;
    REG m4_map        *addr;

    IOCHANRG &= ~CHN3EN;          /* disable I/O channel 3 interrupts */

    aux_state = (mstate *)init_state; /* aux_state points at initialized array */

    if ( no_device((LONG)M400_ADDR) ) return;

    addr = M400_ADDR;
    caddr = &addr->m4_7201c[0]; /* address of A side control register */
    caddrb = &addr->m4_7201c[2]; /* address of B side control register */
    addr->m4_piaac = 0;          /* set up pia data direction regs */
    addr->m4_piaad = 0x18;
    addr->m4_piaac = 4;
    addr->m4_piabc = 0;
    addr->m4_piabd = 0xff;
    addr->m4_piabc = 4;
    addr->m4_piaad = 0;          /* fail led off */

                                /* set the baud rate for both ports */
    addr->m4_piabd = (aux_state[M400_1].baud << 4) /* A port */
                    | aux_state[M400_2].baud; /* B port */

    *caddr = CHANRST;          /* reset channels */
    linit1:                    /* label to force generation of next line */
    *caddr = CHANRST;          /* write twice so */
    *caddrb = CHANRST;         /* it is sure to be */
    linit2:
    *caddrb = CHANRST;         /* written to cr0 */
                                /* Initialize the control registers NEC 7201
                                Communications controller chip */

    WRITE(caddr, SELREG2, M400_1, cr2);
    WRITE(caddrb, SELREG2, M400_2, cr2);
    WRITE(caddr, SELREG4, M400_1, cr4);
    WRITE(caddr, SELREG3, M400_1, cr3);
    WRITE(caddr, SELREG5, M400_1, cr5);
    WRITE(caddr, SELREG1|RSTT_XINT, M400_1, cr1);
    WRITE(caddrb, SELREG4, M400_2, cr4);
    WRITE(caddrb, SELREG3, M400_2, cr3);
    WRITE(caddrb, SELREG5, M400_2, cr5);
    WRITE(caddrb, SELREG1|RSTT_XINT, M400_2, cr1);
}
/*****

```

Listing H-1. (continued)

```

** Read a character from one of the asynchronous ports on the mvme400
...../

BYTE m400_in(port)

REG WORD      port;
{
    m4_map      *addr;

    while (!(((M400_ADDR)->m4_7201c[port*2]) & RXCHAR)); /*wait for char. ready*/
    return((M400_ADDR)->m4_7201d[port*2]);                /* get the char */
}

/*****
** Write a character to one of the asynchronous ports on the mvme400
...../

VOID m400_out(port, ch)

REG WORD      port;
REG BYTE      ch;
{
    /*wait till ready to send */
    while (!(((M400_ADDR)->m4_7201c[port*2]) & TXBUFEMP));
    (M400_ADDR)->m4_7201d[port*2] = ch; /* output the character */
    return;
}
#endif

#if ! LOADER

#if MVME410
/*****
 *      m410 driver for the VME/10.
 *
 *      ROUTINES:
 *
 *      m410_init()      - initialize mvme410 card.
 *      m410_stat()      - check the status of the mvme410 card.
 *      m410_out()       - put a character out on the mvme410 card.
 */

/*
 *      base address of the mvme410
 *      control block.
 */

#define M410BASE      ((struct m410_ctl_blk *) 0xF1C1E0)

struct m410_ctl_blk
{

```

Listing H-1. (continued)

```

char    pad0;
char    data_a;
char    pad1;
char    ctl_a;
char    pad2;
char    data_b;
char    pad3;
char    ctl_b;
};

/*
 *   define some simple aliases for structure members
 *   above.
 */
#define stat_a      ctl_a
#define stat_b      ctl_b
#define ddr_a       data_a      /* ddr - data direction register */
#define ddr_b       data_b
#define prdy        data_b
#define strobe       ctl_a
#define acknowledge  ctl_a
#define ready        data_b

/*
 *   some control values - bit positions.
 */
#define PAPEROUT     0x2
#define SELECT       0x1

/*
 *   data strobe low and high following write.
 */
#define STROBELOW     0x34
#define STROBEHIGH    0x3C

/*
 *   Data acknowledge is set after printer accepts
 *   character.
 */
#define DATA_ACK     0x80

/*
 *   m410_stat return values.
 */
#define NOTREADY      0x00
#define READY         0xFF

/*

```

Listing H-1. (continued)

```

*      Initialization values.
*/
#define INITDDR      0x38
#define INITCTL      0x3C
#define OUTPUT       0xFF
#define INPUT        0x00

/*
*      m410_init()
*
*      initialize the mvme 410 board for use as parallel
*      printer interface.
*/
m410_init()
{
    register struct m410_ctl_blk  *m410_base;

    m410_base = M410BASE;

    /*
     *      test if a 410 card is available before performing
     *      initialization.
     */
    if ( no_device((LONG)M410BASE) ) return;

    /*
     *      initialize a side of controller to output.
     */
    m410_base->ctl_a = INITDDR;
    m410_base->ddr_a = OUTPUT;
    m410_base->ctl_a = INITCTL;

    /*
     *      initialize b side of controller to input.
     */
    m410_base->ctl_b = INITDDR;
    m410_base->ddr_b = INPUT;
    m410_base->ctl_b = INITCTL;
}

/*
*      m410_stat()
*
*      determine status of mvme 410 board.  Is the device ready to write to.
*
*      returns:
*
*          $FF - if ready

```

Listing H-1. (continued)

```

*           $00 - if not ready
*/

m410_stat()
{
    register struct m410_ctl_blk    *m410_base;
    register short    status;

    m410_base = M410BASE;

    status = m410_base->prdy & (PAPEROUT|SELECT);
    if( status != SELECT )
        return( NOTREADY );

    return( READY );
}

/*
*   m410_out()
*
*   wait until device is ready,
*   write a single character 'c' to the 410 card,
*   wait for acknowledgement.
*/

m410_out(c)
register char    c;
{
    register struct m410_ctl_blk    *m410_base;

    while( m410_stat() == NOTREADY )
        ;

    m410_base = M410BASE;

    m410_base->data_a = c;           /* write character out */
    c = m410_base->data_a;          /* dummy read to clear acknowledge */

    m410_base->strobe = STROBELOW;
    m410_base->strobe = STROBEHIGH;

    /*
    *   wait for data acknowledge
    */

    while( (m410_base->acknowledge & DATA_ACK) == 0 )
        ;

    return;
}

/*****
endif /* MVME410 */

```

Listing H-1. (continued)

```

#endif /* ! LOADER */

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

LONG cbios(d0, d1, d2)
REG WORD    d0;
REG LONG    d1, d2;
{
    switch(d0)
    {
        case 0: biosinit();                /* INIT      */
        break;

#if ! LOADER
        case 1: flush();                    /* WBOOT     */
        initdsk();
        wboot();
        /* break; */

        case 2: return(cons_stat());        /* CONST     */
        /* break; */

        case 3: return(cons_in());          /* CONIN     */
        /* break; */

#endif
        case 4: cons_out((char)d1);         /* CONOUT    */
        break;

#if ! LOADER
        case 5: m410_out((char)d1);         /* LIST      */
        break;

        case 5: m400_out(M400_2, (char)d1); /* LIST      */
        break;

#endif
        case 6: m400_out(M400_1, (char)d1); /* PUNCH     */
        break;

        case 7: return(m400_in(M400_1));    /* READER    */
        /* break; */

#endif
        case 8: homedsk();                  /* HOME      */
        break;
    }
}

```

Listing H-1. (continued)

```

case 9:
#if LOADER
    dl = rcnvdsk[bootdsk]; /* disk booter was loaded from */
    return((LONG)slctdsk((char)dl, (char)d2)); /* SELDSK */
    /* break; */

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

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

case 12: setdma = (BYTE *)dl; /* SETDMA */
    break;

case 13: return(read()); /* READ */
    /* break; */

#if ! LOADER
case 14: return(write((char)dl)); /* WRITE */
    /* break; */

case 15:
    return(m410_stat());
#else
    /* auxiliary input status */
    break;
#endif /* MVME410 */
#endif /* ! LOADER */

case 16: return(sectran((int)dl, d2)); /* SECTRAN */
    /* break; */

#if ! LOADER
case 18: return((LONG)&mementab); /* GMRTA */
    /* break; */

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

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

case 21: if (flush()) return(OL); /* FLUSH */
    else return(0xffffL);
    /* break; */

#endif
case 22: return(setxvect((int)dl,d2)); /* SETXVECT */
    /* break; */

#if ! LOADER
/*****
/* This function is not part of a standard BIOS. */
/* It is included only for convenience, and will */
/* not be supported in any way, nor will it */
*****/

```


Listing H-1. (continued)

```
/*      necessarily be included in future versions of      */
/*      CP/M-68K                                              */
/*****
case 63: return( ! format((int)d1) ); /* Disk Formatter */
/* break; */
#endif

        default: return(0L);
        break;
) /* end switch */

] /* END OF BIOS */

/* End of C Bios */
```

Listing H-2. VME/10 VT52.C File

```

/*      @(#)vt52.c      3.6      */
/*      Copyright 1984, Motorola Inc.  */
/*****

```

Function Name: cons_out

Description: Main entry point for the VME/10 screen driver. This VME/10 screen driver is an implementation of the VT52. It supports the following functions:

carriage return - cursor is placed at column 0 on the current line.
How Recognized : <cr> = 0D(hex)

line feed - cursor is placed at the same column on the next line.
How Recognized : <lf> = 0A(hex)

backspace - cursor will move 1 column to the left on the screen. If the cursor is already at column zero then it will be placed at the last column of the previous line. If the cursor is currently on the top line of the display at column zero; it will remain where it is.
How Recognized : <bs> = 08(hex)

tab - cursor moves to next tab stop on current line. Cursor moves to end of line if there are no more tab stop on the line. Tab stops are set every eight columns.
How Recognized : <tab> = 09(hex)

clear to end of screen - blanks are written to the cursor and all positions on the screen following the cursor. The cursor will remain in its current position.
How Recognized : <esc>J = 1B 4A (hex)

clear to end of line - blanks are written to the cursor and all positions on the line following the cursor. The cursor will remain in its current position.
How Recognized : <esc>K = 1B 4B (hex)

home cursor - cursor will be placed on the first line at the first column on the screen.
How Recognized : <esc>H = 1B 48 (hex)

move cursor right - cursor will be moved one character position right without affecting the character at that position. The cursor will not move past the right margin.
How Recognized : <esc>C = 1B 43 (hex)

move cursor up - cursor will be moved one up one line. If the

Listing H-2. (continued)

cursor is on the top line then it will stay there.
How Recognized : <esc>A = 1B 41 (hex)

reverse line feed - move the cursor up one line. If the cursor was already on the top line the screen will be scrolled down one line. The line at the top of the screen will be cleared. The cursor will remain in the same column.
How Recognized : <esc>I = 1B 49 (hex)

position cursor - cursor will be moved to the specified line and column. If either value is beyond the edge of the screen then the maximum value for that argument will be used. The line position is sent before the column position. Both values are added to the ASCII space character value so that the values received are printable characters.
How Recognized : <esc>Y<line> ' ' <col> ' ' = 1B 59 (hex)

place character - will place any printable character (i.e. 20 - 7E (hex)) at the present cursor position on the screen.
How Recognized : any printable character (which is not part of a control function sequence).

Inputs: char - a 7-bit ascii character.

Outputs: None.

External variables: None

```

*****/

#define MAXCOLS      79      /*last column on physical screen */
#define MAXLINES     23      /*last line on physical screen */
#define ESC          0x1b    /* <esc> character */
#define TAB          0x09    /* <tab> character */
#define CR           0x0d    /* carriage return */
#define LF           0x0a    /* line feed */
#define BS           0x08    /* back space */
#define INITATTR     0x4400  /*default attributes in character RAM*/
#define FALSE        0
#define TRUE         1
#define UP            0      /*indicates scroll screen up */
#define DOWN         1      /*indicates scroll screen down */

/* states in which the screen driver can exist whenever a character
   is received. Basically indicates possible previous sequences of
   characters which have been received. */

#define any_char      0      /* ready to receive any character */
#define esc           1      /* last character was an <esc> */
#define curs_line     2      /* next char is encoded cursor line pos */
#define curs_col      3      /* next char is encoded cursor column pos */

/* Logical Screen Control Block - contains information on each logical

```

Listing H-2. (continued)

```

screen (i.e., window) which is available on the physical screen of the
VME/10 (the first version only supports 1 logical screen and it is
mapped to all but the last line of the physical screen. */

typedef struct
{
    short  cur_line;      /*current line cursor is on      */
    short  cur_col;       /*current column cursor is at    */
    short  cur_cursor;    /*current offset of cursor on the*/
                        /*screen; first position = 0;    */
    short  base_line;     /*first line of the logical screen*/
    short  base_col;      /*first column of the logical screen*/
    short  max_line;      /*last line on screen relative to base*/
    short  max_col;       /*last col on screen relative to base*/
    char   tabstop[MAXCOLS+1]; /*the element corresponding to each
                        /*column on the screen contains the
                        /*column where the next tabstop is
                        /*located. */
} lscb;

/* chr_map describes the word in the character and attribute RAM which
corresponds to a character on the screen. Note that the chr field
in the declaration is 8 bits. This assumes that software attribute
bit 2 is not being used. */

typedef struct
{
    char   attr;          /*attributes of character:
                        /*bit 7 - software attribute 1
                        /*bit 6 - don't display character
                        /*bit 5 - blink character
                        /*bit 4 - underline character
                        /*bit 3 - inverse video
                        /*bit 2 - color or intensity bit 1
                        /*bit 1 - color or intensity bit 2
                        /*bit 0 - color or intensity bit 3
    char   chr;           /* 7 - bit ascii character */
} chr_map;

/* attribute and character RAM area
maps the VME/10 screen for regular character I/O */

#define SCRN_ADDR ((short *)0xf17000) /*address of character RAM*/

/* information necessary to access the CRTC (the screen controller) */

typedef struct /*map of the registers for the CRTC */
{
    char   addr_reg;      /*address register selects which
                        /*register to write in register file */
    char   fill12, reg_file; /*data to registers goes here */
} crt_map;

#define CRTC_ADDR ((crt_map *)0xf1a021) /*address of the CRTC */
#define CURHIGH 14

```

Listing H-2. (continued)

```

#define CURLOW          15

/*      VARIABLE DECLARATIONS      */

static int      state;          /* current state of screen driver */
static lscb     scrn_tbl = {    /* Logical Screen Control Table */
    MAXLINES,    /* current line */
    0,           /* current column */
    0,           /* current cursor */
    0,           /* base line */
    0,           /* base column */
    MAXLINES,    /* max line on screen relative to base */
    MAXCOLS,     /* max col for screen relative to base */
    /* tab stops */
    8, 8, 8, 8, 8, 8, 8, 8, 16, 16, 16, 16, /* 0-11 */
    16, 16, 16, 16, 24, 24, 24, 24, 24, 24, 24, 24, /* 12-23 */
    32, 32, 32, 32, 32, 32, 32, 32, 40, 40, 40, 40, /* 24-35 */
    40, 40, 40, 40, 48, 48, 48, 48, 48, 48, 48, 48, /* 36-47 */
    56, 56, 56, 56, 56, 56, 56, 56, 64, 64, 64, 64, /* 48-59 */
    64, 64, 64, 64, 72, 72, 72, 72, 72, 72, 72, 72, /* 60-71 */
    79, 79, 79, 79, 79, 79, 79, 79          /* 72-79 */
};

static short     mcurslin;      /* move cursor - line position */
static short     char_mask = [INITATTR]; /* attribute & character mask */
char             scrvr[] = "screen driver 3.6";

cons_out(chr)
char chr;
{
    chr &= 0x7f; /*7-bit ascii character */
    /*strip the high order bit */

    switch ((int)state) /*perform action based on the current*/
    { /*state of the screen handler. */
        case any_char: /*waiting to accept any character (i.e.
                        are not processing control sequence*/
            switch (chr)
            {
                case ESC: /*received an <esc> character */
                    state = esc;
                    break;

                case TAB: /*received a <tab> character */
                    scrn_tbl.cur_col = scrn_tbl.tabstop[scrn_tbl.cur_col];
                    pos_cursor(FALSE);
                    break;

                case CR: /*received a carriage return */
                    scrn_tbl.cur_col = 0;
                    pos_cursor(FALSE); /*position cursor per scrn_tbl*/
                    break;

                case LF: /*received a line feed */
                    /*at end of screen?*/
                    if (scrn_tbl.cur_line == scrn_tbl.max_line)
                        scroll(UP, MAXCOLS); /* scroll up 1 line */
            }
        }
    }
}

```

Listing B-2. (continued)

```

else
    scrn_tbl.cur_line++;    /* cursor down 1 line */

case BS:    /*received a back space character */
    /*at beginning of a line? */
    if (scrn_tbl.cur_col == 0)
    {
        if (scrn_tbl.cur_line != 0)
        {
            /* not at first line of display */
            /*place cursor at last column of
            previous line */
            scrn_tbl.cur_line--;
            scrn_tbl.cur_col = scrn_tbl.max_col;
        }
        else
            break; /* at first line of display */
    }
    else /* not at beginning of a line */
        scrn_tbl.cur_col--;
    pos_cursor(FALSE); /*position cursor per scrn_tbl */
    break;

default:
{
    /* character and attributes for RAM */
    register short chr_attr;

    if (chr >= ' ') /* printable character? */
    {
        /* chr in 16-bit template */
        chr_attr = chdr_mask | (short)chr;
        /*place character on the screen */
        (SCRN_ADDR[scrn_tbl.base_line+scrn_tbl.cur_line]
        pos_cursor(TRUE); /* put cursor at next pos */
    }
}
} /* end of switch(chr) */
break;

case esc:    /* process a control function sequence */
    state = any_char; /* except position cursor is special */
    switch (chr)
    {
        case 'A':    /* cursor up a line */
            if (scrn_tbl.cur_line != scrn_tbl.base_line)
                scrn_tbl.cur_line--;
            pos_cursor(FALSE);
            break;

        case 'C':    /* nondestructive space */
            pos_cursor(TRUE);
            break;
    }

```

Listing H-2. (continued)

```

case 'H':          /* home cursor */
    scrn_tbl.cur_col = scrn_tbl.base_col;
    scrn_tbl.cur_line = scrn_tbl.base_line;
    pos_cursor(FALSE); /* position the cursor */
    break;

case 'I':          /* reverse line feed */
    if ( scrn_tbl.cur_line != scrn_tbl.base_line )
        scrn_tbl.cur_line--;
    else
        scroll(DOWN, MAXCOLS);
    pos_cursor(FALSE);
    break;

case 'J':          /* clear display from cursor */
    {
        int line;

        line = scrn_tbl.cur_line;
        clr_line(line++, scrn_tbl.cur_col);
        while ( line <= scrn_tbl.max_line )
            clr_line(line++, scrn_tbl.base_col);
    }
    break;

case 'K':          /* clear line from cursor */
    clr_line(scrn_tbl.cur_line, scrn_tbl.cur_col);
    break;

case 'Y':          /* position cursor */
    state = curs_line; /* next comes cursor line */
    break;
}
break;

case curs_line:
    mcurslin = chr - ' '; /* save new line position */
    if ( mcurslin > MAXLINES ) mcurslin = MAXLINES; /* limit it */
    state = curs_col; /* next comes cursor column */
    break;

case curs_col:
    scrn_tbl.cur_line = mcurslin; /* update cursor position */
    chr = chr - ' ';
    if ( chr > MAXCOLS ) chr = MAXCOLS; /* limit position */
    scrn_tbl.cur_col = chr;
    pos_cursor(FALSE); /* move it */
    state = any_char;
    break;
} /* end of switch(state) */
}

pos_cursor(incr)

```

Listing B-2. (continued)

```

short   incr;
{
    register short   curpos; /*byte offset to cursor position */

    if (incr)           /*bump cursor 1 position or does screen
                           table already have correct position?*/
    {
        /*bump cursor */
        if (scrn_tbl.cur_col != scrn_tbl.max_col) /*end of line?*/
            scrn_tbl.cur_col++; /*no, bump cursor position*/
    }

    curpos = (scrn_tbl.base_line + scrn_tbl.cur_line) * (MAXCOLS + 1)
             + (scrn_tbl.base_col + scrn_tbl.cur_col);
    CRTC_ADDR->addr_reg = CURLOW;
    CRTC_ADDR->reg_file = (char)curpos;
    CRTC_ADDR->addr_reg = CURHIGH;
    CRTC_ADDR->reg_file = (char)(curpos >> 8);
}

clr_line(line,col)      /* clear to end of line starting at col */
int   line, col;
{
    register short   chr_attr;      /* attribute & character */
    register int     j;
    register short   *p;           /* address of character */

    /*set up to move blanks to screen RAM*/
    chr_attr = char_mask | 0x0020;
    j = scrn_tbl.max_col + 1 - col; /*decremented for each char */
    /*address of 1st character*/
    p = &(SCRN_ADDR[line][col]);
    do { /*write blanks to line */
        *p++ = chr_attr;           /*write blank to a char pos*/
    } while (j--);
}

```


Listing H-2. (continued)

```

    } while (--j);
}

scroll(direction, num_chrs)
int    direction; /*direction to scroll the screen */
int    num_chrs;  /*number of characters to scroll */
{
    register int    i, j, line;
    register short  chr_attr; /*attribute & character */
    register short  *p1, *p2; /*address of character */

    if (direction == UP) /*scroll screen text up */
    {
        line = scrn_tbl.base_line; /*used to step down lines */
        i = scrn_tbl.max_line; /*decremented for each line */
        do {
            j = scrn_tbl.max_col + 1; /*decremented for each char */

            p1 = &(SCRN_ADDR[line++][scrn_tbl.base_col]);
            p2 = &(SCRN_ADDR[line][scrn_tbl.base_col]);
            do {
                /*move a line at a time */
                *p1++ = *p2++; /*move character up a line */
            } while (--j);
        } while (--i);
    }
    else /*direction == DOWN*/
    {
        line = scrn_tbl.max_line; /*used to step up lines */
        i = scrn_tbl.max_line; /*decremented for each line */
        do {
            j = scrn_tbl.max_col + 1; /*decremented for each char */

            p1 = &(SCRN_ADDR[line--][scrn_tbl.base_col]);
            p2 = &(SCRN_ADDR[line][scrn_tbl.base_col]);
            do {
                /*move a line at a time */
                *p1++ = *p2++; /*move character up a line */
            } while (--j);
        } while (--i);
    }

    chr_attr = char_mask | 0x0020; /*move blanks to screen RAM*/
    /*address of 1st character */
    p1 = &(SCRN_ADDR[line][scrn_tbl.base_col]);
    j = scrn_tbl.max_col + 1; /*decremented for each char */
    do {
        *p1++ = chr_attr; /*write blank to a char pos*/
    } while (--j);
}

```

Listing H-3. VME/10 BIOSTYPS.H File

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

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

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

/*****
```

Listing H-4. VME/10 NORMBIOS.H File

```
/*      @(#)normbios.h  1.5          */
#define LOADER  0
#define CTLTYPE 0
#define MEMDSK  4
#define DISKB   0
#define DISKC   5
#define DISKD   0
#define NO_ASM_SUPPORT  0

/*
 *      the preprocessor variable MVME410 controls assignment
 *      of the 1st: device in bios.c.  if the variable is defined
 *      == 1 code is included to support an mvme-410 parallel
 *      port card, if == 0 the 1st: device is assumed to be
 *      the second port of the mvme-400 serial card.
 *
 *      the initialization sequence of the mvme-410 will not prevent
 *      booting if the code is included and the card is not installed.
 *      factory default settings are assumed.
 */
#define MVME410 1
```

Listing H-5. VME/10 LOADBIOS.H File

```
/*      @(#)loadbios.h  1.3          */
#define LOADER  1
#define CTLTYPE 0
#define MEMDSK  0
#define DISKB   0
#define DISKC   5
#define DISKD   0
#define NO_ASM_SUPPORT  0
```

Listing H-6. VME/10 BIOSA.S File

```

*      @(#)bios.a.s      2.3
*      .text
*
*      Global Code addresses
*
*      .globl  _init
*      .globl  _biosinit
*      .globl  _flush
*      .globl  _wboot
*      .globl  _cbios
*      .globl  _setimask
*      .globl  _ccp
*      .globl  _cpm
*      .globl  _end
*
*      Global data addresses
*
*      .globl  _memtab
*      .globl  _dpb3
*      .globl  _memdisk
*
*      Vector Addresses
*
*      trap3: .equ    $8c
*      buserr: .equ    $8
*
*      _init: lea      entry,a0
*             move.l   a0,trap3
*
*      Auto-Size TPA
*
*      lea      _memtab,a0
*      move.w   #1,(a0)+
*      move.l   #$b00,(a0)+
*      move.l   $cpm-$b08,(a0)+
*
*      Auto-Size RAM disk
*
*      move.l   buserr,-(sp)
*      lea      _end,a0
*      add.l    #cpm,a0
*      move.l   a0,_memdisk
*      move.l   #quit,buserr
*
*      loop:   tst.w   (a0)+
*             bra      loop
*
*      quit:   add.l   #58,a7
*             move.l   (a7)+,buserr
*             sub.l    #_end,a0
*             sub.l    #cpm,a0

```

* Lowest addr of CP/M
* Highest addr of CP/M

* memory region table
* RAM disk dpb address
* -> First memory disk location

* Trap 3 vector
* Bus error vector

* a0 -> Memory region table
* 1 region
* TPA starts at b00
* Ends where CP/M begins

* Push bus err vector
* a0 -> Last location in CP/M
* Linker doesn't reloc this!!
* -> first location in RAM disk
* set up vector -> ourselves

* Find
* End of memory

* Clear buserr gorp
* Pop buserr vector
* a0 = # bytes in RAM disk
* Relocation bug

Listing H-6. (continued)

```

        move.l  a0,d0          *      Into D reg for shift
        move.l  #11,d1         *      Load shift count
        lsr.l   d1,d0          *      Divide by 2048
        move.w  d0,_dpb3+6     *      Load DRM field of dpb

        move     #$2000,sr
        jsr      _biosinit
        clr.l   d0
        rts

*
_wboot:  clr.l   d0
        jmp      _ccp
*
entry:   move.l  d2,-(a7)
        move.l  d1,-(a7)
        move.w  d0,-(a7)
        jsr      _cbios
        add     #10,a7
        rte

*
_setimask: move  sr,d0
        lsr     #8,d0
        and.l   #7,d0
        move    sr,d1
        ror.w   #8,d1
        and.w   #$fff8,d1
        add.w   4(a7),d1
        ror.w   #8,d1
        move    d1,sr
        rts

*
        .globl  _rddat
        .globl  _wrdat

*
* without considering wait states movep.l makes
* the faster loop. however, accounting for wait
* states makes the 68010 loop mode faster.
*
_rddat:  move.l  4(sp),a0
        move.w   #127,d0
        move.l   #$flc0d9,a1
rdlp:    move.b  (a1),(a0)+
        dbf      d0,rdlp
        rts

*
_wrdat:  move.l  4(sp),a0
        move.w   #127,d0
        move.l   #$flc0d9,a1
wrlp:    move.b  (a0)+,(a1)
        dbf      d0,wrlp
        rts

*
        .globl  _no_device

*
_no_device:

```

Listing H-6. (continued)

```
    move.l 4(sp),a0
    move.l buserr,-(sp)
    move.l #ndber,buserr
    move.b (a0),d0
    moveq.l #0,d0
    move.l (sp)+,buserr
    rts
ndber:
    moveq.l #1,d0
    add.l #58,sp
    move.l (sp)+,buserr
    rts
    .end
```

Listing H-7. VME/10 LDBIOSA.S File

```

*      @(#)ldbiosa.s    1.3      *
      .text
      .globl  _bios
      .globl  _biosinit
      .globl  _cbios
      .globl  _setimask
      .globl  _cons_out
*
*
*
_bios:  link      a6,#0
        move.l    d2,-(a7)
        move.l    d1,-(a7)
        move.w    d0,-(a7)
        jsr       _cbios
        unlk      a6
        rts
*
_setimask: move    sr,d0
            lsr     #8,d0
            and.l   #7,d0
            move    sr,d1
            ror.w   #8,d1
            and.w   #$fff8,d1
            add.w   4(a7),d1
            ror.w   #8,d1
            move    dl,sr
            rts
*
_cons_out:
        move.l    a6,-(sp)
        move.l    a5,-(sp)
        lea.l     l3(sp),a5
        lea.l     1(a5),a6
        trap      #15
        .dc.w     6
        move.l    (sp)+,a5
        move.l    (sp)+,a6
        rts
*
        .globl  _rddat
        .globl  _wrdat
*
* wihtout considering wait states movepl makes
* the faster loop. however, accounting for wait
* states makes the 68010 loop mode faster.
*
_rddat: move.l    4(a7),a0
        move.w    #127,d0
        move.l    #$flc0d9,a1

```


Listing B-7. (continued)

```
rdlp:  move.b  (a1),(a0)+
       dbf     d0,rdlp
       rts

*
*
_wrdat move.l  4(a7),a0
       move.w  #127,d0
       move.l  #$f1c0d9,a1
wrlp:  move.b  (a0)+,(a1)
       dbf     d0,wrlp
       rts

*
*       .globl  _no_device
_no_device:
       move.l  4(sp),a0
       move.l  8,-(sp)
       move.l  #ndberr,8
       move.b  (a0),d0
       moveq.l #0,d0
       move.l  (sp)+,8
       rts
ndberr:
       moveq.l #1,d0
       add.l   #58,sp
       move.l  (sp)+,8
       rts
       .end
```

Listing H-8. VME/10 BOOTER S. File

```

*          @(#)booter.s      1.6          *
*****
*      Information to go on track 0 and   *
*      start of boot of a VME/10        *
*****

first:      .text
            .dc.l      'CP/M'          * volume identifier
            .dc.l      0                * user number
            .dc.w      1                * garbage
            .dc.w      1                * length of SAT
            .dc.l      2                * secondary directory start
            .dc.l      0                * primary directory PSN list start
            .dc.l      8                * start of boot loader
            .dc.w      32               * length of boot loader
            .dc.l      $0              * boot execution address
            .dc.l      $1300           * boot load address
            .dc.b      '9/30'          * generation date
            .dc.b      'CP/M-68K of 9/30/82 ' * volume descriptor
            .dc.b      '0020'          * version/revision
            .dc.w      $0a484          * checksum (god help us)
            .dc.l      $0fle2d3c       * diagnostic test pattern
            .dc.l      $4b5a6978
            .dc.l      $8796a5b4
            .dc.l      $c3d2elf0
            .dc.l      $0fle2d3c       * diagnostic test pattern
            .dc.l      $4b5a6978
            .dc.l      $8796a5b4
            .dc.l      $c3d2elf0
            .dc.l      $flf2f4f8       * diagnostic test pattern
            .dc.l      $f9fafcfe
            .dc.l      $ff7fbfdf
            .dc.l      $ef6fafcf
            .dc.l      $4f8f0f07       * diagnostic test pattern
            .dc.l      $0b0d0e06
            .dc.l      $0a0c0408
            .dc.l      $04020100
            .dc.l      0                * diagnostic test area directory
            .dc.l      0                * start of dump area
            .dc.w      0                * length of dump area
            .dc.l      0                * start of sector lockout table
            .dc.w      0                * length of sector lockout table
            .dc.l      1                * configuration parameters sector
            .dc.b      1                * length of config area (sectors)
            .dc.b      0,0,0           * unused reserved
            .dc.l      0,0,0,0,0       * unused, reserved
            .dc.l      0,0,0,0,0,0
            .dc.l      0,0,0,0,0,0,0
            .dc.l      0,0,0,0,0,0,0
            .dc.l      0,0,0,0,0,0,0
            .dc.b      'EXORMACS'      * let's hear it for Motorola

*
*      end of volume id

```

Listing H-8. (continued)

```

*
*      begin configuration area
*
      .dc.l    0,0          * reserved
      .dc.w    $0f         * attributes word
      .dc.w    0           * reserved
      .dc.l    0,0,0
      .dc.b    8           * sectors per track on disk
      .dc.b    2           * sides on disk
      .dc.w    80          * cylinders on disk
      .dc.w    0           * reserved
*
* WARNING: the following word is modified by
* putboot for the hard disk. It must remain
* at this offset.
*
      .dc.w    512         * sector size
      .dc.l    0           * reserved
      .dc.w    40          * precompensation cylinder
      .dc.l    0           * reserved
*
*      end configuration area
*
      .ds.b    $800-(*-first) * skip rest of first track
*
*      begin boot info proper
*
      .globl   _bootdisk
      .even
      .dc.l    $4000       * stack pointer
      .dc.l    start       * program counter
start: move    #$2700,sr
      move.w   d0,_bootdisk
      and.b    #$fc,$f19f05
      move.b   #$10,$f19f09
      clr.b    $f19f0b
      move.b   #$80,$f19f11
      .end

```

NOTES