CP/M-68K™ Operating System Programmer's Guide

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

The CP/M-68K Operating System Programmer's 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 operating system designed for the MOTOROLA® MC68000 or a compatible 68000 microprocessor. CP/M-68K requires a minimum of 64K bytes of random access memory (RAM) to run its base-level system, which contains the CP/M® commands and the utilities listed below.

- CP/M Built-in Commands:

  DIR
  DIRS
  ERA
  REN
  SUBMIT
  TYPE
  USER

- Standard CP/M Utilities:

  DDT-68K™
  ED
  PIP
  STAT

- Programming Utilities:

  Archive (AR68)
  DUMP
  Relocation (RELOC)
  SIZE68
  SENDC68

- Programming Tools

  Assembler (AS68)
  Linker (LO68)
  C Compiler*
  C Preprocessor*

* Described in the C Programming Guide for CP/M-68K.

CP/M-68K requires a minimum of 128K bytes of RAM to run the programming tools distributed with CP/M-68K.

The CP/M-68K file system is based on and is upwardly compatible with the CP/M-80™ Version 2.2 and CP/M-86® Version 1.1 file systems. However, CP/M-68K supports a much larger file size with a maximum of 32 megabytes per file.

CP/M-68K supports a maximum of 16 disk drives, with 512 megabytes per drive. CP/M-68K supports other peripheral devices that the Basic I/O System (BIOS) assigns to one of the four logical devices: LIST, CONSOLE, AUXILIARY INPUT, or AUXILIARY OUTPUT.

This guide describes the programming interface to CP/M-68K. The first few sections in this guide discuss the CP/M-68K architecture, memory models, executable programs, and file system access functions. Latter sections of this guide describe programming tools and utilities distributed with your CP/M-68K system.

This guide assumes you are an experienced programmer familiar with the basic programming concepts of assembly language. If you are not familiar with the Motorola 68000 assembly language, refer to Motorola manuals listed below.

- <u>16-BIT Microprocessor User's Manual,</u> third edition MC68000UM(AD3)
- <u>M68000 Resident Structured Assembler Reference Manual</u> M68KMASM(D4)

Before you can use the facilities in this guide, your CP/M-68K system must be configured for your hardware environment. Normally, your system is configured for you by the manufacturer of your computer or the software distributor. However, if you have an unusual hardware environment, this may not be the case. Refer to the <u>CP/M-68K Operating System System Guide</u> for details on how to configure your system for a custom hardware environment.

New Functions and Implementation Changes

CP/M-68K has six new Basic Disk Operating System (BDOS) functions and additional implementation changes in the BDOS functions and data structures that differ from other CP/M systems. The new BDOS functions and implementation changes are listed in Appendix F.

Table F-4 in Appendix F contains functions and commands supported by other CP/M systems, but that are not supported by CP/M-68K.

Table of Contents

Table of Contents
(continued)

Table of Contents (continued)

Table of Contents (continued)

Table of Contents
(continued)

Appendixes

Appendixes (continued)

Tables, Figures, and Listings

Tables

Tables, Figures, and Listings (continued)

Section 1
Introduction to CP/M-68K

CP/M-68K contains most of the facilities of other CP/M systems with additional features required to address up to sixteen megabytes of main memory available on the 68000 microprocessor. 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 supports a maximum of sixteen drives with up to 512 megabytes on each drive and a maximum file size of 32 megabytes.

1.1 CP/M-68K Architecture

The CP/M-68K operating system resides in the file CPM.SYS on the system disk. A cold start loader resides on the first two tracks of the system disk and loads the CPM.SYS file into memory during a cold start. The CPM.SYS file contains the three program modules described in Table 1-1.

Table 1-1. Program Modules in the CPM.SYS File

Module	Mnemonic	Description
Console Command Processor	CCP	User interface that parses the user command line.
Basic Disk Operating System	BDOS	Provides functions that access the file system.
Basic I/O System	BIOS	Provides functions that interface peripheral device drivers for I/O processing.

The sizes of the CCP and BDOS modules are fixed for a given release of CP/M-68K. The BIOS custom module, normally supplied by the computer manufacturer or software distributor depends on the system configuration, which varies with the implementation. Therefore, the size of the BIOS also varies with the implementation.

The CP/M-68K operating system can be loaded to execute in any portion of memory above the locations reserved in the 68000 architecture for the exception vectors (0000H through 03FFH). All CP/M-68K modules remain resident in memory. The CCP cannot be used as a data area subsequent to transient program load.

1.2 Transient Programs

After CP/M-68K is loaded in memory, the remaining contiguous address space that is not occupied by the CP/M-68K operating system is called the Transient Program Area (TPA). CP/M-68K loads executable files, called command files, from disk to the TPA. These command files are also called transient commands or transient programs because they temporarily reside in memory, rather than being permanently resident in memory and configured in CP/M-68K. The format of a command file is described in Section 3.

1.3 File System Access

Programs do not specify absolute locations or default variables when accessing CP/M-68K. Instead, programs invoke BDOS and BIOS functions. Section 4 describes the BDOS functions in detail. Appendix A lists the BIOS calls. Refer to the CP/M-68K Operating System System Guide for detailed descriptions of the BIOS functions. In addition to these functions, CP/M-68K decreases dependence on absolute addresses by maintaining a base page in the TPA for each transient program in memory. The base page contains initial values for the File Control Block (FCB) and the Direct Memory Access (DMA) buffer. For details on the base page and loading transient programs, refer to Section 2.

1.4 Programming Tools and Commands

CP/M-68K contains a full set of programming tools that include an assembler (AS68), linker, (LO68), Archive Utility (AR68), Relocation Utility (RELOC), DUMP Utility, SIZE68, and SENDC68. Each of these tools is discussed in the latter part of this guide. Table 1-3 lists the commands that invoke these tools. Table 1-2 describes command conventions used in this manual. Tables 1-4 and 1-5 list other commands supported by CP/M-68K and the manual in which they are documented.

Table 1-2. CP/M-68K Programmer's Guide Conventions

Convention	Meaning
[]	Square brackets in a command line enclose optional parameters.
nH	The capital letter H follows numeric values that are represented in hexadecimal notation.
numeric values	Unless otherwise stated, numeric values are represented in decimal notation.
(n)	BDOS function numbers are enclosed in parentheses when they appear in text.

Table 1-2. (continued)

Convention	Meaning
. . . or ...	A vertical or horizontal elipsis indicates missing elements in a series unless noted otherwise.
RETURN	The word RETURN refers to the RETURN key on the keyboard of your console. Unless otherwise noted, to invoke a command, you must press RETURN after you enter a command line from your console.
CTRL-X	The mnemonic CTRL-X instructs you to press the key labeled CTRL while you press another key indicated by the variable X. For example, CTRL-C instructs you to press the CTRL key while you simultaneously press the key lettered C.

Table 1-3 describes commands used in the CP/M-68K Operating System Programmer's Guide.

Table 1-3. CP/M-68K Commands (Programmer's Guide)

Command	Description
AR68	Invokes the Archive Utility (AR68). AR68 creates a library and/or deletes, adds, or extracts object modules from an existing library, such as the C Run-time Library.
AS68	Invokes the Assembler (AS68).
DDT	Invokes DDT-68K, the CP/M-68K debugger.
DUMP	Invokes the DUMP Utility that prints the contents of a file in hexadecimal and ASCII notation.
LO68	Invokes the Linker.
NM68	Invokes the NM68 Utility that prints the symbol table of an object or command file.

Table 1-3. (continued)

Command	Description
RELOC	Invokes the Relocation Utility that relocates a command file containing relocation information to an absolute address.
SENDC68	Invokes the SENDC68 Utility that converts a command file to the MOTOROLA S-record format.
SIZE68	Invokes the SIZE68 Utility that prints the total size of a command file and the size of each program segment in the file.

Table 1-4 describes commands used in the CP/M-68K Operating System User's Guide.

Table 1-4. CP/M-68K Commands (User's Guide)

Command	Description
DIR*	Displays the directory of files on a specified disk.
DIRS*	Displays the directory of system files on a specified disk.
ED	Invokes the CP/M-68K text editor.
ERA*	Erases one or more specified files.
PIP	Copies, combines, and transfers specified files between peripheral devices.
REN*	Renames an existing file to the new name specified in the command line.
SUBMIT*	Executes a file of CP/M commands.
TYPE*	Displays the contents of an ASCII file on the console.
USER*	Displays or changes the current user number.

* CP/M-68K built-in commands

Table 1-5 describes commands used in the C Programming Guide for CP/M-68K.

Table 1-5. CP/M-68K Commands (C Manual)

Command	Description
C	Invokes a submit file that invokes the C compiler for compiling CP/M-68K C source files.
CP68	Invokes the C preprocessor for processing macros when you compile CP/M-68K C source files.
C068	Invokes the C parser when you compile CP/M-68K C source files.
C168	Invokes the assembly language code generator for the CP/M-68K C compiler when you compile C source files.

1.5 CP/M-68K File Specification

The CP/M-68K file specification is compatible with other CP/M systems. The format contains three fields: a 1-character drive select code (d), a 1- through 8-character filename (f...f), and a 1- through 3-character filetype (ttt) field as shown below.

Format	d:ffffffff.ttt
Example	B:MYRAH.DAT

The drive select code and filetype fields are optional. A colon (:) delimits the drive select field. A period (.) delimits the filetype field. These delimiters are required only when the fields they delimit are specified.

Values for the drive select code range from A through P when the BIOS implementation supports 16 drives, the maximum number allowed. The range for the drive code is dependent on the BIOS implementation. Drives are labeled A through P to correspond to the 1 through 16 drives supported by CP/M-68K. However, not all BIOS implementations support the full range.

The characters in the filename and filetype fields cannot contain delimiters (the colon and period) and must be upper-case for the CCP to parse the file specification. The CCP cannot access a file that contains delimiters or lower-case characters. A command line and its file specifications, if any, that are entered at the CCP level are automatically put in upper-case internally before the CCP parses them.

However, not all commands and file specifications are entered at the CCP level. CP/M-68K does not prevent you from including delimiters or lower-case characters in file specifications that are created or referenced by functions that bypass the CCP. For example, the BDOS Make File Function (22) allows you to create a file specification that includes delimiters and lower-case characters, although the CCP cannot parse and access such a file.

In addition to the delimiter characters already mentioned, you should avoid using the delimiter characters in Table 1-6 in the file specification of a file you create. Several CP/M-68K built-in commands and utilities have special uses for these characters.

Table 1-6. Delimiter Characters

Character	Description
[]	square brackets
()	parentheses
<>	angle brackets
=	equals sign
*	asterisk
&	ampersand
,	comma
!	exclamation point
|	bar
?	question mark
/	slash
$	dollar sign
.	period
:	colon
;	semicolon
+	plus sign
-	minus sign

1.6 Wildcards

CP/M-68K supports two wildcards, the question mark (?) and the asterisk (*). Several utilities and BDOS functions allow you to specify wildcards in a file specification to perform the operation or function on one or more files. However, BDOS functions support only the ? wildcard.

The ? wildcard matches any character in the character position occupied by this wildcard. For example, the file specification M?RAH.DAT indicates the second letter of the filename can be any alphanumeric character if the remainder of file specification matches. Thus, the ? wildcard matches exactly one character position.

The * wildcard matches one or more characters in the field or remainder of a field that this wildcard occupies. CP/M-68K internally pads the field or remaining portion of the field occupied by the * wildcard with ? wildcards before searching for a match. For example, CP/M-68K converts the file B*.DAT to B???????.DAT before searching for a matching file specification. Thus, any file that starts with the letter B and has a filetype of DAT matches this file specification.

For details on wildcard support by a specific BDOS function, refer to the description of the function in Section 4 of this guide. For additional details on these wildcards and support by CP/M-68K utilities, refer to the CP/M-68K Operating System User's Guide.

1.7 CP/M-68K Terminology

Table 1-7 lists the terminology used throughout this guide to describe CP/M-68K values and program components.

Table 1-7. CP/M-68K Terminology

Term	Meaning
Nibble	4-bit value
Byte	8-bit value
Word	16-bit value
Longword	32-bit value
Address	32-bit value that specifies a location in storage
Offset	A fixed displacement defined by the user to reference a location in storage, other data source, or destination.
Text Segment	The section of a program that contains the program instructions.
Data Segment	The section of a program that contains initialized data.
Block Storage Segment (bss)	The section of a program that contains uninitialized data.

End of Section 1

Section 2
The CCP and Transient Programs

This section discusses the Console Command Processor (CCP), built-in and transient commands, loading and exiting transient programs, and CP/M-68K memory models.

2.1 CCP Built-in and Transient Commands

After an initial cold start, CP/M-68K displays a sign-on message at the console. Drive A, containing the system disk, is logged in automatically. The standard prompt (>), preceded by the letter A for the drive, is displayed on the console screen. This prompt informs the user that CP/M-68K is ready to receive a command line from the console.

In response to the prompt, a user types the filename of a command file and a command tail, if required. CP/M-68K supports two types of command files, built-in commands and transient commands. Built-in commands are configured and reside in memory with CP/M-68K. Transient commands are loaded in the TPA and do not reside in memory allocated to CP/M-68K. The list below contains the seven built-in commands that CP/M-68K supports.

```
DIR
DIRS
ERA
REN
TYPE
USER
SUBMIT
```

A transient command is a machine-readable executable program file in memory. A transient command file is loaded from disk to memory. Section 3 describes the format of transient command files.

When the user enters a command line, the CCP parses it and tries to execute the file specified. The CCP assumes a file is a command file when it has any filetype other than .SUB. When the user specifies only the filename but not the filetype, the CCP searches for and tries to execute a file with a matching filename and a filetype of either 68K or three blanks. The CCP searches the current user number and user number 0 for a matching file. If a command file is not found, but the CCP finds a matching file with a filetype of SUB, the CCP executes it as a submit file.

2.2 Loading A Program In Memory

Either the CCP or a transient program can load a program in memory with the BDOS Program Load Function (59) described in Section 4.5. After the program is loaded, the TPA contains, the program segments (text, data, and bss), a user stack, and a base page. A base page exists for each program loaded in memory. The base page is a 256-byte data structure that defines a program's operating environment. Unlike other CP/M systems, the base page in CP/M-68K does not reside at a fixed absolute address prior to being loaded. The BDOS Program Load Function (59) determines the absolute address of the base page when the program is loaded into memory. The BDOS Program Load Function (59) and the CCP or the transient program initialize the contents of the base page and the program's stack as described below.

2.2.1 Base Page Initialization By The CCP

The CCP parses up to two filenames following the command in the input command line. The CCP places the properly formatted FCBs in the base page. The default DMA address is initialized at an offset of 0080H in the base page. The default DMA buffer occupies the second half of the base page. The CCP initializes the default DMA buffer to contain the command tail, as shown in Figure 2-1. The CCP invokes the BDOS Program Load Function (59) to load the transient program before the CCP parses the command line.

Program Load, Function 59, allocates space for the base page and initializes base page values at offsets 0000H through 0024H from the beginning of the base page (see Appendix C). Values at offsets 0025H through 0037H are not initialized; but the space is reserved. The CCP parses the command line and initializes values at offsets 0038H through 00FFH. Before the CCP gives control to the loaded program, the CCP pushes the address of the transient program's base page and a return address within the CCP on the user stack. When the program is invoked, the top of the stack contains a return address within the CCP, which is pointed to by the stack pointer, register A7. The address of the program's base page is located at a 4-byte offset from the stack pointer.

2.2.2 Loading Multiple Programs

Multiple programs can reside in memory, but the CCP can load only one program at a time. However, a transient program, loaded by the CCP, can load one or more additional programs in memory. A program loads another program in memory by invoking the BDOS Program Load Function (59). Normally, the CCP supplies FCBs and the command tail to this function. The transient program must provide this information, if required, for any additional programs it loads when the CCP is not present.

2.2.3 Base Page Initialization By A Transient Program

A transient program invokes the BDOS Program Load Function (59) to load an additional program. The BDOS Program Load Function allocates space and initializes base page values at offsets 0000H through 0024H for the program as described in Section 2.2.1. The transient program must initialize the base page values that the CCP normally supplies, such as FCBs, the DMA address, and the command tail, if the program being loaded requires these values. The command tail contains the command parameters but not the command. The format of the command tail in the base page consists of a 1-byte character count, followed by the characters in the command tail, and terminated by a null byte as shown in Figure 2-1. The command tail cannot contain more than 126 bytes plus the character count and the terminating null character.

Count	Characters in the Command Tail	0
1 byte	N bytes $\leq$ 126 bytes	

Figure 2-1. Format of the Command Tail in the DMA Buffer

Unlike the CCP, a transient program does not necessarily push the address of its base page and a return address on the user stack before giving control to the program that it loads with the Program Load Function. The transient program can be designed to push these addresses on the user stack of the program it loads if the program uses the base page.

The address of the base page for the loaded program is not pushed on the user stack by the Program Load Function (59). Instead, it is returned in the load parameter block (LPB), which is used by the BDOS Program Load Function. Appendix C summarizes the offsets and contents of a base page. Appendix B contains two examples, an assembly language program and a C language program, which illustrate how a transient program loads another program with the BDOS Program Load Function (59), but without the CCP.

2.3 Exiting Transient Programs

CP/M-68K supports the two ways listed below to exit a transient program and return control to the CCP.

- Interactively, the user types CTRL-C at the console, the default I/O device

- Program a return to the CCP with either:

  1) a Return From Subroutine (RTS) Instruction

  2) the BDOS System Reset Function (0)

A user typing CTRL-C from the console returns control to the CCP only if the program uses any of the BDOS functions listed below.

- Console Output (2)
- Print String (9)
- Read Console Buffer (10)

On input, CTRL-C must be the first character that the user types on the line. CTRL-C terminates execution of the main program and any additional programs loaded beyond the CCP level. For example, a user who types CTRL-C while debugging a program terminates execution of the program being debugged and DDT-68K before the CCP regains control.

Typing CTRL-C in response to the system prompt resets the status of all disks to read-write.

To program a return to the CCP, specify a Return from Subroutine (RTS) Instruction or the BDOS System Reset Function (0).

The RTS instruction must be the last one executed in the program and the top of the stack must contain the system-supplied return address for control to return to the CCP. When a transient program begins execution, the top of the stack contains this system-supplied return address. If the program modifies the stack, the top of the stack must contain this system-supplied return address before an RTS instruction is executed.

Invoking the BDOS System Reset Function (0) described in Section 4.5 is equivalent to programming a return to the CCP. This function performs a warm boot, which terminates the execution of a program before it returns program control to the CCP.

2.4 Transient Program Execution Model

The memory model shown in Figure 2-2 illustrates the normal configuration of the CP/M-68K operating system after the CCP loads a transient program. CP/M-68K divides memory in two categories: System and the Transient Program Area (TPA).

CP/M-68K System memory contains the Basic Disk Operating System (BDOS), the Basic I/O System (BIOS), the Console Command Processor (CCP), and Exception Vectors. The bootstrap program initializes the memory locations in which these components reside. Other than exception vectors, which reside in memory locations 0000H through 03FFH, the remaining components can reside anywhere in memory, provided the BDOS and CCP are contiguous.

The TPA consists of contiguous memory locations that are not occupied by the CP/M-68K operating system. A user stack, a base page, the three program segments: a text segment, an initialized data segment, and a block storage segment (bss) exist for each transient program loaded in the TPA. The BDOS Program Load Function (59) loads a transient program in the TPA. If memory locations are not specified when the transient program is linked, the program is loaded in the TPA as shown in Figure 2-2.

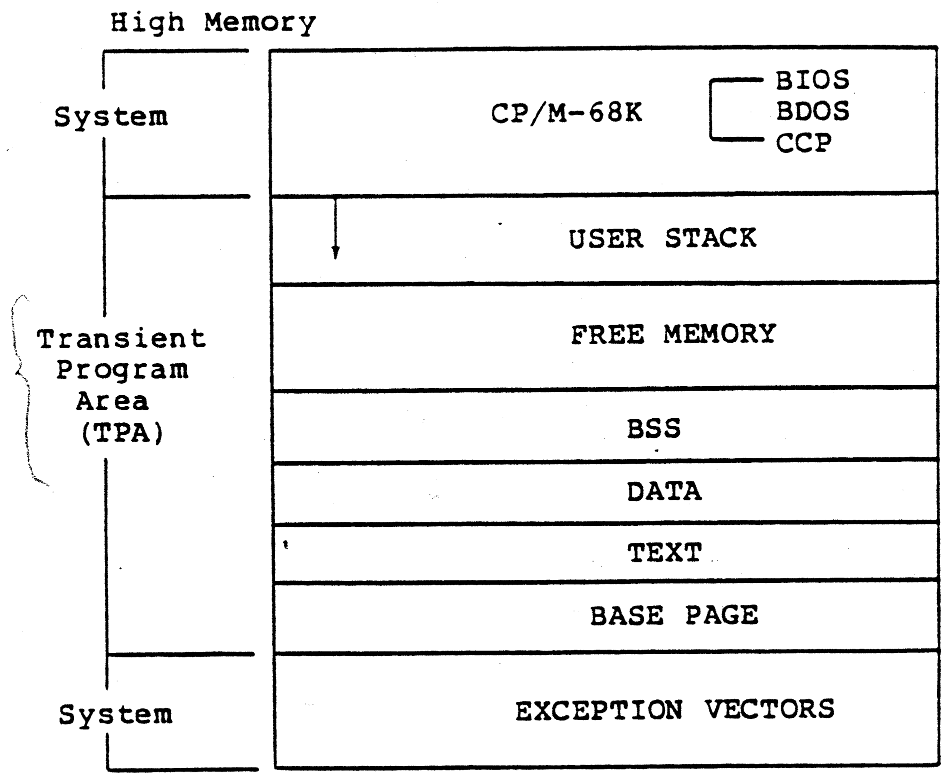

Figure 2-2. CP/M-68K Default Memory Model

Some systems can configure and load CP/M-68K in such a manner that one or more portions of memory cannot be addressed by the CP/M-68K operating system (see Figure 2-3). CP/M-68K cannot access this memory. CP/M-68K does not know the memory exists and cannot define or configure the memory in the BIOS because CP/M-68K requires that the TPA is one contiguous area. However, a transient program that knows this memory exists can access it. Also, note that CP/M-68K does not support or require memory management.

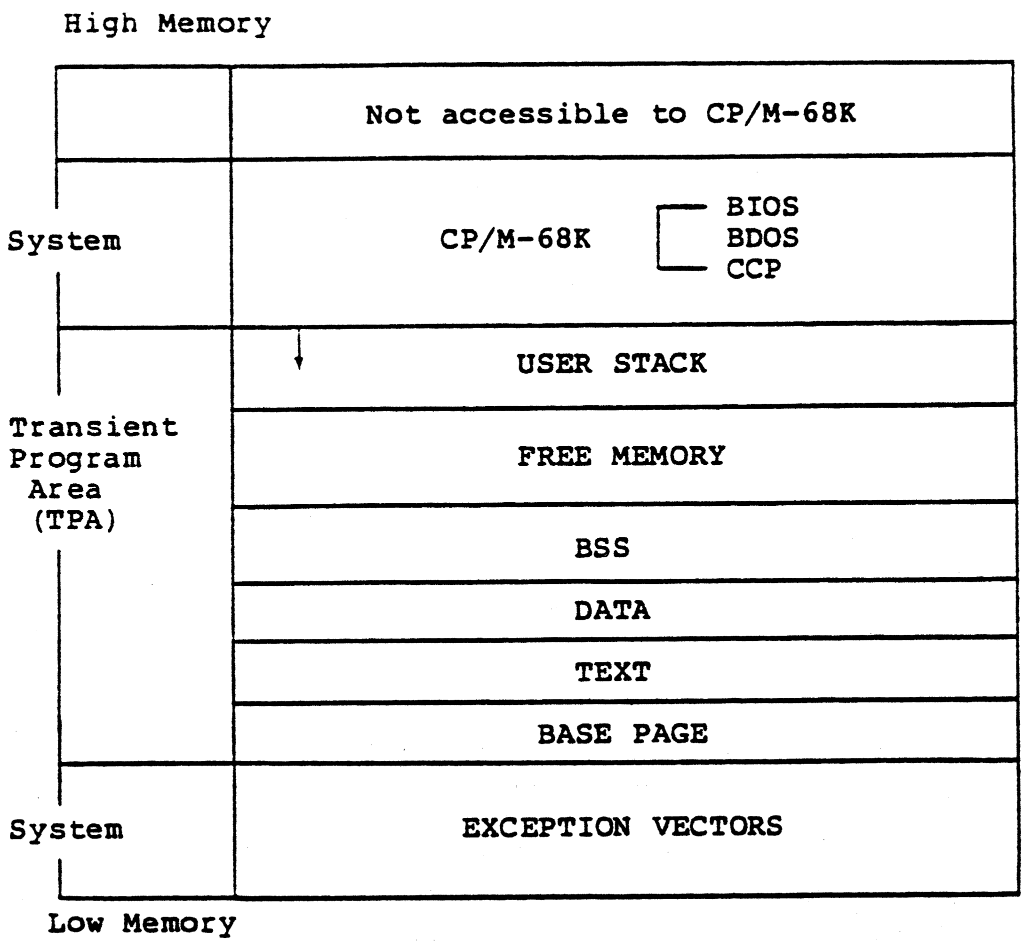

Figure 2-3. CP/M-68K Memory Model with Inaccessible Memory

End of Section 2

Section 3
Command File Format

This section describes the format of a command file. The linker processes one or more compiled or assembled files to produce an executable machine-readable file called a command file. By default, a command file has a filetype of 68K.

A command file always contains a header, two program segments (a text segment and an initialized data segment), and optionally contains a symbol table and relocation information. These components are described in the following sections.

3.1 The Header and Program Segments

The header, the first component in the file, specifies the size and starting address of the other components in the command file, which are listed below.

- Program segments:

  text: contains the program instructions. Hart kein Beginn

  data: contains data initialized within the command file.

  block storage segment (bss):
  specifies space for uninitialized data generated by the program during execution. Although space for the bss is specified in the source command file, the space is not allocated until the command file is loaded in memory. Therefore, the source command file on the disk contains no uninitialized data.

- Symbol table: defines referenced symbols.

- Relocation information:
  specifies the relative relocation of each word within each program segment, if required.

The command file format supports two types of headers. The size and content of each type differs. The contiguity of the program segments determines which type of header a command file contains. When the program segments must be contiguous, the file contains a 14-word header in the format shown in Figure 3-1. When the program segments can be noncontiguous, the file contains an 18-word header in the format shown in Figure 3-2. The first word of each header contains a hexadecimal integer that defines which type of header the file contains.

unterscheiden [illegible]

Byte Offset	Sample Values	Size	Contents
0H	601AH	1 Word	Integer 601AH denotes text, data, and bss are contiguous
2H	2376H	1 Longword	Number of bytes in text segment
6H	422H	1 Longword	Number of bytes in data segment
0AH	1806H	1 Longword	Number of bytes in bss
0EH	142H	1 Longword	Number of bytes in symbol table
12H	0000H	1 Longword	Reserved; always zero
16H	500H	1 Longword	Beginning of text segment and of program execution
1AH	00H	1 Word	Integer flag for relocation bits; if 0, relocation bits exist; if not 0, no relocation bits exist.

Figure 3-1. Header for Contiguous Program Segments

To create a file that can contain noncontiguous program segments, specify the -T, -D, and -B linker options described in Section 6 when you link the files. The header, identified by 601BH denotes the size and location of each program segment. Note that this header indicates the program segments can be noncontiguous and does not imply the segments must be noncontiguous. See Figure 3-2.

Byte Offset	Sample Values	Size	Contents
0H	601BH	1 Word	Integer 601BH denotes text, data, and bss can be noncontiguous
2H	57864H	1 Longword	Number of bytes in text segment
6H	446H	1 Longword	Number of bytes in data segment
0AH	2568H	1 Longword	Number of bytes in bss
0EH	69H	1 Longword	Number of bytes in symbol table
12H	0000H	1 Longword	Reserved; always zero
16H	500H	1 Longword	Beginning of text segment and of program execution
1AH	00H	1 Word	Integer flag for relocation bits; if 0, relocation bits exist; if not 0, no relocation bits exist.
1CH	57D64H	1 Longword	Starting address of data segment
20H	581AAH	1 Longword	Starting address of bss

Figure 3-2. Header for Noncontiguous Program Segments

The linker computes the size of the segments in bytes. The result is always rounded up to an even number. For example, the linker adds a byte to a program segment that contains an odd number of bytes. The linker does not include the size of the header when it computes the size of the segments.

After a program is linked and loaded in memory, it contains three program segments: text, initialized data, and uninitialized data (bss). The BDOS Program Load Function (59) zeroes the bss when a program is loaded. A program begins execution at the beginning of the text segment. See Figures 3-1 and 3-2 above.

3.2 The Symbol Table

The symbol table lists all the symbols specified in a program. Each symbol in the table consists of a 7-word entry that describes the symbol name, type, and value. See Figure 3-3.

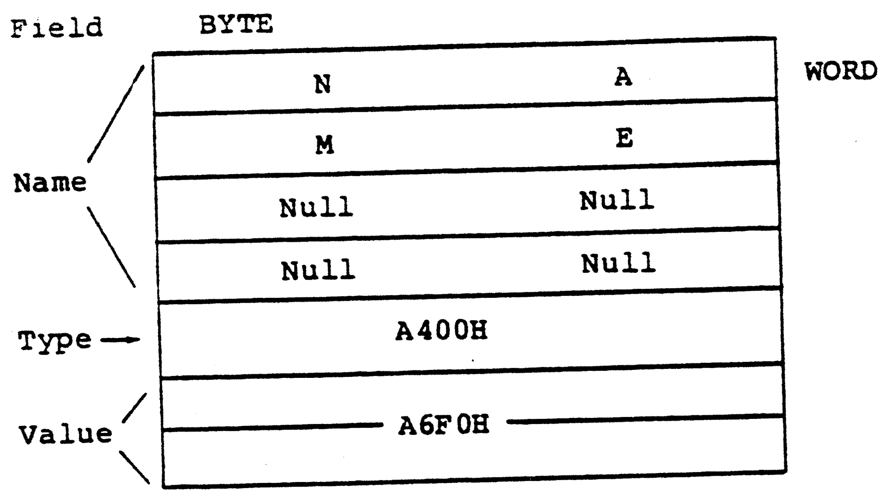

Figure 3-3. Entry in Symbol Table

The name field, the first four words, contains the ASCII name of the symbol. This field is padded with null characters when the ASCII name is less than eight characters. The fifth word contains the symbol type. Valid values are listed in Table 3-1.

Table 3-1. Values For Symbol Types

Type	Value
defined	8000H
equated	4000H
global	2000H
equated register	1000H
external reference	800H
data based relocatable	400H
text based relocatable	200H
bss based relocatable	100H

When specifying a symbol type with multiple characteristics, the linker uses an OR instruction to combine several of the above values. For example, to specify a defined, global, data based, relocatable symbol, the linker combines the values of each characteristic for a value of A400H.

The last field in an entry is the value field. It consists of a longword that contains the value of the symbol. The value can be an address, a register number, the value of an expression, or some other value. When the value field is nonzero and the type field contains an external symbol, the linker interprets the symbol to be a common region in which the size of the region equals the value of the symbol.

3.2.1 Printing The Symbol Table

Use the NM68 Utility to print the symbol table of an object or command file. To invoke this utility, specify the NM68 command and filename as shown below.

```
NM68 filename.O [>filespec]
```

You must enter the filename of an object file or a command file. You can optionally redirect the NM68 output from your console to a file. To redirect the NM68 output to a file, specify a greater than sign (>) followed by a file specification after the filename and filetype of the file from which NM68 prints the symbol table.

The NM68 utility does not sort the symbols; it prints them in the order in which they appear in the file. Each symbol name is printed, followed by its value and one or more of the type descriptors listed below:

- equ (equated)
- global
- equreg (equated register)
- external
- data
- text
- bss
- abs (absolute)

3.3 Relocation Information

Relocation information is optional. The header relocation word, the last word in the header, indicates whether relocation information exists. When its value is zero, relocation information exists. None exists when the its value is nonzero.

Relocation information specifies the relocation of words in program segments. One word of relocation information, called a relocation word, exists for each word in each of the program segments. The assembler and compiler generate relocation words for external symbols and address constants referenced in the text and data program segments. The linker and sometimes the BDOS Program Load Function (59) use these relocation words as described below.

The linker resolves external symbols when linking files by modifying bits 0 through 2 of each relocation word that references an external symbol. After being modified, the relocation word indicates the program segment that the symbol references. Therefore, instead of referencing an external symbol, the relocation word references a word located in one of the program segments. Because the linker only modifies relocation words that refer to external symbols, relocation words that do not reference this type of symbol have the same value in the source file input to the linker and the executable file output by the linker.

The BDOS Program Load Function uses relocation words when it loads a program in a location other than the one at which it was linked. The Program Load Parameter Block (LPB) used by the Program Load Function specifies where the program is loaded. When the LPB specifies a location other than the linked location, the BDOS computes a bias (the difference between where a program segment is linked and where it will be loaded in memory). When loading the program, the BDOS adds the bias as indicated by the relocation words to the address of the relocatable words in the text and/or data segments. However, when the BDOS loads the program in the memory locations at which it was linked, the BDOS does not use the relocation words.

3.3.1 The Format Of A Relocation Word

A relocation word is a 16-bit quantity. Bits 0 through 2 in each relocation word indicate the type of address referenced and, if applicable, designate the segment to which the relocation word refers. Values for these bits are described in Table 3-2.

Table 3-2. Relocation Word Values (bits 0 through 2)

Value	Description
00	no relocation information required; the reference is absolute
01	reference relative to the base address of the data segment
02	reference relative to the base address of the text segment
03	reference relative to the base address of the bss
04	references an undefined symbol
05	references the upper word of a longword; the next relocation word contains the value determining whether the reference is absolute or dependent on the base address of the text or data segments, or the bss.

Table 3-2. (continued)

Value	Description
06	16-bit PC-relative reference
07	indicates the first word of an instruction, which does not require relocation information.

The remaining bits, 3 through 15, are not used unless the program references an external symbol. In that case, these bits contain an index to the symbol table. The index specifies the entry number of the symbol listed in the symbol table. Entry numbers in the symbol table are numbered sequentially starting with zero.

End of Section 3

Section 4
Basic Disk Operating System (BDOS) Functions

To access a file or a drive, to output characters to the console, or to reset the system, your program must access the CP/M-68K file system through the Basic Disk Operating System (BDOS). The BDOS provides functions that allow your program to perform these tasks. Table 4-1 summarizes the BDOS functions.

Table 4-1. CP/M-68K BDOS Functions

F#	Function	Type
0	System Reset	System/Program Control
1	Console Input	Character I/O, Console Operation
2	Console Output	Character I/O, Console Operation
3	Auxiliary Input*	Character I/O, Additional Serial I/O
4	Auxiliary Output*	Character I/O, Additional Serial I/O
5	List Output	Character I/O, Additional Serial I/O
6	Direct Console I/O	Character I/O, Console Operation
7	Get I/O Byte*	I/O Byte
8	Set I/O Byte*	I/O Byte
9	Print String	Character I/O, Console Operation
10	Read Console Buffer	Character I/O, Console Operation
11	Get Console Status	Character I/O, Console Operation
12	Return Version Number	System Control
13	Reset Disk System	Drive
14	Select Disk	Drive
15	Open File	File Access
16	Close File	File Access
17	Search for First	File Access
18	Search for Next	File Access
19	Delete File	File Access
20	Read Sequential	File Access
21	Write Sequential	File Access
22	Make File	File Access
23	Rename File	File Access
24	Return Login Vector	Drive
25	Return Current Disk	Drive
26	Set DMA Address	File Access
28	Write Protect Disk	Drive
29	Get Read-Only Vector	Drive
30	Set File Attributes	File Access
31	Get Disk Parameters	Drive
32	Set/Get User Code	System/Program Control
33	Read Random	File Access
34	Write Random	File Access
35	Compute File Size	File Access

* Must be implemented in the BIOS

Table 4-1. (continued)

F#	Function	Type
36	Set Random Record	File Access
37	Reset Drive	Drive
40	Write Random With Zero Fill	File Access
46	Get Disk Free Space	Drive
47	Chain To Program	System/Program Control
48	Flush Buffers	System/Program Control
50	Direct BIOS Call	System/Program Control
59	Program Load	System/Program Control
61	Set Exception Vector	Exception
62	Set Supervisor State	Exception
63	Get/Set TPA Limits	Exception

4.1 BDOS Functions and Parameters

To invoke a BDOS function, you must specify one or more parameters. Each BDOS function is identified by a number, which is the first parameter you must specify. The function number is loaded in the first word of data register D0 (D0.W). Some functions require a second parameter, which is loaded, depending on its size, in the low order word (D1.W) or longword (D1.L) of data register D1. Byte parameters are passed as 16-bit words. The low order byte contains the data, and the high order byte should be zeroed. For example, the second parameter for the Console Output Function (2) is an ASCII character, which is a byte parameter. The character is loaded in the low order byte of data register D1 (D1.W). Some BDOS functions return a value, which is passed in the first word of data register D0 (D0.W). The hexadecimal value FFFF is returned in register D0.W when you specify an invalid function number in your program. Table 4-2 illustrates the syntax and summarizes the registers that BDOS functions use.

Table 4-2. BDOS Parameter Summary

BDOS Parameter	Register
Function Number	D0.W
Word Parameter	D1.W
Longword Parameter	D1.L
Return Value, if any	D0.W

4.1.1 Invoking BDOS Functions

After the parameters for a function are loaded in the appropriate registers, the program must specify a trap 2 instruction to access the BDOS and invoke the function. The example below illustrates the assembler syntax required to invoke the Console Output Function (2).

```
        move.w #2,d0      *Moves the function number to the first
                          *word in data register D0.

        move.w #'U,dl     *Moves the ASCII character upper-case U
                          *to the first word in data register Dl.

        trap #2           *Accesses the BDOS to invoke the function.
```

The example above outputs the ASCII character upper-case U to the console. The assembler moves instructions load register D0.W with the number 2 for the BDOS Console Output Function and register Dl.W with the ASCII character upper-case U. A pair of single ('') or double ("") quotation marks must enclose an ASCII character. The trap 2 instruction invokes the BDOS Output Console Function, which echos the character on the console's screen.

4.1.2 Organization Of BDOS Functions

The parameters and operation performed by each BDOS function are described in the following sections. Each BDOS function is categorized according to the function it performs. The categories are listed below.

- File Access
- Drive Access
- Character I/O
- System/Program Control
- Exception

As you read the description of the functions, notice that some functions require an address parameter designating the starting location of the direct memory access (DMA) buffer or file control block (FCB). The DMA buffer is an area in memory where a 128-byte record resides before a disk write function and after a disk read operation. Functions often use the DMA buffer to obtain or transfer data. The FCB is a 33- or 36-byte data structure that file access functions use. The FCB is described in Section 4.2.1.

4.2 File Access Functions

This section describes file access functions that create, delete, search for, read, and write files. They include the functions listed in Table 4-3.

Table 4-3. File Access Functions

Function	Function Number
Open File	15
Close File	16
Search For First	17
Search For Next	18
Delete File	19
Read Sequential	20
Write Sequential	21
Make File	22
Rename File	23
Set DMA Address	26
Read Random	33
Write Random	34
Compute File Size	35
Write Random With Zero Fill	40

4.2.1 A File Control Block (FCB)

Most of the file access functions in Table 4-3 require the address of a File Control Block (FCB). A FCB is a 33- or 36-byte data structure that provides file access information. The FCB can be 33 or 36 bytes when a file is accessed sequentially, but it must be 36 bytes when a file is accessed randomly. The last three bytes in the 36-byte FCB contain the random record number, which is used by random I/O functions and the Compute File Size Function (35). The starting location of a FCB must be an even-numbered address. The format of a FCB and definitions of each of its fields are below.

Field	dr	f1	f2	...	f8	t1	t2	t3	ex	s1	s2	rc	d0	...	dn	cr	r0	r1	r2
Byte	00	01	02	...	08	09	10	11	12	13	14	15	16	...	31	32	33	34	35

dr	drive code (0 - 16) 0 => use default drive for file 1 => auto disk select drive A, 2 => auto disk select drive B, ... 16=> auto disk select drive P.
f1...f8	contain the filename in ASCII upper-case. High bit should equal 0 when the file is opened.
t1,t2,t3	contain the filetype in ASCII upper-case. The high bit should equal 0 when the file is opened. For the Set File Attributes Function (see Section 4.2.13), t1', t2', and t3' denote the high bit. The list below indicates which attributes are set when these bits are set and equal the value 1. t1' = 1 => Read-Only file t2' = 1 => SYS file t3' = 1 => Archive
ex	contains the current extent number, normally set to 00 by the user, but is in the range 0 - 31 (decimal) for file I/O
s1	reserved for internal system use
s2	reserved for internal system use, set to zero for Open (15), Make (22), Search (17,18) file functions.
rc	record count field, reserved for system use
d0...dn	filled in by CP/M, reserved for system use
cr	current record to be read or written; for a sequential read or write file operation, the program normally sets this field to zero to access the first record in the file
r0,r1,r2	optional, contain random record number in the range 0-3FFFFH; bytes r0, r1, and r2 are a 24-bit value with the most significant byte r0 and the least significant byte r2. Random I/O functions use the random record number in this field.

For users of other versions of CP/M, note that both CP/M-80 Version 2.2 and CP/M-68K perform directory operations in a reserved area of memory that does not affect the DMA buffer contents, except for the Search For First (17) and Search For Next (18) Functions in which the directory record is copied to the current DMA buffer.

4.2.2 File Processing Errors

When a program calls a BDOS function to process a file, an error condition can cause the BDOS to return one of five error messages to the console:

- CP/M Disk read error
- CP/M Disk write error
- CP/M Disk select error
- CP/M Disk change error
- CP/M Disk file error: ffffffff.ttt is read-only

Except for the CP/M Disk file error, CP/M-68K displays the error message at the console in the format:

```
"error message text" on drive x
```

The "error message text" is one of the error messages listed above. The variable x is a one-letter drive code that indicates the drive on which CP/M-68K detects the error. CP/M-68K displays the CP/M Disk file error in the format shown above.

When CP/M-68K detects one of these errors, the BDOS traps it. CP/M-68K displays a message indicating the error and, depending on the error, allows you to abort the program, retry the operation, or continue processing. Each of these errors and their options are described below.

CP/M issues a CP/M Disk read or write error when the BDOS receives a hardware error from the BIOS. The BDOS specifies BIOS read and write sector commands when the BDOS executes file-related system functions. If the BIOS read or write routine detects a hardware error, the BIOS returns an error code to the BDOS that results in CP/M-68K displaying a disk read or write error message at your console. In addition to the error message, CP/M-68K also displays the option message:

```
Do you want to Abort (A), Retry (R), or Continue with bad data (C)?
```

In response to the option message, you type one of the letters enclosed in parentheses and a RETURN. Each of these options is described below.

Table 4-4. Read-Write Error Message Response Options

Option	Action
A	The A option or CTRL-C aborts the program and returns control to the CCP. CP/M-68K returns the system prompt (>) preceded by the drive code.
R	The R option retries the operation that caused the error. For example, it rereads or rewrites the sector. If the operation succeeds, program execution continues as if no error occurred. However, if the operation fails, the error message and option message is displayed again.
C	The C option ignores the error that occurred and continues program execution. The C option is not an appropriate response for all types of programs. Program execution should not be continued in some cases. For example, if you are updating a data base and receive a read or write 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. Using an editor, add the data that was not transferred due to the bad sector.

Any response other than an A, R, C, or CTRL-C is invalid. The BDOS reissues the option message if you enter any other response.

The CP/M Disk select error occurs when you select a disk but you receive an error due to one of the conditions below.

- You specified a disk drive not supported by the BIOS.
- The BDOS receives an error from the BIOS.
- You specified a disk drive outside the range A through P.

Before the BDOS issues a read or write function to the BIOS, the BDOS issues a disk select function to the BIOS. If the BIOS does not support the drive specified in the function, or if an error occurs, the BIOS returns an error to the BDOS, which in turn, causes CP/M-68K to display the disk select error at your console. If the error is caused by a BIOS error, CP/M-68K returns the option message:

```
     Do you want to Abort (A) or Retry (R)?
```

To select one of the options in the message, specify one of the letters enclosed in parentheses. The A option terminates the program and returns control to the CCP. The R option tries to select the disk again. If the disk select function fails, CP/M-68K redisplays the disk select error message and the option message.

However, if the error is caused because you specify a disk drive outside the range A through P, only the CP/M Disk select error is displayed. CP/M-68K aborts the program and returns control to the CCP.

Your console displays the CP/M Disk change error message when the BDOS detects the disk in the drive is not the same disk that was logged in previously. Your program cannot recover from this error. Your program terminates. CP/M-68K returns program control to the CCP.

You log in a disk by accessing the disk or resetting the disk or disk system. The Select Disk Function (14) resets a disk. The Reset Disk System Function (13) resets the disk system. Files cannot be open when your program invokes either of these functions.

You receive the CP/M Disk file error and option messages (shown below) if you call the BDOS to write to a file that is set to read-only status. Either a STAT command or the BDOS Set File Attributes Function (30) sets a file to read-only status.

```
CP/M Disk file error:  ffffffff.ttt is read-only.

Do you want to: Change it to read/write (C), or Abort (A)?
```

The variable ffffffff.ttt in the error message denotes the filename and filetype. To select one of the options, specify one of the letters enclosed in parentheses. Each option is described below.

Table 4-5. Disk File Error Response Options

Option	Action
C	Changes the status of this file from read-only to read-write and continues executing the program that was being processed when this error occurred.
A	Terminates execution of the program that was being processed and returns program control to the CCP. The status of the file remains read-only. If you enter a CTRL-C, it has the same effect as specifying the A option.

CP/M-68K reprompts with the option message if you enter any response other than those described above.

4.2.3 Open File Function

FUNCTION 15: OPEN FILE	
Entry Parameters:	
Register D0.W:	0FH
Register D1.L:	FCB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H - 03G
	error: FFH

The Open File Function matches the filename and filetype fields of the FCB specified in register D1.L with these fields of a directory entry for an existing file on the disk. When a match occurs, the BDOS sets the FCB extent (ex) field and the second system (S2) field to zero before the BDOS opens the file. Setting these one-byte fields to zero opens the file at the base extent, the first extent in the file. In CP/M-68K, files can be opened only at the base extent. In addition, the physical I/O mapping information, which allows access to the disk file through subsequent read and write operations, is copied to fields d0 through dn of the FCB. A file cannot be accessed until it has been opened successfully. The open function returns an integer value ranging from 00H through 03H in D0.W when the open operation is successful. The value FFH is returned in register D0.W when the file cannot be found.

The question mark (?) wildcard can be specified for the filename and filetype fields of the FCB referenced by register D1.L. The ? wildcard has the value 3FH. For each position containing a ? wildcard, any character constitutes a match. For example, if the filename and filetype fields of the FCB referenced by D1.L contain only ? wildcards, the BDOS accesses the first directory entry. However, you should not create a FCB of all wildcards for this function because you cannot ensure which file this function opens.

Note that the current record field (cr) in the FCB must be set to zero by the program for the first record in the file to be accessed by subsequent sequential I/O functions. However, setting the current record field to zero is not required to open the file.

4.2.4 Close File Function

FUNCTION 16: CLOSE FILE		
Entry Parameters:		
Register D0.W:	10H	
Register D1.L:	FCB Address	
Returned Values:		
Register D0.W:	Return Code	
	success:	00H - 03H
	error:	FFH

The Close File Function performs the inverse of the Open File Function. When the FCB passed in D1.L was opened previously by either an Open File (15) or Make File (22) Function, the close function updates the FCB in the disk directory. The process used to match the FCB with the directory entry is identical to the Open File Function (15). An integer value ranging from 00H though 03H is returned in D0.W for a successful close operation. The value FFH is returned in D0.W when the file cannot be found in the directory. When only read functions access a file, closing the file is not required. However, a file must be closed to update its disk directory entry when write functions access the file.

4.2.5 Search For First Function

FUNCTION 17: SEARCH FOR FIRST		
Entry Parameters:		
Register D0.W:	11H	
Register D1.L:	FCB Address	
Returned Values:		
Register D0.W:	Return Code	
	success:	00H - 03H
	error:	FFH

The Search For First Function scans the disk directory allocated to the current user number to match the filename and filetype of the FCB addressed in register D1.L with the filename and filetype of a directory entry. The value FFH is returned in register D0.W when a matching directory entry cannot be found. An integer value ranging from 00H through 03H is returned in register D0.W when a matching directory entry is found.

The directory record containing the matching entry is copied to the buffer at the current DMA address. Each directory record contains four directory entries of 32 bytes each. The integer value returned in D0.W indexes the relative location of the matching directory entry within the directory record. For example, the value 01H indicates that the matching directory entry is the second one in the directory record in the buffer. The relative starting position of the directory entry within the buffer is computed by multiplying the value in D0.W by 32 (decimal), which is equivalent to shifting the binary value of D0.W left 5 bits.

When the drive (dr) field contains a ? wildcard, the auto disk select function is disabled and the default disk is searched. All entries including empty entries for all user numbers in the directory are searched. The search function returns any matching entry, allocated or free, that belongs to any user number. An allocated directory entry contains the filename and filetype of an existing file. A free entry is not assigned to an existing file. If the first byte of the directory entry is E5H, the entry is free. A free entry is not always empty. It can contain the filename and filetype of a deleted file because the directory entry for a deleted file is not zeroed.

4.2.6 Search For Next Function

FUNCTION 18: SEARCH FOR NEXT		
Entry Parameters:		
Register D0.W:	12H	
Returned Values:		
Register D0.W:	Return Code	
	success:	00H - 03H
	error:	FFH

The Search For Next Function scans the disk directory for an entry that matches the FCB and follows the last matched entry, found with this or the Search For First Function (17).

A program must invoke a Search For First Function before invoking this function for the first time. Subsequent Search For Next Functions can follow, but they must be specified without other disk related BDOS functions intervening. Therefore, a Search For Next Function must follow either itself or a Search For First Function.

The Search For Next Function returns the value FFH in D0.W when no more directory entries match.

4.2.7 Delete File Function

FUNCTION 19: DELETE FILE		
Entry Parameters:		
Register D0.W:	13H	
Register D1.L:	FCB Address	
Returned Values:		
Register D0.W:	Return Code	
	success:	00H
	error:	FFH

The Delete File Function removes files and deallocates the directory entries for and space allocated to files that match the filename in the FCB pointed to by the address passed in D1.L. The filename and filetype can contain wildcards, but the drive select code cannot be a wildcard as in the Search For First (17) and Search For Next (18) Functions. The value FFH is returned in register D0.W when the referenced file cannot be found. The value 00H is returned in D0.W when the file is found.

4.2.8 Read Sequential Function

FUNCTION 20: READ SEQUENTIAL	
Entry Parameters:	
Register D0.W:	14H
Register D1.L:	FCB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H
	error: 01H

The Read Sequential Function reads the next 128-byte record in a file. The FCB passed in register D1.L must have been opened by an Open File (15) or the Make File Function (22) before this function is invoked. The program must set the current record field to zero following the open or make function to ensure the file is read from the first record in the file. After the file is opened, the Read Sequential Function reads the 128-byte record specified by the current record field from the disk file to the current DMA buffer. The FCB current record (cr) and extent (ex) fields indicate the location of the record that is read. The current record field is automatically incremented to the next record in the extent after a read operation.

When the current record field overflows, the next logical extent is automatically opened and the current record field is reset to zero before the read operation is performed. After the first record in the new extent is read, the current record field contains the value 01H.

The value 00H is returned in register D0.W when the read operation is successful. The value of 01H is returned in D0.W when the record being read contains no data. Normally, the no data situation is encountered at the end of a file. However, it can also occur when this function tries to read either a previously unwritten data block or a nonexistent extent. These situations usually occur with files created or appended with the BDOS Write Random Function (34).

4.2.9 Write Sequential Function

FUNCTION 21: WRITE SEQUENTIAL	
Entry Parameters:	
Register D0.W:	15H
Register D1.L:	FCB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H
	error: 01H or 02H

The Write Sequential Function writes a 128-byte record from the DMA buffer to the disk file whose FCB address is passed in register D1.L. The FCB must be opened by either an Open File (15) or Make File (22) Function before your program invokes the Write Sequential Function. The record is written to the current record, specified in the FCB current record (cr) field.

The current record field is automatically incremented to the next record. When the current record field overflows, the next logical extent of the file is automatically opened and the current record field is reset to zero before the write operation. After the write operation, the current record field in the newly opened extent is set to 01H.

Records can be written to an existing file. However, newly written records can overlay existing records in the file because the current record field usually is set to zero after a file is opened or created to ensure a subsequent sequential I/O function accesses the first record in the file.

The value 00H is returned in register D0.W when the write operation is successful. A nonzero value in register D0.W indicates the write operation is unsuccessful due to one of the conditions described below.

Table 4-6. Unsuccessful Write Operation Return Codes

Value	Meaning
01	No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.
02	No available data block - This condition is encountered when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

4.2.10 Make File Function

```
FUNCTION 22:  MAKE FILE

Entry Parameters:
   Register D0.W:  16H
   Register D1.L:  FCB Address

Returned  Values:
   Register D0.W:  Return Code

                   success:  00H - 03H
                     error:  FFH
```

The Make File Function creates and opens a new file on a specified disk or the default disk. The address of the FCB for the file is passed in register D1.L. You must ensure the FCB contains a filename that does not already exist in the referenced disk directory. The drive field (dr) in the FCB indicates the drive on which the directory resides. The disk directory is on the default drive when the FCB drive field contains a zero.

The BDOS creates the file and initializes the directory and the FCB in memory to indicate an empty file. The program must ensure that no duplicate filenames occur. Invoking the Delete File Function (19) prior to the Make File Function excludes the possibility of duplicate filenames.

Register D0.W contains an integer value in the range 00H through 03H when the function is successful. Register D0.W contains the value FFH when a file cannot be created due to insufficient directory space.

4.2.11 Rename File Function

FUNCTION 23: RENAME FILE	
Entry Parameters:	
Register D0.W:	17H
Register D1.L:	FCB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H
	error: FFH

The Rename File Function uses the FCB specified in register D1.L to change the filename and filetype of all directory entries for a file. The first 12 bytes of the FCB contains the file specification for the file to be renamed as shown in Figure 4-1. Bytes 16 through 27 (d0 through d12) contain the new name of the file. The filenames and filetypes specified must be valid for CP/M. Wildcards cannot be specified in the filename and filetype fields. The FCB drive field (dr) at byte position 0 selects the drive. This function ignores the drive field at byte position 16, if it is specified for the new filename. Register D0.W contains the value zero when the rename function is successful. It contains the value FFH when the first filename in the FCB cannot be found during the directory scan.

FCB byte position

0	1	2	3	4	5	6	7	8	9	10	11...	...16	17	18	19	20	21	22	23...	...27	...
dr	f1	f2	f3	f4	f5	f6	f7	f8	t1	t2	t3...	...d0	d1	d2	d3	d4	d5	d6	d7...	...d12	...

old file specification new file specification

Figure 4-1. FCB Format for Rename Function

In the above figure, horizontal ellipses indicate FCB fields that are not required for this function. Refer to Section 4.1.2 for a description of all FCB fields.

4.2.12 Set Direct Memory Access (DMA) Address Function

FUNCTION 26: SET DMA ADDRESS	
Entry Parameters:	
Register D0.W:	1AH
Register D1.L:	DMA Address
Returned Values:	
Register D0.W:	00H

The Set DMA Address Function sets the starting address of the 128-byte DMA buffer. DMA is an acronym for Direct Memory Access, which often refers to disk controllers that directly access memory to transfer data to and from the disk subsystem. Many computer systems use nonDMA access in which the data is transferred through programmed I/O operations. In CP/M the term DMA is used differently. The DMA address in CP/M-68K is the beginning address of a 128-byte data buffer, called the DMA buffer. The DMA buffer is the area in memory where a data record resides before a disk write operation and after a disk read operation. The DMA buffer can begin on an even or odd address.

4.2.13 Set File Attributes Function

```
FUNCTION 30:  SET FILE ATTRIBUTES

Entry Parameters:
   Register D0.W:  1EH
   Register D1.L:  FCB Address

Returned  Values:
   Register D0.W:  Return Code

                   success:  00H
                     error:  FFH
```

The Set File Attributes Function sets or resets file attributes supported by CP/M-68K and user defined attributes for application programs. CP/M-68K supports read-only, system, and archive attributes.

The high bit of each character in the ASCII filename (f1 through f8) and filetype (t1 through t3) fields in the FCB denotes whether attributes are set. When the high bit in any of these fields has the value 1, the attribute is set. Table 4-7 denotes the FCB fields and their attributes.

The address of the FCB is passed in register D1.L. Wildcards cannot be specified in the filename and filetype fields.

This function searches the directory on the disk drive, specified in the FCB drive field (dr), for directory entries that match the FCB filename and filetype fields. All matching directory entries are updated with the attributes this function sets.

A zero is returned in register D0.W when the attributes are set. However, if a matching entry cannot be found, register D0.W contains FFH.

Table 4-7. File Attributes

Field	Attribute
f1 through f4	User-defined attributes for application programs.
f5 through f8	Reserved for future use by CP/M-68K.
t1	The Read-Only attribute indicates the file status is Read-Only. The BDOS does not allow write commands to write to a file whose status is Read-Only. The BDOS does not permit a Read-Only file to be deleted.
t2	The System attribute indicates the file is a system file. Some built-in commands and system utilities differentiate between system and user files. For example, the DIRS command provides a directory of system files. The DIR command provides a directory of user files for the current user number. For details on these commands, refer to the <u>CP/M-68K Operating System User's Guide</u>.
t3	The Archive attribute is reserved but not used by CP/M-68K. If set, it indicates that the file has been written to backup storage by a user-written archive program. To implement this facility, the archive program sets this attribute when it copies a file to backup storage; any programs updating or creating files reset this attribute. The archive program backs up only those files that have the Archive attribute reset. Thus, an automatic backup facility restricted to modified files can be implemented easily.

The Open File (15) and Close File (16) Functions do not use the high bit in the filename and filetype fields when matching filenames. However, the high bits in these fields should equal zero when you open a file. Also, the Close File Function does not update the attributes in the directory entries when it closes a file.

4.2.14 Read Random Function

```
              FUNCTION 33:  READ RANDOM

    Entry Parameters:
       Register D0.W:  21H
       Register D1.L:  FCB Address

    Returned  Values:
       Register D0.W:  Return Code

                       success:  00H
                         error:  01H, 03H
                                 04H, 06H
```

The Read Random Function reads records randomly, rather than sequentially. The file must be opened with an Open File Function (15) or a Make File Function (22) before this function is invoked. The address of a 36-byte FCB is passed in register D1.L. The FCB random record field denotes the record this function reads. The random record field is a 24-bit field, with a value ranging from 00000H through 3FFFFH. This field spans bytes r0, r1, and r2 which are bytes 33 through 35 of the FCB. The most significant byte is first, r0, and the least significant byte, r2, is last. This byte sequence is consistent with the addressing conventions for the 68000 microprocessor but differs from other versions of CP/M.

The random record number must be stored in the FCB random record field before the BDOS is called to read the record. After reading the record, register D0.W either contains an error code (see Table 4-8), or the value 00H which indicates the read operation was successful. In the latter case, the current DMA buffer contains the randomly accessed record. The record number is not incremented. The FCB extent and current record fields are updated to correspond to the location of the random record that was read. A subsequent Read Sequential (20) or Write Sequential (21) Function starts from the record which was randomly accessed. Therefore, the randomly read record is reread when a program switches from randomly reading records to sequentially reading records. This is also true for the Write Random Functions (34, 40). The last record written is rewritten if the program switches from randomly writing records to sequentially writing records with the Write Sequential Function (21). However, a program can obtain the effect of sequential I/O operations by incrementing the random record field following each Read Random Function (33) or Write Random Function (34, 40).

Numeric codes returned in register D0.W following a random read operation are listed in Table 4-8.

Table 4-8. Read Random Function Return Codes

Code	Meaning
00	Success - returned in D0.W when the Read Random Function succeeds.
01	Reading unwritten data - returned when a random read operation accesses a previously unwritten data block.
03	Cannot close current extent - returned when the BDOS cannot close the current extent prior to moving to the new extent containing the FCB random record number. This error can be caused by an overwritten FCB or a read random operation on an FCB that has not been opened.
04	Seek to unwritten extent - returned when a random read operation accesses a nonexistent extent. This error situation is equivalent to error 01.
06	Random record number out of range - returned when the value of the FCB random record field is greater than 3FFFFH.

4.2.15 Write Random Function

FUNCTION 34: WRITE RANDOM	
Entry Parameters:	
Register D0.W:	22H
Register D1.L:	FCB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H
	error: 02H, 03H
	05H, 06H

The Write Random Function writes a 128-byte record from the current DMA address to the disk file that matches the FCB referenced in register D1.L. Before this function is invoked, the file must be opened with either the Open File Function (15) or the Make File Function (22).

This function requires a 36-byte FCB. The last three bytes of the FCB contain the random record field. It contains the record number of the record that is written to the file. To append to an existing file, the Compute File Size Function (35) can be used to write the random record number to the FCB random record field. For a new file, created with the Make File Function (22), you do not need to use the Compute File Size Function to write the first record in the newly created file. Instead, specify the value 00H in the FCB random record field. The first record written to the newly created file is zero.

When an extent or data block must be allocated for the record, the Write Random Function allocates it before writing the record to the disk file. The random record number is not changed following a Write Random Function. Therefore, a new random record number must be written to the FCB random record field before each Write random Function is invoked.

However, the logical extent number and current record field of the FCB are updated and correspond to the random record number that is written. Thus, a Read Sequential (20) or Write Sequential (21) Function that follows a Write Random Function, either rereads or rewrites the record that was accessed by the Read or Write Random Function. To avoid overwriting the previously written record and simulate sequential write functions, increment the random record number after each Write Random Function.

After the random write function completes, register D0.W contains either an error code (see Table 4-9), or the value 00H that indicates the operation was successful.

Table 4-9. Write Random Function Return Codes

Code	Meaning
00	Success - returned when the Write Random Function succeeds without error.
02	No available data block - occurs when the Write Random function attempts to allocate a new data block to the file, but the selected disk does not contain any unallocated data blocks.
03	Cannot close current extent - occurs when the BDOS cannot close the current extent prior to moving to the new extent that contains the record specified by the FCB random record field. This error can be caused by an overwritten FCB or a write random operation on an FCB that has not been opened.
05	No available directory space - occurs when the write function attempts to create a new extent that requires a new directory entry but the selected disk drive does not have any available directory entries.
06	Random record number out of range - returned when the value of the FCB random record field is greater than 3FFFFH.

4.2.16 Compute File Size Function

FUNCTION 35: COMPUTE FILE SIZE		
Entry Parameters:		
Register D0.W:	23H	
Register D1.L:	FCB Address	
Returned Values:		
Register D0.W:	00H	
	success:	File Size written to FCB random Record Field
	error:	Zero written to FCB Random Record Field

The Compute File Size Function computes the size of a file and writes it to the random record field of the 36-byte FCB whose address is passed in register D1.L.

The FCB filename and filetype are used to scan the directory for an entry with a matching filename and filetype. If a match cannot be found, the value zero is written to the FCB random record field. However, when a match occurs, the virtual file size is written in the FCB random record field.

The virtual file size is the record number of the record following the end of the file. The virtual size of a file corresponds to the physical size when the file is written sequentially. However, the virtual file size may not equal the physical file size when the records in the file were created by random write functions. The Compute File Size Function computes the file size by adding the value 1 to the record number of last record in a file. However, for files that contain randomly written records, the record number of the last record does not necessarily indicate the number of records in a file. For example, the number of the last record in a sparse file does not denote the number of records in the file. Record numbers for sparse files are not usually sequential. Therefore, gaps can exist in the record numbering sequence. You can create sparse files with the Write Random Functions (34 and 40).

In addition to computing the file size, you can use this function to determine the end of an existing file. For example, when you append data to a file, this function writes the record number of the first unwritten record to the FCB random record field. When you use the Write Random (34) or the Write Random With Zero Fill (40) Function, your program more efficiently appends data to the file because the FCB already contains the appropriate record number.

4.2.17 Set Random Record Function

FUNCTION 36: SET RANDOM RECORD	
Entry Parameters:	
Register D0.L:	24H
Register D1.L:	FCB Address
Returned Values:	
Register D0:	00H
Register FCB:	Random Record Field Set

The Set Random Record Function calculates the random record number of the current position in the file. The current position in the file is defined by the last operation performed on the file. Table 4-10 lists the current position relative to operations performed on the file.

Table 4-10. Current Position Definitions

Operation	Function	Current Position
Open file	Open File (15)	record 0
Create file	Make File (22)	record 0
Random read	Read Random (33)	last record read
Random write	Write Random (34) Write Random With Zero Fill (40)	last record written
Sequential read	Read Sequential (20)	record following the last record read
Sequential write	Write Sequential (21)	record following the last record written

This function writes the random record number in the random record field of the 36-byte FCB whose address your program passes in register D1.L.

You can use this function to set the random record field of the next record your program accesses when it switches from accessing records sequentially to accessing them randomly. For example, your program sequentially reads or writes 128-byte data records to an arbitrary position in the file that is defined by your program. Your program then invokes this function to set the random record field in the FCB. The next random read or write operation that your program performs accesses the next record in the file.

Another application for this function is to create a key list from a file that you read sequentially. Your program sequentially reads and scans a file to extract the positions of key fields. After your program locates each key, it calls this function to compute the random record position for the record following the record containing the key. To obtain the random record number of the record containing the key, subtract one from the random record number that this function calculates. CP/M-68K reads and writes 128-byte records. If your record size is also 128 bytes, your program can insert the record position minus one into a table with the key for later retrieval. By using the random record number stored in the table when your program performs a random read or write operation, your program locates the desired record more efficiently.

Note that if your data records are not equal to 128 bytes, your program must store the random record number and an offset into the physical record. For example, you must generalize this scheme for variable-length records. To find the starting position of key records, your program stores the buffer-relative position and the random record number of the records containing keys.

4.2.18 Write Random With Zero Fill Function

FUNCTION 40: WRITE RANDOM WITH ZERO FILL		
Entry Parameters:		
Register D0.W:	28H	
Register D1.L:	FCB Address	
Returned Values:		
Register D0.W:	Return Code	
	success:	00H
	error:	02H, 03H
		05H, 06H

The Write Random With Zero Fill Function, like the Random Write Function (34), writes a 128-byte record from the current DMA buffer to the disk file. The address of a 36-byte FCB is passed in register D1.L. The last three bytes contain the FCB random record field. This field specifies the record number of the record that this write random function writes to the file. Refer to Write Random Function (34) for details on the FCB and setting its random record field.

Like the Write Random Function, this function allocates a data block before writing the record when a block is not already allocated. However, in addition to allocating the data block, this function also initializes the block with zeroes before writing the record. If your program uses this function to write random records to files, it ensures that the contents of unwritten records in the block are predictable.

After the random write function completes, register D0.W contains either an error code (see Table 4-9), or the value 00H, which indicates the operation was successful.

4.3 Drive Functions

This section describes drive functions that reset the disk system, select and write-protect disks, and return vectors. They include the functions listed in Table 4-11.

Table 4-11. Drive Functions

Function	Function Number
Reset Disk System	13
Select Disk	14
Return Login Vector	24
Return Current Disk	25
Write Protect Disk	28
Get Read-Only Vector	29
Get Disk Parameters	31
Reset Drive	37
Get Disk Free Space	46

4.3.1 Reset Disk System Function

FUNCTION 13: RESET DISK SYSTEM
Entry Parameters: Register D0.W: 0DH
Returned Values: Register D0.W: 00H

The Reset Disk System Function restores the file system to a reset state. All disks are set to read-write (see Write Protect Disk (28) and Get Read-Only Vector (29) Functions), and all the disk drives are logged out. This function can be used by an application program that requires disk changes during operation. The Reset Drive Function (37) can also be used for this purpose. All files must be closed before your program invokes this function.

4.3.2 Select Disk Function

FUNCTION 14: SELECT DISK	
Entry Parameters:	
Register D0.W:	0EH
Register D1.W:	Disk Number
Returned Values:	
Register D0.W:	00H

The Select Disk Function designates the disk drive specified in register D1.W as the default disk for subsequent file operations. The decimal numbers 0 through 15 correspond to drives A through P. For example, D1.W contains a 0 for drive A, a 1 for drive B, and so forth through 15 for a full 16-drive system. In addition, the designated drive is logged-in if it is currently in the reset state. Logging in a drive places it in an on-line status which activates the drive's directory until the next cold start, or Reset Disk System (13) or Reset Drive (37) Function.

When the FCB drive code equals zero (dr = 0H), this function references the currently selected drive. However, when the FCB drive code value is between 1 and 16, this function references drives A through P.

If this function fails, CP/M-68K returns a CP/M Disk select error, which is described in Section 4.2.2.

4.3.3 Return Login Vector Function

FUNCTION 24: RETURN LOGIN VECTOR	
Entry Parameters:	
Register D0.W:	18H
Returned Values:	
Register D0.W:	Login Vector

The Return Login Vector Function returns in register D0.W a 16-bit value that denotes the log-in status of the drives. The least significant bit corresponds to the first drive A, and the high order bit corresponds to the sixteenth drive, labeled P. Each bit has a value of zero or one. The value zero indicates the drive is not on-line. The value one denotes the drive is on-line. When a drive is logged in, its bit in the log-in vector has a value of one. Explicitly or implicitly logging in a drive sets its bit in the log-in vector. The Select Disk Function (14) explicitly logs in a drive. File operations implicitly log in a drive when the FCB drive field (dr) contains a nonzero value.

4.3.4 Return Current Disk Function

FUNCTION 25: RETURN CURRENT DISK	
Entry Parameters:	
Register D0.W:	19H
Returned Values:	
Register D0.W:	Current Disk

The Return Current Disk Function returns the current default disk number in register D0.W. The disk numbers range from 0 through 15, which correspond to drives A through P. Note that this numbering convention differs from the FCB drive field, which specifies integers 1 through 16 correspond to drives labeled A through P.

4.3.5 Write Protect Disk Function

FUNCTION 28: WRITE PROTECT DISK
Entry Parameters: Register D0.W: 1CH
Returned Values: Register D0.W: 00H

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

```
Disk change error on drive x
```

Your program terminates when this error occurs. Program control returns to the CCP.

4.3.6 Get Read-Only Vector Function

FUNCTION 29: GET READ-ONLY VECTOR	
Entry Parameters:	
Register D0.W:	1DH
Returned Values:	
Register D0.W:	Read-Only Vector Value

The Get Read-Only Vector Function returns a 16-bit vector in register D0.W. The vector denotes drives that have the temporary read-only bit set. Similar to the Return Login Vector Function (24), the least significant bit corresponds to drive A, and the most significant bit corresponds to drive P. The Read-Only bit is set either by an explicit call to the Write Protect Disk Function (28), or by the automatic software mechanisms within CP/M-68K that detect changed disks.

4.3.7 Get Disk Parameters Function

FUNCTION 31: GET DISK PARAMETERS	
Entry Parameters:	
Register D0.W:	1FH
Register D1.L:	CDPB Address
Returned Values:	
Register D0.W:	00H
Register CDPB:	Contains DPB Values

The Get Disk Parameters Function writes a copy of the 16-byte BIOS Disk Parameter Block (DPB) for the current default disk, called the CDPB, at the address specified in register D1.L. Figure 4-2 illustrates the format of the DPB and CDPB. The values in the CDPB can be extracted and used for display and space computation purposes. Normally, application programs do not use this function. For more details on the BIOS DPB, refer to the CP/M-68K Operating System System Guide.

SPT	BSH	BLM	EXM	RES	DSM	DRM	RES	CKS	OFF
16	8	8	8	8	16	16	16	16	16

Figure 4-2. DPB and CDBP

Table 4-12 lists the fields in the DPB and CDPB.

Table 4-12. Fields in the DPB and CDPB

Field	Description
SPT	Number of 128-byte logical sectors per track
BSH	Block shift factor
BLM	Block mask
EXM	Extent mask
RES	Reserved byte
DSM	Total number of blocks on the disk
DRM	Total number of directory entries on the disk
RES	Reserved for system use
CKS	Length (in bytes) of the checksum vector
OFF	Track offset to disk directory

4.3.8 Reset Drive Function

FUNCTION 37: RESET DRIVE	
Entry Parameters:	
Register D0.W:	25H
Register D1.W:	Drive Vector
Returned Values:	
Register D0.W:	00H

The Reset Drive function restores specified drives to the reset state. A reset drive is not logged-in and its status is read-write. Register D1.W contains a 16-bit vector indicating the drives this function resets. The least significant bit corresponds to the first drive, A. The high order bit corresponds to the sixteenth drive, labeled P. Bit values of 1 indicate the drives this function resets.

To maintain compatibility with other Digital Research operating systems, this function returns the value zero in register D0.W.

4.3.9 Get Disk Free Space Function

FUNCTION 46: GET DISK FREE SPACE	
Entry Parameters:	
Register D0.W:	2EH
Register D1.W:	Disk Number
Returned Values:	
Register DMA Buffer:	Free Sector Count

The Get Free Disk Space Function returns the free sector count, the number of free 128-byte sectors on a specified drive, in the first four bytes of the current DMA buffer. The drive number is passed in register D1.W. CP/M-68K assigns disk numbers sequentially from 0 through 15 (decimal). Each number corresponds to a drive in the range A through P. For example, the disk number for drive A is 0 and for drive B, the number is 1.

Note that these numbers do not correspond to those in the drive field of the FCB. The FCB drive field (dr) uses the numbers 1 through 16 (decimal) to designate drives.

4.4 Character I/O Functions

Character I/O functions read or write characters serially to a peripheral device. Character I/O functions supported in CP/M-68K are described in this section and listed in Table 4-13.

Table 4-13. Character I/O Functions

Function	Function Number
Console Operations	
Console Input	1
Console Output	2
Direct Console I/O	6
Print String	9
Read Console Buffer	10
Get Console Status	11

Table 4-13. (continued)

Function	Function Number
Additional Serial I/O	
Auxiliary Input	3
Auxiliary Output	4
List Output	5
I/O Byte	
Get I/O Byte	7
Set I/O Byte	8

4.4.1 Console I/O Functions

This section describes functions that read from, write to, and report the status of the logical device CONSOLE.

Console Input Function

FUNCTION 1: CONSOLE INPUT
Entry Parameters: Register D0.W: 01H
Returned Values: Register D0.W: ASCII Character

The Console Input function reads the next character from the logical console device (CONSOLE) to register D0.W. Printable characters, along with carriage return, line feed, and backspace (CTRL-H), are echoed to the console. Tab characters (CTRL-I) are expanded into columns of eight characters. Other CONTROL characters, such as CTRL-C, are processed. The BDOS does not return to the calling program until a character has been typed. Thus, execution of the program is suspended until a character is ready.

Console Output Function

FUNCTION 2: CONSOLE OUTPUT	
Entry Parameters:	
Register D0.W:	02H
Register D1.W:	ASCII Character
Returned Values:	
Register D0:	00H

The ASCII character from D1.W is sent to the logical console. Tab characters expand into columns of eight characters. In addition, a check is made for stop scroll (CTRL-S), start scroll (CTRL-Q), and the printer switch (CTRL-P). This function also processes CTRL-C, which aborts the operation and warm boots the system. If the console is busy, execution of the calling program is suspended until the console accepts the character.

Direct Console I/O Function

FUNCTION 6: DIRECT CONSOLE I/O	
Entry Parameters:	
Register D0.W:	06H
Register D1.W:	0FFH (input) 0FEH (status) or Character (output)
Returned Values:	
Register D0.W:	Character or Status

Direct Console I/O is supported under CP/M-68K for those specialized applications where character-by-character console input and output are required without the control character functions CP/M-68K supports. This function bypasses all of CP/M-68K's normal CONTROL character functions such as CTRL-S, CTRL-Q, CTRL-P, and CTRL-C.

Upon entry to the Direct Console I/O Function, register D1.W contains one of the values listed below.

Table 4-14. Direct Console I/O Function Values

Value	Meaning
FFH	denotes a CONSOLE input request
FEH	denotes a CONSOLE status request
ASCII character	output to CONSOLE where CONSOLE is the logical console device

When the input value is FFH, the Direct Console I/O Function calls the BIOS Conin Function, which returns the next console input character in D0.W but does not echo the character on the console screen. The BIOS Conin function waits until it receives a character. Thus, execution of the calling program remains suspended until a character is ready.

When the input value is FEH, the Direct Console I/O Function returns the status of the console input in register D0.W. When register D0.W contains the value zero, no console input exists. However, when the value in D0.W is nozero, console input is ready to be read by the BIOS Conin Function.

When the input value in D1.W is neither FEH nor FFH, the Direct Console I/O Function assumes that D1.W contains a valid ASCII character, which is sent to the console.

Print String Function

FUNCTION 9: PRINT STRING	
Entry Parameters:	
Register D0.W:	09H
Register D1.L:	String Address
Returned Values:	
Register D0.W:	00H

The Print String function sends the character string stored in memory at the location given in register D1.L to the logical console device (CONSOLE) until a dollar sign ($) is encountered in the string. Tabs are expanded as in the Console Output Function (2), and checks are made for stop scroll (CTRL-S), start scroll (CTRL-Q), and the printer switch (CTRL-P).

Read Console Buffer Function

FUNCTION 10: READ CONSOLE BUFFER	
Entry Parameters:	
Register D0.W:	0AH
Register D1.L:	Buffer Address
Returned Values:	
Register D0.W:	00H
Register Buffer:	Character Count and Characters

The Read Buffer function reads a line of edited console input from the logical console device (CONSOLE) to a buffer address passed in register D1.L. Console input is terminated when the input buffer is filled, or, a RETURN (CTRL-M) or a line feed (CTRL-J) character is entered. The input buffer addressed by D1.L takes the form:

```
D1.L:  +0 +1 +2 +3 +4 +5 +6 +7 +8    . . .    +n

       mx nc c1 c2 c3 c4 c5 c6 c7    . . .    ??
```

The variable mx is the maximum number of characters the buffer holds. The variable nc is the total number of characters placed in the buffer. Your program must set the mx value prior to invoking this function. The mx value can range in value from 1 through 255 (decimal). The characters entered from the keyboard follow the nc value. The value nc is returned to the buffer. It can range from 0 to the value of mx. If the nc value is less than the mx value, uninitialized characters follow the last character. Uninitialized characters are denoted by the double question marks (??) in the above figure. A terminating RETURN or line feed character is not placed in the buffer and is not included in the total character count nc.

This function supports several editing control functions, which are briefly described in Table 4-15.

Table 4-15. Line Editing Controls

Keystroke	Result
RUB/DEL	removes and echoes the last character
CONTROL-C	reboots when it is the first character on a line
CONTROL-E	causes physical end-of-line
CONTROL-H	backspaces one character position
CONTROL-J	(line feed) terminates input line
CONTROL-M	(return) terminates input line
CONTROL-P	starts and stops the echoing of console output to the logical LIST device
CONTROL-Q	restarts console I/O after CTRL-S halts it
CONTROL-R	retypes the current line on the next line
CONTROL-S	halts console I/O and waits for CTRL-Q to restart it
CONTROL-U	echoes a pound sign (#) indicating ignore characters previously input on the current line before it positions the cursor on the next line
CONTROL-X	backspaces to beginning of current line

Certain functions that position the cursor to the leftmost position (for example, CONTROL-X) move the cursor to the column position where the cursor was prior to invoking the Read Console Buffer Function. This convention makes your data input and line correction more legible.

Get Console Status Function

FUNCTION 11: GET CONSOLE STATUS	
Entry Parameters:	
Register D0.W:	0BH
Returned Values:	
Register D0.W:	Console Status

The Get Console Status Function checks whether a character has been typed at the logical console device (CONSOLE). If a character is ready, a nonzero value is returned in register D0.W; otherwise the value 00H is returned in D0.W.

4.4.2 Additional Serial I/O Functions

This section describes additional serial I/O functions that read and write data to devices defined by I/O Byte Functions (7,8).

Auxiliary Input Function

FUNCTION 3: AUXILIARY INPUT
Entry Parameters: Register D0.W: 03H
Returned Values: Register D0.W: ASCII Character

The Auxiliary Input function reads the next character from the auxiliary input device into register D0.W. Execution of the calling program remains suspended until the character is read. This function assumes the BIOS implements its Auxiliary Input Function. When more than one auxiliary input port exists, the BIOS should implement the I/O Byte Function. For details on the BIOS Auxiliary Input and I/O Byte Functions, refer to the CP/M-68K Operating System System Guide.

Auxiliary Output Function

FUNCTION 4: AUXILIARY OUTPUT	
Entry Parameters:	
Register D0.W:	04H
Register D1.W:	ASCII Character
Returned Values:	
Register D0.W:	00H

The Auxiliary Output function sends a character from register D1.W to the auxiliary output device. Execution of the calling program remains suspended until the hardware buffer receives the output character. This function assumes the BIOS implements its Auxiliary Output Function. When more than one auxiliary output port exists, the BIOS should implement the I/O Byte Function. For details on the BIOS Auxiliary Output and I/O Byte Functions, refer to the CP/M-68K Operating System System Guide.

List Output Function

```
                FUNCTION 5:  LIST OUTPUT

            Entry Parameters:
               Register D0.W:   05H
               Register D1.W:   ASCII Character

            Returned  Values:
               Register D0.W:   00H
```

The List Output function sends the ASCII character in register D1.W to the logical list device (LIST).

4.4.3 I/O Byte Functions

This section describes the I/O Byte Functions. The I/O Byte is an 8-bit value that assigns physical devices, represented by 2-bit fields, to each of the logical devices: CONSOLE, AUXILIARY INPUT, AUXILIARY OUTPUT, and LIST as shown in Figure 4-3. The I/O Byte functions allow programs to access the I/O byte to determine its current value (Get I/O Byte) or to modify it (Set I/O Byte). These functions are valid only if the BIOS implements its I/O Byte Function. Refer to the CP/M-68K Operating System System Guide for details on implementing the I/O Byte Function.

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

Figure 4-3. I/O Byte

The value in each field ranges from 0-3. The value defines the assigned source or destination of each logical device, as shown in Table 4-16.

Table 4-16. I/O Byte Field Definitions

CONSOLE field (bits 1,0)	
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)	
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:)
AUXILIARY OUTPUT field (bits 5,4)	
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)	
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:)

The implementation of the BIOS I/O Byte Function is optional. PIP and STAT are the only CP/M-68K utilities that use the I/O Byte. PIP accesses physical devices. STAT designates and displays logical to physical device assignments. For details on implementing the I/O Byte Function, refer to the CP/M-68K Operating System System Guide.

Get I/O Byte Function

FUNCTION 7: GET I/O BYTE	
Entry Parameters:	
Register D0.W:	07H
Returned Values:	
Register D0.W:	I/O Byte Value

The Get I/O Byte Function returns the current value of I/O Byte in register D0.W. The I/O Byte contains the current assignments for the logical devices CONSOLE, AUXILIARY INPUT, AUXILIARY OUTPUT, and LIST. Note that this function is valid only if the BIOS implements its I/O Byte Function. Refer to the CP/M-68K Operating System System Guide for details on implementing the BIOS I/O Byte Function.

Set I/O Byte Function

FUNCTION 8: SET I/O BYTE	
Entry Parameters:	
Register D0.W:	08H
Register D1.W:	I/O Byte Value
Returned Values:	
Register D0.W:	00H

The Set I/O Byte Function changes the system I/O Byte value to the value passed in register D1.W. This function allows programs to modify the current assignments for the logical devices CONSOLE, AUXILIARY INPUT, AUXILIARY OUTPUT, and LIST in the I/O Byte. This function is valid only if the BIOS implements its I/O Byte Function. Refer to the CP/M-68K Operating System System Guide for details on implementing the I/O Byte Function.

4.5 System/Program Control Functions

The System and program control functions described in this section warm boot the system, return the operating system version number, call the Basic I/O System (BIOS) functions, and terminate and load programs. These functions are listed in Table 4-17.

Table 4-17. System and Program Control Functions

Function	Function Number
System Reset	0
Return Version Number	12
Set/Get User Code	32
Chain to Program	47
Flush Buffers	48
Direct BIOS Call	50
Program Load	59

4.5.1 System Reset Function

FUNCTION 0:	SYSTEM RESET
Entry Parameters: Register D0.W:	00H
Returned Values:	Function Does Not Return to Calling Program

The System Reset Function terminates the current program and returns program control to the CCP command level.

4.5.2 Return Version Number Function

FUNCTION 12: RETURN VERSION NUMBER
Entry Parameters: Register D0.W: 0CH
Returned Values: Register D0.W: Version Number

The Return Version Number Function provides information that allows version dependent programming. The one-word value 2022H is the version number returned in register D0.W for Release 1.1 of CP/M-68K. Table 4-18 lists the version numbers this function returns for Digital Research operating systems.

Table 4-18. Version Numbers

Operating System	Version	Version Number
CP/M-68K	1.1	2022H
CP/M-80	1.4	0014H
CP/M-80	2.2	0022H
CP/M-80	3.0	0031H
MP/M-80™	1.1	0122H
MP/M-80	2.0	0130H
MP/M-80	2.1	0130H
CP/M-86	1.0	1022H
CP/M-86	1.1	1022H
MP/M-86™	2.0	1130H
MP/M-86	2.1	1130H
Concurrent CP/M-86™ (for the IBM Personal Computer)	1.0	1430H
Concurrent CP/M-86	2.0	1431H

Add the hexadecimal value 0200 to any version number when the system implements CP/NET® For example, CP/M-80 Release 2.2 returns the version 0222H when the system implements CP/NET.

4.5.3 Set/Get User Code

FUNCTION 32: SET/GET USER CODE	
Entry Parameters:	
Register D0.W:	20H
Register D1.W:	FFH (get) or User Code (set)
Returned Values:	
Register D0.W:	Current User Number

An application program can change or obtain the currently active user number by calling the Set/Get User Code Function. If the value in register D1.W is FFH, the value of the current user number is returned in register D0.W. The value ranges from 0 to 15 (decimal). If register D1.W contains a value in the range 0 through 15 (decimal), the current user number is changed to the value in register D1.W. When the program terminates and control returns to the CCP, the user number reverts to the BDOS default user number. The BDOS assumes the default is zero unless you explicitly specify the USER command to set an alternate default.

4.5.4 Chain To Program Function

FUNCTION 47: CHAIN TO PROGRAM	
Entry Parameters:	
Register D0.W:	2FH
Returned Values:	
Register D0.W:	Function Does Not Return to Calling Program

The Chain to Program Function terminates the current program and executes the command line stored in the current DMA buffer. The format of the command line consists of a one-byte character count (N), the command line characters, and a null byte as shown in Figure 4-4. The character count contains the number of characters in the command line. The count must be no more than 126 characters. If an error occurs, you receive one of the CCP errors described in Appendix E.

N	Command Line (N characters)	0
1 byte	N bytes $\leq$ 126 bytes	1 byte

Figure 4-4. Command Line Format in the DMA Buffer

4.5.5 Flush Buffers Function

FUNCTION 48: FLUSH BUFFERS	
Entry Parameters:	
Register D0.W:	30H
Returned Values:	
Register D0.W:	Return Code
	success: 00H
	error: nonzero value

The Flush Buffers Function calls a BIOS Flush Buffers Function (21), which forces the system to write the contents of any unwritten or modified disk buffers to the appropriate disks. Control and editing applications use this function to ensure data is periodically physically written to the appropriate disks. When the buffers are successfully flushed, this function returns the value 00H in register D0.W. However, if an error occurs, and this function does not complete successfully, this function returns a nonzero value in register D0.W.

4.5.6 Direct BIOS Call Function

FUNCTION 50: DIRECT BIOS CALL	
Entry Parameters:	
Register D0.W:	32H
Register D1.L:	BPB Address
Returned Values:	
Register D0.L:	BIOS Return Code (if any)

Function 50 allows a program to call a BIOS function and transfers control through the BDOS to the BIOS. The D1.L register contains the address of the BIOS Parameter Block (BPB), a 5-word memory area containing two BIOS function parameters, P1 and P2, as shown in Figure 4-5. When a BIOS function returns a value, it is returned in register D0.L.

Like other BDOS functions, your program must specify a Trap 2 Instruction to invoke this BDOS function after the registers are loaded with the appropriate parameters. The starting location of the BPB must be an even-numbered address.

Field	Size
Function Number	1 word
Value P1	1 longword
Value P2	1 longword

Figure 4-5. BIOS Parameter Block (BPB)

In the above figure, the function number is a BIOS function number. See Appendix A. The two values, P1 and P2, are 32-bit BIOS parameters, which are passed in registers D1.L and D2.L before your program invokes the BIOS function. Appendix A contains a list of BIOS functions. For more details on BIOS functions, refer to the CP/M-68K Operating System System Guide.

4.5.7 Program Load Function

FUNCTION 59: PROGRAM LOAD	
Entry Parameters:	
Register D0.W:	3bH
Register D1.L:	LPB
Returned Values:	
Register D0.W:	Return Code
success:	00H
error:	01H - 03H

The Program Load function loads an executable command file into memory. In addition to the function code, passed in register D0.W, the address of the Load Parameter Block (LPB) is passed in register D1.L. After a program is loaded, the BDOS returns one of the return codes listed below in register D0.W.

Table 4-19. Program Load Function Return Codes

Code	Meaning
00	the function is successful
01	insufficient memory exists to load the file or the header is bad
02	a read error occurs while the file is loaded in memory
03	bad relocation bits exist in the program file

The LPB describes the program and denotes the address at which it is loaded. The format of the LPB is outlined in Figure 4-6. The starting location of the LPB must be an even-numbered address.

[illegible]

Byte Offset	Content	Size
0H	address of FCB of successfully opened program file	1 longword
4H	lowest address of area in which to load program	1 longword
8H	highest address of area in which to load program +1	1 longword
CH	address of base page (returned by BDOS)	1 longword
10H	default user stack pointer (returned by BDOS)	1 longword
14H	loader control flags	1 word

Figure 4-6. Format of the Load Parameter Block (LPB)

Before a program specifies the Program Load function, the file must be opened with an Open File Function (15). The memory addresses specified for the program in the LPB must lie within the TPA. When the CCP calls the Program Load function to load a transient program, the LPB addresses are the boundaries of the TPA.

The loader control flags in the LPB select loader options as shown in Table 4-20.

Table 4-20. Load Parameter Block Options

Bit Number	Value	Meaning
0 (least significant byte)	0	load program in the lowest possible part of the supplied address space
	1	load program in the highest possible part of the supplied address space
1 - 15 (decimal)	0	Reserved, should be set to zero.

The CCP uses the Program Load Function to load a command file. The CCP places a return address to itself on the top of the stack for the program being loaded. The program can exit and return to the CCP by performing a Return from Subroutine (RTS) instruction. However, if the program modifies the stack, it must reset the top of the stack to point to the CCP address before the program executes a RTS instruction. The CCP also places the base page address on the program's stack. The base page address is located four bytes from the address pointed to by register A7, the stack pointer. The assembly language notation for this offset is 4(A7) or 4(sp). The format of the base page is outlined in Appendix C.

The BDOS allocates memory for the base page within the limits set by the low and high addresses in the LPB and returns the address of the allocated base page in the LPB. Locations 0000H - 0024H of the base page are initialized by the BDOS. Locations 0025H through 0037H are not initialized but are allocated and reserved by the BDOS. The CCP initializes the remaining base page values when it loads a program.

The BDOS allocates a user stack located in the highest address of the TPA. The maximum size of the stack equals the address of the stack pointer minus the last address of the program plus 1. The value of the initial stack pointer is passed to the LPB by the BDOS.

For programs loaded by a transient program rather than the CCP, refer to Section 2.2.3. Appendix B contains two examples, an assembly language program and a C language program, that illustrate how a transient program loads another program with the Program Load Function but without the CCP.

4.6 Exception Functions

This section describes the Set Exception (61), Set Supervisor State, (62), and the Get/Set TPA Limits Functions that set exceptions for error handling and other exception processing.

4.6.1 Set Exception Vector Function

FUNCTION 61: SET EXCEPTION VECTOR	
Entry Parameters:	
Register D0.W:	3DH
Register D1.L:	EPB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H
	error: FFH

The Set Exception Vector Function allows a program to reset current exception vectors, set new exception vectors, and create exception handlers for the 68000 microprocessor.

In addition to passing the function number in register D0.W, a program must pass the address of the Exception Parameter Block (EPB) in register D1.L. The EPB is a 10-byte data structure containing a one-word vector number and two longword vector values. See Figure 4-7. The EPB specifies the exception and the address of the new exception handler. Table 4-21 lists valid exceptions that correspond to 68000 microprocessor hardware. The starting location of the EPB must be an even-numbered address.

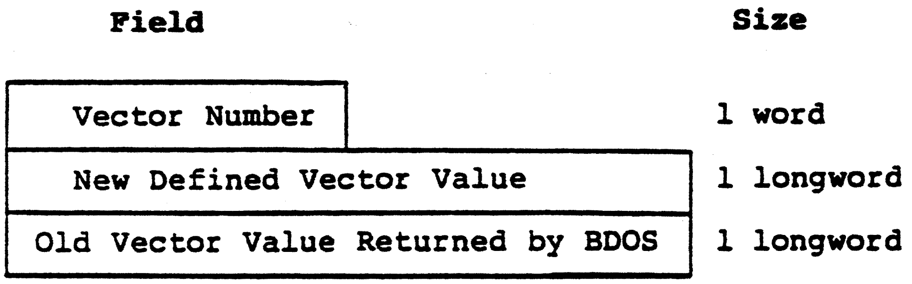

Figure 4-7. Exception Parameter Block (EPB)

The vector number identifies the exception. The New Vector Value specifies the address of the new exception handler for the specified exception. The BDOS returns in the Old Vector Value Field, the value that the exception vector contained before this function was invoked. The BDOS replaces the old vector value with the new vector value in its table of exception handlers and returns the address of the old exception handler to the old vector value in the EPB. After the BDOS sets the new exception vector, it passes the value 00H in register D0.W. However, if an error, such as a bad vector, occurs while the vector is being set, this function passes the value FFH in register D0.W. The bad vector error occurs when a vector other than one listed in Table 4-21 is specified for this function.

When an exception occurs, before the BDOS passes control to an exception handler, the BDOS restores the system state (user or supervisor) to the state of the system before it incurred the exception. To return from an exception handler to the normal processing state, the last instruction an exception handler executes is a Return and Restore (RTR) instruction.

Bus and address errors require special handling because they push four additional words onto the stack. For example, when a bus error occurs, the system pushes flags, the access address, and the instruction register onto the stack. An exception handler must pop these off the stack before it executes a RTR instruction.

If an exception handler does not exist for an exception, when that exception occurs, the BDOS default exception handler returns an exception message to the logical console device (CONSOLE) before it aborts the program. The BDOS exception message format is defined below:

```
Exception nn at user address aaaaaaaa. Aborted.
```

where:

nn	is a hexadecimal number in the range 2 through 17 or 24 through 2F that defines all exceptions excluding reset, hardware interrupts, and system Traps 0 through 3.
aaaaaaaa	is the address of the instruction following the one that caused the exception.

Except for exceptions handled by resident system extensions (RSXs), the BDOS reinitializes all vectors to the default exception handler when the BDOS System Reset Function (0) is invoked. Any exception vectors, which your program sets, are reset after the BDOS warm boots the system. An RSX is a program that is not configured in the operating system but remains resident in memory after it is loaded. RSXs normally provide additional system functions. The Get/Set TPA Limits Function (63) allows you to create an area in the TPA in which one or more RSXs can reside.

Table 4-21. Valid Vectors and Exceptions

Vector	Exception
2	Bus Error
3	Address Error
4	Illegal Instruction
5	Zero Divide
6	CHK Instruction
7	TRAPV Instruction
8	Privilege Violation
9	Trace
10	Line 1010 Emulator
11	Line 1111 Emulator
32*	Trap 0
33*	Trap 1
36**	Trap 4
37**	Trap 5
38**	Trap 6
39**	Trap 7

* Vectors reserved for Resident System Extensions (RSX) implemented with the Get/Set TPA Limits Function (63).

** Recommended Trap vectors for applications.

4.6.2 Set Supervisor State

FUNCTION 62: SET SUPERVISOR STATE
Entry Parameters: Register D0.W: 3EH
Returned Values: Register D0.W: 00H

The Set Supervisor Function puts the calling program in supervisor state. This function should not be used by novice programmers and experienced programmers should be careful when invoking this function.

The user stack is swapped when the program enters supervisor state. On return from this function, the stack pointer, register A7, is the supervisor stack pointer and not the user stack pointer. Thus, you cannot use register A7 to reference the user stack.

The supervisor stack is used by the BDOS and BIOS. This stack is 300 longwords or 1200 bytes long. The percent of the stack used by the system is dependent on the operation being performed and those previously performed. Therefore, you cannot predict how much of the stack is available for program operations. To avoid stack overflow and overwriting the system, you should not push more than a few dozen bytes onto the stack, especially when you call BDOS and BIOS functions.

An alternate method of avoiding stack overflow is to switch to a private supervisor stack. You create the stack by loading into A7 the address of an area in memory that you use as the supervisor stack. The address must be an even address. If you call BDOS and BIOS functions, your private supervisor stack should be 300 longwords, more than the space required by the program. If your program exits supervisor mode, make sure your program restores the system stack pointer to its original value. The supervisor stack is reinitialized when the system warm boots.

Note that in future CP/M-68K upward compatible systems, this function may not exist, or will require privilege for the calling process to access this function, or the function will fail. If it fails the value FFH will be passed to D0.W. However, no privilege is currently necessary. The function is always successful and the value 00H is passed in register D0.W.

4.6.3 Get/Set TPA Limits

FUNCTION 63: GET-SET TPA LIMITS	
Entry Parameters:	
Register D0.W:	3FH
Register D1.L:	TPAB Address
Returned Values:	
Register D0.W:	00H
Register TPAB:	Contains TPA Values

The Get/Set TPA Limits Function allows you to obtain or set the boundaries of the Transient Program Area (TPA). You must load the address of the Transient Program Area Block (TPAB) in register D1.L. The TPAB is a 5-word data structure consisting of one word and two longwords. You create the TPAB in the TPA as illustrated in Figure 4-8.

Byte Offset	Field	Size
00H	Parameters	1 word
02H	Low TPA address	1 longword
06H	High TPA address + 1	1 longword

Figure 4-8. Transient Program Parameter Block

The value of the first two bits in the one-word Parameters Field determines whether this function gets or sets the TPA limits and which fields you supply. Figure 4-9 illustrates the format of the parameters field.

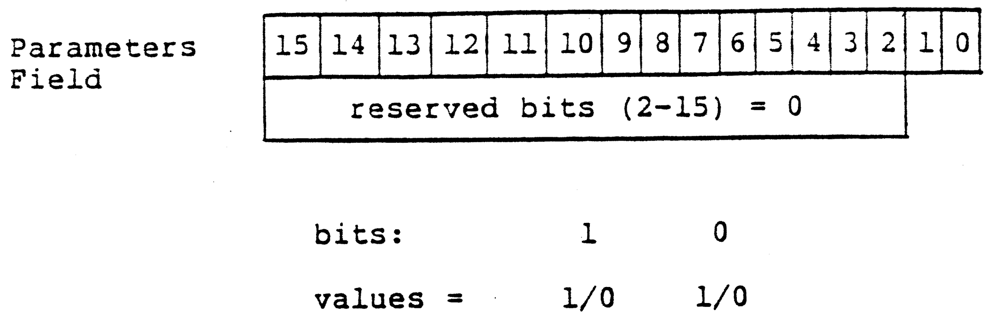

Figure 4-9. Parameters Field in TPAB

Bit Zero determines whether you get or set the TPA limits. When the value of bit zero is zero, the BDOS returns the current TPA boundaries in the Low and High Address fields of the TPAB. When the value of bit zero is one, the BDOS sets new TPA boundaries. The BDOS uses the values that you specify in the Low and High TPA address fields of the TPAB to set the new TPA boundaries.

When you set the TPA boundaries, bit one determines whether the boundaries are temporary or permanent. When the value of bit one is zero, the TPA boundaries that you set are temporary; when the system warm boots, the previous TPA limits are restored. When the value of bit one is one, the TPA values that you set are permanent; they are not changed when the system warm boots.

Bits 2 through 15 contain zeroes. These bits are reserved for future use. Table 4-22 summarizes the values of bits zero and one.

Table 4-22.
Values For Bits 0 and 1 in the TPAB Parameters Field

Bit	Value	Explanation
0	0	Return boundaries of current TPA in TPAB Low and High Address Fields.
	1	Set new TPA boundaries with the values loaded in TPAB Low and High address fields.
1	0	Restore previous TPA values when the system warm boots.
	1	Permanently replace the TPA boundaries with the ones you specify in the Low and High TPAB Address Fields.

The examples below illustrate and explain values for bits zero and one.

Examples:

1) Get TPA Limits

1	0
0	0

This function returns the boundaries of the current TPA in the Low and High Address Fields of the TPAB when the value of bit zero equals 0.

2) Temporarily Set TPA Limits

1	0
0	1

This function temporarily sets the TPA boundaries to the boundaries that you supply in the Low and High Address Fields of the TPAB when bit zero equals 1 and bit one equals 0. The TPA boundaries are reset when the system warm boots.

3) Permanently Set TPA Limits

1	0
1	1

This function permanently sets the TPA boundaries to the values that you supply in the Low and High Address Fields of the TPAB when the value of bit zero equals 1 and bit one equals 1. The TPA limits remain set until this function is called to reset the boundaries or you cold boot system.

End of Section 4

Section 5
AS68 Assembler

5.1 Assembler Operation

The CP/M-68K Assembler, AS68, assembles an assembly language source program for execution on the a 68000 microprocessor. It produces a relocatable object file and, optionally, a listing. The assembly language accepted by AS68 is identical to that of the Motorola 68000 assembler described in the Motorola manuals: M68000 Resident Structured Assembler Reference Manual M68KMASM(D4) and the 16-bit Microprocessor User's Manual, third edition MC68000UM(AD3). Appendix D contains a summary of the instruction set. Exceptions and additions are described in Sections 5.6 and 5.7.

5.2 Initializing AS68

If the file AS68SYM.DAT is not on your disk, you must create this file to initialize AS68 before you can use AS68 to assemble files. To initialize AS68, specify the AS68 command, the -I option, and the filename AS68INIT as shown below.

```
AS68 -I AS68INIT
```

AS68 creates the output file AS68SYMB.DAT, which AS68 requires when it assembles programs. After you create this file, you need not specify this command line again unless you reconfigure your system to have different TPA boundaries.

5.3 Invoking the Assembler (AS68)

Invoke AS68 by entering a command of the following form:

```
AS68 [-F d:] [-P] [-S d:] [-U] [-L] [-N] [-I]
[-O object filename]
source filename [>listing filename]
```

Table 5-1. Assembler Options

Option	Meaning
-F d:	The -F option specifies the drive on which temporary files are created. The variable d: is the drive designation, which must be followed by a colon. If this option is not specified, the temporary files that AS68 creates are created on the current drive.
-I	The -I option initializes the assembler. See Section 5.2 for details.
-P	If specified, AS68 produces and prints a listing on the standard output device which, by default, is the console. You can redirect the listing, including error messages, to a file by using the >listing filename parameter. Note that error messages are produced whether or not the -P option is specified. No listing is produced, however, unless you specify the -P option.
-S d:	The -S option indicates the drive on which the assembler initialization file, AS68SYMB.DAT, resides. This file is created when you initialize AS68. See Section 5.2. AS68 reads the file AS68SYMB.DAT before it assembles a source file. The variable, d:, is the drive designation; it must be followed by a colon. If you do not specify this option, AS68 assumes the initialization file is on the default drive.
-U	Causes all undefined symbols in the assembly to be treated as global references.

Table 5-1. (continued)

Option	Meaning
-L	Ensures all address constants are generated as longwords. Use the -L option for programs that require more than 64K for execution or if the TPA is not contained in the first 64K bytes of memory. If -L is not specified, the program is assembled to run in the first 64K bytes of memory. If an address in the assembly does not fit within one word an error occurs. Appendix E describes all AS68 errors.
-N	Disables optimization of branches on forward references. Normally, wherever possible, AS68 uses the 2-byte form of the conditional branch and the 4-byte BSR instruction to speed program execution and reduce the instruction size instead of the 6-byte JSR instruction.
source filename	This is the only required parameter. It is the file specification of the assembly language source program to be assembled.
>listing filename	If specified, the listing requested with the -P option is directed to the specified file rather than to your console terminal, the standard output device. The error messages are produced in the listing file. Note that if you do not request a listing file, you can still redirect the error messages to a file by specifying the greater than symbol (>) immediately followed by a file specification.

5.4 Assembly Language Directives

This section alphabetically lists and briefly describes the directives AS68 supports.

Table 5-2. Assembly Language Directives

Directive	Meaning
comm label, expression	The common directive (comm) specifies the label and size of a common area, which can be shared by separately assembled programs. The linker, LO68, links all common areas with the same label to the same address. The size of the common area is determined by the value of the largest expression when more than one common area with the same label exists.
data	The data directive instructs AS68 to change the assembler base segment to the data segment.
bss	The bss directive instructs AS68 to change the assembler base segment to the block storage segment (bss). Instructions and data cannot be assembled in the bss. However, symbols can be defined and storage can be reserved with the .ds directive in the bss.
dc operand [,operand, ...]	The define constant directive (dc) defines one or more constants in memory. When you specify more than one operand, separate each with a comma. The operand can contain a symbol or an expression that is assigned a numeric value by AS68, or the value of the constant in decimal, hexadecimal, or ASCII notation. If you specify an ASCII value, you must enclose the string in single quotes ('). Although an ASCII character is only seven bits in length, each character is assigned a byte of memory. The eighth bit always equals zero.

Table 5-2. (continued)

Directive	Meaning
	You can specify the constants to be bytes, words, or longwords. The list below illustrates the notation for each of these size specifications and describes the rules that apply to them. dc.b The constants are byte constants. If you specify an odd number of bytes, AS68 fills the odd byte on the right with zeroes unless the next statement is another dc.b directive. When the next statement is a dc.b directive, the dc.b uses the odd byte. Byte constants are not relocatable. dc.w The constants are word constants. If you specify an odd number of bytes, AS68 fills the last word on the right with zeroes to force an even byte count. The only way to specify an odd number of bytes is with an ASCII constant. Word constants can be relocated. dc.l The constants are longword constants. If less than a multiple of four bytes is entered, AS68 fills the last longword on the right with zeroes to force a multiple of four bytes. Longword constants can be relocated.
ds operand	The define storage directive (ds) reserves memory locations. The contents of the memory that it reserves is not initialized. The operand specifies the number of bytes, words, or longwords that this directive reserves. The notation for these size specifications is shown below. ds.b reserves memory locations in bytes ds reserves memory locations in words ds.l reserves memory locations in longwords

Table 5-2. (continued)

Directive	Meaning
end	The end directive informs AS68 that no more source code follows this directive. Code, comments, or multiple carriage returns cannot follow this directive.
endc	The endc directive denotes the end of the code that is conditionally assembled. It is used with other directives that conditionally assemble code.
equ expression	The equate directive (equ) assigns the value of the expression in the operand field to the symbol in the label field that precedes the directive. The syntax for the equate directive is below. label EQU expression The label and operand fields are required. The label must be unique; it cannot be defined anywhere else in the program. The expression cannot include an undefined symbol or one that is defined following the expression. Forward references to symbols are not allowed for this directive.
even	The even directive increments the location counter to force an even boundary. For example, if specified when the location counter is odd, the location counter is incremented by one so that the next instruction or data field begins on an even boundary in memory.