DIGITAL RESEARCH™

CP/M-68K™

Operating System

Programmer's Guide

COPYRIGHT

DISCLAIMER

TRADEMARKS

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.

Second Edition: June 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 following CP/M commands and utilities:

- 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 Language 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 1/0 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 the following Motorola manuals:

o 16-BIT Microprocessor User's Manual, third edition MC68000UM(AD3)
o M68000 Resident Structured Assembler Reference Manual 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 CP/M-68K Operating System System Guide 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)

Table of Contents (continued)

Appendixes

Table of Contents (continued)

Appendixes (continued)

Table of Contents (continued)

Tables

Table of Contents (continued)

Figures

Listings

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 filesystem 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 1/0 System	BIOS	Provides functions that interface peripheral device drivers for I/0 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 1-2 lists the commands that invoke these tools. Tables 1-3 and 1-4 list other commands supported by CP/M-68K and the manual in which they are documented.

Table 1-2 describes commands used in the CP/M-68K Operating System Program- mer's Guide.

Table 1-2. 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.
L068	Invokes the Linker.
NM68	Invokes the NM68 Utility that prints the symbol table of an object or command file.
RELOC	Invokes the Relocation Utility that relocates a command file contain- ing 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 1-3 describes commands used in the CP/M-68K Operating System User s Guide.

Table 1-3. 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 1-4 describes commands used in the C Language Programming Guide for CP/M-68K.

Table 1-4. 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 I- through 8-character filename (f ... f), and a I- 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). 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 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, 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 1-5 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 1-5. 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 ? wltdcard 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 1-6 lists the terminology used throughout this guide to describe CP/M-68K values and program components.

Table 1-6. 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.

Table 1-7 describes conventions used in this manual.

Table 1-7. 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.
. . or	A vertical or horizontal ellipsis 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.

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 following list 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 property 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 Is 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 tall 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. I. The transient program must initialize the base page values that the CCP normally supplies, such as FCBS, the DMA address, and the command tall, if the program being loaded requires these values. The command tall contains the command parameters but not the command. The format of the command tall in the base page consists of a 1-byte character count, followed by the characters in the command tall, 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 < 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 two ways to exit a transient program and return control to the CCP:

- o Interactively, the user types CTRL-C at the console, the default I/O device
- o 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 following BDOS functions.

- o Console Output (2)
- o Print String (9)
- o 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.

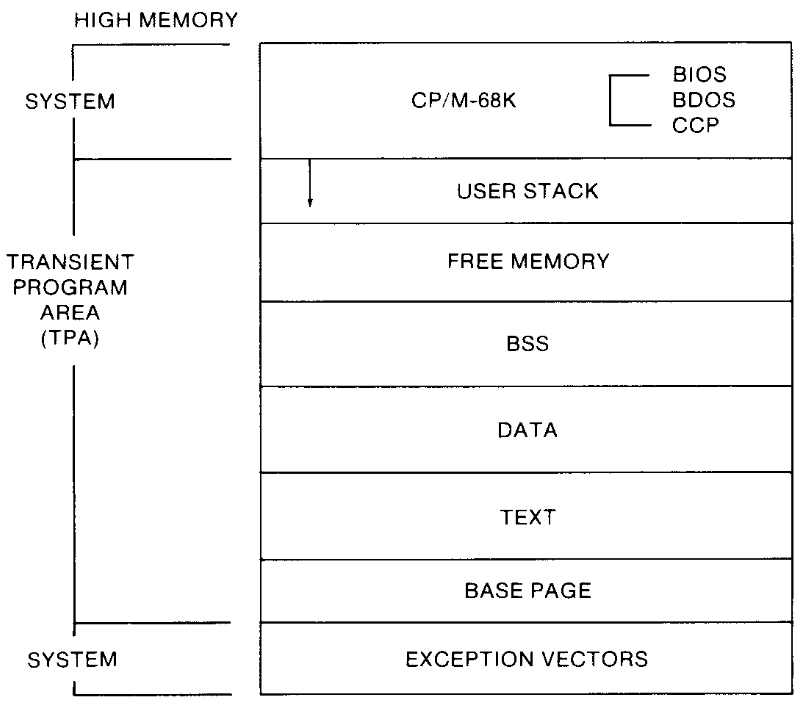

Figure 2-2. CP/M-68K Default Memory Model

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.

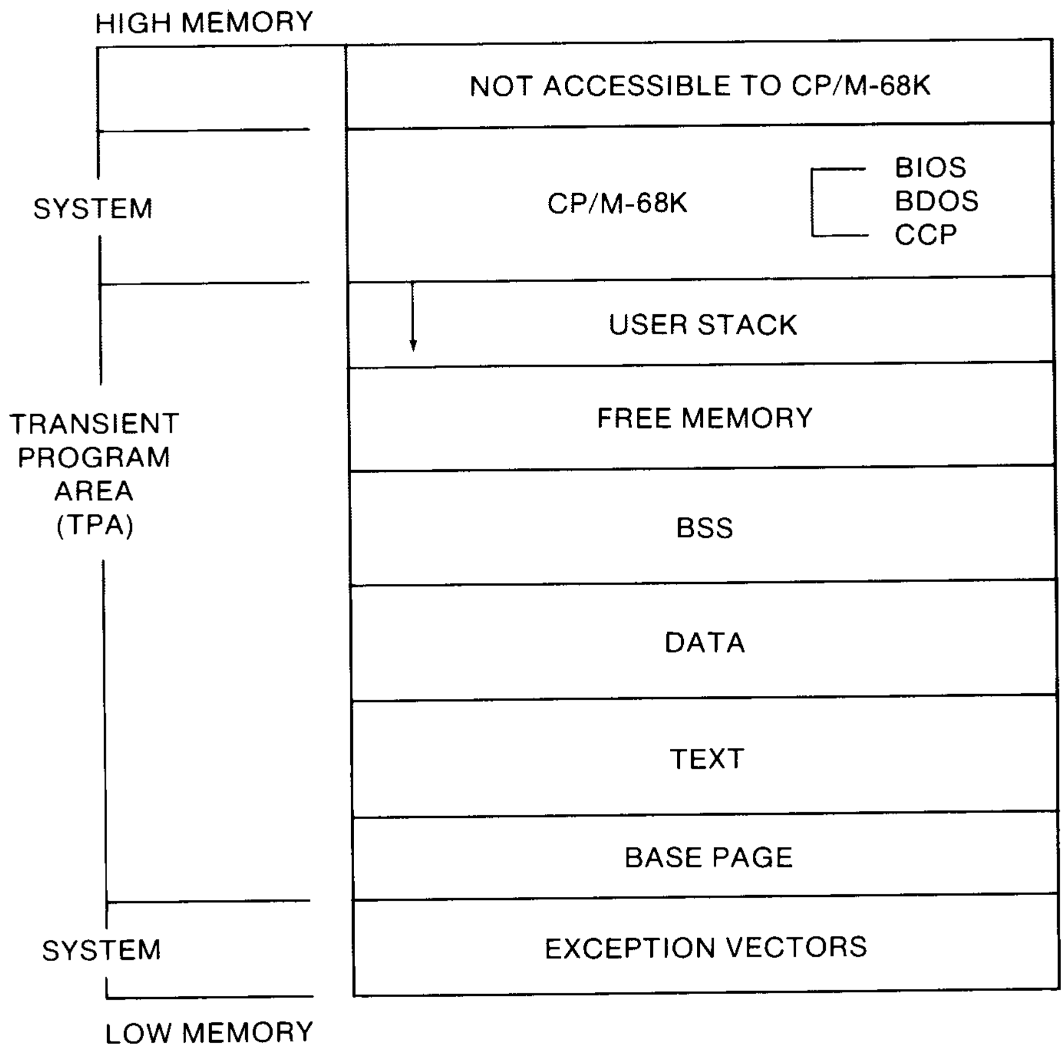

Figure 2-3. CP/M-68K Memory Model with Inaccessible Memory

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.

o Program segments:

text: contains the program instructions.

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.

o Symbol table: defines referenced symbols.

o 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.

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 60 I BH 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 (5 9) 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.

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.

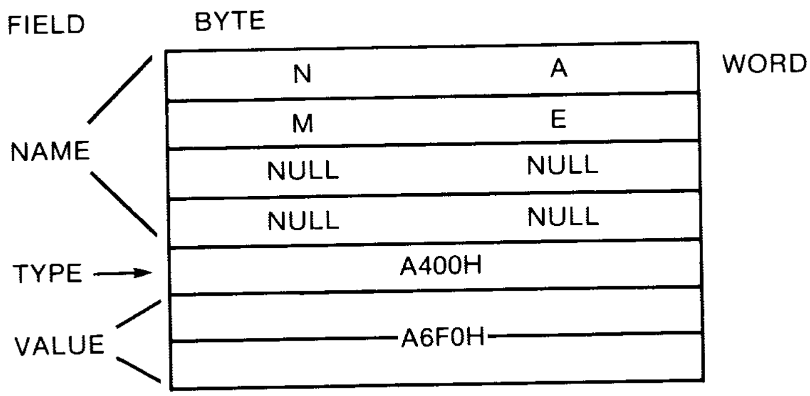

Figure 3-3. Entry in Symbol Table

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 preceding 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.

NM68 filename.0 [>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 following type descriptors:

- 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 in Table 3-2.

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.
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 FiIe	File Access
17	Search for First	File Access
18	Search for Next	File Access
19	Delete File	File Access
20	Read Sequential	File Access

*Must be implemented in the BIOS

Table 4-1. (continued)

F#	Function	Type
2t	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
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 DO (DO.W). Some functions require a second parameter, which is loaded, depending on its size, in the low order word (Dl.W) or longword (Dl.L) of data register Dl. 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 D 1 (D l.W). Some BDOS functions return a value, which is passed in the first word of data register DO (DO.W). The hexadecimal value FFFF is returned in register DO.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	Dl.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 following example 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',d1    *Moves the ASCII character upper-case U
                     *to the first word in data register D1.

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

The example above outputs the ASCII character upper-case U to the console. The assembler move instructions load register DO.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.

o File Access
o Drive Access
o Character I/O
o System/Program Control
o 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 3 6 bytes when a file is accessed randomly. The last three bytes in the 3 6-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 follow.

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 (O - 1 6)
0 = > use default drive for file
1 = > auto disk select drive A,
2 = > auto disk select drive B,
...
16 = > auto disk select drive P.

f 1...f 8 — contain the filename in ASCII upper-case. High bit should equal 0 when the file is opened.

tl,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), tl ', t2', and t3' denote the high bit. The following list indicates which attributes are set when these bits are set and equal the value 1.
tl' = 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 (I 5), 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,rl,r2	optional, contain random record number in the range 0-3FFFFH-, bytes r0, rl, 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 (I 7) and Search For Next (1 8) 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:

o CP/M Disk read error
o CP/M Disk write error
o CP/M Disk select error
o CP/M Disk change error
o 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 preceding format.

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 recessing. Each of these errors and their options are described in Table 4-4.

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. Table 4-4 describes each of these options.

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 falls, 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 following conditions.

o You specified a disk drive not supported by the BIOS.
o The BDOS receives an error from the BIOS.
o 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 in Table 4-5.

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 Dl.L:	FCB Address
	Returned Values:	
	Register D0.W:	Return Code
		success: 00H-03H error: FFH

The Open File Function matches the filename and filetype fields of the FCB specified in register Dl.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 Dl.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 Dl.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 DI.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 DO.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 Dl.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 Dl.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 DO.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 DO.W indexes the relative location of the matching directory entry within the directory record. For example, the value OIH 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 DOW 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 DO.W when no more directory entries match.

4.2.7 Delete File Function

FUNCTION 19: DELETE FILE

Entry Parameters:	
Register D0.W:	13H
Register Dl.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 Dl.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 Dl.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 Dl.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-bvte 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 DO.W when the read operation is successful. The value of 01H is returned in DO.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 DI.L:	FCB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H error: FFH

The Write Sequential Function writes a 128-byte record from the DMA buffer to the disk file whose FCB address is passed in register Dl.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 o he 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 following conditions.

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 Dl.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 DO.W contains an integer value in the range 00H through 03H when the function is successful. Register DO.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 DI. L:	FCB Address
Returned Values:	
Register D0.W:	Return Code
	success: 00H
	error: FFH

The Rename File Function uses the FCB specified in register Dl.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 dl2) contain the new name of the file.The filenames and file types 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.

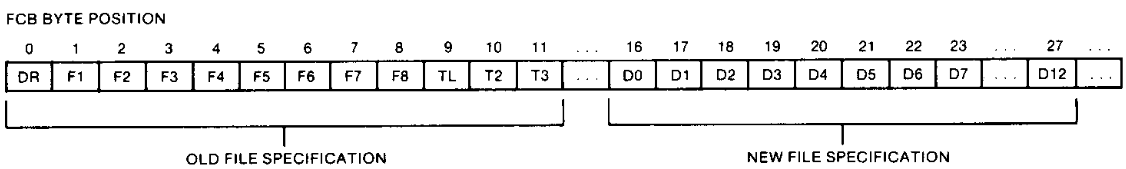

Figure 4-1. FCB Format for Rename Function

In the above figure, horizontal ellipses indicate FCB fields that are not required for this function. Refer to Section 4.2.1 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 DI.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 non-DMA 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 Dl.L:	CB Address
Returned Values:	
Register D0.W:	Return Code
	Success: 00H
	or: 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 (fl through f8) and filetype (t1 through t3) fields in the FCB denotes whether attributes are set. When the high bit n 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 Dl.L. Wildcards cannot be specified in he 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
fl through f4	User-defined attributes for application programs.
f5 through f8	Reserved for future use by CP/M-68K.
tl	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 or renamed.
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 CP/M-68K Operating System User's Guide.
t3	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 (1 5) and Close File (1 6) 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 Dl.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 Dl.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, r I, 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 Dl.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 Dl.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 (2 1) 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 Write Random 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 with- out error.
02	No available date 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 at- tem to 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 Dl.L:		FCB Address

Returned Values:

Register D0.W:	00H	
	success:	File Size written o 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 Dl.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 DI.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 per- formed on the file.

Table 4-10. Current Position Definitions

Operation	Function	Current Position
Open file	Open File (1 5)	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 (2 1)	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 Dl.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 DI.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 Dl.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 DOW 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 4-1 1. Drive Functions

Function	Function Number
Reset Disk System	1 3
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 DI.W:	Disk Number

Returned Values:

Register D0.W:	0H

The Select Disk Function designates the disk drive specified in register Dl.W as the default disk for subsequent file operations. The decimal numbers 0 through 15 correspond to drives A through P. For example, Dl.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 3 1: GET DISK PARAMETERS

Entry Parameters:	
Register D0.W:	1FH
Register Dl.L:	CDPB Address
Returned Values:	
Register D0.W:	00H
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 Dl.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 Dl.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 Dl.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 Dl.W:	Disk Number

Returned Values:

Register D0.W:	00H
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 Dl.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 I 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
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-1) 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 Dl.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 Dl.W contains one of the following values.

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 nonzero, console input is ready to be read by the BIOS Conin Function.

When the input value in Dl.W is neither FEH nor FFH, the Direct Console I/O Function assumes that Dl.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 Dl.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 DI.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 Dl.L takes the form:

D1.L: +0	+1	+ 2	+ 3	+ 4	+ 5	+ 6	+ 7	+ 8 ...	+n
mx	nc	cl	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 I 1: 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 DO.W; otherwise the value 00H is returned in DO.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 DI.W:	ASCII Character

Returned Values:

Register D0.W:	00H

The Auxiliary Output function sends a character from register DI.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,O)
- 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 (URI:)
- 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 DO.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 Dl.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 Dl.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 4-1

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: VersionNumber

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 TM	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 TM	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 Dl.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 Dl.W contains a value in the range 0 through 15 (decimal), the current user number is changed to the value in register DI.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 ≤ 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 (2 1), 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 Dl.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 Dl.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 Dl.L. After a program is loaded, the BDOS returns one of the following return codes 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.

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	DEFAUT 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 basepage 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 in the TPA normally at the highest address. 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.

FIELD	SIZE
VECTOR NUMBER	1 WORD
NEW DEFINED VECTOR VALUE	1 LONGWORD
OLD VECTOR VALUE RETURNED BY BDOS	1 LONGWORD

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 as follows.

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 handier 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 Dl.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 Dl.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.

PARAMETERS FIELD	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RESERVED BITS (2-15) = 0															

BITS: 1 0
VALUES = 1/0 1/0

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 following examples 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 I 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.

```
AS88 -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 -1 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.
-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.

Table 5-1. (continued)

Option	Meaning
-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, L068, 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. You can specify the constants to be bytes, words, or longwords. The following list illustrates the notation for each of these size specifications and describes the rules that apply to them.

Table 5-2. (continued)

Directive	Meaning	
	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.1	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.w	reserves memory locations in words
	ds.1	reserves memory locations in longwords
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.	

Table 5-2. (continued)

Directive	Meaning
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 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.
glob[label[,Iabel xdef label[,Iabel xref label[,Iabel]	These directives make the label(s) external. If the labels are defined in the current assembly, this statement makes them available to other routines during a load by L068. If the labels are not defined in the current assembly, they become undefined external references, which L068 links to external values with the same label in other routines. If you specify the -U option, the assembler makes all undefined labels external.

Table 5-2. (continued)

Directive	Meaning
ifeq expression ifne expression ifle expression lflt expression lfge expression lfgt expression	All of the directives listed above are conditional directives in which the expression is tested against zero for the condition specified by the directive. If the expression is true, the code following is assembled; otherwise, the code is ignored until an end conditional directive (endc) is found. The directives and the conditions they test are listed below. ifeq equal to zero ifne not equal to zero ifle less than or equal to zero iflt less than zero lfge greater or equal to zero fgt greater than zero
ifc 'stringl', 'string2' ifnc 'stringl', 'string2'	The conditional string directive compares two strings. The 'c' condition is true if the strings are exactly the same. The 'nc' condition is if they do not match.

Table 5-2. (continued)

Directive	Meaning
offset expression	The offset directive creates a dummy storage section by defining a table of offsets with the define storage directive (ds). The storage definitions are not passed to the linker. The offset table begins at the address specified 'in the expression. Symbols defined in the offset table are internally maintained. No instructions or code-generating directives, except the equate (equ) and register mask (reg) directives, can be used in the table. The offset directive is terminated by one of the following directives: bss data end section text
org expression	The absolute origin directive (org) sets the location counter to the value of the expression. Subsequent statements are assigned absolute memory locations with the new value of the location counter. The expression cannot contain any forward, undefined, or external references.
page	The page directive causes a page break which forces text to print on the top of the next page. It does not require an operand or a label and it does not generate machine code. The page directive allows you to set the page length for a listing of code. If you use this directive and print the source code by specifying the -P option in the AS68 command line, pages break at predefined rather than random places. The page directive does not appear on the printed program listing.

Table 5-2. (continued)

Directive	Meaning
reg reglist	The register mask directive builds a register mask that can be used by movem instruction. One or more registers can be listed in ascending order in the format: R?[-R[/R?[-R?...] ...]] Replace the R in the above format with a register reference. Any of the following mnemonics are valid: A0 - A7 D0-D7 R0 - R15 The following example illustrates a sample register list. A2-A/A7/D1/D3-D5 You can also use commas to separate registers: A1,A2,D5,D7
section	The section directive defines a base segment. The sections can be numbered from 0 to 15 inclusive. Section 14 always maps to data. Section 15 is bss. All other section numbers denote text sections.
text	The text directive instructs AS68 to change the assembler base segment to the text segment. Each assembly of a program begins with the first word in the text segment.

5.5 Sample Commands Invoking AS68

A>AS68 -U -L TEST.S

This command assembles the source file TEST.S and produces the object file TEST.O. Error messages appear on the screen. Any undefined symbols are treated as global.

A>AS68 -P SMPL.S >SMPL.L

This command assembles the source file SMPL.S and produces the object file SMPL.O. The program must run in the first 64K of memory; that is, no address can be larger than 16 bits. Error messages and the listing are directed to the file SMPL.L.

5.6 Assembly Language Differences

The syntax differences between the AS68 assembly language and Motorola's assembly language are described in the following list.

1. All assembler directives are optionally preceded by a period(.).For example,
   .equ or equ
   .ds or ds
2. AS68 does not support, but accepts and ignores the following Motorola directives:
   comline
   mask2
   idnt
   opt
3. The Motorola set directive is implemented as the equate directive (equ).
4. AS68 accepts upper- and lower-case characters. You can specify instructions and directives in either case. However, labels and variables are case sensitive. For example, the label START and Start are not equivalent.
5. For AS68, all labels must terminate with a colon (:). For example,
   A :
   FOO:
   However, if a label begins in column one, it need not terminate with a colon

6. For AS68, ASCII string constants can be enclosed in either single or double quotes. For example,

   'ABCD '

   "ac14"

7. For AS68, registers can be referenced with the following mnemonics:

   r0-r15

   R0-R15

   d0-d7

   D0-D7

   a0-a7

   A0-A7

   Upper- and lower-case references are equivalent. Registers RO-R7 are the same as DO-D7 and R8-RI5 are the same as AO-A7.

8. For AS68, comment lines cannot begin with an asterisk that is immediately followed by an equals sign (* =), since the location counter can be manipulated with a statement of the form:

   *= expr

9. Use caution when manipulating the location counter forward. An expression can move the counter forward only. The unused space is filled with zeros in the text or data segments.

10. For AS68, comment lines can begin with an asterisk followed by an equals sign but only if one or more spaces exist between the asterisk and the equals sign:

    This command loads RI with zeros.

    Branch to subroutine XYZ

11. For AS68, the syntax for short form branches is bxx.b rather than bxx.s

12. The Motorola assembler supports a programming model in which a program consists of a maximum of 16 separately relocatable sections and an optional absolute section. AS68 distributed with CP/M-68K does not support this model. Instead, AS68 supports a model in which a program contains three segments, text, data, and bss as described in Sections 2 and 3 of this guide.

5.7 Assembly Language Extensions

The following enhancements have been added to AS68 to aid the assembly language programmer by making the assembly language more efficient:

1 . When the instructions add, sub, cmp are used with an address register in the source or destination, they generate adda, suba, and cmpa. When the clr instruction is used with an address register (Ax), it generates sub Ax, Ax.
2. add, and, cinp, eor, or, sub are allowed with immediate first operands and actually generate addi, andi, cmpl, eorl, orl, subi, instructions if the second operand is not register direct.
3. All branch instructions generate short relative branches where possible, including forward references.
4. Any shift instruction with no shift count specified assumes a shift count of one. For example, a s 1 r I is equivalent to a s I # I o r I .
5. A jsr instruction is changed to a bsr instruction if the resulting bsr is shorter than the jsr instruction.
6. The text directive causes the assembler to begin assembling instructions in the text segment.
7. The data directive causes the assembler to begin assembling 'initialized data in the data segment.
8. The bss directive instructs the assembler to begin defining storage in the bss. No instructions or constants can be place in the bss because it is for uninitialized data only. However, the ds directives can be used to define storage locations, and the location counter (*) can be incremented.
9. The glob] directive in the form:

   .globl label[,label] ...

   makes the labels external. If they are otherwise defined (by assignment or appearance as a label) they act within the assembly exactly as if the globl directive was not given. However, when linking this program with other programs, these symbols are available to other programs. Conversely, if the given symbols are not defined within the current assembly, the linker can combine the output of this assembly with that of others which define the symbols.

10. Thecommon directive (comm) defines a common region, which can be accessed by programs that are assembled separately. The syntax for the common directive is

    COMM label, expression

    The expression specifies the number of bytes that is allocated in the common region. If several programs specify the same label for a common region, the size of the region is determined by the value of the largest expression.

    The common directive assumes the label is an undefined external symbol in the current assembly. However, the linker, L068, is special-cased, so all external symbols, which are not otherwise defined, and which have a nonzero value, are defined to be in the bss, and enough space is left after the symbol to hold expression bytes. All symbols which become defined in this way are located before all the explicitly defined bss locations.
11. The even directive causes the location counter (*), if positioned at an odd address, to be advanced by one byte so the next statement is assembled at an even address.
12. The instructions, move, add, and sub, specified with an immediate first operand and a data (D) register as the destination, generate Quick instructions, where possible.

5.8 Error Messages

Appendix E lists the error messages generated by AS68.

End of Section 5

Section 6
L068 Linker

6.1 Linker Operation

L068 is the CP/M-68K Linker that combines several AS68 assembled (object) programs into one executable command file. All external references are resolved. The linker must be used to create executable programs, even when a single object program contains no unresolved references. The argument routines are concatenated in the order specified. The entry point of the output is the first instruction of the first routine.

6.2 Invoking the Linker (LO68)

Invoke L068 by entering a command of the following form:

```
L068  [-Fd:.] [-R] [-S] [-I] [-Uname]
      [-O filename] [-X] [-Zaddress]
      [-Daddress] [-Baddress] object filename [object filename]
      [>message filename]
```

Table 6-1. Linker Command Options

Option	Meaning
-F d:	The -F option specifies the drive on which temporary files are created. The variable d: is the drive designation.
-R	The -R option preserves the relocation bits so the resulting executable program is relocatable.
	If specified, the output is stripped; the symbol table and relocation bits are removed to save memory space.

Table 6-1. (continued)

Option	Meaning
-I	If -1 'is specified, no 16-bit address overflow messages are generated. When you assemble a program without the AS68 -L option, the linker may generate address overflow messages if the program contains addresses longer than 16 bits.
-Uname	Forces linking of a library module which resolves the name parameter, even if the name is not referred to by any other module being linked. Normally library modules are only linked when they are needed to resolve references in other modules. You can use this option to create a program from a library if the module resolving the name parameter calls other modules in the library.
-0 filename	If specified, the object file produced by L068 has the filename that you specify following the -0 option. The -0 option and filename are separated by one or more spaces. If you do not specify a filename, the object file has the name C.OUT.
-X	If specified, the symbol table includes all local symbols except those that begin with the letter L. If not specified, L068 puts only global symbols in the symbol table. This option is provided so that you can discard compiler internally-generated labels that begin with the letter L while retaining symbols local to routines.
-Taddress -Zaddress	The -T and -Z options are equivalent. If specified, the hexadecimal address given is defined by L068 as the beginning address for the text segment. This address defaults to zero, or it can be specified as any even hexadecimal number between 0 and FFFFFFFFH. This option is especially useful for stand-alone programs, or for putting programs in ROM. Hexadecimal characters can be in upper-case or lower-case.

Table 6-1. (continued)

Option	Meaning
-Daddress	If specified, the hexadecimal address given is defined by L068 as the beginning address for the data segment. This address defaults to the next byte after the end of the text segment, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF. This option is especially useful for stand-alone programs or for putting programs in ROM. Hexadecimal address characters can be in upper-case or lower-case.
-Baddress	If specified, the hexadecimal address given is defined by L068 as the beginning address for the bss. This address defaults to the next byte after the end of the data segment, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF.
object filename [object filename]	The name of one or more object files produced by the assembler AS68. These are the object files that L068 links.
>message filename	If specified, error messages produced by L068 are redirected to the file that you specify immediately after the greater than (>) sign. If you do not specify a filename, error messages are written to the standard default output device, which typically is your console terminal.

6.3 Sample Commands Invoking LO68

A>LO68 -S -O TEST.68K TEST.O

This command links assembled file TEST.0 into file TEST.68K and strips out the symbol table and relocation bits.

A>LO68 -T4000 -D8000 -BC000 A.O B.O C.O

This command links assembled files A.0, B.0, and C.0 to the default output file C.OUT. The text segment starts at location 4000H; the data segment starts at location 8000H; and the bss starts at location C000H.

A>LO68 -I -O TEST.68K TEST.O TEST1.O >ERROR

This command links assembled files TEST.O and TEST1.O to file TEST.68K. Any 16-bit address overflow errors are ignored; error messages are directed to the file ERROR.

6.4 LO68 Error Messages

Appendix E lists the error messages that LO68 displays.

End of Section 6

Section 7
Programming Utilities

CP/M-68K supports five programming utilities: Archive (AR68), DUMP, Relocation (RELOC), SIZE68, and SENDC68. AR68 allows you to create and modify libraries. DUMP displays the contents of files in hexadecimal and ASCII notation. RELOC creates an absolute command file from a relocatable command file. SIZE68 displays the total size of a memory image command file and the size of each of its program segments. SENDC68 creates a file of Motorola S-records from a command file. S-records are, described in the CP/M-68K Operating System System Guide. This section describes each of these utilities in a separate subsection.

7.1 Archive Utility

The Archive Utility, AR68, creates a library or replaces, adds, deletes, lists, or extracts object modules in an existing library. AR68 can be used on the C Run-Time Library distributed with CP/M-68K and documented in the C Language Programming Guide for CP/M-68K for the 68000 microprocessor.

7.1.1 AR68 Syntax

To invoke AR68, specify the components of the following command line. Optional components are enclosed in square brackets ([ ]).

AR68 DRTWX[AV] [F D:] [OPMOD] ARCHIVE OBMOD! [OBMOD2...] [>filespec]

You can specify multiple object modules in a command line provided the command line does not exceed 127 bytes. The delimiter character between components consists of one or more spaces.

Table 7-1. AR68 Command Line Components

Component Meaning

AR68 invokes the Archive Utility. However, if you specify only the AR68 command, AR68 returns the following command line syntax and system prompt shown below.

```
A> AR68
usage: AR68 DRTWX[AV][FD:][OPMOD]ARCHIVE OBMOD1 [OPMOD2[][>FILESPEC]
A>
```

DRTWX indicates you must specify one of these letters as an AR68 command. Each of these one-letter commands and their options are described in Section 7.1.3.

AV indicates you can specify one or both of these one-letter options. These options are described with the commands in Section 7.1.3.

p

OPMOD is an object module within the library that you specify. The OPMOD parameter indicates the position in which additional object modules reside when you incorporate modules in the library and specify the A option.

F D: specifies the drive on which the temporary file created by AR68 resides. The variable D is the drive select code; it must be followed by a colon (:). AR68 creates a temporary file called AR68.TMP that AR68 uses as a scratchpad area.

ARCHIVE is the file specification of the library.

OBMOD1 [OBMOD2 ...]

indicates one or more object modules in a library that AR68 deletes, adds, replaces, or extracts.

Table 7-1. (continued)

Component	Meaning
>filespec	redirects the output to the file specification that you specify, rather than sending the output to the standard output device, which is usually the console device (CONSOLE). You can redirect the output for any of the AR68 commands described in Section 7.1.3.

7.1.2 AR68 Operation

AR68 sequentially parses the command line only once. AR68 searches for, inserts, replaces, or deletes object modules in the library in the sequence in which you specify them in the command line. Section 7.1.3 describes each of the commands AR68 supports.

When AR68 processes a command, it creates a temporary file called AR68.TMP. AR68 creates and uses AR68.TMP when it processes AR68 commands. After the operation is complete AR68 erases AR68.TMP. However, depending on when an error occurs, AR68.TMP is not always erased. If this occurs, erase AR68.TMP with the ERA command and refer to Appendix E for error messages output by AR68.

7.1.3 AR68 Commands and Options

This section describes AR68 commands and their options. Examples illustrate the effect and interaction between each command and the options it supports.

Table 7-2. AR68 Commands and Options

Command	Option	Meaning
D		deletes from the library one or more object modules specified in the command. You can specify the V option for this command.
	V	lists the modules in the library and indicates which modules are retained and deleted by the D command. The V option precedes modules retained in the library with the lower-case letter c and modules deleted from the library with the lower-case letter d.

Table 7-2. (continued)

Command	Option	Meaning
		A>AR68 DV MYRAH.ARC ORC.O c red.o c blue.o d orc.o c white.o A> The D command deletes the module ORC.0 from the library MYRAH.ARC. In addition to listing the modules in the library, the V option indicates which modules are retained and deleted.
R		creates a library when the one specified in the command line does not exist or replaces or adds object modules to an existing library. You must specify one or more object modules. You can replace more than one object module in the library by specifying their module names in the command line. However, when the library contains more than one module with the same name, AR68 replaces only the first module it finds that matches the one specified in the command line. AR68 replaces modules already in the library only if you specify their names prior to the names of new modules to be added to the library. For example, if you specify the name of a module that you want replaced after the name of a module that you are adding to the library, AR68 adds both modules to the end of the library. By default, the R command adds new modules to the end of the library. The R command adds an object module to a library if: o The object module does not already exist in the library. o You specify the A option in the command line. o The name of a module follows the name of a module that does not already exist in the library.

Table 7-2. (continued)

Command	Option	Meaning
		The A option indicates where AR68 adds modules to the library. You specify the relative position by including the OPMOD parameter with the A option. In addition to the A option, the R command also supports the V option, which lists the modules in the library and indicates the result of the operation performed on the library. All options are described below. Examples illustrate their use.
	A	adds one or more object modules following the module specified in the command line: A>AR68 RAV SDAV.O MTRAH.ARC WORK.O MAIL.O c much.o c sdav.0 a work.o a mail.o c less.o The RAV command adds the object modules WORK.0 and MAIL.0 after the module SDAV.0 in the library MYRAH.ARC. The V option, described below, lists all the modules in the library. New modules are preceded by the lower-case letter a and existing modules are preceded by the lower-case letter c.
	V	lists the object modules that the R command replaces or adds. A>AR68 RV JNNK.MAN NAIL.O WRENCH.O c saw.o c ham.0 r nail.o c screw.o a wrench.o A>

Table 7-2. (continued)

Command	Option	Meaning
		The R command replaces the object module NAIL.0 and adds the module WRENCH.0 to the library JNNK.MAN. The V option lists object modules in the library and indicates which modules are replaced or added. Each object module that is replaced is preceded with the lower-case letter r and each one that is added is preceded with the lower-case letter a.
T		requests AR68 print a table of contents or a list of specified modules in the library. The T command prints a table of contents of all modules in the library only when you do not specify names of object modules in the command line.
	V	displays the size of each file in the table of contents as shown in the following example. A>AR68 TV WINE.BAD rw-rw-rw- 0/0 6818 rose.o rw-rw-rw- 0/0 2348 white.o rw-rw-rw- 0/0 396 red.o A> The T command prints a table of contents in the library WINE.BAD. In addition to listing the modules in the library, the V option indicates the size of each module. The character string rw-rw-rw- 0/0 that precedes the module size is meaningless for CP/M-68K. However, if the file is transferred to a UNIX(R) system, the character string denotes the file protection and file owner. The size specified by the decimal number that precedes the object module name indicates the number of bytes in the module.

Table 7-2. (continued)

Command	Option	Meaning
W		writes a copy of an object module in the library to the >filespec parameter specified in the command line. This command allows you to extract a copy of a module from a library and rename the copy when you write it to another disk, as shown below. For this command, the >filespec parameter is not optional. A>AR68 GO.ARC NOW.O >B:NEWNAME.O The W command writes a copy of the object modute NOW.0 from the library GO.ARC to the file NEWNAME.0 on drive B.
X		extracts a copy of one or more object modules from a library and writes them to the default disk. If no object modules are specified in the command line, the X command extracts a copy of each module in the library.
	V	lists only those modules the X command extracts from the library. It precedes each extracted module with the lower-case letter: A>AR68 XV JNNK.MAN SAW.O HAM.O SCREW.O x saw.o x ham.o x screw.o The V option with the X command lists only the modules SAW.0, HAM.0, and SCREW.0 that the X command extracts from the library JNNK.MAN and precedes each of these modules with the lower-case letter x.

7.1.4 Errors

When AR68 incurs an error during an operation, the operation is not completed. The original library is not modified if the operation would have modified the library. Thus, no modules in the library are deleted, replaced, added, or extracted. Refer to Appendix E for error messages output by AR68.

When you specify the >filespec parameter in the command line to redirect the output and one or more errors occur, the error messages are sent to the output file. Thus, you cannot detect the errors without displaying or printing the file to which the output was sent. If the contents of the output file is an object file (see the W command), you must use the DUMP Utility described in Section 7.2 to read any error messages.

7.2 DUMP Utility

The DUMP Utility (DUMP) displays the contents of a CP/M file in both hexadecimal and ASCII notation. You can use DUMP to display any CP/M file regardless of the format of its contents (binary data, ASCII text, an executable file).

7.2.1 Invoking DUMP

Invoke DUMP by entering a command in the following format.

DUMP [-sxxxx] filenamel [>filename2]

Table 7-3. DUMP Command Line Components

Component	Meaning
-sxxxx	xxxx is an optional offset (in hexadecimal) into the file. If specified, DUMP starts dumping the contents of the file from the byte-offset xxxx and continues until it displays the contents of the entire file. By default, DUMP starts dumping the contents of the file from the beginning of the file until it dumps the contents of the entire file.
filenamel	is the name of the file you want to dump.
>fitename2	the greater than sign (>) followed by a filename or logical device optionally redirects the output of DUMP. You can specify any valid CP/M specification, or one of the logical device names CON: (console) or LST: (list device). If you do not specify this optional parameter, DUMP sends its output to the console.

7.2.2 DUMP Output

DUMP sends the output to the console (or to a file or device, if specified), 8 words per line, in the following format:

rrrroo (ffffff): hhhh hhhh hhhh hhhh hhhh hhhh hhhh hhhh *aaaaaaaaaaaaaaaa*

Table 7-4. DUMP Output Components

Component	Meaning
rrrr	is the record number (CP/M records are 128 bytes) of the current line of the display.
00	is the offset (in hex bytes) from the beginning of the CP/M record.
ffffff	is the offset (in hex bytes) from the beginning of the file.
hhhh	is the contents of the file displayed in hexadecimal.
aaaaaaaa	is the contents of the file displayed as ASCII characters. If any character is not representable in ASCII, it is displayed as a period

7.2.3 DUMP Examples

The following example shows the DUMP Utility. The example shows the contents of a command file that contains both binary and ASCII information.

```
A>dump dump,68K
0000 00 (000000):  601a 0000 1b34  0000  011d 0000 0e5e 0000  *',,.,4....... ^..*
0000 10 (000010):  0000 0000 0000  0000  0900 ffff   6034 4320 *........... '4C *
0000 20 (000020): 5275 6e74 696d   6520 436f 7079 7269 6766  *Runtime Copyrigh*
0000 30 (000030): 7420 3139 3632   2062 7920 4469 6769 7461 *t 1962 by Djgjta*
0000 40 (000040): 6c20 5265 7365   6172 6369 2056 3031 2c30 *1 Research V0l.0*
0000 50 (000050): 3320 206f  0004   2268 0016 2649 d3e6 00lc   *3   0.."h,,&ISh..*

.          . (and so on) . . .
```

7.3 Relocation Utility

The Relocation Utility (RELOC) creates an absolute file from a relocatable command file. See Section 3 for a description of the CP/M-68K command file format. An absolute file is a file that is loaded at an absolute address. RELOC creates the absolute file by relocating the address constants in the file before it strips off the relocation bits. Thus, RELOC creates a new file but does not alter the original file.

The advantage of using RELOC is RELOC decreases the size of the file and increases performance. You can load the absolute command file into memory approximately twice as fast as its relocatable counterpart and it occupies half the disk storage space.

7.3.1 Invoking RELOC

You invoke RELOC by entering a command in the format:

RELOC [-Baddress] input filename output filename

Table 7-5. RELOC Command Line Components

Component	Meaning
-Baddress	The address parameter is the absolute address for the command file. The address parameter is optional. If you do not specify the address parameter, RELOC uses the base address at which it runs as the default address for relocating the input file. See the first example in Section 7.3.2. The base address of the file is normally the lowest address in the TPA + TOOH.
input filename	The input filename is the file specification of the relocatable command file that RELOC converts to an absolute file.
output filename	The output filenarne is the file specification of the absolute file RELOC creates.

7.3.2 RELOC Examples

This section contains two examples of RELOC. The first example illustrates how to relocate a file with the filetype of REL to the bottom of the TPA. You can use this example to create an absolute command file that runs in the bottom of the TPA. The second example illustrates how to specify an alternate address for a command file.

1 . In this example, the RELOC.REL file distributed with CP/M-68K is used to relocate itself. The resulting file, RELOC.68K, uses its base address for the absolute address of an input file when the address parameter of the input file is not specified. You can use this example to relocate other utilities with a filetype of REL so that they also run in the bottom of the TPA.

```
A>RELOC.REL RELOC.REL RELOC.68K
```

The RELOC.REL file relocates itself and outputs the file RELOC.68K. The command file RELOC.68K is an absolute file that runs at the bottom of the TPA.

2. In this example, RELOC creates an absolute file that must be loaded at a specific address.

   A>RELOC -B500 JUNK.REL JUNK.68K

   RELOC converts the relocatable command file, JUNK.REL, to the absolute command file, JUNK.68K, which must load into memory at location 500H.

7.4 SIZE68 Utility

The SIZE68 Utility (SIZE68) displays the sizes of each program segment within one or more command files and the total memory needed by each file. CP/M-68K command files usually have a filetype of .68K or REL. The size of a command file returned by SIZE68 and the size of a command file returned by the STAT command are not equal. The file size returned by SIZE68 includes the size of the text, data, and bss program segments but does not include the size of the header, symbol table, and relocation bits. For more details on the CP/M-68K command file format, refer to Section 3. For more details on the STAT command, refer to the CP/M-68K Operating System User's Guide.

7.4.1 Invoking SIZE68

You invoke SIZE68 by entering the SIZE68 command line in the following format:

SIZE68 filename [filename2 filename3 ... [>outfile]

Table 7-6. SIZE68 Command Line Components

Component	Meaning
filename	the file specification of a file whose size you want to determine.
filenamel filename2	one or more additional file specifications of files whose size you want to determine. SIZE68 can process multiple files, provided the command line does not exceed 128 bytes.
>outfile	specifies the file specification to which SIZE68 sends its output. If you do not specify an output file specification, SIZE68 sends the output to the console. For the output file specification, you can specify a valid CP/M filename, or one of the logical device names CON: (console), or LST: (list device).

7.4.2 SIZE68 Output

SIZE68 produces one output line for each input file you specify. The output line should be in the following format:

filename: csize + dsize + bsize = totsize (hexsize) stack size = ssize

Table 7-7. SIZE68 Output Components

Component	Meaning
csize	is the size, in decimal bytes, of the text segment of the file.
dsize	is the size, in decimal bytes, of the data segment of the file.
bsize	is the size, in decimal bytes, of the block storage segment (bss) of the file.
totsizc	is the total size, in decimal bytes, of the memory image occupied by the file. It is the sum of csize, dsize, and bsize.
hexsize	is the same value as totsize, expressed in hexadecimal bytes.
ssize	is the size of the stack required by the file.

For an explanation of the program segments of a command file, see Section 3, Command File Format.

7.4.3 SIZE68 Examples

This section contains examples of the SIZE68 Utility.

1. The SIZE68 command line specified in this example returns the size of one command file and its program segments.

```
A>sixe68 reloc.68k
reloc.68k:11330+1012+2992=15264 (3BA0) stack size = 0
```

   The program file reloc.68k contains a 11330-byte (decimal) text segment, a 1012-byte (decimal) data segment, and a 2922-byte (decimal) bss. The total size of the program file is 15264 decimal bytes, which is the same as 3BAO hexadecimal bytes. The header in the Reloc.68k file does not specify a minimum stack size. However, when CP/M-68K loads a command file, CP/M-68K always reserves at least 256 bytes for the user stack. CP/M-68K also creates a 256-byte base page. Therefore, to run reloc.68k, the minimum size of the TPA cannot be less than 15776 decimal bytes (15264 bytes for the program, 256 bytes for the stack, and 256 bytes for the base page).

2. The SIZE68 command line specified in this example returns the size of several program files and their program segments.

   A>size68 size68k, dump.68k
   size68k:7010+388+3706=11104 (2B60) stack size = 0
   dump.68k:6964+286+3678=10928 (2AB0) stack size = 0

   When you specify multiple file specifications in a command line, use a comma to delimit each file specification.

3. If you specify a file that is not a common file, SIZE68 returns an error message as shown below.

   A>size68 clink.sub
   Not c.out format: clink.sub

   SIZE68 printed an error message because clink.sub is an ASCII file and not a command file. Files input to SIZE68 must be command files. Refer to Section 3 for the format of CP/M-68K command files.

7.5 SENDC68 Utility

The SENDC68 Utility (SENDC68) creates a file with Motorola S-record format from an absolute command file. S-records are a means of representing an absolute program in ASCII character form. For a detailed description of the S-record format, refer to the CP/M-68K Operating System System Guide.

7.5.1 Invoking SENDC68

You invoke SENDC68 by entering a command in the following format:

SENDC68 [-] input file [output file]

Table 7-8. SENDC68 Command Line Components

Component	Meaning
-	The hyphen is optional. If you specify the hyphen, SENDC68 does not create any S-records for the bss program segment. If you do not specify the hyphen, SENDC68 fills the bss with zeroes. Thus, if you specify the hyphen, SENDC68 creates a smaller S-rccord file.
input file	The file specification for the command file that SENDC68 converts to S-record format. The command file must be an absolute file in the format produced by L068 or RELOC.
output file	The file specification of the SENDC68 output file containing the S-records. If you do not specify a file, SENDC68 sends the S-record that it outputs to the console.

7.5.2 SENDC68 Example

This section contains an example of the SENDC68 command line. The following example illustrates how to create a file that contains Motorola S-records from an absolute command file.

```
A>SEND68 - JUNK.68K JUNK.SR
```

In the above example, SENDC68 creates the S-record file JUNK.SR from the absolute command file JUNK.68K. However, the file JUNK.SR does not contain S-records for the bss program segment.

End of Section 7

Section 8
DDT-68K

8.1 DDT-68K Operation

DDT-68K allows you to test and debug programs interactively in a CP/M-68K environment. You should be familiar with the MC68000 Microprocessor, the assembler (AS68) and the CP/M-68K operating system.

8.1.1 Invoking DDT-68K

Invoke DDT-68K by entering one of the following commands:

```
DDT
DDT filename
```

The first command loads and executes DDT-68K. After displaying its sign-on message and the hyphen (-) prompt character, DDT-68K is ready to accept commands. The second command invokes DDT-68K and loads the file specified by filename. If the filetype is not specified, it defaults to the 68K filetype. The second form of the command is equivalent to the sequence:

```
A>DDT
DDT-68K
Copyright 1982, Digital Research
-Efilename
```

At this point, the program that was loaded is ready for execution.

8.1.2 DDT-68K Command Conventions

When DDT-68K is ready to accept a command, it prompts you with a hyphen (-). In response, you can type a command line or a CONTROL-C (^C) to end the debugging session (see Section 8.1.4). A command line can have as many as 64 characters, and must be terminated with a RETURN. While entering the command, use standard CP/M line-editing functions to correct typing errors. See Table 4-15. DDT-68K does not process the command line until you enter a RETURN.

The first nonblank character of eaich command line determines the command action. Table 8-1 summarizes DDT-68K commands. They are defined individually in Section 8.2.

Table 8-1. DDT-68K Command Summary

Command	Action
D	display memory in hexadecimal and ASCII
E	load program for execution
F	fill memory block with a constant
G	begin execution with optional breakpoints
H	hexadecimal arithmetic
I	set up file control block and command tall
L	list memory using MC68000 mnemonics
M	move memory block
R	read disk file into memory
S	set memory to new values
T	trace program execution
U	untrace program monitoring
V	show memory layout of disk file read
W	write contents of memory block to disk
X	examine and modify CPU state

The command character can be followed by one or more arguments, which may be hexadecimal values, filenames, or other information, depending on the command. Some commands can operate on byte, word, or longword data. The letter W for word or a L for longword must be appended to the command character for commands that operate on multiple data lengths. Details for specific commands are provided with the command descriptions. Arguments are separated from each other by commas or spaces.

8.1.3 Specifying Addresses

Most DDT-68K commands require one or more addresses as operands. All addresses are entered as hexadecimal numbers of up to eight hexadecimal digits (32 bits).

8.1.4 Terminating DDT-68K

Terminate DDT-68K by typing a TC in response to the hyphen prompt. This returns control to the CCP.

8.1.5 DDT-68K Operation with Interrupts

DDT-68K operates with interrupts enabled or disabled, and preserves the interrupt state of the program being executed under DDT-68K. When DDT-68K has control of the CPU, either when it is initially invoked, or when it regains control from the program being tested, the condition of the interrupt mask is the same as It was when DDT-68K was invoked, except for a few critical regions where 'interrupts are disabled. While the program being tested has control of the CPU, the user's CPU state, which can be displayed with the X command, determines the state of the interrupt mask.

Note that DDT-68K uses the Trace and Illegal Instruction exceptions. Therefore, programs debugged under test should not use these.

8.2 DDT-68K Commands

This section defines DDT-68K commands and their arguments. DDT-68K commands give you control of program execution and allow you to display and modify system memory and the CPU state.

8.2.1 The D (Display) Command

The D command displays the contents of memory as 8-bit, 16-bit, or 32-bit hexadecimal values and in ASCII. The forms are:

```
D
Ds
Ds,f
DW
DWs
DWs,f
DL
DLs
DLs,f
```

where s is the starting address, and f is the last address that DDT-68K displays.

Memory is displayed on one or more lines. Each line shows the values of up to 16 memory locations. For the first three forms, the display line appears as follows:

```
aaaaaaaa bb bb ... bb cc ... cc
```

where aaaaaaaa is the address of the data being displayed. The bb's represent the contents of the memory locations in hexadecimal, and the c's represent the contents of memory in ASCII. Any nongraphic ASCII characters are represented by periods.

In response to the Ds form of the D command, shown above, DDT-68K displays 12 lines that start from the current address. Form Ds,f displays the memory block between locations s and f. Forms DW, DWs, and DWs,f are identical to D, Ds, and Ds,f except the contents of memory are displayed as 16-bit values, as shown below:

aaaaaaaa wwww wwww... wwww cccc ... cc

Forms DL, DLs, and DLs,f are identical to D, Ds, and Ds,f except the contents of memory are displayed as 32-bit or longword values, as shown below:

aaaaaaaa 11111111 1111111 ... 11111111 cccccccc ...

During a display, the D command may be aborted by typing any character at the console.

8.2.2 The E (Load for Execution) Command

The E command loads a file in memory so that a subsequent G, T or U command can begin program execution. The syntax for the E command is:

E<filename>

where <filename> is the name of the file to be loaded. If no file type is specified, the filetype 68K is assumed.

An E command reuses memory used by any previous E command. Thus, only one file at a time can be loaded for execution.

When the load is complete, DDT-68K displays the starting and ending addresses of each segment in the file loaded. Use the V command to display this information at a later time.

If the file does not exist or cannot be successfully loaded in the available memory, DDT-68K displays an error message. See Appendix E for error messages returned by DDT-68K.

8.2.3 The F (Fill) Command

The F command fills an area of memory with a byte, word, or longword constant. The forms are

Fs,f,b
FWs,f,w
FLs,f,l

where s is the starting address of the block to be filled, and f is the address of the final byte of the block within the segment specified in s.

In response to the first form, DDT-68K stores the 8-bit value b in locations s through f. In the second form, the 16-bit value w is stored in locations s through f in standard form: the high 8 bits are first, followed by the low 8 bits. In the third form, the 32-bit value I is stored in locations s through f with the most significant byte first.

If s is greater than f, DDT-68K responds with a question mark. Also, if b is greater than FF hexadecimal (255), w Is greater than FFFF hexadecimal (65,535), or I is greater than FFFFFFFF hexadecimal (4,294,967,295), DDT-68K responds with a question mark. DDT-68K displays an error message if the value stored in memory cannot be read back successfully. This error indicates a faulty or nonexistent RAM location.

8.2.4 The G (Go) Command

The G command transfers control to the program being tested, and optionally sets one to ten breakpoints. The forms are

G
G,bl,...bl0
Gs
Gs,bl,...bl0

where s is the address where program begins executing and bl through b IO are addresses of breakpoints.

In the first two forms, no starting address is specified. DDT-68K starts executing the program at the address specified by the program counter (PC). The first form transfers control to your program without setting any breakpoints. The second form sets break- points before passing control to your program. The next two forms are analogous to the first two except that the PC is first set to S.

Once control has been transferred to the program under test, it executes in real time until a breakpoint is encountered. At this point, DDT-68K regains control, clears all breakpoints, and displays the CPU state in the same form as the X command. When a breakpoint returns control to DDT-68K, the instruction at the breakpoint address has not yet been executed. To set a breakpoint at the same address, you must specify a T or U command first.

8.2.5 The H (Hexadecimal Math) Command

The H command computes the sum and difference of two 32-bit values. The form is:

Ha,b

where a and b are the values whose sum and difference DDT-68K computes. DDT-68K displays the sum (ssssssss) and the difference (dddddddd) truncated to 32 bits on the next line:

ssssssss dddddddd

8.2.6 The I (Input Command Tall) Command

The I command prepares a file control block (FCB) and command tall buffer in the base page of the last file loaded with the E command. The form is

I<command tall>

where <command tall> is the character string which usually contains one or more filenames. The first filename is parsed into the default file control block at 005CH. The optional second filename, if specified, is parsed into the second default file control block beginning at 0038H. The characters in the <command tall> are also copied to the default command buffer at 0080H. The length of the <command tall> is stored at 0080H, followed by the character string terminated with a binary zero.

If a file has been loaded with the E command, DDT-68K copies the file control block and command buffer from the base page of DDT-68K to the base page of the program loaded.

8.2.7 The L (List) Command

The L command lists the contents of memory in assembly language. The forms are

```
L
Ls
Ls,f
```

where s is the starting address, and f is the last address in the list.

The first form lists 12 lines of disassembled machine code from the current address. The second form sets the list address to s and then lists 12 lines of code. The last form lists disassembled code from s through f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. When DDT-68K regains control from a program being tested (see G, T and U commands), the list address is set to the address in the program counter (PC).

Long displays can be aborted by typing any key during the list process. Or, enter CONTROL-S (TS) to halt the display temporarily. A CONTROL-Q (IQ) restarts the display after TS halts it.

The syntax of the assembly language statements produced by the L command is described in the Motorola 16-Bit Microprocessor User's Manual, third edition, MC68000UM(AD3).

8.2.8 The M (Move) Command

The M command moves a block of data values from one area of memory to another. The form is

```
Ms,f,d
```

where s is the starting address of the block to be moved, f is the address of the final byte to be moved, and d 'is the address of the first byte of the area to receive the data. Note that if d is between s and f, part of the block being moved will be overwritten before it is moved, because data is transferred starting from location s.

8.2.9 The R (Read) Command

The R command reads a file to a contiguous block in memory. The format is

R<filename> is

where <filename> is the name and type of the file to be read.

DDT-68K reads the file into memory and displays the starting and ending addresses of the block of memory occupied by the file. A Value (V) command can redisplay the information at a later time. The default display pointer (for subsequent Display (D) commands) is set to the start of the block occupied by the file.

8.2.10 The S (Set) Command

The S command can change the contents of bytes, words, or longwords in memory. The forms are

Ss
SWs
SLs

where s is the address where the change is to occur.

DDT-68K displays the memory address and its current contents on the following line. In response to the first form, the display is

aaaaaaaa bb

In response to the second form, the display is

aaaaaaaa wwww

In response to the third form, the display is

aaaaaaaa 11111111

where bb, wwww, and 11111111 are the contents of memory in byte, word, and longword formats, respectively.

In response to one of the above displays, you can alter the memory location or leave it unchanged. If a valid hexadecimal value is entered, the contents of the byte, word, or longword in memory is replaced with the value entered. If no value is entered, the contents of memory are unaffected and the contents of the next address are displayed. In either case, DDT-68K continues to display successive memory addresses and values until either a period or an invalid value is entered.

DDT-68K displays an error message if the value stored in memory cannot be read back successfully. This error indicates a faulty or nonexistent RAM location.

8.2.11 The T (Trace) Command

The T command traces program execution for I to OFFFFFFFFH program steps. The forms are

```
T
Tn
```

where n is the number of instructions to execute before returning control to the console.

After DDT-68K traces each instruction, it displays the current CPU state and the disassembled instruction in the same form as the X command display.

Control transfers to the program under test at the address indicated in the PC. If n is not specified, one instruction is executed. Otherwise, DDT-68K executes n instructions and displays the CPU state after each step. You can abort a long trace before all the steps have been executed by typing any character at the console.

After a Trace (T) command, the list address used in the L command is set to the address of the next instruction to be executed.

Note that DDT-68K does not trace through a BDOS interrupt instruction, since DDT-68K itself makes BDOS calls and the BDOS is not reentrant. Instead, the entire sequence of instructions from the BDOS interrupt through the return from BDOS is treated as one traced instruction.

8.2.12 The U (Untrace) Command

The U command is 'identical to the Trace (T) command except that the CPU state is displayed only after the last instruction is executed, rather than after every step. The forms are

```
u
Un
```

where n 'is the number of instructions to execute before control returns to the console. You can abort the Untrace (U) command before all the steps have been executed by typing any key at the console.

8.2.13 The V (Value) Command

The V command displays information about the last file loaded with the Load For Execution (E) or Read (R) commands. The form is

```
V
```

If the last file was loaded with the E command, the V command displays the starting address and length of each of the segments contained in the file, the base page pointer, and the initial stack pointer. The format of the display 'is

```
Text base=00000500 data base=00000B72 bss base=000003FDA
text length=00000672 data length=00003468 bss length=0000A1B0
base page address=00000400 initial stack pointer=000066D4
```

If no file has been loaded, DDT-68K responds to the V command with a question mark (?).

8.2.14 The W (Write) Command

The W command writes the contents of a contiguous block of memory to disk. The forms are

```
W<filename>
W<filename>,s,f
```

The <filename> is the file specification of the disk file that receives the data. The letters s and f are the first and last addresses of the block to be written. If f does not specify the last address, DDT-68K uses the same value that was used for s.

If the first form is used, DDT-68K assumes the values for s and f from the last file read with a R command. If no file is read by an R command, DDT-68K responds with a question mark (?). This form is useful for writing out files after patches have been installed, assuming the overall length of the file is unchanged.

If the file specified in the W command already exists on disk, DDT-68K deletes the existing file before it writes the new file.

8.2.15 The X (Examine CPU State) Command

The X command displays the entire state of the CPU, including the program counter (PC), user stack pointer (usp), system stack pointer (ssp), status register (by field), all eight data registers, all eight address registers, and the disassembled instruction at the memory address currently in the PC. The forms are

```
X
Xr
```

where r is one of the following registers:

D0 to D7, A0 to A7, PC, USP, or SSP

The first form displays the CPU state as follows:

```
PC=0016000 USP=00001000 SSP=00002000 ST=FFFF=>(etc.)
D 00001000 00000D01 ... 00000001
A 00B0A00 00A0010 ... 00000000
```

The first line includes:

PC	Program Counter
USP	User Stack Pointer
SSP	System Stack Pointer
ST	Status Register

Following the Status Register contents on the first display line, the values of each bit in the Status Register are displayed, as shown in the following sample:

```
TR SUP IM=7 EXT NEG ZER OFL CRY
```

This sample display includes:

TR	Trace Bit
SUP	Supervisor Mode Bit
IM=7	Interrupt Mask = 7
EXT	Extend
NEG	Negative
ZER	Zero
OFL	Overflow
CRY	Carry

The second form, Xr, allows you to change the value in the registers of the program being tested. The r denotes the register. DDT-68K responds by displaying the current contents of the register, leaving the cursor on that line. If you type a RETURN, the value is not changed. If you type a new valid value and a RETURN, the register is changed to the new value. The contents of all registers except the Status Register can be changed.

8.3 Assembly Language Syntax for the L Command

In general, the syntax of the assembly language statements used in the L command is standard Motorola 68000 assembly language. Several minor exceptions are given in the following list:

- o DDT-68K prints all numeric values in hexadecimal.
- o DDT-68K uses lower-case mnemonics.
- o DDT-68K assumes word operations unless a byte or longword specification is explicitly stated.

End of Section 8

Appendix A
Summary of BIOS Functions

Table A-1 lists the BIOS functions supported by CP/M-68K. For more details on these functions, refer to the CP/M-68K Operating System System Guide.

Table A-1. Summary of BIOS Functions

Function	F#	Description
Init	0	Called for Cold Boot
Warm Boot	1	Called for Warm Start
Const	2	Check for Console Character Ready
Conin	3	Read Console Character In
Conout	4	Write Console Character Out
List	5	Write Listing Character Out
Auxiliary Output	6	Write Character to Auxiliary Output Device
Auxiliary Input	7	Read from Auxiliary Input Device
Home	8	Move to Track 00
Seldsk	9	Select Disk Drive
Settrk	10	Set Track Number
Setsec	11	Set Sector Number
Setdma	12	Set DMA Offset Address
Read	13	Read Selected Sector
Write	14	Write Selected Sector
Listst	15	Return List Status
Sectran	16	Sector Translate
Get Memory Region Table Address	18	Address of Memory Region Table
Get I/O Byte	19	Get PO Mapping Byte
Set I/O Byte	20	Set PO Mapping Byte
Flush Buffers	21	Writes Modified Buffers
Set Exception Vector	22	Sets Exception Vector

End of Appendix A

Appendix B
Transient Program Load Examples

This appendix contains two examples, an assembly language program and aC language program. Both illustrate how a transient program loads another program with the BDOS Program Load Function (59) but without the CCP

Examples:

1. The following example is an AS68 assembly language program that loads another program into the TPA.

```
*         BDOS Function Definitions
*
reboot          =        0
printstr        =        9
open            =       15
setdma          =       26
pgmldf          =       59
gettpa          =       63
                .text

*
*         OPEN file to be  loaded
*
start:          link         a6,$0            *mark stack frame
                move.l       8(a6) ,a0        *get the address of the base page
                lea          $5c(a0) ,al      *get address of  1st parsed FCB in base page
                move.l       al ,dl           *put that address in  register dl
                move.w       #open d0         *put BDOS function number in  register d0
                trap         #2               *try to open the file to be  loaded
                cmpi         #255,d0          *test d0 for BDOS error return code
                beq          openerr          *if d0 = 255 then goto openerr
*
*          Compute Address to Load File
*
```

Listing B-1. Transient Program Load Example 1

```
        move .1     $18 (a0),d2     *get starting address of bss from base page
        move.l      $1c(a0),d3      *get length of bss
        add.1       d2 ,d3          *compute first free byte of memory
                                    *after bss
        move.l      $20(a0),d4      *get length of free memory after bss
        sub         #100 ,d4        *leave some extra room
        move.l      d4 ,d5          *save that length in register d5
        add .1      d3 ,d4          *compute high end of free memory after bss
        move.l      d3 ,a3          *get the starting address of free memory
                                    *into a3
        sub         #l ,d5          *adjust loop counter
clear:  c1r.b       (a3)+           *clear out free memory
        dbf         d5 ,clear       *dec rement loop counter and loop until
*                                   *negative
*
*       FILL the LPB
*       Low address becomes first free byte of memory after bss
*       High address of area in which to load program becomes
*       the Low address plus length of free memory
*
*
        move.l      d3 ,lowadr      *get low end of area in which to load
*                                   *program
        move.l      d4,hiadr        *get high end of area in which to load
*                                   *program
        move.l      al ,LPB         *put add ress of open FCB into LPB
        move .w     #pgmldf ,d0     *get BDOS program load function
                                    *number
        move.l      #LPB,dl         *put address of LPB into register dl
        trap        #2              *do the program load
        tst         #d0             *was the load successful?
        bne         lderr           *if not then print error message
*
*       Set default DMA address
*
        move.l      baspag,dl       *dl points to new program's base page
        add         #$80 ,dl        *d1 points to default dma in base page
        move .w     #setdma,dO      *get BDOS function numbs r
        trap        #2              *set the default dma address
```

Listing B-1. (continued)

```
*
*
*       Now push needed addresses on stack
*
                movea .1        usrstk ,a?              *set up user stack pointer
                move.1          baspag,al               *get address of base page
                move.l          al ,-(sp)               *push base page address
                move .1         #cmd rtn ,- Ccp )       *push return address
                move,1          8(al) ,-(sp)            *push address to jump to
                rts                                     *jump to new program
*
*       Print ERROR message
*
openerr:
                move .1         #openmsg,dl             *get address of error message
*                                                       *to be printed
                bra             print
lderr:          move,l          #loaderr,dl             *get address of error message to
                                                        *be printed
print:          move .w         #printst r ,d0          *get BOOS function numbs r
                trap    #2                              *print the message
cmdrtn :        move, w         a reboot ,d0            *get BOOS function numbs r
                trap    #2                              *warmboot and  return to the CCP
*
*               DATA
*
                .dat a
                .even
*
```

Listing B-1. (continued)

```
*           LOAD PARAMETER BLOCK
*
LPB:        .ds.l      1          *FCB  address of program rile
lowadr:     .ds.1      1          *Low  boundary  of  area  in which
*                                 *to  load  program
hiadr:      .ds.1      1          *Hig'h  boundary  of  area  in which  to
*                                 *to  load  program
baspag:     .ds.1                 *Base  pox's  address  of  loaded  program
usrstk:     .ds.1      1          *Loaded  program's  initial  stack  pointer
flags:      .dc.w      0          *Load  program function  control  flax's
*
*           TPA Parameter    Block
*
            .even
TPAB:       .dc.w      0
low:        .ds.1      1
high:       .ds.1      1

            .even

loaderr:    .dc.b   13,10,'Program Load Error$'
openmag:    .dc.b   13.10. 'Unable  to Open File$'
            .end
```

Listing B-1. (continued)

2. The following example is a C language transient program that loads anotherprogram in the TPA without the assistance of the CCP The C language programcalls an AS68 assembly language routine to perform tasks not permitted by theC language.

```
/*-----------------------------------------------------------------------------------------*
|                                                                                         |
|                'C'  Language  Program to Load Anothe r                                  |
|                     Program  into  the TPA                                              |
|                                                                                         |
*-----------------------------------------------------------------------------------------*/

/*      DEFINES       */

#define         BSS__OFFSET        (long)0x18
#def ine        FCB__OFFSET        (long)0x5C
#define         BSS__LENCTH        (long)0x1C
#define         FREE__MEMORY       (long)0x20
#define         DMA__OFFSET        (long)0x80
#define         ROOM               (long)0x100
#define         NULL               '0'
#define         CR                 (long)13
#define         LF                 (long)10
#define         REBOOT             0
#define         CON__OUT           2
#define         PRINTSTR           9
#define         OPEN               15
#define         SETOMA             26
#define         PCMLDF             59
#define         GETTPA             63
```

Listing B-2. Transient Program Load Example 2

```
/* Error Messages       */

char    openmsg[20] = "Unable  to Open Fi le$" ;
char    loadmsg[19] = "Program Load Error$" ;

/* Load  Parameter Block */

extern  long LPB ,lowadr ,hiadr,baspag ,usrgtk ;
extern  int  flags ;

/* TPA Parameter Block */

extern  int TPAB;
extern  long  low ,high ;
```

Listing B-2. (continued)

```
openfile(baseaddr)                          /*****************************/
resister char    *baseaddr;                 /* base pase address          */
{                                           /*                            */
        register long   *tl ,*t2;           /* Pointers to Ions word values */
        register long   count ;             /* long word value            */
        register char   *ptrl ,*ptr2;       /* Pointers to character values */
                                            /*                            */
                                            /*                            */
        ptrl = baseaddr + FCB_OFFSET;       /* get address of FCB         */
        if(bdos(OPEN,ptrl) <= 3)            /* try to open the file       */
        {                                   /*                            */
                t1 = baseaddr +             /* set pointer to STARTING addr */
                        BSS_OFFSET;         /* of the BSS segment         */
                t2 = baseaddr +             /* set pointer to LENGTH of   */
                        BSS_LENGTH;         /* the BSS segment            */
                lowadr = *tl + *t2;         /* compute the first free byte */
                                            /* address of memory after the */
                                            /* BSS segment                */
                ptr2 = lowadr               /* *ptr2 now points to first  */
                                            /* free byte in memory        */
                t2 = baseaddr +             /* get length of free memory  */
                        FREE_MEMORY;        /* after the BES segment      */
                                            /*                            */
                hiadr = *t2 + lowadr        * compute high end ofavailable */
                                            /* memory                     */
                count = *t2 - ROOM          /* Leave some extra room in Mem */
                while(count--)              /* Clear out available Memory */
                        *ptr2++ = NULL;     /* with NULL byte values      */
                LPB = ptrl ;                /* first long of parameter blk */
                                            /* gets the address of the FCB */
                                            /*****************************/
                /*-------------------------------------------------------*
                |       If the Load is Successful                        |
                |       1.      Set the Default DMA address              |
                |       2.      Call Assembly Code to push               |
                |               the base page address, the               |
                |               return address, and the                  |
                |               address you wish to Jump to.             |
                *-------------------------------------------------------*/

        if(bdos(PGMLDF,&LPB)  == 0)
        {
                bdos(SETDMA,(baspag + DMA=OFFSET)) ;
                push( ) ;
        }
```

```
        else
                error(PGMLDF) ;
        }
        else
                error(OPEN) ;
}
error(flaS)
int     flag;
{
        bdos(CON_OUT,CR) ;
        bdos(CON_OUT,LF) ;
        if(flag == OPEN)
                bdos(PRINTSTR,openmsg) ;
        else
                bdos(PRINTSTR,loadmsg);
        bdos(REBDOT,(lons)0) ;
}

main( )
{
        bdos(REBOOT,(long)0);
}
*****************************************************
*                                                   *
*       Assembly Language Module Needed  to         *
*       Assist  'C'  code to Load  a Program intothe TPA   *
*                                                   *
*****************************************************
*
*       Make All  these  labels GLOBAL
*
        .globl          _bdos
        .globl          _LPB
        .globl          _lowadr
        .globl          _hiadr
        .globl          _baspag
        .globl          _usrstk
        .globl          _flags
        .globl          _TPAB
        .globl          _low
        .globl          _high
        .globl          _start
        .glob           _openfile
        .globl          _push
        .globl          _main
```

```
*
*       Get the address of the base page
*
__start:
        link        a6 ,#0          *link and allocate
        move.1      8(a6) ,-(sp)    *push the address of the  base page
        jsr         __openfile      *jump to  'C' code to open  the file
*
*       Call the BOOS
*
__bdos:
        move.w      4(sp) ,d0       *get  the BDOS function number
        move.1      6(sp) ,dl       *get  the BDOS parameter
        trap        #2              *call  the BDOS
        rts                         * return
*
*       Push the needed addresses on to the stack
*
__push:
        movea .1    __usrstk ,a7    *set  up the user stack  pointer
        move .1     __baspag ,al    *get  address of  user base  page
        move. 1     al ,- ( sp)     *push base page add ress
        move .1     #__main,-(sp)   *push  return   add ress
        move. 1     8(al) ,- (sp)   *push  add ress  to  jump to
        rts                         *jump  to new program
*
*       DATA
        .data
        .even,

*
*       Load  Parameter      Block
*
__LPB:        .ds.l   1             *FCB address  of  program file
__lowadr:     .ds.l   1             *Low boundary  of  area  in  which
*                                   *to load  program
__hiad r      .ds.l   1             *High boundary of area  in which  to
*                                   *to load program
__baspag      .ds.l   1             *Base  page  address  of  loaded  pro gram
__usrstk:     .ds.l   1             *loaded program's  initial  stack pointer
__flags:      .dc.w   0             *Load  program function  control  flags
*
*       TPA Parameter Block
*
```

```
          .even

__TPAB:        .dc.w   0
__low:         .ds.l   1
__high :       .ds.l   1
*
*         END of Assembly Language Code
*
          .end
```

End of Appendix B

Appendix C
Base Page Format

Table C-1 shows the format of the base page. The base page describes a program's environment. The Program Load Function (59) allocates space for a base page when this function is invoked to load an executable command file. For more details, on the Program Load Function and command files, refer to the appropriate sections in this manual.

Table C-1. Base Page Format: Offsets and Contents

0ffset	Contents
0000 - 0003	Lowest address of TPA (from LPB)
0004 - 0007	1 + Highest address of TPA (from LPB)
0008 - 000B	Starting address of the Text Segment
000C - 000F	Length of Text Segment (bytes)
0010 - 0013	Starting address of the Data Segment (initialized data)
0014 - 0017	Length of Data Segment
0018 - 001B	Starting address of the bss (uninitialized data)
001C - 001F	Length of bss
0020 - 0023	Length of free memory after bss
0024 - 0024	Drive from which the program was loaded

Table C-1. (continued)

Offset	Contents
0025 - 0037	Reserved, unused
0038 - 005B	2nd parsed FCB from Command Line
005C - 007F	lst parsed FCB from Command Line
0080 - 00FF	Command Tail and Default DMA Buffer

End of Appendix C

Appendix D
Instruction Set Summary

This appendix contains two tables that describe the assembler instruction set distributed with CP/M-68K. Table D-1 summarizes the assembler (AS68) instruction set. Table D-2 lists variations on the instruction set listed in Table D-1. For details on specific instructions, refer to Motorola's 16-Bit Microprocessor User's Manual, third edition, MC68000UM(AD3).

Table D-1. Instruction Set Summary

Instruction	Description
abcd	Add Decimal with Extend
add	Add
and	Logical AND
asl	Arithmetic Shift Left
asr	Arithmetic Shift Right
bcc	Branch Conditionally
bchg	Bit Test and Change
bclr	Bit Test and Clear
bra	Branch Always
bset	Branch Test and Set
bsr	Branch to Subroutine
btst	Bit Test
chk	Check Register Against Bounds
clr	Clear Operand
cmp	Compare
dbcc	Test Condition, Decrement and Branch
divs	Signed Divide
divu	Unsigned Divide

Table D-1. (continued)

Instruction	Description
eor	Exclusive Or
exg	Exchange Registers
ext	Sign Extend
jmp	jump
jsr	jump to Subroutine
lea	Load Effective Address
link	Link Stack
lsl	Logical Shift Left
lsr	Logical Shift Right
move	Move
movem	Move Multiple Registers
movep	Move Peripheral Data
muls	Signed Multiply
mulu	Unsigned Multiply
nbcd	Negate Decimal with Extend
neg	Negate
nop	No Operation
no	Ones Complement
or	Logical OR
pea	Push Effective Address
reset	Reset External Devices
rol	Rotate Left without Extend
ror	Rotate Right without Extend
roxl	Rotate Left with Extend
roxr	Rotate Right with Extend
rte	Return from Exception
rtr	Return and Restore
rts	Return from Subroutine

Table D-1. (continued)

Instruction	Description
sbcd	Subtract Decimal with Extend
scc	Set Conditional
stop	Stop
sub	Subtract
swap	Swap Data Register Halves
tas	Test and Set Operand
trap	Trap
trapv	Trap on Overflow
tst	Test
unlk	Unlink

Table D-2. Variations of Instruction Types

Instruction	Variation	Description
add	add	Add
	adda	Add Address
	addq	Add Quick
	addi	Add Immediate
	addx	Add with Extend
and	and	Logical AND
	andi	AND Immediate
	andi to ccr	AND Immediate to Condition Code
	andi to sr	AND Immediate to Status Register
cmp	cmp	Compare
	cmpa	Compare Address
	cmpm	Compare Memory
	cmpi	Compare lmmediate
eor	cor	Exclusive OR
	eori	Exclusive OR Immediate
	cori to ccr	Exclusive Immediate to Condition Codes
	eori to sr	Exclusive OR Immediate to Condition Codes
move	move	Move
	movea	Move Address
	moveq	Move Quick
	move to ccr	Move to Condition Codes
	move to sr	Move to Status Register
	move from sr	Move from Status Register
	move to usp	Move to User Stack Pointer
neg	neg	Negate
	negx	Negate with Extend
or	or	Logical OR
	ori	OR Immediate
	ori to ccr	OR Immediate to Condition Codes
	ori to sr	OR Immediate to Status Register

Table D-2. (continued)

Instrucion	Variation	Description
sub	sub	Subtract
	suba	Subtract Address
	subi	Subtract immediate
	subq	Subtract Quick
	subx	Subtract with Extend

End of Appendix D

Appendix E
Error Messages

This appendix lists the error messages returned by the internal components of CP/M-68K and by the CP/M-68K programmer's utilities. The sections arc arranged alphabetically by the name of the internal component or utility. The error messages are listed alphabetically within each section, with explanations and suggested user responses.

E.1 AR68 Error Messages

The CP/M-68K Archive Utility, AR68, returns two types of fatal error messages: diagnostic and logic. Both types of fatal error messages are returned at the console as they occur.

E.1.1 Fatal Diagnostic Error Messages

The AR68 errors are listed in Table E-1 in alphabetic order with explanations and suggested user responses.

Table E-1. AR68 Fatal Diagnostic Error Messages

Message	Meaning
filename not in srchive file	The object module indicated by the variable "filename" is not in the library. Check the filename before you reenter the command line.
cannot create filename	The drive code for the file indicated by the variable "filename" is invalid, or the disk to which AR68 is writing is full. Check the drive code. If it is valid, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the command line.

Table E-1. (continued)

Message	Meaning
cannot open filename	The file indicated by the variable "filename" cannot be opened because the filename or the drive code is incorrect. Check the drive code and the filename before you reenter the command line.
invalid option flag: x	The symbol, letter, or number in the command line indicated by the variable x is an invalid option. Refer to the section of this manual on AR68 for an explanation of the command line options. Specify a valid option and reenter the command line.
not archive format: filename	The file indicated by the variable "filename" is not a library. Ensure that you are using the correct filename before you reenter the command line.
not object file: filename	The file indicated by the variable 'filename" is not an object file, and cannot be added to the library. Any file added to the library must be an object file, output by the assembler, AS68, or the compiler. Assemble or compile the file before you reenter the AR68 command line.
one and only one of DRTWX flags required	The AR68 command line requires one of the D, R, T, W, or X commands, but not more than one. Reenter the command line with the correct command. Refer to the section of this manual on AR68 for an explanation of the AR68 commands.
filename not in library	The object module indicated by the variable "filename" is not in the library. Ensure that you are requesting the filename of an existing object module before you reenter the command line.

Table E-1. (continued)

Message Meaning

Read error on filename

The file indicated by the variable "filename" cannot be read. This message means one of three things: the file listed at "filename" is corrupted; a hardware error has occurred; or when the file was created, it was not correctly written by AR68 due to an error in the internal logic of AR68.

Cold start the system and retry the operation. If you receive this error message again, you must erase and recreate the file. Use your backup file, if you maintained one. If the error reoccurs, check for a hardware error. If the error persists, contact the place you purchased your system for assistance. You should provide the following information:

o Indicate which version of the operating system you are using.
o Describe your system's hardware configuration.
o Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

temp file write error

The disk to which AR68 was writing the temporary file is full. Erase unnecessary files, if any, or insert a new disk before you reenter the command line.

usage:AR68 DRTWX[AV][FD:][OPMOD] ARCHIVE OBMOD1 [OBMOD2...][>FILESPEC]

This message indicates a syntax error in the command line. The correct format for the command line is given, with the possible options in brackets. Refer to the section in this manual on AR68 for a more detailed explanation of the command line.

Table E-1. (continued)

Message	Meaning
Write error on filename	The disk to which AR68 is writing the file indicated by the variable "filename" is full. Erase unnecessary files, if any, or insert a new disk before you reenter the command line.

E. 1.2 AR68 Internal Logic Error Messages

This section lists messages indicating fatal errors in the internal logic of AR68. If you receive one of these messages, contact the place you purchased your system for assistance. You should provide the following information:

1. Indicate which version of the operating system you are using.
2. Describe your system's hardware configuration.
3. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

cannot reopen filename

seek error on library

Seek error on tempname

Unable to re-create--library is in filename

Note: for the above error, "Unable to re-create-- library is in filename", you should rename the temporary file indicated by the variable "filename". AR68 used the library to create the temporary file and then deleted the library in order to replace it with the updated temporary file. This error occurred because AR68 cannot write the temporary file back to the original location. The entire library is in the temporary file.

E-2 AS68 Error Messages

The CP/M-68K assembler, AS68, returns both nonfatal, diagnostic error messages and fatal error messages. Fatal errors stop the assembly of your program. There are two types of fatal errors: user-recoverable fatal errors and fatal errors in the internal logic of AS68.

E.2.1 AS68 Diagnostic Error Messages

Diagnostic messages report errors in the syntax and context of the program being assembled without interrupting assembly. Refer to the Motorola 16-Bit Microprocessor User's Manual for a full discussion of the assembly language syntax.

Diagnostic error messages appear in the following format:

& line no. error message text

The ampersand (&) indicates that the message comes from AS68. The "line no." indicates the line in the source code where the error occurred. The "error message text" describes the error. Diagnostic error messages are printed at the console after assembly, followed by a message indicating the total number of errors. In a printout, they are printed on the line preceding the error. The AS68 diagnostic error messages are listed in Table E-2 in alphabetic order.

Table E-2. AS68 Diagnostic Error Messages

Message	Meaning
& line no. backward assignment to *	The assignment statement in the line indicated illegally assigns the counter (*) backward. Change the location counter to a forward assignment and reassemble the source file.
& line no. bad use of symbol	A symbol in the source line indicated has been defined as both global and common. A symbol can be either global or common, but not both. Delete one of the directives and reassemble the source file.

Table E-2. (continued)

Message	Meaning
& line no.	constant required An expression on the line indicated requires a constant. Supply a constant and reassemble the source file.
& line no.	and statement not at end of source The end statement must be at the end of the source code. The end statement cannot be followed by a comment or more than one carriage return. Place the end statement at the end of the source code, followed by a single carriage return only, and reassemble the source file.
& line no.	illegal addressing mode The instruction on the line Indicated has an invalid addressing mode. Provide a valid addressing mode and reassemble the source file.
& line no.	illegal constant The line indicated contains an illegal constant. Supply a valid constant and reassemble the source file.
& line no.	illegal expr The line indicated contains an illegal expression. Correct the expression and reassemble the source file.
& line no.	illegal external The line indicated illegally contains an external reference to an 8-bit quantity. Rewrite the source code to define the reference locally or use a 16-bit reference and reassemble the source file.
& line no.	illegal format An expression or instruction in the line indicated is illegally formatted. Examine the line. Reformat where necessary and reassemble the source file.
& line no.	illegal index register The line indicated contains an invalid index register. Supply a valid register and reassemble the source file.

Table E-2. (continued)

Message	Meaning
& line no.	illegal relative address An addressing mode specified is not valid for the instruction in the line indicated. Refer to the Motorola 16-Bit Microprocessor User's Manual for valid register modes for the specified instruction. Rewrite the source code to use a valid mode and reassemble the file.
& line no.	illegal shift count The instruction in the line indicated shifts a quantity more than 31 times. Modify the source code to correct the error and reassemble the source file.
& line no.	illegal size The instruction in the line indicated requires one of the following three size specifications: b (byte), w (word), or I (longword). Supply the correct size specification and reassemble the source file.
& line no.	illegal string The line indicated contains an illegal string. Examine the line. Correct the string and reassemble the source file.
& line no.	illegal text delimiter The text delimiter in the line indicated is in the wrong format. Use single quotes ('text') or double quotes ("text") to delimit the text and reassemble the source file.
& line no.	illegal 8-bit displacement The line indicated illegally contains a displacement larger than 8-bits. Modify the code and reassemble the source file.
& line no.	illegal 8-bit immediate The line indicated illegally contains an immediate operand larger than 8-bits. Use the 16- or 32-bit form of the instruction and reassemble the source file.

Table E-2. (continued)

Message	Meaning
& line no.	illegal 16-bit displacement The line indicated illegally contains a displacement larger than 16-bits. Modify the code and reassemble the source file.
& line no.	illegal 16-bit immediate The line indicated illegally contains an immediate operand larger than 16-bits. Use the 32-bit form of the instruction and reassemble the source file.
& line no.	invalid data list One or more entries in the data list in the line indicated is invalid. Examine the line for the invalid entry. Replace it with a valid entry and reassemble the source file.
& line no.	invalid first operand The first operand in an expression in the line indicated is invalid. Supply a valid operand and reassemble the source file.
& line no.	invalid instruction length The instruction in the line indicated requires one of the following three size specifications: b (byte), w (word), or I (longword). Supply the correct size specification and reassemble the source file.
& line no.	invalid label A required operand is not present in the line indicated, or a label reference in the line is not in the correct format. Supply a valid label and reassemble the source file.
& line no.	invalid opcode The opcode in the line indicated is nonexistent or invalid. Supply a valid opcode and reassemble the source file.
& line no.	invalid second operand The second operand in an expression in the line indicated is invalid. Supply a valid operand and reassemble the source file.

Table E-2. (continued)

Message	Meaning
& line no. label redefined	This message Indicates that a label has been defined twice. The second definition occurs in the line indicated. Rewrite the source code to specify a unique label for each definition and reassemble the source file.
& line no. missing)	An expression in the line indicated is missing a right parenthesis. Supply the missing parenthesis and reassemble the source file.
& line no. no label for operand	An operand in the line indicated is missing a label. Supply a label and the source file.
& line no. opcode redefined	A label in the line indicated has the same mnemonics as a previously specified opcode. Respecify the label so that it does not have the same spelling as the mnemonic for the opcode. Reassemble the source file.
& line no. register required	The instruction in the line indicated requires either a source or destination register. Supply the appropriate register and reassemble the source file.
& line no. relocation error	An expression in the line indicated contains more than one externally defined global symbol. Rewrite the source code. Either make one of the externally defined global symbols a local symbol, or evaluate the expression within the code. Reassemble the source file.
& line no. symbol required	A statement in the line indicated requires a symbol. Supply a valid symbol and reassemble the source file.

Table E-2. (continued)

Message	Meaning
& line no. undefined symbol in equate	One of the symbols in the equate directive in the line indicated is undefined. Define the symbol and reassemble the source file.
& line no. undefined symbol	The line indicated contains an defined symbol that has no been declared global. Either define the symbol within the module or define it as a global symbol and reassemble the source file.

E.2.2 User-recoverable Fatal Error Messages

The fatal error messages for AS68 are described in Table E-3. When an error occurs because the disk is full, AS68 creates a partial file. You should erase the partial file to ensure that you do not try to link it.

Table E-3. User-recoverable Fatal Error Messages

Message	Meaning
& cannot create init: AS68SYMB.DAT	AS68 cannot create the initialization file because the drive code is incorrect or the disk to which it was writing the file is full. If you used the -S switch to redirect the symbol table to another disk, check the drive code. If it is correct, the disk is full. Erase unnecessary files, 'if any, or insert a new disk before you reinitialize AS68. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& expr opstk overflow	An expression in the line indicated contains too many operations for the operations stack. Simplify the expression before you reassemble the source code.
& expr tree overflow	The expression tree does not have space for the number of terms in one of the expressions in the indicated line of source code. Rewrite the expression to use fewer terms before you reassemble the source file.

Table E-3. (continued)

Message	Meaning
& I/O error on loader output file	The disk to which AS68 was writing the loader output file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& I/O write error on it file.	The disk to which AS68 was writing the intermediate text file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& it read error itoffset-no.	The disk to which AS68 was writing the intermediate text file is full. AS68 wrote a partial file. The variable Itpffset=no. indicates the first zero-relative byte number not read. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& Object file write error	The disk to which AS68 was writing the object file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& overflow of external table	The source code uses too many externally defined global symbols for the size of the external symbol table. Eliminate some externally defined global symbols and reassemble the source file.
& Read Error on Intermediate File: ASXXXXn	The disk to which AS68 was writing the intermediate text file ASXXXX is full.AS68 wrote a partial file. The variable "n" indicates the drive on which ASXXXX is located. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

Table E-3. (continued)

Message Meaning

& symbol table overflow
he program uses too many symbols for the symbol table. Eliminate some symbols before you reassemble the source code.

& Unable to open file filename
The source filename indicated by the variable "filename" is invalid or, has an invalid drive code or user number. Check the filename, drive code, and user number. Respecify the command line before you reassemble the source file.

& Unable to open input file
The filename in the command line indicated does not exist, or has an invalid drive code or user number. Check the filename, drive code, and user number. Respecify the command line before you reassemble the source file.

& Unable to open temporary file
Invalid drive code or the disk to which AS68 was writing is full. Check the drive code. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reassemble the source file.

& Unable to open init file: AS68SYMB.DAT
The drive code or user number used to specify the initialization file is invalid or the assembler has not been initialized. Check the drive code and user number. Respecify the command line before you reassemble the source file. If the assembler has not been initialized. refer to the section in this manual on AS68 for instructions.

& Write error on init file: AS68SYMB.DAT
The disk to which AS68 was writing the initialization file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

Table E-3. (continued)

Message	Meaning
& write error on it file	The disk to which AS68 was writing the intermediate text is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk. Erase the partial file that was created on the full disk to ensure that you do not try to link it. Reassemble the source file.

E.2.3 Internal Logic Error Messages

This section lists messages indicating fatal errors in the internal logic of AS68. If you receive one of these messages, contact the place you purchased your system for assistance. You should provide the following information.

1. Indicate which version of the operating system you are using.
2. Describe your system's hardware configuration.
3. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

Errors:

& doitrd: buffer bitch pitix=nnn itbuf-nnn end=nnn

& doitwr: it buffer botch

& invalid radix in oconst

& i.t. overflow

& it sync error itty=nnn

& seek error on it file

& outword: bad rlflg

E.3 BDOS Error Messages

The CP/M-68K Basic Disk Operating System, BDOS, returns fatal error messages at the console. The BDOS error messages are listed in Table E-4 in alphabetic order with explanations and suggested user responses.

Table E-4. BDOS Error Messages

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

Table E-4. (continued)

Message	Meaning
CP/M Disk read error on drive x Do you want to: Abort (A), Retry, (R), or Continue with bad data (C)?	
	BDOS. This message indicates a hardware error. Specify one of the options enclosed in parentheses. Each option is described below.

Option	Action
A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.
R	Retries the operation. If the retry falls, the system reprompts with the option message.
C	Ignores the error and continues program execution. Be careful if you use this option. Program execution should not be continued for some types of programs. For example, if you are updating a data base and receive this error but continue program execution, you can corrupt the index fields and the entire data base. For other programs, continuing program execution is recommended. For example, when you transfer a long text file and receive an error because one sector is bad, you can continue transferring the file. After the file is transferred, review the file, and add the data that was not transferred due to the bad sector.

Message	Meaning
CP/M Disk select error on drive x Do you want to: Abort (A), Retry (R)	
	There is no disk in the drive or the disk is not inserted correctly. Ensure that the disk is securely inserted in the drive. If you enter the R option, the system retries the operation. If you enter the A option or CTRL-C the program terminates and CPM-68K returns the system prompt.

Table E-4. (continued)

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

E.4 BIOS Error Messages

The CP/M-68K BIOS error messages are listed in Table E-5 in alphabetic order with explanations and suggested user responses.

Table E-5. BIOS Error Messages

Message	Meaning
BIOS ERROR -- DISK X NOT SUPPORTED	
	The disk drive indicated by the variable '/.,' is not supported by the BIOS. The BDOS supports a maximum of 16 drives, lettered A through P. Check the manufacturer's documentation for your system configiration to find out which of the BDOS drives your BIOS implements. Specify the correct drive code and reenter the command line.
BIOS ERROR -- Invalid Disk Status	
	The disk controller returned unexpected or incomprehensible information to the BIOS. Retry the operation. If the error persists, check the hardware. If the error does not come from the hardware, it Is caused by an error in the internal logic of the BIOS. Contact the place you purchased your system for assistance. You should provide the following information. 1. Indicate which version of the operating system you are using. 2. Describe your system's hardware configuration. 3. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

E.5 CCP Error Messages

The CP/M-68K Console Command Processor, CCP, returns two types of error messages at the console: diagnostic and internal logic error messages.

E.5.1 Diagnostic Error Messages

The CCP error messages are listed in Table E-6 in alphabetic order with explanations and suggested user responses.

Table E-6. CCP Diagnostic Error Messages

Message Meaning

Bad relocation information

This message is a result of a BDOS Program Load Function (59) error. It indicates that the file specified in the command line is not a valid executable command file, or that the file has been corrupted. Ensure that the file is a command file. Section 3 of this manual describes the format of a command file. If the file has been corrupted, reassemble or recomplie the source file, and relink the file before you reenter the command line.

File already exists

This error occurs during a REN command. The name specified in the command line as the new filename already exists. Use the ERA corn-mand to delete the existing file if you wish to replace it with the newfile.If not, select another filename and reenter the REN command line.

insufficient memory or bad file header

This error could result from one of three causes:

1. The file is not a valid executable command file. Ensure that you are requesting the correct file. This error can occur when you enter the filename before you enter the command for a utility. Check the appropriate section of this manual or the CP/M-68K Operating System User's Guide for the correct command syntax before you reenter the command line. If you are trying to run a program when this error occurs, the program file may have been corrupted. Reassemble or recompile the source file and relink the file before you reenter the command line.

2. The program is too large for the available memory. Add more memory boards to the system configuration, or rewrite the program to use less memory.

Table E-6. (continued)

Message	Meaning
	3. The program is linked to an absolute location in memory that cannot be used. The program must be made relocatable, or linked to a usable memory location. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the memory space that is available for loading programs.
No file	The filename specified in the command line does not exist. Ensure that you use the correct filename and reenter the command line.
No wildcard filenames	The command specified in the command line does not accept wildcards in file specifications. Retype the command line using a specific filename.
read error on program load	This message indicates a premature end-of-file. The file Is smaller than the header information indicates. Either the file header has been corrupted or the file was only partially written. Reassemble, or recompile the source file, and relink the file before you reenter the command line.
SUB file not found	The file requested either does not exist, or does not have a filetype of SUB. Ensure that you are requesting the correct file. Refer to the section on SUBMIT in the CP/M-68K Operating System User's Guide for information on creating and using submit files.
Syntax: REN newfile-oldfile	The syntax of the REN command line is incorrect. The correct syntax is given in the error message. Enter the REN command followed by a space, then the new filename, followed immediately by an equals sign (=) and the name of the file you want to rename.

Table E-6 (continued)

Message	Meaning
Too many arguments: argument?	The command line contains too many arguments. The extraneous arguments are indicated by the variable "argument". Refer to the CP/M-68K Operating System User's Guide for the correct syntax for the command. Specify only as many arguments as the command syntax allows and reenter the command line. Use a second command line for the remaining arguments, if appropriate.
User # range is [0..15]	The user number specified in the command line is not supported by the BIOS. The valid range is enclosed in the square brackets in the error message. Specify a user number between 0 and 15 (decimal) when you reenter the command line.

E. 5.2 CCP Internal Logic Error Messages

The following message indicates an undefined failure of the BDOS Program Load Function (59).

Program Load Error

If you receive this message, contact the place you purchased your system for assistance. You should provide the following information.

1. Indicate which version of the operating system you are using.
2. Describe your system's hardware configuration.
3. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

E.6 DDT-68K Error Messages

The CP/M-68K debugger, DDT-68K, returns two types of error messages: nonfatal diagnostic error messages and fatal errors in the internal logic of DDT-68K.

E.6.1 Diagnostic Error Messages

Diagnostic error messages are returned at the console as the error occurs. The DDT-68K error messages are listed in Table E-7 in alphabetic order with explanations and suggested user responses.

Table E-7. DDT-68K Diagnostic Error Messages

Message	Meaning
Bad or nonexistanrt RAM at HEX no.	This error occurs in response to a Set (S), Set Word (SW), or Set Longword (SL) command. The message indicates one of two things. 1. The memory location at HEX no . is read-only, an I/O port, or nonexistent. Use another location. 2. The memory location is damaged. Check the hardware.
Bad relocation bits	This message is returned from the BDOS Program Load Function (59), and means one of two things. 1. The command file has been corrupted. Rebuild the file. Reassemble or recompile the source file, and relink the file before you reenter the DDT-68K command line. 2. The file is linked to an absolute location in memory that is already occupied by DDT-68K. Link the file to another location: DDT-68K occupies approximately 20K of memory, and resides at the highest addresses within the TPA. The recommended location for linking your file is the base address of the TPA + 100H. BDOS Function 63, Get/Set TPA Limits, returns the high and low boundaries of the TPA.

Table E-7. (continued)

Message	Meaning
Cannot create file	This error occurs during a Write (W) command. The disk to which DDT-68K is writing has no more directory space available: in effect, the disk is full. If you have another drive available, reenter the Write (W) command and direct the file to the disk on that drive. If you do not have another drive available, you must exit DDT-68K (and lose the contents of memory). Erase unnecessary files, 'if any, or insert a new disk.
Cannot open file	This error occurs during a Read (R) command. It indicates an incorrect user number, drive code, or filename. Check the user number, drive code, and filename before you reenter the command line.
Cannot open program file	This message occurs in response to a Load for Execution (E) command. It indicates an incorrect user number, drive code, or filename. Check the user number, drive code, and filename before you reenter the command line.
ERROR, no program or file loaded.	This error message occurs in response to a Value (V) command when you specify the command but no file is loaded. Load a file before you reenter the V command. The file can be loaded with a Load for Execution (E) or Read (R) command, or by specifying the filename when you invoke DDT-68K.
File too gig -- read truncated	This message occurs during a Read (R) command when the file being read is too large to fit in memory. DDT-68K reads only the portion of the file that can be read into the existing memory. To debug this program, additional memory boards must be added to the system configuration.

Table E-7. (continued)

Message	Meaning
File write error	The disk to which DDT-68K is writing is full or the disk contains a bad sector. Retry the command. If the error persists, and you have another disk drive available, redirect the output to the disk on that drive. If you do not have another drive available, you must exit DDT-68K. Use the STAT command to check the space on the disk. If it is full, erase unnecessary files, if any, or insert a new disk. Because the contents of memory are lost when you exit DDT-68K, you must reload the file in memory. If the disk was not full, it has a bad sector. You should replace the disk.
**illegal size fold	This error occurs during a List (L) command. The size field of the instruction being disassembled has an illegal value. Use a Display (D) command to display the location of the error. This error could be caused by one of three things: 1. The memory location being disassembled does not contain an instruction. Ensure that the area selected is an instruction. If not, reenter the L command with a correct location. 2. The size field of the instruction has been corrupted. Use the debugging commands in DDT-68K to look for an error that causes the program to overwrite itself. Refer to the section in this manual on DDT-68K for a complete description of the DDT-68K commands and options. 3. An invalid instruction was generated by the compiler or assembler used to create the program. Recompile or reassemble the source file before you reinvoke DDT-68K.

Table E-7. (continued)

Message	Meaning
Insufficient memory or bad file header	
	This message occurs in response to a Load for Execution (E) command. The error could be caused by one of three things: 1 . The system you are using does not have enough memory available. Ensure that the program and DDT-68K fit into the TPA. Exit DDT-68K. Use the SIZE68 Utility to display the amount of space your program occupies in memory. DDT-68K is approximately 20K bytes. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the TPA. If you do not have sufficient space in the TPA to execute your command file and DDT-68K simultaneously, additional memory boards must be added to the system configuration. 2. The file is not a command file or has a corrupted header. If the command file does not run', but you are sure that your memory space is adequate, use the R command to took at the file and check the format. You may be trying to debug a file that is not a command file. If it is a command file, the header may have been corrupted. Reassemble or recompile the source file before you reenter the E command line. If the error persists, it may be caused by an error in the internal logic of DDT-68K. Contact the place you purchased your system for assistance. You should provide the following information: a. Indicate which version of the operating system you are using. b. Describe your system's hardware configuration. c. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

Table E-7. (continued)

Message	Meaning
	3. The command file you are debugging is linked to an absolute location in memory that 'is already occupied by DDT-68K. DDT-68K is approximately 20K bytes, and usually resides in the highest addresses of the TPA. The recommended location for linking your file is the base address of the TPA + 100H. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the TPA.
Read error	This message may indicate one of three things. Try the operation again. If the error persists, try the responses indicated: 1. A write error at the time the file was created. You must recreate the file. If the error reoccurs, or if you cannot write to the disk, the disk is bad. 2. A bad disk. Use PIP or COPY to copy the file from the bad disk to a new disk. Any files that cannot be copied must be recreated or replaced from backup files. Discard the damaged disk. 3. A hardware error. If the error persists, check your hardware.
unknown opcode	This error occurs in response to a List (L) command if the memory location being disassembled does not contain a valid instruction. The error may have been caused by one of three things: 1. You gave the L command the wrong address. Reenter the L command with the correct address. 2. The file is not a command file. Ensure that the file you specify is a command file and reenter the L command.

Table E-7. (continued)

Message	Meaning
	3. The command file has been corrupted. Reassemble or recompile the source file before you reread it into memory with a Load for Execution (E) or Read (R) command, as appropriate. If the problem persists, use the debugging commands in DDT-68K to look for an error in the program that causes it to overwrite itself. Refer to the section in this manual on DDT-68K for a complete description of the DDT-68K commands and options.

E.6.2 DDT-68K Internal Logic Error Messages

This section lists fatal errors in the internal logic of DDT-68K. If you receive one of these messages, contact the place you purchased your system for assistance. You should provide the following information.

1. Indicate which version of the operating system you are using.

2. Describe your system's hardware configuration.

3. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

Errors:

illegal instruction format #

Unknown program load error

E.7 DUMP Error Messages

DUMP returns fatal, diagnostic error messages at the console. The DUMP error messages are listed in 'Fable E-9 'in alphabetic order with explanations and suggested user responses.

Table E-8. DUMP Error Messages

Message	Meaning
Unable to open filename	Either the drive code for the input file indicated by the variable "filename" is 'incorrect', or the filename is misspelled. Check the filename and drive code before you reenter the DUMP command line.
Usage: dump [-shhhhhh] file	The command line syntax is incorrect. The correct syntax is given in the error message. Specify the DUMP command and the filename. If you want to display the contents of the file from a specific address in the file, specify the -S option followed by the address. Refer to the section in this manual on the DUMP Utility for a discussion of the DUMP command line options.

E.8 L068 Error Messages

The CP/M-68K Linker, L068, returns two types of fatal error messages: diagnostic and logic. Both types of fatal error messages have the following format:

: error message text

The colon (:) indicates that the error message comes from L068. The "error message text" describes the error.

E.8.1 Fatal Diagnostic Error Messages

A fatal diagnostic error prevents your program from linking. When the error is caused by a full disk, erase the partial file that L068 created on the disk that received the error to ensure that you do not use the file. The L068 diagnostic errors are listed in Table E-9 in alphabetic order with explanations and suggested user responses.

Table E-9. L068 Fatal Diagnostic Error Messages

Message	Meaning
: duplicate definition in p,filename	The symbol indicated by the variable p is defined twice. The variable "filename" indicates the file -in which the second definition occurred. Rewrite the source code. Provide a unique definition for each symbol and reassemble or recompile the source code before you relink the file.
: file fornat error: filename	The file indicated by the variable "filename" is either not an object file or the file has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. Reassemble or recompile the file before you relink it.
: File Format Error: Invalid symbol flags = flags	L068 does not recognize the symbol flags indicated by the variable f I a g s. The file L068 read is either not an object file or it has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. Reassemble or recomp'le the file before you relink it.
: File Format Error: invalid relocation flag in filename	The contents of the file indicated by the variable "filename" are incorrectly formatted. The file either is not an object file or has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. If the file is an object file but this error occurs, the file has been corrupted. Reassemble or recompile the file before you relink it.
: File Format Error: no relocation bits in filename	The file indicated by the variable "filename" either is not an object file or has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. If the file is an object file but this error occurs, then the file has been corrupted. Reassemble or recompile the file before you relink it.
: Illegal option p	The option in the command line indicated by the variable p is invalid. Supply a valid option and relink.

Table E-9. (continued)

Message	Meaning
: Invalid lo68 argument list	This message indicates format errors or invalid options in the command line. Examine the command line to locate the error. Correct the error and relink.
: output file write error	The disk to which L068 is writing is full. Erase unnecessary files, if any, or insert a new disk before you reenter the L068 command line.
: read error on file: filename	The object file indicated by the variable "filename" does not have enough bytes. The file either is incorrectly formatted or has been corrupted. This error is commonly caused when the input to L068 is a partially assembled or compiled object file. The assembler, AS68, and some compilers create partial object files when they receive the "disk full abort" message while assembling or compiling a file. Ensure that the file 'is a complete object file. Reassemble or recompile the file before you relink it.
: symbol table overflow	The object code contains too many symbols for the size of the symbol table. Rewrite the source code to use fewer symbols. Reassemble or recompile the source code before you relink the file.
: Unable to create filename	Either the output file indicated by "filename" has an invalid drive code, or the disk to which L068 is writing is full. Check the drive code. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the L068 command line.
: unable to open filename	The filename indicated by the variable "filename" is invalid, or the file does not exist. Check the filename before you reenter the L068 command line.

Table E-9. (continued)

Message	Meaning
: Unable to open temporary file: filename	
	Either the file, indicated by "filename" has an invalid drive code, specified by the f option, or the disk to which L068 is writing's full. Check the drive code. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the L069 command line.
: Undefined symbol(s)	
	The symbol or symbols which are listed one per line on the lines following the error message are undefined. Provide a valid definition and reassemble the source code before you reenter the L068 command line.

E.8.2 L068 Internal Logic Error Messages

This section lists messages indicating fatal errors in the internal logic of L068. If you receive one of these messages, contact the place you purchased your system for assistance. You should provide the following information:

1. Indicate which version of the operating system you are using.

1

2. Describe your system's hardware configuration.
3. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

Errors:

: assignment botch

: finalwr: text size error

: relative address overflow at lx in sn

: seek error on file filename

: short address overflow in filename

: unable to reopen filename

E.9 NM68 Error Messages

NM68 returns fatal diagnostic error messages at the console. The NM68 error messages are listed in Table E-10 in alphabetic order with explanations and suggested user responses.

Table E-10. NM68 Error Messages

Message	Meaning
File format error: filename	The input file indicated by the variable "filename" is neither an object file nor a command file. This message can also 'indicate a corrupted file. NM68 prints the symbol table of an object file or a command file. Ensure that the file is one of these types of file. If the file is an object or command file and you receive this message, the file is corrupted. Rebuild the file with the compiler or assembler. If the file is a command file, relink it. Reenter the NM68 command line.
read error on file: filename	The input file indicated by the variable "filename" is truncated. Rebuild the file with the compiler or assembler. If the file is a command file, relink it. Reenter the NM68 command line.
unable to open filename	The filename indicated by the variable "filename" is incorrect. Check the spelling of the filename and reenter the command line.
Usage: nm68 objectfile	The command line syntax is incorrect. Use the syntax given in the error message and reenter the command line.

E.10 RELOC Error Messages

The Relocation Utility (RELOC) returns fatal error messages at the console. RELOC error messages are listed in Table E-11 in alphabetic order with explanations and suggested user responses.

Table E-11. RELOC Error Messages

Message	Meaning
create filename	Either the drive code for the output file indicated by the variable "filename" is incorrect, or the disk to which RELOC is writing is full. Check the drive code. If it is correct, the disk is full. Erase unnecessary files, 'if any, or insert a new disk before you reenter the RELOC command line.
Cannot open file	The input file indicated by the variable "filename" does not exist. Ensure that you type the correct filename when you reenter the RELOC command line.
Cannot re-open filename	This error message indicates a hardware error. Check the hardware for errors. This error most often occurs in the disk, disk drive, or memory.
file format error: filename	This error occurs because the first word in the header record of the command file must contain the value 60lAH and the file must contain relocation bits. If your file does not meet these criteria, you cannot use RELOC. 1. The file indicated by the variable "filename" is not a command file with contiguous program segments (the first word in the header record is 601AH). If the file is an object file, link it before you reenter the RELOC command line. 2. The file does not have relocation bits because it is already linked to an absolute location. Use the original source file that contains relocation bits with RELOC.
Illegal base address=hex no.	The odd base address indicated by the variable "hex no.", is invalid under CP/M-68K. Base addresses must be even. Specify an even base address and reenter the RELOC command line.

Table E- 1 1. (continued)

Message	Meaning
Illegal option: x	The option specified for the RELOC command must be -b. The invalid option is indicated by the variable x . Replace the invalid option with -b and reenter the RELOC command line.
Illegal reloc = x at address	This message may indicate one of two things: 1. The command file is truncated or corrupted. RELOC recognized the error because the relocation value indicated by the variable x is invalid. The variable a d -J r e s s indicates the location in memory of the invalid relocation value. Rebuild the file. Reassemble or recompile, and relink the file before you reenter the RELOC command line. 2. The file has no relocation bits. Use the original source code with relocation bits and try again.
Read error on filename	The input file indicated by the variable "filename" is truncated or corrupted. Rebuild the file. Reassemble, or recompile, and relink the file before you reenter the RELOC command line.
16-bit overflow at address	The address indicated by the variable "address" cannot contain a 16-bit quantity. Source code that uses 16-bit offsets must fit in the first 64K bytes of memory. BDOS Function 63, Get/Set TPA Limits, returns the high and low boundaries of the memory available for loading programs. SIZE68 displays the amount of memory space a program occupies. Use the Get/Set TPA Limits Function and SIZE68 to ensure that the program fits in the first 64K of memory. If the program does not fit, you must rewrite the source code to use 32-bit offsets.

Table E- I 1. (continued)

Message	Meaning
Usage: reloc -bhhhhhh input output where hhhhhh is new base address input is relocatable file output is absolute file	This message indicates a syntax error in the RELOC command line. The correct syntax is given in the error message. Retype the command line with the correct syntax. Refer to the section in this manual on the RELOC Utility for more detailed information on the command line syntax.
Write error on filename Offset = x data = x error = x	The disk to which RELOC is writing is full. Erase unnecessary files, if any, or insert a new disk before you reenter the RELOC command line.

E.11 SENDC68 Error Messages

SENDC68 returns two types of fatal error messages: diagnostic and internal logic error messages.

E. 11. 1 Diagnostic Error Messages

The SENDC68 diagnostic error messages are listed in Table E-12 in alphabetic order with explanations and suggested user responses.

Table E-12. SENDC68 Diagnostic Error Messages

Message	Meaning
file format error: filename	The file indicated by the variable "filename" is not a command file. The file input to SENDC68 must be a command file, output by the linker (LO68). Ensure that the file specified is a command file.
read error on file: filename	The file indicated by the variable "filename" is truncated. Rebuild the file by recompiling or reassembling, and relink it before you reenter the SENDC68 command line.
unable to create filename	This message indicates an invalid drive code for the output file indicated by the variable "filename" , It can also mean that the disk to which SENDC68 is writing is full. Check the drive code. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the SENDC68 command line.
unable to open filename	The input file indicated by the variable "filename" does not exist. Check the filename and retype the SENDC68 command line.
Usage: sendc68 [-] command file [output file]	This message indicates a syntax error in the SENDC68 command line. The correct syntax is given in the error message. Retype the command line using the correct syntax.

E.11.2 SENDC68 Internal Logic Error Messages

The following is a fatal error in the internal logic of SENDC68.

seek error on file filename

If you receive this message, contact the place you purchased your system for assistance. You should provide the following information.

1. Indicate which version of the operating system you are using.
2. Describe your system's hardware configuration.
3. Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

E. 12 SIZE68 Error Messages

SIZE68 returns fatal, diagnostic error messages at the console. The SIZE68 error messages are listed in Table E-13 in alphabetic order with explanations and suggested user responses.

Table E-13. SIZE68 Error Messages

Message	Meaning
File format error: filename	The file indicated by the variable "filename" is neither an object file nor a command file. SIZE68 requires either an object file, output by the assembler or the compiler, or a command file, output by the linker. Ensure that the file specified is one of these and reenter the SIZE68 command line.
read error on filename	The file indicated by the variable "filename" is truncated. Rebuild the file. Reassemble or recompile, and relink the source file before you reenter the SIZE68 command line.
unable to open filename	Either the drive code is incorrect, or the file indicated by the variable "filename" does not exist. Check the drive code and filename. Reenter the SIZE68 command line.

End of Appendix E

Appendix F
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.

Table F-1. New BDOS Functions

Function	Number
Get Free Disk Space	46
Chain To Program	47
Flush Buffers	48
Set Exception Vector	61
Set Supervisor State	62
Get/Set TPA Limits	63

F.1 BDOS Function and Data Structure Changes

Implementation changes in CP/M-68K BDOS functions and data structures are described in the following table:

Table F-2. BDOS Function Implementation Changes

BDOS Function	Number	Implementation Change
Return Version Number	12	Contains the version number 2022H indicating CP/M-68K Version 1. 1.
Reset Disk System	14	Does not log in disk drive A when it resets the disk system.
Open File	15	Opens a file only at extent 0, the base extent.
Get Disk Parameters	31	Returns a copy of the Disk Parameter Block (DPB).

Table F-3. BDOS Data Structure Implementation Changes

Structure	Implementation Change
Base Page	Additional information has been added. The base page is no longer located at a fixed address. Appendix C outlines the structure of the base page.
File Control Block	The byte sequence for the Random Record Field has changed. The most significant byte (r0) is first and the least significant byte (r2) is last.

F.2 BDOS Functions Not Supported by CP/M-68K

The following table contains functions and commands supported by other CP/M systems, but that are not supported by CP/M-68K.

Table F-4. BDOS Functions Not Supported by CP/M-68K

BDOS Function	Number
Get Address of Allocation Vector	27
Set DMA Base+	51
Get DMA Base+	52
Get Maximum Memory*	53
Get Absolute Memory*	54
Allocate Absolute Memory*	5 5
Free Memory*	56
Free All Memory*	57

+ The 68000 microprocessor does not have a segmented architecture. Therefore, functions involving segment registers are not relevant to CP/M-68K.

* CP/M-68K does not have memory management functions.

In addition to the above BDOS Functions, CP/M-68K does not support the Assemble (A) command in DDT-68K.

End of Appendix F

Index

F

G

H

I