Table 5-2. (continued)

Directive	Meaning
globl label[,label...] xdef label[,label...] xref label[,label...]	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 LO68. If the labels are not defined in the current assembly, they become undefined external references, which LO68 links to external values with the same label in other routines. If you specify the -U option, the assembler makes all undefined labels external.
ifeq expression ifne expression ifle expression iflt expression ifge expression ifgt 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 ifge greater or equal to zero ifgt greater than zero
ifc 'string1', 'string2' ifnc 'string1', 'string2'	The conditional string directive compares two strings. The 'c' condition is true if the strings are exactly the same. The 'nc' condition is true 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 example below illustrates a sample register list. A2-A4/A7/D1/D3-D5 You can also use commas to separate registers as shown below. 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 listed below.

1) All assembler directives are optionally preceded by a period (.). For example,

   .equ or equ
   .ds or ds

2) AS68 does not support 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 R0-R7 are the same as D0-D7 and R8-R15 are the same as A0-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 as shown below.

```
* = This command loads R1 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 enhancements listed below 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, cmp, eor, or, sub are allowed with immediate first operands and actually generate addi, andi, cmpi, eori, ori, 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, "asl r1" is equivalent to "asl #1,r1".

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

.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, LO68, 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
LO68 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 (L068)

Invoke L068 by entering a command of the following form:

```
L068 [-F d:] [-R] [-S] [-I] [-Umodname]
     [-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.
-S	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 -I 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.
-Umodname	Links a module within the library with other modules in the command line. The module name, specified by the modname parameter, cannot exceed eight characters. You can use this option to create a program from modules within a library, if the module following the U option calls other modules in the library.
-O filename	If specified, the object file produced by L068 has the filename that you specify following the -O option. The -O 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.

Table 6-1. (continued)

Option	Meaning
-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 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.
-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 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 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.

Table 6-1. (continued)

Option	Meaning
>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 L068

```
A>L068 -S -O TEST.68K TEST.O
```

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

```
A>L068 -T4000 -D8000 -BC000 A.O B.O C.O
```

This command links assembled files A.O, B.O, and C.O 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>L068 -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 L068 Error Messages

Appendix E lists the error messages that L068 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 Reference Manual for the 68000 microprocessor.

7.1.1 AR68 Syntax

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

```
AR68 DRTWX[AV][F D:] [OPMOD] ARCHIVE OBMOD1 [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 command line syntax and the system prompt as shown below.
	`A>AR68`
	`usage: AR68 DRTWX[AV][F D:] [OPMOD] ARCHIVE OBMOD1 [OBMOD2...][>filespec]`
	`A>`

Table 7-1. (continued)

Component	Meaning
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.
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 as shown below.

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.O 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:

Table 7-2. (continued)

Command/Option	Meaning
	• The object module does not already exist in the library. • You specify the A option in the command line. • The name of a module follows the name of module that does not already exist in the library. 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 as shown below. **A>AR68 RAV SDAV.O MYRAH.ARC WORK.O MAIL.O** c much.o c sdav.o a work.o a mail.o c less.o The RAV command adds the object modules WORK.O and MAIL.O after the module SDAV.O 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.

Table 7-2. (continued)

Command/Option	Meaning
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.o r nail.o c screw.o a wrench.o A> The R command replaces the object module NAIL.O and adds the module WRENCH.O 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 example below. **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>

Table 7-2. (continued)

Command/Option	Meaning
	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® 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.
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 W GO.ARC NOW.O >B:NEWNAME.O** The W command writes a copy of the object module NOW.O from the library GO.ARC to the file NEWNAME.O 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.

Table 7-2. (continued)

Command/Option	Meaning
V	Lists only those modules the X command extracts from the library. It precedes each extracted module with the lower-case letter as shown below. **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.O, HAM.O, and SCREW.O 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 ] filename1 [ >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.
filename1	is the name of the file you want to dump.
>filename2	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:

```
rrrr oo (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.
oo	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 diplayed as a period (.).

7.2.3 DUMP Examples

An example of the DUMP Utility is shown below. 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 6768 *Runtime Copyrigh*
0000 30 (000030):  7420 3139 3832 2062 7920 4469 6769 7461 *t 1982 by Digita*
0000 40 (000040):  6c20 5265 7365 6172 6368 2056 3031 2c30 *l Research V01.0*
0000 50 (000050):  3320 206f 0004 2268 0018 2649 d3e8 001c *3  o.."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 shown below.

```
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 the lowest address in the TPA.
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 filename 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 User's Guide.

7.4.1 Invoking SIZE68

You invoke SIZE68 by entering the SIZE68 command line in the format shown below.

```
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.
filename1 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 format of the output line is shown below.

```
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.
totsize	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>size68 reloc.68k
reloc.68k:11330+1012+2922=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 3BA0 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 size.68k, dump.68k
size68.68k: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.

```
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 format shown below.

```
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-record 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 LO68 or RELOC.

Table 7-8. (continued)

Component	Meaning
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 example below illustrates how to create a file that contains Motorola S-records from an absolute command file.

```
A>SENDC68 - 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 4-12. DDT-68K does not process the command line until you enter a RETURN.

The first nonblank character of each 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 tail
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 may 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 ↑C 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 llllllll llllllll ... llllllll 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 l 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 l 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,b1,...b10
Gs
Gs,b1,...b10
```

where s is the address where program begins executing and b1 through b10 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 breakpoints 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 16 bits on the next line as shown below:

```
ssssssss dddddddd
```

8.2.6 The I (Input Command Tail) Command

The I command prepares a file control block (FCB) and command tail buffer in DDT-68K's base page, and copies the information in the base page of the last file loaded with the E command. The form is

```
I<command tail>
```

where <command tail> 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 tail> are also copied to the default command buffer at 0080H. The length of the <command tail> 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 (↑S) to halt the display temporarily. A CONTROL-Q (↑Q) restarts the display after ↑S 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>

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

DDT-68K read the file in 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 llllllll

where bb, wwww, and llllllll 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 1 to 0FFFFFFFFH 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 before 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=00003FDA
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-D7, A0 to A7, PC, USP, or SSP

The first form displays the CPU state as follows:

```
PC=00016000 USP=00001000 SSP=00002000 ST=FFFF=> (etc.)
D  00001000 00000D01 ... 00000001
A  000B0A00 000A0010 ... 00000000
lea $16028,A0
```

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. A sample is shown below:

```
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 listed below.

- DDT-68K prints all numeric values in hexadecimal.
- DDT-68K uses lower-case mnemonics.
- 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 I/O Mapping Byte
Set I/O Byte	20	Set I/O 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 a C 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 example below 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),a1      *get address of 1st parsed FCB in base page
        move.l  a1,d1           *put that address in register d1
        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
*
        move.l  $18(a0),d2      *get starting address of bss from base page
        move.l  $1c(a0),d3      *get length of bss
        add.l   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.l   d3,d4           *compute high end of free memory after bss
        move.l  d3,a3           *get the starting address of free memory
                                *into a3
        sub     #1,d5           *adjust loop counter
clear:  clr.b   (a3)+           *clear out free memory
```

Listing B-1. Transient Load Program Example 1

```
        dbf     d5,clear        *decrement 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  a1,LPB          *put address of open FCB into LPB
        move.w  #pgmldf,d0      *get BDOS program load function number
        move.l  #LPB,d1         *put address of LPB into register d1
        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,d1       *d1 points to new program's base page
        add     #$80,d1         *d1 points to default dma in base page
        move.w  #setdma,d0      *get BDOS function number
        trap    #2              *set the default dma address
*
*       Now push needed addresses on stack
*
        movea.l usrstk,a7       *set up user stack pointer
        move.l  baspag,a1       *get address of base page
        move.l  a1,-(sp)        *push base page address
        move.l  #cmdrtn,-(sp)   *push return address
        move.l  8(a1),-(sp)     *push address to jump to
        rts                     *jump to new program
*
*       Print ERROR message
*
openerr:
        move.l  #openmsg,d1     *get address of error message
*                               *to be printed
        bra     print
lderr:  move.l  #loaderr,d1     *get address of error message to
                                *be printed
print:  move.w  #printstr,d0    *get BDOS function number
        trap    #2              *print the message
cmdrtn: move.w  #reboot,d0      *get BDOS function number
        trap    #2              *warmboot and return to the CCP
*
*       DATA
*
        .data
        .even
*
```

Listing B-1. (continued)

```
*       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
hiadr:      .ds.l       1       *High boundary of area in which to
*                               *to load program
baspag:     .ds.l       1       *Base page address of loaded program
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

        .even

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

Listing B-1. (continued)

2) The example below is a C language transient program that loads another program in the TPA without the assistance of the CCP. The C language program calls an AS68 assembly language routine to perform tasks not permitted by the C language.

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


/*      DEFINES         */

#define         BSS_OFFSET      (long)0x18
#define         FCB_OFFSET      (long)0x5C
#define         BSS_LENGTH      (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         SETDMA          26
#define         PGMLDF          59
#define         GETTPA          63

/* Error Messages       */

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

/* Load Parameter Block */

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

/* TPA Parameter Block */

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

```
openfile(baseaddr)                          /**********************************/
register char   *baseaddr;                  /* base page address              */
{                                           /*                                */
        register long   *t1,*t2;            /* pointers to long word values */
        register long   count;              /* long word value                */
        register char   *ptr1,*ptr2;        /* pointers to character values */
                                            /*                                */
                                            /*                                */
        ptr1 = baseaddr + FCB_OFFSET;       /* get address of FCB             */
        if(bdos(OPEN,ptr1) <= 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 = *t1 + *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 BSS segment          */
                                            /*                                */
                hiadr = *t2 + lowadr        /* compute high end of avaiable */
                                            /* memory                         */
                count = *t2 - ROOM          /* Leave some extra room in Mem */
                while(count--)              /* Clear out available Memory     */
                        *ptr2++ = NULL;     /* with NULL byte values          */
                LPB = ptr1;                 /* 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);
```

Listing B-2. (continued)

```
        }
        else
                error(OPEN);
}

error(flag)
int   flag;
{
        bdos(CON_OUT,CR);
        bdos(CON_OUT,LF);
        if(flag == OPEN)
                bdos(PRINTSTR,openmsg);
        else
                bdos(PRINTSTR,loadmsg);
        bdos(REBOOT,(long)0);
}

main()
{
        bdos(REBOOT,(long)0);
}

        ****************************************************************
        *                                                              *
        *         Assembly Language Module Needed to                   *
        *     Assist 'C' code to Load a Program into the 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
        .globl _openfile
        .globl _push
        .globl _main

*
*       Get the address of the base page
*
```

Listing B-2. (continued)

```
_start:
        link    a6,#0           *link and allocate
        move.l  8(a6),-(sp)     *push the address of the base page
        jsr     _openfile       *jump to 'C' code to open the file
*
*       Call the BDOS
*
_bdos:
        move.w  4(sp),d0        *get the BDOS function number
        move.l  6(sp),d1        *get the BDOS parameter
        trap    #2              *call the BDOS
        rts                     *return
*
*       Push the needed addresses on to the stack
*
_push:
        movea.l _usrstk,a7      *set up the user stack pointer
        move.l  _baspag,a1      *get address of user base page
        move.l  a1,-(sp)        *push base page address
        move.l  #_main,-(sp)    *push return address
        move.l  8(a1),-(sp)     *push address 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
_hiadr:   .ds.l  1              *High boundary of area in which to
*                               *to load program
_baspag:  .ds.l  1              *Base page address of loaded program
_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
```

Listing B-2. (continued)

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

Offset	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
0025 - 0037	Reserved, unused
0038 - 005B	2nd parsed FCB from Command Line
005C - 007F	1st 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
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

Table D-1. (continued)

Instruction	Description
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
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 Immediate
eor	eor	Exclusive OR
	eori	Exclusive OR Immediate
	eori 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
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 are 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 below in alphabetic order with explanations and suggested user responses.

Table E-1. Fatal Diagnostic Error Messages

Message	Meaning
filename not in archive 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.

Table E-1. (continued)

Message	Meaning
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.
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: • Indicate which version of the operating system you are using. • Describe your system's hardware configuration. • 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.

Table E-1. (continued)

Message	Meaning
`usage: AR68 DRTWX[AV][F D:] [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.
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 below 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 location 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. end 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.

Table E-2. (continued)

Message	Meaning
& 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.
& 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 l (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.

Table E-2. (continued)

Message	Meaning
& 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.
& 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.

Table E-2. (continued)

Message	Meaning
& 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 l (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.
& 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.

Table E-2. (continued)

Message	Meaning
& 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 reassemble 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 undefined symbol that has not 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

Described below are the fatal error messages for AS68. 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.

Table E-3. (continued)

Message	Meaning
& 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.
& 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 "Itoffset= 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.

Table E-3. (continued)

Message	Meaning
& 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.
& line no. 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.
& symbol table overflow	The 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.

Table E-3. (continued)

Message	Meaning
& 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 read 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.
& 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 AS68 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 information below.

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

2) Describe your system's hardware configuration.

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

Errors:

& doitrd: buffer botch 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 below 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.
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.

Table E-4. (continued)

Message	Meaning	
	Option	Action
	A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.
	R	Retries the operation. If the retry fails, 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.
CP/M Disk write error on drive x Do you want to: Abort (A), Retry (R), or Continue with bad data (C)?	BDOS. This message indicates a hardware error. Specify one of the options enclosed in parentheses. Each option is described below.	

Table E-4. (continued)

Message	Meaning	
	Option	Action
	A or CTRL-C	Terminates the operation and CP/M-68K returns the system prompt.
	R	Retries the operation. If the retry fails, 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.
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 below 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 "X" 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 configuration 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 information below.

Table E-5. (continued)

Message	Meaning
	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 below in alphabetic order with explanations and suggested user responses.

Table E-6. CCP Diagnostic Error Messages

Message	Meaning
bad relocation information bits	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 recompile the source file, and relink the file before you reenter the command line.

Table E-6. (continued)

Message	Meaning
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 command to delete the existing file if you wish to replace it with the new file. If not, select another filename and reenter the REN command line.
insufficient memory or bad file header	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. 3) The program is linked to an absolute location in memory that cannot be used. The program must be made relocatable, or linked to a usable memory location. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the memory space that is available for loading programs.

Table E-6. (continued)

Message	Meaning
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 information below.

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

2) Describe your system's hardware configuration.

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

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 below in alphabetic order with explanations and suggested user responses.

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

Message	Meaning
Bad or nonexistent 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.

Table E-7. (continued)

Message	Meaning
	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.
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.

Table E-7. (continued)

Message	Meaning
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 big -- 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.
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.

Table E-7. (continued)

Message	Meaning
**illegal size field	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.
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:

Table E-7. (continued)

Message	Meaning
	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 look 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 information below. 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:

Table E-7. (continued)

Message	Meaning
	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. 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 information below.

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

2) Describe your system's hardware configuration.

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

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 below 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 and options.

E.8 LO68 Error Messages

The CP/M-68K Linker, LO68, 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 LO68. 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 LO68 created on the disk that received the error to ensure that you do not use the file. The LO68 diagnostic errors are listed below in alphabetic order with explanations and suggested user responses.

Table E-9. LO68 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 format 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	LO68 does not recognize the symbol flags indicated by the variable "flags." The file LO68 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 recompile 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.

Table E-9. (continued)

Message	Meaning
: 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.
: 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 LO68 is writing is full. Erase unnecessary files, if any, or insert a new disk before you reenter the LO68 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 LO68 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.

Table E-9. (continued)

Message	Meaning
: 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 LO68 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 LO68 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 LO68 command line.
: 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 LO68 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 LO68 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 LO68 command line.

E.8.2 LO68 Internal Logic Error Messages

This section lists messages indicating fatal errors in the internal logic of LO68. If you receive one of these messages, contact the place you purchased your system for assistance. You should provide the information below.

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

2) Describe your system's hardware configuration.

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

Errors:

: asgnext 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 below 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 below 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 filename	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 601AH 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.

Table E-11. (continued)

Message	Meaning
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.
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 "address" 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.

Table E-11. (continued)

Message	Meaning
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.
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 below 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 [-] commandfile [outputfile]	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.

INTERNAL LOGIC ERROR: seek error on file filename

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

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

2) Describe your system's hardware configuration.

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

E.12 SIZE68 Error Messages

SIZE68 returns fatal, diagnostic error messages at the console. The SIZE68 error messages are listed below 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.

Table E-13. (continued)

Message	Meaning
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 list below 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*	55
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.

DDT-68K does not support the Assemble (A) command.

End of Appendix F

Index