ATARI CORP
1196 Borregas Avenue
Sunnyvale, CA 94086

GEM DOS PROGRAMMERS GUIDE

Table of contents:

1 ASSEMBLER OPERATION

The GEM DOS Assembler, AS68, assembles an assembly language source program for execution on the 68000 microprocessor. It produces a relocatable object file and, optionally, a listing. You can find a summary of the AS68 instruction set at the end of this section. Exceptions and additions to the standard Motorola instruction set appear in sections 1.6 and 1.7.

1.2 INITIALIZING AS68

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

```
{a}AS68 -I AS68INIT
```

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

1.3 INVOKING THE ASSEMBLER (AS68)

Invoke AS68 by entering a command with the following form:

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

Table 1-1 lists and describes the AS68 command line options.

Table 1-1. Assembler Options

Option Meaning

-F pathname

Specifies the directory in which the temporary files are created. If this option is not specified, AS68 creates the temporary files in the current default directory.

-I

Initializes the assembler. See Section 1.2 for details.

-P

If specificed, AS68 produces and prints a listing on the standard output device, which, by default, is the console. Redirect the listing, including error messages, to a file with the >listing filename parameter. Note that error messages are produced whether or not the -P option is specified. No listing is produced, however, unless you specify the -P option.

-S pathname

Indicates the directory that contains the assembler initialization file, AS68SYMB.DAT. This file is created when you initialize AS68. AS68 reads the file AS68SYMB.DAT before it assembles a source file. If you do not specify this option, AS68 assumes the initialization file is located in the current default directory.

-U

Causes all undefined symbols in the assembly to be treated as global references.

-L

Ensures all address constants are generated as longwords. Use the -L option for programs that require more than 64K for execution or if the TPA is not contained in the first 64K bytes of memory. If -L is not specified, the program is assembled to run in the first 64K bytes of memory. If an address in the assembly does not fit within one word, an error occurs.

-N

Disables optimization of branches on forward references. Normally, AS68 uses the 2-byte form of the conditional branch and the 4-byte BSR instruction wherever possible (instead of the 6-byte JSR instruction) to speed program execution and reduce instruction size.

-T

Enables AS68 to accept the 68010 microprocessor opcodes.

source filename

Specifies the file to assemble; you must supply this parameter.

>listing filename

Sends a program listing to the standard output device. Use the greater-than symbol, >, to direct the listing to a disk file. The listing includes assembler error messages. Note that if you do not specify -P with a listing filename, only the error messages are redirected to the listing file.

1.4 ASSEMBLY LANGUAGE DIRECTIVES

Table 1-2 lists the AS68 directives.

Table 1-2. AS68 Directives

Directive	Meaning

.comm label, expression

The comm (common) directive specifies a label and the size of a common area that programs assembled separately can share. The LO68 linker links all areas with the same label to the same address. The largest common area of a group with the same label determines the final size of the program common area.

.data

The data directive instructs AS68 to change the assembler base segment to the data segment.

.bss

The bss (block storage segment) directive instructs AS68 to change the assembler base segment to the block storage segment. You cannot assemble instructions and data in the bss. However, you can define symbols and reserve storage in

the bss with the ds directive.

.dc operand[,operand,...]

The dc (define constant) directive defines one or more constants in memory. The operands can be symbols or expressions assigned numeric values by AS68, or explicit numeric constants in decimal or hexadecimal, or strings of ASCII characters. You must separate operands with commas. You must enclose string constants in single quotation marks. Each ASCII character is assigned a full byte of memory. The eighth bit is always 0.

You can specify the length of each constant with a single letter parameter (byte = b, word = w, longword = l). You must separate the letter from the dc with a period as shown in the following explanations.

.dc.b

The constants are byte constants. If you specify an odd number of bytes, AS68 fills the odd byte on the right with zeros unless the next statement is another dc.b directive. When the next statement is a dc.b directive, the dc.b uses the odd byte. Byte constants are not relocatable.

.dc.w

The constants are word constants. If you specify an odd number of bytes, AS68 fills the last word on the right with zeros to force an even byte count. The only way to specify an odd number of bytes is with an ASCII constant. Word constants can be relocated.

.dc.l

The constants are longword constants. If less than a multiple of four bytes is entered, AS68 fills the last longword on the right with zeros to force a multiple of four bytes. Longword constants can be relocated.

.ds operand

The define storage directive (ds) reserves memory locations. The contents of the memory that it reserves is not initialized. The operand specifies the number of bytes, words, or longwords that this directive reserves. The notation for these size specifications is shown below.

```
.ds.b          reserves memory locations in bytes
.ds.w          reserves memory locations in words
.ds.l          reserves memory locations in longwords
```

.end

The end directive informs AS68 that no more source code follows this directive. Code, comments, or multiple carriage returns cannot follow this directive.

.endc

The endc directive denotes the end of the code that is conditionally assembled. It is used with other directives that conditionally assemble code.

.equ (or =) expression

The equate directive (equ or =) assigns the value of the expression in the operand field to the symbol in the label field that precedes the directive. The syntax for the equate directive are:

```
label .equ expression
label = expression
```

The label and operand fields are required. The label must be unique; it cannot be defined anywhere else in the program. The expression cannot include an undefined symbol or one that is defined following the expression. Forward references to symbols are not allowed for this directive.

.even

The even directive increments the location counter to force an even boundary. For example, if specified when the location counter is odd, the location counter is incremented by one so that the next instruction or data field begins on an even boundary in memory. .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

These directives test an expression against zero for a specified condition. If the expression is true, the code following is assembled; if false, the code is ignored until an end conditional directive (endc) is found. The directives and the conditions they test are:

.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.

.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.

.reg reglist

The register mask directive builds a register mask that can be used by a movem instruction. (See Table 1-1.) One or more registers can be listed in ascending order in the

format:

```
R?[-R[/R?[-R?...]...]]
```

Replace the R in the above format with a register reference. Any of the following mnemonics are valid:

```
A0-A7
D0-D7
R0-R15
```

The following example illustrates a sample register list.

```
A2-A4/A7/D1/D3-D5
```

You can also use commas to separate registers as follows:

```
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.

1.5 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.

1.6 ASSEMBLY LANGUAGE DIFFERENCES

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

* In AS68, all assembler directives are optionally preceded by a period (.). For example,

```
.equ or equ
.ds or ds
```

* AS68 does not support, but accepts and ignores the following Motorola directives:

```
comline
mask2
idnt
opt
```

* The Motorola .set directive is implemented as the equate directive (equ).

* AS68 accepts upper- and lowercase 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.

* For AS68, all labels must terminate with a colon (:). For example,

```
A:      FOO:
```

However, if a label begins in column 1, it need not terminate with a colon.

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

```
'ABCD'      "ac14"
```

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

```
r0-r15
R0-R15
d0-d7
D0-D7
a0-a7
A0-A7
```

Upper- and lowercase references are equivalent. Registers R0-R7 are the same as D0-D7 and R8-R15 are the same as A0-A7.

* Use caution when manipulating the location counter forward in AS68. An expression can move the counter forward only. The unused space is filled with zeros in the text or data segments.

* 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 follows:

```
* = This command loads R1 with zeros.
* = Branch to subroutine XYZ.
```

Be sure to include a space after the asterisk, as the location counter is manipulated with a statement of the form:

```
*=expr
```

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

* 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. The AS68 distributed with GEMDOS does not support this model. Instead, AS68 supports a model in which a program contains three segments, text, data, and bss as described in Section 3, "Command File Format."

1.7 ASSEMBLY LANGUAGE EXTENSIONS

The following enhancements have been added to AS68 to make the assembly language more efficient:

* When the instructions add, sub, and 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.

* add, and, cmp, eor, or, sub are allowed with immediate first operands and generate addi, andi, cmpi, eori, ori, and subi instructions if the second operand is not register-direct.

* All branch instructions generate short relative branches where possible, including forward references.

* Any shift instruction with no shift count specified assumes a shift count of one. For example, asl rl is equivalent to asl #1,rl.

* A jsr instruction is changed to a bsr instruction if the resulting bsr instruction is shorter than the jsr instruction.

* The .text directive causes the assembler to begin assembling instructions in the text segment. The .data directive causes the assembler to begin assembling initialized data in the data segment.

* The .bss directive instructs the assembler to begin defining storage in the bss. No instructions or constants can be placed in the bss because the bss is for uninitialized data only. However, the .ds directives can be used to define storage locations, and the location counter (*) can be incremented.

* 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 were 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.

* 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:

```
.comm label, expression
```

The expression specifies the number of bytes 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 non-zero 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.

* 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.

* 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.

1.8 ERROR MESSAGES

Appendix D lists the error messages generated by AS68.

1.9 INSTRUCTION SET SUMMARY

This section contains two tables that describe the assembler instruction set distributed with GEMDOS. Table 1-3 summarizes the assembler (AS68) instruction set. Table 1-4 lists variations on the instruction set listed in Table 1-3. For details on specific instructions, refer to Motorola's 16-Bit Microprocessor User's Manual, third edition, MC68000UM(AD3).

Table 1-3. 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
illegal	Illegal Instruction
jmp	Jump
jsr	Jump to Subroutine
lea	Load Effective Address
link	Link Stack
lsl	Logical Shift Left
lsr	Logical Shift Right
move	Move
movem	Move Multiple Registers
movep	Move Peripheral Data
muls	Signed Multiply
mulu	Unsigned Multiply
nbcd	Negate Decimal with Extend

neg	Negate
nop	No Operation
no	One's 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 1-4. Variations of Instruction Types

Instruction		Variation
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
	andi	to sr
cmp	cmp	Compare
	cmpa	Compare Address
	cmpm	Compare Memory
	cmpi	Compare Immediate
eor	eor	Exclusive OR
	eori	Exclusive OR Immediate
	eori to ccr	
	eori to sr	
move	move	Move
	movea	Move Address
	moveq	Move Quick
	move to ccr	
	move to sr	
	move from sr	
	move to usp	
neg	neg	Negate
	negx	Negate with Extend
or	or	Logical OR
	ori	OR Immediate
	ori to ccr	
	ori to sr	OR Immediate to Status Register
sub	sub	Subtract
	suba	Subtract Address
	subi	Subtract Immediate
	subq	Subtract Quick
	subx	Subtract with Extend

2.0 GEM DOS LINKERS

Two linkers are provided with GEM DOS: LINK68 and LO68. Both create relocatable command files in the format required by the operating system.

Note that both LINK68 and LO68 produce files in CP/M-68K relocatable format. Use the RELMOD format conversion utility to translate linker output files from CP/M-68K format to the GEM DOS relocatable format. RELMOD is described in Section 2.3.

2.1 LINK68

LINK68 is a linkage editor that combines object files into a command file. LINK68 accepts object files produced by other language processors, such as AS68 or the C compiler. LINK68 produces an executable file in either contiguous or noncontiguous command file format. Noncontiguous command files are not supported by GEM DOS. See Section 3 for a description of the contiguous file format.

LINK68 resolves all references to external symbols and concatenates the object files in the order specified. The entry point of the resulting command file is the first instruction in the first object file.

Note: You must use LINK68 to create an executable command file even when a single object file contains no unresolved references.

If you use the AS68 common directive to specify a common area shared by separate modules, LINK68 resolves all common areas with the same name to the same address in

the bss segment. If more than one file specifies static storage with the same name, LINK68 uses the largest size for allocation.

2.1.1 Invoking LINK68

To invoke LINK68, enter a command of the following form:

```
LINK68 [file =] object-file-1 [,object-file-2,...object-file-n]
```

If you invoke LINK68 without any command tail, the linker lists the options, and returns to the operating system.

The first file specification is the name of the command file you want to create, and object-file-1 through object-file-n are the object files to link. For example, the following command creates the output file MATH:

```
{a}link68 math = sin,cos,tan
```

If you omit the filename to the left of the equal sign, LINK68 creates the output file using the first filename in the command line, and assigns the default filetype 68K. For example, the following command creates SIN.68K:

```
{a}link68 sin,cos,tan
```

2.1.2 LINK68 Command-Line Options

When you invoke LINK68, you can specify command-line options that control the link operation. There are two kinds of options: global and local. Global options apply to the entire link operation. Local options apply only to the individual files being linked. You enclose global options in square brackets immediately preceding the output filename (if specified) in the command line. You enclose local options in square brackets immediately following the filename to which they apply.

You can use spaces between filenames to improve readability in the command line, and you can put more than one option in square brackets by separating them with commas. LINK68 also allows you to abbreviate an option name to its shortest unambiguous form. Table 2-1 lists the LINK68 options and explains their use. Abbreviated forms are in parentheses.

Note: GEM DOS requires that the linked file have the magic number 601AH. LINK68 produces the output file with this magic number only when you request a relocatable file with contiguous text, data, and bss segments. Consequently, to create a command file to run under GEM DOS, do not select the ABSOLUTE, DATABASE[n], BSSBASE[n], or TEXTBASE[n]

options when you invoke LINK68.

Table 2-1. LINK68 Command-line Options

Option Function

ABSOLUTE (AB)

Tells LINK68 to generate an absolute file with no relocation bits. The default is a relocatable program. Note that command files must be relocatable to run under GEM DOS.

ALLMODS (AL)

Tells LINK68 to load all modules from a library, even if they are not referenced. The default is to include only those modules that are actually referenced.

BSSBASE[n] (B[n])

Sets the base address for the uninitialized data segment (bss) in discontiguous programs. The n is a hexadecimal value. The default value is the first even word after the data segment.

COMMAND (C)

Tells LINK68 that the following named file contains the rest of the command line. LINK68 ignores the rest of the main command line. Nested command files are not allowed.

The format of this option is:

COMMAND [filename]

where filename is the file containing the rest of the command line.

DATABASE[n] (D[n])

Specifies the base address of the data segment in discontiguous programs. The n is a hexadecimal value. The default is the first even word after the text segment.

IGNORE (IG)

Tells LINK68 to ignore 16-bit address overflow.

INCLUDE (IN)

Tells LINK68 to load an unreferenced module from a library. The format for this option is the following, where module-name is the module you want to load:

filename [INCLUDE [module-name]]

LOCALS (L)

Tells LINK68 to put local symbols in the symbol table. The default is no local symbols. LOCALS only applies from the point in the command line that it appears.

The NOLOCALS option turns this option off. Use LOCALS and NOLOCALS in combination to put local symbols from specific files in the symbol table. LINK68 always ignores local symbols starting with L.

NOLOCALS (NO)

See LOCALS.

SYMBOLS (S)

Tells LINK68 to put the symbol table in the output file. The default is no symbol table in the output file.

TEMPFILES[d:] (TEM[d:])

Tells LINK68 to use drive d for temporary files. The default is the currently logged-in drive. If you use TEMPFILES, it must precede any input files on the command line.

TEXTBASE[n] (TEX[n])

Specifies the base address for the text segment. The n is a hexadecimal value. The default is 0H.

UNDEFINED (U)

Tells LINK68 to ignore the presence of undefined symbols in the input files. LINK68 lists the undefined symbols, then continues processing. The default action is to list any undefined symbols, then stop processing.

The following are examples of LINK68 command lines. Addresses are in hexadecimal. The first command line example links the files FOOMAIN and FOOLIB into a command file named FOOBAZ. It also tells LINK68 to include the symbol table in FOOBAZ, and place the temporary files on drive B.

```
{a}link68 [sym, tem[b:]] foobaz = foomain, foolib
```

The next example tells LINK68 to read the command line from the file LINKIT.INP. Note that closing brackets are not needed.

```
{a}link68 [com[linkit.inp
```

The file LINKIT.INP might contain the following commands:

```
{a}link68 [ab, tex[500], d[2a00], b[3000]]
          screen = scrns1 [1], iolib[a1]
```

This command creates the file SCREEN from the files SCRNS1 and IOLIB. The command tells LINK68 to create SCREEN as an absolute file with the text segment starting at 500H, the data segment starting at 2A00H, and the initialized data segment starting at 3000H. It also tells LINK68 to include local symbols from SCRNS1 and all the modules in IOLIB.

2.1.3 REDIRECTING DIAGNOSTIC OUTPUT

Normally, LINK68 sends all diagnostic output to the console. However, you can redirect this output by using the > character in the command line. For example, the following command creates MYFILE.68K on drive A, using drive B for the temporary files, and sends the diagnostics output to the file LNKMSGS.TXT on drive D:

```
{a}link68 [tem[b:] myfile.68k = moda, modb
          >d:lnkmsgs.txt
```

2.2 LO68

The LO68 linker combines AS68 assembled (object) programs to produce a relocatable 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.

Note: GEM DOS requires that the linked file have the magic number 601AH. LO68 produces the output file with this magic number only when you request a relocatable file with contiguous text, data, and bss segments. Consequently, to create a command file to run under GEM DOS, do not select the -T, -Z, -D, -B, and -S options and select the -R option when you invoke LO68.

Appendix D lists the error messages that LO68 displays.

2.2.1 Invoking LO68

Invoke LO68 by entering a command of the following form. Table 2-2 describes the options.

```
LO68 [-F pathname] [-R] [-S] [-I] [-Uname]
     [-O filename] [-X] [-Zaddress]
```

```
[-Daddress] [-Baddress]
object filename [object filename]
[>message filename]
```

Table 2-2. L068 Options

Option	Meaning

-F pathname

Specifies the path name to the directory in which temporary files are created.

-R

Preserves the relocation bits so the resulting executable program is relocatable.

-S

Strips output (if specified); the output is stripped of the symbol table and relocation bits.

-I

Does not generate 16-bit address overflow messages. When you assemble a program without the AS68 -L option, the linker might generate address overflow messages if the program contains addresses longer than 16 bits.

-Uname

Forces linking of a library module that resolves the name parameter, even if the name is not referred to by any other module being linked. Normally, library modules are only linked when they are needed to resolve references in other modules. You can use this option to create a program from a library if the module resolving the name parameter calls other modules in the library.

-O filename

Gives the object file produced by L068 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

Includes all local symbols in the symbol table except those that begin with the letter L. If not specified, L068 puts only global symbols in the symbol table. This option allows you to discard compiler internally generated labels that begin with the letter L while retaining symbols

local to routines.

-Taddress*
-Zaddress*

The -T and -Z options are equivalent. The hexadecimal address given is defined by LO68 as the beginning address for the text segment. This address defaults to zero, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF. This option is useful for putting programs in ROM. Hexadecimal characters can be in uppercase or lowercase.

-Daddress*

The hexadecimal address given is defined by LO68 as the beginning address for the data segment. This address defaults to the next byte after the end of the text segment, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF. This option is especially useful for for putting programs in ROM. Hexadecimal address characters can be in uppercase or lowercase.

-Baddress*

The hexadecimal address given is defined by LO68 as the beginning address for the bss. This address defaults to the next byte after the end of the data segment, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF.

object filename [object filename]

The name of one or more object files produced by the assembler AS68. These are the object files that LO68 links.

>message filename

If specified, error messages produced by LO68 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.

2.2.2 Sample Commands Invoking LO68

{a}lo68 -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}lo68 -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}lo68 -i -o test.68k test.o test1.o > error
```

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

2.3 RELMOD FORMAT CONVERSION UTILITY

LINK68 and LO68 produce files in CP/M-68K relocatable format. Use the RELMOD command to translate linker output file from CP/M-68K format to GEM DOS relocatable format.

The RELMOD command line is in the following form:

```
RELMOD inputfile outputfile
```

where inputfile is the CP/M-68K format file produced by the linker and output file is the name of the file translated to GEM DOS format.

RELMOD can also be used to translate GEM DOS format files back to CP/M-68K. This is done by simply specifying the GEM DOS file as the input file and the CP/M-68K file as the output file.

3. ARCHIVE UTILITY

3.1 INTRODUCTION

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 GEM DOS and documented in the GEM DOS Supplement to the C Language Programmer's Guide for CP/M-68K.

3.2 AR68 SYNTAX

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

```
AR68 DRTWX[AV][F pathname][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 lists and describes the components of the AR68 command line.

Table 7-1. AR68 Command Line Components

Component	Meaning
AR68	Invokes the Archive utility. However, if you specify

only the AR68 command, AR68 returns the following command line syntax and system prompt.

```
{a}ar68

usage:  AR68 DRTWX[AV][F pathname] [OPMOD] ARCHIVE
                        OBMOD1 [OBMOD2...][>filespec]

{a}
```

DRTWX

Indicates you must specify one of these letters as an AR68 command. Each of these one-letter commands and its options are described in Section 3.4.

AV

Indicates you can specify one or both of these one-letter options. These options are described with the commands in Section 3.4.

F pathname

Specifies the path to the directory in which the temporary file created by AR68 resides. If no path name is specified, the current default directory is used. AR68 creates a temporary file called AR68.TMP that AR68 uses as a scratch pad area.

OPMOD

Indicates 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.

ARCHIVE File specification of the library.

OBMOD1 [OBMOD2 ...]

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

>filespec

Redirects the output to the file specification 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 3.4.

3.3 AR68 OPERATION

AR68 sequentially parses the command line 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 3.4 describes each of the commands AR68 supports.

When AR68 processes a command, it creates a temporary file called AR68.TMP, which it uses as a scratch pad. After the operation is complete AR68 erases AR68.TMP. However, AR68.TMP is not always erased if an error occurs. If this occurs, erase AR68.TMP with the ERA command and refer to Appendix D for error messages output by AR68.

3.4 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.

3.4.1 The D Command

The D command deletes from the library one or more object modules specified in the command. The D command supports the following option:

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 lowercase letter c and modules deleted from the library with the lowercase letter d as follows:

```
{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.

3.4.2 The R Command

The R command 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 the module names in the command

line. However, when the library contains two or more modules 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 you want replaced after the name of a module 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:

* The object module does not already exist in the library.

* You specify the A option in the command line.

* The name of the module follows the name of a 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.

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. Both the A and V options are described below.

A

The A option adds one or more object modules following the module specified in the command line:

```
{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 lowercase letter a and existing modules are preceded by the lowercase letter c.

V

The V option 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 lowercase letter r and each one that is added is preceded with the lowercase letter a.

3.4.3 The T Command

The T command requests that 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. It supports the following option.

V

The V option displays the size of each file in the table of contents as shown in the following example.

```
{a}ar68 tv wine.bad
rw-rw-rw- 0/0    6818 rose.o
rw-rw-rw- 0/0    2348 white.o
rw-rw-rw- 0/0     396 red.o

{a}
```

The T command prints a table of contents in the library WINE.BAD. In addition to listing the modules in the library, the V option requests the size of each module. The character string rw-rw-rw- 0/0 that precedes the module size is meaningless for GEM DOS. 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.

3.4.4 The W Command

The W command 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 required.

```
{a}ar68 w go.arc now.o > b:\root\newd\file.o
```

The W command writes a copy of the object module NOW.O from the library GO.ARC to the file FILE.O in the NEWD subdirectory on drive B.

3.4.5 The X Command

The X command 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. The X command supports the following option.

V

The V option lists only those modules the X command extracts from the library. It precedes each extracted module with the lowercase letter x as follows:

```
{a}ar68 xv jnnk.man saw.o ham.o screw.o
x saw.o
x ham.o
x screw.o
```

3.5 AR68 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 D 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 8 to read any error messages.

4. MORE PROGRAMMING UTILITIES

4.1 INTRODUCTION

This section describes the DUMP, SIZE68, SENDC68, and XREF programming utilities. DUMP displays the contents of files in hexadecimal and ASCII notation. SIZE68 displays the total size of a memory image command file and the size of each of its program segments. XREF produces a cross reference symbol table for GEM DOS object files. SENDC68 creates a file of Motorola S-records from a command file. (Refer to Appendix B for a detailed description of the S-record format.)

4.2 DUMP UTILITY

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

4.2.1 Invoking DUMP

Invoke DUMP by entering a command with the following input components in the following format:

```
DUMP [ -sxxxx ] filename1 [ >filename2 ]
```

Table 4-1 lists the DUMP command line components and their meanings.

Table 4-1. 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	Name of the file you want to dump.
>filename2	The greater than sign (>) followed by a filename or logical redirects the output of DUMP. You can specify any valid GEM DOS 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.

4.2.2 DUMP Output

DUMP sends its 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 *aaaaaaaaaaaaaaaa*
```

The components of a DUMP command line are as follows:

Component	Meaning
rrrr	Record number (GEM DOS records are 128 bytes) of the current line of the display.
oo	Offset (in hex bytes) from the beginning of the GEM DOS record.
ffffff	Offset (in hex bytes) from the beginning of the file.
hhhh	Contents of the file displayed in hexadecimal.

aaaaaaaa

Contents of the file displayed as ASCII characters. If any character is not representable in ASCII, it is displayed as a period (.).

4.2.3 DUMP Examples

In the following example, DUMP is invoked to display the contents of a command file that contains data in both binary and ASCII form.

```
{a}dump dump.68k

00 (000000): 601a 0000 .... 0000 *`....4.......^..*
10 (000010): 0000 0000 .... 4320 *............`4C *
20 (000020): 5275 6e74 .... 6768 *RuntimeCopyright*
30 (000030): 7420 3139 .... 7461 *t 1982 by Digita*
40 (000040): 6c20 5265 .... 2c30 *l Research V01.0*
50 (000050): 3320 206f .... 001c *3  o.."h..&ISh..*

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

4.3 SIZE68 UTILITY

The SIZE68 utility (SIZE68) indicates if the program segments within one or more command files are contiguous or non-contiguous, displays the size of each program segment and the symbol table, and reports if the command files are relocatable or non-relocatable. SIZE68 displays both decimal and hexadecimal values for the sizes of the program segments and the symbol table. GEM DOS command files usually have a filetype of .PRG or .REL. The total size of a command file's segments returned by SIZE68 and the size of a command file returned by the DIR command are not equal. The file size returned by SIZE68 includes the size of the text, data, and bss program segments and the size of the symbol table but does not include the size of the header and and relocation bits. For more details on the DIR command, refer to the GEM DOS User's Guide.

4.3.1 Invoking SIZE68

Invoke SIZE68 by entering a command line with the following format:

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

The SIZE68 command line components have the following meaning:

Component	Meaning

filename

File specification of a file whose size you want to determine.

filename2 filename3 ....

One or more additional file specifications of files whose sizes you want to determine. SIZE68 can process multiple files, provided the command line does not exceed 128 bytes. Note that SIZE68 also accepts wildcard file specifications.

>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 GEM DOS filename, or one of the logical device names, CON:(console) or LST: (list device).

4.3.2 SIZE68 Examples and Output

This section contains two examples that show SIZE68 command lines and output.

1. The following SIZE68 command line example returns information about the program segments in one command file.

```
{a}SIZE68 SIZE68.PRG
SIZE68.PRG:
     Contiguous
     .text length              =   9312        2460
     .data length              =   1178         49A
     .bss length               =   9140        23B4
     Symbol table length       =      0           0
     Start of .text            =      0           0
     File is relocatable.
```

SIZE68.PRG contains a 9312-byte (decimal) text segment, a 1179-byte (decimal) data segment, and a 9140-byte (decimal) bss; the segments are contiguous and SIZE68 is relocatable. Hexadecimal notations for the decimal values are displayed in the last column of SIZE68 output.

2. The following SIZE68 command line uses a wildcard file specfication to return information on an object file, a command file, and a text file.

```
{a}SIZE68 FI*.*
 FIND.O:
        Contiguous
        .text length              =   1072         430
```

```
            .data length              =    188          BC
            .bss length               =      0           0
            Symbol table length       =   1708         6AC
            Start of .text            =      0           0
            File is relocatable.

    FIND.PRG:
            Contiguous
            .text length              =   9888        26A0
            .data length              =   1060         424
            .bss length               =   9396        24B4
            Symbol table length       =      0           0
            Start of .text            =      0           0
            File is relocatable.

    FILE.MSG:
            Not a program file.
```

Notice that when you specify a file that is not a command file (FILE.MSG, an ASCII file, for example), SIZE68 displays:

```
Not a program file.
```

When you specify an absolute program file whose segments are non-contiguous in a SIZE68 command line, SIZE68 includes the following messages in its output:

```
Non-contiguous

No relocation information in file.
```

4.4 SENDC68 UTILITY

SENDC68 creates a file with Motorola S-record format from an absolute command file. S-records are a way to represent an absolute program in ASCII character form. For a detailed description of the S-record format, refer to Appendix E.

4.4.1 Invoking SENDC68

Invoke SENDC68 by entering a command line in the following format:

```
SENDC68 [-] inputfile [outputfile]
```

The SENDC68 command line components are described below.

Component	Meaning
-	A hyphen is optional. If you specify a hyphen, SENDC68 does not create any S-records for the bss segment.

The result is a smaller S-record file. If you do not specify a hyphen, SENDC68 fills the bss segment with zeros.

inputfile

File that SENDC68 converts to the S-record format. The command file must be an absolute file in the format produced by LINK68, LO68, or RELOC.

outputfile

File that SENDC68 sends the new S-record file to. If you do not specify an output file, SENDC68 sends the S-records to the console screen.

4.4.2 SENDC68 Example

The following command line example illustrates how to convert an absolute command file into a file in the Motorola S-record format. In this example, SENDC68 creates an S-record file named PROG.SR from an absolute command file named PROG.68K.

```
{a}>sendc68 - prog.68k prog.sr
```

Note that the hyphen directs SENDC68 not to create S-records for the bss segment.

4.5 XREF UTILITY

The XREF utility generates a cross-reference table of symbols for GEM DOS object files. XREF output lists the symbols, the object file in which they are defined, and the object files in which they are accessed. XREF provides a separate listing for undefined symbols and the object files that call them. XREF accepts wildcards in the object file specifications.

4.5.1 Invoking XREF

To use XREF, enter a command of the following format:

```
XREF objectfile1 [objectfile2 ... objectfilelast]
```

Where "objectfile1" is the name of the object file for whose symbols you want a cross-referenced table. Multiple object file names may be specified.

4.5.2 XREF Examples

This section contains two example XREF command lines and shows the format of XREF output.

1. The following XREF command generates a cross-referenced table of symbols for the file

GSXVAR.O.

```
{a}XREF GSXVAR.O

Block Data:
  Symbol        Defined In   Accessed In

  cur_ms_s        GSXVAR
  disab_cn        GSXVAR
  draw_fla        GSXVAR

     .               .

     .               .

 _angle           GSXVAR      MONOBJ    MONOUT
 _beg_ang         GSXVAR      MONOBJ    MONOUT

 Undefined External References:
  Symbol       Defined In  Accessed In

_ABLINE        ********  MONOUT      _CHUP
```

2. The example XREF command shown below uses a wildcard file specification to generate a cross-referenced table of symbols for each object file in the current directory.

```
{a}XREF *.O

Functions:
  Symbol       Defined In  Accessed In

_arb_cor       MONOBJ      MONOUT
_arrow         MONOUT
_Calc_pt       MONOUT

     .     .

     .     .
```

5. SID68 DEBUGGER

SID68 allows you to test and debug programs interactively in the GEM DOS environment. The presentation of information in this section assumes you are familiar with the MC68000 microprocessor, the assembler (AS68), and the GEM DOS operating system.

5.1 INVOKING SID68

Invoke SID68 by entering one of the following commands:

```
SID
SID filename
```

The first command loads and executes SID68. After it is loaded, SID68 displays its sign-on message and the hyphen (-) prompt character to show it is ready to accept commands.

The second command invokes SID68 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 following sequence in which the first command is issued and then, at the SID68 prompt, the E command is issued to load a file for execution under SID68:

```
{a}SID

SID68 Copyright 1982, Digital Research

-Efilename
```

5.1.1 SID68 Command Conventions

When SID68 is ready to accept a command, it prompts you with a hyphen (-). In response, you can type a command line or a Ctrl-C (^C) to end the debugging session. A command line can have as many as 64 characters, and must be terminated with a Return. When entering the command, use standard GEM DOS line-editing functions to correct typing errors. SID68 does not process the command line until you enter a Return.

Table 9-1 summarizes SID68 commands. They are defined individually later in this section.

Table 9-1. SID68 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	Use hexadecimal arithmetic.
I	Set up file control block and command tail.
L	List memory using MC68000 mnemonics.
M	Move memory block.
P	Set and remove permanent breakpoints.
R	Read disk file into memory.
S	Set memory to new values.
T	Trace program execution.
U	Untrace program monitoring.
V	Show memory layout of disk file read.
W	Write contents of memory block to disk.
X	Examine and modify CPU state.

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

See Section 5.2 for more details on each command.

5.1.2 Address Specifications

Most SID68 commands require one or more addresses as operands. All addresses are entered as hexadecimal numbers of up to eight hexadecimal digits (32 bits).

5.1.3 Symbol References

SID68 allows you to reference symbols in place of addresses. You can display the symbols, along with the other disassembled instructions in your executable file, by

using the L command described in Section 5.2.7. When entering a command that specifies a symbol, precede the symbol name with a period as follows:

```
command.symbol
```

For example, to direct the GO command to execute from the current Program Counter (PC) to the symbol QUIT in the object file, enter:

```
g,.quit
```

The C compiler puts an underscore (_) at the beginning of external symbols. For example, to specify the GO command with a breakpoint at the C function BLIVOT, enter:

```
g,._blivot
```

5.1.4 Stopping SID68

Stop SID68 by typing a ^C in response to the hyphen prompt. This returns control to the CCP.

5.1.5 SID68 Operation with Interrupts

SID68 operates with interrupts enabled or disabled, and preserves the interrupt state of the program being executed under SID68. When SID68 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 SID68 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 SID68 uses the Trace and Illegal Instruction exceptions. Therefore, programs debugged under test should not use these.

5.2 SID68 COMMANDS

This section defines SID68 commands and their arguments. SID68 commands allow you to control program execution and display and modify system memory and the CPU state.

5.2.1 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 SID68 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 cc's represent the contents of memory in ASCII. Any non-graphic ASCII characters are represented by periods.

In response to the Ds form of the D command, shown above, SID68 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 or word values, as shown below:

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

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, you can abort the D command by typing any character at the console.

5.2.2 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

Efilename

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

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

When the load is complete, SID68 displays the starting and ending addresses of each segment in the file. 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, SID68

displays an error message. See Appendix A for error messages returned by SID68.

5.2.3 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, SID68 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, SID68 responds with a question mark. Also, if b is greater than FF hexadecimal (255), w greater than FFFF hexadecimal (65,535), or l greater than FFFFFFFF hexadecimal (4,294,967,295), SID68 responds with a question mark. SID68 displays an error message if it cannot read back the value stored in memory successfully. This error indicates a faulty or non-existent RAM location.

5.2.4 The G (Go) Command

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

```
G G,b1,...b10 Gs Gs,b1,...b10
```

where s is the address at which the program begins executing and b1 through b10 are addresses of breakpoints.

In the first two forms, no starting address is specified. SID68 starts executing the program at the address specified by the Program Counter (PC). The first form transfers control to the program without setting any breakpoints. The second form sets breakpoints before passing control to the program. The last 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 it encounters a breakpoint. At this point, SID68 regains control, clears all breakpoints, and displays the CPU state in the same form as the X command. When a breakpoint returns control to

SID68, 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.

5.2.5 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 SID68 computes. SID68 displays the sum (ssssssss) and the difference (dddddddd) truncated to 16 bits as follows:

```
ssssssss dddddddd
```

5.2.6 The I (Input Command Tail) Command

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

```
Icommandtail
```

where commandtail is the character string which usually contains one or more filenames. The first filename is passed into the default File Control Block at 005CH. The optional second filename, if specified, is passed 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, SID68 copies the File Control Block and command buffer from the base page of SID68 to the base page of the program loaded.

5.2.7 The L (List) Command

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

```
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 SID68 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 pressing any key during the list process. Or, enter Ctrl-S (^S) to halt the display temporarily. A Ctrl-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). Section 5.2.17 describes some minor differences between the assembly language statements produced by the L command and standard Motorola 68000 assembly language.

5.2.8 The M (Move) Command

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

```
Ms,f,d
```

where s is the starting address of the block to be moved, f is the address of the final byte of the block 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.

5.2.9 The P (Pass Points) Command

The P command sets, clears, and displays pass points. The forms are as follows:

```
Pa,n Pa -P
```

A pass point is a permanent breakpoint that remains in effect until you explicitly remove it, as opposed to breakpoints set with the G command that must be reentered with each G command. Pass points have associated pass counts ranging from 1 to 0FFFFH. The pass count indicates how many times the instruction at the pass point executes before the control returns to the console. SID68 can set up to 16 pass points at a time.

An important distinction between breakpoints and pass points is that when execution stops at a breakpoint, the instruction at the breakpoint has not been executed. When execution stops due to a pass point whose pass count has reached 1, the instruction at the pass point has been executed. This makes it simple to proceed from a pass point

with a G command without encountering the same pass point.

Forms Pa,n and Pa set pass points. Form Pa,n sets a pass point at address a (pass point address) with a pass count of n (from 1 to OFFFH). If a pass point is already active at a, the pass count is changed to n. SID68 responds with a question mark if there are already 16 active pass points.

Form Pa sets a pass point at address a with a pass count of 1. If a pass point is already active at address a, the pass count is changed to 1 if it is not 1. SID68 responds with a question mark if there are already 16 active pass points.

The -P form is used to clear pass points.

5.2.10 The R (Read) Command

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

```
Rfilename
```

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

SID68 reads the file into memory and displays the starting and ending addresses of the block of memory occupied by the file. A Value (V) command can redisplay the information at a later time. The default display pointer, for subsequent Display (D) commands is set to the start of the block occupied by the file.

5.2.11 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 at which the change is to occur.

SID68 displays the memory address and its current contents. In response to the first form, the display is

```
aaaaaaaa bb
```

In response to the second form, the display is

```
aaaaaaaa wwww
```

In response to the third form, the display is

aaaaaaaa 11111111

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

In response to one of the above displays, you can alter the memory location or leave it unchanged. If you enter a valid hexadecimal value, the contents of the byte, word, or longword in memory is replaced with that value. If you do not enter a value, the contents of memory are unaffected and the contents of the next address are displayed. In either case, SID68 continues to display successive memory addresses and values until either a period or an invalid value is entered.

SID68 displays an error message if it cannot read back the value stored in memory successfully. This error indicates a faulty or non-existent RAM location.

5.2.12 The T (Trace) Command

The T command traces program execution for 1 to 0FFFFFFFFH program steps. The forms are

T Tn Tw

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

After SID68 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 program counter. If you do not specify n, one instruction is executed. Otherwise, SID68 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 pressing any character at the console.

The Tw form traces execution without calls to subroutines. The entire subroutine called from the program level being traced is treated as a single program step and executed in real time. This allows tracing at a high level of the program, ignoring subroutines that are already debugged.

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

Note that SID68 does not trace through a BDOS interrupt instruction since SID68 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.

5.2.13 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 command before all the steps have been executed by pressing any key at the console.

5.2.14 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, SID68 responds to the V command with a question mark.

5.2.15 The W (Write) Command

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

```
Wfilename Wfilename,s,f
```

The filename is the file specification of the disk file that receives the data.

If you use the first form, SID68 assumes the values for s and f from the last file read with an R command. If no file has been read by an R command, SID68 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.

In the second form 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, SID68 uses the same value that was used for s.

If the file specified in the W command already exists on disk, SID68 deletes the existing file before it writes the new file.

5.2.16 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 (ST, displayed by field), all eight data registers, all eight address registers, and the disassembled instruction at the memory address currently in the PC. The forms are

```
X Xr
```

where r is one of the following registers:

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

The first form displays the CPU state as follows:

```
PC=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, you see the values of each bit in the status register, as shown in the following sample:

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

This sample display includes:

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

The second form, Xr, allows you to change the value in

the registers of the program being tested. The r identifies the register. SID68 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 then a Return, the register is changed to the new value. You can change the contents of all registers except the status register.

5.2.17 Assembly Language Syntax for the L Command

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

* SID68 prints all numeric values in hexadecimal.

* SID68 uses lowercase mnemonics.

* SID68 assumes word operations unless a byte or longword specification is explicitly stated.

A. ERROR MESSAGES

This appendix lists the error messages returned by the internal components of GEM DOS and by the GEM DOS programmer's utilities. subsections are arranged alphabetically by the name of the internal component or utility. Each subsection has error messages listed alphabetically, with explanations and suggested user responses.

A.1. AS68 ERROR MESSAGES

The GEM DOS 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.

A.1.1. 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 appear at the console after assembly, followed by a message indicating the total

number of errors. In a print-out, they print on the line preceding the error. Table A-1 lists the AS68 diagnostic error messages in alphabetical order.

Table A-1. 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.

& 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 only by a single carriage return, and reassemble the source file.

& line no. illegal addressing mode

The instruction on the line indicated has an invalid addressing mode. Provide a valid addressing mode and reassemble the source file.

& line no. illegal constant

The line indicated contains an illegal constant. Supply a valid constant and reassemble the source file.

& line no. illegal expr

The line indicated contains an illegal expression. Correct the expression and reassemble the source file.

& line no. illegal external

The line indicated illegally contains an external reference to an 8-bit quantity. Rewrite the source code

to define the reference locally or use a 16-bit reference and reassemble the source file.

& line no. illegal format

An expression or instruction in the line indicated is illegally formatted. Examine the line. Reformat where necessary and reassemble the source file.

& line no. illegal index register

The line indicated contains an invalid index register. Supply a valid register and reassemble the source file.

& 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.

& 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.

& 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 non-existent 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.

& 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.

& 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.

A.1.2. User-recoverable Fatal Error Messages

Table A-2 describes fatal error messages for AS68. When an error occurs because the disk is full, AS68 creates a partial file. Erase the partial file to ensure that you do not try to link it.

Table A-2. AS68 User-recoverable Fatal Error Messages

Message	Meaning
&	cannot create init: AS68SYMB.DAT

AS68 cannot create the initialization file because the path name 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 path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reinitialize AS68. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

&	expr opstk overflow

An expression in the line indicated contains too many operations for the operations stack. Simplify the expression before you reassemble the source code.

&	expr tree overflow

The expression tree does not have space for the number of terms in one of the expressions in the indicated line of source code. Rewrite the expression to use fewer terms before you reassemble the source file.

&	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.

& Object file write error

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

& overflow of external table

The source code uses too many externally defined global symbols for the size of the external symbol table. Eliminate some externally defined global symbols and reassemble the source file.

& Read Error On Intermediate File: ASXXXXn

The disk to which AS68 was writing the intermediate text file ASXXXX is full. AS68 wrote a partial file. The variable n indicates the drive on which ASXXXX is located. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

& 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 path name. Check the path name and the filename. Respecify the command line before you reassemble the source file.

& Unable to open input file

The filename in the command line indicated does not exist or has an invalid path name. Check the path name and

the filename. Respecify the command line before you reassemble the source file.

& Unable to open temporary file

You used an invalid path name or the disk to which AS68 was writing is full. Check the path name. 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 path name used to specify the initialization file is invalid or the assembler has not been initialized. Check the path name. Respecify the command line before you reassemble the source file. If the assembler has not been initialized, refer to Section 5 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.

A.1.3. AS68 Internal Logic Error Messages

The following are messages indicating fatal errors in the internal logic of AS68:

```
& 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
```

A.2. LO68 ERROR MESSAGES

The GEM DOS 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.

A.2.1. 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 in Table A-3 in alphabetic order with explanations and suggested user responses.

Table A-3. 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 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.

: 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.

: 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 pathname

The output file indicated by pathname has an invalid path name, or the disk to which LO68 is writing is full. Check the path name. 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: pathname

Either the file, indicated by pathname, has an invalid drive code in the path name, specified by the f option, or the disk to which LO68 is writing is full. Check the path name. 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 following the error message are undefined. Provide a valid definition and reassemble the source code before you reenter the LO68 command line.

A.2.2. LO68 Internal Logic Error Messages

This section lists messages indicating fatal errors in the internal logic of LO68.

: 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

A.3. AR68 ERROR MESSAGES

The GEM DOS Archive utility, AR68, returns two types of fatal error messages: diagnostic and logic. Both types of fatal error messages show at the console as they occur.

A.3.1. Fatal Diagnostic Error Messages

Table A-4 lists AR68 fatal error messages in alphabetical order with explanations and suggested user responses.

Table A-4. AR68 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 path name for the file indicated by the variable filename is invalid, or the disk to which AR68 is writing is full. Check the path name. If it is valid, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the command line.

cannot open filename

The file indicated by the variable filename cannot be opened because the filename or the path name is incorrect. Check the path name 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 Section 3 of this manual for an explanation of the AR68 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 Section 7 for an explanation of the AR68 commands.

filename not in library

The object module indicated by the variable filename is not in the library. Ensure that you are requesting the filename of an existing object module before you reenter the command line.

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.

temp file write error

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

usage: AR68 DRTWX[AV][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.

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.

A.3.2. AR68 Internal Logic Error Messages

The following are messages that indicate fatal errors in the internal logic of AR68:

cannot reopen filename
seek error on library
Seek error on tempname
Unable to recreate--library is in filename

For the last error, Unable to recreate--library is in filename, you should rename the temporary file indicated by the variable filename. AR68 used the library to create the temporary file, 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.

A.4. DUMP ERROR MESSAGES

DUMP returns fatal, diagnostic error messages at the console. Table A-5 lists the DUMP error messages in alphabetical order with explanations and suggested user responses.

Table A-5. DUMP Error Messages

Message	Meaning
Unable to open filename	Either the path name for the input file indicated by the variable filename is incorrect, or the filename is misspelled. Check the filename and path name 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 Section 8.2 for discussion of the DUMP command line and options.

A.5. SIZE68 ERROR MESSAGES

SIZE68 returns fatal, diagnostic error messages at the console. Table A-6 lists the SIZE68 error messages in alphabetical order with explanations and suggested user responses.

Table A-6. 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, then relink the source file before you reenter the SIZE68 command line.
unable to open filename	Either the path name is incorrect, or the file indicated by the variable filename does not exist. Check the path name and filename. Reenter the SIZE68 command line.

A.6. SENDC68 ERROR MESSAGES

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

A.6.1. Diagnostic Error Messages

Table A-7 lists the SENDC68 diagnostic error messages in alphabetical order with explanations and suggested user responses.

Table A-7. 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 path name 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 path name. 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.

A.6.2. SENDC68 Internal Logic Error Messages

The following is a message that indicates a fatal error in the internal logic of SENDC68:

seek error on file filename

A.7. SID68 ERROR MESSAGES

The GEM DOS debugger, SID68, returns two types of error messages: nonfatal diagnostic error messages, and messages indicating fatal errors in the internal logic of SID68.

A.7.1. Diagnostic Error Messages

Diagnostic error messages are returned at the console as the errors occur. Table A-8 lists the SID68 error messages in alphabetical order with explanations and suggested user responses.

Table A-8. SID68 Diagnostic Error Messages

Message Meaning

Bad or non-existent 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:

* The memory location at HEX no. is Read-Only, an I/O port, or non-existent. Use another location.

* 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:

* The command file has been corrupted. Rebuild the file. Reassemble or recompile the source file, then relink the file before you reenter the SID68 command line.

* The file is linked to an absolute location in memory that is already occupied by SID68. Link the file to another location.

SID68 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 SID68 is writing has no more directory space available; in effect, the disk is full. If you have another drive available, reenter the W command and

direct the file to the disk on that drive. If you do not have another drive available, you must exit SID68 (and lose the contents of memory). Erase unnecessary files, if any, or insert a new disk.

Cannot open file

This error occurs during a R command. It indicates an incorrect path name or filename. Check the path name and filename before you reenter the command line.

Cannot open program file

This message occurs in response to an E command. It indicates an incorrect path name or filename. Check the path name and filename before you reenter the command line.

ERROR, no program or file loaded

This error message occurs in response to a 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 an E or R command, or by specifying the filename when you invoke SID68.

File too big -- read truncated

This message occurs during a R command when the file being read is too large to fit in memory. SID68 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 SID68 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 SID68. 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 SID68, you must reload the file in memory. If the disk was not full, it has a bad sector. You should replace the disk.

**illegal size field

This error occurs during an L command. The size field of the instruction being disassembled has an illegal value. Use a D command to display the location of the error. This error could be caused by one of three things:

* 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.

* The size field of the instruction has been corrupted. Use the debugging commands in SID68 to look for an error that causes the program to overwrite itself. Refer to Section 5 for a complete description of the SID68 commands and options.

* An invalid instruction was generated by the compiler or assembler used to create the program. Recompile or reassemble the source file before you reinvoke SID68.

Insufficient memory or bad file header

This message occurs in response to an E command. The error could be caused by one of three things:

* The system you are using does not have enough memory available. Ensure that the program and SID68 fit into the TPA. Exit SID68. Use the SIZE68 utility to display the amount of space your program occupies in memory. SID68 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 SID68 simultaneously, additional memory boards must be added to your system configuration.

* 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 might be trying to debug a file that is not a command file. If it is a command file, the header might have been corrupted. Reassemble or recompile the source file before you reenter the E command line. If the error persists, it might be caused by an error in the internal logic of SID68.

* The command file you are debugging is linked to an absolute location in memory that is already occupied by SID68. SID68 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 indicates one of three things. Try the operation again. If the error persists, you have one

of the following problems:

* 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.

* 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.

* A hardware error. If the error persists, check your hardware.

unknown opcode

This error occurs in response to a L command if the memory location being disassembled does not contain a valid instruction. The error might have been caused by one of three things:

* You gave the L command the wrong address. Reenter the L command with the correct address.

* The file is not a command file. Make sure that the file you specify is a command file and reenter the L command.

* 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 SID68 to look for an error in the program that causes it to overwrite itself. Refer to the Section 9 for a complete description of the SID68 commands and options.

A.7.2. SID68 Internal Logic Error Messages

The following are messages indicating fatal errors in the internal logic of SID68:

illegal instruction format #

Unknown program load error

B. MOTOROLA S-RECORD FORMAT

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

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

S	type	length	address	data	cksum
1	1	2	2, 4 or 6	varies	2

(size in characters)

Figure B-1. S-record Format

Table B-1. S-record Field Descriptions

Field Description

S

The ASCII character S. This signals the beginning of the S-record.

type

A digit between 0 and 9, represented in ASCII, with the exceptions that 4 and 6 are not allowed. The use of each type value is explained in Table B-2.

length

The number of character pairs in the record, excluding the first three fields. (That is, one half the number of total characters in the address, data, and checksum fields.) This field has two hexadecimal digits, representing a one-byte quantity.

address

The address at which the data portion of the record is to reside in memory. The data goes to this address and successively higher numbered addresses. The length of this field is determined by the record type in the second byte of the S-record.

data

A variable length field containing the actual data to be loaded into memory. Specify each byte of data as a pair of hexadecimal digits in ASCII.

cksum

A checksum compute over the length, address, and data fields. Compute the checksum as follows:

* add the values of the character pairs in the length, address, and data fields
* take the one's complement of the sum
* drop the most significant byte

Enter the least significant byte value in the checksum field as two ASCII hexadecimal digits.

There are eight types of S-records. They can be divided into two categories: records containing actual data and records used to define and delimit groups of data-containing records. Types 1, 2, and 3 are reserved for records in the first category; types 0, 5, 7, 8, and 9 for reserved for

records in the second category. Types 4 and 6 are not allowed. Table B-2 defines the types.

Note: All byte values in Table B-2 are expressed as two ASCII characters representing the hexadecimal value.

Table B-2. S-Record Type Definitions

Type	Meaning
0	This type is a header record used at the beginning of a group of S-records. The data field can contain any desired identifying information. The address field is two bytes long and is normally zero.
1	This type of record contains normal data. The address field is two bytes long.
2	This type is the same as type 1 except the address field is 3 bytes long.
3	This type is the same as type 1 except the address field is 4 bytes long.
5	This type indicates the number of type 1, 2, and 3 records in a group of S-records. The count is placed in the address field. The data field is empty.
7	This record signals the end of a block of type 3 S-records. If desired, the 4-byte address field can be used to contain an address at which to pass control. The data field is empty.
8	This type is the same as type 7 except that it ends a block of type 2 S-records and the address field is 3 bytes long.
9	This is the same as type 7 except ending a block of type 1 S-records and the address field is 2 bytes long.

INDEX TO GEM DOS PROGRAMMERS GUIDE

Index to Figures

Index to Tables