

M68LLD(D4)

SEPTEMBER 1979

MDOS LINKING LOADER

REFERENCE MANUAL

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

EXORciser®, EXORDisk, and EXbug are trademarks of Motorola Inc.

Fourth Edition
©Copyright 1979 by Motorola Inc.
Third Edition March 1978



TABLE OF CONTENTS

		<u>Page</u>
CHAPTER 1	GENERAL INFORMATION	
1.1	INTRODUCTION	1-1
1.2	OPERATING ENVIRONMENT	1-1
1.3	ADVANTAGES OF THE LINKING LOADER	1-1
1.4	RELOCATION	1-4
1.5	LINKING	1-4
1.6	MODULE LIBRARIES	1-6
1.7	MEMORY ASSIGNMENT	1-6
1.8	LOAD MAPS	1-6
CHAPTER 2	LINKING LOADER COMMANDS	
2.1	INVOKING THE LINKING LOADER	2-1
2.2	LOADER INPUT	2-1
2.3	COMMAND FORMAT	2-1
2.4	LOADER COMMANDS	2-2
2.4.1	Command Nomenclature	2-2
2.5	CONTROL COMMANDS	2-3
2.5.1	EXIT	2-3
2.5.2	IDOF - Suppress Printing of Module ID	2-3
2.5.3	IDON - Print Module ID	2-3
2.5.4	IF - Intermediate File	2-4
2.5.5	IFOF - Intermediate File Mode Off	2-4
2.5.6	IFON - Intermediate file Mode On	2-4
2.5.7	INIT - Initialize Loader	2-4
2.5.8	MO - Map Output	2-5
2.5.9	OBJ - Produces Load Module	2-5
2.6	LOAD DIRECTIVES	2-6
2.6.1	LIB - Library Search	2-6
2.6.2	LOAD - Load a File	2-7
2.7	STATE COMMANDS	2-7
2.7.1	BASE - Initialize Minimum Load Address	2-7
2.7.2	CUR - Set Current Location Counter	2-8
2.7.3	DEF - Loader Symbol Definition	2-9
2.7.4	END - Ending Address	2-9
2.7.5	MAP - Prints Load Maps	2-10
2.7.6	STR - Starting Address	2-10
CHAPTER 3	SAMPLE OPERATIONS WITH THE LINKING LOADER	
3.1	INTRODUCTION	3-1
3.2	SIMPLIFIED LOADER OPERATION	3-1
3.3	LOADER OPERATIONS USING INTERMEDIATE FILES	3-10
3.4	LOADER OPERATIONS USING A LIBRARY FILE/CREATING AN MDOS COMMAND	3-12
3.5	LOADER OPERATIONS USING A CHAIN FILE	3-14
APPENDIX A	A SUMMARY OF LINKING LOADER COMMANDS	A-1
APPENDIX B	LINKING LOADER ERROR MESSAGES	B-1

LIST OF ILLUSTRATIONS

	<u>Page</u>
FIGURE 1-1. Load Maps - Example 1	1-3
1-2. Load Map - Example 2	1-5
1-3. Loader-Produced Memory Map	1-7
3-1. Message Program 1 (PG1)	3-2/3-5
3-2. Message Program 2 (PG2)	3-6/3-7
3-3. Message Program 3 (PG3)	3-8
3-4. Basic Loader Operation	3-9
3-5. Using an Intermediate File	3-11
3-6. Using a Library File	3-13
3-7. Listing of Chain File Invoking RLOAD	3-15
3-8. Using a Chain File and RLOAD	3-16
3-9. Map Output File Listing	3-17

GENERAL INFORMATION This chapter contains general information about the Linking Loader, including its purpose, operation, and usage.

1.1 INTRODUCTION

The MDOS Linking Loader combines relocatable object modules produced by the Resident M6800 and Macro Assemblers, M6800 Resident FORTRAN Compiler, or Resident MPL Compiler into an absolute load module. This resultant load module is in a format suitable for loading by either the EXORciser loader or disk operating system loader.

The Linking Loader is a two-pass loader requiring each input module to be read twice. During Pass 1, a global symbol table is constructed describing the attributes of the various global symbols. During Pass 2, the input modules are read again and assigned absolute memory addresses. Module relocation and linking is performed during the second pass, and an absolute load module is produced.

1.2 OPERATING ENVIRONMENT

The minimum equipment required to use the Linking Loader is:

- a. An EXORciser system consisting of a terminal, a disk drive, and a monitor program.
- b. An EXORDisk II or EXORDisk III floppy disk drive system.
- c. An EXORciser-compatible terminal.
- d. 24K of Random Access Memory.
- e. Motorola Disk Operating System software (MDOS).

1.3 ADVANTAGES OF THE LINKING LOADER

In conjunction with the Resident M6800 Assembler, Macro Assembler, MPL Compiler, and FORTRAN Compiler, the Linking Loader permits the user to:

- Segment source programs and data
- Relocate object modules
- Link modules via global symbols
- Search user created libraries to satisfy unresolved global symbols
- Dynamically assign memory
- Create a memory map describing the location of each object module and data block loaded
- Create a larger system than possible without linking by making smaller assembly modules.

ASCT - Absolute Section (non-relocatable)

There may be an unlimited number of absolute sections in a user's program. These sections are used to allocate/load/initialize memory locations assigned by the programmer rather than the loader; for example, addresses assigned to ACIA's and PIA's.

BSCT - Base Section (direct addressing)

There is only one base section. The Linking Loader allocates portions of this section to each module that needs space in BSCT. BSCT is generally used for variables that will be referenced via direct addressing. BSCT is limited to locations within the addressing range of 0 through 255 (\$0 through \$00FF).

CSCT - Blank Common (uninitialized)

There is only one CSCT. This section is used for blank common (similar to FORTRAN blank common). This section cannot be initialized.

DSCT - Data Section

There is only one data section. The Linking Loader allocates portions of this section to each module that needs a part of DSCT. DSCT is generally used for variables (RAM) which are to be accessed via extended mode addressing (\$100-\$FFFF).

PSCT - Program Section

PSCT is similar to DSCT except that it is intended to be used for instructions. The PSCT/DSCT division was made to facilitate a RAM/ROM dichotomy.

This section concept is preserved by the Loader during the load process. As a module is being loaded, each of its sections is combined with the corresponding sections of previously-loaded modules. As a result, the absolute load module produced by the Loader will contain one continuous memory area for each section type encountered during the load operation.

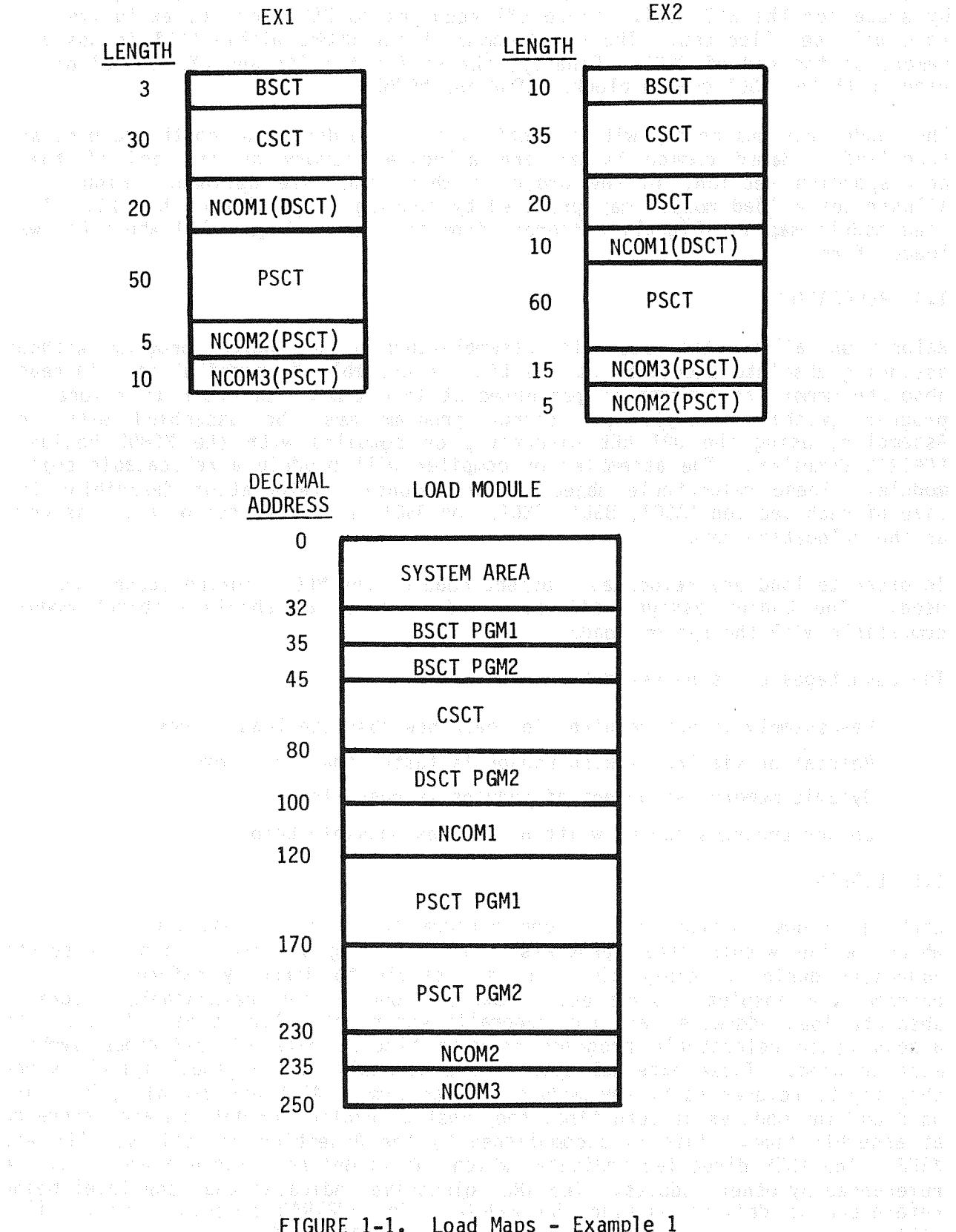
In addition to the program segmentation provided by the section concept, the relocation and linking scheme supports named common. The named common concept provides the function of initialization common areas within BSCT, DSCT, and PSCT. In processing named common definitions, the Loader will:

- Assign to each named common area a size equal to the largest size defined for the named common during the load process.
- Allocate memory at the end of each section for the named common blocks defined within that section.

The load maps shown in Figure 1-1 describe the load process with regard to sections and named common. The module EX1 requires memory to be reserved in BSCT, CSCT, DSCT, and PSCT, although the only space necessary in DSCT is for the named common NCOM1. The module EX2 requires that memory be allocated in BSCT, CSCT, DSCT, and PSCT. Neither module defines any ASCT blocks.

The hardware load map is a chart showing the physical locations of memory modules and peripheral devices.

The logical load map is a chart showing the logical locations of memory modules and peripheral devices. It shows the logical addresses and lengths of the programs and data areas.



The load module map illustrates a typical memory map that might be produced by loading EX1 and EX2. The BSCT for both EX1 and EX2 are allocated memory within the first 256 bytes of memory. As shown, the first 32 (\$20 hex) bytes of BSCT are reserved by the Loader for use by the disk operating system, unless otherwise directed. After BSCT, space for blank common is allocated, followed by space for the EX2 DSCT. Since EX1 requires no DSCT for its exclusive use, none will be allocated. The named common block NCOM1 within DSCT is assigned memory at the end of DSCT. Finally, the PSCT's for EX1 and EX2 are allocated along with the PSCT common blocks NCOM2 and NCOM3.

The Loader assigns memory within sections in the order in which the modules are specified. Named common blocks are allocated memory at the end of their corresponding section, in the order in which they are defined. Figure 1-2 illustrates a load module map produced by loading EX2, followed by EX1. This load module map is slightly different from the map in Figure 1-1 where EX1 was loaded first.

1.4 RELOCATION

Relocation allows the user to assemble/compile a source program without assigning absolute addresses at the time of assembly or compilation. Instead, absolute memory assignment is performed at load time. In order to relocate a program (within memory), the source program must be assembled with the Assembler, using the OPT REL directive, or compiled with the M6800 Resident FORTRAN Compiler. The assembler or compiler will produce a relocatable object module. These relocatable object modules contain information describing the size of each section (ASCT, BSCT, CSCT, and DSCT) and named common area, as well as the relocation data.

In order to load any relocatable object module, the MDOS Linking Loader must be used. The Loader assigns addresses and produces an absolute object module compatible with the system loader.

The advantages of using relocation are:

- Re-assembly is not required for each new absolute load address
- Relocation via the Linking Loader is faster than re-assembly
- Dynamic memory assignment of modules is possible
- Larger programs can be written than was possible before.

1.5 LINKING

Linking allows instructions in one program to refer to instructions or data which reside within other programs. If all programs are assigned absolute addresses during assembly time, it is possible to directly reference another program via absolute addresses. However, when using relocatable programs, absolute load addresses are not generally known until load time. In order to access other relocatable programs or data blocks, external reference symbols must be used. These external symbols are commonly called global symbols since they may be referenced by any module at load time. Although global symbols are used to link modules at load time, they must be explicitly defined and referenced at assembly time. This is accomplished by the Assembler directives, XDEF and XREF. The XDEF directive indicates which labels defined within a module can be referenced by other modules. The XREF directive indicates that the label being referenced is defined outside the module. For FORTRAN programs, the compiler will generate an XDEF and XREF for each SUBROUTINE and CALL statement, respectively.

loads or subloads, a load file contains one or more LOAD MODULES, each of which has a unique name. There is no defined minimum number of LOAD MODULES required for a program, though each module must be loaded before the program can begin execution.

DECIMAL ADDRESS

0

Program memory begins at address 0. It is a general area for data, programs, and other required memory. All data stored in this area is available to all programs running simultaneously. Address 32 is reserved for the system area.

32

Address 42 is reserved for the system area.

42

Address 45 is reserved for the system area.

45

LOAD MODULE

80

Address 80 contains the CSCT (Central System Control Task) program. This task is responsible for managing the system's resources and performing various system functions.

CSCT

100

Address 100 contains the DSCT (Data System Control Task) program. This task is responsible for managing data storage and retrieval.

DSCT PGM2

120

Address 120 contains the NCOM1 (Network Communication Module) program. This module is responsible for managing network communication.

NCOM1

180

Address 180 contains the PSCT (Power System Control Task) program.

PSCT PGM2

230

Address 230 contains the PSCT (Power System Control Task) program.

PSCT PGM1

245

Address 245 contains the NCOM3 (Network Communication Module) program.

NCOM3

250

Address 250 contains the NCOM2 (Network Communication Module) program.

NCOM2

FIGURE 1-2. Load Map - Example 2

At load time, global references are matched with their corresponding global definitions. Any reference within a module to a global symbol is updated with the load address of the global symbol. If the Loader detects a global reference without an associated global definition, an undefined global error will be printed and a load address of zero will be assigned to the reference.

1.6 MODULE LIBRARIES

The Linking Loader can automatically search a file for modules which contain definitions satisfying any unresolved global symbols. Such a file is called a library file and is composed of one or more object modules merged together. The Loader sequentially searches the library file. If a module is found that contains a symbol definition satisfying an unresolved global symbol, that module will be loaded. Only those modules which can satisfy an unresolved reference will be loaded. Since a library file is searched only once, modules which reference other modules within the library file should occur within the library file before the referenced module. Otherwise, the user must direct the Loader to search the library again.

1.7 MEMORY ASSIGNMENT

During the load process, absolute addresses are assigned to the program sections within the specified modules. Normally, the loader will automatically perform this assignment by allocating memory by sections in the order: ASCT, BSCT, CSCT, DSCT, and PSCT. However, the user may define the starting and/or ending address of any non-ASCT section. In this case, the Loader will first reserve memory for those sections with defined load addresses before allocating space for any other section. The Loader also permits a user to specify the relative section offset of a module within a section. However, a section of a module is always loaded in the associated load section in the order in which the module was specified. Named common blocks are always assigned memory at the end of the associated load section.

1.8 LOAD MAPS

The Loader will optionally produce a load map describing the memory layout resulting from the loading of the specified modules. Figure 1-3 is an example of some of the features included in a typical load map. In addition to this full load map, the Loader may be directed to product partial load maps listing only the undefined global symbols or section load addresses.

NO UNDEFINED SYMBOLS

MEMORY MAP

S	SIZE	STR	END	COMM
A	0006	4510	4515	
A	0006	4406	440B	
B	001A	0000	0019	0000
C	0030	0020	004F	0030
D	0042	0400	0441	0020
P	0088	1000	1087	0000

MODULE NAME	BSCT	DSCT	PSCT
PG1	0000	0400	1000
PG3	0005	040E	1060
PG2	0005	040E	1070

COMMON SECTIONS

NAME	S	SIZE	STR
DCOMM	D	0008	0422
DCOMM2	D	0018	042A

DEFINED SYMBOLS

MODULE NAME: PG1

CR	A	000D	EOT	A	0004	EXBPRT	A	F024	LF	A	000A
MSG1	P	1000	MSG2	D	0400	MSGSZ	B	0000	PGINE	P	1016
START	P	100A									

MODULE NAME: PG3

ATEST	A	4406	POWERS	P	1060
-------	---	------	--------	---	------

MODULE NAME: PG2

EXBENT	A	F564	MSG3	D	040E	MSG4	D	0418	PGM2	P	1070
STACK	B	0019									

FIGURE 1-3. Loader-Produced Memory Map



CHAPTER 2

LINKING LOADER COMMANDS

2.1. INVOKING THE LINKING LOADER is the command used to invoke the Linking Loader. It is a command of the MDOS disk operating system. The "Linking" Loader must be called while under the control of the MDOS disk operating system. When the user types the command:

`MDOS> LLOAD=RLOAD <c/r>` and the carriage return is typed, the disk executive will load the Linking Loader. Upon entry, the Loader prints:

M6800 LINKING LOADER REV n.m
? (where n.m is the revision number)

The character ? is the Loader prompt, and is printed whenever the Loader has completed the last command and is ready for another.

2.2 LOADER INPUT

The input to the Loader is in one of two forms -- commands or object modules. The Loader commands control the relocation and linking of desired object modules. Object modules are produced by the MPL Compiler, or Assembler, or Resident FORTRAN Compiler. Each source program assembled or compiled creates a single relocatable object module on a disk file. These disk files, or those files created by merging one or more of these files, are used as the input to the Loader. The Loader command structure provides for the loading of an entire file or selected modules within a file. In addition, a disk file may be used as a library file. The Loader may also be run under the MDOS CHAIN command.

2.3 COMMAND FORMAT

Each Loader command line consists of a sequence of commands and comments, followed by a carriage return. The first space in a command line terminates the command portion of the line, and the remainder is assumed to be comments. Multiple commands may appear on a line by using a semicolon (;) as a command separator. The format of a command line may thus be defined as:

$$[<\text{command}>[;<\text{command}>]^99]_0 [<\text{space}>[<\text{comments}>]] <\text{c/r}>$$

EXAMPLE: `STRB=0;STRD=$1000;STRP=$4000`

The commands in a command line are executed only after the Loader detects a carriage return.

If a command line is entered incorrectly, the line may be corrected in either of two manners. First, the command line may be deleted completely by typing CTRL X (the CTRL and X keys typed simultaneously). This causes the Loader to ignore the current command line, and issue a CR, LF, and await a new command input line. However, instead of deleting the entire command line, it may be corrected by deleting the character(s) in error. This is accomplished by typing a RUBOUT to delete the last character typed. The typing of a RUBOUT also causes the last character entered to be printed. After deleting the character(s) in error, the

corrected version of the command line may be entered. The (MDOS) CTRL D key allows the operator to redisplay the line to show a "clean" copy of the line for operator inspection. Thus, full compatibility is maintained with the normal MDOS .KEYIN special character functions.

The Loader will execute all the commands in a command line before another prompt is issued. If an error is detected while attempting to process a command, that command will be terminated. The remaining commands in the command line will be ignored.

When using multiple commands per line, it should be noted that selected commands require that they are the last command on a line, and include:

- INIT
- all intermediate file commands (IF, IFOF, IFON)
- OBJ

2.4 LOADER COMMANDS

The Loader commands are divided into three classes:

1. control commands
2. load directives
3. state directives.

The control commands are used to initiate Passes 1 and 2 of the Loader, as well as to return to EXbug or the disk operating system. The load directives are used to identify the modules to be loaded. Finally, the state directives direct the assignment of memory to the various program sections and the production of a load map.

2.4.1 Command Nomenclature

<f-name> - Used to indicate the name of a disk file to be used by the Loader. Unless specified, the file is assumed to have a suffix of "R0" and drive number of Ø. For the format of the file name, consult the MDOS Manual. (Example: PG1.R0:1)

<number> - Used to indicate a decimal or hexadecimal number. Unless preceded by a \$ character (which is used to denote hexadecimal), the number will be interpreted as decimal. Unless explicitly stated otherwise, the allowable number range will be:

Ø - 65,535 (decimal)
\$Ø - \$FFFF (hexadecimal)

- [] - Used to indicate that the enclosed directive(s) is optional.
- []⁹⁹₀ - Used to indicate that the enclosed directive may be repeated from Ø to 99 times, up to a total of 79 characters maximum.
- { } - Indicates that one of the enclosed options must be used.

2.5 CONTROL COMMANDS

2.5.1 EXIT

FORMAT: `EXIT [= {<number>}] {<name1>}`

The <number> option specifies the absolute starting address of the object program. The <name1> option specifies the name of the Loader configuration file.

DESCRIPTION: The EXIT command causes control to be returned to the disk operating system after all Loader files have been closed.

The MDOS version of the Loader allows the user to define the starting execution address of the object program. If the <number> option is specified, the given absolute number will be used as the starting execution address. This address must be a valid address within the program. The <name1> option is similar to the <number> option except that <name1> must be a valid global symbol. If neither option is used, the starting address defaults to the address associated with the label appearing in the operand field of the END statement in the assembled program. If two or more modules have END statements with operands, the operand associated with the first module loaded will be used as the starting address.

2.5.2 IDOF - Suppress Printing of Module ID

FORMAT: IDOF

DESCRIPTION: This command suppresses the printing of the name and printable information associated with each object module loaded or encountered in a library file. For assembly language programs, this information is specified via the NAM and IDNT directives.

2.5.3 IDON - Print Module ID

FORMAT: IDON

DESCRIPTION: This command causes the name and printable information associated with each object module loaded or encountered in a library file to be printed at the console device. For assembly language programs, this information is specified via the NAM and IDNT directives.

2.5.4 IF - Intermediate File

FORMAT: IF=<f-name>

DESCRIPTION: The IF command defines a file to be used as an intermediate file. An intermediate file is a copy of all Pass 1 Loader commands and object modules. It is used to direct the load operation during Pass 2, instead of requiring the user to retype the Pass 1 command sequence during Pass 2. The IF command also automatically places the Loader in intermediate file mode similar to the IFON command. Like the IFON command, the IF command must be the last command in a command line.

The IF file name must be a valid disk file name and may not be the name of an existing file on the specified diskette. Upon proper exiting from the Loader, the IF file is deleted.

EXAMPLE: IF=IFILE Defines IFILE on drive 0 as the intermediate file.
 Default suffix is "IF".

2.5.5 IFOF - Intermediate File Mode Off

FORMAT: IFOF

DESCRIPTION: IFOF temporarily suppresses the creation of the intermediate file until an IFON directive is encountered. This command must be the last command in a command line.

2.5.6 IFON - Intermediate File Mode On

FORMAT: IFON

DESCRIPTION: This command directs the Loader to write all further commands and object modules onto the intermediate file. This directive remains in effect until an IFOF or Pass 2 command is detected. The IFON command must be the last command on a command line. IFON is implied when the intermediate file is defined by the IF command. If an intermediate file is to be used during Pass 2, the IFON directive must be in effect.

2.5.7 INIT - Initialize Loader

FORMAT: INIT

DESCRIPTION: INIT initializes the Loader for Pass 1. This command is performed automatically when the Loader is first initiated. The use of this command permits the user to restart the Loader when entry errors are made, without having to exit back to MDOS. Any previously created object and/or intermediate files will be deleted. The INIT command must be the last command in a command line.

2.5.8 MO - Map Output

FORMAT: MO=[<f-name>]
[<device>]

DESCRIPTION: The MO command is used to specify the media on which the map output is to be produced. The MAP output will default to the console printer.

If a file name is specified, it must not be the name of an existing disk file. The map cannot be directed to a file during Pass 2 or whenever an intermediate file is being used.

A map can be produced on the console printer or line printer by specifying the mnemonic #CN or #LP, respectively.

EXAMPLE: MO=MAPFL All output generated by the MAP command will be written on file MAPFL on drive Ø.

MO=#LP The line printer will be used for all future map output.

2.5.9 OBJ - Produces Load Module

FORMAT: OBJA=<file-name>

OBJX=<file-name>[,printed information]

DESCRIPTION: This loader command is used with the MDOS Loader to initiate the second pass of the Loader. During this pass, an object file is created on disk with the name <file-name>. This file may not be the name of an existing file on the specified disk. The file will be created on disk Ø unless disk 1 is specified in <file-name>. The type of object file produced by the Loader is determined by the command form as follows:

OBJA - This format creates an absolute memory image file suitable for loading via the MDOS LOAD command. A default file suffix of 'LO' and drive Ø will be used if none are specified.

OBJX - An object file in EXORciser loadable format (SØ, S1, and S9 records) is created via this command form. This file may not be loaded via the MDOS LOAD command without first using the MDOS EXBIN command. However, files created in EXORciser loadable format may be copied to cassette or paper tape and loaded via EXbug. A default suffix of 'LX' and drive Ø will be used if none are specified with the file name.

If an intermediate file (IF) was generated during the first pass of the Loader, the second pass automatically processes the commands entered during the first pass. In the event that an intermediate file was not created, the same sequence of commands used during the first pass must be repeated. Regardless of the use of an intermediate file, the OBJA or OBJX command must be the last command on the command line.

EXAMPLES: OBJX=SORT,BINARY SORT PROGRAM

This command initiates the second pass of the Loader, which will create an EXORciser loadable file on disk file 'SORT.LX:0'. The S0 record will contain the file named SORT and the ASCII character string 'BINARY SORT PROGRAM'.

OBJA=REPORT:1

The Loader will create the absolute object file on file
'REPORT.LO' on drive 1.

2.6 LOAD DIRECTIVES

2.6.1 LIB - Library Search

FORMAT: LIB=<f-name> [, [<f-name>]]

DESCRIPTION: The LIB command instructs the Loader to search the specified file name(s) for those modules which satisfy any undefined global references. Any module that satisfies an unresolved global reference will be loaded. A suffix of .RO and logical drive of :0 are assumed for <f-name>.

A library file is a collection of individual relocatable object modules which were merged into a single file.

Modules loaded via the LIB command may also reference global symbols that are not defined. Since a library file is searched only once for each LIB command, it should be made with care so that no module has any reference to a prior (higher level) module, or multiple passes of the same library must be done.

It should be noted that the Macro Assembler and certain compilers (FORTRAN) produce a single relocatable object module in a file. Since these single object module files can be merged together into other (library) files, the terms "object file" and "object module" are not necessarily equivalent.

EXAMPLE: LIB=MLIB:1 The modules on file MLIB.R0 on drive 1 will be searched to resolve any unsatisfied global references.

2.6.2 LOAD - Load a File

FORMAT: LOAD=<f-name>[,<f-name>]99
0

DESCRIPTION: The LOAD command directs the Loader to load the specified object files. The LOAD command is used to load one or more object modules found in the specified file(s). The LOAD command, unlike the LIB command, will load each object module found, regardless of whether or not it is needed. A suffix of .RO and logical drive :0 are assumed.

EXAMPLE: LOAD=PGM1:1 Loads all modules within file PGM1.RO on disk drive 1. The logical drive :0 is assumed if no logical drive is specified.

LOAD=PGM1, RAM:1, PGM2, PGM3 Loads all modules within files PGM1.RO on drive 0, RAM.RO on drive 1, PGM2.RO on drive 0, and PGM3.RO on drive 0.

2.7 STATE COMMANDS

2.7.1 BASE - Initialize Minimum Load Address

FORMAT: BASE [=<number>]

DESCRIPTION: The BASE command allows the user to specify an address above which his program will load. The BASE command affects only the memory assignment of CSCT, DSCT, and PSCT. Memory assignments related to BSCT, ASCT, and those sections with defined starting/ending addresses (via commands STR or END) are not affected by this command.

The use of the <number> option is used to define the lowest address which may be assigned to CSCT, DSCT, or PSCT. If the <number> option is not specified, the lowest assignable address will default to the next modulo 8 address following MDOS. This format of BASE allows the user to load his program above MDOS without having to know where MDOS ends. If the BASE command is not specified, a default address of \$20 (32 decimal) will be used as the lowest load address during memory assignment.

EXAMPLE: BASE Unassigned CSCT, DSCT, and PSCT will be assigned load addresses above MDOS.

2.7.2 CUR - Set Current Location Counter

FORMAT: CUR {B
D}=[\] <number>
P []

DESCRIPTION: The CUR command is used to modify the Loader's current relative loading address of the specified section (BSCT, DSCT, or PSCT). The CUR command must be used prior to the LOAD or LIB command so as to update the loading address first. If the '\' option is not specified, the relative load address for the appropriate section will be set equal to the given <number> starting section plus its value (see STR command). This <number> must be equal to or greater than the section's current relative load address. This form of the CUR command allows the user to start a module section at a defined address. For PSCT, the <number> entered is added to the absolute value for STRP to obtain the new PSCT load address value. The following example loads four 1K EPROM's at \$4400, \$4800, \$5000, and \$8C00 from multiple files. Each LOAD command utilizes less than \$400 bytes in PSCT (starting PSCT=\$4400).

EXAMPLE:

?STRP=\$4400	
?LOAD=FILE11, FILE12,FILE13	EPROM at \$4400
?CURP=\$400	
?LOAD=FILE21,FILE22,FILE23	EPROM at \$4800 (\$4400 + \$400)
?CURP=\$C00	
?LOAD=FILE31,FILE32	EPROM at \$5000 (\$4400 + \$C00)
?CURP=\$4800	
?LOAD=FILE41,FILE42,FILE43,FILE44	EPROM at \$8C00 (\$4400 + \$4800)

The '\' option affects the section's relative load address in a different manner. This option causes all future modules to be loaded at an address which is a power of two relative to the start of the section (2,4,8, etc.). The specified <number> defines the given power of two. This option remains in effect until the option is specified again or until the current pass of the Loader is complete. If the '\' option is in effect when memory is assigned to the starting section addresses, the starting address of the section will also be assigned a load address which is a power of two. This option does not apply to named common blocks within the specified section.

If the CUR directive is not used, each module will normally be loaded at the next load address in the appropriate section (contiguously loaded modules). However, modules created via the FORTRAN Compiler will be loaded at the next even address.

EXAMPLE:

CURP=\$100	Sets the relative PSCT location counter to \$100 plus STRP value.
CURP=\16	Causes the Loader to load all future PSCT sections at a relative address within PSCT which is modulo 16 plus the STRP value.

NOTE

When using the CUR command within an MDOS chain file, the '\' option must use '\\\' instead of '\'. (See CHAIN command description in the MDOS Manual.)

EXAMPLE: STRP=\$4001
CURP= \$400
LOAD=PG1,PG2,PG3

If each file is a single module with less than 1K of PSCT in each one, then each module's starting PSCT address would be assigned as follows:

PG1=\$4001 (pg 1 of 1K) - first 1K of PSCT
PG2=\$4401

PG3=\$4801 (pg 2 of 1K) - second 1K of PSCT

2.7.3 DEF - Loader Symbol Definition

FORMAT: DEF:<name1>=<number>[<name2>] [ASCT] [BSCT] [DSCT] [PSCT]

DESCRIPTION: The DEF command is used to define a global symbol and enter it in the global symbol table. The symbol to be defined is given by name1 and must be a valid Macro Assembler variable name. The symbol may not currently be defined. If the <number> option is used, the symbol will be defined with the given number as the relative address within the specified section. The DEF command may be used to provide another name for a previously defined symbol by using the <name2> option. <name2> must be a currently defined global symbol. The section options -- ASCT, BSCT, DSCT, PSCT -- are used to define the section associated with the defined section. ASCT is the default section.

EXAMPLE: DEF:ACIA1=\$EC10,ASCT : Defines symbol ACIA1 as an ASCT symbol with absolute address \$EC10 (hexadecimal).

2.7.4 END - Ending Address

FORMAT: END{B}{C}=<number>
D
P

DESCRIPTION: The END commands are used to set the absolute ending address of the program in the associated section (BSCT, CSCT, DSCT, PSCT). If both an ending and starting address are defined, the size described by these boundaries must be equal to or greater than the size of the associated section.

NOTE

An ending address of \$0000 will reset any previous ending address for the corresponding section.

EXAMPLE: ENDB=255 : BSCT will be allocated such that the last address reserved is 255 (decimal).

2.7.5 MAP - Prints Load Maps

FORMAT: MAP {C
F
S
U}

DESCRIPTION: The MAP commands are used to display the current state of the modules loaded or the Loader's state directives.

MAPC - Prints the current size, user defined starting address, and user defined ending address for each of the sections, as well as the size, starting address, and ending address for each ASCT defined.

MAPF - A full map of the state of the loaded modules is produced after the Loader assigns memory. This map includes a list of any undefined symbols, a section load map, a load map for each defined module and named common, and a defined global symbol map. If a user assignment error (UAE) exists, this command cannot be completed. Use the MAPC command to determine the cause of the error.

MAPS - The Loader assigns memory to those sections not defined by a user supplied starting and/or ending address. A memory load map, which defines the size, starting address and ending address for each section, is printed. If a user assignment error (UAE) exists, this command cannot be completed. Use the MAPC command to determine the cause of the error.

MAPU - Prints a list of all global references which currently remain undefined.

2.7.6 STR - Starting Address

FORMAT: STR {B
C} = {<number>
D } {<global ASCT symbol>}

DESCRIPTION: The STR commands set the absolute starting address of the associated section (BSCT, CSCT, DSCT, PSCT). Those sections whose starting address is not defined by the user will be assigned a starting address by the loader.

NOTE

A starting address of \$FFFF will reset any previous STR directive for the corresponding section. This will allow the Loader to define the starting address.

EXAMPLE: STRP=\$1000 PSCT will be allocated memory starting at \$1000.

CHAPTER 3

SAMPLE OPERATIONS WITH THE LINKING LOADER

3.1 INTRODUCTION

This chapter provides a description of the operation of the Loader in typical applications. To demonstrate the use of the Loader, a simple message printing program will be used. This program consists of three modules which reference instruction sequences or data within each other. As assembly listing of each module is shown in Figures 3-1, 3-2, and 3-3.

3.2 SIMPLIFIED LOADER OPERATION

The simplest form of the Loader's operation is shown in Figure 3-4. In this example, all three files -- PG1, PG2, and PG3 -- are loaded, and the object file PG123 is created. The sequence of steps shown in Figure 3-4 is as follows:

1. The LOAD command loads the first file, PG1.R0:0. During all load operations, a global symbol table of all external definitions and references is built.
2. The LOAD command loads the next two files, PG2 and PG3. Notice the default suffix 'R0' and drive number '0' are assumed.
3. The OBJA command starts pass 2 of the load function, which will create an absolute memory image object file named PG123 on drive 0 with the suffix 'LO'. This command also assigns memory addresses to the various program sections. The use of the OBJX command, instead of OJBA, would have a similar effect, except an EXORciser load image would be produced.
4. Since an intermediate file was not created in pass 1, all commands entered in pass 1, with the exception of MAP commands, must be repeated. In pass 2, the LOAD command generates the absolute code for the object file. Notice that all three files are loaded with one load command this time.
5. The MAPU command is not really necessary here, but was entered to verify that no undefined symbols exist.
6. A complete memory map is produced by the MAPF command. In the first part of the map (6a), any undefined external references are listed. In the next part (6b), the section type, the size, starting address, ending address, and size of the section's common block are listed for each program section. For example, PG123's DSCT area will have a size of 42 (hex) bytes, of which 20 (hex) bytes are in common. The DSCT area will start at address \$6A and end at \$AB. The starting address of the various sections for each program module is given in the next map part (6c). As seen from the map, PG2 PSCT starts at address \$FD, which corresponds to the PG2 instruction:

PGM2 CLRA

PAGE 001 PG1 .SA:1 PG1 PROGRAM TO PRINT OUT MESSAGES (MAIN)

00001 NAM PG1
 00002 OPT REL,CREF,NOG
 00003 TTL PROGRAM TO PRINT OUT MESSAGES (MAIN)
 00004 IDNT 08/10/79 MAIN MSG PROGRAM - MODULE #1

00006 * ASSEMBLY PROCEDURE: RASM 3.00 MDOS 3.00
 00007 * =RASM PG1;LN=76
 00008 *
 00009 * PROGRAM PARTS: PG1, PG2, PG3
 00010 * COMPUTER: M6800

00012 F024 A EXBPRT EQU \$F024 EXBUG PRINT ROUTINE

00014 * ASCII CHARACTER EQUATES
 00015 *
 00016 0004 A EOT EQU 4 END OF TEXT
 00017 000A A LF EQU \$A LINE FEED
 00018 000D A CR EQU \$D CARRIAGE RETURN

00020 * EXTERNAL REFERENCES
 00021 *
 00022 XREF ATTEST
 00023 XREF DSCT:MSG3,MSG4,ANY:STACK
 00024 XREF EXBENT,PGM2

00026 * EXTERNAL DEFINITIONS
 00027 *
 00028 XDEF MSG2,MSG1,EXBPRT,START,PGLINE
 00029 XDEF MSGSIZ,EOT,LF,CR

FIGURE 3-1. Message Program 1 (PG1)

PAGE 002 PG1 .SA:1 PG1 PROGRAM TO PRINT OUT MESSAGES (MAIN)

00031	* COMMON MESSAGE AREA				COMMON
00032	* (NAMED COMMON "DCOMM" IN DSCT)				STOR
00033	*				DATA
00034N 0000	DCOMM	COMM	DSCT	PTR TO MESG 1 (IN PSCT)	DATA
00035N 0000	0000 P MSG1P	FDB	MSG1	PTR TO MESG 2 (IN DSCT)	DATA
00036N 0002	0000 D MSG2P	FDB	MSG2	PTR TO MESG 3 (XREF IN DSCT)	DATA
00037N 0004	0000 A MSG3P	FDB	MSG3	PTR TO MESG 4 (XREF IN DSCT)	DATA
00038N 0006	0000 A MSG4P	FDB	MSG4	PTR TO MESG 1 (IN PSCT)	DATA
00040	* MESSAGES 1 AND 2				DATA
00041	* (NEW NAMED COMMON "DCOMM2" IN DSCT)				DATA
00042	*				DATA
00043N 0000	DCOMM2	COMM	DSCT	COMMON MESSAGE COUNT	DATA
00044N 0000	0001 A CMSGCT	RMB	1	COMMON MESSAGE	DATA
00045N 0001	0014 A CMSG	RMB	20	RESERVE 16 BYTES	DATA
00047C 0000	CSCT			BLANK COMMON SECTION	DATA
00048C 0000	0010 A MSGCST	RMB	16	RESERVE 16 BYTES	DATA
00050D 0000	DSCT			DATA SECTION	DATA
00051D 0000	4D A MSG2	FCC	\MESSAGE	2\	DATA
00052D 0009	04 A	FCB	EOT	DELINATE END OF MESSAGE	DATA
00054P 0000	PSCT			PROGRAM SECTION	DATA
00055P 0000	4D A MSG1	FCC	\MESSAGE 1\		DATA
00056P 0009	04 A	FCB	EOT		DATA
00058B 0000	BSCT			BASE SECTION	DATA
00059B 0000	0001 A MSGSIZ	RMB	1	MESG SIZE STORAGE	DATA

FIGURE 3-1. Message Program 1 (PG1) (cont'd)

PAGE 003 PG1 .SA:1 PG1 PROGRAM TO PRINT OUT MESSAGES (MAIN)

00061	* PROGRAM SECTION				
00062	* EXECUTION STARTS AT "START"				
00063	*				
00064P 000A		PSCT	PROGRAM SECTION		
00066P 000A 8E 0000 A	START	LDS	#STACK	SET UP STACK REGISTER (XREF)	
00067P 000D FE 0000 N		LDX	MSG1P	GET MESSAGE 1 POINTER	
00C68P 0010 BD F024 A		JSR	EXBPRT	PRINT MESSAGE 1	
00069P 0013 7E 0000 A		JMP	PGM2	GO TO PROGRAM 2 (XREF)	
00070	*				
00071	* PROGRAM 2 RETURNS TO THIS POINT (XDEF)				
00072	*				
00073P 0016 CE 0000 A	PGINE	LDX	#MSG3	GET MESSAGE 3 ADDRESS	
00074P 0019 BD F024 A		JSR	EXBPRT	PRINT MESSAGE 3	
00075P 001C FE 0004 N		LDX	MSG3P	GET MESSAGE 3 POINTER	
00076P 001F BD F024 A		JSR	EXBPRT	PRINT MESSAGE 3 AGAIN	
00077P 0022 CE 0000 A		LDX	#MSG4	PRINT MESSAGE 4	
00078P 0025 BD F024 A		JSR	EXBPRT		
00079	*				
00080	* MOVE MESSAGE FROM CMSG IN DCOMM2 TO BLANK COMMON				
00081	*				
00082P 0028 CE 0000 C		LDX	#MSGCST	MESSAGE DESTINATION ADDRESS	
00083P 002B FF 0003 B		STX	TOPNTR		
00084P 002E CE 0001 N		LDX	#CMSG	MESSAGE ADDRESS (FROM)	
00085P 0031 FF 0001 B		STX	FROMPT		
00086P 0034 F6 0000 N		LDA8	CMSGCT	MESSAGE LENGTH	
00087P 0037 D7 00 B		STAB	MSGSIZ	SAVE MESG LENGTH	
00088P 0039 FE 0001 B	LOOP1	LDX	FROMPT	GET SOURCE POINTER	
00089P 003C A6 00 A		LDAA	0,X	GET BYTE	
00090P 003E 03		INX		UPDATE SOURCE POINTER	
00091P 003F FF 0001 B		STX	FROMPT		
00092P 0042 FE 0003 B		LDX	TOPNTR	GET DESTINATION POINTER	
00093P 0045 A7 00 A		STA8	0,X	SAVE BYTE	
00094P 0047 08		INX		UPDATE DESTINATION POINTER	
00095P 0048 FF 0003 B		STX	TOPNTR		
00096P 0048 5A		DEC8		UPDATE CHARACTER COUNTER	
00097P 004C 26 EB 0039		BNE	LOOP1	LOOP	
00098P 004E 7E 0000 A		JMP	ATEST	GOTO PROGRAM W/ASCT REGIONS	
00100B 0001		BSCT	DIRECT ADDRESSING SECTION		
00101	* NOTE: IF FORWARD REFERENCED, EXTENDED ADDR IS USED.				
00102	* THEREFORE ALL BSCT VARIABLES SHOULD BE				
00103	* DEFINED BEFORE REFERENCED.				
00104	*				
00105B 0001 0002 A	FROMPT RMB	2	FROM POINTER		
00106B 0003 0002 A	TOPNTR RMB	2	TO POINTER		
00108D 000A		DSCT	DATA SECTION		
00109D 000A 96 01 B		LDA8	FROMPT	***DIRECT ADDRESSING USED***	
00110D 000C DE 03 B		LDX	TOPNTR	(EXAMPLES ONLY - NOT EXECUTED)	
00112		TTL	CROSS REFERENCE TABLE		
00113 000A P		END	START		
TOTAL ERRORS 00000--00000					

FIGURE 3-1. Message Program 1 (PG1) (cont'd)

PAGE 004 PG1 .SA:1 PG1 CROSS REFERENCE TABLE

FIGURE 3-1. Message Program 1 (PG1) (cont'd)

PAGE 001 PG2 .SA:1 PG2 MESSAGE PRINTER SUBPROGRAM

00001 NAM PG2
 00002 OPT CREF,REL,NOG
 00003 TTL MESSAGE PRINTER SUBPROGRAM
 00004 IDNT 08/10/79 MESG PRNTR SUBPROG - MODULE #2

00006 * ASSEMBLY PROCEDURE: RASM 3.00 MDOS 3.00
 00007 * =RASM PG2;LN=76
 00008 *
 00009 * PROGRAM PARTS: PG1, PG2, PG3
 00010 * COMPUTER: M6800

00012 F564 A EXBENT EQU \$F564 EXBUG ENTRY POINT

00014 *
 00015 * XDEFS AND XREFS
 00016 *
 00017 XDEF MSG3,MSG4,STACK,EXBENT,PGM2
 00018 XREF BSCT:MSG5IZ
 00019 XREF EXSPRT,PG1NE,MSG1,MSG2
 00020 XREF EOT,CR,LF

00022 * MESSAGE POINTER AREA (DCOMM)
 00023 *
 00024N 0000 DCOMM COMM DSCT
 00025N 0000 0002 A MSG1PT RMB 2
 00026N 0002 0002 A MSG2PT RMB 2
 00027N 0004 0002 A MSG3PT RMB 2
 00028N 0006 0002 A MSG4PT RMB 2

00030N 0000 DCOMM2 COMM DSCT
 00031N 0000 17 A CMSGCT FCB CMSGE-CMSG . COMMON MESSAGE CHAR COUNT!
 00032N 0001 43 A CMSG FCC \COMMON TEST PROGRAM\
 00033N 0014 0C A FCB CR,LF,LF,EOT
 00034 0018 N CMSGE EQU * END OF MESSAGE

00036 * MESSAGES 3 AND 4
 00037 *
 00038D 0000 DSCT
 00039D 0000 4D A MSG3 FCC \MESSAGE 3\
 00040D 0009 00 A FCB EOT
 00041D 000A 4D A MSG4 FCC \MESSAGE 4\
 00042D 0013 00 A FCB EOT

FIGURE 3-2. Message Program 2 (PG2)

PAGE 002 PG2 .SA:1 PG2 MESSAGE PRINTER SUBPROGRAM

00044 * START OF PROGRAM 2
 00045 *

00046P 0000	PSCT	10000
00047P 0000 4F	PGM2 CLRA	80000
00048P 0001 97 00 A	STA A MSGSIZ	10000
00049P 0003 FE 0000 N	LDX MSG1PT	10000
00050P 0006 BD 0000 A	JSR EXBPRT	10000
00051P 0009 CE 0000 A	LDX #MSG2	10000
00052P 000C BD 0000 A	JSR EXBPRT	10000
00053P 000F FE 0002 N	LDX MSG2PT	10000
00054P 0012 BD 0000 A	JSR EXBPRT	10000
00055P 0015 7E 0000 A	JMP PG1NE	10000

00057B 0000
 00058B 0000
 00059B 0014

BSCT RMB 20
 RMB 1

DIRECT ADDRESSING SECTION
 STACK STORAGE AREA

00061 END

TOTAL ERRORS 00000--00000

ND 0001 CMSG 00031 00032*
 ND 0000 CMSGCT 00031*
 ND 0018 CMSGE 00031 00034*
 R CR 00020*00033
 ND DCOMM 00024*
 ND DCOMM2 00030*
 R EOT 00020*00033 00040 00042
 D F564 EXBENT 00012*00017
 R EXBPRT 00019*00050 00052 00054
 R LF 00020*00033 00033
 R MSG1 00019*
 ND 0000 MSG1PT 00025*00049
 R MSG2 00019*00051
 ND 0002 MSG2PT 00026*00053
 DD 0000 MSG3 00017 00039*
 ND 0004 MSG3PT 00027*
 DD 000A MSG4 00017 00041*
 ND 0006 MSG4PT 00028*
 RB MSGSIZ 00018*00048
 R PG1NE 00019*00055
 DP 0000 PGM2 00017 00047*
 DS 0014 STACK 00017 00059*

FIGURE 3-2. Message Program 2 (PG2) (cont'd)

PAGE 001 PG3 .SA:1 PG3 ***PROGRAM TO ILLUSTRATE USE OF ASCT

00001 NAM PG3
00002 TTL ***PROGRAM TO ILLUSTRATE USE OF ASCT
00003 OPT REL,CREF
00004 IDNT 08/10/79 ASCT ILLUSTRATION - MODULE #3

00006 * ASSEMBLY PROCEDURE: RASM 3.00 MDOS 3.00
00007 * =RASM PG3:1;LN=76
00008 *
00009 * PROGRAM PARTS: PG1, PG2, PG3
00010 * COMPUTER: M6800

00012 XDEF ATEST,POWERS
00013 XREF EXBPRT,EXBENT

00015 * BLANK COMMON
00016 *
00017C 0000 CSCT
00018C 0000 0030 A CMSG RMB \$30

00020A 0000 ASCT UNNECESSARY!
00021A 4406 ORG \$4406 * ORG CAUSES ASCT!
00022A 4406 CE 0000 C ATEST LDX #CMSG START OF COMMON MESSAGE
00023A 4409 7E 4510 A JMP ATEST2

00025A 4510 ORG \$4510
00026A 4510 BD 0000 A ATEST2 JSR EXBPRT PRINT MESSAGE
00027A 4513 7E 0000 A JMP EXBENT GOTO EXBUG/DON'T STOP

00029P 0000 PSCT PROGRAM SECTION
00030P 0000 0001 A POWERS FDB 1 POWERS OF TEN TABLE
00031P 0002 000A A FDB 10
00032P 0004 0064 A FDB 100
00033P 0006 03E8 A FDB 1000
00034P 0008 2710 A FDB 10000

00036 END
TOTAL ERRORS 00000--00000

D 4406 ATEST 00012 00022*
4510 ATEST2 00023 00026*
C 0000 CMSG 00018*00022
R EXBENT 00013*00027
R EXBPRT 00013*00026
DP 0000 POWERS 00012 00030*

FIGURE 3-3. Message Program 3 (PG3)

The sample outlined in the following steps can be used to load and run a program named PG123. This loader will also load programs PG2 and PG3. The loader will then start executing the code contained in PG123. It is assumed that the user has already loaded the MDOS linking loader and that the command =LOAD is the first line of the input file. The first command is =LOAD

=LOAD or **MDOS LINKING LOADER REV 03.00**
MDOS LINKING LOADER REV 03.00
COPYRIGHT BY MOTOROLA 1977

(1)?LOAD=PG1.RD:0 ----- LOAD FIRST FILE
 (2)?LOAD=PG2,PG3 ----- LOAD OTHER TWO FILES
 (3)?OBJA=PG123 ----- START PASS 2
 (4)?LOAD=PG1,PG2,PG3 ----- REPEAT PASS 1 COMMANDS
 (5)?MAPU ----- PRINT UNDEFINED SYMBOLS MAP
 NO UNDEFINED SYMBOLS
 (6)?MAPF ----- PRINT FULL MEMORY/SYMBOL MAP
 NO UNDEFINED SYMBOLS 6a

MEMORY MAP
 S SIZE STR END COMM
 A 0006 4510 4515
 A 0006 4406 440B
 B 001A 0020 0039 0000 6b
 C 0030 003B 0069 0030
 D 0042 006A 00AB 0020
 P 0073 00AC 011E 0000

MODULE NAME BSCT DSCT PSCT
 PG1 0020 006A 00AC 6c
 PG2 0025 0078 00FD
 PG3 003A 008C 0115

COMMON SECTIONS
 NAME S SIZE STR 6d
 DCOMM D 0008 008C
 DCOMM2 D 0018 0094

DEFINED SYMBOLS
 MODULE NAME: PG1 6e
 CR A 0000 EDT A 0004 EXPFRT A F024 LF A 0008
 MSG1 P 00AC MS62 D 006A MSGSIZ B 0020 PG1ME P 00C2
 START P 00B6
 MODULE NAME: PG2 6f
 EXRFNT A F564 MS63 D 0078 MSG4 D 0082 PGM2 P 00FD
 STACK B 0039
 MODULE NAME: PG3
 ATEST A 4406 POWERS P 0115 6g

(7)?EXIT ----- RETURN TO MDOS
 =LOAD PG123;V ----- LOAD OBJECT PROGRAM FILE
 *E ;P ----- START PROGRAM EXECUTION
 MESSAGE 1
 MESSAGE 1
 MESSAGE 2
 MESSAGE 3
 MESSAGE 3
 MESSAGE 4

COMMON TEST PROGRAM
 EXBUG 2.1
 *E

FIGURE 3-4. Basic Loader Operation

The fourth area of the map (6d) defines the size and starting address of any named common blocks. Thus, the PG1 variable CMSGST, which is the first variable in the DCOMM2 common block, will be located at address \$8C. The final map feature provides an alphatized list of all global symbols by modules (6e, 6f, 6g). The modules are listed in the order that they were loaded. Thus, the PG1 variable START has an absolute address of \$B6.

7. To return to MDOS, the EXIT command is used. This command may, in addition, be used to assign a starting execution address. In this example, PG123's starting address will be at address \$B6, since the variable START appears as the operand on PG1's END statement. Two alternate methods of defining the execution address are:

 EXIT=START
 or EXIT=\$B6

3.3 LOADER OPERATIONS USING INTERMEDIATE FILES

As shown in the previous example, most commands must be re-entered during pass 2 of the Loader. The use of an intermediate file eliminates the need to retype Loader commands. Figure 3-5 is an example of the use of intermediate files. Commands used in the sequence are explained below, with the exception of those commands previously discussed.

1. The intermediate file feature is invoked by defining a new file for use as the intermediate file.
2. The IDON command turns the identifier option on to allow printing of the IDNT assembly directive as entered in the files.
3. This command line shows how more than one command may be specified on the same line by using the ';' feature. The STR command is used to define the starting section addresses of \$400 and \$1000 for DSCT and PCST, respectively. These starting addresses are reflected in the map generated in pass 2.
4. The CUR command with the '\' option causes the PSCT section of each module to start at an address which is modulo \$10 from the start of PSCT. This feature permits the user to easily debug relocatable programs, since modules start at convenient addresses. Thus, in the example of Figure 3-5, the first PSCT code for module PG2 will start at \$1070.
5. Notice that the loading order is different from the example in Figure 3-4. As each file/module is loaded, its identifier is printed (5a).
6. As in the previous example, the OBJA command initiates pass 2 of the Loader. However, since the intermediate file feature is being used, the second pass 2 is automatically performed without the user re-entering the commands. Notice the identifiers are also printed here as each file/module is loaded (6a).
7. The Loader has completed processing all commands entered in pass 1; the user may now enter any non-load command such as a MAP command or EXIT. In this case, all map output is directed to the line printer with the MO=#LP command.

the following listing. It shows how to use an intermediate file to link several modules together. Note that the first four lines of the listing are part of the MDS command language.

After the first four lines, the first two lines of the listing are the contents of the intermediate file.

When using the MDS linking loader, it is often necessary to link several modules together, and then link the resulting module to another module. This can be done by creating an intermediate file which contains the commands required to link the modules together. Then, when the linking loader is run, it reads the intermediate file, and performs the linking operation indicated in the file. This allows the user to link several modules together and then link the resulting program to another module. This is a very useful feature of the MDS linking loader.

The following listing shows how to use an intermediate file to link several modules together.

```
=RLOAD  
MDS LINKING LOADER REV 03.00 08/10/79 08:00:00 AM 08/10/79 08:00:00 AM  
COPYRIGHT BY MOTOROLA 1977  
(1) ?IF=TEMP----- CREATE INTERMEDIATE FILE = TEMP  
(2) ?IDON ----- TURN ON IDENTIFIERS  
(3) ?STRD=$4000;STRP=$1000;STRB=0----- DEFINE STARTING SECTION ADDRESSES  
(4) ?CURP=>$10----- START PSCT ON MODULO TO (HEX) BOUNDARIES  
(5) ?LOAD=PG1,PG3,PG2----- LOAD FILES  
    PG1      08/10/79 MAIN MSG PROGRAM - MODULE #1  
(5a) PG3      08/10/79 ASCT ILLUSTRATION - MODULE #3  
    PG2      08/10/79 MSG PPNTR SUBPROG - MODULE #2  
(6) ?OBJA=PG132----- START PASS 2 - CONTROLLED BY INTERMEDIATE FILE  
    PG1      08/10/79 MAIN MSG PROGRAM - MODULE #1  
(6a) PG3      08/10/79 ASCT ILLUSTRATION - MODULE #3  
    PG2      08/10/79 MSG PPNTR SUBPROG - MODULE #2  
(7) ?MDP=OLP----- ASSIGN MAP OUTPUT TO LINE PRINTER  
(8) ?MAPF----- FULL MEMORY/SYMBOL MAP TO LINE PRINTER  
(9) ?EXIT----- RETURN TO MDS  
=LOAD PG132;V----- LOAD OBJECT PROGRAM FILE  
  
◆ E :P ----- START PROGRAM EXECUTION  
MESSAGE 1  
MESSAGE 1  
MESSAGE 2  
MESSAGE 2  
MESSAGE 3  
MESSAGE 3  
MESSAGE 4  
COMMON TEST PROGRAM  
EXBUG 2.1  
◆ E
```

The first four lines of the listing are part of the MDS command language. The remaining lines are the contents of the intermediate file. The intermediate file contains the commands required to link the three modules together, and then start the program execution.

The first line of the intermediate file is the command ?LOAD, followed by the names of the three modules to be linked together.

The second line of the intermediate file is the command ?OBJA, followed by the name of the output file for the linked program.

FIGURE 3-5. Using an Intermediate File

8. A full map is sent to the line printer to produce a hard copy with the MAPF command. The line printer map output is shown in Figure 1-3.
9. The object file is closed and control is returned to MDOS via the EXIT command.

3.4 LOADER OPERATIONS USING A LIBRARY FILE/CREATING AN MDOS COMMAND

The previous examples have described the loading procedure performed via the LOAD command. In these examples, the user was aware of each module that had to be loaded. However, in other cases, the user may be aware of only the entry point name required to perform a desired function. In such instances, the user can create a file which contains a collection of utility modules. The Loader may be used to extract only the required modules from this library file. The use of a library file is shown in Figure 3-6, and a description of the various steps is explained below:

1. The MDOS MERGE command is used to build a library file PGLIB. This file contains the modules in files PG1, PG2, and PG3.
2. The use of the BASE command directs the Loader to assign memory for CSCT, DSCT, and PSCT above the MDOS system area. As a result, the user program may be invoked directly as an MDOS command without using the LOAD command. However, if the program has initialized BSCT, the MDOS LOAD command must be used to execute the program. The effect of the BASE command is shown in the program's memory map where CSCT, DSCT, and PSCT are assigned memory above \$2000.
3. All currently undefined symbols are listed via the MAPU command. In this example, the six undefined symbols correspond to the six external references in PG1.
4. The LIB command searches the file PGLIB for any modules which satisfy the current undefined symbols. Since PG2 and PG3 are modules in PGLIB that satisfy these undefined symbols (i.e., PG2 and PG3 have XDEF's for ATTEST, EXBENT MSG3, MSG4, PGM2, and STACK), they will be loaded via the LIB command. PG1, which is also in PGLIB, will not be loaded again.
5. The second MAPU command shows that all external references have now been satisfied.
6. The second pass of the Loader is initiated with the OBJA command, and creates an object file with the name MESSAGE. The use of the suffix 'CM', along with the Loader's BASE command, permits the created file to be treated as an MDOS command (see item 9).
7. Since an intermediate file was not created during pass 1, all commands entered in pass 1 must be repeated in pass 2. The MAP, END, and STR commands are the only exceptions to this rule.
8. The EXIT command completes pass 2 of the Loader and returns to MDOS.
9. The file created by the Loader is treated as an MDOS command and, therefore, is loaded and executed automatically.

(1)=MERGE PG1.RD,PG2.RD,PG3.RD,PGLIB.RD -----BUILD LIBRARY FILE
 =RELOAD
 MDOS LINKING LOADER REV 03.00
 COPYRIGHT BY MOTOROLA 1977
 (2)?BASE -----LOCATE PROGRAM ABOVE MDOS
 ?LOAD=PG1 -----LOAD FIRST FILE
 (3)?MAPU -----PRINT UNDEFINED SYMBOLS
 ATEST EXBENT MSG3 MSG4 PGM2 STACK
 0006 UNDEFINED SYMBOLS
 (4)?LIB=PGLIB -----SEARCH LIBRARY FILE
 (5)?MAPU -----PRINT UNDEFINED SYMBOLS
 NO UNDEFINED SYMBOLS
 (6)?OBJA=MESSAGE.CM -----START PASS 2 - BUILD COMMAND FILE
 (7)?BASE -----REPEAT PASS 1 COMMANDS
 ?LOAD=PG1;LIB=PGLIB
 ?MAPF -----PRINT FULL MEMORY/SYMBOL MAP
 NO UNDEFINED SYMBOLS

MEMORY MAP

S	SIZE	STR	END	COMM
A	0006	4510	4515	EXBENT MSG3
A	0006	4406	440B	MSG4
B	001A	0020	0039	0000
C	0030	2000	202F	0030
D	0042	2030	2071	0020
P	0073	2072	20E4	0000

MODULE NAME BSCT DSCT PSCT
 PG1 0000 0020 2030 2072
 PG2 0000 0025 203E 20C3
 PG3 0000 003A 2052 20DB

COMMON SECTIONS

NAME	S	SIZE	R	STR	END	COMM
DCOMM	D	0008		2052		
DCOMM2	D	0018		205A		

DEFINED SYMBOLS

MODULE NAME:	PG1							
CR	A	0000	EOT	A	0004	EXBPRT	A	F024
MSG1	P	2072	MSG2	D	2030	MSG5IZ	B	0020
START	P	207C				PG1ME	P	2088

MODULE NAME:	PG2							
EXBENT	A	F564	MSG3	D	203E	MSG4	D	2048
STACK	B	0039				PGM2	P	20C3

MODULE NAME:	PG3						
ATEST	A	4406	POWERS	P	20DB		

(8)?EXIT ----- RETURN TO MDOS
 (9)=MESSAGE ----- LOAD AND EXECUTE NEW MDOS COMMAND

MESSAGE 1
 MESSAGE 1
 MESSAGE 2
 MESSAGE 2
 MESSAGE 3
 MESSAGE 3
 MESSAGE 4
 COMMON TEST PROGRAM

EXBUG 2.1
 ♦E

FIGURE 3-6. Using a Library File

3.5 LOADER OPERATIONS USING A CHAIN FILE

For programs requiring more than a few modules, the use of the MDOS CHAIN command to link them becomes a virtual necessity. It also provides a self-documenting listing of how to link the program. A sample chain file is shown in Figure 3-7. The use of this chain file is shown in Figure 3-8, and a description of the various steps is explained below.

1. The chain file (LINK.CF) is invoked using the MDOS CHAIN command. There are five option parameters which will be passed on to the chain file. This is the only line entered by the operator until (7).
2. The chain file pauses here to give the operator a chance to abort, if so desired, without destroying anything.
3. The previous map and object file are deleted.
4. The Linking Loader is invoked via the RLOAD command. The parameters from the command line (1) are substituted to define the section values.
5. Map output is directed to an output file called PG321.M0. This provides a permanent listing of the map output which can be listed at any time.
6. The MDOS LIST command is invoked to produce a hard copy of the map file on the line printer. Note the header option is used and the DATE command line parameter is substituted. The line printer listing of the map output files is shown in Figure 3-9.
7. The chain file processing ends and the input stream returns to the keyboard for operator input.

PAGE 001 LINK .CF:0

```
/*
/* ***** LINK MESSAGE PROGRAMS CHAIN PROCESSOR ****
/* ** 08/10/79 ** (C) 1979 BY THE AIR FORCE SYSTEMS COMMAND
/*
/* ***** ***** ***** ***** ***** ***** ***** ***** ***** */
/*
/* WARNING! GOING TO DELETE THE FOLLOWING FILES:
/*
----- PG321.LO:0 (OLD OBJECT)
/*
----- PG321.MO:0 (OLD RLOAD MAP)
/*
ABORT WITH 'BREAK' KEY OR RETURN
/*
STRIKE 'RETURN' TO CONTINUE...
/*
SET,M 8
DEL PG321.LO,PG321.MO
SET,M 0
RLOAD
IDON
STRD=$%D%;STRP=$%P%;STRB=$%B%
/IFS CP
CURP=\$%CP%
/XIF
LOAD=PG3,PG2,PG1
MAPU
OBJA=PG321
STRD=$%D%;STRP=$%P%;STRB=$%B%
/IFS CP
CURP=\$%CP%
/XIF
LOAD=PG3,PG2,PG1
MAPU
MO=PG321.MO
MAPF
EXIT
/*
LIST PG321.MO;LH
MESSAGE PROGRAM TEST RLOAD MAP - %DATE%
/*
/IFC B,D,P,DATE
/*
/* COCKPIT ERROR DETECTED!
/*
/* MUST SPECIFY THE FOLLOWING OPTIONS:
/*
-----
/* B = START BASE SEGMENT ADDRESS (HEX, NO $)
/* D = " DATA " " (HEX, NO $)
/* P = " PROGRAM " " (HEX, NO $)
/* DATE = TODAY'S DATE FOR MAP LISTING
/*
/*
OPTIONAL
CP = HEX VALUE (NO $) FOR "CURP=\\" COMMAND
/*
/* *** CHAIN ABORTED ***
/*
/ABORT
/XIF
```

FIGURE 3-7. Listing of Chain File Invoking RLOAD

(1) =CHAIN LINK;DATE=10 AUG. 1979%,B%0%,D%400%,P%1000%,CP%100%

```
*****  
♦♦ LINK MESSAGE PROGRAMS CHAIN PROCESSOR ♦♦  
♦♦ 08/10/79 ♦♦  
*****
```

♦♦
♦♦ WARNING! GOING TO DELETE THE FOLLOWING FILES:
♦♦ -----
♦♦ PG321.LO:0 (OLD OBJECT)
♦♦ PG321.MO:0 (OLD LOAD MAP)
♦♦
♦♦ ABORT WITH 'BREAK' KEY OR
(2) ♦♦ STRIKE 'RETURN' TO CONTINUE...

♦♦
?SET F0FF 0800
(3) DEL PG321.LO,PG321.MO
PG321 .LO:0 DELETED
PG321 .MO:0 DELETED
?SET F0FF 0000
(4) RLOAD
M003 LINKING LOADER REV 03.00
COPYRIGHT BY MOTOROLA 1977
?IDON
?STRD=\$400;STRP=\$1000;STRB=\$0
?CURP=~\$100
?LOAD=PG3,PG2,PG1
PG3 08/10/79 ASCT ILLUSTRATION - MODULE #3
PG2 08/10/79 MESS PRNTR SUBPROG - MODULE #2
PG1 08/10/79 MAIN MESS PROGRAM - MODULE #1
?MAPU
NO UNDEFINED SYMBOLS
?OBJA=PG321
?STRD=\$400;STRP=\$1000;STRB=\$0
?CURP=~\$100
?LOAD=PG3,PG2,PG1
PG3 08/10/79 ASCT ILLUSTRATION - MODULE #3
PG2 08/10/79 MESS PRNTR SUBPROG - MODULE #2
PG1 08/10/79 MAIN MESS PROGRAM - MODULE #1
?MAPU
NO UNDEFINED SYMBOLS
(5) ?MO=PG321.MO
?MAFF
?EXIT
♦♦
(6) LIST PG321.MO;LH
ENTER HEADING: MESSAGE PROGRAM TEST LOAD MAP - 10 AUG. 1979
♦♦
END CHAIN
(7) =LOAD PG321;V ----- LOAD OBJECT PROGRAM
(8) *E ;P ----- START PROGRAM EXECUTION
MESSAGE 1
MESSAGE 1
MESSAGE 2
MESSAGE 2
MESSAGE 3
MESSAGE 3
MESSAGE 4
COMMON TEST PROGRAM

EXBUG 2.1
*E

FIGURE 3-8. Using a Chain file and RLOAD

PAGE 001 PG321 .MO:0 MESSAGE PROGRAM TEST RLOAD MAP - 10 AUG. 1979

NO UNDEFINED SYMBOLS

MEMORY MAP

S	SIZE	STR	END	COMM
A	0006	4510	4515	
A	0006	4406	440B	
B	001A	0000	0019	0000
C	0030	0020	004F	0030
D	0042	0400	0441	0020
P	0251	1000	1250	0000

MODULE NAME	BSCT	DSCT	PSCT
PG3	0000	0400	1000
PG2	0000	0400	1100
PG1	0015	0414	1200

COMMON SECTIONS

NAME	S	SIZE	STR
DCOMM	D	0008	0422
DCOMM2	D	0018	042A

DEFINED SYMBOLS

MODULE NAME: PG3
ATEST A 4406 POWERS P 1000

MODULE NAME: PG2
EXBENT A F564 MSG3 D 0400 MSG4 D 040A PGM2 P 1100
STACK B 0014

MODULE NAME: PG1
CR A 000D EOT A 0004 EXBPRT A F024 LF A 000A
MSG1 P 1200 MSG2 D 0414 MSGSIZ B 0015 PG1NE P 1216
START P 120A

FIGURE 3-9. Map Output File Listing



A SUMMARY OF LINKING LOADER COMMANDS

<u>COMMAND</u>	<u>FUNCTION</u>
<u>CONTROL COMMANDS</u>	
BASE[=<number>]	LOAD CSCT, DSCT, and PSCT above defined address (default=MDOS compatible)
EXIT {<name1>} {<number>}	Give control to the disk operating system
IDOF	Suppress identification printing
IDON	Print module identification information
IF=<f-name>	Specify the intermediate file
IFOF	Intermediate file mode off
IFON	Intermediate file mode on
INIT	Initialize the Loader
OBJ[A][X]=<f-name>	Initiates Pass 2
MO={<device>} {<f-name>}	MAP output

LOAD DIRECTIVES

LIB=<f-name> [, [<f-name>]] ⁹⁹ 0	Enter file mode
LOAD=<f-name> [, [<f-name>]] ⁹⁹ 0	Load the indicated file(s)/module(s)

<u>COMMAND</u>	<u>FUNCTION</u>
<u>STATE COMMANDS</u>	
CUR {B D}=[\]<number>	Set current location counter
DEF: <name1>={<number> <name2>} [ASCT BSCT ,DSCT PSCT]	Define a symbol
END {B C D}=<number>	Set section ending address
MAPC	List user assigned section sizes and addresses
MAPF	List full load map
MAPS	List loader assigned section sizes and addresses
MAPU	List undefined symbols
STR {B C D}=<number>	Set section starting address

APPENDIX B
LINKING LOADER ERROR MESSAGES

Errors detected by the Linking Loader, while processing a command or loading a module, will result in an error message being printed at the user terminal. These errors are divided into two classifications: fatal errors and non-fatal (warning) errors. When the Loader detects a non-recoverable error, a fatal error message will be printed. Any commands not processed on the last command line will be ignored and a new prompt printed. If the Loader can recover from an error, only a warning message will be printed.

**FATAL
ERROR MESSAGES**

MESSAGE	DESCRIPTION
BAE	BSCT Assignment Error - the combined size of BSCT is greater than the amount that can be allocated in the defined BSCT area.
COV	Common Overflow - the size of a section's common is greater than 65,535.
GAE	General Assignment Error - the Loader cannot assign absolute memory addresses. This may result from: <ul style="list-style-type: none">address conflicts associated with ASCT'suser assignment of section addressesthe combined length of all sections exceeding 65,535the order in which the Loader assigns memory.
ICM	Illegal Command
IOR	Illegal Object Record - the input module is not a valid relocatable object module.
ISA	Illegal Stream Assignment - this error occurs when an invalid I/O device is assigned to a Loader I/O stream.
ISY	Illegal Syntax - error in the option or specification field of a command. This error may also occur when a command is not terminated by a semicolon, space, or carriage return.
LOV	Local Symbol Table Overflow - not enough memory for all the local (external) symbols defined by the current object module. Check for contiguous memory from location \emptyset .
GOV	Global Symbol Table Overflow - not enough memory for all the global (external) symbols defined by the object modules. Check for contiguous memory from location \emptyset .
PHS	Phase Error - the absolute address assigned to a global symbol at the end of Pass 1 does not agree with the address computed during Pass 2.
SOV	Section Overflow - the size of a section is greater than 65,535.

FATAL ERROR MESSAGES

MESSAGE

UAE User Assignment Error - the user has incorrectly defined load addresses. Use the MAPC command to produce a map for determining the cause of this error. The UAE error occurs when:

- the user defined end address is less than the user defined start address
- the space allocated by the user defined start and end addresses is less than that required for the section.
- the user has defined load section addresses which overlap
- the user defined execution address is out of range
- the user has defined ASCT below \$20
- the user has initialized locations in BSCT which are assigned below \$20

UIF Undefined IF File

UOI Undefined Object Input File

WARNING MESSAGES

IAM - <address> - Illegal Address Mode - a global symbol is referenced as a one-byte operand, and the most significant byte of the global symbol address is non-zero. One byte relocation is performed, using only the least significant byte of the global symbol address. The warning message indicates the absolute address of such a reference.

MDS - <symbol> - Multiply Defined Symbol - the Loader has encountered another definition for the previously defined global symbol. Only the first definition will be valid. This can also be caused by section conflicts for the symbol -- i.e., defined via an EQU directive (ASCT) and referenced in another module as BSCT.

UDS - <symbol> - Undefined Symbol - the symbol was not defined during Pass 1. A load address of zero will be assumed.