LINKING LOADER
User's Guide

Document Number 7278l a

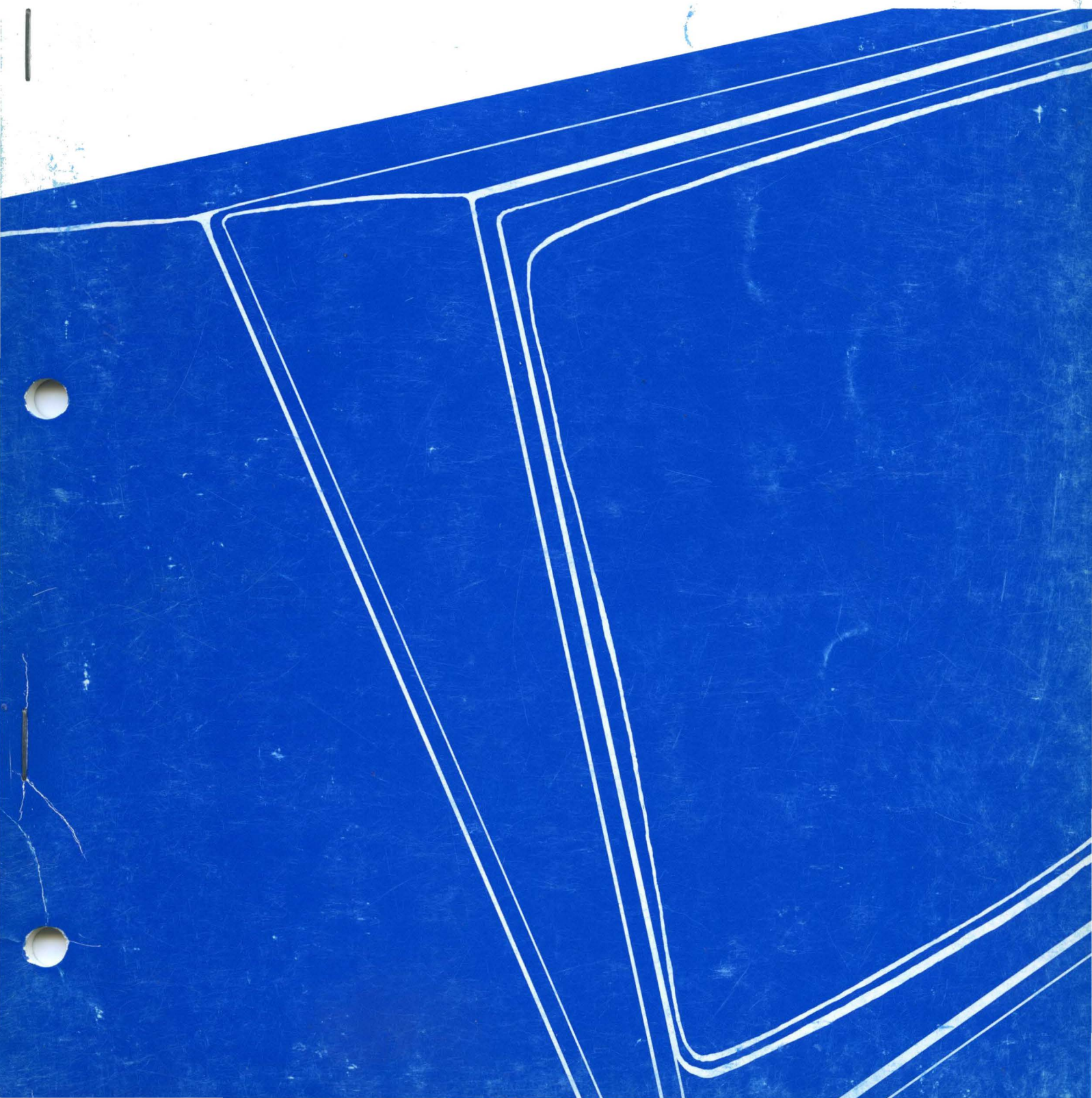

Table of Contents Page

List of Figures Page

CHAPTER 1 - INTRODUCTION

The Linking Loader combines relocatable object modules produced by the Macroassembler to produce a complete executable program.

Since the Linking Loader permits load origins and inter-module references to be resolved at linkage time rather than at assembly time, modules are not required to specify absolute addresses (only relative addresses are required). As the modules are loaded by the Linking Loader, labels are automatically assigned absolute addresses, satisfying the references to such labels located in other modules. This process greatly speeds program development and permits time to be spent on correcting program design rather than on defining label addresses.

The Linking Loader also allows modules to be compiled separately, enabling the user to build a library of standard or commonly-used program modules for inclusion in other programs.

1.1 HARDWARE SUPPORT REQUIRED

The minimum hardware configuration required to support Linking Loader consists of:

- CDX-68 Basic Display Terminal with the appropriate firmware options
- 32K bytes of user memory (RAM)
- 10 Mb Disk or 1 Mb Diskette Storage (CDX-DS/FR or CDX-FS Series)

1.2 OPTIONAL HARDWARE SUPPORTED

Linking Loader also supports a variety of printers, including Matrix and Character printers (the Codex SP Series). These optional printers are linked to the Basic Display Terminal through either the Microcomputer Module D or the Printer Interface Module (CDX-PI).

1.3 SOFTWARE SUPPORT REQUIRED

No additional software is required to run the Linking Loader as it comes shipped on the system disk.

1.4 SOFTWARE INSTALLATION

There is no software installation that need be performed. All Linking Loader software is on the disk containing the CODOS system software.

CHAPTER 2 - LINKING LOADER FEATURES

2.1 RELOCATION

Relocation allows the user to assemble/compile a source program without assigning absolute addresses at the time of assembly or compilation. Absolute memory assignment is performed at load time. In order to relocate a program (within memory), the source program must be assembled with the Macroassembler, using the OPT REL directive. The Macroassembler produces 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 Linking Loader must be used. The Linking Loader assigns addresses and produces an absolute object module compatible with the system loader.

2.2 LINKING

Linking allows instructions in one program to refer to instructions or data residing within other programs. If all programs are assigned absolute addresses during assembly time, it is possible to directly reference another program by absolute addresses. However, when using relocatable programs, absolute load addresses are not generally known until load time. In order to obtain other relocatable programs or data blocks, external reference symbols are 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 Macroassembler 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 referenced label is defined outside the module.

At load time, global references match with their corresponding global definitions. Any reference within a module to a global symbol updates with the load address of the global symbol. If the loader detects a global reference without an associated global definition, an undefined global error occurs, and a load address of zero is assigned to the reference.

2.3 SECTIONS

The section concept is preserved by the Linking Loader during the load process. As a module loads, each of its sections combines with the corresponding sections of previously loaded modules. As a result, the absolute load module produced by the Linking Loader, contains one continuous memory area for each section type encountered. The following is a brief definition of each section type.

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 (i.e., 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 are referenced through direct addressing. BSCT is limited to locations within the addressing range of 0 through 255 (hexadecimal locations 0 through 00FF).

CSCT - Blank Common (uninitialized)

There is only one CSCT. This section is used for 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) obtained through extended mode addressing (hexadecimal locations 100-FFFF).

PSCT - Program Section

PSCT is similar to DSCT except that it is normally used for instructions.

2.4 NAMED COMMON

In addition to the program segmentation provided by the section concept, the relocation and linking scheme supports named common. The named common concept designates common areas within BSCT, DSCT, or PSCT. In processing named common definitions, the Linking Loader (1) assigns to each named common area a size equal to the largest size defined for the named common during the load process, and (2) allocates memory at the end of each section for the named common blocks defined within that section.

The load maps shown in Figure 2-1 describe the load process with regard to sections and named common. The module EX1 requires reserved memory in BSCT, CSCT, DSCT, and PSCT. The only space necessary in DSCT is for the named common NCOM1. The module EX2 requires that memory is allocated in BSCT, CSCT, DSCT, and PSCT. Neither module defines ASCT blocks.

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 Linking Loader for use by the disk operating system, unless otherwise directed. After BSCT, space for blank common is allocated, followed by space for EX2 DSCT. Since EX1 requires no DSCT for its exclusive use, none is 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 Linking 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 defined order. Figure 2-2 illustrates a load module map produced by loading EX2 followed by EX1. This load module map is slightly different from the map in Figure 2-1 where EX1 was loaded first.

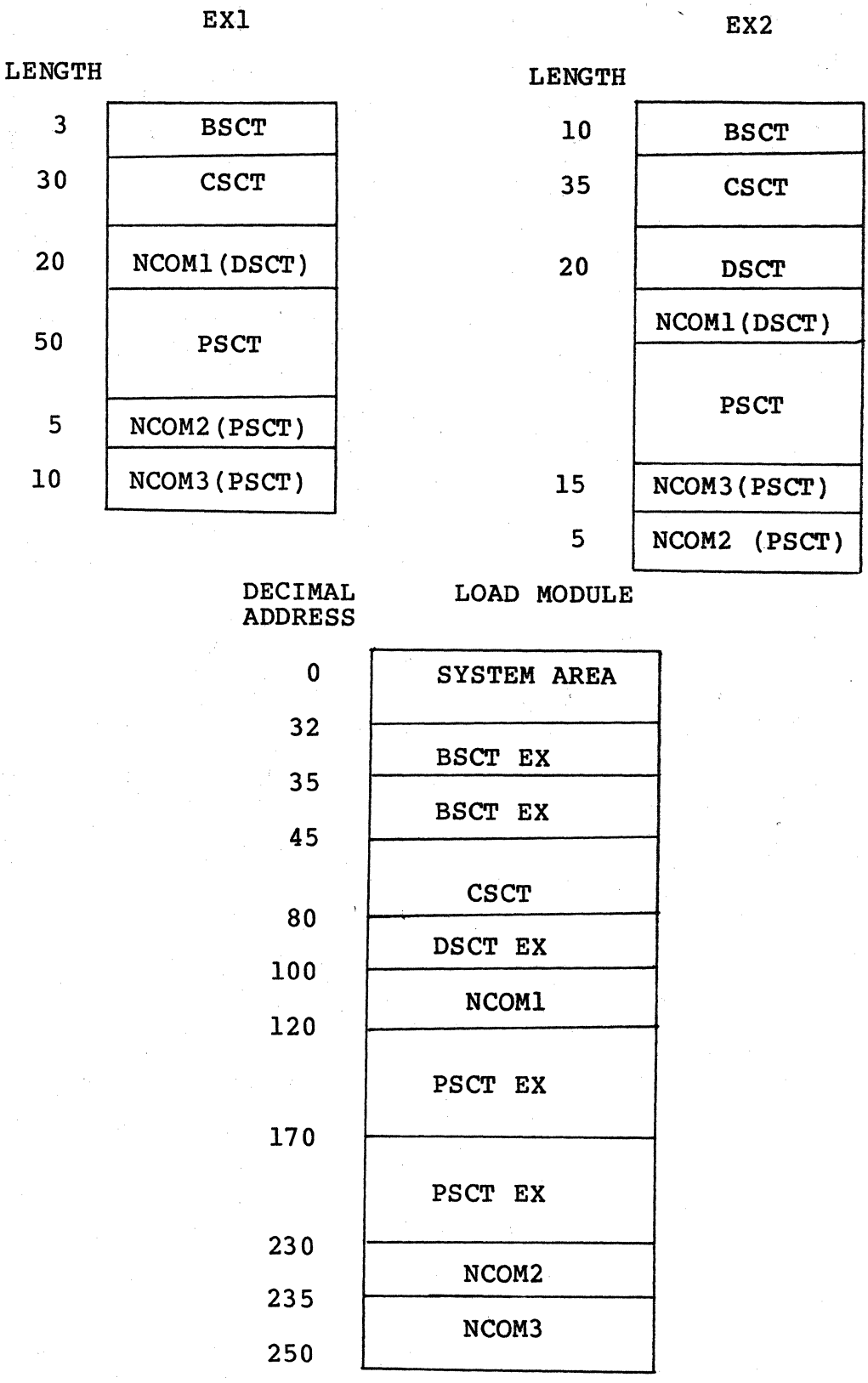

Figure 2-1. Load Maps

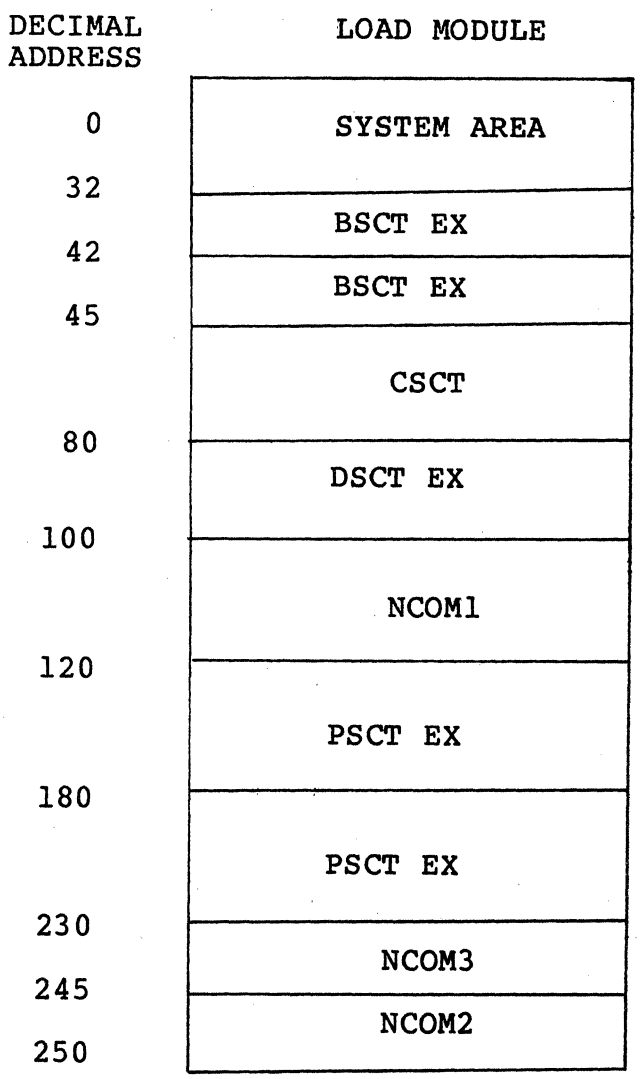

Figure 2-2. Load Map

2.5 MODULE LIBRARIES

The Linking Loader can automatically search a file for modules containing definitions satisfying any unresolved global symbols. Such a file is called a library file. It is composed of one or more merged object modules. The Linking Loader sequentially searches the library file. If a module contains a symbol definition satisfying an unresolved global symbol, that module is loaded. Only those modules which satisfy an unresolved reference are 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 Linking Loader to search the library again.

2.6 MEMORY ASSIGNMENT

During the load process, absolute addresses are assigned to the program sections within the specified modules. Normally, the Linking Loader automatically performs 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 Linking Loader first reserves memory for those sections with defined load addresses before allocating space for any other section.

The Linking 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 which the module is specified. Named common blocks are always assigned memory at the end of the associated load section.

2.7 LOAD MAPS

The Linking Loader optionally produces a load map describing the memory layout results. Figure 2-3 is an example of some of the features included in a typical load map. In addition to this full load map, the Linking Loader may be directed to produce partial load maps that list only the undefined global symbols or section load addresses.

```
  NO UNDEFINED SYMBOLS

MEMORY MAP

  S SIZE  STR  END COMN
  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   MSGSIZ B 0000  PG1NE 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 2-3. Loader-Produced Memory Map

CHAPTER 3 - LINKING LOADER COMMANDS

The Linking Loader must be called while under the control of CODOS. When the user types the command:

```
=RLOAD
```

the disk executive loads the Linking Loader. Upon entry, the loader prints:

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

The character "?" is the Linking Loader prompt and prints whenever the Linking Loader completes the last command and is ready for another.

3.1 LINKING LOADER INPUT

The input to the Linking Loader is in one of two forms (1) commands or (2) object modules. The Linking Loader commands control the relocation and linkage of object modules. Object modules are produced by the Macroassembler. 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 Linking Loader. In addition, a disk file may be used as a library file. The Linking Loader may also run under the CODOS CHAIN command.

Nomenclature

<f-name> - Used to indicate the name of a disk file to be used by the Linking Loader. Unless specified, the file is assumed to have a suffix of RO and a drive number of 0. For the format of the file name, consult the CODOS manual. (Example: PG1.RO:1)

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

```
0 - 65,535 (decimal)
$0 - $FFFF (hexadecimal)
```

[ ] - Used to indicate that the enclosed directive(s) is optional.

$\overset{99}{[\ ]}_{0}$ - Used to indicate that the enclosed directive may be repeated from 0 to 99 times, up to a maximum total of 79 characters.

{ } - Indicates that one of the enclosed options must be used.

3.2 COMMAND FORMAT

Each Linking 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 be defined as:

$$\left[\text{<command>}[;\text{<command>}]\overset{99}{\underset{0}{\ }}\right]\left[\text{<space>}[\text{<comments>}]\right]\text{<c/r>}$$

```
Example:  STRB=0;STRD=$1000;STRP=$4000
          IDON
          LOAD=PG1
```

The commands in a command line execute only after the Linking Loader detects a carriage return.

If a command line is incorrectly entered, 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 Linking Loader to ignore the current command line, 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 "SHIFT DEL" (the SHIFT and DEL keys typed simultaneously). After deleting the character(s) in error, the corrected version of the command line may be entered. The "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 CODOS ".KEYIN" special character functions.

The Linking Loader executes 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

is terminated, and the remaining commands in the command line are ignored.

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

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

3.3 LINKING LOADER COMMANDS

The Linking Loader commands are divided into three classes:

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

The control commands initiate Passes 1 and 2 of the Linking Loader, as well the return to the disk operating system. The load directives identify the modules to be loaded; and the state directives direct the assignment of memory to the various program sections and produce a load map.

3.4 CONTROL COMMANDS

3.4.1 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 with the NAM and IDNT directives.

3.4.2 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 print at the console device. For assembly language programs, this information is specified via the NAM and IDNT directives.

3.4.3 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 Linking Loader commands and object modules. It directs 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 Linking 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 disk. Upon proper exiting from the Linking Loader, the IF file is deleted.

Example: IF=IFILE

Defines IFILE on drive 0 as the intermediate file. Default suffix is IF.

3.4.4 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.

3.4.5 IFON - INTERMEDIATE FILE MODE ON

Format: IFON

Description: This command directs the Linking 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.

3.4.6 INIT - INITIALIZE LOADER

Format: INIT

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

3.4.7 MO - MAP OUTPUT

Format: MO= [<f-name> / <device>]

Description: The MO command specifies the media on which the map output is to be produced. The MAP output output defaults to the console.

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 used.

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

Example:

MO=MAPFL	All output generated by the MAP command is written on file MAPFL on drive 0.
MO=#LP	The line printer is used for all future map output.

3.4.8 OBJ - PRODUCES LOAD MODULE

Format: OBJA=<file-name>

Description: This command is used with the Linking Loader to initiate the second pass of the Linking 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 is created on disk 0 unless disk 1 is specified in <file-name>. The type of object file produced by the Linking Loader is determined by the command form as follows:

OBJA - This format creates an absolute memory image file suitable for loading via the CODOS LOAD command. A default file suffix of 'LO' and drive 0 is used if none are specified.

If an intermediate file (IF) is generated during the first pass of the Linking Loader, the second pass automatically processes the commands entered during the first pass. In the event that an intermediate file is not created, the same sequence of commands used during the first pass must be repeated.

Examples:

```
OBJA=REPORT:1
LOAD=REPORT;OBJA=REPORT;LOAD=REPORT
```

The Linking Loader creates the absolute object file on file, 'REPORT.LO' on drive 1.

3.4.9 EXIT

Format: EXIT [= { <number> / <name1> }]

Description: The "EXIT" command causes control to return to the disk operating system after all Linking Loader files are closed.

The CODOS version of the Linking Loader allows the user to define the starting execution address of the object program. If the <number> option is specified, the given absolute number is 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 <name> 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 is used as the starting address.

3.5 LOAD DIRECTIVES

3.5.1 LIB - LIBRARY SEARCH

Format: LIB=<f-name> $\left[,[<f\text{-}name>]\right]_{0}^{99}$

Description: The LIB command instructs the Linking 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 is 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 merged into a single file.

Modules loaded via the LIB command may also reference undefined global symbols. 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 made.

It should be noted that the Macroassembler produces 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.RO on drive 1 are searched to resolve any unsatisfied global references.

3.5.2 LOAD - LOAD A FILE

Format: LOAD=<f-name> [,[<f-name>]]$_{0}^{99}$

Description: The LOAD command directs the Linking Loader to load the specified object files.

The LOAD command directs the Linking Loader to load all object modules found in the specified file name(s). The file name could be a library file, but the LOAD command unlike the LIB command, loads each object module found.

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.

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.

3.6 STATE COMMANDS

3.6.1 BASE - INITIALIZE MINIMUM LOAD ADDRESS

Format: BASE [=<number>]

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

Example: BASE

Unassigned CSCT, DSCT, and PSCT are assigned load addresses above CODOS.

3.6.2 STR - STARTING ADDRESS

Format: STR{B,C,D,P}={<number>, <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 is assigned a starting address by the loader.

NOTE: A starting address of $FFFF resets any previous STR directive for the corresponding section. This allows the Linking Loader to define the starting address.

Example: STRP=$1000

PSCT is allocated memory starting at $1000.

3.6.3 CUR - SET CURRENT LOCATION COUNTER

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

Description: The CUR command modifies the Linking 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 in order to update the loading address. If the "\" option is not specified, the relative load address for the appropriate section is 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 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 adds 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 $8000 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 $8000
                                        ($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 2 relative to the start of the section (2,4,8, etc.). The specified <number> defines the given power of 2. This option remains in effect until the option is specified again or until the current pass of the Linking Loader is complete. If the "\" option is in effect when memory is assigned to the starting section addresses, the starting address of the section is assigned a load address which is a power of 2. This option does not apply to named common blocks within the specified section.

If the CUR directive is not used, each module normally loads at the next load address in the appropriate section (contiguously loaded modules).

Example: CURP=$100

Sets the relative PSCT location counter to $100 plus STRP value.

CURP=\16

Causes the Linking 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 a CODOS chain file, the "\" option must use "\\" instead of "\."

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 is assigned as follows:

```
PG1=$4001
PG2=$4401
PG3=$4801
```

3.6.4 DEF - LOADER SYMBOL DEFINITION

Format: DEF: <name1>= {<number> / <name2>} ,ASCT / ,BSCT / ,DSCT / ,PSCT

Description: The DEF command defines a global symbol and enters it in the global symbol table. The symbol to be defined is given by "name1" and must be a valid Macroassembler variable name. The symbol may not currently be defined. If the <number> option is used, the symbol is 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, and PSCT, 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).

3.6.5 END - ENDING ADDRESS

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

Description: The END commands set the absolute ending address of 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" resets any previous END directive for the corresponding section.

Example: ENDB=255

BSCT is allocated such that the last address reserved is 255 (decimal).

3.6.6 MAP - PRINTS LOAD MAPS

Format: MAP{C F S U}

Description: The MAP commands display the current state of the modules loaded or the Linking 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 Linking 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 complete. Use the MAPC command to determine the cause of the error.

MAPS - The Linking 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, prints. If a user assignment error (UAE) exists, this command cannot complete. Use the MAPC command to determine the cause of the error.

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

CHAPTER 4 - SAMPLE OPERATIONS WITH THE LINKING LOADER

This chapter provides a description of the Linking Loader operations in typical applications. To demonstrate the use of the Linking Loader, a simple message printing program is used, consisting of three modules which reference instruction sequences or data within each other. An assembly listing of each module is shown in Figures 4-1, 4-2, and 4-3.

4.1 SIMPLIFIED LINKING LOADER OPERATION

The simplest form of the Linking Loader's operation is shown in Figure 4-4. 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 4-4 is as follows:

1. The LOAD command loads the first file, PG1.RO: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 RO and drive number 0 are assumed.

3. The OBJA command starts pass 2 of the load function, which creates 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.

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.

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 has a size of 42 (hex) bytes, of which 20 (hex) bytes are in common. The DSCT area starts at address $6A and ends 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
```

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

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

```
        EXIT=STAR.

or      EXIT=$B6
```

```
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  06/06/80 MAIN MESG PROGRAM-MODULE #1

00006              * ASSEMBLY PROCEDURE:  CMAP X.XX CODOS X.XX
00007              *     =CMAP PG1;LN=76
00008              *
00009              *   PROGRAM PARTS:  PG1, PG2, PG3
00010              *        COMPUTER:  CDX-68


00012    F024    A EXBPRT EQU    $F024    COBUG PRINT ROUTINE


00014              * ASCII CHARACTER EQUATES
00015              *
00016    0004    A EOT   EQU    $04      END OF TEXT
00017    000A    A LF    EQU    $0A      LINE FEED
00018    000D    A CR    EQU    $0D      CARRIAGE RETURN


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


00026              * EXTERNAL DEFINITIONS
00027              *
00028                    XDEF   MSG2,MSG1,EXBPRT,START,PG1NE
00029                    XDEF   MSGSIZ,EOT,LF,CR
```

Figure 4-1. Message Program 1 (PG1)

```
00031                 * COMMON MESSAGE AREA
00032                 * (NAMED COMMON "DCOMM" IN DSCT)
00033                 *
00034N 0000           DCOMM COMM DSCT
00035N 0000 0000 P MSG1P FDB MSG1  PTR TO MESG 1(IN PSCT)
00036N 0002 0000 D MSG2P FDB MSG2  PTR TO MESG 2(IN DSCT)
00037N 0004 0000 A MSG3P FDB MSG3  PTR TO MESG 3
                                      (XREF IN DSCT)
00038N 0006 0000 A MSG4P FDB MSG4  PTR TO MESG 4
                                      (XREF IN DSCT)


00040                 * MESSAGES 1 AND 2
00041                 * (NEW NAMED COMMON "DCOMM2" IN DSCT)
00042                 *
00043N 0000           DCOMM2 COMM DSCT
00044N 0000 0001 A CMSGCT RMB  1    COMMON MESSAGE COUNT
00045N 0001 0014 A CMSG   RMB  20   COMMON MESSAGE


00047C 0000                 CSCT      BLANK COMMON SECTION
00048C 0000 0010 A MSGCST RMB  16   RESERVE 16 BYTES


00050D 0000                 DSCT      DATA SECTION
00051D 0000 4D   A MSG2   FCC  \MESSAGE 2\
00052D 0009 04   A        FCB  EOT  DELINEATE END OF MESSAGE


00054P 0000                 PSCT      PROGRAM SECTION
00055P 0000 4D   A MSG1   FCC  \MESSAGE 1\
00056P 0009 04   A        FCB  EOT


00058B 0000                 BSCT      BASE SECTION
00059B 0000 0001 A MSGSIZ RMB  1    MESG SIZE STORAGE
```

Figure 4-1. 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 BE 0000 A START LDS #STACK SET UP STACK REGISTER
                                                      (XREF)
00067P 000D FE 0000 N       LDX MSG1P  GET MESSAGE 1 POINTER
00068P 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 PG1NE 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       LDAB 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 08              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       STAA 0,X     SAVE BYTE
00094P 0047 08              INX  UPDATE DESTINATION POINTER
00095P 0048 FF 0003 B       STX  TOPNTR
00096P 004B 5A              DECB   UPDATE CHARACTER COUNTER
00097P 004C 26 EB 0039      BNE  LOOP1   LOOP
00098P 004E 7E 0000 A JMP ATEST GOTO PROGRAM W/ASCT REGIONS
```

Figure 4-1. Message Program 1
(PG1 - cont'd)

```
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 LDAA 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 4-1. Message Program 1
(PG1 - cont'd)

```
PAGE  004   PG1      .SA:1   PG1    CROSS REFERENCE TABLE

R        ATEST    00022*00098
ND 0001  CMSG     00045*00084
ND 0000  CMSGCT   00044*00086
D  000D  CR       00018*00029
ND       DCOMM    00034*
ND       DCOMM2   00043*
D  0004  EOT      00016*00029 00052 00056
R        EXBENT   00024*
D  F024  EXBPRT   00012*00028 00068 00074 00076 00078
 B 0001  FROMPT   00085 00088 00091 00105*00109
D  000A  LF       00017*00029
 P 0039  LOOP1    00088*00097
DP 0000  MSG1     00028 00035 00055*
ND 0000  MSG1P    00035*00067
DD 0000  MSG2     00028 00036 00051*
ND 0002  MSG2P    00036*
RD       MSG3     00023*00037 00073
ND 0004  MSG3P    00037*00075
RD       MSG4     00023*00038 00077
ND 0006  MSG4P    00038*
 C 0000  MSGCST   00048*00082
DB 0000  MSGSIZ   00029 00059*00087
DP 0016  PG1NE    00028 00073*
R        PGM2     00024*00069
R        STACK    00023*00066
DP 000A  START    00028 00066*00113
 B 0003  TOPNTR   00083 00092 00095 00106*00110
```

Figure 4-1. 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 08/10/80 MESG PRNTR SUBPROG-MODULE #2

00006               * ASSEMBLY PROCEDURE: CMAP X.XX CODOS X.XX
00007               *    =CMAP PG2;LN=76
00008                  *
00009                  *   PROGRAM PARTS:  PG1, PG2, PG3
00010                  *        COMPUTER:  CDX-68

00012          F564 A EXBENT EQU  $F564  COBUG ENTRY POINT


00014                  *
00015                  * XDEFS AND XREFS
00016                  *
00017                        XDEF MSG3,MSG4,STACK,EXBENT,PGM2
00018                        XREF BSCT:MSGSIZ
00019                        XREF EXBPRT,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   00   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 4-2. Message Program 2 (PG2)

```
PAGE   002   PG2     .SA:1  PG2   MESSAGE PRINTER SUBPROGRAM

00044                 * START OF PROGRAM 2
00045                 *
00046P 0000                   PSCT
00047P 0000 4F        PGM2    CLRA
00048P 0001 97 00     A       STAA MSGSIZ  INIT. MESG LENGTH
00049P 0003 FE 0000 N         LDX  MSG1PT  PRINT MESSAGE 1
00050P 0006 BD 0000 A         JSR  EXBPRT
00051P 0009 CE 0000 A         LDX  #MSG2   PRINT MESSAGE 2
00052P 000C BD 0000 A         JSR  EXBPRT
00053P 000F FE 0002 N         LDX MSG2PT PRINT MESSAGE 2 AGAIN
00054P 0012 BD 0000 A         JSR  EXBPRT
00055P 0015 7E 0000 A         JMP  PG1NE RETURN TO PROGRAM ONE



00057B 0000                   BSCT   DIRECT ADDRESSING SECTION
00058B 0000    0014 A         RMB  20
00059B 0014    0001 A STACK   RMB  1  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*
DB 0014 STACK    00017 00059*
```

Figure 4-2. 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 08/10/80 ASCT ILLUSTRATION-MODULE #3

00006              * ASSEMBLY PROCEDURE:  CMAP X.XX  CODOS X.XX
00007              *    =CMAP PG3:1;LN=76
00008              *
00009              *    PROGRAM PARTS:  PG1, PG2, PG3
00010              *         COMPUTER:  CDX-68

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 COBUG/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    03EB 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       EXBENT 00013*00026
DP 0000 POWERS 00012 00030*
```

Figure 4-3. Message Program 3 (PG3)

```
   =RLOAD
   CODOS LINKING LOADER REV X.XX
   COPYRIGHT BY CODEX 1980
(1)?LOAD=PG1.RO: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 COMN
     A 0006 4510 4515
     A 0006 4406 440B
     B 001A 0020 0039 0000      (6b)
     C 0030 003A 0069 0030
     D 0042 006A 00AB 0020
     P 0073 00AC 011E 0000
   MODULE NAME BSCT DSCT PSCT
     PG1        0020 006A 00AC
     PG2        0025 0078 00FD  (6c)
     PG3        003A 008C 0115
   COMMON SECTIONS
      NAME  S SIZE  STR
     DCOMM  D 0008 008C         (6d)
     DCOMM2 D 0018 0094
   DEFINED SYMBOLS
MODULE NAME:  PG1
CR    A 000D EOT  A 0004  EXBPRT A F024 LF    A 000A
MSG1  P 00AC MSG2 D 006A  MSGSIZ B 0020 PG1NE P 00C2 (6e)
START P 00B6
MODULE NAME:  PG2
EXBENT A F564  MSG3 D 0078  MSG4 D 0082  PGM2 P 00FD (6f)
STACK  B 0039
MODULE NAME:  PG3
ATEST  A 4406  POWERS P 0015   (6g)
(7)?EXIT ---------------------- RETURN TO CODOS
=
```

Figure 4-4. Basic Loader Operation

4.2 LOADER OPERATIONS USING INTERMEDIATE FILES

As shown in the previous example, most commands must be re-entered during pass 2 of the Linking Loader. The use of an intermediate file eliminates the need to retype Linking Loader commands. Figure 4-5 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. In Figure 4-5, the first PSCT for module PG2 starts at $1070.

5. Notice that the loading order is different from the example in Figure 4-4. As each file/module loads, its identifier prints (5a).

6. As in the previous example, the OBJA command initiates pass 2 of the Linking Loader. However, since the intermediate file feature is being used, pass 2 automatically performs without the user re-entering the commands. Notice the identifiers also print as each file/module loads.

7. The Linking 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.

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 2-3.

9. The object file is closed and control is returned to CODOS via the EXIT command.

```
   =ROLOAD
   CODOS LINKING LOADER REV X.XX
   COPYRIGHT BY CODEX 1980
(1)?IF=TEMP-----------------CREATE INTERMEDIATE FILE = TEMP
(2)?IDON--------------------TURN ON IDENTIFIERS
(3)?STRD=$400;STRP=$1000;STRB=0-DEFINE STARTING SECTION
                                                    ADDRESSES
(4)?CURP=\$10-------START PSCT ON MODULO 10 (HEX)BOUNDARIES
(5)?LOAD=PG1,PG3,PG2------------LOAD FILES
     PG1     08/10/80 MAIN MESG PROGRAM  - MODULE 1
(5a) PG3     08/10/80 ASCT ILLUSTRATION  - MODULE 3
     PG2     08/10/80 MESG PRNTR SUBPROG - MODULE 2
(6)?OBJA=PG132-START PASS 2-CONTROLLED BY INTERMEDIATE FILE
     PG1     08/10/80 MAIN MESG PROGRAM  - MODULE 1
     PG3     08/10/80 ASCT ILLUSTRATION  - MODULE 3
     PG2     08/10/80 MESG PRNTR SUBPROG - MODULE 2
(7)?MO#LP------------ASSIGN MAP OUTPUT TO LINE PRINTER
(8)?MAPF-------------FULL MEMORY/SYMBOL MAP TO LINE PRINTER
(9)?EXIT-------------RETURN TO CODOS
```

Figure 4-5. Using an Intermediate File

4.3 LOADER OPERATIONS USING A LIBRARY FILE

The previous examples described the loading procedure performed with the LOAD command. In these examples, the user was aware of each module to be loaded. 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 containing a collection of utility modules. The Linking Loader may be used to extract only the required modules from this library file. The use of a library file is shown in Figure 4-6, and a description of the various steps is explained below:

1. The CODOS 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 Linking Loader to assign memory for CSCT, DSCT, and PSCT above the CODOS system area. As a result, the user program may be invoked directly as a CODOS command without using the LOAD command. However, if the program initializes BSCT, the CODOS 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 list 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 load via the LIB command. PG1, which is also in PGLIB, is not loaded again.

5. The second MAPU command shows that all external references have now been satisified.

6. The second pass of the Linking Loader initiates 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 a CODOS command (see item 9).

7. Since an intermediate file was not created during pass 1, all commands entered in pass 1 must repeat 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 Linking Loader and returns to CODOS.

9. The file created by the Linking Loader acts as a CODOS command and loads and executes automatically.

```
(1)=MERGE PG1.RO,PG2.RO,PG3.RO,PGLIB.RO--BUILD LIBRARY FILE
   =RLOAD
   CODOS LINKING LOADER REV X.XX
   COPYRIGHT BY CODEX 1980
(2)?BASE-----------------------LOCATE PROGRAM ABOVE CODOS
   ?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 COMN
     A 0006 4510 4515
     A 0006 4406 440B
     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        0020 2030 2072
     PG2        0025 203E 20C3
     PG3        0038 2052 20DB

   COMMON SECTIONS
      NAME   S SIZE  STR
     DCOMM   D 0008 2052
     DCOMM2  D 0018 205A

   DEFINED SYMBOLS
   MODULE NAME:  PG1
     CR      A 000D EOT    A 0004 EXBPRT A F024 LF     A 000A
     MSG1    P 2072 MSG2   D 2030 MSGSIZ B 0020 PG1NE  P 2088
     START   P 207C
   MODULE NAME:  PG2
     EXBENT A F564 MSG3   D 203E MSG4    D 2048 PGM2   P 20C3
     STACK  B 0039
   MODULE NAME:  PG3
     ATEST  A 4406 POWERS P 20DB
(8)?EXIT-----------------RETURN TO CODOS
(9)=MESSAGE--------------LOAD AND EXECUTE NEW CODOS COMMAND
```

Figure 4-6. Using a Library File

4.4 LOADER OPERATIONS USING A CHAIN FILE

For programs requiring more than a few modules, the use of the CODOS 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 4-7. The use of this chain file is shown in Figure 4-8, and a description of the various steps is explained below.

1. The chain file (LINK.CF) is invoked using the CODOS CHAIN command. There are five option parameters which are 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 without destroying anything.

3. The previous map and object file delete.

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.MO. This provides a permanent listing of the map output which can be listed at any time.

6. The CODOS 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 4-9.

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 **
/*    **                 08/10/80               **
/*    ******************************************
/*
@*
@*   WARNING!  GOING TO DELETE THE FOLLOWING FILES:
@*   -------       PG321.LO:0   (OLD OBJECT)
@*                 PG321.MO:0   (OLD RLOAD MAP)
@*
@*             ABORT WITH 'BREAK' KEY OR
@.             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
@*
```

Figure 4-7. Listing of Chain File Invoking RLOAD

```
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 4-7. Listing of Chain File Invoking RLOAD (cont'd)

```
(1)
=CHAIN LINK;DATE%10 AUG. 1980%,B%0%,D%400%,P%1000%,CP%100%
    ******************************************
    ** LINK MESSAGE PROGRAMS CHAIN PROCESSOR **
    **                 08/10/80               **
    ******************************************
   @*
   @*  WARNING!  GOING TO DELETE THE FOLLOWING FILES:
   @*  -------       PG321.LO:0   <OLD OBJECT>
   @*                PG321.MO:0   <OLD RLOAD 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
   CODOS LINKING LOADER REV X.XX
   COPYRIGHT BY CODEX 1980
   ?IDON
   ?STRD=$400;STRP=$1000;STRB=$0
   ?CURP=\$100
   ?LOAD=PG3,PG2,PG1
     PG3      08/10/80 ASCT ILLUSTRATION  - MODULE 3
     PG2      08/10/80 MESG PRNTR SUBPROG - MODULE 2
     PG1      08/10/80 MAIN MESG PROGRAM  - MODULE 1
   ?MAPU
     NO UNDEFINED SYMBOLS
   ?OBJA=PG321
   ?STRD=$400;STRP=$1000;STRB=$0
   ?CURP=\$100
   ?LOAD=PG3,PG2,PG1
     PG3      08/10/80 ASCT ILLUSTRATION  - MODULE 3
     PG2      08/10/80 MESG PRNTR SUBPROG - MODULE 2
     PG1      08/10/80 MAIN MESG PROGRAM  - MODULE 1
   ?MAPU
     NO UNDEFINED SYMBOLS
(5)?MO=PG321.MO
   ?MAPF
   ?EXIT
   @*
(6)LIST PG321.MO;LH
ENTER HEADING:MESSAGE PROGRAM TEST RLOAD MAP-10 AUG. 1980
   @*
   END CHAIN
(7)=LOAD PG321;V ----------------------LOAD OBJECT PROGRAM
```

Figure 4-8. Using a Chain file and RLOAD

```
PAGE 001 PG321  .MO:0 MESSAGE PROGRAM TEST RLOAD MAP -
                                                10 AUG. 1980

 NO UNDEFINED SYMBOLS

MEMORY MAP

  S SIZE  STR  END COMN
  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 4-9. Map Output File Listing

APPENDIX A - LINKING LOADER COMMANDS

Command	Function
Control Commands	
BASE[=<number>]	LOAD CSCT, DSCT, and PSCT above defined address (default=CODOS 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
OBJA=<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)

Command	Function
State Commands	
CUR{B D P}=[\]<number>	Set current location counter
DEF: <name1>={<number> <name2>}[,ASCT ,BSCT ,DSCT ,PSCT]	Define a symbol
END {B C D P}=<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 P}=<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, results in an error message printing at the user terminal. These errors are divided into two classifications: fatal errors and non-fatal (warning) errors. When the Linking Loader detects a non-recoverable error, a fatal error message prints. Any commands not processed on the last command line are ignored and a new prompt prints. If the Linking Loader can recover from an error, only a warning message prints.

Fatal Error Messages

Message

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 Linking Loader cannot assign absolute memory addresses. This may result from:

- address conflicts associated with ASCT's
- user assignment of section addresses
- the combined length of all sections exceeding 65,535
- the 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 Linking 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 0.

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 0.

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.

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 Linking Loader has encountered another definition for the previously defined global symbol. Only the first definition is 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 is assumed.

codex

A Subsidiary of MOTOROLA INC.

CODEX CORPORATION
20 Cabot Boulevard
Mansfield, Massachusetts 02048

CODEX PHOENIX
INTELLIGENT TERMINAL SYSTEMS
2002 West 10th Place
Tempe, Arizona 85281
(602) 994-6580

Member of
IDCMA

Printed in U.S.A.