60429800

CONTROL DATA

CYBER LOADER VERSION 1 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 2

NOS/BE 1

QUICK REFERENCE SUMMARY AND INDEX

CONTROL STATEMENTS

	Page
CAPSULE(pname,...)	8-1
EXECUTE(eptname,p,...)	2-4
GROUP(groupname)	8-1
LDSET(option,...)	2-7
COMMON	2-10
COMMON=lcbname/...	2-10
EPT=eptname/...	2-10
ERR=p (ALL,FATAL,NONE)	2-9
FILES=lfn/...	2-10
LIB=libname/...	2-8
MAP=p/lfn (N or combination of S, B, E, X)	2-8
NOEPT=eptname/...	2-10
NOREWIN	2-9
OMIT=eptname/...	2-10
PD=p	2-10
PRESET=p	2-8
PRESETA=p	2-8
PS=p	2-10
REWIND	2-9
STAT=lfn/...	2-10
SUBST=pair/...	2-10
USE=eptname/...	2-10
USEP=pname/...	2-9
LIBLOAD(libname,eptname,...)	2-3
LIBRARY(libname,...)	3-1
LIBRARY(libname,.../p)	3-1
LOAD(lfn,...)	2-3
MAP(p) (OFF, PART, ON, FULL)	3-1
NOGO(lfn,eptname,...)	2-5
REDUCE.	3-2
RFL(CM=n,EC=m)	3-3
SATISFY(libname,...)	2-6
SEGLOAD(I=lfn,B=lfn,LO=DT)	7-6
SLOAD(lfn,name,...)	2-4
TRAP(I=lfn,L=lfn)	5-1

USER CALL LOADER MACROS

LOADER paddr,CMM	4-1
LDREQ option,p,...	4-2
BEGIN,fwasc,lwasc,fwalc,lwalc	4-2
CMLOAD,fwa,end	4-4
DMP,p1,p2	4-5
ECLOAD,fwa,end	4-4
END	4-4
ENTRY,(eptname,...)	4-5
EXECUTE,eptname,(p,...)	4-4
FILES,(lfn,...)	4-3
LIB,(libname,...)	4-3
LIBLOAD,libname,(eptname,...)	4-3
LOAD,(lfn,...)	4-3
MAP,p,lfn	4-3
NOGO	4-5
OMIT,(eptname,...)	4-3
PASSLOC,((id,t,b,a),...)	4-5
PD,p	4-3
PRESET,p	4-3
PRESETA,p	4-3
PS,p	4-3
SATISFY,(libname,...)	4-3
SLOAD,lfn,(name,...)	4-3
SUBST,(pair,...)	4-3
USE,(eptname,...)	4-3
USEP,(pname,...)	4-3

FAST DYNAMIC LOADER MACROS

		Page
ENTHDR		8-6
ENT	eptname,addr	8-6

OVERLAY

OVCAP.	8-6
OVCAP(lfn)	8-6
OVERLAY(lfn,l_1,l_2,OV=n)	6-4
OVERLAY(lfn,l_1,l_2,origin)	6-4
LOADREQ paddr,rcl,flag	6-6

FAST OVERLAY LOADING

RJ	FOL.LOV	load overlay (CMM inactive)	6-10
RJ	CMM.LOV	load overlay (CMM active)	6-10
RJ	FOL.LOD	load overlay as data	6-10
RJ	FOL.GDE	get directory entry	6-11

SEGMENTATION DIRECTIVES

	END	eptname,...	7-15
bname	EQUAL	bname,...	7-14
segname	GLOBAL	bname,...	7-12
segname	INCLUDE	program,...	7-9
	LEVEL		7-9
tname	TREE	expression	7-8
	COMMON	bname,...	7-14

CAPSULE AND OVCAP LOADING AND UNLOADING

RJ	FDL.LDC	load capsule	8-3
RJ	FDL.ULC	unload capsule	8-4
RJ	FDL.CDG	compact group directory	8-4
RJ	FDL.UGD	unload group directory	8-4
RJ	FDL.LOC	load OVCAP	8-7
RJ	FDL.UOC	unload OVCAP	8-7
RJ	FDL.REL	relocate and link	8-5
RJ	FDL.UNR	delink	8-5

TRAP DIRECTIVES

FRAME	5-1
AT addr	5-3
EVERY n	5-3
FOR n	5-3
FROM addr	5-3
REG	5-3
START n	5-3
UNTIL n	5-3
TRACK	5-1
EVERY n	5-4
FROM addr	5-4
START n	5-4
TO addr	5-4
UNTIL n	5-4
WHEN addr	5-4
WHEN reg	5-4

60429800

CONTROL DATA

CYBER LOADER VERSION 1 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 2

NOS/BE 1

REVISION RECORD

Revision	Description
A (10/01/75)	Original printing. This manual is successor publication to publication number 60344200, for users of NOS 1.0, NOS/BE 1.0 and SCOPE 2.1 operating systems.
B (03/01/76)	This revision documents Version 1.1 of the CYBER loader. Features documented include CP139, 5400 tables, and CP147, COMPASS LDSET instruction.
C (07/16/76)	This revision documents Version 1.2 of the CYBER loader. The Common Segment Loader, feature 149, is included in this release.
D (10/29/76)	This revision to Version 1.2 of the CYBER loader documents default library changes, feature CP146A, and the loader interface with Common Memory Manager. This revision is effective with release of PSR level 439.
E (03/25/77)	This revision documents Version 1.3 of the CYBER loader, which introduces the Fast Dynamic Loading capability, feature CP161. (References to SCOPE 2 have been deleted from this manual.) This revision is effective with release of PSR level 446.
F (04/15/78)	This revision documents Version 1.4 of the CYBER Loader, which introduces the Fast Overlay Loader and multiple entry points in main overlay, extends FDL and SEGLOAD directive syntax, and adds new DEBUG PRESET option, and 8 lines/inch on map. This revision is effective with release of PSR level 472.
G (06/15/79)	This revision documents Version 1.5 of the CYBER Loader which introduces SEGLOAD common blocks, the PTEXT table, and supports FORTRAN Version 5 SAVE option. This revision is effective with release of PSR level 498.
H (02/26/82)	This revision documents Version 1.5 of the CYBER Loader which introduces interactive use of the loader under the batch subsystem of NOS and includes changes to the LIBRARY control statement under NOS 2. This revision is effective with release of PSR level 552. This is a complete reprint.
J (09/07/84)	This revision documents Version 1.5 of the CYBER Loader. It includes miscellaneous corrections and modifications to the PD and PS options of the LDSET statement, LOADREQ macro, and OVERLAY directives. Revised at PSR level 601.
K (04/04/86)	This revision documents Version 1.5 of the CYBER Loader. It includes miscellaneous corrections and modifications. Revised at PSR level 647. With this revision, the manual no longer applies to NOS Version 1.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Inside Front Cover	K
Title Page	-
ii	K
iii/iv	K
v thru ix	K
1-1 thru 1-9	K
2-1	K
2-2	K
2-3	J
2-4	H
2-5 thru 2-7	K
2-8	H
2-9	K
2-10	J
2-11 thru 2-13	K
3-1 thru 3-3	K
4-1	J
4-2 thru 4-5	H
4-6	J
5-1	K
5-2	K
5-3	H
5-4	H
5-5	K
5-6	H
6-1	H
6-2	F
6-3	H
6-4	K
6-5	H
6-6	J
6-7	K
6-8 thru 6-11	H
7-1	K
7-2	H
7-3	H
7-4 thru 7-15	K
8-1	F
8-2 thru 8-3	H
8-4 thru 8-7	K
A-1	H
A-2	H
B-1	H
B-2	K
B-2.1/B-2.2	K
B-3	K
B-4 thru B-6	H
B-7	K
B-8 thru B-18	H
C-1 thru C-3	H
D-1	K
D-2	H
D-3 thru D-5	K
D-6	J
D-7	J
D-8	K
D-9 thru D-15	H
D-16 thru D-28	K
E-1	H
E-2	H
E-3 thru E-22	K
Index-1 thru -3	K
Comment Sheet/Mailer	K
Inside Back Cover	K
Back Cover	-

PREFACE

This publication describes features of CYBER Loader Version 1.5, which is included as part of the following operating systems:

- NOS 2 for the CONTROL DATA® CYBER 180 Series; CYBER 170 Series; and CYBER 70 Models 71, 72, 73, and 74
- NOS/BE 1 for the CDC® CYBER 180 Series; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems

Extended memory for the CYBER 170 Model 176 is large central memory (LCM) or large central memory extended (LCME). Extended memory for all other NOS or NOS/BE computer systems is extended core storage (ECS) or extended semiconductor memory (ESM). In this manual, the acronym ECS refers to all forms of extended memory unless otherwise noted. Programming information for the various forms of extended memory can be found in the COMPASS reference manual and in the appropriate computer system hardware reference manual.

You might also want to consult the NOS System Information manual. It is an online manual that includes brief descriptions of all NOS and NOS product manuals. You can access this manual by logging into NOS and simply entering the command EXPLAIN.

The following publications are listed alphabetically within groupings that indicate relative importance to the reader.

The following manuals are of primary interest:

Publication	Publication Number	NOS 2	NOS/BE
Common Memory Manager Reference Manual	60499200	X	X
COMPASS Version 3 Reference Manual	60492600	X	X
INTERCOM Version 5 Reference Manual	60455010		X
NOS Version 2 Reference Set, Volume 2 Guide to System Usage	60459670	X	
NOS Version 2 Reference Set, Volume 3 System Commands	60459680	X	
NOS Version 2 Reference Set, Volume 4 Program Interface	60459690	X	
NOS/BE Version 1 Reference Manual	60493800		X
NOS/BE Version 1 System Programmer's Reference Manual	60494100		X

The following manuals are of secondary interest:

Publication	Publication Number	NOS 2	NOS/BE
CYBER Record Manager Advanced Access Methods Version 2 Reference Manual	60499300	X	X
CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700	X	X

United States sites can order CDC manuals from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103. Other sites should order CDC manuals through their local country sales office.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

The last page of this manual is a comment sheet. Please use it to give your opinion on the manual's usability, to suggest specific improvements, and to report any errors. If the comment sheet has already been used, you may mail your comments to:

Control Data Corporation
Publications and Graphics Division
P.O. Box 3492
Sunnyvale, CA 94088-3492

Additionally, if you have access to SOLVER, an online facility for reporting problems, you can use it to submit comments about the manual. Use LDR as the product identifier.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

INTRODUCTION 1

Compilers and assemblers generate object code that needs further processing before it can be executed. The loader is the system program that provides for the placement of object code into memory and makes the object code ready for execution. The loader also performs load-related services such as generation of a load map, presetting of unused memory to a user-specified value, and generation of overlays, segments, capsules, and OVCAPs.

INPUT REQUIREMENTS

Loader input is obtained from one or more local files and/or libraries. File names beginning with Zs (for example, ZZZZZDF) are reserved by the operating systems for system use. The user should refrain from using file names beginning with Zs.

It should be noted that parts of both the operating system and the product set use special characters such as the $ (dollar sign), the = (equals sign) and the . (period). Use of these symbols can cause naming conflicts. To avoid this conflict, it is recommended that names for entry points, common blocks, and program modules (a main program, subprogram or function) follow the appropriate operating system conventions used for local file names.

Entry point names, common block names, and program names used during load operations are often referred to as Linkage Symbols. In addition to conforming to any syntax requirements for the operating system and compiler or assembler which generates the object code, they must also conform to the loader requirements for such symbols. The loader requirements are as follows:

- A name must be no longer than seven characters in length.
- The first character must be alphabetic.
- If a name contains any special characters, such as non-alphanumeric, then the entire name must be delimited with dollar signs.
- If there are any dollar signs as part of the name, then two dollar signs must appear in the name where each dollar sign is to appear, and the entire name must be delimited by dollar signs.

Figure 1-1 shows examples of valid linkage symbols.

Example 1 represents the symbol SYM0001. Example 2 represents the symbol CTL.RM. Example 3 represents the symbol CTL$VAL.

```
Example 1:

    SYM0001

Example 2:

    $CTL.RM$

Example 3:

    $CTL$$VAL$
```

Figure 1-1. Examples of Valid Linkage Symbols

TYPES OF LOAD OPERATIONS

The type of loading that takes place during a load operation is dependent upon program size and program organization. For example, the input to the loader can be either one or more relocatable programs or a single absolute program. When the input to a load operation is one or more relocatable programs, the loading operation that takes place is called a relocatable load. When the input to a load operation is a single absolute program, the loading operation that takes place is called an absolute load.

A program can be too large to be efficiently loaded into central memory at one time. A user might, therefore, wish to divide a program into either overlays, segments, OVCAPs, or capsules to utilize central memory (CM) more efficiently.

RELOCATABLE LOAD

A relocatable load involves the allocation of one or more contiguous storage areas known as blocks. These blocks are defined by object programs and contain machine instructions that require relocation. They do not, however, specify origin addresses. During loading, the loader establishes origins for all the blocks and adjusts all addresses accordingly. It is, therefore, possible to load together object programs produced by independent compilations and assemblies.

ABSOLUTE LOAD

An absolute load consists of loading one or two blocks, one of which must reside in central memory and one of which is optional, and, if present, resides in extended core storage (ECS). The origins of the blocks are specified in the absolute program. Absolute loading involves no relocation of addressesor linking of externals because this is done when the absolute image is built.

BASIC LOAD

A basic load is one in which all of the object code is loaded into memory at the same time, resulting in a single core image. Most programs are loaded in this manner because they are relatively small compared to the amount of available memory.

SEGMENTATION

For very large programs, basic loading might use memory inefficiently because much of central memory could be occupied by programs not currently in execution. Also, a program could be too large to fit in the available memory. For these reasons, a user might decide to divide a program into several small portions called segments.

With segmented loading, only certain portions of the executing program need to be in central memory concurrently because the total program consists of more than one segment. Different segments reside in the same area of memory at different times. Depending on execution requirements, different segments are loaded and/or unloaded automatically.

Segmented loading is initiated by the execution of a SEGLOAD control statement that causes SEGLOAD directives to be processed. The SEGLOAD directives contain information necessary for segmentation.

Some features of segmentation are:

- After segments are generated, their loading is automatic.
- References between segments can be either upward or downward in memory.
- Object programs that are to be included in segments can be selected from more than one file.
- The job field length can be adjusted dynamically during program execution as segments are loaded and unloaded if the user so desires.

OVERLAYS

Overlays are an alternative to segmentation. Overlay generation is initiated when an OVERLAY directive is encountered on a load file. Loading of overlays resulting from overlay generation is not automatic. A currently loaded overlay must explicitly request the loading of other overlays as needed.

Some features of overlays are:

- Three levels of overlays are possible. One main overlay is allowed at the lowest level. Up to 64 overlays are allowed at the primary level. Up to 64 overlays are allowed at the secondary level for each overlay at the primary level.
- The main overlay can be loaded in various ways; higher level overlays must be loaded by an explicit call from a currently loaded overlay.
- Memory references between one overlay and another can be downward only (except for the initial entry to a just-loaded overlay). That is, programs in the main overlay cannot reference entry points in any other overlay; programs in a primary overlay can reference entry points in the main overlay; and programs in a secondary overlay can reference entry points in either the main overlay or the associated primary overlay.

The NOS user should be aware of a condition that can occur when using libraries in the global library set. When a main overlay is loaded from a library, all subsequent overlay loads are from that same library.

If multiple main overlays (and their associated primary and secondary overlays) reside on the same library, a request to load an overlay for program A could result in the load of a primary or secondary overlay belonging to program B. This load could occur because LIBGEN does not diagnose duplicate program names during library generation. Thus, duplicates could be created unknowingly. Knowledge of this condition is especially inportant for NOS users of libraries containing execute-only files because the load map is always suppressed for such files.

CAPSULES

The Fast Dynamic Loading (FDL) capability provides a way for the COMPASS user to load specially processed relocatable code during program execution. Before routines can be loaded by FDL, the relocatable code must be formed into capsules that can be loaded at arbitrary addresses. The process of forming routines into capsules is called capsule generation. Capsule generation takes place during a load operation when one or more load files are read and grouped according to directives provided by the user.

Some features of capsules are:

- Every capsule must be a member of a group, which provides a means of associating related capsules with each other.
- Capsules must be placed in libraries before they can be loaded by an executing program.
- An executing program must call an FDL subroutine, specifying a group name and a capsule name, to load or unload a capsule.
- Entry points and external references of capsules in the same group are automatically linked or delinked when the capsule is loaded or unloaded by the FDL subroutine.
- Statically loaded code can interface with dynamically loaded code through the use of PASSLOC and/or ENTRY tables.
- A capsule can also be loaded by the basic loader in a relocatable load sequence.

OVCAPS

A special type of capsule called OVCAP (overlay-capsule) provides a way for overlayed programs to load specially processed relocatable code during program execution via FDL. An OVCAP is analogous to a primary overlay because common block and entry points of the preceding main overlay are used for linking. An OVCAP, therefore, is not a standalone entity; it requires an associated main overlay to be in central memory for proper execution.

OVCAP generation takes place during a load sequence that is already known to be an overlay generation load sequence. Object directives encountered in the load input stream instruct the loader to generate OVCAPs.

OVCAPs have the same features as capsules except that they cannot be loaded by the basic loader in a relocatable load sequence.

CALLING THE LOADER

A load operation is initiated by control statements executed within a job stream or by user calls executed within a running program. A load operation can be further affected by loader object directives.

CONTROL STATEMENT LOADS

Most loads are initiated by control statements that are executed within a job stream. These statements must conform to the syntax requirements of the operating system defined in its reference manual. The types of control statements are name call statements, loader statements, and completion statements.

Name Call Statements

A name call statement specifies, as its keyword, either a file to be loaded or an entry point in some program to be loaded. If the initial statement in a sequence is a name call statement, it is the only statement in the loader sequence.

Loader Statements

A loader statement specifies an explicitly defined loader command that is to be performed. When a loader statement is encountered, the system reads and translates into loader requests all subsequent control statements until it encounters a completion statement. The loader statements are as follows:

- LOAD

  Specifies files from which object programs are to be loaded

- SLOAD

  Specifies a local file from which selected programs are to be loaded

- SEGLOAD

  Specifies that the load is to be a segmented load

- LIBLOAD

  Specifies a library from which one or more programs are to be loaded

- SATISFY

  Specifies libraries that are to be used to satisfy externals prior to normal satisfaction at load completion

- LDSET

  Specifies user control of a variety of load operations for the current load only; the options are summarized in table 1-1

- GROUP

  Specifies the name of a capsule group that is to be generated

- CAPSULE

  Specifies the name of a capsule that is to be generated

Completion Statements

The loader completion statements cause completion of a load operation initiated by a loader statement. The loader completion statements are as follows:

- EXECUTE

  Specifies that execution of the loaded program is to take place after load completion

- NOGO

  Specifies that execution of the loaded program is not to take place after load completion

When a name call statement is encountered within a load sequence, it has the same effect as a load completion statement; that is, it causes completion of the preceding load operation.

LOADER-RELATED CONTROL STATEMENTS

Some control statements that are processed by the operating system can also affect loader performance. These control statements are as follows:

- MAP

  Specifies the default option for load maps for load sequences requested by the job

- LIBRARY

  Specifies a set of global libraries to be searched for externals and name call statements, and the order in which the libraries are to be considered

- RFL

  Specifies central memory (CM) field length for program execution

- REDUCE

  Controls automatic reduction of execution field length

- TRAP

  Specifies that the debugging aids execution time routine, TRAPPER, is to be loaded with the next relocatable load sequence

TABLE 1-1. LDSET OPTION SUMMARY

Option	Function
MAP	Specifies the type of load map to be generated
PRESET, PRESETA	Specifies the values to which unused memory is set prior to execution of the loaded program
ERR	Specifies method of handling loader errors
REWIND, NOREWIN	Alters the default option for rewinding of files prior to loading
USEP	Specifies that the indicated object programs are to be loaded whether or not they are needed to satisfy external references
USE	Forces loading of programs containing the indicated entry points to assure that specified entry points are included in the load
SUBST	Changes external references to entry point names to use other entry point names instead
OMIT	Specifies entry point names that are to remain unsatisfied whether or not the module containing these entry point names is loaded
FILES, STAT	Permits CYBER Record Manager users to ensure that library programs are loaded for the processing of specified files
EPT, NOEPT	Controls the availability of entry points within capsules and OVCAPs
LIB	Specifies libraries that are to make up the local library set
PD	Specifies print density for load map
PS	Specifies page size for load map
COMMON	Specifies common blocks to be made available to all segments which reference them in a SEGLOAD

USER CALLS

User calls allow for the initiation of load operations from within a running program. In order to call the loader directly, the user must format the necessary request tables.

The request tables are formatted by use of the COMPASS LOADER macro and the LDREQ macros. The allowable LDREQ macro options are listed in table 1-2.

TABLE 1-2. LDREQ OPTION SUMMARY

Option	Function
BEGIN	Specifies the beginning of a sequence of LDREQ calls
END	Specifies the end of a sequence of LDREQ calls
LOAD LIBLOAD SLOAD EXECUTE NOGO SATISFY	See corresponding control statements
LIB MAP PRESET/ PRESETA USEP USE SUBST OMIT FILES/ STAT	See corresponding LDSET options in table 1-1
CMLOAD	Specifies that load input is to be fetched directly from central memory
ECLOAD	Specifies that load input is to be fetched directly from extended core storage (ECS).
ENTRY	Specifies entry point names that are currently being loaded and/or have been loaded previously for which an address is to be supplied
PASSLOC	Specifies addresses needed by the load during execution of overlayed, segmented, or encapsulated programs

LOADER OBJECT DIRECTIVES

Loader requests encountered within the load input stream are referred to as loader object directives. These requests are processed as they are encountered during the physical loading process.

Object directives can appear either in the form of LDSET loader tables or in the form of card images in the same manner as their control statement equivalents. The following directives are permitted:

- OVERLAY

  Specifies overlay generation

- OVCAP, GROUP, CAPSULE

  Controls capsule or OVCAP generation

THE LOADING PROCESS

The input to the loader can be either absolute overlays or relocatable object code. The input is fetched from local files and libraries.

The loading process involves the following sequence of events:

1. Read and check all control statements in the load sequence.
2. Process all control statements in order (includes loading and relocating programs).
3. Search libraries to satisfy externals.
4. Determine execution field length.
5. Write load map.
6. Initiate program execution.

LOADING AND RELOCATION

The loading process by which the loader assigns a portion of memory to each program and adjusts the addresses accordingly is called relocation.

For absolute overlays, memory was allocated at the time the absolute image was built. Thus, loading of absolute overlays does not require relocation.

Relocation is necessary for relocatable object code because the object code consists of independent programs that have no fixed origin and no addresses in references to common blocks.

LINKING PROGRAMS AND SATISFYING EXTERNALS

A program can reference common blocks and other externals. The loader must match up, by name, external references with entry points prior to program execution. In the process, addresses of external references are set accordingly. This operation is known as satisfying externals.

The loader uses entry points that were loaded from the load files to satisfy external references. Then, the loader searches libraries for entry points that are still unsatisfied. For example, the loader might search the FORTRAN library for the entry point SQRT. (See the detailed discussion of library searching later in this section.)

WEAK EXTERNALS

Normal externals force the loading of routines to satisfy their requirements. Under certain circumstances, it might be desirable to link to a routine only if another external forced it to load. When this is the case, a weak external is used instead of a normal one. The program can check the presence or absence of the external linkage by checking the 2^{17} bit in the referencing instruction. If the bit is set, the external was not satisfied.

This mechanism is used, for example, in utility routines which allow the user (via LDSET,USE) to determine the routines actually loaded. Such routines can test for the presence or absence of loaded modules and react accordingly.

FIELD LENGTH DETERMINATION

The field length (FL) is the amount of central memory available for execution of a program. Field length assignment is normally the function of the operating system; however, there are circumstances in which the user might wish to override this facility. (See the detailed discussion of field length assignment later in this section.)

LOAD MAP

The load map is a printout showing how memory is allocated by the loader. The load map can be printed out automatically, depending on the installation option, or under control of the MAP options of the LDSET statement and LDREQ macro call.

Load maps are available only on relocatable loads. Under NOS, the load map is always suppressed for execute-only files.

PROGRAM INITIATION

After writing the load map, if any, program execution is begun automatically, unless the load sequence is terminated by a NOGO control statement.

DEBUGGING AIDS

The loader provides an execution time routine, TRAPPER, that can be loaded with the user's program and used as an aid to debugging.

The TRAP control statement requests that TRAPPER be loaded. The statement constitutes a separate load sequence and affects the next relocatable load sequence; it must, therefore, precede all the statements in the load sequence to which it applies. The TRAP statement has two directives that specify the type of debugging information that is to be supplied to the user. These directives are FRAME and TRACK.

The FRAME directive requests snapshot dumps of registers and areas of memory at selected locations within the program.

The TRACK directive requests information needed to analyze a series of instruction executions within a program. This information consists of the contents of locations and registers after execution of instructions in the designated range.

The TRAPPER routine is discussed in section 5.

LIBRARY SEARCHING

Libraries are searched either to satisfy externals or to locate programs called by name call statements. (Weak externals are ignored.) Two lists of libraries are used by the loader: the global library set and the local library set. One additional library SYSLIB can also be searched.

Global libraries are introduced by a LIBRARY statement (see section 3). They remain in effect from the time a LIBRARY statement is first encountered until a new LIBRARY statement is encountered.

Local libraries are introduced by the LIB parameter on the LDSET loader statement, or the LIB parameter on the LDREQ macro, or a LIB directive in LDSET tables contained within programs being loaded.

The order of search for externals is as follows:

1. The global library set
2. The local library set
3. SYSLIB (not searched by default for capsule or OVCAP generation)

The loader's order of search for name call statements is as follows:

1. Local files
2. The global library set
3. The local library set
4. NUCLEUS library (NOS/BE only)

Other components of the operating system process name call statements differently, as discussed in section 2.

When a library search begins, the first library in the library set is searched for programs needed to satisfy all externals that can be satisfied from the library. These programs are loaded. If new unsatisfied externals are generated by this loading, the same library is searched again in an attempt to satisfy any new unsatisfied externals. The loader continues to search the same library until no new externals are loaded. The process is repeated for each library in turn.

After the last library is searched, the loader goes back to the first library automatically if all externals are not yet satisfied. This circular search continues as long as necessary. The loader stops searching, however, as soon as either all externals are satisfied, or a complete circular search establishes that no more externals can be satisfied from any of the libraries.

If an internal LDSET(LIB=xxx) table is encountered while loading a program, the search of the current library is completed in the normal way. At the end of the search of the current library, however, the library set is altered by adding the new library at the end of the local library set (if it is not already a part of the set) before the search continues with the next library.

If an LDSET(LIB) table with no library list is encountered while loading a program, the search of the current library is completed in the normal way. However, at the end of the search of the current library, the local libraries are removed from the library set and the search continues from the beginning of the remaining (global) library set. Under NOS, if the load consists of a name call statement that specifies an execute-only file, all user libraries are skipped during library searches.

The following rules for duplication of program or entry point names apply to loading in general.

- Duplicate programs:

  Are skipped, causing a nonfatal error, when the second or subsequent occurrence is from a load file

  Are loaded, causing a nonfatal error, when the second or subsequent occurrence is during the satisfying of externals under NOS/BE

  Are skipped, with no message being issued, when satisfying externals under NOS

- Duplicate entry point names are ignored; the loader issues a nonfatal error.

When a user library is created under NOS with the NX=n parameter of the LIBGEN control statement either omitted or specified as zero, the satisfying of externals from libraries can result in duplicate entry points. These duplicate entry points occur because all dependent programs are cross-linked. This cross-linkage causes the loader to load all dependent programs without checking for duplicate entry points. Refer to the NOS reference manual, volume 1 for more information about the NX parameter of the LIBGEN control statement.

FIELD LENGTH CONTROL

Normally, the user need not worry about the amount of central memory assigned to the job because this is automatically handled by the loader. The following subsections describe how the loader assigns memory for job execution.

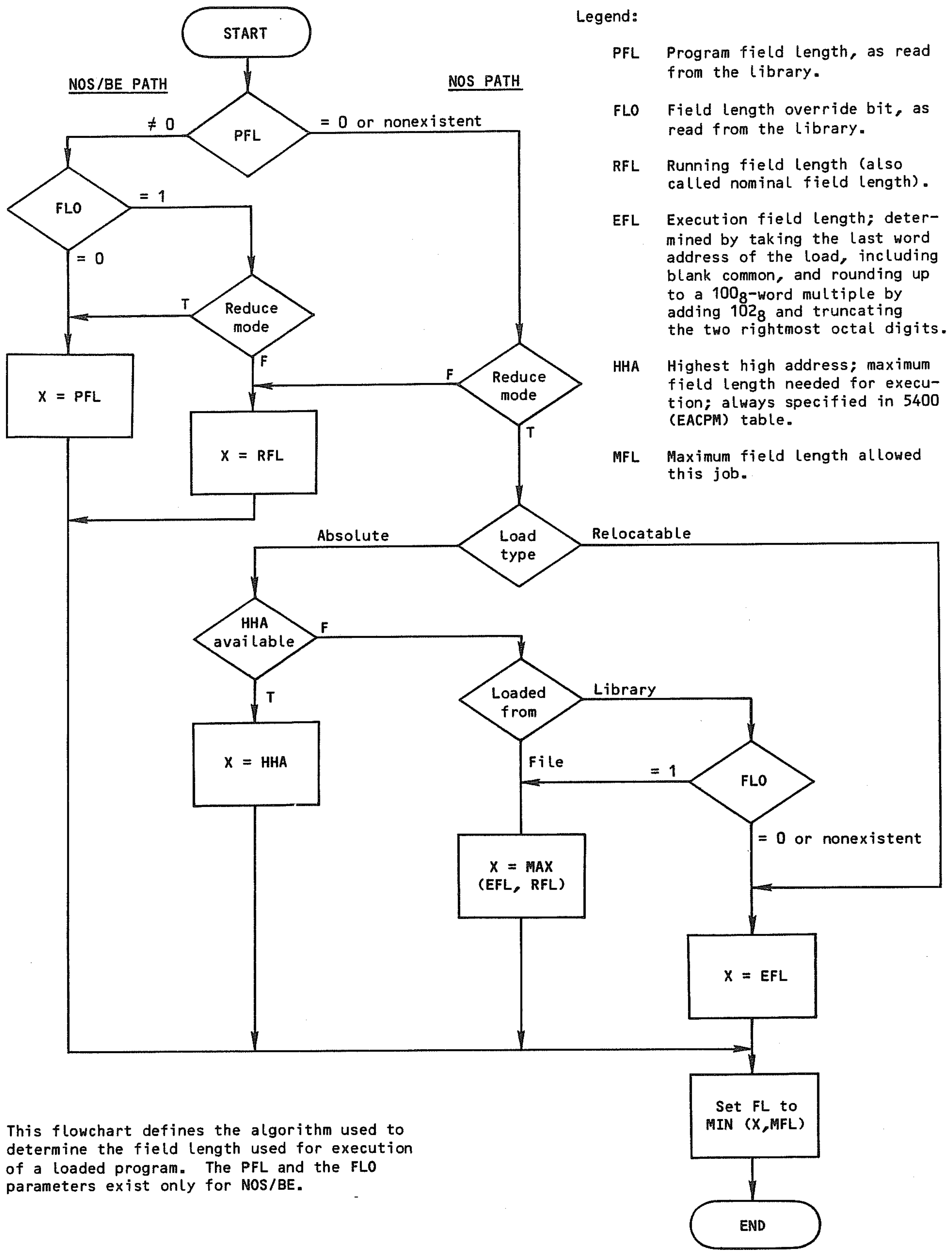
CONTROL STATEMENT LOADS

For control statement loads, the loader automatically obtains the memory necessary for the loading operation. (NOS/BE users should not use RFL statements that specify less than 14000_8 words because this prevents initiation of the loader.)

Programs Loaded by CYBER Loader

When the loader transfers control to a program to begin execution, it sets the field length according to the algorithm illustrated in either figure 1-2 (for nonsegmented programs) or figure 1-3 (for segmented programs).

If a job is in REDUCE mode, the setting of field length is controlled. All jobs start in REDUCE mode except jobs running under NOS/BE that have CM on the job statement. A job gets out of REDUCE mode by issuing a REDUCE(-) statement for NOS, or an RFL or REDUCE(OFF) statement for NOS/BE. The use of REDUCE(-) is inefficient and is not recommended. The job returns to REDUCE mode by issuing a REDUCE statement for NOS and NOS/BE batch jobs, or a REDUCE(ON) statement for NOS/BE INTERCOM jobs.



Legend:

PFL	Program field length, as read from the library.
FLO	Field length override bit, as read from the library.
RFL	Running field length (also called nominal field length).
EFL	Execution field length; determined by taking the last word address of the load, including blank common, and rounding up to a 100_8-word multiple by adding 102_8 and truncating the two rightmost octal digits.
HHA	Highest high address; maximum field length needed for execution; always specified in 5400 (EACPM) table.
MFL	Maximum field length allowed this job.

This flowchart defines the algorithm used to determine the field length used for execution of a loaded program. The PFL and the FLO parameters exist only for NOS/BE.

Figure 1-2. Field Length Algorithm for Nonsegmented Programs

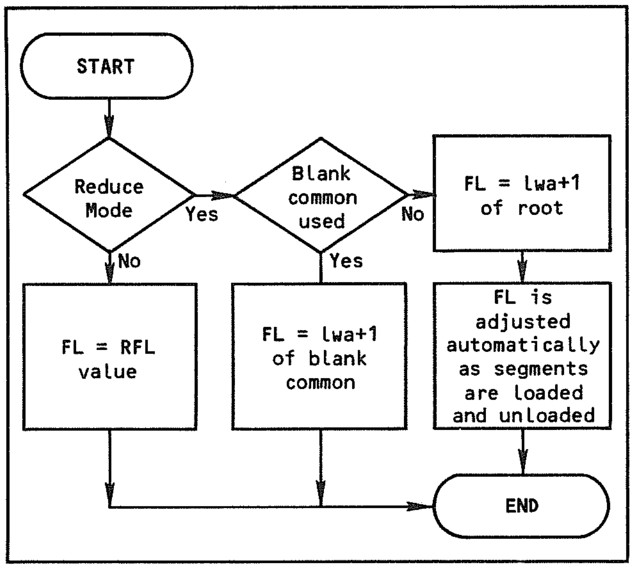

START
Reduce Mode
Yes
No
Blank common used
Yes
No
FL = lwa+1 of root
FL = RFL value
FL = lwa+1 of blank common
FL is adjusted automatically as segments are loaded and unloaded
END

Figure 1-3. Field Length Algorithm for Segmented Program

The field length is also controlled by either the value of the last RFL statement or the CM parameter on the job statement, if no RFL statement is present. (A default is used if neither an RFL statement nor the CM parameter is present.)

Under NOS, do not use REDUCE(-) without an RFL statement because the RFL value is zero if no RFL statement is present and there is no CM parameter on the job statement.

The MFL statement sets the RFL value to zero and the maximum available field length for loading.

NOS/BE System Programs

The NOS/BE operating system uses the same algorithm as the loader (see the flowchart in figure 1-2) except that the test for HHA is always false because HHA is not recognized.

NOS System Programs

For an operating system load under NOS, the first of the following values that is available is used for field length assignment:

- *FL value from LIBDECK or SYSEDIT input
- Value of either the RFL= or MFL= entry point (see the NOS reference manual, volume 2, for a detailed discussion)
- RFL value from CM on the job statement or RFL statement
- HHA value from the 5400-table header
- System default value

USER CALL LOADS

Normally, adequate field length for a user call load must be provided by a MEMORY macro call prior to calling the user call loader. This is not necessary, however, if the Common Memory Manager (CMM) is used to allocate memory (as explained in section 4).

OVERLAY LOADS

If the CMM parameter is present on the LOADREQ macro call, CMM ensures that adequate memory is available to load the overlay. Otherwise, enough memory must be available prior to the LOADREQ call. This normally requires no special action by the user because when the loader sets the execution field length for the main overlay, it ensures that there is enough central memory to load all higher level overlays.

CAPSULE AND OVCAP LOADS

Memory assignment is always handled by CMM for capsule and OVCAP loads.

MEMORY ALLOCATION RULES

The discussion in the following subsections applies to relocatable loads only. Memory is allocated for absolute programs when the ABS image is built.

BASIC LOADS AND CAPSULE GENERATION LOADS

For basic loads and capsule generation loads, programs are assigned memory in the order in which they are encountered. This means that the programs from load files are assigned memory first, followed by programs from libraries. Programs from libraries are loaded randomly, and the order in which the object programs are loaded can vary from run to run.

Labeled common blocks are interspersed within the program. Each block is placed before the first program that references the block. Blank common is placed at the end of all programs.

Figures 1-4 and 1-5 show how programs can be assigned in memory. Figure 1-4 shows a normal load diagram of a nonsegmented program. Figure 1-5 shows a load diagram of a nonsegmented program in which the TRAPPER debugging aid is included.

NOTE

Data is normally loaded into labeled common blocks by only one of the programs declaring the block. The data is normally, but not necessarily, loaded by the first program declaring the labeled common block (for example: a COBOL or FORTRAN main program). If more than one program contains text to be loaded into the same common block and the text contains addresses to be relocated, then the addresses are relocated each time the text is loaded. This relocation causes unpredictable (such as areas being overwritten) and usually fatal results.

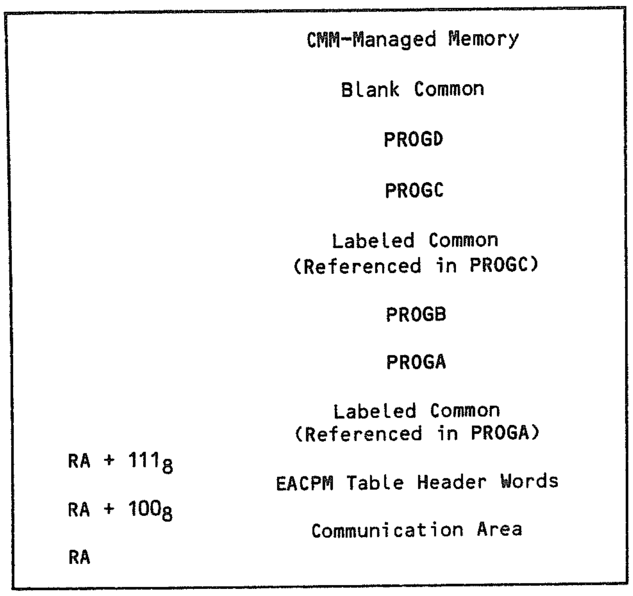


Figure 1-4. Normal Load Diagram

USER CALL LOADS AND OVERLAY GENERATION LOADS

Memory allocation for user call loads and overlay generation loads is the same as described for basic loads, except for the assignment of common blocks. If a common block is first declared on the original basic load (for a user call load) or in a lower overlay (for overlay generation load), a new copy of the common block is not created. The first copy of the common block is used by all programs.

SEGMENT GENERATION LOADS

For segment generation loads, programs are assigned to various segments according to directives supplied by the user. These rules are discussed in section 7.

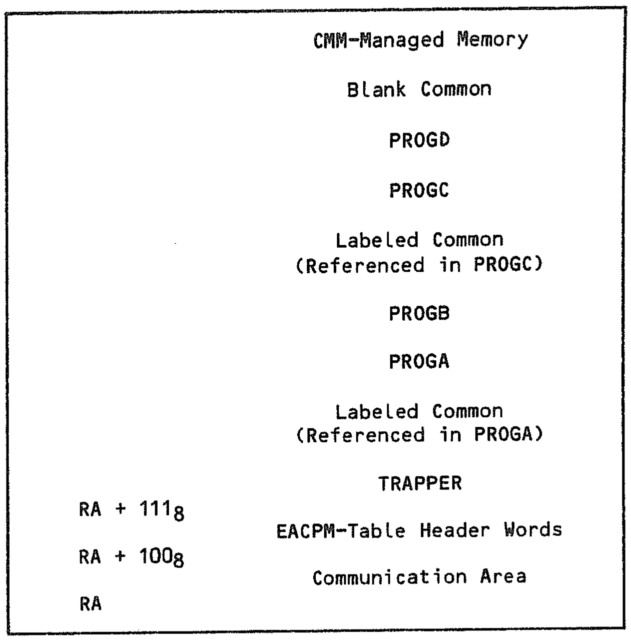


Figure 1-5. Load Diagram Containing TRAPPER

Labeled common blocks are duplicated in every segment from which they are referenced unless they are declared as COMMON or GLOBAL. If declared as GLOBAL, labeled common blocks are assigned to the segment that specifies labeled common blocks; however, they are addressable by all segments other than those that actually overwrite them. If declared as COMMON, labeled common blocks are moved to the nearest common ancestor of all segments that reference them. Blank common is assigned at the end of all segments.

CAPSULE AND OVCAP LOADS

Any capsule or OVCAP that is loaded during execution is placed in the area of memory controlled by CMM.

LOADER CONTROL STATEMENTS 2

Most loads are initiated by control statements. These statements must conform to the syntax requirements of the operating system defined in the appropriate reference manual.

STATEMENT DESCRIPTORS

The three types of control statements are name call statements, loader statements, and completion statements. (The completion statements, EXECUTE, and NOGO are also loader statements; name call statements also function as completion statements.)

If the initial statement in a sequence is a name call statement, it is the only statement in the loader sequence; the system is able to initiate and complete the load operation without additional control statements.

If the initial statement is a loader statement, the system reads and translates into loader requests all subsequent control statements until it encounters one of the following completion statements:

- EXECUTE
- NOGO
- name call

The name call statement is a completion statement when it occurs within a load sequence. It causes completion of the load sequence in effect at the time it is executed.

During the search for a load sequence completion statement, control statements that are normally operating system control statements (for example, LIBRARY, COMMENT, REWIND) are interpreted as name call statements. That is, if an operating system control statement is encountered during a load sequence, the system attempts to complete the load sequence as if a name call statement were executed. Exceptions are MAP and REDUCE, which the system recognizes inside the loader control statement sequence for compatibility with previous systems. The DMP statement is also recognized; however, it is ignored and a dayfile message is issued. (The MAP, LIBRARY, REDUCE, and RFL control statements are discussed in section 3.) An EXIT statement must not appear inside a load sequence.

Upon encountering the last loader control statement in the sequence, the loader performs the requested load operations by interpreting the table of loader requests resulting from control statements. Figure 2-1 shows how control statements are processed by both the operating system and the loader.

NAME CALL STATEMENT

A name call statement specifies either a file, or on NOS/BE, an entry point name within a program that is to be loaded into central memory. In response to a name call statement, the system is able to initiate and complete a load operation. The format of the name call statement is shown in figure 2-2.

The loader determines the name type, as follows:

1. The loader searches the list of local files belonging to the job. If it finds the name in the list, it processes the call. On interactive name call loads where the name is the same as a system routine, the system routine is loaded and executed.

2. The loader checks the local library set for an entry point identical to that on the name call statement. If any library in the local library set contains a program that has an entry point the same as the name call, the loader loads the program, completes the load, and begins execution at the entry point name.

   The loader searches the libraries in the sequence specified for the library set. If the name is in more than one of the libraries, the loader uses the first one it encounters. On NOS/BE, this process is continued for the global library set.

3. The loader checks the system library NUCLEUS on NOS/BE, and the central library directory on NOS, for an entry point or a program name identical with that on the name call statement. If it finds the program containing the entry point, the loader loads the program, completes the load, and begins execution at the entry point.

Characteristics of the load differ according to whether the call is for a file name or an entry point name.

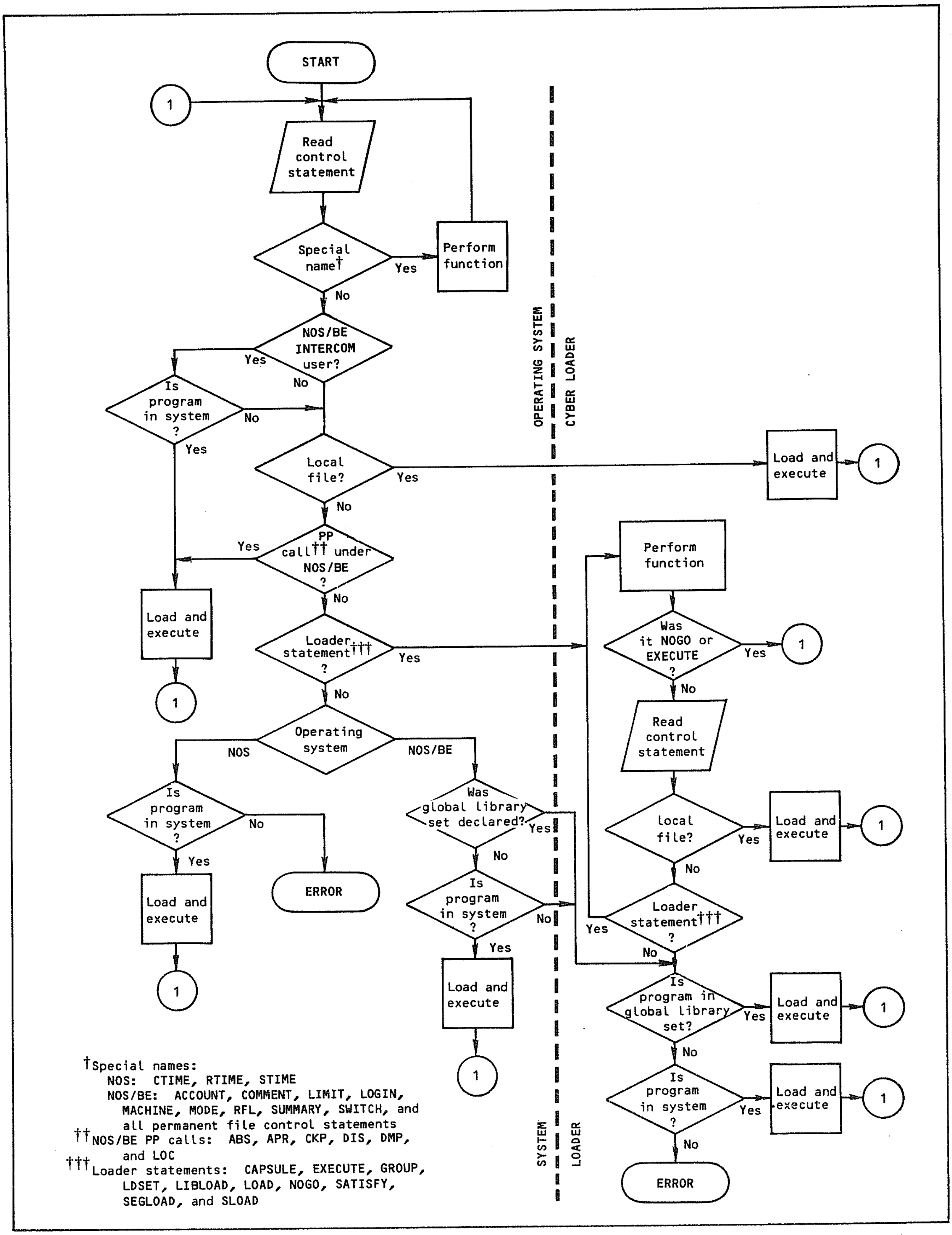
The file name call completes a loading process and initiates execution. The call can be interpreted as specifying a basic load, an overlay generation, or a segment generation.

The entry point name call statement causes loading and execution of a program containing the named entry point. The loader sequence results in a basic load; no overlay generation, capsule generation, or segmentation is possible.

Parameters on name call load statements are passed to the loaded program. Figure 2-3 shows examples of name call statements.

On NOS, if the LOAD involves an execute-only file, the loading of that file must be specified by a name call statement, and the load sequence must not include any other statements which specify loading.

In example 1, object programs from the files X and LGO are loaded. The resulting combined program is executed. Execution begins at the last transfer address encountered in the loading of object programs from files X and LGO.



†Special names:
NOS: CTIME, RTIME, STIME
NOS/BE: ACCOUNT, COMMENT, LIMIT, LOGIN, MACHINE, MODE, RFL, SUMMARY, SWITCH, and all permanent file control statements

††NOS/BE PP calls: ABS, APR, CKP, DIS, DMP, and LOC

†††Loader statements: CAPSULE, EXECUTE, GROUP, LDSET, LIBLOAD, LOAD, NOGO, SATISFY, SEGLOAD, and SLOAD

Figure 2-1. NOS and NOS/BE Control Statement Processing

```
name.

name(p1,...,pn)
```

name	Standard file name, or entry point name; one through seven characters.
p_i	Parameters to be passed to the loaded program. Specific requirements are according to the needs of the program. Specific parameter formats depend on the language that generated the program; they are described in the language reference manual.

Figure 2-2. Name Call Statement Formats

```
        Example 1:

            LOAD(X)
            LGO.

        Example 2:

            HIJ(A,B)
```

Figure 2-3. Examples of Name Call Statements

In example 2, a utility routine in the library is in absolute form and has HIJ as an entry point. The name call statement causes the loader (on NOS/BE) to load the associated absolute overlay and, after passing parameters A and B, to begin execution at entry point HIJ.

LOAD STATEMENT

In response to a LOAD statement (figure 2-4), the loader performs a load of object programs from each of the files specified in the sequence. If the LOAD statement specifies loading of an absolute overlay, only one file can be specified. Loading of an absolute overlay must be followed by an EXECUTE statement. The loader terminates reading of programs from a file when it finds an empty record (except under NOS), an end-of-file, or an end-of-information.

Consider the following LOAD statement:

```
LOAD(REDDOG,LOG/R,PLUM/NR)
```

This statement requests that object programs be loaded from files REDDOG, LOG, and PLUM. File REDDOG is either rewound or remains at its current position, depending on the rewind default; file LOG is initially rewound; file PLUM is loaded from its current position. Following the load, the next operation that takes place (possibly execution) depends on the next control statement in the loader sequence.

```
LOAD(lfn1,...,lfnn)
```

lfn_i	Name of file, optionally accompanied by a rewind indicator. Specification of a nonexistent or empty file results in a fatal error.	
	lfn	The rewind indicator is absent. Except for the INPUT file, the file is rewound unless this default is changed, either by an LDSET(NOREWIN) or by installation option. INPUT cannot be rewound by default.
	lfn/R	Forces rewind prior to loading. If rewind of INPUT is explicitly requested (by the R subparameter), the loader rewinds INPUT and makes no attempt to skip over the control statements.
	lfn/NR	Inhibits rewind prior to loading.

Figure 2-4. LOAD Statement Format

LIBLOAD STATEMENT

The LIBLOAD statement (figure 2-5) specifies that the loader is to load one or more programs from a particular library. Programs are specified through entry point names.

```
LIBLOAD(libname,eptname1,...,eptnamen)
```

libname	The name of the library containing object programs having the specified entry point names. The library must exist as either a user library (local file) or a system library.
$eptname_i$	Entry point names. If more than one entry point name refers to the same object program, fewer programs than names specified are loaded.

Figure 2-5. LIBLOAD Statement Format

The LIBLOAD request can specify entry points of either relocatable programs or absolute programs. However, relocatable programs and absolute programs cannot be loaded together; and for an absolute load, only one program can be loaded. Loading of an absolute overlay must be followed by an EXECUTE statement.

The loader allows LIBLOAD to be used for capsule generation, overlay generation (except as initial loader input), overlay execution, and segment generation. When used for capsule generation, LIBLOAD must be preceded by either a LOAD or a SLOAD statement. LIBLOAD cannot be used to load any of the programs named on the CAPSULE statement.

In processing a LIBLOAD statement, the entry point names are saved in a list. The library directory is scanned and each entry in the directory is compared against the list of entry points. The programs containing the specified entry points are loaded. If the loader fails to find an entry point in the library, a nonfatal error results.

Examples of the LIBLOAD statement are shown in figure 2-6.

```
        Example 1:

            LIBLOAD(ALGOL,AL77,AL78)
            HEIDI.

        Example 2:

            LIBLOAD(XXXLIB,BUN)
            EXECUTE(BUN)
```

Figure 2-6. Examples of the LIBLOAD Statement

In example 1, programs on library ALGOL containing the entry points AL77 and AL78 are loaded. The loader then processes the name call for file HEIDI, completes loading, and begins execution.

In example 2, the loader loads the program containing entry point BUN from the library named XXXLIB. Loading is completed and execution begins at the entry point BUN, as specified by the EXECUTE statement.

SLOAD STATEMENT

The SLOAD statement (figure 2-7) specifies that the loader is to load selected programs from a local file. Only programs specified on the request are loaded; all others are bypassed. Programs are loaded in the sequence encountered on the file. Searching takes place, as follows:

1. The file is rewound or not rewound, depending on the optional rewind subparameter or default.

2. Each requested program is loaded as it is encountered.

3. When all the specified programs have been found or when the loader encounters an empty record (except under NOS), end-of-file, or end-of-information, processing of the SLOAD statement is terminated. The file is left at its current position.

All programs on load files specified on SLOAD requests must contain prefix tables. When the file is read, the program name is extracted from the prefix table to determine whether the program should be loaded or discarded.

SLOAD(lfn,$name_1$,...,$name_n$)

lfn	Local file name, optionally accompanied by a rewind indicator:
	lfn — The rewind indicator is absent. Except for the INPUT file, the file is rewound unless this default is changed by an LDSET(NOREWIN) or by installation option. INPUT cannot be rewound by default.
	lfn/R — Forces rewind prior to loading. If rewind of INPUT is explicitly requested (by the R subparameter), the loader makes no attempt to skip over the control statements.
	lfn/NR — Inhibits rewind prior to loading.
$name_i$	Program names. At least one program must be specified.

Figure 2-7. SLOAD Statement Format

SLOAD ignores object directives; however, SLOAD can be used for overlay or capsule generation if the necessary object directives are established prior to the SLOAD request. If possible, the object directives should be included in the control statements of the load sequence. If the object directives cannot be included in the control statements of the load sequence, the SLOAD statement must be preceded by a LOAD statement.

If a SLOAD statement specifies loading of an absolute program, only one program can be specified. Loading of an absolute program must be followed by an EXECUTE statement.

A fatal error occurs if the file specified does not exist. A nonfatal error occurs if any of the programs specified are not on the file. For all of the formats, the file must be a sequential file; it cannot be a library file.

Consider the following example of the SLOAD statement:

```
    SLOAD(MOONDOG/R,SIN,COS,GETBA)
```

File MOONDOG is rewound and object programs SIN, COS, and GETBA are loaded from MOONDOG.

EXECUTE STATEMENT

The EXECUTE statement (figure 2-8) causes completion of the load followed immediately by execution of the loaded program at an entry point optionally specified. Optionally included are execution parameters that are to be passed to the loaded program.

```
EXECUTE.

EXECUTE(eptname)

EXECUTE(eptname,p1,...,pn)

EXECUTE(,p1,...,pn)
```

eptname	The name of the entry point in one of the loaded modules at which execution is to begin.
p_i	Execution-time parameters to be passed to the loaded program.

Figure 2-8. EXECUTE Statement Formats

For relocatable programs, processing of EXECUTE involves the following steps:

1. Subroutines are first loaded from the global library set and then the local library set to satisfy as many unsatisfied externals (not counting weak externals) as possible.

2. If any externals other than a weak external remain unsatisfied, a nonfatal error results. Any address field containing an unsatisfied external reference is filled with addr+400000_8, where addr is the address of the reference.

3. If blank common is declared by any program, its origin is established and references to it are satisfied.

4. If requested, a map of the completed core image is generated. Default map generation is an installation option.

5. Execution parameters, if any, are set up for the loaded program. Parameters are stored in the job communication area according to specifications defined in the appropriate operating system reference manual.

6. The field length is set to the amount needed for execution, according to the algorithm illustrated in figure 1-3 in section 1.

7. Any unused memory is preset if so requested by an LDSET option in the loader sequence. If presetting is not specified, it is controlled by the installation default.

8. Execution begins at an entry point determined as follows:

   If the EXECUTE statement specifies an entry point, execution begins at that entry point. If the loader is unable to locate the specified entry point, a fatal error results.

   If the EXECUTE statement does not specify an entry point, execution begins at the last transfer address encountered in the object program processed for a basic load or the first entry point on an END segment directive. In this case, at least one such symbol must have been encountered. If the loader is unable to locate the entry point, a fatal error results.

At the time the EXECUTE statement is processed, at least one object program must have been loaded; otherwise, a fatal error results. This means that at least one request for loading must precede the EXECUTE statement.

For an absolute program, processing of EXECUTE involves steps 5 through 8 as previously described for relocatable program processing.

Examples of the EXECUTE statement are shown in figure 2-9.

```
Example 1:

    LOAD(RED)
    EXECUTE(ALTERN,INPUT,OUTPUT)

Example 2:

    LOAD(GREEN)
    EXECUTE(,INPUT,OUTPUT)

Example 3:

    LOAD(NUTS,BOLTS)
    EXECUTE(START,A7,25)
```

Figure 2-9. Examples of the EXECUTE Statement

In example 1, the pair of LOAD and EXECUTE statements causes execution to begin at entry point ALTERN with execution parameters INPUT and OUTPUT. In example 2, the pair of LOAD and EXECUTE statements causes execution to begin at the last encountered transfer symbol. A transfer symbol is required for execution. The sequence in example 3 causes files NUTS and BOLTS to be loaded, and also passes parameters A7 and 25 to the loaded program. Execution begins at the entry point named START.

NOGO STATEMENT

The NOGO statement (figure 2-10) causes completion of the load, but program execution is inhibited.

```
NOGO.

NOGO(lfn)

NOGO(lfn,eptname1,...,eptnamen)
```

lfn	The name of the file on which the absolute overlay is to be written; cannot be used for segmentation.
$eptname_i$	The names of entry points to be included in the overlay header as program entry points. Names are ignored if either capsules are being generated or OVERLAY directives control overlay generation.

Figure 2-10. NOGO Statement Formats

Completion of a load using a NOGO statement is similar to steps 1 through 4 of the EXECUTE statement. It differs from the EXECUTE statement in the following ways:

- Execution of the loaded program does not take place.
- For relocatable loads, the memory image is optionally written and saved as a single absolute program. This occurs if the NOGO statement specifies a file on which the memory image is to be written. One or more entry point names can also be specified for inclusion in the program.
- For a segmented load, no file is specified because the output file for segments is specified on the SEGLOAD statement. The only allowable form is NOGO.
- For overlay generation, the output file for overlays can be specified either on the NOGO card or on the overlay directives. If both are present, the file name on the NOGO card is used. If neither is present, the file name ABS is used.
- For capsule generation, the output file can be specified on the NOGO card; if not, the file name ABS is used.
- A DMP following a NOGO does not produce a dump of the loaded program (as it did with some earlier loaders), because the loader does not build the image of the program in place.

Each absolute overlay written to a file by NOGO is a binary record with no end-of-file marker.

Entry points in addition to those specified on the NOGO statement can be included in the header; entry points in the LDSET EPT table encountered in the load file are added to the header. See section 6 for details.

Examples of the NOGO statement are shown in figure 2-11.

```
        Example 1:

            LOAD(TWICE)
            NOGO(OVER)
            OVER.

        Example 2:

            LOAD(FREIGHT)
            NOGO(OVER,ALTA,ALTB,ALTC)
            LOAD(OVER)
            EXECUTE(ALTB)
```

Figure 2-11. Examples of the NOGO Statement

In example 1, the load sequence causes the object program on file TWICE to be loaded and written out as an absolute program. Then, the absolute program is loaded and executed through the name call statement OVER. The sequence shown in example 2 causes execution to begin at ALTB. The program saved on file OVER can be loaded subsequently with execution beginning at any of the entry points: ALTA, ALTB, or ALTC.

SATISFY STATEMENT

The SATISFY statement (figure 2-12) provides for satisfaction of unsatisfied externals before normal satisfaction at load completion The statement also provides satisfaction of externals from user-specified libraries. The search takes place when the request is processed.

```
SATISFY.

SATISFY(libname1,...,libnamen)


libnamei    Name of a system or user library.
            If the libname is unknown, a non-
            fatal error occurs.
```

Figure 2-12. SATISFY Statement Format

SATISFY with no parameters causes the current library set to be searched in a circular fashion until no more externals can be satisfied. When libraries are specified, they apply to currently unsatisfied externals only. Each library is searched once, in the order specified.

SATISFY cannot be used with segmented load sequences. During overlay generation, a SATISFY applies to only one overlay, the one begun by the last OVERLAY directive read from the last or only file named by the most recent LOAD statement.

Note that map information, as it pertains to the entire load, might be incomplete if the map selection is changed during the load sequence. Specifically, if the default MAP option for the job is null (no MAP), and SATISFY precedes LDSET,MAP=BSEX..., certain information is lost from the LOAD MAP. These omissions are:

- Relocatable module descriptions on the block map -DATE, PROCSSR, VER, LEVEL, HARDWARE, COMMENTS.
- List of weak and unsatisfied external references on the entry point map.
- Cross references in the entry point map.

Examples of the SATISFY statement are shown in figure 2-13.

```
        Example 1:

            LOAD(BEFORE)
            SATISFY(MINE,YOURS)
            LGO.

        Example 2:

            LOAD(MOONDOG)
            SATISFY(LIB1,LIB2,LIB3)
            UTOPIA.

        Example 3:

            LIBRARY(XYZ)
            LOAD(AREOFIL)
            SATISFY(ABCD)
            EXECUTE.
```

Figure 2-13. Examples of the SATISFY Statement

In example 1, after loading BEFORE but before loading LGO, the loader searches libraries MINE and then YOURS for externals. After loading from LGO, the loader searches the currently defined library set during load completion.

In example 2, file MOONDOG is loaded. As many externals as possible are satisfied from LIB1. If unsatisfied externals remain, LIB2 is searched for externals, and finally LIB3 is searched. UTOPIA is then loaded and its externals are satisfied from the library set, not from the libraries given on the SATISFY statement.

The sequence shown in example 3 illustrates how a user library is used for satisfying externals. The user library ABCD is searched before XYZ. If, while satisfying externals for AREOFIL, another unsatisfied external (contained in both XYZ and ABCD) is encountered, the external is satisfied from ABCD.

SEGLOAD STATEMENT

The SEGLOAD control statement specifies that segmentation is to take place during the loading process. The SEGLOAD statement is discussed in section 7.

LDSET STATEMENT

The LDSET statement (figure 2-14) provides user control of a variety of load operations. Options specified through LDSET apply for the current load sequence only. A loader completion statement (name call, EXECUTE, or NOGO) terminates the effects of the LDSET options. Each LDSET statement can be used to set several options, or several LDSET statements can be used.

LDSET(option_1,...,option_n)

Each option is specified in one of the following forms:

key
key=param
key=param_1/.../param_n

Figure 2-14. LDSET Statement Format

The LDSET options are explained in table 2-1. Unless otherwise noted, if either a key is repeated in the table or a second LDSET in the sequence resets an option, the loader uses the most recently encountered setting. The loader ignores LDSET options other than LIB, PRESET, PRESETA, ERR, REWIND, and NOREWIND when loading an absolute program. Installation-defined parameters are used as defaults for any of the LDSET options not specified. The following paragraphs explain examples of some of the LDSET options.

Examples of the LIB option are shown in figure 2-15.

```
        Example 1:

            LOAD(A)
            LDSET(LIB=LIB1/LIB2)
            LOAD(LGO)
            LDSET(LIB=LIB3)
            EXECUTE.

        Example 2:

            LOAD(LGO)
            SATISFY.
            LOAD(A)
            LDSET(LIB)
            SATISFY(USER)
            LOADGO.
```

Figure 2-15. Examples of the LIB Option of the LDSET Statement

In example 1, file LGO contains an internal LDSET table with a LIB request to add a system library called RUNSON. Libraries are searched in the following order during load completion:

1. Libraries specified by the global library set
2. LIB1
3. LIB2
4. RUNSON
5. LIB3
6. SYSLIB

TABLE 2-1. LDSET OPTIONS

Key	Parameters	Description
LIB	$libname_1$/ .../$libname_n$	This form of the LIB option specifies one or more libraries comprising the local library set. The loader searches for unsatisfied externals from the local library set if unsatisfied externals remain after searching all libraries in the global library set. Each use of LIB causes the specified names to be added to the end of the local library set previously defined, if not already in the list. Any name already in the global or local library set is ignored.
LIB		LIB with no parameters causes the local library set to be cleared. It causes externals to be processed as if no LIB request were encountered during the current load. NOTE The global library set is altered by use of the LIBRARY control statement (see section 3).
MAP	p/lfn or /lfn or p	This option controls the generation of the load map. lfn Specifies the file to receive the map. The default is OUTPUT. The file is not rewound, either before or after the map is written. p Specifies map contents: p omitted Current job default, as set by the last MAP statement or by installation default N No map S Statistics B Block map E Entry point map X Entry point cross-references Any of the options S, B, E, and X can be combined by concatenation; for example, LDSET(MAP=SB).
PRESET and PRESETA	p p	The PRESET and PRESETA options specify the values to which unused memory is set before execution of the load program. The loader presets memory at various times during loading, always using the most recent preset selection. Memory is preset as acquired by the loader, not at the start of each program block. For this reason a new preset directive does not have an immediate effect during a load. For PRESETA, the lower 17 bits (central memory) or lower 24 bits (ECS) of each word contain its address. For example, if PRESETA=ONES were specified with locations RA+1000_8 and RA+1001_8 unused, they would be set to: 7777 7777 7777 7740 1000 and 7777 7777 7777 7740 1001

TABLE 2-1. LDSET OPTIONS (Contd)

Key	Parameters	Description
		Under NOS/BE, p can be an octal number of 1 to 20 digits, optionally prefixed by + or - and optionally suffixed by the letter B. Under all operating systems, p can be one of the following keywords: p — Octal Preset Value NONE — No presetting for ECS; same as ZERO for CM ZERO — 0000 0000 0000 0000 0000 ONES — 7777 7777 7777 7777 7777 INDEF — 1777 0000 0000 0000 0000 INF — 3777 0000 0000 0000 0000 NGINDEF — 6000 0000 0000 0000 0000 NGINF — 4000 0000 0000 0000 0000 ALTZERO — 2525 2525 2525 2525 2525 ALTONES — 5252 5252 5252 5252 5252 DEBUG — 6000 0000 0004 0040 0000
ERR	p	The ERR option selects one of three methods of handling loader errors. If no ERR option is specified, the installation default is used. Catastrophic errors always result in job abortion. There are also informative errors, which never result in job abortions. p — Significance ALL — The program is aborted for the following types of errors: Fatal Nonfatal Catastrophic FATAL — The program is aborted for the following types of errors: Fatal Catastrophic NONE — Only catastrophic errors cause job abortion. For any other type of error, processing continues if possible.
REWIND and NOREWIN		The REWIND and NOREWIN options alter the default option for rewind of files by LOAD and SLOAD statements. The selection of /R and /NR on the statements takes precedence over these options. These options, however, do not alter the action of a name call statement specifying a file name. Such a file is always rewound.
USEP	pname_1/ .../pname_n	The USEP option causes the indicated object programs to be loaded whether or not they are needed to satisfy external references. The loader loads the programs on the next occasion that it satisfies externals, either as a result of a SATISFY statement or as a result of a load completion statement. If the loader is unable to find an object program name in the libraries searched, it flags a nonfatal error. During overlay generation, a USEP applies to only one overlay, the one begun by the last OVERLAY directive read from the last or only file named in the most recent LOAD directive. If the USEP occurs before the first LOAD directive, it applies to the very first overlay.

TABLE 2-1. LDSET OPTIONS (Contd)

Key	Parameters	Description
USE	$eptname_1$/ .../$eptname_n$	The USE option forces the loading of object programs to assure that specified entry points are included in the load. The loader loads the programs on the next occasion that it satisfies externals, either as a result of a SATISFY statement or as the result of a load completion statement. If the loader is unable to find an entry point name in the libraries searched, it flags a nonfatal error. During overlay generation, a USE applies to only one overlay, the one begun by the last OVERLAY directive read from the last or only file named in the most recent LOAD directive. If the USE occurs before the first LOAD directive, it applies to the very first overlay.
SUBST	$pair_1$/... /$pair_n$	$pair_i$ Pair of entry point names in the form: $eptname_1$-$eptname_2$ The SUBST option changes external references to entry point names to other entry point names. This feature can be used to cause loading of object programs other than those that would normally be loaded. As a result of SUBST, a reference to external $eptname_1$ becomes a reference to external $eptname_2$.
OMIT	$eptname_1$/ .../$eptname_n$	The OMIT option directs that the specified entry point names are to remain unsatisfied, whether or not the program containing these entry point names is loaded. The specified entry point names are processed the same as other unsatisfied names but do not result in errors. Some programs containing these entry point names can be loaded to satisfy other externals, but the specified entry points are not linked. An OMIT request takes effect from the time encountered until the end-of-load or until superseded by a USE; for example, OMIT(XYZ) later followed by USE(XYZ).
FILES or STAT	lfn_1/ .../lfn_n	The FILES/STAT option permits CYBER Record Manager to ensure that library programs are loaded for the processing of specified files. FILES/STAT is treated as a no-op in segment generation. In overlay generation, its application is the same as in USE.
EPT and NOEPT	$eptname_1$/ .../$eptname_n$	The EPT and NOEPT options provide control over the entry points of capsules, overlays, and OVCAPs. (See sections 6 and 8.)
PD	p	The PD option provides control over the print density of the load map. Valid densities are 6 and 8 lines per inch. The density cannot be changed once printing has started. If an invalid option is specified, it is ignored; the job default is used instead.
PS	p	The PS option provides control over the page size of the load map. The map is printed at p lines per page. p must be at least 10 and at most 1000000. The page size cannot be changed once printing has started. If an invalid option is specified, it is ignored; the job default is used instead.
COMMON	$lcbname_1$/ .../$lcbname_n$	The named labeled common blocks are moved to the nearest common ancestor of all segments that reference them. Applies to SEGLOAD only. (See section 7.)
COMMON		COMMON with no parameters specified causes all labeled common blocks to be moved to the nearest common ancestor of all segments that reference them.

In example 2, the global library set is empty. In addition, file LGO has an internal LIB request to select a system library called SMALGOL. File LGO is loaded. The SATISFY statement specifies that the currently defined library set (SMALGOL) is searched for externals, and then SYSLIB is searched. File A is loaded. As a result of the LDSET(LIB) request, the local library set is emptied. The library called USER is searched for externals. File LOADGO is loaded, load completion is performed, and execution begins. Because the library set is empty, only SYSLIB is available to satisfy any unsatisfied externals introduced by the loading of LOADGO.

Figure 2-16 shows an example of the NOREWIND option of the LDSET statement. The example assumes the installation default is set for rewind. A is rewound and loaded; B is loaded without rewinding. C, D, E, and F are loaded without rewinding; G is rewound and loaded. H is rewound, loaded, and executed. (A file called by name is always rewound.) J is rewound, loaded, and executed. For a new load, the original default applies.

```
LOAD(A,B/NR)
LDSET(NOREWIN)
LOAD(C,D,E,F,G/R)
H.
LOAD(J)
EXECUTE.
```

Figure 2-16. Example of the NOREWIND Option of the LDSET Statement

Figure 2-17 shows an example of the MAP option of the LDSET statement. The example assumes the map default option is off. No map is produced for FILE1 and FILE3. For FILE2, a block map is produced. For FILE4, statistics, blocks, entry points, and cross-references are listed; however, only referenced entry points are listed because MAP(ON) selects the X option but not the E option.

```
FILE1.
  .
  .
  .
LDSET(MAP=B)
FILE2.
  .
  .
  .
FILE3.
  .
  .
  .
MAP(ON)
FILE4.
  .
  .
  .
LDSET(MAP=/MAPFILE)
FILE6.
```

Figure 2-17. Example of the MAP Option of the LDSET Statement

For FILE6, the same items are listed on the map as in the load for FILE4, but the map is written on MAPFILE instead of OUTPUT (the default file).

An example of the USE option of the LDSET statement is shown in figure 2-18. In the example, none of the entry points has been encounterd during the load. Library program NED contains the entry points ENT1 and ENT2. Library program RED contains the entry point ENT3. Programs NED and RED are loaded when encountered during library satisfying. Under NOS/BE, the loading occurs even if a program named NED or RED has been loaded from a load file, but a nonfatal error is noted. Under NOS, duplicate program names are skipped with no message. A nonfatal error is also noted if ENT1, ENT2, or ENT3 cannot be found in any library programs.

```
LDSET(USE=ENT1/ENT2/ENT3)
```

Figure 2-18. Example of the USE Option of the LDSET Statement

Examples of the SUBST option of the LDSET statement are shown in figure 2-19. In example 1, any reference to MEAT is treated as a reference to BEANS, and any reference to either CORN or PEAS is treated as a reference to RICE.

```
Example 1:

    LDSET(SUBST=MEAT-BEANS)
    LDSET(SUBST=CORN-RICE)
    LDSET(SUBST=PEAS-RICE)
    LGO.

Example 2:

    LDSET(SUBST=TAN-SEC/SEC-TAN)
    LDSET(SUBST=SIN-CSC)
    LDSET(SUBST=CSC-SIN)
    LDSET(SUBST=A-B/A-C)
    LOAD(FILE1)
    LDSET(SUBST=A-D)
    FILE2.
```

Figure 2-19. Examples of the SUBST Option of the LDSET Statement

Example 2 illustrates how conflicts with use of SUBST are resolved. So that the most recent SUBST requests can take precedence in the case of a conflict, the entry pairs given in a request are stored at the front of the substitution table in the order they occur. For example 2, they are stored as follows:

```
A-D        (Most recent SUBST)
A-C
A-B
CSC-SIN
SIN-CSC
SEC-TAN
TAN-SEC    (First SUBST)
```

During the load of FILE1, any reference to A is treated as a reference to C. This shows that the A-B specification is meaningless. During the loading of FILE2, references to A are treated as references to D. The first three statements do not cause any conflicts. The processing of substitutions is not iterative. Hence, all the original references to CSC become references to SIN, while all the original references to SIN become references to CSC, and so forth.

Figure 2-20 shows examples of the OMIT option of the LDSET statement. In the sequence in example 1, the linking of external references to TAN is not inhibited during the loading of FILE1; however, any references to TAN loaded subsequently to the OMIT request remain unsatisfied.

```
Example 1:

    LOAD(FILE1)
    LDSET(OMIT=TAN)
    FILE2.

Example 2:

    LOAD(FILE1)
    LDSET(OMIT=TAN/SEC)
    FILE2.

Example 2:

    LDSET(SUBST=A-B,OMIT=B)
    LGO.

Example 3:

    LDSET(SUBST=A-B/C-A,OMIT=A)
    LGO.
```

Figure 2-20. Examples of the OMIT Option of the LDSET Statement

Example 2 assumes that entry point TAN exists on FILE1 and entry point SEC exists on FILE2. External references to TAN from programs on FILE1 are satisfied when FILE1 is loaded, while external references to TAN from programs on FILE2 remain unsatisfied. All external references to SEC remain unsatisfied.

In example 3, original external references to A are treated as external references to B. All external references to B, including the original external references to A, remain unsatisfied.

In example 4, original external references to A are treated as external references to B and original external references to C are treated as external references to A. At load completion, original external references to A are satisfied (by B) and original external references to C remain unsatisfied.

INTERNAL LDSET SPECIFICATIONS

LDSET information can be provided to the loader either by control statements or by LDSET (7000) tables in relocatable binaries. Most compilers include LDSET tables with their binary output; COMPASS programmers can use the LDSET pseudo instruction to generate an LDSET table.

A COMPASS program can contain any number of LDSET instructions. COMPASS collects all LDSET options and writes a single LDSET (7000) table in the relocatable binary output between the PRFX (7700) and PIDL (3400) tables. If there are no LDSET instructions, no LDSET table is written.

The format of the COMPASS pseudo instruction that is used to generate an LDSET table is shown in figure 2-21.

Location	Operation	Variable Subfields
	LDSET	$\text{option}_1,\ldots,\text{option}_n$

Option:

Each option is specified in one of the following forms:

keyword
keyword=parameter
$\text{keyword=parameter}_1/\ldots/\text{parameter}_n$

The keywords and options are the same as those described for the LDSET control statement.

Figure 2-21. COMPASS LDSET Pseudo Instruction Format

An example of the LDSET pseudo instruction is shown in figure 2-22. In the example, the LDSET option ERR is set to NONE. A fatal error occurs when the instruction MACHINE CMU is encountered. Processing continues if possible.

```
IDENT
ENTRY      A
LDSET      ERR=NONE
MACHINE    CMU
   .
   .
   .
END        A
```

Figure 2-22. Example of the LDSET Pseudo Instruction

The internal format of the LDSET table is included in appendix D.

INTERACTIVE USE

Interactive use is possible only under the batch subsystem of NOS, where the entry of a loader control statement begins a dialogue with the loader as follows:

```
/load,file1.
LDR>?libload,xxx,a.
LDR>?execute.
```

Each time the loader receives a loader control statement that is not a load sequence terminator, it responds with a prompt requesting the next statement of the load sequence. When a load sequence terminator is received, the loader stops prompting the user and performs the load.

Interactive use can be simulated on both NOS and NOS/BE by the use of a procedure file. For NOS/BE users, a CCL interactive procedure could be specified as shown in figure 2-23.

CYBER Control Language is described in the appropriate operating system reference manual.

```
CONNECT,PROCFIL.
BEGIN,SAM.
.PROC,SAM.
    .
    .
    .
%EOF
```

Figure 2-23. NOS/BE CCL Interactive Procedure File

LOADER-RELATED CONTROL STATEMENTS

Control statements discussed in this section are processed by the operating system rather than by the loader. They are described here because they affect loader processing.

MAP STATEMENT

The MAP statement (figure 3-1) specifies the default option for load maps for load sequences requested by the job. The option selected remains in effect either throughout the job or until changed by another MAP statement. If no MAP statement is used with a job, the loader uses a system-defined default.

```
MAP.

MAP(p)
```

p A parameter that specifies the type of map:

OFF	No map	Corresponds to: LDSET(MAP=N)
PART	Statistics, block map	Corresponds to: LDSET(MAP=SB)
ON	Statistics, block map, entry point cross-reference	Corresponds to: LDSET(MAP=SBX)
FULL	Statistics, block map, entry point map, entry point cross-reference	Corresponds to: LDSET(MAP=SBEX)

Specifying the MAP statement with no parameter resets the load map option back to the installation default (IP.MAP).

Figure 3-1. MAP Control Statement Formats

The effects of MAP can be overridden on a temporary basis for specific loader request sequences through either the MAP parameter on the LDSET statement (section 2), or the MAP option with the LDREQ macro (section 4). For compatibility with previous versions of the operating system, MAP can occur within a loader control statement sequence. Loader maps are illustrated in appendix E.

LIBRARY STATEMENT

The LIBRARY statement (figure 3-2) specifies a set of global libraries that are to be searched for externals and name call statements, and the order in which the libraries are to be considered. Weak externals are ignored.

```
LIBRARY.

LIBRARY(libname1,...,libnamen)

LIBRARY(libname1,...,libnamen/p)
```

$libname_i$	The name of the library. If a local file and system library have the same name, the local file takes precedence (except for the NOS/BE library NUCLEUS).
	Specifying the LIBRARY statement with no parameters clears the global library set. The maximum number of libraries allowed is:
	24 system libraries, and 0 user libraries
	13 system libraries, and 1 user library
	2 system libraries, and 2 user libraries
p	Specifies one of the following options:
	A Add the specified library or libraries to the global library set.
	D Delete the specified library or libraries from the global library set.
	R Replace the global library set with the library set specified on the control statement.
	The default is R.

Figure 3-2. LIBRARY Statement Formats

Under NOS, libraries are built by the LIBGEN utility and can contain relocatable, absolute, capsule, OVCAP, overlay programs, and CCL procedures. Relocatable programs can be loaded either to satisfy externals or in response to a name call statement. Additionally, a name call statement can load an absolute program or a CCL procedure. Overlays must be referenced by overlay name and not by level number.

Under NOS/BE, libraries are built by the EDITLIB utility and can contain both relocatable and absolute programs (except for segmented programs). Relocatable programs can be loaded either to satisfy externals or in response to a name call statement. Additionally, a name call statement can load an absolute program.

LIBGEN is described in the NOS reference manual; EDITLIB is described in the NOS/BE reference manual.

Because it is not a loader control statement, LIBRARY cannot occur within a loader control statement sequence. Libraries introduced through the LIBRARY statement are global; that is, the statement applies from the time it is encountered until the end of the job or until another LIBRARY statement is encountered. If the user desires that a set of libraries be used for only a single load sequence, the LIB parameter on either the LDSET loader statement (section 2) or the LDREQ macro (section 4) should be used.

The order of search for externals is:

1. The global library set specified by LIBRARY statements

2. The local library set defined on LDSET(LIB=...) directives

3. SYSLIB (not searched by default for capsule or OVCAP generation)

The order of search for name call statements is:

1. The list of local files

2. The global library set

3. The local library set

4. NUCLEUS library (NOS/BE only)

Examples of the LIBRARY statement are shown in figure 3-3. In example 1, the library set consists of only SYSLIB; therefore, only SYSLIB is searched for externals unless other libraries specified by the object directives on file A are generated by the compiler.

In example 2, file LGO1 is loaded, the library set is empty except for the default system library, SYSLIB. Therefore, SYSLIB is the only library searched to satisfy externals for file LGO1.

The LIBRARY(FORTRAN) control statement specifies that the global library set is to consist of FORTRAN. The LDSET(LIB=USER) loader statement specifies that the current sequence will use a local library named USER. Thus, when file LGO2 is loaded, libraries are searched in the following order to satisfy externals: FORTRAN, USER, and SYSLIB.

When file LGO3 is loaded, only FORTRAN and SYSLIB are searched to satisfy externals. (USER no longer exists because it was local to the load sequence completed by LGO2. FORTRAN is still in the global library set because no new LIBRARY control statement was encountered.)

```
Example 1:

    LIBRARY.
    LOAD(A)
    EXECUTE.


Example 2:

    LGO1.
    LIBRARY(FORTRAN)
    LDSET(LIB=USER)
    LGO2.
    LGO3.
    LIBRARY.
    LGO4.
    LIBRARY(AX,USER2)
    LDSET(LIB=ALGOL)
    LGO5.
```

Figure 3-3. Examples of the LIBRARY Statement

The LIBRARY control statement creates an empty library set except for the default system library. Therefore, when LGO4 is loaded, only SYSLIB is searched to satisfy externals.

The control statement LIBRARY(AX,USER2) specifies that the global library set consists of AX and USER2. The LDSET(LIB=ALGOL) loader statement specifies a local library named ALGOL. When file LGO5 is loaded, libraries are searched in the following sequence to satisfy externals: AX, USER2, ALGOL, and SYSLIB.

Under NOS, the no-auto-drop status is set for any file inserted in the global library set and cleared when a library is removed from the global library set.

REDUCE STATEMENT

The REDUCE statement (figure 3-4) specifies whether or not the loader is to determine field length assignment. When the loader determines field length assignment, the job is said to be in REDUCE mode.

```
REDUCE.          NOS/BE batch and NOS only

REDUCE(ON)       NOS/BE Intercom only

REDUCE(-)        NOS only

REDUCE(OFF)      NOS/BE Intercom only

REDUCE.       Specifies that the loader assigns
REDUCE(ON)    field length.

REDUCE(-)     Inhibits reduction of the field
REDUCE(OFF)   length by the loader.
```

Figure 3-4. REDUCE Statement Formats

Examples of the use of the REDUCE control statement are shown under the discussion of the RFL control statement. See section 1 for a detailed discussion of field length control.

RFL STATEMENT

The RFL statement (figure 3-5) controls the amount of field length that is used to execute programs.

```
RFL(n)

RFL(n,m)

RFL(CM=n,EC=m)
```

n	The new central memory field length in octal. The maximum value for NOS is 377777_8. The maximum value for NOS/BE is set by installation option.
m	The new ECS field length in octal. The maximum value for NOS is 7777_8. The maximum value for NOS/BE is set by installation option.

The value specified for n or m cannot exceed the value specified on the last MFL statement nor can it exceed the maximum allowed for the job.

Figure 3-5. RFL Control Statement Formats

The RFL statement remains in effect until another RFL statement is executed. An installation parameter defines the maximum field length that can be requested. User validation limits can further restrict the maximum field length.

For NOS/BE, the RFL statement sets REDUCE mode to off. See section 1 for a detailed discussion of field length control.

Figure 3-6 shows the use of the RFL and REDUCE statements. The job starts off in REDUCE mode. The program is compiled, loaded, and executed in the minimum required field length. The field length is then set to 30000_8 words, and the job is loaded and executed a second time, using 30000_8 words of central memory. The job is then returned to REDUCE mode. If the job is run under NOS/BE 1, the REDUCE(-) statement should not be included because the RFL card inhibits field length reduction automatically and the REDUCE(-) is considered an error statement.

```
FTN5.
LGO.
RFL(30000)
REDUCE(-)
LGO.
REDUCE.
```

Figure 3-6. Example of the RFL and REDUCE Statements

USER REQUESTS 4

This section describes how the user can call the user call loader during program execution, how to format request tables, and how the loader processes the loader call. (The portion of the loader that handles user requests is sometimes referred to as the user call loader.) Processing of a loader request table differs from processing of control statements in one major respect; that is, externals are not automatically satisfied during completion of a loader request sequence.

Request tables are used for relocatable loads only. Also, debugging aids, overlay generation, segmentation, and capsule generation cannot be requested during this type of load.

The COMPASS language user can either format requests and issue loader calls directly, or take advantage of the loader macros provided with the operating systems to facilitate loader use. The LOADER and LDREQ macros are available through macro definitions supplied on the system text overlay LDRTEXT. The LOADREQ macro is available through macro definitions supplied on the system text overlay CPUTEXT. The source of system macro definitions can be changed through the S or G option on the COMPASS control statement. (See the COMPASS reference manual for further details.)

CALLING THE LOADER

A program calls the loader either through the COMPASS LOADER macro or through an expanded form of the macro call. The LOADER macro call loads relocatable programs; the LOADREQ macro call (section 6) should be used to load absolute overlays. The formats of the COMPASS LOADER calls are shown in figure 4-1. In both calls shown in the figure, paddr is required and points to the first word address of a table of formatted loader requests. This is the symbol in the location field of an LDREQ BEGIN call.

The loader does not normally perform automatic field length management during user call loads; the user should first obtain sufficient memory through use of the MEMORY macro or via an RFL control statement. (The RFL statement should be followed by a REDUCE(-) statement under NOS.) However, if Common Memory Manager (CMM) is specified on the LOADER call, and fwasc and lwasc are absent from the LDREQ BEGIN, memory is allocated automatically through CMM. (See the Common Memory Manager reference manual for more details on this feature.)

External references in programs loaded by a user call can be satisfied by entry points loaded in earlier user call loads or the original basic load; the loader saves linkage information on a scratch file. An entry point is not available, however, if it is within the loadable area of the current load because it can be overwritten.

Location	Operation	Variable Subfields
	LOADER	paddr
	LOADER	paddr,CMM

paddr	The address of LDREQ BEGIN.
CMM	Indicates that the Common Memory Manager (CMM) is to be used during loading. It must be specified if CMM is active at the time of the call.

The first format causes the following COMPASS code to be generated:

Location	Operation	Variable Subfields
	RJ	=XLOADER=
	VFD	42/0,18/paddr

The second format causes the following COMPASS code to be generated:

Location	Operation	Variable Subfields
+	RJ	=XLOADER.
-	VFD	12/0,18/paddr

Figure 4-1. COMPASS LOADER Macro Call Formats

Entry points are also not automatically available if the program is segmented or overlayed because the scratch file is not available. If the entry points are needed, their addresses can be made available to the loader through use of a PASSLOC request. Entry points loaded by a user call will not satisfy externals that were unsatisfied on a previous load.

The loadable area either is specified in the request table or is assumed, by default, to be the remaining unloaded area available to the job. Loading can be specified so as to overwrite the previously loaded area, but this must be done with care. For this purpose, words RA+65_8 and RA+66_8 in the job communication area give the default central memory and ECS load limits. Upon completion of the user call, the loader updates these fields to reflect the new limits unless the CMM parameter is specified.

Unlike control statement sequences, internal requests need not be terminated by EXECUTE or NOGO. The way the loader processes requests specified in the table and performs load completion depends on whether or not the load sequence is terminated by EXECUTE.

For a load sequence not terminated by EXECUTE:

1. The loader is initialized and performs requests specified in the load sequence.

2. The memory image is generated in the loadable area.

3. Execution of the calling program continues.

For a load sequence terminated by EXECUTE:

1. The loader is initialized and performs requests specified in the load sequence.

2. The memory image is generated in the loadable area.

3. The job communication area is set with the parameters passed in the EXECUTE request.

4. Execution of the loaded program is initiated at the specified entry point. If no entry point is specified, execution begins at the last encountered transfer symbol.

REQUEST TABLES

A request table must accompany each loader call. It consists of a header followed by one or more loader requests in internal form. An END request terminates the requests. The table can either be generated directly by the programmer or through a sequence of three or more LDREQ macros. The internal format of the request table is shown in appendix D.

LDREQ MACROS

The following subsections describe the set of LDREQ macros used to facilitate generation of request tables. The LDREQ macros are not contained in the default system text (SYSTEXT) used by COMPASS. A separate text called LDRTEXT is available, which contains the LOADER and LDREQ macros. A COMPASS call specifying that both texts are to be used is as follows:

```
COMPASS (S=SYSTEXT,S=LDRTEXT)
```

The general format of the LDREQ macro is shown in figure 4-2. The LOAD, LIBLOAD, SLOAD, and SATISFY options provide the same capability and have the same parameters as their corresponding loader statements, which are described in section 2. The formats of these options are shown in figure 4-3.

The LIB, MAP, PRESET, PRESETA, USEP, USE, SUBST, OMIT, FILES, PD and PS options provide the same capability and have the same subparameters as their corresponding parameters on the LDSET statement described in section 2. The format of these options is shown in figure 4-4.

Location	Operation	Variable Subfields
symbol	LDREQ	option,p_1,...,p_n

symbol	For the BEGIN option, the symbol is assigned the first word address (paddr) of the request table. For any other option, the symbol is assigned the first word address of the entry in the table.
option	Required; specifies the table entry. The first LDREQ macro in a sequence must have BEGIN as the option. The last LDREQ macro must have END as the option.

The options that can be specified are:

BEGIN	PRESET
END	PRESETA
LOAD	USEP
LIBLOAD	USE
SLOAD	SUBST
CMLOAD	ENTRY
ECLOAD	OMIT
EXECUTE	FILES
NOGO	DMP
SATISFY	PASSLOC
LIB	PS
MAP	PD

p_i	Parameters required by the option selected.

Figure 4-2. General Format of an LDREQ Macro

Each sequence of LDREQ macros must begin with the BEGIN option and end with the END option. The BEGIN option generates the request table header, and the END option generates a terminal zero word. No check is made when the table is assembled, but the loader issues error messages if it is unable to use the table during execution.

BEGIN Option

A sequence of LDREQ calls must start with BEGIN as the option, which causes generation of the loader request table header. The format of the BEGIN option is shown in figure 4-5.

If the CMM parameter is specified on the LOADER macro call and both fwasc and lwasc are not specified, then CMM is used to allocate a default area. The CMM reference manual provides further information about this feature.

If fwasc and lwasc are absent and the CMM parameter is not specified, the loadable area used by default is that between the lwasc+1 in bits 17 through 0 of RA+65_8 and the end of the currently assigned field length. Similarly, if fwalc and lwalc are absent, the lwalc in bits 58 through 36 of RA+65_8 and the currently assigned ECS field length are used as defaults.

Location	Operation	Variable Subfields
	LDREQ	LOAD,(lfn$_1$,...,lfn$_n$)
	.	
	LDREQ	LIBLOAD,libname,(eptname$_1$,...,eptname$_n$)
	.	
	LDREQ	SLOAD,lfn,(name$_1$,...,name$_n$)
	.	
	LDREQ	SATISFY,(libname$_1$,...,libname$_n$)

Figure 4-3. LOAD, LIBLOAD, SLOAD, and SATISFY Option Formats

Location	Operation	Variable Subfields
	LDREQ	LIB,(libname$_1$,...,libname$_n$)
	.	
	LDREQ	LIB
	.	
	LDREQ	MAP,p,lfn
	.	
	LDREQ	PRESET,p
	.	
	LDREQ	PRESETA,p
	.	
	LDREQ	USEP,(pname$_1$,...,pname$_n$)
	.	
	LDREQ	USE,(eptname$_1$,...,eptname$_n$)
	.	
	LDREQ	SUBST,(pair$_1$,...,pair$_n$)
	.	
	LDREQ	OMIT,(eptname$_1$,...,eptname$_n$)
	.	
	LDREQ	FILES,(lfn$_1$,...,lfn$_n$)
	.	
	LDREQ	PD,p
	.	
	LDREQ	PS,p

Figure 4-4. LIB, MAP, PRESET, PRESETA, USEP, USE, SUBST, OMIT, FILES, PD, and PS Options Formats

Location	Operation	Variable Subfields
paddr	LDREQ	BEGIN,fwasc,lwasc,fwalc,lwalc

paddr	First word address of the loader request table.
fwasc	First word address of the central memory loadable area.
lwasc	Ending address+1 of the central memory loadable area. lwasc must be $\leq$ FL-3.
fwalc	First word address of the ECS loadable area.
lwalc	Ending address+1 of the ECS loadable area.

Figure 4-5. BEGIN Option Format

END Option

A sequence of LDREQ calls must terminate with an END option (figure 4-6), which causes generation of a zeroed word at the end of the loader request table.

Location	Operation	Variable Subfields
	LDREQ	END

Figure 4-6. END Option Format

CMLOAD Option

The CMLOAD option (figure 4-7) specifies that load input is to be fetched directly from central memory; it is available only as a user call loader request.

Location	Operation	Variable Subfields
	LDREQ	CMLOAD,fwa,end

fwa	An absolute or relocatable expression, specifying the first word address of load input in central memory.
end	An absolute or relocatable expression, specifying the last word address+1 of load input in central memory.

Figure 4-7. CMLOAD Option Format

Physical loading is performed similar to loading for the LOAD request. Instead of specifying a file name and rewind indicator, the request supplies first word and last word addresses.

The area specified in central memory must be within the job's field length but must be outside of the specified loadable area.

ECLOAD Option

The ECLOAD option (figure 4-8) specifies that load input is to be fetched directly from extended core storage; it is available only as a user call loader request.

Location	Operation	Variable Subfields
	LDREQ	ECLOAD,fwa,end

fwa	An absolute or relocatable expression, specifying the first word address of load input from ECS.
end	An absolute or relocatable expression, specifying the last word address+1 of load input from ECS.

Figure 4-8. ECLOAD Option Format

Physical loading is performed similar to loading for the LOAD request. Instead of specifying a file name and rewind indicator, the request supplies first word and last word addresses in ECS. The area specified in ECS (between fwa and end on the ECLOAD option) must contain object text consistent with the type of load being performed.

The area specified in ECS must be within the job's field length, but must be outside of the specified loadable area.

EXECUTE Option

The EXECUTE option (figure 4-9) has the same parameters and provides similar capability as the EXECUTE statement described in section 2. The EXECUTE option differs from the EXECUTE statement as follows:

- Libraries are not searched automatically.
- Blank common is established only if it is declared in the load for the first time.
- Field length is not altered.

Location	Operation	Variable Subfields
	LDREQ	EXECUTE
	LDREQ	EXECUTE,eptname
	LDREQ	EXECUTE,eptname,(p_1,...,p_n)
	LDREQ	EXECUTE,,(p_1,...,p_n)

Figure 4-9. EXECUTE Option Formats

The SATISFY request is required before an EXECUTE request if libraries are to be searched to satisfy external references.

NOGO Option

The NOGO option (figure 4-10) requests load completion. Control is returned to the calling program, not the newly loaded program. Unlike the NOGO control statement, externals are not satisfied unless a SATISFY request precedes the NOGO request.

Location	Operation	Variable Subfields
	LDREQ	NOGO

Figure 4-10. NOGO Option Format

ENTRY Option

The ENTRY option (figure 4-11) allows an executing program to obtain the addresses of entry points that are currently being loaded and/or have been loaded previously. The loader replaces the entry point names in the table with their addresses.

The ENTRY option is available only as an LDREQ option. A user call that includes an ENTRY request must not specify limits of the loadable area such that the user call request area is overwritten. The updated request table must be available to the program after the user call is completed.

DMP Option

The DMP option (figure 4-12) allows a user to request a dump within a loader sequence. When the loader encounters a DMP request, it issues an RA+1 DMP call to the operating system. The DMP option is intended primarily for the use of system analysts.

PASSLOC Option

The PASSLOC option (figure 4-13) allows an executing program to supply addresses to the loader. It is available as an LDREQ option only.

PASSLOC is not needed for a basic relocatable program because the loader retains such information in its tables during execution; however, for either an overlay, a segmented program, or a memory image that was previously saved, the loader tables are not available during execution. Such an absolute program can use PASSLOC to supply information for the loader to add to its tables.

Location	Operation	Variable Subfields
	LDREQ	ENTRY,(eptname$_1$,...,eptname$_n$)

eptname$_i$ Entry point names.

Figure 4-11. ENTRY Option Format

Location	Operation	Variable Subfields
	LDREQ	DMP,p$_1$,p$_2$

p$_i$	Consult the appropriate operating system reference manual for a description of DMP parameters.

Figure 4-12. DMP Option Format

USER CALL LOADER EXAMPLE

An example of the use of the user call loader is shown in figure 4-14. After file A is loaded, libraries LIB1 and LIB2 are added to the local library set. After LGO is loaded, library LIB3 is added to the local library set. The SATISFY request causes the global library set, then the libraries in the local library set, and finally SYSLIB to be searched for externals. EXECUTE causes load completion and transfers control to the newly loaded program.

LOADREQ (REQUEST BASIC LOAD)

The loader provides an alternate means for running programs to initiate a basic load from an initial state, completely discarding anything loaded at the time. The LOADREQ macro (figure 4-15) specifies that the relocatable loader is to be loaded to perform a function indicated by the contents of the RA communication area.

Before issuing the call, the user should set the contents of RA+64$_8$ to perform one of the following functions.

The first function specifies that the relocatable loader is to load and execute a local file. The function is specified as follows:

RA+64$_8$=42/lfn,18/0

The parameter lfn identifies the local file to be loaded. The local file name must be specified as left-justified and zero filled.

The second function specifies that loading and execution is to be performed as directed by a user-specified table of loader requests. The request table should contain control statements in internal form as generated by the LDREQ macro (see appendix D). Any LDREQ options that have a corresponding control statement can be used. Any

Location	Operation	Variable Subfields
	LDREQ	PASSLOC,((id$_1$,t$_1$,b$_1$,a$_1$),...,(id$_n$,t$_n$,b$_n$,a$_n$))

id_i	The name of the program block, entry point, or common block.
t_i	The type of name:
	0 Entry point 1 Program block } A null id_i is illegal
	2 Central memory common block 3 ECS common block } A null id_i designates a blank common block
b_i	The block length in words; ignored if t is zero.
a_i	The address of the entry point or first word address of the program or common block.

Figure 4-13. PASSLOC Option Format

Location	Operation	Variable Subfields
	LDREQ	BEGIN
	LDREQ	LOAD,(A)
	LDREQ	LIB,(LIB1,LIB2)
	LDREQ	LOAD,(LGO)
	LDREQ	LIB,(LIB3)
	LDREQ	SATISFY
	LDREQ	EXECUTE
	LDREQ	END

Figure 4-14. User Call Loader Example

Location	Operation	Variable Subfields
	LOADREQ	

Figure 4-15. LOADREQ Macro Format

LDREQ options that apply only to the user call loader should not be used. The function is specified as follows:

RA+64_8=12/0,18/length,12/0,18/fwa

Loading begins at fwa and continues to length. The request table is restricted such that RA+2 is less than or equal to fwa, and fwa+length is less than or equal to RA+54_8.

If an explicit EXECUTE or NOGO is not specified, a NOGO is assumed. There is no internal form for SEGLOAD; thus, only overlay and relocatable loads are permitted. The calling program is responsible for ensuring the validity of the request table because the loader does not check the table for errors.

The calling program is overwritten by loading of the relocatable loader; therefore, control cannot be returned to the calling program.

DEBUGGING AIDS

The loader provides the TRAPPER routine as an aid to execution time debugging. The TRAPPER routine is requested by the TRAP control statement and is the first program loaded in its applicable load sequence. TRAP can only be used with basic loads; it cannot be used with overlayed programs. CYBER Interactive Debug cannot be used with programs using the TRAPPER routine.

TRAP CONTROL STATEMENT

The TRAP statement (figure 5-1) causes the TRAPPER routine to be loaded with and become applicable to the next load sequence.

```
TRAP(I=lfn1,L=lfn2)
```

lfn_1	The input file that contains the TRAP directives. If omitted, directives comprise the next section on the INPUT file.
lfn_2	The output file on which the listing of directives and the resulting dumps is written. This file should not be a file being written on by the trapped program. If omitted, output is written on the file named TRAPS.

Figure 5-1. TRAP Statement Format

The TRAP control statement is a separate load sequence and affects the next relocatable load sequence. The TRAP statement must, therefore, immediately precede the first statement in the load sequence to which it applies.

TRAP DIRECTIVES

TRAP directives specify output to be written on the TRAP output file. The types of directives are:

- FRAME

  Requests snapshot dumps of registers and areas of central memory at selected locations within the program.

- TRACK

  Provides information to analyze a series of instruction executions in a program. This information consists of the contents of locations and registers after execution of instructions in the specified range.

The syntax for TRAP directives is shown in table 5-1. The format of the directives is shown in figure 5-2.

Label	Verb	Specification
	FRAME or TRACK	$parameter_1 ... parameter_n$

$parameter_i$	See tables 5-2 and 5-4 for parameters that can be used with FRAME and TRACK directives.

Figure 5-2. TRAP Directive Format

FRAME DIRECTIVE PARAMETERS

Parameters for the FRAME directive are summarized in tables 5-2 and 5-3. Any parameter other than AT is optional and has a default value. AT is required; it must be used in all FRAME directives. Rules for use of the FRAME directive are as follows:

- The first phrase encountered in a directive (for example, PROGRAM, PROG, BLOCK, ABS) becomes the default until another one is encountered.
- If specified, a parameter must be complete; that is, keyword, phrase (if applicable), and value must be present. If no parameter is specified, the item listed in the default column is substituted.
- Multiple specification of any given parameter (for example, FROM) causes FRAME to ignore all but the last specification of that parameter.
- The octal address fields are limited to ten digits, including leading zeros. Also, they are limited to a maximum value of 131,071. Otherwise, the fatal diagnostic "ILLEGAL NUMBER ON ABOVE TRAP" occurs. (See appendix B).

FRAME output consists of a dump of all the registers (if requested; if not, only the contents of the P register appears in the dump), and a central memory dump of the area specified in the directives. Both the octal and display code representations of the area are included in the central memory dump.

The beginning address of a dump is the closest multiple of four that is less than or equal to the requested fwa. The ending address plus one is the FL or the closest multiple of four that is greater than the lwa, whichever is less. Any n line(s) that contains four times n identical words is compressed into one line indicating the range.

TABLE 5-1. TRAP DIRECTIVE SYNTAX

Specification	Meaning
Comment	When column 1 of a line contains an asterisk, the line is considered a comment statement. It is copied to the output file, but has no effect on the type of output produced during execution.
Page eject	A slash in column 1 causes a page eject in the listing of the TRAP directives. The line itself is not listed with the directives. Continuations are not allowed.
Label field	The label field of the TRAP directive is an identifier of one to seven characters that starts in column 1. It is terminated by one or more blanks, a comma, hyphen, left parenthesis, or right parenthesis. A label can contain any character not mentioned as a terminator. A longer label is truncated to seven characters but does not generate an error. A blank in column 1 indicates an empty label field. The label, if present, is printed with any output related to the directive.
Verb	The verb field begins with the first nonblank character following the label field. If the label field is empty, this field begins with the first nonblank field of the line. The terminators for the verb field are comma, hyphen, right parenthesis, left parenthesis, or one or more blank characters. The verb can be FRAME or TRACK.
Specification field	The specification field begins with the first nonblank character following the verb field. This field consists of a list of parameters separated by commas or blanks. A parameter consists of a keyword and its associated phrase and value (if these are applicable). A phrase is a keyword and value that have meaning only when associated with another keyword. Multiple delimiters are permitted between parameters. Parameters can appear in the specification field in any order.
Continuation	A comma in column 1 of a line indicates continuation of the previous line. The rest of the line is treated as if column 2 immediately followed column 72 of the preceding card image. Any number of continuations is allowed. Any line without a comma in column 1 terminates the directive or comment statement being defined by the preceding line. A keyword or value cannot be split between lines.

Figure 5-3 shows examples of the FRAME directive.

In example 1, a dump is produced every other time location 20 of program HOOK is executed, starting the 10th time, and ending the 35th time. The dump consists of locations 10_8 through 235_8 of common block B, and is labeled SYM. No register dump is produced.

In example 2, a dump is produced every time location 75 of program TAB is executed. The first 144_8 locations of central memory blank common are dumped, and a full register dump is produced. The dumps are labeled AX.

In example 3, no label is printed with dumps because the label field is empty. Every time location RA+2147_8 is executed, a full register dump and a dump of 28 ECS words, starting at absolute ECS address 100010, is produced.

Example 1

Label	Verb	Specification
SYM	FRAME	AT PROGRAM HOOK 20, FROM
,		BLOCK B 10, FOR 150,
,		START 10, EVERY 2, UNTIL 35

Example 2

Label	Verb	Specification
AX	FRAME	FROM BLOCK // 0 FOR 100
,		AT PROG TAB 75 REG

Example 3

Label	Verb	Specification
	FRAME	AT 2147, FROM ABS ECS
,		100010, FOR 25, REG

Figure 5-3. FRAME Directive Examples

TRACK DIRECTIVE PARAMETERS

Parameters for the TRACK directive are summarized in tables 5-4 and 5-5. When the program reaches a tracking range, as set by the FROM parameter on the TRACK directive, TRAPPER gains control. TRAPPER interprets the code from this point on, picking up

TABLE 5-2. FRAME DIRECTIVE PARAMETERS

Keyword	Optional Phrase†	Parameter	Default (No Parameter Specified)	Description
AT	BLOCK name PROGRAM name PROG name ABS CM (default)	octal CM address octal CM address octal CM address octal CM address	None (Parameter is required.)	Specifies the address of the instruction at which the dump is to be taken.
FROM	BLOCK name PROGRAM name PROG name ABS CM (default) ABS ECS (default)	octal address octal address octal address octal address octal address	0	First word address for snapshot dump.
FOR	None	Decimal number	fl-fwa (used only if the first word address is in central memory). No default for ECS.	Specifies the number of words to be dumped.
START	None	Decimal number	1	Iteration of AT instruction at which the first dump is to be taken.
EVERY	None	Decimal number	1	Interval after which the dump is to be repeated.
UNTIL	None	Decimal number	131071	Last interaction at which the dump is to be taken.
REG	None	None	No register dump.	Presence of REG specifies that the dump is to include the contents of registers.

†If the optional phrase is specified, the name portion of the phrase can be omitted.

TABLE 5-3. PHRASES FOR FRAME PARAMETERS

Phrase		Value Type	Description
Keyword	Value		
PROGRAM or PROG	name	Alphanumeric program block name, one to seven characters.	Name of the program block containing the address specified by AT or FROM.

BLOCK	name	Alphanumeric block name, one to seven characters. For blank common:
		// CM
		//L ECS (FROM only)
			Name of the block containing the address specified by AT or FROM. //L applies to FROM only.

ABS	type	Core memory type: CM ECS	Type of memory containing the absolute address specified by AT or FROM. If type is omitted, CM is assumed.

TABLE 5-4. TRACK DIRECTIVE PARAMETERS

Keyword	Optional Phrase†	Parameter	Default (No Parameter Specified)	Description
FROM	PROGRAM name PROG name BLOCK name ABS CM (default)	octal address octal address octal address octal address	Entry point (XFER address).	First instruction in tracing range.
TO	PROGRAM name PROG name BLOCK name ABS CM (default)	octal address octal address octal address octal address	Last word address of the program.	Last instruction in tracing range.
WHEN	PROGRAM name PROG name BLOCK name ABS CM ABS ECS (default) $\left.\begin{matrix}\text{Ai}\\ \text{Bi}\\ \text{Xi}\end{matrix}\right\}$ †† P	octal address (optional) octal address (optional) octal address (optional) octal address (optional) octal address (optional)	A change to any location or register causes a dump.	Dump condition: a dump is taken if the contents of the specified location or register change value. (P) must change through a jump. Normal incrementation of P does not cause a dump.
START	None	Decimal number	1	Iteration through the tracing range at which to take the first dump.
EVERY	None	Decimal number	1	Interval after which the dump is to be repeated.
UNTIL	None	Decimal number	131071	Last iteration at which the dump is to be taken.

†If the optional phrase is used, the name portion of the phrase can be omitted.

††i refers to a register, where $(0 \leq i \leq 7)$.

TABLE 5-5. PHRASES FOR TRACK PARAMETERS

Phrase		Value Type	Description
Keyword	Value		
PROGRAM or PROG	name	Alphanumeric program block name, one to seven characters.	Program block referred to by the FROM, TO, or WHEN parameter.

BLOCK	name	Alphanumeric block name, one to seven characters. Blank common is indicated as follows:
		// CM
		//L ECS
			Common block referred to by the FROM, TO, or WHEN parameter.

ABS	type	Core memory type: CM ECS	Type of memory containing absolute address specified by the FROM, TO, or WHEN parameter. If type is omitted, CM is assumed.

one instruction at a time and simulating its effect. TRAPPER also prints a trace of the instructions executed. This interpreting process is complex and proceeds much more slowly than direct execution of the same instructions.

Rules for use of the TRACK directive are as follows:

- The first phrase encountered in a directive (for example, PROGRAM, PROG, BLOCK, ABS) becomes the default until another one is encountered.
- If specified, a parameter must be complete; that is, keyword, phrase (if applicable) and value must be present. If no parameter is specified, the item listed in the default column is used.
- With the exception of the WHEN parameter, which is processed each time it is encountered, no more than one value for a given parameter is processed for a given TRACK directive. A maximum of 15 WHEN parameters can be used in a single directive. Multiple specification of a parameter causes all but the last specification to be ignored.
- The octal address fields are limited to ten digits, including zeros. Also, they are limited to a maximum value of 131,071. Otherwise, the fatal diagnostic "ILLEGAL NUMBER ON ABOVE TRAP DIRECTIVE" will occur. (See appendix B.)

Output from TRACK consists of a dump of any registers or memory locations changed by the instruction, the COMPASS image of the instruction, and full register dumps at the beginning and the end of each range.

Examples of the TRACK directive are shown in figure 5-4.

Example 1

Label	Verb	Specification
Q , ,	TRACK	FROM PROGRAM S 60, TO PROGRAM E 70, START 3, EVERY 2, UNTIL 7, WHEN B4, WHEN B7

Example 2

Label	Verb	Specification
TAG ,	TRACK	FROM PROG MAIN 100 TO PROG MAIN 150

Example 3

Label	Verb	Specification
JUMP	TRACK	WHEN P

Figure 5-4. TRACK Directive Examples

In example 1, all output is labeled Q. Tracing extends from location 60_8 of program S to location 70_8 of program E. If register B4 or B7 changes value during the third, fifth, or seventh times through the range, output is produced.

In example 2, tracing extends from location 100_8 through location 150_8 of the program MAIN. Output is produced whenever any instruction in this range changes the value of a memory location or a register.

In example 3, the entire program is traced. A dump is produced each time a jump is executed. P must change by more than 1 for a dump to be taken.

TRAPPER CALL INSERTED INTO USER'S PROGRAM

The TRAP control statement causes a TRAPPER call to be automatically inserted in the user's program. The TRAPPER call replaces the instruction within the user's program specified by either the FRAME AT (addr) or the TRACK FROM (fwa) parameter. The replaced instruction is saved in a TRAPPER execution-time table and executed, as follows:

- For FRAME, the replaced instruction is executed after the TRAPPER dump is taken
- For TRACK, the replaced instruction is the first traced instruction

The internal format of the TRAPPER call is shown in figure 5-5.

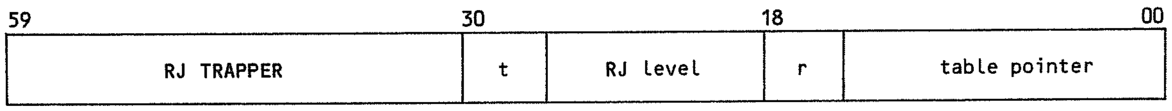
Neither the TRACK FROM location nor the FRAME AT location should be the destination address of a return jump instruction. If so, the TRACK or FRAME is not executed.

TRAPPER also adds an RJ TRAPPER to the destination of an RJ instruction that jumps outside a simulated TRACK. If the program modifies the destination word or uses the calling address for an argument list (such as CPC), TRAPPER does not function properly. The destination word must be included with the TRACK that references it. Blocks containing absolute program relocation can cause TRAPPER to be overwritten. In addition, a FRAME is not taken correctly if the location being framed contains a return jump instruction where the next location is a parameter. This means that return jump calls to CPC should not be framed.

TRACK output does not begin until the word specified by the TRACK FROM is executed. A jump into the TRACK range does not cause TRAPPER to generate output. To generate a TRACK of a subroutine where the entry appears within the middle of the TRACK range, two TRACKS are needed. For example, the TRACK directives needed to track the subroutine shown in figure 5-6 are:

```
TRACK FROM PROG SUB 0 TO PROG SUB 1

TRACK FROM PROG SUB 3 TO PROG SUB 4
```



t Type of call:

0	TRACK
1	FRAME

r Return flag:

0	Normal call
1	Return from RJ within a track

table pointer	Identifies the TRAPPER execution-time table containing output specifications for the call.
RJ level	Number of return jumps within active TRACKS (initially zero).

Figure 5-5. TRAPPER Call Internal Format

Location	Operation	Variable Subfields	Comments
	IDENT	SUB	
	ENTRY	SUB	
SUBA	SB1	2	word 0 of sub
	SX0	3	
+	SA2	SUBA	word 1 of sub
SUB	PS		word 2 of sub
+	SB2	1	word 3 of sub
	BX1	X2	
+	EQ	SUBA	word 4 of sub
	END		

Figure 5-6. Tracked Subroutine

A jump, other than a return jump, outside the tracking range turns off TRACK, and it can only be restarted by reentering at the FROM address. If a return jump jumps outside the tracking range, however, tracking resumes if return to the subroutine is through the normal exit.

TRAP ERROR CONDITIONS

The TRAP directive translator converts FRAME and TRACK directives into internally formatted tables. The syntax of a directive is checked and if an error is detected, the directive is skipped and an error message is written in the output file. In addition, dayfile messages and execution time errors are generated as applicable (see appendix B).

For both FRAME and TRACK, buffers are flushed before control is passed to user code; therefore, no loss of output is possible on a mode error. If TRACK detects a mode error during a tracking operation, a message is issued and the job aborts.

OVERLAYS 6

A user can divide a large program into sections, called overlays, to reduce the amount of memory required for job execution. Different overlays can occupy the same storage locations at different times.

Each overlay contains data and instructions needed at different times during job execution. However, commonly used routines should be placed in the main overlay, which is in memory throughout job execution, to reduce time required for loading overlays.

A user should be aware of the following:

- Overlays, other than the main overlay, are loaded by an explicit user call. The main overlay can either be loaded into memory by terminating the overlay generation run with an EXECUTE statement, or by issuing a name call statement that references the file to which the generated overlays were written.
- Main overlays can have multiple entry points and execution can begin at more than one entry point in the main overlay.
- Each higher level overlay must have a single entry point designated as a transfer address.
- The main overlay cannot reference entry points in any other overlay; primary overlays can reference entry points in the main overlay; and secondary overlays can reference entry points in either the associated primary overlay or the main overlay if the entry points have not been overwritten. See OVERLAY Directives later in this section.

Segments and capsules provide other means of dividing large programs into sections to better utilize memory and time during job execution. A special type of capsule, OVCAP, is designed to be used with overlays (see section 8).

OVERLAY GENERATION

Overlays are generated from programs as directed by OVERLAY directives. These programs are loaded and relocated, and images of the programs are created in absolute form. These images are written on a designated file as overlays. Later, the overlays can be reloaded from the file for execution without requiring relocation.

OVERLAY LEVELS

Each overlay is identified by an ordered pair of octal level numbers, 0 through 77. The three levels of overlays that are possible are the main, primary, and secondary levels of overlays.

The main overlay must be level (0,0). It remains in memory throughout job execution.

The primary level is subordinate to the main overlay. It is denoted by a nonzero primary level number and a zero secondary level number. For instance, (1,0), (2,0), and (3,0) are examples of primary level numbers. Primary overlays are loaded in response to a request issued by the main overlay. Up to 63 (decimal) primary levels can exist in a structure.

A hierarchy exists between primary and secondary overlays. The secondary overlay (both level numbers are nonzero) is subordinate to the primary overlay having the same primary level number. Thus, secondary overlays associated with primary overlay (1,0) could be (1,1), (1,2), and (1,3). A secondary overlay can be called into memory by its associated primary overlay or by the main overlay. Up to 63 (decimal) secondary levels can exist for each primary overlay.

During overlay generation, a primary overlay is assigned an origin at the next location immediately following the main overlay (0,0), unless an origin parameter on the OVERLAY directive is specified. If the main overlay includes blank common, loading begins after blank common.

The origin of a secondary overlay immediately follows its associated primary overlay (including any blank common), unless an origin parameter is specified. No more than three overlays can be loaded concurrently: the main overlay, one primary overlay, and one secondary overlay associated with the primary overlay.

Figure 6-1 illustrates a possible sequence of overlay generation. The loader prepares 12 overlays in this example. The sequence of generation does not imply that the programs are loaded in memory in the same sequence, or that they remain in memory for a set period of time when they are executed.

In figure 6-1, the main overlay (0,0) is always in memory. Primary overlays (1,0), (2,0), (4,0), (6,0), and (7,0), and/or secondary overlays (1,1), (1,2), (2,1), (4,1), (4,2), and (4,3) are in memory as needed. The main overlay can call either a primary overlay or secondary overlay into memory. A primary overlay once in memory can call an associated secondary overlay.

In figure 6-1, overlay (6,0) is assigned an origin within the blank common area because it has a nonzero origin parameter. This example also illustrates that the numbers assigned to primary overlays need not be consecutive.

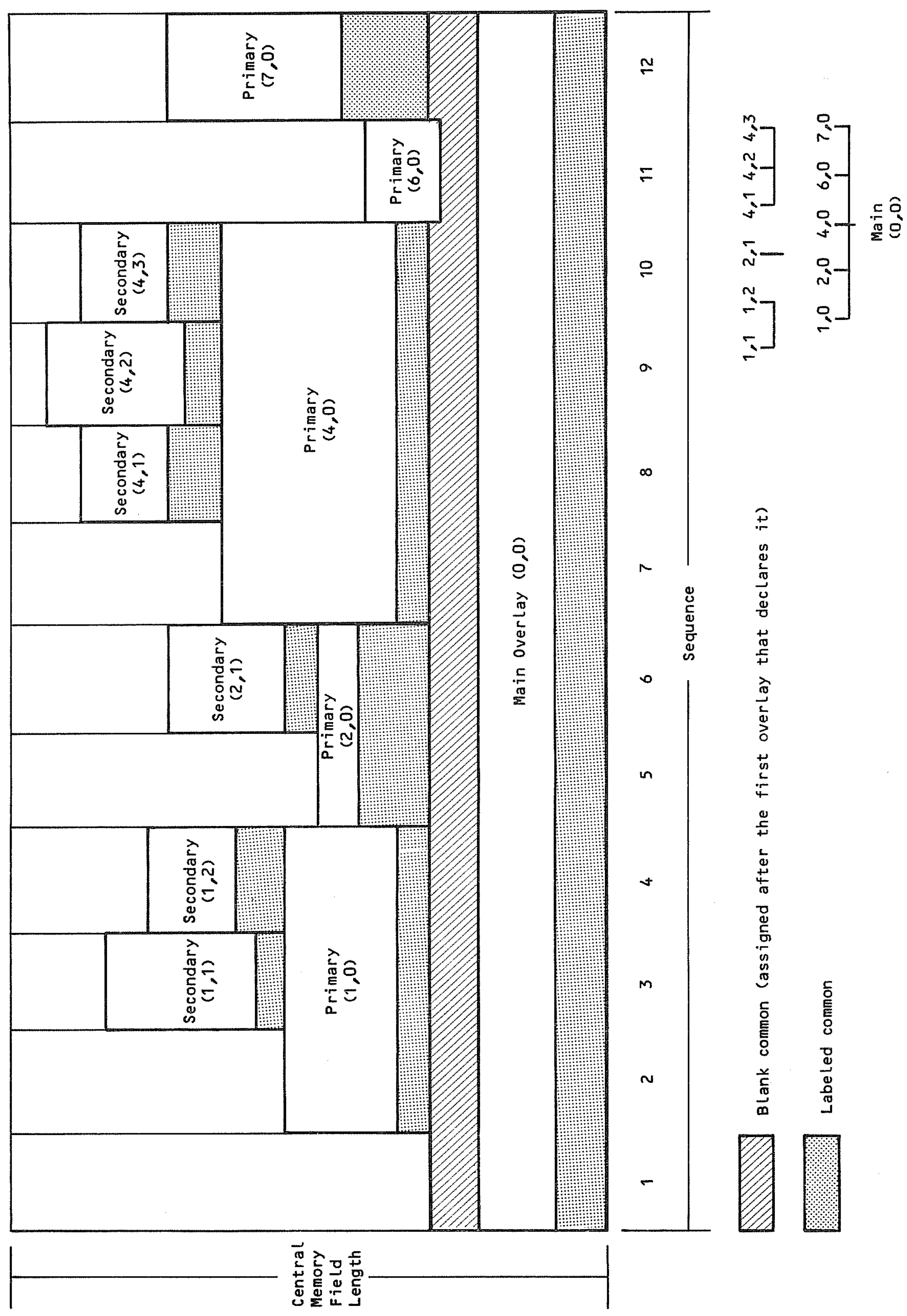


Figure 6-1. Overlay Structure

OVERLAY DIRECTIVES

The loader is instructed to create overlays by OVERLAY directives encountered as separate records in the load input stream to the loader. An OVERLAY directive must precede the first binary table of the series of one or more records that comprise the overlay. The formats of the overlay directive are shown in figure 6-2.

An OVERLAY directive must be a distinct record in its file within the load input stream to the loader. Some rules for the use of OVERLAY directives are:

- So that the load operation can be recognized from the beginning of an overlay generation, the first loader input encountered must be an OVERLAY directive. It must specify a main (0,0) overlay.

- The hierarchy of overlay levels must be adhered to in the loader input. That is, any programs composing a group of secondary overlays must appear immediately after the programs for the corresponding primary overlay.

  More than one directive can specify the same overlay level. This results in more than one overlay of the same level; it does not cause information to be added to the previously generated overlay. More than one main (0,0) overlay can be generated; which, in effect, results in more than one separate overlay structure.

- A common block in an overlay is defined to be not overwritten for a subsequent overlay if the common block first-word-address (fwa) is lower than the origin of the subsequent overlay. An entry point in an overlay is defined to be not overwritten for a subsequent overlay if the fwa of the block in which the entry point is defined is lower than the origin of the subsequent overlay. Otherwise, the common block or entry point is defined to be overwritten. Once a common block or entry point is overwritten, it no longer exists (and therefore can be redefined).

  References to entry points or common blocks in the main overlay can be made from that main overlay or from any associated primary of secondary overlays for which the entry point or common block is not overwritten.

  References to entry points or common blocks in a primary overlay can be made from that primary overlay or from any of its associated secondary overlays for which the entry point or common block is not overwritten.

  References to entry points or common blocks in a secondary overlay can be made only from that secondary overlay.

  Data can be preset into a labeled common block only by the lowest level overlay that declares it.

- The writing of each overlay resembles a normal relocatable load completion, except that execution does not begin. At least one transfer symbol must occur in the load input for each overlay.

- An empty (zero-length) logical record terminates the reading of programs during overlay generation on both NOS and NOS/BE.

- Absolute relocation is not permitted during overlay generation.

In processing overlay directives, the following sequences are illegal:

- 1,0 0,0 2,0

  Illegal because the main overlay must be first.

- 0,0 1,0 2,2 2,0

  Illegal because the secondary overlay 2,2 should follow primary overlay 2,0.

The following sequences are legal:

- 0,0 1,0 77,0 77,44 77,12 3,0

- 0,0 1,0 0,0 1,0 1,0 1,1 1,1

OVERLAY MODULES

The object programs immediately following one OVERLAY directive record and up to the next OVERLAY directive record or end-of-file form an overlay. As with basic loads, the request terminating an overlay generation load is either EXECUTE, NOGO, or a file name call.

A NOGO merely causes the completion of the last overlay. An EXECUTE causes load completion, followed by loading and execution of the first main (0,0) overlay. All subsequent overlays to be loaded must be loaded by user calls. (See the discussion of overlay loading and execution later in this section.) Overlays can be loaded at execution time without regard to the sequence used during their generation, except that secondary overlays should be loaded only after loading the associated primary overlay.

After the object programs that comprise the overlay have been loaded, the loader completes loading by satisfying undefined external references from the library set. It then writes the overlay on the file.

Each overlay generated is composed of a 7700 (PRFX) table followed by a 5400 (EACPM) table. The 5400 table can contain multiple entry points for a main (0,0) overlay. These multiple entry points are generated by including COMPASS LDSET pseudo instructions (EPT option) in the first relocatable program after the (0,0) OVERLAY directive. If no such LDSET instructions are encountered, then a single entry point is generated whose name is the same as the overlay name, and whose address is that of the last transfer symbol (XFER table, appendix D) encountered. Primary and secondary overlays can only have a single entry point whose name is the same as the overlay name; its address is that of the last transfer symbol encountered.

```
OVERLAY(lfn,0,0,OV=n)

OVERLAY(lfn,l1,l2,origin)
```

lfn — The file name on which the overlay is to be written.

If a file is specified on a NOGO statement, it overrides the file named on any overlay directive. If a file is specified on a previous directive but not on the current directive, then the previously specified file is used. If neither the directives nor a NOGO statement specified a file name, the file ABS is used.

If the load input file is specified, a fatal error occurs and the load is aborted.

l_1 — The primary level number, in octal (0 through 77). For the first OVERLAY directive, l_1 and l_2 must be 0,0 (main level number).

l_2 — The secondary level number, in octal (0 through 77). The number must be zero for primary overlays.

origin — An optional parameter specifying the origin of the overlay; not allowed for the (0,0) overlay. The parameter specifies the address of the first EACPM table header word; the actual program and common blocks begin five words beyond this address. The loader accepts any of the following forms:

Cnnnnnn	The overlay is loaded nnnnnn words from the start of blank common. nnnnnn must be an octal number, up to six digits. If this overlay is a primary overlay, blank common must be declared in the (0,0) overlay. If this overlay is a secondary overlay, blank common must be declared in the primary overlay, and not the (0,0) overlay.
O=nnnnnn	The overlay is loaded at the address specified; nnnnnn must be an octal number greater than or equal to 110_8.
O=eptname	The overlay is loaded at the address of the entry point specified, which must have been declared in a lower level overlay, in a block whose fwa has not been overlayed.
O=eptname nnnnnn	Same as above, but the address is biased by the amount of the offset.

Be very careful when specifying the origin for an overlay. If the fwa of an overlay is less than the lwa+1 of an associated lower level overlay, the overlapping area will be overwritten at overlay load time. The overwritten area must not contain data or subroutines needed later; no attempt is made to save or restore the content of the overwritten area. Note that a program or common block which is partially overwritten is defined to be not overwritten (see OVERLAY Directives rule 3).

If the origin parameter is omitted, the preceding comma must also be omitted.

OV=n — An optional parameter specifying that the overlay generator is to generate an overlay structure suitable for Fast Overlay Loader (FOL). OV=n can only be specified on the 0,0 overlay. n specifies the decimal number of higher level overlays and OVCAPs in the overlay structure; it must fall in the range of 0 n 20000.

If OV=0 is specified, a FOL directory is not generated.

Figure 6-2. OVERLAY Directive Formats

When any program in an overlay structure is modified, the entire overlay structure should be reprocessed through the loader. If an attempt is made to rebuild only a subset of the overlay structure, the overlays not rebuilt might still contain references to entry points in the original main overlay. These references could be different even if no code is changed in the main overlay because programs that are loaded to satisfy externals are loaded in random, and possibly changing, order.

Examples of overlay generation are shown in figure 6-3.

Example 1 illustrates the creation under NOS of a permanent file containing a single overlay binary program file. The most common reason for doing this is to eliminate the need to relocate and link the program each time it is used. The FORTRAN 5 compiler copies the OVERLAY directive onto the binary object deck written on file OVERBIN. LOAD directs the loading from OVERBIN. The loader reads OVERBIN and encounters the OVERLAY directive instructing it to create a main overlay (0,0) and to write it on file ABCD. The NOGO statement inhibits execution of the program after loading is complete.

Example 2 shows a COMPASS example of multiple overlay generation and execution on NOS/BE. COMPASS assembles the source decks and writes them on file LGO (the system default file for binary output). The binary output is interspersed with records containing OVERLAY directives resulting from LCC pseudo instructions.

```
Example 1:

Job statement
USER statement
CHARGE statement
DEFINE,ABCD.
FTN5,B=OVERBIN.
LOAD,OVERBIN.
NOGO,ABCD.
7/8/9

    OVERLAY(ABCD,0,0)
    FORTRAN 5 Source Deck

6/7/8/9


Example 2:

Job statement
ACCOUNT statement
COMPASS.
LGO.
FRANK.
7/8/9

    COMPASS source deck containing the following
    LCC pseudo instructions:

       LCC OVERLAY(FRANK,0,0)
       LCC OVERLAY(JOHN,1,0)
       LCC OVERLAY(JOHN,1,1)

7/8/9

    Data to be used during execution

7/8/9

    Data to be used during execution

6/7/8/9
```

Figure 6-3. Examples of Overlay Generation

The LGO control statement causes loading from the LGO file, and the loader processes the OVERLAY directives. The main overlay is written on file FRANK. Primary overlay (1,0) and secondary overlay (1,1) are written on file JOHN. Both files, FRANK and JOHN, are local to the job and are released when the job terminates. This type of job might be used for preliminary overlay creation and checkout.

After all overlays have been created, execution begins by using the first data file in the job. Because a second LGO statement would cause the entire process to be repeated, including unnecessary recreation of all overlays, the second run is expedited by using name call statement FRANK instead of a second LGO. Name call statement FRANK causes the main overlay to be loaded without relocation from file FRANK. It can then process the second data file and call for loading of the overlays.

ERROR PROCESSING DURING OVERLAY GENERATION

The following processing occurs during overlay generation:

1. If a fatal error occurs during overlay generation, the loading process terminates at the completion of that overlay.

2. If a map is requested, it is written, reflecting the overlay being generated at the time the fatal error occurred.

3. If ERR=NONE is not selected on the LDSET statement or as an installation default, the job is aborted after supplying the above information. If ERR=NONE is selected the load operation is terminated, but the job is not aborted.

As with basic loading, nonfatal errors do not interrupt the progress of the load, although their eventual effect might lead to a fatal error.

OVERLAY LOADING AND EXECUTION

After overlays are generated, they reside on the files specified. Unlike segments, there is no provision for automatic loading of overlays.

When the program is to be executed, the main overlay is brought into memory either as a result of an EXECUTE request terminating the load sequence or as a result of a name call statement in the job deck. Thereafter, additional overlays are called into memory by the executing program. Overlays can be loaded at execution time without regard to the sequence used during their generation, as long as the hierarchical structure is observed.

FORTRAN-compiled programs commonly use the CALL OVERLAY statement described in the FORTRAN reference manuals. A user call in a COMPASS language program consists of a LOADREQ macro. The Fast Overlay Loader provides an alternative to the COMPASS LOADREQ macro.

LOADREQ (REQUEST OVERLAY LOAD)

The LOADREQ macro (figure 6-4) requests that an overlay be loaded into memory. Word paddr is the first word address of a table of parameters. No macro exists for generating this table. The user must generate the table by using COMPASS code. The internal format of the table is shown in figure 6-5.

The operation performed depends on the values of the parameter words. LOADREQ call processing is not fully compatible for the NOS and NOS/BE operating systems.

For LOADREQ calls on NOS, the following processing is performed:

1. If u is zero, n is ignored and name is the name of the file containing the overlay. The overlay is loaded by level number (level 1 and level 2 are required), and ovlname and eptname are ignored if present.

2. If u is 1, the overlay is loaded by entry point name from the system. If n is zero, name is the entry point name. If n is 1, ovlname is the entry point name and name is ignored. If n is 2, eptname is the entry point name and ovlname and name are ignored.

3. If fwa is zero, the overlay is loaded at the address specified by the overlay.

4. If fwa is nonzero, the overlay is loaded at the specified address.

5. If level 1=level 2=0 on the request, or e is 1, control is transferred to the called overlay; otherwise, control is returned to the caller with fwa=entry address.

6. lwa is ignored. The last word address to which the overlay can extend is assumed to be FL-1.

7. v=1 must be specified for an overlay load.

For LOADREQ calls on NOS/BE, the following processing is performed:

1. If u is zero and n is zero, name is a local file name and the overlay is loaded by levels 1 and 2.

2. If u is zero and n is 1, name is the name of the file containing the overlay and ovlname specifies the overlay name. The overlay is loaded by ovlname. Level 1 and level 2 are used if ovlname is zero.

3. If u is zero and n is 2, name is the logical file name, ovlname is the overlay name, and eptname is the entry point name. Level 1 and level 2 are optional but are used by default if ovlname is zero; otherwise, the overlay indicated by ovlname is loaded and the entry point name indicated by eptname is used. If eptname does not exist in the overlay, the first entry point in the overlay header is used.

4. If u is 1 and n is zero, name is the overlay name to be searched for in the global library set, followed by the default library NUCLEUS. Level 1 and level 2 are ignored.

5. If u is 1 and n is 1, name is the library name and ovlname is the overlay name. Level 1 and level 2 are ignored.

6. If u is 1 and n is 2, name is the library name, ovlname is the overlay name, and eptname is the entry point name. Level 1 and level 2 are ignored. If eptname does not exist in the overlay, the first entry point in the overlay header is used.

7. If fwa is zero, the overlay is loaded at the address specified in the overlay header.

8. If fwa is nonzero, the overlay is loaded at the specified address. It is recommended that fwa be specified whenever possible as this yields more efficient processing. If fwa is not specified, the reading of directories and overlay headers takes place at an area in central memory that the user might not want destroyed. That is, the location specified in bits 17 through 0 of RA+65_8 is used until the first word address from the overlay header is obtained.

9. If lwa is zero, the default last word address to which the overlay can extend is assumed to be FL-3.

10. If lwa is nonzero, it specifies the last address+1 to which the overlay can extend. In this case, lwa is less than or equal to FL-3.

11. v=1 must be specified for an overlay load.

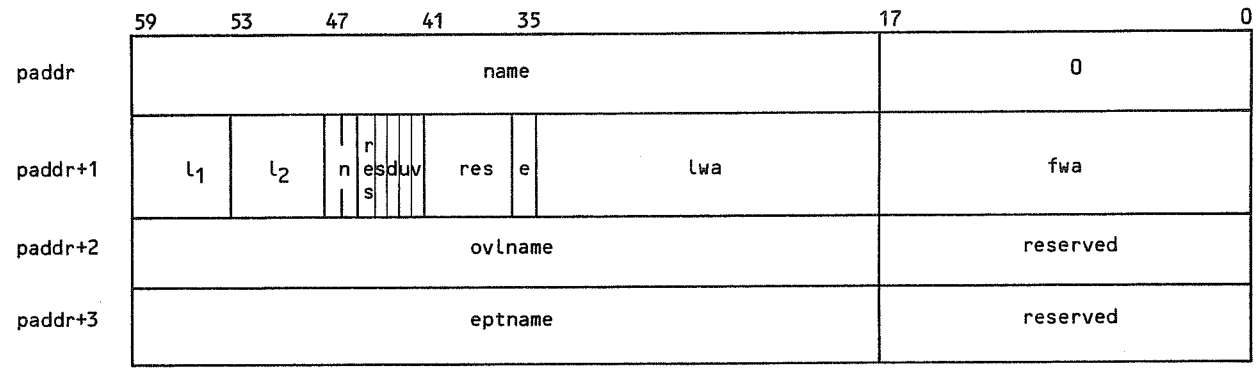
12. If e is zero, control is returned to the calling program.

13. If e is one, control is transferred to the loaded overlay. This should not be used when auto-recall is specified in the LOADREQ macro.

Location	Operation	Variable Subfields
	LOADREQ	paddr,rcl,flag

paddr	An address expression, specifying the first word of the parameter area.	
rcl	Auto-recall indicator (optional):	
	null	Control returns immediately
	not null	Control returns after the call is completed
flag	A keyword indicating special action. Required if the Common Memory Manager (CMM) is active at the time of the call. The keywords are:	
	null	No special action; CMM must not be active.
	CMM	Indicates an overlay load by a job that uses CMM. This call activates CMM if it is not already active.
	DATA	Indicates that the loaded overlay is to be treated as data, such as a system text. This call can be made when CMM is either active or inactive, and neither activates nor deactivates it if it is present. Both fwa and lwa must be supplied in the parameter area, and the e bit must not be set.

Figure 6-4. LOADREQ Macro Format (Overlay Load)



name	A library name, overlay name, or entry point name (see below), left-justified with zero fill.
l_1	Primary overlay level.
l_2	Secondary overlay level.
n	Number of words-2 in the request (bits 47 and 46).
s,d	Used by CMM.
u	Load option (bit 42).
v	Overlay load flag (bit 41).
e	Automatic execute flag (bit 36).
lwa†	The last word address available for load. For NOS/BE, lwa is the last word address available +1.
fwa	The first word address available for load.
ovlname	The overlay name, left-justified with zero fill.
eptname	The entry point name, left-justified with zero fill.
res reserved	Fields that can be used only by CDC.

†Under NOS/BE, lwa should be specified whenever possible. If lwa is not specified, then the reading of library directories and/or overlay headers can take place in an area of CM (from fwa to FL-3) which the user might not want destroyed.

Figure 6-5. LOADREQ Table Internal Format

Error processing and return information from LOADREQ processing differs for the NOS and NOS/BE operating systems.

Under the NOS operating system, errors encountered during LOADREQ processing cause the job step to abort. If no errors occur and the parameter words are not destroyed by the loaded overlay, the parameter words are updated as shown in figure 6-6.

Under the NOS/BE operating system, information is returned in the parameter area (as shown in figure 6-7), if the parameter words are not overwritten by the loaded overlay.

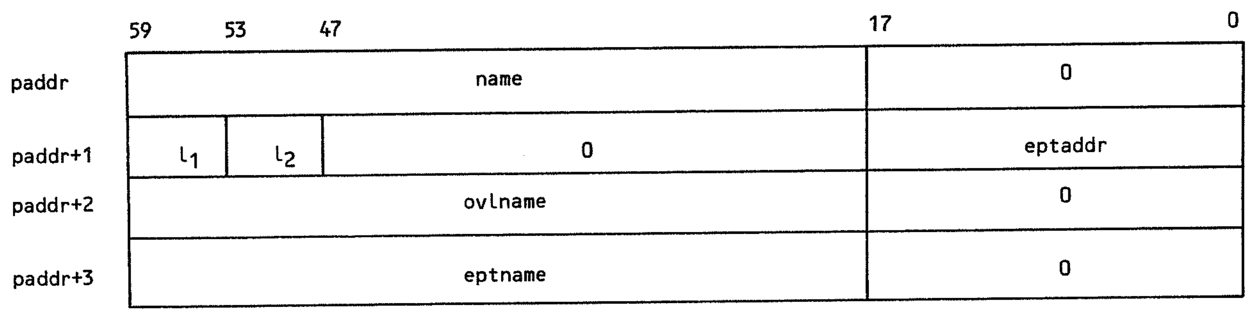
For both NOS and NOS/BE, fields other than those shown in table 6-1 are unchanged.

Request Processing

The loading of the requested overlay takes place by execution of the peripheral processor program LDV under NOS/BE or the peripheral processor program LDR under NOS. The LOADREQ macro always calls LDV; NOS translates an LDV call into an LDR call.

When auto-recall is specified, control is not returned to the user program until loading of the requested overlay is completed.

If the e bit is set in the request, it is assumed that the overlay being loaded will overwrite the overlay making the call. Here, no updating of the request takes place. Upon completion of the load, the specified entry address is entered directly. A fatal error causes job abortion.

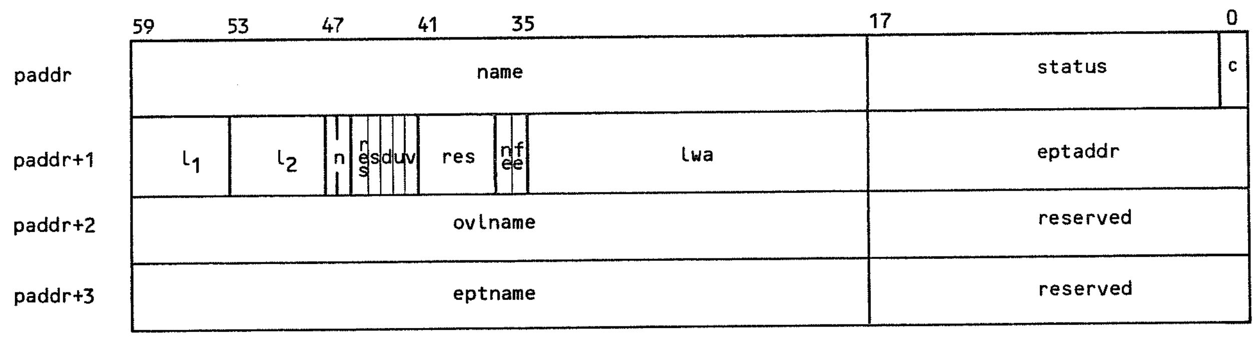


l_1 Overlay level number 1

l_2 Overlay level number 2

eptaddr Entry point address of the overlay; if n is 2, eptaddr is the address of eptname.

Figure 6-6. LOADREQ Table Internal Format After Overlay Load Under NOS



l_1 Overlay level number 1

l_2 Overlay level number 2

Figure 6-7. LOADREQ Table Internal Format After Overlay Load Under NOS/BE

Making LOADREQ Calls Compatible for NOS and NOS/BE

Because some differences exist in LOADREQ processing on NOS and NOS/BE, it is wise for programs that must run on both systems to make LOADREQ calls that are compatible. The compatible calls are:

- Two-word calls (n=0) to load overlays from nonlibrary local files by level number.
- Two-word calls (n=0) to load single entry point overlays from the default system library. (Here, the entry point name is the same as the overlay name.)
- Three-word calls (n=1) and four-word calls (n=2) to load overlays from the default system library.

The user should also be aware of the following notes and recommendations:

- Under NOS, LOADREQ calls to load from local files are always treated as two-word calls (n=0).
- Under NOS, a LOADREQ call specifying a (0,0) level overlay (level 1=level 2=0) results in control being passed to the loaded overlay, regardless of the setting of the e bit.
- When issuing LOADREQ calls, ensure that both the overlay name and level numbers are correct.
- When issuing LOADREQ calls, completely reset the parameter words, as LDV/LDR return information overwrites different parts of the parameter words.
- Two-word calls (n=0) to load from the default system library must specify the overlay name under NOS/BE and entry point name under NOS.

TABLE 6-1. PARAMETER AREA FIELDS CHANGED BY OVERLAY LOAD UNDER NOS/BE

Word	Bits	Field	Significance
paddr	17 through 1	status	Contents of the field depend on ne and fe, as follows: ne fe Contents of Status Field 0 0 Zero 0 1 Fatal error code 1 0 Error code for first nonfatal error 1 1 Fatal error code Refer to appendix B for error codes.
	0	c	Completion flag: 1 Call is completed
paddr+1	37	ne	Nonfatal error flag: 0 No nonfatal errors have occurred 1 One or more nonfatal errors has occurred
	36	fe	Fatal error flag: 0 No fatal errors have occurred 1 A fatal error has occurred
	17 through 0	eptaddr	Address of the entry point of the overlay. If n is 2, this field contains the address of the eptname in word 3, except if the specified eptname was not found, in which case the field contains the address of the first entry point.

Recommended Procedure for Overlay Loading

To make a program most flexible, a single subroutine should be used for all overlay loading. The subroutine should be written by using the following rules:

- Set v to 1, e as desired, and n to 1 (three-word call).
- Set both ovlname and level 1 and level 2 correctly.
- Set fwa and lwa if known.
- Set name from top 42 bits of RA+64_8.
- Set u from bit 18 of RA+65_8.
- If name is NUCLEUS and u is 1, change name to SYSOVL.

If the preceding steps are followed, the program can be either run from a local file, run from a NOS/BE user library, or installed as part of either operating system without requiring any changes to the program.

FAST OVERLAY LOADER

The Fast Overlay Loader (FOL) provides an alternate method of loading primary and secondary overlays. The FOL overlay structure is generated with an overlay directory built into the main (0,0) overlay. Specifying the OV parameter on the (0,0) OVERLAY directive (figure 6-2) initiates generation of the directory; if OV=0, then no directory is generated. All higher level overlays are required to be written onto the same file as the main overlay. The structure of the overlay file is required to remain unchanged for proper functioning of the FOL; FOL overlay structures can be put onto libraries provided that the order of the programs is unchanged.

A small resident loader, FOL.RES, loads the primary and secondary overlays; it is a part of the main overlay.

When the main overlay is loaded, its file and random address are known and are placed in the 5400 table of the (0,0) overlay. When a call to the FOL resident occurs, the higher level overlay is loaded from the same file that the main overlay resides on. The random address of the overlay is calculated by the random address of the main overlay and a relative address bias to the desired overlay; hence, the requirements to preserve the order of the overlay structure and to write all overlays onto the same file.

Fast Overlay Loading

If an FOL overlay structure has been built, then the higher level overlays can be loaded by calls to the FOL resident. LOADREQ calls are still valid but are discouraged due to performance considerations and incompatibilities of NOS and NOS/BE.

The entry point FOL.LOV is called to load an overlay if CMM is not active; the entry point CMM.LOV is called if CMM is active. The following parameters should be passed to FOL.LOV or CMM.LOV by the calling program:

- The name of the overlay
- The level numbers of the overlay (level numbers are used if and only if the overlay name is set to zero)

The exact calling sequence for loading higher level overlays is shown in figure 6-8.

Automatic execution of the loaded overlay does not occur. Upon return from the FOL resident, the entry point address is in B7; the user must jump to this address to begin execution. (Note that B7 is negative if an error occurred.)

Load Overlay as Data

The FOL resident provides a facility to load an overlay at an address that is specified at execution time. Such an overlay is a data overlay which contains no executable code and can reside anywhere in memory.

The entry point FOL.LOD is called to load data overlays. The following parameters should be passed to FOL.LOD by the calling program:

- The overlay name
- The address at which to load the overlay

The exact calling sequence for loading data overlays is shown in figure 6-9.

The user is responsible for ensuring that the overlay does not destroy needed parts of memory.

Get Directory Entry

The FOL resident provides a facility for the user to obtain a directory entry for a higher level overlay from the FOL directory of the currently loaded main overlay.

The entry point FOL.GDE is called to obtain directory entries. The following parameters should be passed to FOL.GDE by the calling program:

- The overlay name
- The level numbers of the overlay (level numbers are used if and only if the overlay name is set to zero)

The exact calling sequence for obtaining directory entries is shown in figure 6-10.

RJ = XFOL.LOV (if CMM is not active)
RJ = XCMM.LOV (if CMM is active)

Entry:	(X1) = 42/ovlname,6/0,6/l_1,6/l_2.
Exit:	(B1) = 1. (B6) = fwa. (B7) = epta.
If error:	(B6) = (B7) = Error code: 777001 = OVERLAY NOT IN DIRECTORY. 7771nn = ERROR nn RETURNED FROM LOADQ. 7772nn = ERROR nn RETURNED FROM LOADREQ.
Saves:	A0, X0, B2, B3, X5.
Calls:	SYS=.
ovlname	The name of the overlay, left-justified with zero fill.
l_1,l_2	The level numbers of the higher level overlay. Level numbers are used if and only if ovlname=0.
fwa	The address of the first word of the 5400 table for the loaded overlay.
epta	Address of the first entry point.

Figure 6-8. Calling Sequence for Loading Overlays by the FOL

RJ = XFOL.LOD

Entry:	(X1) = 42/ovlname,18/fwa.
Exit:	(B1) = 1. (B6) = fwa. (B7) = epta.
If error:	(B6) = (B7) = Error code: 777001 = OVERLAY NOT IN DIRECTORY. 7771nn = ERROR nn RETURNED FROM LOADQ. 7772nn = ERROR nn RETURNED FROM LOADREQ.
Saves:	A0, X0, B2, B3, X5.
Calls:	SYS=.
ovlname	The name of the overlay, left-justified with zero fill.
fwa	The address at which to load the overlay which is the address of the first word of the 5400 table for the loaded overlay.
epta	Address of the first entry point.

Figure 6-9. Calling Sequence for Loading Data Overlays

```
RJ = XFOL.GDE

Entry:      (X1) = 42/ovlname,6/0,6/l1,6/l2

Exit:       (B1) = 1.
            (X6) = 42/ovlname,18/fwa
            (X7) = 6/l1,6/l2,30/undefined,
                   18/lwa+1

If error:   (X6) = 60/0 = error code; entry not
                          found.

Saves:      A0, X0, B2, B3, X5.

ovlname     The name of the overlay, left-
            justified with zero fill.

l1,l2       The level numbers of the overlay.
            Level numbers are used on entry if
            and only if ovlname=0.

fwa         The address of the first word of the
            overlay.

lwa         The address of the last word of the
            overlay (lwa+1 must be ≤ FL-3).
```

Figure 6-10. Calling Sequence for Obtaining a Directory Entry

Processing by the FOL Resident

How the FOL resident loads higher level overlays depends on certain conditions. The FOL checks these conditions in the order that follows; as soon as one is met, the FOL resident takes the appropriate action:

1. If the main overlay comes from the NUCLEUS library (NOS/BE), then LOADREQ is used to load higher level overlays from the SYSOVL library.

2. If the (0,0) directory information is available, then the FOL directory and LOADQ are used to load higher level overlays from the same file as the main overlay.

3. If the main overlay comes from the Central Library Directory (CLD) then LOADREQ from the CLD is used (NOS).

4. If none of the above conditions are met, then the file name from RA+64 is used; the main overlay is assumed to be at PRU 1. The FOL directory and LOADQ are used to load the higher level overlays from the same file as the main overlay.

SEGMENTATION 7

Segmentation provides an alternative to overlays for dividing large programs into sections to reduce the amount of memory required for job execution.

Segmentation is more flexible than overlays because of the following:

- Each segment can have more than one entry point
- Segment loading is done automatically, as needed, under control of a resident program (SEGRES)
- Up to 4096 segments can be specified in a segmented program
- Up to a combined total of 8192 program names, common block names, and levels can be specified by segment directives in a segmented program
- Up to a combined total of 8192 program names and common block names can be defined through object program input to the segment loader in a segmented program
- The job field length can be adjusted dynamically during program execution as segments are loaded and unloaded if the user so desires

Segmentation cannot be used for system programs, and segmented programs cannot be placed in libraries. A segment must be less than 400000_8 words.

Capsules provide an alternate means to segmentation and overlays for dividing large programs into sections to better utilize memory and time during job execution.

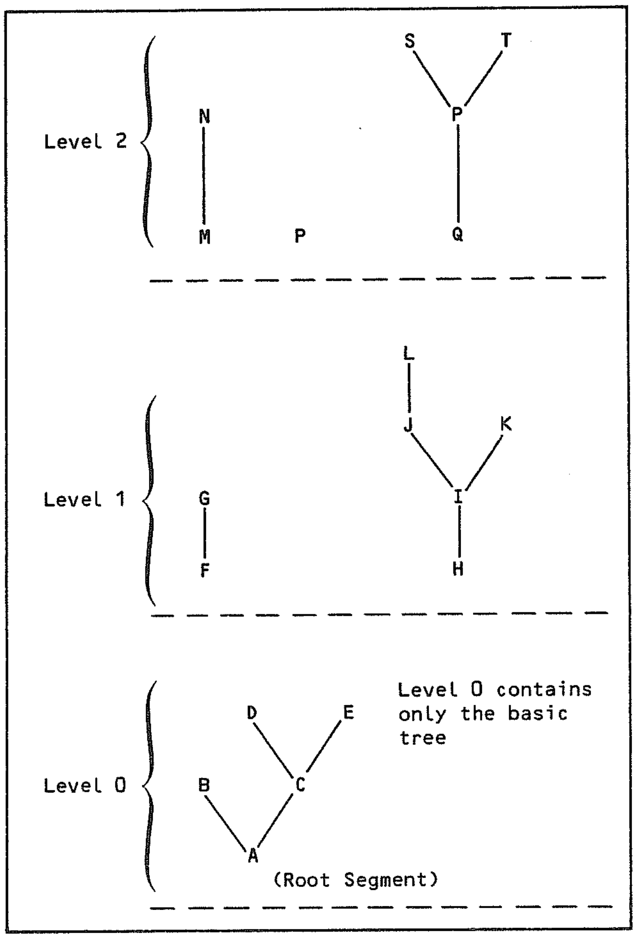


Figure 7-1. Sample Tree Structure

ORGANIZING SEGMENTS

A program can have up to 4096 segments. One main segment called the root segment must always remain in memory; it is never reloaded and no other segment can overwrite it. (The root segment is similar to a main overlay.)

Segments are organized in a structure resembling a tree. That is, segments branch out from the root segment in a fanwise (upward in memory) direction of higher numbered addresses. A sample tree structure is shown in figure 7-1, where A is the root segment.

The structure of a tree is defined by the user through the TREE directives. Each segment in the tree contains programs assigned to the segment, either by the user through directives or by the loader while satisfying external references.

Segments are organized within the tree structure by levels. In figure 7-1, segments A, B, C, D, and E are in the first level (lowest addresses in memory). This level constitutes the basic tree. Immediately above the topmost point of the basic tree, a new ground level is established from which two more trees spring: one tree is composed of segments F and G; the other tree is composed of segments H, I, J, K, and L.

A third level is also included in the example from which three new trees spring.

Any two segments that can be loaded into memory at the same time are called coexisting segments. Segments can coexist if either:

- They belong to two different trees springing from two different ground levels. (For example, segments in the tree springing from H can coexist with segments from the tree springing from Q.)

- They are in the same tree but are not connected by a path that involves moving down and then up in the branch structure. (For example, segment J can coexist with segment L.)

In figure 7-1, it is possible for segments A, C, E, and H to be in memory at the same time. Segments B and C, D and E, and F and H cannot be in memory at the same time because they occupy the same memory space. Segments and programs within segments are said to be compatible if they can coexist in memory.

Any set of segments has a nearest common ancestor, defined as follows:

- If the set contains a single segment, that segment is the nearest common ancestor of itself.
- If one of the segments in the set is the ancestor of all the others, it is the nearest common ancestor of itself and all the others. In figure 7-1, I is the nearest common ancestor of I and L.
- If the segments of the set are all in the same tree, but no one segment is the ancestor of all the others, their nearest common ancestor is, among all of their common ancestors (at least the root segment of the tree is such a common ancestor), the one closest to the set. In figure 7-1, C is the nearest common ancestor of D and E, although A is also a common ancestor.
- If the segments of the set are in different trees, the root segment of the bottom tree is their nearest common ancestor. In figure 7-1, A is the nearest common ancestor of either G and H, or G and P.

In addition, a segment can have either a direct ancestor or a direct descendant. For example, A is a direct ancestor of B and C; B and C are direct descendants of A.

ASSIGNMENT OF PROGRAMS TO SEGMENTS

The loader reads programs from the load files and divides the programs into fixed programs and movable programs.

Fixed programs are those assigned to segments by the user through TREE or INCLUDE directives.

Movable programs are programs not explicitly assigned to segments that are encountered by the loader while either loading programs or satisfying external references. The loader then has the task of assigning each movable program to a segment that can be accessed by all the segments referencing the program.

General rules for the assignment of programs to segments are:

1. The specification field of a TREE directive defines the structure of a tree, using the names of segments and/or other trees. Each name is a tree name if it is used in the label field of another TREE directive; otherwise, it is a segment name. The program named segname is implicitly included in the segment named segname only if the program named segname does not appear in the specification field of another INCLUDE directive; the program named segname must be explicitly listed if it appears in the specification field of another INCLUDE directive.

2. A copy of each program named on an INCLUDE directive is placed in the segment specified. If several INCLUDE directives name the same program, a copy of the program is placed with each of the segments indicated.

Assignment of Fixed Programs

A program named on a TREE directive is sought first among the programmer-defined load files and then in a library. If the loader is unable to find the program, a fatal error is generated.

The loader makes the lists of entry points and externals of all programs, and discards any externals for which there are matching entry points. Externals between incompatible fixed programs are not discarded.

Next, the loader searches the libraries for fixed programs that were not located on programmer-defined load files, and for unsatisfied externals of programs that were found on load files. As programs are loaded from the library, they are divided into fixed and movable programs and thereafter treated like programs from load files.

When all the programs have been read, the loader makes new lists of entry points and externals and begins the task of assigning movable programs to segments.

Assignment of Labeled Common Blocks

A labeled common block not declared GLOBAL or COMMON is assigned with the program that defines it. A labeled common block declared GLOBAL is assigned to the specified segment. A labeled common block declared COMMON is assigned to a segment which is the nearest common ancestor of all segments which reference the block. When a labeled common block is assigned in this manner, it is processed as a GLOBAL-SAVE block. If a labeled common block is declared both GLOBAL and COMMON, the GLOBAL specification overrides.

Assignment of Movable Programs

Assignment of movable programs is carried out in three steps, of which the second might be repeated:

1. The loader, whenever possible, satisfies externals of fixed programs with matching entry points in compatible fixed programs.

2. Step 2 has three parts, which are repeated until an iteration results in no reassignments of movable programs. Observe that these assignments are provisional until that stable iteration. A movable program might be initially unassigned, might then be assigned to a segment, later reassigned to a lower segment, and then ultimately assigned to the root segment if that is the only common ancestor. Reassignment is always along the ancestral chain toward the root segment.

a. On each iteration, the list of movable programs is scanned and each program is assigned or reassigned to the nearest common ancestor of the segments containing that program and any programs that call it.

b. If a movable program calls a fixed program that is incompatible with the segment to which the movable program is currently assigned, the movable program is reassigned to the nearest common ancestor of the segments containing the movable program and the program it calls.

c. Labeled common blocks declared COMMON (and not GLOBAL) are assigned to the nearest common ancestor of all segments which reference the block. Then all movable programs which preset the labeled common block but are assigned to an incompatible segment are assigned to the same nearest common ancestor.

During an iteration, an unassigned program remains unassigned if all the programs calling it are unassigned. (Iterations of the second step eventually terminate, because an iteration either results in no reassignments, or it approaches the point of having all the movable programs in the root segment, at which time iteration must stop.)

3. Any program that is unassigned when step 2 terminates is assigned to the root segment.

The final assignments of movable programs are such that a call to a movable program never causes loading; thus, it never causes part of memory to be overwritten unexpectedly. A call from a movable program to a fixed program can cause loading, but because all fixed programs are foreseen by the programmer, the effects of the loading are also foreseen (presumably).

SEGMENT LOADING

Any word containing an external reference to an entry point in another segment is replaced with a call to the segment loader resident while the other segment is not loaded. Execution of the word containing the jump to the resident causes the resident to load the needed segment and replace the original word.

This method of loading segments implies that external references between segments should only appear in executable code. For example, use of pointer words containing external references does not cause segment loading and leads to incorrect results. FORTRAN and SYMPL programs should not pass external references as parameters in subroutine or function calls. (These restrictions apply only to references between segments. References within a given segment are unrestricted.)

This method of loading segments can cause a segment to be loaded and not executed if within a single word there is a conditional jump followed by a jump to another segment. (This will not happen with FORTRAN programs because the compiler does not generate such code.) In the following COMPASS code sequence, for example, the segment containing the entry point SEG2 is loaded whenever CHECK is executed, regardless of the result of the ZR test.

```
CHECK  ZR    X1,MISS
       RJ    =XSEG2
MISS   BSS   0
```

A user must be sufficiently aware of segment structure to avoid such problems as the following:

> Consider segment A as the root segment having two branches, B and C. A is loaded automatically and execution begins at an entry point in A. The first reference outside of A is a nonreturn jump to an entry point in segment B. The loader traps this reference, loads segment B above segment A and executes the jump to B. B, however, contains a return jump to segment A. The execution of this return jump embeds in A an exit jump back to an address in B. Before this exit is taken, the loader encounters a jump reference to segment C, which causes C to be loaded automatically in memory formerly occupied by B. When A attempts to return to B, it cannot do so but, instead, actually jumps to an irrelevant location in segment C.

The preceding paragraphs describe the way in which central memory is assigned to and used by segments. The purpose of SEGLOAD is to allow segments to share central memory without requiring the user to do the necessary loading and reloading. However, the programs that comprise a segment can include blocks assigned to ECS as well as blocks in central memory. ECS is handled by SEGLOAD in a much less elaborate way. All the ECS belonging to a segment is gathered into a continuous block, and the blocks for all the segments are assigned space in ECS, beginning at RA(ECS)+0, with no overlapping. ECS blank common begins immediately after the last ECS block. At execution time, before any loading into central memory is done, all these blocks are read into ECS. Thereafter, they are never reloaded because the resident and the system do nothing to destroy them.

DUPLICATION OF NAMES

Restriction on the duplication of segment names, program names, and entry point names is explained in the following subsections.

Segment Names

Each segment in a tree must have a unique segment name, but these names need not differ from program, entry point, and common block names.

Program Names

Object programs need not have unique names, but the loader must be able to determine the segment to which a program is to belong. Thus, a program name that is not unique must not be in the list of programs on an INCLUDE directive because the loader has no way of knowing what program is meant. Any duplication of a program name causes a warning message.

Entry Point Names

The following rules govern the use of duplicate entry point names:

- All entry point names in fixed programs in one segment of a tree structure must be different; otherwise, an error message is issued and the first entry encountered is used.
- Duplicate entry point names are allowed in fixed programs in different segments of a tree structure.
- Fixed program entry point names can be duplicated in movable programs. The loader attempts to satisfy externals first with entry point names in fixed programs, and second with entry point names in movable programs. Thus, movable programs are searched only to satisfy externals not previously satisfied by fixed programs.
- Duplication of entry point names in movable programs on the load file or library is permitted when satisfying externals. The first occurrence of duplicate entry points in movable programs is used for satisfying external references not satisfied by a fixed program. Duplicate entry points in other movable programs are ignored.

LOCAL SAVE BLOCK

A special labeled common block named S$A$V$E is recognized as the LOCAL SAVE block. It is special-cased by the segment loader and treated as a GLOBAL-SAVE block for the program that declares it. Multiple occurrences of this block name are allowed; each occurrence generates a unique block.

Figure 7-2 shows examples of LOCAL SAVE block creation.

```
        Example 1:

           FTN5 program:

               SUBROUTINE SUB
               .
               .
               SAVE /COMM/
               COMMON /COMM/ VAL . . .
               .
               .
        Example 2:

           COMPASS program:

               IDENT SUB
               .
               .
               USE       /S$A$V$E/
           VAL BSS       1
               .
               .
               USE       *
```

Figure 7-2. Examples of LOCAL SAVE Block Creation in FORTRAN and COMPASS

Example 1 shows a LOCAL SAVE block creation in a FORTRAN5 program. Example 2 shows a LOCAL SAVE block creation in a COMPASS program. In both cases, a common block of name S$A$V$E is created, and the Segment Loader treats it as a LOCAL SAVE block.

RULES FOR REFERENCING COMMON AND ABSOLUTE BLOCKS

The following rules apply when referencing common and absolute blocks.

- Blank common can be referenced freely.
- Labeled common that has been declared on a COMMON directive, and not on a GLOBAL directive, can be referenced freely.
- Labeled common cannot be referenced by a segment other than the one that defines it unless it is made global through a GLOBAL or COMMON directive. If it is not made global, an attempt to reference the block results in a new common block in the referencing segment.
- A GLOBAL common block can be referenced by the segment to which it belongs and by all segments in higher branches of the tree structure that contain the segment as a base. Such references are safe if the GLOBAL block has been marked -SAVE; that is, the contents of the common block are not disturbed by segment loading. Such references are not necessarily safe if the GLOBAL block has not been saved, but the risk is entirely the user´s.
- A GLOBAL common block can be referenced by a segment that is in a direct path between the block´s owning segment and the root segment, or by a segment on a different level from the owner. This is risky because the owning segment might not be resident when the referencing segment is loaded. This differs from the case of a jump to an external symbol, which causes loading of the segment containing the external symbol (if necessary) before instruction execution.
- A GLOBAL common block should not be referenced by a segment that cannot be loaded legally at the same time as the segment that owns it. Such a reference results in unpredictable results. The same case applies to a reference to an external symbol; a fatal error occurs if the referencing and referenced segments cannot coexist.
- Absolute information is not permitted within segments.

SATISFYING OF EXTERNALS

When a segment is generated, all externals are considered to be unsatisfied. The unsatisfied externals are satisfied from the various segments in the tree structure according to the following order:

1. From a segment that is a direct ancestor, closest to the root segment, of the segment with the external reference
2. From the segment with the external reference

3. From a segment that is a direct descendant, closest to the root segment, of the segment with the external reference

4. From a segment that is in a different level, beginning with level 0 and continuing up through all levels

The following references refer to figure 7-1. In generating segment L, segment L has an external XL. Segments J, I, and K all have an entry point XL. The external XL in segment L is linked to entry point XL in segment I. Entry point XL in segment K is an invalid reference, and segment I is closer to the root segment than segment J. The satisfying segment is the valid reference that is closest to the root segment. If there is no valid entry point in the tree structure (in levels 0, 1, and 2), the external remains unsatisfied, which causes a nonfatal error.

Externals that remain unsatisfied after all the segments in the family have been considered are satisfied from the library, if possible, in accordance with loader control statements. (See the discussion of library searching in section 1.) For each library in succession, the following steps are carried out:

1. If all the fixed programs are found on load files, the loader continues with step 2. If there are fixed programs unaccounted for from load files, the loader consults the list of programs in the library; if no object programs match, none of the fixed programs are on the library and the loader continues with step 2. If some of the fixed programs are on the library, the loader reads them from the library and thereafter treats them as if read from a load file. This can cause additions to the list of unsatisfied externals, and it could allow some previously unsatisfied externals to be satisfied immediately and removed from the list.

2. The loader matches the list of unsatisfied externals against the list of entry points in the library. It marks the externals that do not appear in the library list as unsatisfiable, and those that do appear in the list as satisfiable. The satisfiable externals are satisfied from the current library; the unsatisfiable externals must be matched against the next library, if any.

3. The loader uses the external reference table of the library to add to the list of satisfiable externals, as necessary. Suppose that the satisfying of external W by using this library means that program Y will be added to the load (figure 7-3), and that Y has an external Z that can be satisfied by an entry point in program V in the library. Further suppose that V has an external U that can be satisfied by an entry point in program T in the library. T, however, has no externals to be satisfied from the library. Then, a consultation of the external reference table, along with the list of satisfiable externals in the family so far, and the list of entry points in the family so far enables the loader to enter the table with W, find Z, and stop the process if there is already a Z in the family. Otherwise, it adds Z to the list of satisfiable externals and reenters the table with Z. On the second pass through the table, the loader finds U and adds it to the list of satisfiable externals unless there is already an entry point named U in the family of segments.

4. Having completed the list of externals to be satisfied by this library, the loader consults other tables of the library to locate the programs that are to be read from the library, and does a first-pass read on them just as for programs from input files.

5. Library programs are assigned to segments on the same basis as nonlibrary programs, for which the rules are given above in the description of program assignment. The assignment of movable programs is done after all the relevant libraries have been consulted. It is done in such a way as to accommodate calls as much as possible (the connections of externals with entry points). But, no attention is paid in this process to the labeled common blocks that might be named by the movable programs, so you must consider the rules for referencing common blocks.

6. A library program can have externals that are not provided for in the external reference table of the library, because they cannot be satisfied by any program in the library. During the first pass read of a library program, these are either satisfied by entry points already known to be in the family, or else are added to the list of unsatisfiable externals.

7. When a library has been completely processed, if there are no remaining unsatisfiable externals, and no remaining program names from INCLUDE and TREE statements for which object programs have not yet been found, no further libraries are used. Otherwise, the unsatisfiable tag is removed from the remaining list of unsatisfied externals, and the next library (if any) is processed.

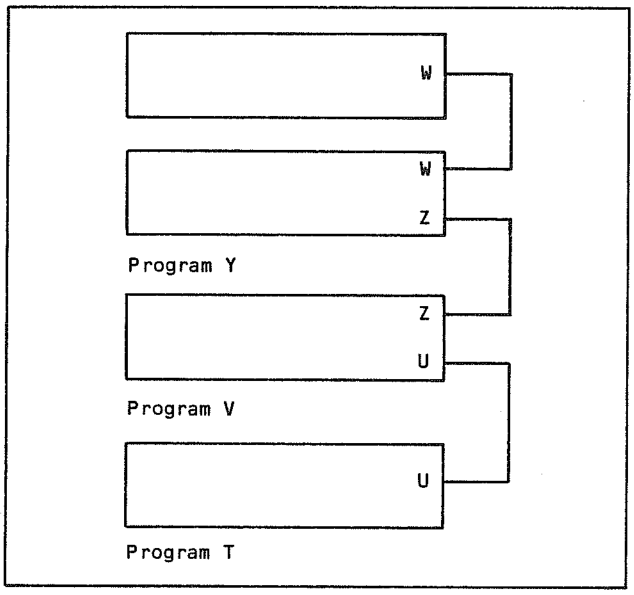


Figure 7-3. Example of Satisfying Externals

SEGLOAD CONTROL STATEMENT

The presence of a SEGLOAD statement (figure 7-4) in the loader control statement sequence specifies segmentation.

Other statements that can be used in the same sequence to augment the SEGLOAD statement are:

- name call
- LOAD
- LIBLOAD
- SLOAD
- EXECUTE
- NOGO
- LDSET

```
SEGLOAD(p1,p2,p3)

pi  Represents the following optional parameters
    in any order:

    I=lfn      Identifies the name of the local
               file containing the segload direc-
               tives. If this parameter is omit-
               ted, directives are assumed to be
               the next record on INPUT.

    B=lfn      Identifies the local file to receive
               segmented binary output. If this
               parameter is omitted, segments are
               written on ABS. If lfn is assigned
               to a magnetic tape device, the seg-
               mented binary output will be written
               at the current file position; other-
               wise, lfn will be rewound.

    LO=c1c2    Identifies list options for the seg-
               ment loader. The characters c1 and
               c2 control the directive list and
               the tree diagram. c1c2 values are:

               0          Neither tree diagram nor
                          directive list

               D          Directive list only

               T          Tree diagram only

               DT     )
               TD     }   Both directive list and
               omitted)   tree diagram

               If the LO parameter is omitted,
               the list options depend on the
               load map produced. If MAP(OFF) or
               LDSET(MAP=N) is specified, LO=0 is
               assumed; otherwise, LO=DT is
               assumed.
```

Figure 7-4. SEGLOAD Statement Format

SEGLOAD accepts programs from files as long as the following are true:

- No program spans record boundaries.
- Any embedded directive occurs in a separate record (an OVERLAY directive is legal and ignored if it is in a separate record).

SATISFY is not to be used with SEGLOAD; if present in the control statement sequence, it causes a nonfatal error.

Entry points defined by ENTRYC are treated as if they were defined by ENTRY.

LDSET options such as MAP, ERR, and PRESET work differently for SEGLOAD. Regardless of where they occur in the load sequence, they are treated as if they all appeared at the beginning of the sequence. If there is more than one option of a given type (for example, two MAP specifications), the last one encountered is the one that takes precedence. Because SEGLOAD does not use the files and libraries in a sequential manner, it cannot respond to changes of MAP or PRESET specifications from file to file. Similar conventions apply to the entire SEGLOAD operation.

If the sequence is completed with a NOGO statement, the segments are written on the binary output file specified on the SEGLOAD statement. The file name and entry points specified on the NOGO statement are ignored.

An EXECUTE statement, or a name call, builds the file as in the case of NOGO with the exception that loading and execution are initiated. If an entry point is not specified on EXECUTE, execution begins at the first entry point named on the END directive. If a file name call completes the SEGLOAD sequence, the file specified contains additional programs to be loaded as specified by directives.

An example of a segmented load is shown in figure 7-5. The loader reads the directives from the INPUT file. The loader then reads each of the load files A, B, G, H, X, U, and W, and copies each object program on these files with the exception of B. From B, only programs B1, B3, and B10 are copied.

```
Job statement
USER statement
CHARGE statement
MAP(ON)
SEGLOAD(B=LIB)
LOAD(A)
SLOAD(B,B1,B2,B3,B10)
LDSET(PRESET=NONE)
LOAD(G,H,X,U,W)
EXECUTE(PN,10,26)
7/8/9

    Directives

6/7/8/9
```

Figure 7-5. Example of a Segmented Load

As the loader reads each of the load files, it determines which external references are not matched by corresponding entry points and satisfies externals from the selected library set. It then determines the structure of the segmented program and generates absolute programs. The absolute programs are written on file LIB.

The EXECUTE statement results in the loading of the root segment. Parameters 10 and 26 are passed to the root segment, and execution is initiated at entry point PN, which must be in the root segment and named by the END directive.

DIRECTIVES

A SEGLOAD directive consists of four fields: label, verb, specification, and comment. A directive uses columns 1 through 72 of a line. A line with any character other than comma, slash, or asterisk in column 1 is considered to be the first or only line of a directive. The syntax for SEGLOAD directives is shown in table 7-1.

The following verbs are allowed in SEGLOAD directives:

- TREE

  Describes the segmentation structure

- INCLUDE

  Forces inclusion of a program into a segment

- LEVEL

  Prevents trees from conflicting in memory

TABLE 7-1. SEGLOAD DIRECTIVE SYNTAX

Specification	Meaning
Continuation	A comma in column 1 of a line indicates continuation of the previous line. The rest of the line is treated as if column 2 immediately followed column 72 or the special character of the last specification field of the preceding line. Any number of continuation lines is allowed. Any line without a comma or slash in column 1 terminates the directive or comment statement defined by the preceding lines.
Comment	If column 1 contains an asterisk, the directive is considered a comment. It is copied on the output file but has no effect on the type of output produced during execution.
Page eject	A line with a slash in column 1 causes a page eject in the listing of the SEGLOAD directives. This line has no other meaning.
Label field	The label field begins in column 1, or in column 2 if column 1 is blank, and is terminated by one or more blanks. If columns 1 and 2 are blank, the label field is considered empty. A label is one through seven alphanumeric characters. Characters other than comma, right and left parentheses, minus, and blank are considered alphanumeric. If a label exceeds seven characters, the superfluous characters are disregarded and a nonfatal error message is given.
Verb field	The verb field begins with the first nonblank character following the label field. If the label field is empty, this field begins with the first nonblank character of the line. This field must contain one of the legal directive verbs described in this section. One or more blanks terminates this field.
Specification field	The specification field begins with the first nonblank character following the verb field. A specification consists of one or more subfields, and possibly one or more of the following special characters: , - ( ) A subfield contains one through seven characters; however, any of the special characters terminates a subfield. If a subfield exceeds seven characters, additional characters are disregarded and a nonfatal error is issued. If a subfield is to contain any of the special characters , - ( ) or has a dollar sign ($) as its first character, the subfield must be written as a literal; that is, a character string delimited by dollar signs. Within a literal, each dollar sign is represented by two adjacent dollar signs in addition to delimiters. For example, the program name COS)$ is written $COS)$$$. The specification field is empty if only blanks follow the verb to the last column of the directive. Although directive syntax does not restrict the contents of subfields, certain directives require that the entries be the names of programs, entry points, and common blocks, and will be subject to the requirements for these symbols.
Comment field	The comment field begins after a specification field special character followed by one or more blanks. The specification field must be terminated by a special character and one or more blanks for the comment field to be recognized.

- GLOBAL

  Renders the labeled common block addressable by all segments that can coexist with the common blocks

- EQUAL

  Makes common block names synonymous

- COMMON

  Renders the labeled common block addressable by all segments

- END

  Signifies the end of directives and, optionally, specifies entry points at which execution can begin

Directives can be in any order preceding the END directive.

TREE DIRECTIVE

The TREE directive (figure 7-6) organizes segments into tree structures.

The pseudo-algebraic expression in the specification field denotes a program structure that, when viewed in time-memory coordinates, resembles a tree. The following examples illustrate how expressions are diagrammed.

Figure 7-7 shows how expressions are diagrammed. The expression A-B is diagrammed in example 1. As shown in the diagram, segments A and B can be in memory at the same time.

In example 2 the expression A,B is diagrammed. Segments A and B cannot be in memory at the same time. The comma used in this expression cannot be used outside of parenthesis.

The expression A-B-(C,D) is diagrammed in example 3. Either segments A, B, and C, or segments A, B and D can be in memory at the same time. Segments C and D cannot be in memory at the same time.

The expression A-(B-C,D), which illustrates the precedence of the hyphen over the comma is shown in example 4. Either segments A, B, and C, or segments A and D can be in memory at the same time.

The expression A-(B,D-(D,E)) is diagrammed in example 5. The tree structure produced by this expression can be thought of as a combination of two trees; segments A, B, and C with the expression A-(B,C), and segments C, D, and E with the expression C-(D,E), where segment C is the same in both expressions. Segment names cannot be duplicated; therefore assigning a separate tree name (for example Z) to the tree formed by segments C, D, and E prevents segment name duplication. Figure 7-8 shows the tree directives that could be used if a separate tree name was assigned.

Label	Verb	Specification
tname	TREE	expression

tname	Optional name assigned to the tree structure by which the tree can be referenced. Directives referring to this name in the expression can precede or follow the TREE directive assigning the name. Tree names must differ from actual segment names. An implicit INCLUDE of the form segname INCLUDE segname is made for each segment name in a TREE directive.
expression	A character string composed of segment names and/or tree names linked by the operators. The operators and their significance are:

-	Resembles an AND; components on the left of the hyphen can coexist with components on the right. The hierarchy moves from left to right with the root segment as the leftmost possible component of a chain.
,	Resembles an EXCLUSIVE OR; each comma indicates mutually exclusive residence. That is, components separated by comma cannot coexist.
()	Algebraic grouping; each pair of parentheses follows a hyphen and delineates a subset separated by commas. A subset consisting of a single component need not be enclosed by parentheses. In the absence of parentheses, the hyphen takes precedence over the comma.

Figure 7-6. TREE Directive Format

The following rules apply to TREE directive expressions:

- Segment and tree names must be unique. Duplication of names in a segmentation program results in a fatal error.
- One or more TREE directives can be used in a segmentation program.
- Segment and tree names can be used only once in TREE directives. (A tree name appears once in the label field of one TREE directive and once in the specification field of another TREE directive.) Multiple use of a name results in a fatal error.

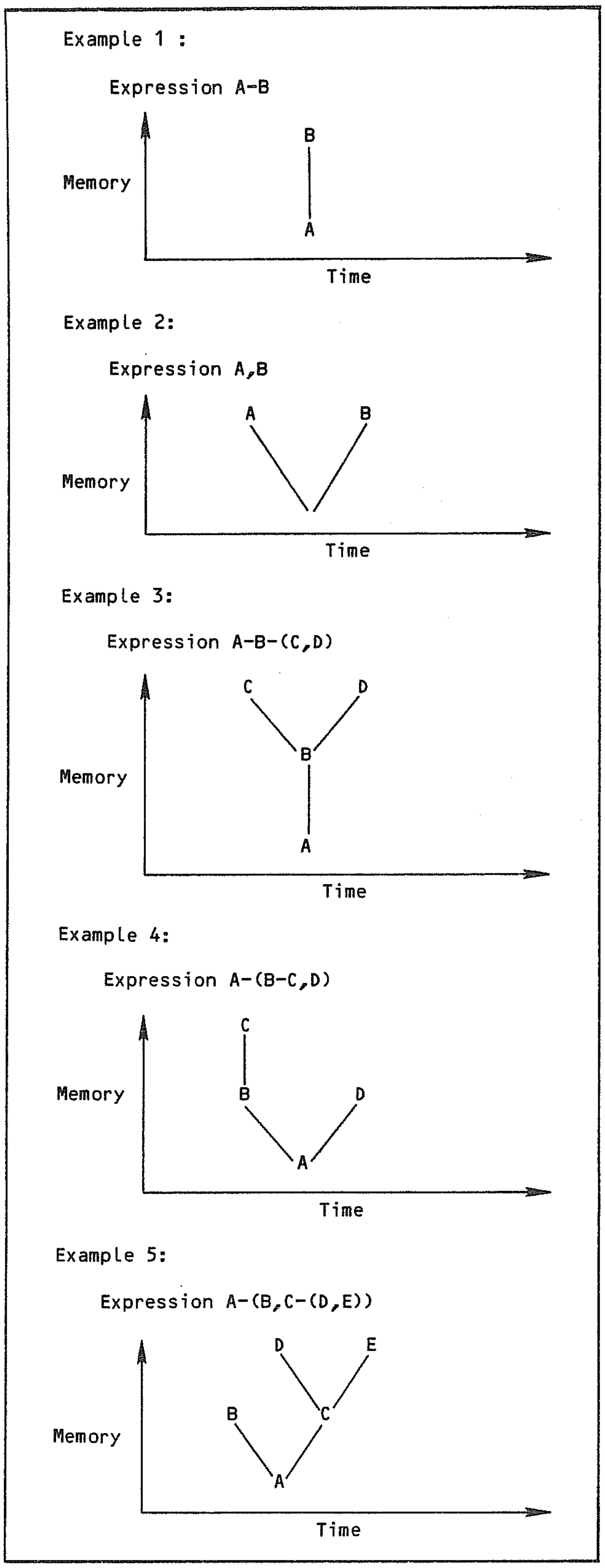


Figure 7-7. Diagram of Sample Expressions Used in the Specification Field of the TREE Directive

Label	Verb	Specification
Z	TREE	C-(D,E)
	TREE	A-(B,Z)

Figure 7-8. Example TREE Directive

- A tree at any level must begin with one segment. Use of an expression such as A,B or A-B,C is illegal when not enclosed in parentheses and preceded by one or more segment names.

The examples of the TREE directives shown in figure 7-9 show how TREE directives can be grouped. In example 1, segments A through O are organized into a tree structure by one TREE directive. In example 2, segments A through O are organized into a tree structure by a group of TREE directives. (The label PLUM is given to the tree, but no label is necessary if the tree is not to be gathered into a larger tree by another TREE directive.) In example 3, segments A through M are gathered into one tree structure; in example 4 these segments are gathered into a group of tree structures.

NOTE

Where label names are used in a TREE directive to combine secondary trees into a larger tree, the label names cannot be used as in-line items followed by a hyphen. A label name of a secondary tree must be the final item in a given branch of the primary tree. A tree cannot be contained inside another tree but must be appended as a terminating element.

INCLUDE DIRECTIVE

The INCLUDE directive (figure 7-10) forces inclusion of object programs into a specific segment, thus overriding the assignment rules (refer to Rules for Assignment in this section). It allows duplicate copies of object programs to be placed in segments.

The TREE directive illustrated in figure 7-11 defines a tree in which segments A, C, and F refer to a subroutine named SUB. Under the assignment rules, SUB is assigned to segment A because it is the nearest common ancestor of A, C, and F.

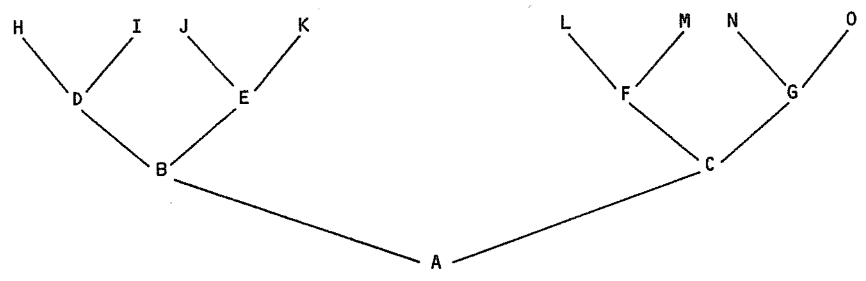
Addition of the directives in figure 7-12 (either preceding or following the TREE directives) forces copies of SUB into segments C and F. References to SUB by segments A and C are satisfied by the copy of SUB in C. References to SUB by object programs in segment F are linked to the copy of SUB loaded with segment F. Segments C and F include object program SUB as a minimum. If the longest branch is A-D, this procedure reduces the amount of field length required for program execution by eliminating SUB from segment A.

Example 1:

Label	Verb	Specification
PLUM	TREE	A-(B-(D-(H,I),E-(J,K)),C-(F-,(L,M),G-(N,O)))

Example 2:

Label	Verb	Specification
P	TREE	D-(H,I)
Q	TREE	E-(J,K)
R	TREE	F-(L,M)
S	TREE	G-(N,O)
T	TREE	B-(P,Q)
U	TREE	C-(R,S)
PLUM	TREE	A-(T,U)



Example 3:

Label	Verb	Specification
PEAR	TREE	A-(B-E-F-(G-J-(K,L,M),H,I),C,D)

Example 4:

Label	Verb	Specification
P	TREE	G-J-(K,L,M)
Q	TREE	F-(P,H,I)
R	TREE	B-E-Q
PEAR	TREE	A-(R,C,D)

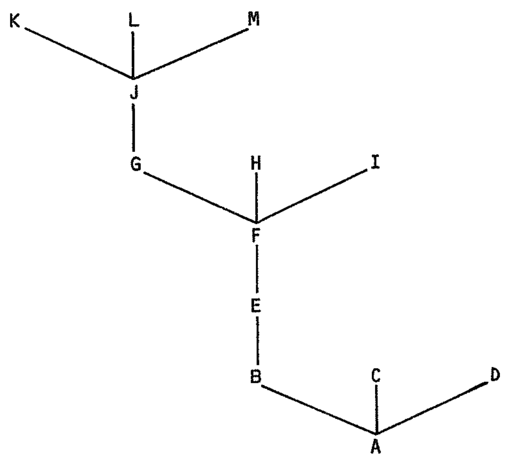


Figure 7-9. Grouping of TREE Directive

Label	Verb	Specification
segname	INCLUDE	$program_1,\ldots,program_n$

segname	Name of the segment in which the named object programs are to be included. If segname is omitted, programs are included in the root segment. The segment must be named on a TREE directive; otherwise, a nonfatal error occurs. The program named segname is implicitly included in the segment named segname only if the program named segname does not appear in the specification field of another INCLUDE directive; the program named segname must be explicitly listed if it appears in the specification field of another INCLUDE directive.
$program_i$	Names of object programs to be included. If any $program_i$ is not found in any load file or library, a nonfatal error occurs.

Figure 7-10. INCLUDE Directive Format

Label	Verb	Specification
LAB	TREE	B-(E,F)
	TREE	A-(LAB,C,D)

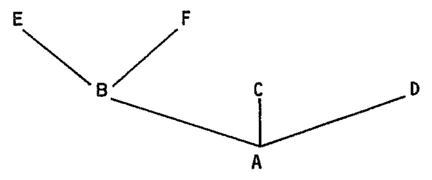


Figure 7-11. Example TREE Directive Showing Assignment of Subroutine SUB

Label	Verb	Specification
C	INCLUDE	SUB
F	INCLUDE	SUB

Figure 7-12. Example of the INCLUDE Directive

The loader attempts to satisfy an external by searching the program entry point list from the bottom of a tree to the top. In the example, if SUB is included in segment B as well as segment F, the reference to SUB by object programs in segment F are linked to SUB loaded in B.

LEVEL DIRECTIVE

The LEVEL directive (figure 7-13) divides memory and allows trees to be located so that they are spatially independent and never overlay each other. Entries in either the label or specification fields, if present, are ignored.

Label	Verb	Specification
	LEVEL	

Figure 7-13. LEVEL Directive Format

Rules for LEVEL:

- In the absence of LEVEL directives, all segments are in a single tree structure.
- Each occurrence of a LEVEL directive defines a new ground level for trees unless the group of directives divided by the LEVEL directive does not contain any TREE directives. If no TREE directives precede the first LEVEL directive, it is ignored. If no TREE directives occur between two LEVEL directives, the LEVEL directive following the treeless group is ignored.

An example of the LEVEL directive is shown in figure 7-14. It is possible for any subprogram in a level to reference any segment in higher or lower levels. For example, in level 0, A can reference segment F in level 1, or segment U in level 2. If A references F, loading of segments above A and below F is not forced. This is possible because the tree structure is independent between levels, and there is no downward dependency between the programs in level 1 and those in level 0. Within a level and within a tree, program dependency is observed. For example, if A references an entry point in program H in level 1, programs G and F are also loaded in level 1.

Another example of the LEVEL directive is shown in figure 7-15. In this example, loading of one or more segments on one level does not disturb the segments already loaded on any other level. Within a level, the loading of a segment causes its related lower order segments in its tree to be loaded or remain loaded, and causes all other segments in the same level to become or remain unloaded. Results of loading one segment can be tabulated, on the status of any other segment, as shown in table 7-2.

There is a row in the table for each segment to be loaded, with the segment name at the left end. For each segment that might be affected, there is a column with the segment name at the top.

Label	Verb	Specification
SUB	TREE	C-(D,E)
	TREE	A-(B,SUB)
	LEVEL	
ALPHA	TREE	M-(N,O)
	TREE	K-(L,ALPHA)
BETA	TREE	G-(H,I,J)
	TREE	F-BETA
	LEVEL	
	TREE	U-(V,W)
	TREE	P-Q-GAMMA
GAMMA	TREE	R-(S,T)

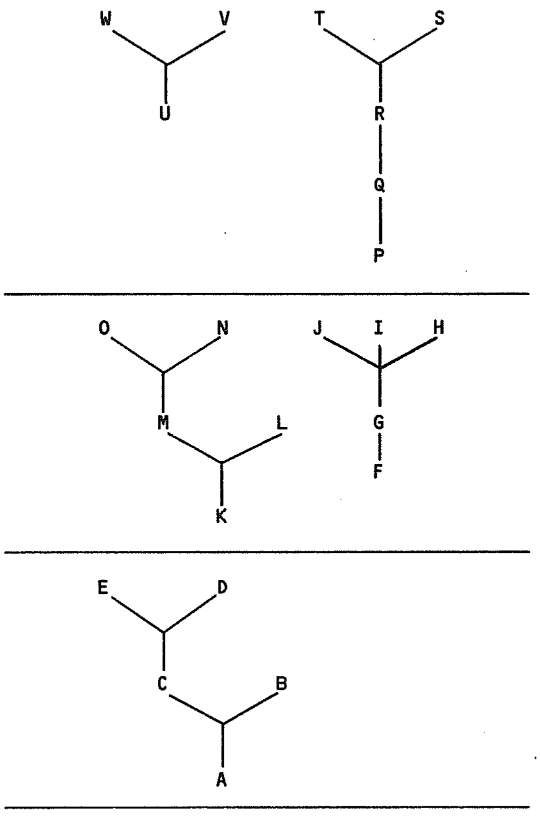


Figure 7-14. Example of the LEVEL Directive

Label	Verb	Specification
	TREE	A-B-(C,D)
	LEVEL	
	TREE	E-F-G
	TREE	H-I-(K,J-L)
	LEVEL	
	TREE	M
	TREE	N
	TREE	P .
	END	

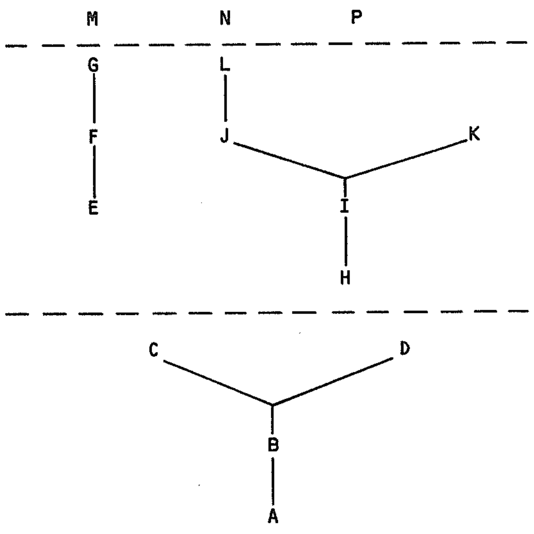


Figure 7-15. Example of Several Segments Loaded at the Same Level

GLOBAL DIRECTIVE

Labeled common, unlike blank common, is normally addressable only from the segment within which it is defined. It is normally destroyed when its segment is overwritten by another segment. The GLOBAL directive (figure 7-16) renders a labeled central memory common block addressable by all segments other than those that actually overwrite it, and optionally preserves and restores its contents.

A global labeled common block is allotted the greatest length specified for it by any programs

TABLE 7-2. SEGMENT STATUS USING LEVEL DIRECTIVE

Segment Loaded	Segment Affected														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P
A	+	–	–	–	0	0	0	0	0	0	0	0	0	0	0
B	+	+	–	–	0	0	0	0	0	0	0	0	0	0	0
C	+	+	+	–	0	0	0	0	0	0	0	0	0	0	0
D	+	+	–	+	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	+	–	–	–	–	–	–	–	0	0	0
F	0	0	0	0	+	+	–	–	–	–	–	–	0	0	0
G	0	0	0	0	+	+	+	–	–	–	–	–	0	0	0
H	0	0	0	0	–	–	–	+	–	–	–	–	0	0	0
I	0	0	0	0	–	–	–	+	+	–	–	–	0	0	0
J	0	0	0	0	–	–	–	+	+	+	–	–	0	0	0
K	0	0	0	0	–	–	–	+	+	–	+	–	0	0	0
L	0	0	0	0	–	–	–	+	+	+	–	+	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	+	–	–
N	0	0	0	0	0	0	0	0	0	0	0	0	–	+	–
P	0	0	0	0	0	0	0	0	0	0	0	0	–	–	+

+ Indicates the segment becomes or remains loaded

– Indicates the segment becomes or remains unloaded

0 Indicates the segment remains unchanged

that name it. (This is also the rule for blank common.) The size of a nonglobal labeled common block is the greatest length specified for it by programs in that segment.

Labeled common blocks assigned to ECS can be specified as global with the GLOBAL directive. But -SAVE would be superfluous, as ECS common blocks never overlay each other and are never written out on the SAVE file. A nonfatal error message informs the user that the given central memory block was not encountered.

Label	Verb	Specification
segname	GLOBAL	bname_1,...,bname_n-SAVE

segname	Optional segment name; it identifies the segment that owns the common blocks declared as global labeled common. Ownership means that when the owning segment is legally unloaded, perhaps because of the loading of some conflicting segment, the owned global common blocks are considered overwritten and unavailable until the owning segment is again loaded. Whether or not the contents of the blocks are preserved depends on the presence of -SAVE on the directive. A fatal error occurs if segname is specified but does not occur in any TREE directive. A global common block can be used as an ordinary labeled common block by programs in the owning segment. These programs can contain instructions that address the block and loader tables that preset data in the block at load time. Programs in other segments can contain instructions that address the block, but they must not attempt to preset data at load time. An attempt to do so results in a nonfatal error. If segname is omitted, the root segment is the owner of the common blocks, in which case the use of -SAVE is superfluous.
bname_i	Names of labeled central memory common blocks that are to be addressable from any segment. A common block name can be defined by more that one segment; if a block name does not appear on the right side of a GLOBAL directive, it designates a different block for each segment in which it appears. If it does appear in a GLOBAL directive, it designates a single block. The user should be careful when using CYBER Record Manager or compiler object library routines that contain common blocks as separate segments. If such a common block is referenced by more than one library subprogram, it should be declared GLOBAL by the user; otherwise, results are unpredictable. Common block names can be the same as segment, entry point, program, and tree names.
-SAVE	Optional; if present, the contents of the owned global block are saved on a scratch file whenever the owning segment becomes unloaded. The contents of the blocks are then restored when the segment is reloaded.

Figure 7-16. GLOBAL Directive Format

In figure 7-17 common blocks A, B, C, D, and E all are to belong to segment Q as global blocks. Only B and D need to be saved and restored when Q is unloaded and reloaded.

Label	Verb	Specification
Q	GLOBAL	B,D-SAVE
Q	GLOBAL	A,C,E

Figure 7-17. Example of the GLOBAL Directive

EQUAL DIRECTIVE

The EQUAL directive (figure 7-18) equates names of common blocks. The blocks named in the specification field do not have to be declared as global.

Label	Verb	Specification
bname_g	EQUAL	bname_1,...,bname_n

bname_g	The name of the global common block named in the specification field of a GLOBAL directive. The GLOBAL directive need not precede the EQUAL directive. If bname_g is not named on a GLOBAL directive, the directive causes a nonfatal error message.
bname_i	Synonyms of bname_g.

Figure 7-18. EQUAL Directive Format

COMMON DIRECTIVE

Labeled common, unlike blank common, is normally addressable only from the segment within which it is defined. It is normally destroyed when its segment is overwritten by another segment. The COMMON directive (figure 7-19) renders a labeled central memory common block addressable by all segments and preserves and restores its contents. The length of such a block is the greatest length specified for it by any programs that declare it.

Labeled common blocks assigned to ECS can be specified as COMMON. If a block is specified as both COMMON and GLOBAL, the GLOBAL specification will override the COMMON specification.

Label	Verb	Specification
	COMMON	$bname_1,\ldots,bname_n$

$bname_i$	Names of labeled central memory common blocks that are to be addressable from any segment. If no block names are specified, then all labeled central memory common blocks are affected. If GLOBAL is also specified, it will override the COMMON specification.

Figure 7-19. COMMON Directive Format

END DIRECTIVE

The END directive (figure 7-20) signals the end of SEGLOAD directives and, optionally, names the entry point where execution is to begin. Any subsequent statements are assumed to be comments.

If the END directive is omitted, the loader supplies the directive upon reading end-of-record. If the END directive is either omitted or does not name an entry point, the execution address and entry point name are derived from the last transfer address encountered in the root segment.

NOTE

Execution must begin at an entry point in the root segment.

Label	Verb	Specification
	END	$eptname_1,\ldots,eptname_n$

$eptname_i$	Optional entry point names in the root segment at which execution can begin. A loader control statement that causes execution must name one of these entry points if it names an entry point. Otherwise, execution begins at $eptname_1$.

A label field symbol, if present, is ignored.

Figure 7-20. END Directive Format

CAPSULES

Capsules are a collection of one or more object programs bound together in a special format to allow quick loading by executing programs. Capsules can also be loaded statically as input to basic loads, overlay generation loads, segment generation loads, and all other situations where standard relocatables are allowed. When capsules are used as input to a normal load, they are treated as any other relocatable object programs.

Capsules that are called into memory by executing programs are loaded by the Fast Dynamic Loading (FDL) facility. The FDL facility loads the capsules in a similar manner to user call loading (section 4), in the sense that both involve the loading of relocatable code at an arbitrary address. The FDL facility, however, requires specialized input (capsules), has a very limited capability, and is more difficult to use. The advantage of the FDL facility is that it requires less memory, is faster, and provides an unloading capability. Before a capsule can be loaded by FDL, the capsule must be generated and placed in a library.

Each capsule is a member of a capsule group. The name of the group is arbitrary and is defined when the capsule is generated. Groups are used as a means of associating related capsules with each other. No relationship exists between groups and libraries; a capsule group can be split among several libraries, and a library can contain several groups.

When an executing program wants to load a capsule, it calls an FDL subroutine, specifying a group name and a capsule name. The subroutine obtains storage for the capsule from the Common Memory Manager (CMM). It then reads the capsule into storage, relocates it, and possibly links it to other capsules. The capsule is then ready to be used by the calling program. When the calling program is through with the capsule, it can call the FDL subroutine again, this time to unload the capsule. Unloading the capsule frees memory formerly occupied by the capsule for other uses.

GENERATING CAPSULES

Capsule generation is signaled by the presence of one or more CAPSULE control statements or object directives in a load sequence. Each CAPSULE statement can be preceded by a GROUP statement that specifies the name of the capsule group to be generated.

Multiple occurrences of GROUP and CAPSULE statements are allowed, although normally a single GROUP statement followed by a single CAPSULE statement is sufficient. The GROUP and CAPSULE statements can be placed either at the beginning of the first file loaded or in the control statement stream. If the statements are placed on the load file, they must all precede the first program loaded. Placing the directives on the load file has the advantage of gathering all the information necessary for capsule generation (structuring information plus object programs) on a single file.

If the GROUP and CAPSULE statements are placed in the control statement stream, there is more flexibility because it is easier to change the statements and, hence, the structure. If both control statements and load file directives are provided, the control statement specifications are used. This provides a temporary means for overriding load file directives until they can be changed.

GROUP STATEMENT

The GROUP statement (figure 8-1) specifies the name of a capsule group. It applies to the capsules specified by the CAPSULE statement following the GROUP statement, and remains in effect until another GROUP statement is encountered.

```
GROUP(groupname)


groupname   A name of one to seven characters.
            Any valid program name acceptable
            to the loader can be used.
```

Figure 8-1. GROUP Statement Format

If the GROUP statement is omitted, the name of the first capsule generated becomes the group name.

CAPSULE CONTROL STATEMENT

The CAPSULE statement (figure 8-2) specifies the components of the capsules to be generated.

CAPSULE(pname$_1$,...,pname$_n$)

pname$_i$	The names of programs from the load file, listed in order, that signal the start of a new capsule. These programs must be loaded by LOAD or SLOAD statements; otherwise, they are not checked and capsule generation is neither initiated nor terminated.

Figure 8-2. CAPSULE Statement Format

Each capsule contains those programs specified by CAPSULE statements, plus any additional programs obtained by searching the library set to satisfy externals. This can be altered through use of the LDSET options OMIT, SUBST, USE, and USEP. SYSLIB is not searched by default for capsule and/or OVCAP generation. A LIBRARY or LDSET statement will cause the loader to search SYSLIB.

All programs on load files specified on LOAD or SLOAD requests used in encapsulation load sequences should contain Prefix tables. When the file is being read, the program name is extracted from the Prefix table to determine if the capsule should be initiated or terminated.

Capsules can contain only central memory text; ECS text is not allowed. All fields requiring relocation or linking must be 18 bits wide and aligned with bit 30, 15, or 0 being the rightmost bit of the field.

The load sequence must terminate with a NOGO statement because capsules cannot be executed directly. If a file name is specified on the NOGO statement, the capsules are written to the specified file; otherwise, they are written to file ABS.

All common blocks, including blank common, are allocated local to the current capsule. Common blocks can be shared by the various routines within a capsule; but if the routines in two different capsules declare common blocks of the same name, each capsule has its own copy of the common block and there is no communication between the two.

Some examples of capsule generation are shown in figure 8-3.

The files used in the examples are the following:

- FILE1 contains, in the order specified, programs A, B, C, D, E, F, and G.

- FILE2 contains, in the order specified, programs H, I, J, K, L, M, N, and O.

- FILE3 contains, in the order specified, programs P, Q, R, S, T, U, V, W, X, Y, and Z.

In example 1 of the figure, the group named MYGROUP contains one capsule, A. Capsule A contains programs A through Z and is written to file MYLFN.

In example 2, the group name and the capsule name are the same because no GROUP statement is present. Capsule M contains programs M through Z and is written to file ABS because no file name is specified on the NOGO statement.

In example 3, group AEI contains capsules A, E, and I. Capsule A contains programs A through D; capsule E contains programs E through H; and capsule I contains programs I through N. Group OU contains capsules O and U. Capsule O contains programs O through T, and capsule U contains programs U through Z. The capsules are written to file YOURLFN.

In example 4, group VOWELS contains one capsule, A. Capsule A contains programs A, E, I, O, and U. The capsule is written to file ABS.

Example 1:

```
GROUP(MYGROUP)
CAPSULE(A)
LOAD(FILE1,FILE2,FILE3)
NOGO(MYLFN)
```

Example 2:

```
CAPSULE(M)
LOAD(FILE1,FILE2,FILE3)
NOGO.
```

Example 3:

```
GROUP(AEI)
CAPSULE(A,E,I)
GROUP(OU)
CAPSULE(O,U)
LOAD(FILE1,FILE2,FILE3)
NOGO(YOURLFN)
```

Example 4:

```
GROUP(VOWELS)
CAPSULE(A)
SLOAD(FILE1,A,E)
SLOAD(FILE2,I,O)
SLOAD(FILE3,U)
NOGO(ABS)
```

Figure 8-3. Examples of Capsule Generation

ENTRY POINT CONTROL

Capsules are linked to each other and to the calling program through the use of entry points and external references. An external reference generated by one of the programs comprising the capsule becomes an external reference of the capsule if it is not satisfied by an entry point of one of the other programs in the capsule.

The entry points of a capsule are normally those that are defined by programs within the capsule and are not referenced within the capsule. This set can be altered through the use of the following LDSET options:

- LDSET(EPT=eptname$_1$/.../eptname$_n$)

- LDSET(NOEPT=eptname$_1$/.../eptname$_n$)

- LDSET(NOEPT)

The EPT option specifies one or more entry points that are to appear as entry points of the capsule, regardless of whether the entry points are referenced within the capsule. The NOEPT option prevents the listed entry points from becoming entry points of the capsule even if they are not referenced within the capsule.

The NOEPT request, which lists no entry points, specifies that the only entry points are to be those explicitly mentioned by the EPT request and an entry point whose name is the same as the capsule name (if any).

DYNAMIC LOADING AND UNLOADING OF CAPSULES

The loading and unloading of capsules is performed by a small subroutine that resides on SYSLIB. This subroutine references CMM and causes a capsule to be either loaded or unloaded, depending on one of two entry points referenced by the calling program.

LOADING CAPSULES

The entry point FDL.LDC is called to load a capsule. The following parameters should be passed to FDL.LDC by the calling program:

- The group name and capsule name specifying the capsule to be loaded
- An optional estimate of the number of capsules in the given group
- An optional list of libraries to be searched for the capsule
- An optional list of addresses of PASSLOC and/or ENTRY tables for use in linking

The exact calling sequence for loading capsules is shown in figure 8-4.

When FDL.LDC is called to load a capsule from a previously unreferenced group, a directory for the entire group is built and used for all subsequent loading from the group. The directory is built by searching first the global library set and then the list of libraries passed with the call, if applicable. The library list is used only the first time a capsule group is accessed.

If the number of capsules in the group is known, this value should be passed to FDL.LDC. This value is used to determine the amount of memory to allocate for the directory. If no value is passed, a default is used. In either case, if the original value is found to be too small, FDL automatically keeps trying to load the directory, each time with a larger value, until it succeeds.

PASSLOC and/or ENTRY tables serve two purposes:

- Provide addresses to be used in satisfying external references of capsules
- Obtain addresses of entry points within capsules

The ENTHDR and ENT macros (described later in this section) can be used to generate ENTRY tables. The LDREQ macro (see section 4) can be used to generate PASSLOC tables. Entries in PASSLOC and/or ENTRY tables that are to be used by FDL must be in ascending display code order.

When FDL.LDC loads a capsule, it attempts first to satisfy external references by checking the PASSLOC and/or ENTRY tables specified in the call. If it finds the name of any corresponding entry points in a PASSLOC and/or ENTRY table with nonzero satisfied addresses, these addresses are used to satisfy the external references. Any remaining external references are satisfied, if possible, by checking the entry points of all other currently loaded capsules within the same capsule group.

```
FDL.LDC (Load Capsule Entry Point)

Entry: (X1) = 42/group name, 18/size estimate.
       (X2) = capsule name.
       (X3) = library list address.
       (X4) = passloc/entry list address.

Exit:  (B1) = 1.
       (X6) = error code:
              0 = NO ERROR.
              1 = BAD LIBRARY LIST.
              2 = UNKNOWN GROUP NAME.
              3 = UNKNOWN CAPSULE NAME.
              4 = BAD CAPSULE FORMAT.
              5 = BAD PASSLOC/ENTRY FORMAT.
              6 = CAPSULE ALREADY IN MEMORY.
              7 = CAPSULE/OVCAP CONFUSION.

Saves: A0, X0, B2, B3, X5.

Calls: CMM.ALF, CMM.CSF, CMM.FRF,
       CMM.SLF, SYS=.
```

group name	The name of the capsule group, left-justified with zero fill.
size estimate	The number of capsules in the capsule group; zero if not known.
capsule name	The name of the capsule, left-justified with zero fill.
library list address	The address of a list of libraries. The list contains one library name in each word (left-justified with zero fill) and is terminated by a zero word. An address of zero indicates no library list is given.
passloc/entry list address	The address of a list of passloc/entry table addresses. The list contains the address of a passloc/entry table in the lower 18 bits of each word. The list is terminated by a zero word. This list must be in ascending display code order. An address of zero indicates that no passloc/entry list is given.

Figure 8-4. Calling Sequence for Loading Capsules

The entry points of a newly loaded capsule are checked against any remaining unsatisfied externals of currently loaded capsules in the same group, and any such externals are now satisfied, if possible. The PASSLOC and/or ENTRY tables are checked for entries matching the names of any of the entry points of the newly loaded capsules. If a matching PASSLOC and/or ENTRY entry has a zero or unsatisfied address, the address of the entry point is inserted. This is the only method by which the static code can reference the capsule.

When FDL.LDC returns control to the calling program, error codes are set as indicated in figure 8-4.

UNLOADING CAPSULES

The entry point FDL.ULC is called to unload a capsule. The following parameters should be passed to FDL.ULC by the calling program:

- The group name and capsule name specifying the capsule to be unloaded
- A list of addresses of PASSLOC and/or ENTRY tables (if any)

The calling sequence for unloading capsules is shown in figure 8-5.

```
FDL.ULC - UNLOAD CAPSULE.

Entry: (X1) = 42/group name, 18/ignored.
       (X2) = capsule name.
       (X4) = passloc/entry list address.

Exit:  (B1) = 1.
       (X6) = error code:
              0 = NO ERROR.
              2 = UNKNOWN GROUP NAME.
              3 = UNKNOWN CAPSULE NAME.
              4 = BAD CAPSULE FORMAT.
              5 = BAD PASSLOC/ENTRY FORMAT.

Saves: A0, X0, B2, B3, X5.

Calls: CMM.FRF.

group name      The name of the capsule group,
                left-justified with zero fill.

capsule name    The name of the capsule, left-
                justified with zero fill.

passloc/entry   The address of a list of passloc/
list address    entry table addresses. The list
                contains the address of a
                passloc/entry table in the lower
                18 bits of each word. The list
                is terminated by a zero word.

                This list must be in ascending
                display code order to ensure
                that entry points are properly
                filled in the list.

                An address of zero indicates
                that no passloc/entry list is
                given.
```

Figure 8-5. Calling Sequence for Unloading Capsules

The entry points and external references of the identified capsule are checked against entry points and external references of other currently loaded capsules in the group, and all links between the capsule to be unloaded and other currently loaded capsules are dissolved.

The address field in any PASSLOC and/or ENTRY table entries that matches one of the capsule's entry points in name and address is zeroed. When FDL.ULC returns control to the calling program, error codes are set as indicated in figure 8-5.

FREEING UNUSED MEMORY

The FDL.CGD and FDL.UGD entry points are available for freeing memory.

The entry point FDL.CGD is called to move all CMM blocks which contain group directories down to the lowest possible memory locations. If no such blocks are present, or if it is not possible to move them any lower, no action is taken. Use of this entry point is entirely optional; in cases of bad memory fragmentation, it may result in large savings of memory. There are no parameters to be passed to, nor any returned by FDL.CGD.

The calling sequence for compacting group directories is shown in figure 8-6.

```
FDL.CDG - COMPACT GROUP DIRECTORY.

Entry: None

Exit:  (B1) = 1.

Saves: A0, X0, B2, B3, X5.

Calls: CMM.CSF, CMM.FRF, CMM.SLF, MVE=.
```

Figure 8-6. Calling Sequence for Compacting Group Directories

The entry point FDL.UGD is called to free the memory containing the directory for a group in which all the capsules have been unloaded. The use of this entry point is entirely optional, but unless it is used, all group directories, once established, remain in memory throughout the job step. FDL.UGD requires that one parameter be passed to it which is the group name specifying the group directory to be unloaded. If the group has any capsules currently loaded, an error is generated.

The calling sequence for unloading group directories is shown in figure 8-7.

```
FDL.UGD - UNLOAD GROUP DIRECTORY.

Entry: (X1) = 42/GROUP NAME, 18/IGNORED.

Exit:  (B1) = 1.
       (X6) = ERROR CODE:
              0 - NO ERROR.
              2 - UNKNOWN GROUP NAME.
              10B - GROUP HAS CAPSULES.

Saves: A0, X0, B2, B3, X5.

Calls: CMM.FRF.

Group Name    The name of the capsule group,
              left-justified with zero fill.
```

Figure 8-7. Calling Sequence for Unloading Group Directories

USER LOADING OF CAPSULES

The user can optionally load a capsule and then call the Fast Dynamic Loader to relocate and link the capsule to PASSLOC and/or ENTRY tables. A capsule so loaded is not considered to be a member of a group. It is a stand-alone capsule which can only be linked to PASSLOC and/or ENTRY tables.

A user-loaded capsule can be linked or delinked depending on one of two entry points referenced by the calling program.

The calls to link and delink user-loaded capsules do not reference CMM and will function properly whether CMM is active or inactive.

LINKING USER-LOADED CAPSULES

The entry point FDL.REL is called to relocate and link a user-loaded capsule to PASSLOC and/or ENTRY tables. The following parameters should be passed to FDL.REL by the calling program:

- The address of the first word of the capsule
- A list of addresses of PASSLOC and/or ENTRY tables

The exact calling sequence for relocating and linking capsules is shown in figure 8-8.

```
RJ = XFDL.REL

Entry: (X2) = 42/0, 18/fwa.
       (X4) = passloc/entry list address.

Exit:  (B1) = 1.
       (X6) = error code:
              0 = NO ERROR.
              4 = BAD CAPSULE FORMAT.
              5 = BAD PASSLOC/ENTRY FORMAT.

Saves: A0, X0, B2, B3, X5.

fwa          The address of the first word of the
             capsule.

passloc/     The address of a list of passloc/
entry        entry table addresses.  The list con-
list         tains the address of a passloc/entry
address      table in the lower 18 bits of each
             word.  The list is terminated by a
             zero word.

             This list must be in ascending dis-
             play code order to ensure that entry
             points are properly filled in the
             list.
```

Figure 8-8. Calling Sequence for Linking and Relocating Capsules

DELINKING USER-LOADED CAPSULES

The entry point FDL.UNR is called to delink a user-loaded capsule from PASSLOC and/or ENTRY tables. The following parameters should be passed to FDL.UNR by the calling program:

- The address of the first word of the capsule
- A list of addresses of PASSLOC and/or ENTRY tables

The exact calling sequence for delinking capsules is shown in figure 8-9.

```
RJ = XFDL.UNR

Entry: (X2) = 42/0, 18/fwa.
       (X4) = passloc/entry list address.

Exit:  (B1) = 1.
       (X6) = error code:
              0 = NO ERROR.
              4 = BAD CAPSULE FORMAT.
              5 = BAD PASSLOC/ENTRY FORMAT.

Saves: A0, X0, B2, B3, X5.

fwa          The address of the first word of the
             capsule.

passloc/     The address of a list of passloc/
entry        entry table addresses.  The list con-
list         tains the address of a passloc/entry
address      table in the lower 18 bits of each
             word.  The list is terminated by a
             zero word.

             This list must be in ascending dis-
             play code order to ensure that entry
             points are properly filled in the
             list.
```

Figure 8-9. Calling Sequence for Delinking Capsules

A FDL.UNR does not unload the capsule; the user is responsible for unloading it (that is freeing the CMM block).

OVCAP

OVCAPs (overlay-capsules) are capsules that are logical extensions of (0,0) overlays. They are similar to a primary overlay in that they require the presence of a specific main overlay. Unlike primary overlays, any number of OVCAPs can be loaded or unloaded in any order and at any time.

OVCAPs are generated, loaded, and unloaded by the FDL and FOL facilities. For an OVCAP to be loadable, the (0,0) overlay must be generated with an FOL directory.

Each OVCAP must be associated with a (0,0) overlay. Common blocks and entry point names of the (0,0) overlay are used and linked into by the OVCAP. Entry points in the OVCAP must not duplicate entry points in the (0,0) overlay. Any common blocks that exist in the OVCAP, but not in the (0,0) overlay, are processed as being local to the OVCAP.

The following entry control statements can be used to control entry point determination for OVCAPs, just as for capsules:

- LDSET(EPT=eptname$_1$/.../eptname$_n$)
- LDSET(NOEPT=eptname$_1$/.../eptname$_n$)
- LDSET(NOEPT)

OVCAP restrictions, consistent with capsule restrictions, are as follows:

- ECS text is disallowed
- Nonstandard relocation is disallowed

The OVCAP load map is similar to the capsule load map, except that the OVCAP directive is also listed. (Only blocks local to the OVCAP are listed in the load map.)

The library set currently defined for the load sequence is searched at load completion time to satisfy externals that are currently unsatisfied.

OVCAP DIRECTIVE

The OVCAP directive (figure 8-10) specifies that an OVCAP is to be generated. The file on which the OVCAP programs are to be written must be the same one that the main overlay is on.

```
OVCAP.

OVCAP(lfn)


lfn   The file name on which the OVCAP is to
      be written; must be the same file on
      which the (0,0) overlay is written.

      If a file is specified on a NOGO
      statement, it overrides the file named
      on any OVCAP directive. If a file is
      specified on a previous directive but
      not on the current directive, then the
      previously specified file is used. If
      neither the directives nor a NOGO
      statement specified a file name, the
      file ABS is used.
```

Figure 8-10. OVCAP Directive Format

Like overlay directives, OVCAP directives are placed in the load file input stream. At least one OVERLAY directive, and all OVERLAY directives for the current overlay structure, must precede OVCAP directives.

The OVCAP uses the program name of the (0,0) overlay as the group name, and the name of the first program encountered after the OVCAP directive as the CAPSULE name in the 6000 table header for the OVCAP (see appendix D). Bit 18 of the 6000 header word is set to distinguish an OVCAP from a capsule.

The basic loader, user call loader, and segment loader cannot statically load an OVCAP. Any such attempts are diagnosed by a fatal loader error.

LOADING AND UNLOADING OVCAPS

OVCAPs are loaded by issuing a call to FDL.LOC and unloaded by a call to FDL.UOC. FDL.LOC and FDL.UOC have the same entry conditions, register convention, and processing capabilities as FDL.LDC and FDL.ULC (capsule loading and unloading), respectively, except that the group name may be zero. If a zero group name is specified, the first entry point name of the (0,0) overlay is used. A call to FDL.LOC causes the FOL directory table to be search for the OVCAP.

OVCAPs can link to supplied PASSLOC and/or ENTRY tables and to other OVCAPs that are loaded, but not to capsules. The calling sequence for loading OVCAPs is shown in figure 8-11; the calling sequence for unloading OVCAPs is shown in figure 8-12.

ENTRY TABLE GENERATION

The ENTHDR and ENT macros are used to generate ENTRY tables to be used by the Fast Dynamic Loader when loading or unloading capsules. These macros are not contained in the default system text (SYSTEXT) used by COMPASS. A separate text called LDRTEXT is available which contains the ENTHDR and ENT macros. A COMPASS call specifying that both texts are to be used is:

```
COMPASS(S=SYSTEXT,S=LDRTEXT)
```

The internal format of the ENTRY table is described in appendix D.

ENTHDR MACRO

The ENTHDR macro initializes a new ENTRY table by generating the table header word. The format of the ENTHDR macro is shown in figure 8-13.

ENT MACRO

The ENT macro enters into the table initialized by the ENTHDR macro a word containing an entry point name and address. The format of the ENT macro is shown in figure 8-14. The entry point names in successive ENT macros must be in ascending display code order. If they are not, an assembly error occurs.

```
FDL.LOC - LOAD OVERLAY-CAPSULE.

Entry: (X1) = 42/group name, 18/size estimate.
       (X2) = overlay-capsule name.
       (X3) = library list address.
       (X4) = passloc/entry list address.

Exit:  (B1) = 1.
       (X6) = error code:
              0 = NO ERROR.
              1 = BAD LIBRARY LIST.
              2 = UNKNOWN GROUP NAME.
              3 = UNKNOWN OVERLAY-CAPSULE NAME.
              4 = BAD OVERLAY-CAPSULE FORMAT.
              5 = BAD PASSLOC/ENTRY FORMAT.
              6 = OVERLAY-CAPSULE ALREADY IN
                  MEMORY.
              7 = CAPSULE/OVCAP CONFUSION.

Saves: A0, X0, B2, B3, X5.

Calls: FDL resident and FOL resident.

group name      The name of the OVCAP group,
                left-justified with zero fill.

                A value of zero indicates that
                the name of the first program
                entry point is to be used as the
                group name.

                The use of a zero value is not
                advised for general use because
                it requires that the first (or
                only) program entry point name
                match the name of the first pro-
                gram in the (0,0) overlay.  It is
                primarily intended for languages
                like FTN which always satisfy
                this requirement.

size estimate   The number of members in the
                OVCAP group; zero if not known.

overlay-        The name of the OVCAP, left-
capsule name    justified with zero fill.

library list    The address of a list of librar-
address         ies.  The list contains one
                library name in each word (left-
                justified with zero fill) and is
                terminated by a zero word.

                An address of zero indicates no
                library list is given.

passloc/entry   The address of a list of passloc/
list address    entry table addresses.  The list
                contains the address of a
                passloc/entry table in the lower
                18 bits of each word.  The list
                is terminated by a zero word.

                This list must be in ascending
                display code order to ensure that
                entry points are properly filled
                in the list.

                An address of zero indicates that
                no passloc/entry list is given.
```

Figure 8-11. Calling Sequence for Loading OVCAPs

```
FDL.UOC - UNLOAD OVERLAY-CAPSULE.

Entry: (X2) = overlay-capsule name.
       (X4) = passloc/entry list address.

Exit:  (B1) = 1.
       (X6) = error code:
              0 = NO ERROR.
              3 = UNKNOWN OVERLAY-CAPSULE NAME.
              4 = BAD OVERLAY-CAPSULE FORMAT.
              5 = BAD PASSLOC/ENTRY FORMAT.

Saves: A0, X0, B2, B3, X5.

Calls: FDL RESIDENT.

overlay-        The name of the OVCAP, left-
capsule name    justified with zero fill.

passloc/entry   The address of a list of passloc/
list address    entry table addresses.  The list
                contains the address of a
                passloc/entry table in the lower
                18 bits of each word.  The list
                is terminated by a zero word.

                This list must be in ascending
                display code order to ensure that
                entry points are properly filled
                in the list.

                An address of zero indicates that
                no passloc/entry list is given.
```

Figure 8-12. Calling Sequence for Unloading OVCAPs

Location	Operation	Variable Subfields
label	ENTHDR	

label	An optional label.

Figure 8-13. ENTHDR Macro Option Format

Location	Operation	Variable Subfields
label	ENT	eptname,addr

label	An optional label.
eptname	Entry point name.
addr	Optional; entry point address. If addr is not specified, then eptname will be used.

Figure 8-14. ENT Macro Option Format

STANDARD CHARACTER SETS

Control Data operating systems offer the following variations of a basic character set:

- CDC 64-character set
- CDC 63-character set
- ASCII 64-character set
- ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect throughout the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described above for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1; 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table (table A-1) are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC			ASCII		
	Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
00†	: (colon)††	8-2	00	: (colon)††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(	0-8-4	34	(	12-8-5	050
52	)	12-8-4	74	)	11-8-5	051
53	$	11-8-3	53	$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	≡	0-8-6	36	#	8-3	043
61	[	8-7	17	[	12-8-2	133
62	]	0-8-2	32	]	11-8-2	135
63	% ††	8-6	16	% ††	0-8-4	045
64	≠	8-4	14	" (quote)	8-7	042
65	↱	0-8-5	35	_ (underline)	0-8-5	137
66	∨	11-0	52	!	12-8-7	041
67	∧	0-8-7	37	&	12	046
70	↑	11-8-5	55	' (apostrophe)	8-5	047
71	↓	11-8-6	56	?	0-8-7	077
72	<	12-0	72	<	12-8-4	074
73	>	11-8-7	57	>	0-8-6	076
74	≤	8-5	15	@	8-4	100
75	≥	12-8-5	75	\	0-8-2	134
76	¬	12-8-6	76	¬ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

ERROR MESSAGES AND CODES B

Errors generated during load operations are described in this appendix according to type of error and type of loader operation. General errors occuring during loader sequences in job control statement streams are discussed first, followed by errors detected by the user call interface routines, errors detected by the segment loading facility, errors detected by the overlay loading facility, and errors detected during capsule and OVCAP operations.

LOADER SEQUENCE ERRORS

The following types of errors are diagnosed during loader sequence operations:

- Catastrophic errors abort the program immediately, without waiting for completion of any loader sequence in progress. These error diagnostics, shown in table B-1, are always written to the dayfile.
- Fatal errors abort the program as soon as any loader sequence in progress is completed. These error diagnostics are shown in table B-2.
- Nonfatal errors do not abort the job, although subsequent job operations which use the code that produced the diagnostic can abort the job. Nonfatal error diagnostics are also shown in table B-2.

Fatal and nonfatal errors are listed either on the load map or in the dayfile.

Errors are written to the load map if a load map is selected either implicitly or explicitly and the load is not an absolute load. (A dayfile message indicates that errors occurred.) Errors are written to the dayfile if no map is selected or the load is absolute.

Errors coded under 4000 are fatal errors; errors coded 4000 and above are nonfatal errors.

The letter C under the column headed Issued By indicates the error is detected by the control statement loader (includes basic loads and overlay, OVCAP, capsule, and segment generation loads). The letter U indicates the error is detected by the user call loader. CU indicates the error is detected by both.

On a user call load, one error code is returned in the LDREQ BEGIN block. (See appendix D.)

The following informative diagnostic message is written to the dayfile under the NOS operating system only:

```
LOAD SEQUENCE IGNORED
GIVING LAST CONTROL CARD BACK TO SYSTEM
```

This diagnostic usually occurs when a load sequence has been initiated by a load statement and the next statement in the sequence is a control statement that the loader assumes is a name call statement. (The statement might be an operating system control statement, or it could be a misspelled name call statement.) The loader searches for a local file with the given name. If the loader does not find the local file, it issues this diagnostic and returns the control statement to the operating system. The operating system then searches the central library directory and attempts to process the control statement. Any preceding load statements are ignored.

ERRORS DETECTED BY USER CALL INTERFACE

Errors detected by the user call interface are shown in table B-3. These errors are written to the dayfile and cause the job to abort.

The loader subroutines UCLOAD and PILOAD provide an interface between running programs and the user call loader. UCLOAD is called when the CMM parameter is absent from the LOADER macro call; PILOAD is called when the CMM parameter is present.

ERRORS DETECTED BY SEGRES

The errors detected by the segment loader resident, SEGRES, during initialization or execution of a segmented program are shown in table B-4.

The errors detected by the fast dynamic loading facility during dynamic capsule and OVCAP operations are shown in table B-5. All entry points used for these operations return error code values in register X6. A zero value in X6 after exit from the call indicates no error occured. A nonzero value indicates occurrence of the error corresponding to the return code shown in the table.

ERRORS DETECTED DURING OVERLAY LOADING

The errors detected during overlay loading for NOS and NOS/BE are discussed in the following paragraphs. Error processing is partially dependent on the operating system in use.

The errors detected by the fast dynamic loading facility during dynamic capsule and OVCAP operations are shown in table B-5. All entry points used for these operations return error code values in register X6. A zero value in X6 after exit from the call indicates no error occured. A nonzero value indicates occurrence of the error corresponding to the return code shown in the table.

NOS

The message LDR ERROR. indicates an error loading an overlay. A second message indicates the exact problem. The job is aborted.

The messages are documented in appendix B of the NOS reference manual, volume 1.

NOS/BE

The error codes listed in table B-6 are returned to the status field of the parameter area following a call to LDV to load an overlay. If the e bit is set in the LOADREQ request indicating automatic execution of the overlay, then no status is returned; but if the error is fatal, the job is aborted. The process is described in section 6.

Errors other than 4001 are also written to the dayfile.

Errors coded under 4000 are fatal errors; errors coded 4000 and above are nonfatal errors.

ERRORS DETECTED BY FDL

Error conditions detected by FDL (Fast Dynamic Loader) are normally returned in registers as described in section 8. However, if FDL detects an error which is unavoidable by the user, the following message is written to the dayfile and the job is aborted:

INTERNAL FDL ABORT - LD$\left\{\begin{matrix}D\\Q\end{matrix}\right\}$ ERR nn

where nn is the error code returned by LDD or LDQ. If this error occurs, a systems analyst should be notified.

ERRORS DETECTED DURING TRAP DIRECTIVE PROCESSING

The errors detected during processing of the TRAP directives are shown in table B-7.

If the TRAP control statement is in error, the message TRAP CARD PARAM ERROR, NO DEBUG DONE is written to the dayfile and the trap directives are ignored.

If errors are detected in the TRAP directives, they are listed on the TRAP output file and the message TRAP DIRECTIVE ERROR(S) is written to the dayfile. If no legal directives are found, the additional message NO TRAP DIRECTIVES FOUND, NO DEBUG DONE is also written to the dayfile.

ERRORS DETECTED BY TRAPPER DURING EXECUTION

The errors detected by TRAPPER at execution time are shown in table B-8.

The dayfile message ERROR DETECTED BY TRAPPER indicates an error discovered during execution of a trapped program. If the problem is related to a particular directive, the message DIRECTIVE NOT PROCESSED - ID=name follows to give the label of the directive in error.

TABLE B-1. CATASTROPHIC ERRORS DETECTED BY THE LOADER

Message	Significance	Action
FDL ERROR x. ABORT.	See figure 8-4, section 8.	
LOADER ABORT. ECS LIMITS ERROR ON USER CALL	ECS parameters on the LDREQ BEGIN call are in error.	Correct the program.
LOADER ABORT. ILLEGAL CONTROL STATEMENT	Illegal characters or parameters of more than 7 characters were specified on loader control statement.	Correct the control statement.
LOADER ABORT. ILLEGAL REQUEST TABLE	On a call to the loader of the type LOADREQ - request for a basic load (section 4), fwa$\leq$1 or lwa+1>54_8.	Correct the program.
LOADER ABORT. ILLEGAL TRANSFER ADDRESS	Transfer address is negative or undefined.	Correct transfer address specification.
LOADER ABORT. INSUFFICIENT FL FOR LOAD	Not enough field length is available to initiate loading.	Provide at least 14000_8 words.
LOADER ABORT. LOADER I/O ERROR nn	Loader aborted. I/O error nn was returned by CIO.	Rerun the job. If problem persists, follow site-defined procedures for reporting software errors or operational problems.
LOADER ABORT. NO TERMINATOR IN ABOVE LOAD SEQUENCE	The load sequence has no completion statement. Completion statements are NOGO, EXECUTE or name call statements.	Correct the load sequence. Use a procedure file if attempting to enter load control statements from a terminal under NOS/BE.

TABLE B-1. CATASTROPHIC ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action
LOADER ABORT. SEG+PROGRAM+COMMON BLK TOTAL GT 8192	The load sequence consists of a larger total number of segments, programs, and common blocks than allowed by the loader.	Re-structure the load.
LOADER ABORT. SEGLOAD INPUT FILE EMPTY OR MISPOSITIONED	The file specified as the source of SEGLOAD directives is empty, or the SEGLOAD statement is in the wrong position.	Correct the job structure and rerun. Rewind the file specified as the source of the SEGLOAD directives.
LOADER ABORT. SYSTEM ERR, EOI ON LIBRARY - name	Bad library format.	Follow site-defined procedures for reporting software errors or operational problems.
LOADER ABORT. SYSTEM ERROR LOADING - name	The loader cannot load one of its own overlays.	Follow site-defined procedures for reporting software errors or operational problems.
LOADER ABORT. ---LOADER I/O ERROR xx FILE---lfn	CIO returned error code xx after a file action request by LOADER on file lfn.	See NOS Volumne 4 reference manual or NOS/BE System Programmer's reference manual for description of CIO error codes. Rerun job. Follow site-defined procedures for reporting software errors or or operational problems.
LOADER ABORT. <name> NOT IN LIBRARY SET	The user library specified by <name> is not present on the library set. This library might have been returned prior to the start of the load sequence.	Correct the job structure and rerun.

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER

Message	Significance	Action	Issued By
100 INSUFFICIENT FL FOR LOAD	For a control statement load, the loader could not obtain enough memory to complete the load.	Increase the limit if set by the CM parameter on the job card or by (NOS) MFL statement. If the program is very large consider breaking it up into into segments or overlays.	CU
	For a user call load, the caller did not provide (either explicitly or implicitly) enough memory for the loading operation.	Modify the calling program to provide more memory; or, if the limit is set implicitly by the job field length, run the job with more memory assigned.	
101 EMPTY LOAD	No programs were loaded. This usually is either the result of another error (such as 220) or of the accidental omission of LOAD, LIBLOAD, or SLOAD from the load sequence.	Correct the load sequence or other problem.	C
102 NO TRANSFER ADDRESS	One main program must occur in each load (or overlay) to provide a starting point for execution.	For COMPASS, add an entry point name to the variable field of one END statement. For other languages, make sure a main program is included in the load (or each overlay).	CU
103 ATTEMPT TO LOAD MORE THAN ONE PROGRAM ON ABS LOAD	A LOAD, LIBLOAD, or SLOAD request followed the loading of an absolute program; but once an absolute program is loaded, nothing else can be loaded. This can also be caused by a superfluous name call, such as a LIBRARY statement, within the load sequence.	Either remove illegal requests or move to a separate load sequence.	C
104 INSUFFICIENT ECS FL FOR LOAD	Not enough ECS is available to contain all the ECS blocks declared by the loaded programs.	Rerun with more ECS requested.	CU

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
106 TRANSFER POINT NOT FOUND (name)	Entry point to overlay or segment was not found.	Ensure correct name is used as transfer address and that it is declared to be an entry point.	C
107 INSUFFICIENT FL FOR EXECUTION	Under NOS/BE, an RFL statement specified a value too small to execute the program.	Adjust the RFL statement, or precede the load sequence with a REDUCE. statement to let the loader automatically set the correct value.	C
	Under NOS, a REDUCE(-) statement was present but a sufficient RFL statement was not provided.	Add an RFL statement to set the execution field length or a REDUCE. statement to reenable automatic management.	
200 ATTEMPT TO LOAD SUPPRESSED BINARY	An attempt was made to load a program that has compilation or assembly errors.	Correct and recompile the programs in error.	CU
202 OVERLAY DIRECTIVE NOT FIRST	The loader encountered an OVERLAY directive, but not until after loading one or more programs. If overlays are used, the first thing loaded must be an OVERLAY directive.	Place the OVERLAY directive before the first program, or remove all OVERLAY directives if overlays are not wanted.	C
203 NO SUCH PROGRAM CALL NAME - name	A name call statement called for a file or a program that was not found.	Check for possible misspelling. If a file call, create or attach the file. If a program call, add a LIBRARY or LDSET(LIB=) statement to declare the library after creating or attaching the file.	C
204 NOT CONTROL-CARD-CALLABLE-name	(NOS/BE only) library directory information indicates that the program is not usable when called by control statement; it can, for example, be a subroutine.	Check for correct usage of the program. If it really can be used from a control statement load, correct the library using the SETAL directive of EDITLIB.	C
205 USER NOT AUTHORIZED FOR PROGRAM - name	(NOS/BE INTERCOM only) the user cannot access this program.	Arrange with the installation for proper permission.	C
206 USEP INVALID FOR ABS LOAD	LDSET(USEP=) cannot be used to load absolute programs.	Use LIBLOAD instead. LIBLOAD requires an entry point name instead of a program name, but these are often the same for absolute programs.	C
210 BAD REQUEST NO. IN USER CALL	In an LDREQ table, the number indicating the request type (LOAD, SATISFY, etc.) is illegal.	Correct the program.	U
211 CANNOT PROCESS ENTRY REQUEST - PARAM AREA OVERWRITTEN	The ENTRY request in user call loads asked the loader to return information in the request table; but on this load, the request table is within the area used for loading so this cannot be done.	Move the request table out of the loadable area.	U

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
220 EMPTY LOAD FILE - filename	An empty record or end-of-file was the first thing read while processing a LOAD or SLOAD request.	Check for correct file name, and that the file was attached or created before calling the loader. If NR was specified on the LOAD or SLOAD request, check that the file was properly positioned.	CU
221 LOAD FILE NOT SPECIFIED	No file names were given on a LOAD or SLOAD request.	Add the needed file names to the request, or remove the request if not needed.	CU
222 FILE CONNECTED - name	The loader was directed to read binary from, or generate overlays or segments to, a connected file.	Correct the request, or return the file.	C
240 OVERLAY OR OVCAP CARD NOT SEPARATE SECTION	Each OVERLAY or OVCAP directive must be the only information in its record.	Correct the file structure.	C
241 SYNTAX ERROR ON OVERLAY OR OVCAP CARD	The OVERLAY or OVCAP directive does not conform to the rules given in sections 4 and 6.	Correct the directive.	C
243 ILLEGAL LEVEL NUMBER	On an overlay directive, one of the level numbers is greater than 77_8.	Correct the level numbers.	C
244 PRIMARY OVERLAY NOT PRECEDED BY (0,0) OVERLAY	The first overlay was not a main overlay (0,0).	Correct the overlay structure.	C
245 SECONDARY OVERLAY NOT PRECEDED BY ITS PRIMARY	A primary (n,0) overlay must precede a secondary (n,m) overlay with only other secondary overlays in between.	Correct the overlay structure.	C
246 INCONSISTENT FILE USAGE - name	An overlay generation load sequence attempted to write an overlay to a load input file.	Change the file name on the OVERLAY directive or NOGO statement to differ from the load input file name.	C
250 INSUFFICIENT FOL DIRECTORY SPACE	The value specified by the OV=n parameter on the OVERLAY directive is too small.	Correct the directive.	C
300 DIRECTIVE OR UNRECOGNIZABLE INPUT IN ABS LOAD	Input error.	Correct ABS load.	C
301 BAD LOADER INPUT OR DIRECTIVE SYNTAX ERROR	This message is followed by a dump of the first 10 words of illegal input. The message indicates an attempt to load a file that is not in the correct format, such as a source program instead of an object program, or that an object directive does not conform to the syntax rules.	Check for use of the correct file name, and check that the programs were compiled or assembled before loading, or correct the object directive.	CU

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
303 ABSOLUTE INPUT IN USER CALL	The user call loader was asked to load an absolute program, but only relocatable programs are allowed.	Change the program to use the LOADREQ macro for absolute loads.	U
304 ABS INPUT IN RELOCATABLE LOAD	The loader was asked to load an absolute program after loading some relocatable programs. The two types cannot be mixed. This can be caused by a superfluous name call, such as a LIBRARY statement within the load sequence, or by an attempt to use TRAP with an absolute program.	Remove the call of an absolute program, or move it to a separate load sequence; do not use TRAP except immediately before relocatable loads.	C
305 ABS INPUT NOT (0,0) LEVEL OVERLAY	The loader was asked to load an absolute overlay that is not a main overlay. Such overlays can be loaded only by the corresponding main overlay.	Correct the job to call the correct main overlay.	C
306 ABS INPUT LOAD ADR LT RA+100	The loader was asked to load an absolute program with a load address less than 100_8. This indicated an incorrect origin field on the IDENT statement of an absolute COMPASS program.	Correct the origin field to be 101_8+n, where n is the number of symbols given on ENTRY statements (if any).	C
307 OVERLAY CARD ENCOUNTERED DURING USER-CALL LOAD	Overlays cannot be generated by the user call loader.	Remove the overlay directive.	U
310 HARDWARE DEFICIENCY - program	In order to run correctly, the program requires a hardware feature not present.	If the program is written in COMPASS, it might need to be changed. For other languages, recompiling the program should solve the problem.	CU
340 BAD LINK BINARY TABLE	Internal error in loader.	Follow site-defined procedures for reporting software errors or operational problems.	CU
341 PROCEDURE DISALLOWED IN USER-CALL LOAD	The user call loader was asked to load a CYBER Control Language (CCL) procedure. This is prohibited.	Check for incorrect entry point names and file names.	U
342 PROCEDURE CALL MUST BE SINGLE CARD LOAD SEQUENCE	The loader was asked to load a procedure by a LOAD or other such loader statement instead of a name call, such as procname(params); or, the name call was preceded by LDSET or other loader statements.	Call the procedure with a name call, and remove LDSET or other such statements.	C
343 PROCEDURE DISALLOWED IN RELOCATABLE LOAD	On a name call to load a file, the loader read first a relocatable program and then a procedure. The two types cannot be mixed.	Restructure files to remove the procedure. Call the procedure separately.	C

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
370 CANNOT PROCESS FILES REQUEST - 1ST RECORD OF ZZZZZDF TOO BIG	An internal file built by CYBER Record Manager for communication with the loader is not in correct format. All LDSET(FILES=) requests are ignored.	Follow site-defined procedures for reporting software errors or operational problems.	CU
371 CANNOT PROCESS STAT REQUEST - ILL-FORMATTED ZZZZZDG FILE	An internal file built by CYBER Record Manager for communication with loader is not in correct format.	Follow site-defined procedures for reporting software errors or operational problems.	CU
400 UNBALANCED PARENTHESES	The parentheses on a segmentation directive are not correctly paired.	Add the missing parentheses, or delete extra ones.	C
401 MISSING PARAMETER.	An expected parameter was not found on a segmentation directive.	Add the missing parameter.	C
402 ILLEGAL SEPARATOR	A segmentation directive contained a separator that is illegal or out of context.	Correct the directive.	C
403 name - UNRECOGNIZABLE DIRECTIVE	The input file of segmentation directives contained a statement that is not a legal directive. This can be caused by presenting the wrong file or a mispositioned file.	Check for incorrect file name or position; misspelled verb; label field beginning beyond column 2; or omission of , or * in column 1 of a continuation or comment statement.	C
404 INCOMPLETE PARAMETER	A $ indicated a literal as a segmentation directive parameter, but no corresponding $ to terminate the literal was found.	Insert a second $ to terminate the literal.	C
405 name - USED ON LOWER LEVEL	A segment or program named on a TREE directive already is declared to be in a lower level of the structure.	Correct the tree structure.	C
406 name - CONFLICTS WITH EARLIER USAGE	A segmentation directive specifies a use of a program, segment, or tree that conflicts with other specifications.	Correct the tree structure.	C
407 MORE THAN ONE ROOT SEGMENT	Segmentation directives asked for multiple trees in the bottommost level.	Correct the structure.	C
410 NO ROOT SEGMENT	The bottommost level of a segment structure is empty.	Correct the structure.	C
411 MORE THAN 4095 SEGMENTS	The maximum number of segments was exceeded.	Combine the segments to reduce the total number.	C
412 name - UNDEFINED SEGMENT	A segment named on a GLOBAL or INCLUDE directive did not appear in a TREE statement.	Check for misspelling. Add the segment to the appropriate TREE directive.	C
420 ABS OR NEG RELOCATION NOT ALLOWED - PROG name	A program being loaded in segment generation used absolute or negative relocation.	Either correct the program or do not use in segmented loads.	C
422 SEGMENT TOO LARGE	Attempted to generate a segment greater than 377777_8 words.	Restructure seqment.	C

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
500 OVERLAY - CAPSULE DIRECTIVES INCOMPATIBLE	Both OVERLAY and CAPSULE directives cannot appear in the same loading operation.	Break into two load sequences: one for overlay generation and one for capsule generation.	C
501 CAPSULE DIRECTIVES NOT ALL AT BEGINNING OF FIRST LOAD FILE	A CAPSULE directive was encountered after loading some programs. All CAPSULE directives must precede all programs.	Move CAPSULE directives.	C
502 CAPSULE DIRECTIVES DISALLOWED IN USER-CALL LOAD	A CAPSULE directive was encountered during a user call load. Capsules cannot be generated in user call loads.	Remove the CAPSULE directive	U
503 ECS TEXT DISALLOWED IN CAPSULES OR OVCAPs	While generating a CAPSULE or OVCAP, the loader read a program that declares one or more ECS blocks. Only central memory blocks are allowed.	Change the program to use central memory blocks instead; or for OVCAPs, declare the ECS blocks in the main overlay.	C
504 NON-STANDARD RELOCATION AT ADDRESS address	The loader read a program containing nonstandard relocation while generating a CAPSULE or OVCAP. Standard relocation requires that the relocated field be exactly 18 bits wide with the rightmost bit being bit 30, 15, or 0. Note that external references are considered relocatable and must also follow these rules.	Correct the program.	C
505 ENCAPSULATION NOT TERMINATED BY NOGO	A capsule generation load sequence must be terminated by a NOGO statement, not by an EXECUTE or name call statement.	Correct the job.	C
506 ENCAPSULATION AND NO CAPSULES SPECIFIED	A GROUP statement specifying a capsule generation load was present, but no CAPSULE statements were present to specify the contents of the capsules.	Correct the job.	C
507 CAPSULE WITH NO ENTRY POINTS	A capsule was generated with no entry points. Such a capsule can never be referenced.	Add LDSET(EPT=) as necessary to declare entry points.	C
520 OVCAP DIRECTIVE ILLEGAL IF NOT IN OVERLAY GENERATION	An OVCAP directive was read but no overlays were generated; there is no main overlay with which the OVCAPs can be linked.	Generate overlays along with the OVCAPs; or change the OVCAPs to capsules if they do not depend on a particular overlay structure.	C
521 OVERLAY DIRECTIVE FOLLOWING OVCAP DIRECTIVE MUST SPECIFY (0,0) LEVEL	An OVERLAY directive followed OVCAP generation but did not specify a new main overlay. This meant it was part of the current overlay structure, violating the restriction that all overlays in a structure precede all the OVCAPs.	Correct the overlay structure.	C

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
525 OVCAP BINARY NOT STATICALLY LOADABLE	The loader was requested to load an OVCAP. OVCAPs can only be loaded in response to a call to FDL.LOC issued by a running program.	Correct the job.	CU
526 OVCAP DIRECTIVE ILLEGAL IN USER-CALL LOAD	An OVCAP directive was encountered in a user call load. OVCAPs cannot be generated in user call loads.	Remove OVCAP directives.	U
4100 UNSATISFIED EXTERNAL REF - name	The loader could not find an entry point to match an external reference.	Check for possible misspelling either on the reference or on the entry point definition. Verify that the name was defined as an entry point; or if it was on a library, verify that the library was in the library set.	CU
4101 COMMON BLOCK REDEFINITION (name)	Common block length exceeds previous definition. Original length is retained.	Correct common block specifications.	CU
4102 DUPLICATE ENTRY POINT NAME (name)	Entry point name which duplicates previously encountered name is ignored.	Check that intended entry point name was selected.	CU
4103 DUPLICATE PROGRAM NAME FROM FILE PROGRAM SKIPPED ---name	During a file load, a program was read that has the same name as a program that is already loaded. The duplicate is skipped.	Change program name.	CU
4104 DUPLICATE PROGRAM NAME PROGRAM LOADED ---name	The program being loaded has the same name as a program already loaded. Duplicate program is skipped under NOS. Both programs are loaded under NOS/BE.	Change program name.	CU
4105 CM BLANK COMMON TRUNCATED BY nnnnnnB WORDS.	The execution field length is large enough for all programs and labeled common blocks but is not large enough when blank common is included. An attempt to access the last nnnnnn words of blank common will cause a job abort. In the case of a segmented load, SEGRES will not allow the program to go into execution, but will abort during the load process with the fatal error message CM FL TOO SMALL.	Enlarge the execution field length. Refer to section 3 for a discussion of the factors that affect execution field length.	CU
4106 SPECIFIED LARGER BLANK COMMON THAN DECLARED AT LOWER LEVEL	During overlay generation, a program declared more blank common than was declared in a lower level overlay.	Increase the definition in the lower level overlay.	C
4107 ABSOLUTE LOAD NOT FOLLOWED BY EXECUTE	A NOGO statement terminated a load sequence that loaded an absolute program.	Change NOGO to EXECUTE, if desired, or remove the load sequence if not needed.	CU

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
4110 INTERACTIVE DEBUG IGNORED ON THIS LOAD	CYBER Interactive Debug is ignored in loads involving SEGLOAD control statements or GROUP/CAPSULE object directives.	Request load without CYBER Interactive Debug.	C
4111 TRAP OVERRIDES INTERACTIVE DEBUG	When both TRAP and Interactive Debug are used during a load, Interactive Debug is ignored.	Reload files; use only one of the debugging routines.	C
4200 LOADER CARD ERROR FOLLOWING CARD IGNORED	A loader statement did not conform to the syntax rules for the particular statement.	Correct the syntax.	CU
4201 PROGRAM NOT FOUND - name	A program requested by an LDSET(USEP=), TREE, or INCLUDE request could not be found.	Check for possible misspelling either on the request or in the program. Check that all necessary libraries are in the library set. For segmentation, check that all the necessary files were loaded.	CU
4204 ILLEGAL ORIGIN SPECIFICATION	On an OVERLAY directive, an origin specification gave an address less than 110_8; the overlay level was (0,0), for which an origin specification is illegal.	Correct the overlay directive.	C
4205 NO BLANK COMMON AT LOWER LEVEL - Cnnnnnn IGNORED	An origin specification on an OVERLAY directive could not be honored because no blank common was declared on any lower overlay.	Remove the Cnnnnnn parameter from the OVERLAY directive.	C
4206 ENTRY NAME ON OVERLAY CARD NOT FOUND - name	An origin specification on an OVERLAY directive could not be honored because it referenced an entry point that was not defined in any lower overlay.	Check for possible misspelling either on the reference or the entry point definition. Verify that the program containing the entry point was loaded in a lower overlay. Verify that the name was defined as an entry point.	C
4207 OBJECT DIRECTIVES NOT ALLOWED	A directive other than OVERLAY, OVCAP, GROUP, or CAPSULE was encountered when reading a load file.	Remove the directive from the load file. If needed, put it in the control statement stream.	CU
4210 FOL GENERATION, WRITING BINARY TO SAME FILE AS MAIN OVERLAY	FOL generation requires all overlays to be written to the same file.	Correct the OVERLAY directive.	C
4211 ILLEGAL OV SPECIFICATION	The OV=n parameter on an OVERLAY directive must be a decimal number; it is valid only on the (0,0) overlay.	Correct the directive.	C
4220 ILLEGAL LOADER REQUEST	A loader request is illegal on this particular type of load; for example, PASSLOC is legal in a user call load but not in overlay generation.	Correct the job.	CU

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
4221 LOAD FILE NAME FORMAT ERROR - name	File names used by the loader must be one to seven alpha-numeric characters, with the first character alphabetic.	Correct the file name.	U
4222 NO PROGRAMS SPECIFIED ON SLOAD	An SLOAD request did not list any programs to be loaded.	Correct the request.	U
4224 SLOAD PROGRAM NOT FOUND - name	A program named on an SLOAD statement could not be found on the specified file.	Check for a misspelled file or program name. Verify that the file was properly created or attached.	CU
4225 FORMAT ERROR ON LIBLOAD REQUEST	The library name is missing from a LIBLOAD request.	Correct the request.	CU
4227 ENTRY ON LIBLOAD NOT FOUND - name	An entry point named on a LIBLOAD request could not be found.	Check for possible misspell-ing either on the reference or on the entry point definition. Verify that the name was de-fined as an entry point.	CU
4230 FORMAT ERROR ON CMLOAD OR ECLOAD REQUEST	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4231 FORMAT ERROR - SATISFY REQUEST - name	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4232 FORMAT ERROR ON LIB REQUEST - name	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4233 FORMAT ERROR ON MAP REQUEST	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4234 FORMAT ERROR ON PRESET REQUEST	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4235 FORMAT ERROR ON USEP REQUEST - name	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4236 FORMAT ERROR ON USE REQUEST - name	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4237 SUBST FORMAT ERROR	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4240 FORMAT ERROR ON OMIT REQUEST - name	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
4241 FORMAT ERROR ON PASSLOC REQUEST	Indicates that a request of the indicated type does not conform to the format defined for that request.	Correct the program.	CU
4242 FORMAT ERR - COMMON REQUEST - name	Indicates that a COMMON request in an LDSET request specified an invalid block name.	Correct the program.	CU
4271 TRANSFER NAME NOT FOUND - name	The name on a COMPASS END statement or in an LDSET EPT table does not exist or was not named as an entry point.	Make sure the correct name is used as the transfer address and that it is declared to be an entry point.	C
4272 TOO MANY PARAMS IN EXECUTE REQUEST	The number of parameters in an execute request exceeded the maximum of 42.	Correct the program.	U
4273 NONEXISTENT LIBRARY GIVEN - name	No file of the given name was present, and no system library of that name exists.	Verify the correct file name (user library) or library name (system library). For a user library, check that the library was created or attached before calling the loader.	CU
4274 LIBRARY NOT ON MASS STORAGE - name	The specified library file is on magnetic tape or other such nonrandom device, and is unusable.	On NOS, use COPYBF to copy the library onto mass storage. On NOS/BE, use the SEQTORAN directive of EDITLIB.	CU
4275 ILL-FORMATTED LIBRARY - name	The specified library is not in the format of a library.	Verify that the correct library file name was given and that it was created as a library by LIBGEN (NOS) or EDITLIB (NOS/BE). For NOS/BE, check to see if the library was copied at any time by any program other than EDITLIB; such copies are no good. Rebuild the library if necessary.	C
4310 POTENTIAL HARDWARE DEFICIENCY - program	If the program is later executed on the same machine, it will fail because it requires a hardware feature that is not present.	If the machine on which the program is to be executed possesses the required features, no action is needed. Otherwise, the program should be recompiled on its target machine. A COMPASS program might need to be changed to avoid the missing feature.	C
4340 TRIED TO LOAD INTO BLOCK BELOW ORIGIN - name	During overlay generation, an attempt was made to load data into a common block declared by a lower overlay; the attempt is ignored. During a user call load, data was loaded into a common block first declared in an earlier load.	Remove presetting from the program being loaded; preset the common blocks where first declared.	CU

TABLE B-2. FATAL AND NONFATAL ERRORS DETECTED BY THE LOADER (Contd)

Message	Significance	Action	Issued By
4341 TRIED TO LOAD INTO ABSOLUTE BLOCK	In an OVERLAY, OVCAP, or CAPSULE generation load, an attempt was made to load data into the absolute block.	Correct the program.	C
4400 PARAMETER NAME TRUNCATED TO 7 CHARACTERS	A name on a segmentation directive is too long. Only the first seven characters are used.	Correct the name.	C
4401 END CARD MISSING	An end-of-record terminated segmentation directives because no END statement was read.	Insert an END statement.	C
4402 name - NOT DECLARED GLOBAL	During segmentation, the loader expected the block to be made global but it did not appear on any GLOBAL directive.	Add a GLOBAL statement.	C
4420 COULD NOT FIND COMMON OR GLOBAL BLOCK - name	During segmentation, a labeled common block specified on a COMMON or GLOBAL directive was not found.	Correct block name spelling. Verify that programs which reference block were loaded.	C
4421 SATISFY IGNORED ON SEGMENT LOAD	Unlike other types of loads, segmentation is a two-pass process with all externals being satisfied at the end of the first pass; thus, a SATISFY statement in a load sequence serves no purpose.	Remove the SATISFY statement. Add LDSET(LIB=) if necessary to specify additional libraries to be searched.	C
4422 TRIED TO LOAD INTO BLOCK OUTSIDE SEGMENT - name	In a segment generation load, attempt was made to load data into a block which is in another segment or the absolute block.	Correct the program.	C
4450 CONFLICTING SEGMENTS CALLED BY SAME WORD ADDRESS	A single word in a segment references entry points in two different segments that cannot coexist.	Ensure correct segment names are used and that tree is structured correctly.	C
4500 FORMAT ERROR ON EPT REQUEST - name	A name on an EPT request is not a legal entry point name.	Correct the request.	C
4501 FORMAT ERROR ON NOEPT REQUEST - name	A name on a NOEPT request is not a legal entry point name.	Correct the request.	C
4502 NOT ALL CAPSULE DIRECTIVES PROCESSED	After loading all specified files, one or more programs named on CAPSULE directives were not found.	Verify that all required files were loaded. Check if CAPSULE directives are correct.	C
4503 EPT REQUEST IGNORED - name	In a non-encapsulation load sequence, a LDSET EPT request in a LDSET table was encountered after physical loading began. It does not affect the entry points in the 5400 table header being generated.	Move the LDSET table to the first relocatable program on the load file.	C

TABLE B-3. ERRORS DETECTED BY THE USER CALL INTERFACE

Message	Significance	Action	Issued By
CMM PARAM MISSING FROM LOADER USER CALL	A LOADER macro call made while CMM was active did not specify CMM on the macro call.	Add the CMM parameter to the LOADER macro call.	UCLOAD
FWA-LWA ERROR ON LOADER USER CALL	The fwa or lwa parameters in the LDREQ BEGIN block are in error. fwa lwa, lwa fl, or (for PILOAD only) fwa is in the user area and lwa is in the CMM area.	Correct the program.	PILOAD UCLOAD
LDREQ BEGIN MISSING	The address passed in a LOADER macro was not that of an LDREQ BEGIN block.	Correct the program.	PILOAD
LDREQ END MISSING	The list of LDREQ blocks is not terminated by an LDREQ END block.	Correct the program.	PILOAD UCLOAD
LDV ERROR LOADING LOADU	PILOAD could not load the user call loader.	Contact a system analyst.	PILOAD
OLD-STYLE SEGMENTATION NOT SUPPORTED	The program issued a SCOPE 3.3 format user call specifying SCOPE 3.3 type segmentation.	Convert the program to use CYBER loader facilities.	UCLOAD
SL LIST TOO BIG - CONSULT UCLOAD IMS	The program issued a SCOPE 3.3 type user call, naming too many programs to allow conversion by UCLOAD.	Correct the program to use current formats.	UCLOAD

TABLE B-4. ERRORS DETECTED BY SEGRES

Message	Significance	Action
BAD SEGLOAD BINARY	The file of segments is not in correct format. The job is aborted.	Regenerate segments.
CM FL TOO SMALL	SEGRES was unable to obtain enough central memory to run the program. The job is aborted.	Rerun the job with larger limits.
ECS FL TOO SMALL	SEGRES was not given enough ECS to run the program. The job is aborted.	Request sufficient ECS, and rerun the job.
ECS WRITE ABORT	An ECS parity error occurred while initializing ECS common blocks. The job is aborted.	Follow site-defined procedures for reporting software errors or operational problems. Rerun the job.
IGNORING EXTRA LOAD FILE	The load sequence requested loading of multiple files. A single file must contain all the segments.	Correct the load sequence.
MISSING EXECUTE	No EXECUTE statement is present in the load sequence. The job is aborted.	Add an EXECUTE statement.
MISSING LOAD	No LOAD statement is present in the load sequence. The job is aborted.	Add a LOAD statement.

TABLE B-4. ERRORS DETECTED BY SEGRES (Contd)

Message	Significance	Action
NONEXECUTABLE WORD LOADING A SEGMENT	A word in which the top six bits are zero contains a reference to another segment that would cause loading. Only executable words can have such references. The job is aborted.	Correct the program.
NO TRANSFER ADDRESS	The root segment has no transfer address.	Correct the program.
SEGRES TOO LARGE	The segment loader resident exceeds its maximum size of 1000_8 words. Also issued when a second SEGLOAD is performed without rewinding/returning the ABS file.	Check if a second SEGLOAD was encountered, or follow site-defined procedures for reporting software errors or operational problems.

TABLE B-5. ERRORS DETECTED DURING DYNAMIC CAPSULE AND OVCAP OPERATIONS

X6 Code	Significance	Action	Issued By
0	No error occurred.	None.	FDL.LDC FDL.ULC XFDL.REL XFDL.UNR FDL.LOC FDL.UOC
1	Bad library list; either the entries or the address is incorrect.	Correct the list and rerun the job.	FDL.LDC FDL.LOC
2	Unknown group name; either the name is incorrectly stored, or the named group cannot be found.	Correct the name and rerun the job.	FDL.LDC FDL.ULC FDL.LOC
3	Unknown capsule or OVCAP name; either the name is incorrectly stored, or the named entity cannot be found.	Correct the name and rerun the job.	FDL.LDC FDL.ULC FDL.LOC FDL.UOC
4	Bad capsule or OVCAP format.	Reformat and rerun the job.	FDL.LDC FDL.ULC XFDL.REL XFDL.UNR FDL.LOC FDL.UOC
5	Bad PASSLOC or ENTRY table format; either an incorrect address was used, or the addresses in the list reference an incorrect table.	Correct the condition and rerun the job.	FDL.LDC FDL.ULC XFDL.REL XFDL.UNR FDL.LOC FDL.UOC
6	Capsule or OVCAP is already in memory.	Remove the call and rerun the job.	FDL.LDC FDL.LOC
7	Either a capsule is being treated as an OVCAP, or an OVCAP is being treated as a capsule.	Change the call and rerun the job.	FDL.LDC FDL.LOC

TABLE B-6. ERRORS DETECTED DURING OVERLAY LOADING (LDV UNDER NOS/BE ONLY)

Message	Significance	Action
0001 NONEXISTENT LFN - xxxxxxx	The file specified for a nonlibrary load does not exist.	Verify that the file is correct and was properly created or attached before execution.
0003 OVERLAY NOT FOUND - xxxxxxx OVERLAY NOT FOUND - (nn,nn)	The specified overlay could not be found. The format of the message differs depending on whether the load was requested by overlay name or by overlay level.	Verify that the overlay name or level is correct on the call and that the overlay is properly installed in its library, if any, or is present on the overlay file.
0004 INSUFFICIENT FL INSUFFICIENT FL - nnnnnn NEEDED	The loadable area is too small to contain the overlay. If a value is given in the message (possible only on loads from a library), this value is the minimum lwa needed to allow loading. If lwa is not specified in the call, FL-3 is used as the lwa.	If lwa is not specified, increase the execution field length. If lwa is specified, increase it to an adequate amount. For overlays built by the loader, the hha field of the EASCM table gives the maximum lwa needed for any overlay and is present in RA+104_8 during execution.
0005 OVERLAY HEADER NOT CORRECT	A program loaded from a library was not an absolute overlay, or a record other than an absolute overlay was found during a file search, or an overlay with ECS text was found and is not loadable by LDV.	Check to see if the program attempted a file load from a file in library format; if so, change the call to request a library load. If loading from a library, rebuild the library. If loading from a file, make sure only overlays are present on the file. If ECS text must be used, put it into the (0,0) overlay.
0006 REQUEST FORMAT ERROR	Some or all of the parameter words were outside the field length, or the contents of the parameter words were illegal or inconsistent.	Correct the program.
0007 E-BIT WITH AUTO-RECALL	LDV was called with automatic recall and the parameters specified automatic execution of the loaded overlay.	Remove the recall parameter from the LOADREQ call.
0011 I/O ERROR STATUS	LDV received an error status when trying to read.	Check the accompanying message; if not clear, refer to the NOS/BE Diagnostic Handbook.
0012 LOADABLE AREA LS 100B	The requested ECS-resident overlay could not be loaded because no fwa was specified in the call and the address in bits 17 through 0 of RA+65_8 was less than 100_8.	Specify fwa if at all possible; otherwise, request system analyst to remove the routine from ECS residency. (ECS residency is supported only for system libraries.)
0013 INCONSISTENT CMM USAGE	Either a non-CMM type LDV call has been made when CMM is active or a CMM type LDV call has been made when CMM is not active.	Correct the program.

TABLE B-6. ERRORS DETECTED DURING OVERLAY LOADING (LDV UNDER NOS/BE ONLY) (Contd)

Message	Significance	Action
4001 (no dayfile message)	The fwa in the overlay header and the fwa in the call were both nonzero but were different. The overlay is loaded at the address given in the call.	Modify the program to supply a correct fwa.
4002 1ST EPT USED - COULDNT FIND name	An entry point name was specified in the call and the loaded overlay contains one or more named entry points, none of which matches the one requested. The first entry point name encountered is used.	Verify that the entry point name is correct. If the overlay was generated as a COMPASS ABS program, verify that the name appeared on an ENTRY statement. If the overlay was generated by the loader, verify that no OVERLAY statements were used in its generation, only NOGO(lfn,...) where the entry point is named on the NOGO statement.
4003 NONEXISTENT LIBRARY - name	The specified library does not exist.	Verify that the library name is correct and that it was correctly created or attached.
4004 FILE NOT A RANDOM RMS USER LIB - name	The file specified is not a random mass storage file or is not of library format. The file is ignored.	If a file load instead of a library load is needed, change the code to request a file load. If a library load is indeed wanted, create a random library by using EDITLIB.
4006 UNSUPPORTED ECS USER FILE - name	The file specified resides in ECS and is not loadable by LDV. The file is ignored.	Copy the file to one that is not ECS-buffered, and use the copied file.
4007 UNSUPPORTED TAPE FILE - name	The file specified resides on a magnetic tape and is not loadable by LDV. The file is ignored.	Copy the file to mass storage and use the copied file.

TABLE B-7. ERRORS DETECTED DURING TRAP DIRECTIVE PROCESSING

Message	Significance	Action
AT PARAM REQUIRED ON FRAME DIRECTIVE	No AT parameter is present on a FRAME directive to indicate the point when a dump is to be taken.	Add an AT clause to the directive.
ILLEGAL DELIMITER ON ABOVE TRAP DIRECTIVE	An illegal character separated two parameters on a TRAP directive.	Correct the directive.
ILLEGAL NUMBER ON ABOVE TRAP DIRECTIVE	A numeric parameter contains a nonnumeric character.	Correct the directive.
ILLEGAL PARAM ON ABOVE TRAP DIRECTIVE	A parameter that should be a keyword is not, or it is only valid for the other (TRACK or FRAME) type of directive.	Correct the directive.

TABLE B-7. ERRORS DETECTED DURING TRAP DIRECTIVE PROCESSING (Contd)

Message	Significance	Action
ILLEGAL VERB IN ABOVE TRAP DIRECTIVE	The verb is neither TRACK nor FRAME.	Check for a label not beginning in column 1. Correct the directive.
PARAM(S) MISSING ON ABOVE TRAP DIRECTIVE	The directive has no verb, or one of the parameters is incomplete (such as START not followed by a number).	Correct the directive. Remove any blank cards.
TOO MANY WHENS ON TRACK DIRECTIVE	More than 15 WHEN clauses appear.	Either reduce the number of WHEN clauses or break the tracking range up and use two separate TRACK directives.
UNTIL CANNOT BE LESS THAN START	The START value must be less than or equal to the UNTIL value to allow at least one iteration to be trapped.	Correct the directive.

TABLE B-8. ERRORS DETECTED BY TRAPPER DURING EXECUTION

Message	Significance	Action
COULD NOT FIND BLOCK name	The program or block indicated could not be found.	Change the directive to specify the correct name.
ERROR ON TRAP FILE ZZZZZ28	The data file written by TRAP for TRAPPER is incorrect.	Follow site-defined procedures for reporting software errors or operational problems.
FOR PARAM MUST BE USED IF FROM IS IN ECS	The FROM parameter of a FRAME directive is in ECS but no FOR clause is present.	Add a FOR clause.
FRAME AT OR TRACK FROM OR TO IS IN ECS	The address in the AT clause of a FRAME directive, or in the FROM or TO clause of a TRACK directive, is in ECS. This is illegal because instructions can only be executed in central memory.	Correct the directive.
INSUFFICIENT BUFFER SIZE	The buffer in TRAPPER is too small to read the ZZZZZ28 file created by TRAP.	Follow site-defined procedures for reporting software errors or operational problems.
MORE THAN 100 WHEN CONDITIONS USED	Over 100 WHEN clauses appeared in the directives.	Reduce the number of WHEN clauses.
REFERENCE INSIDE TRAPPER	User program tried to reference location inside TRAPPER program.	Remove the reference.
TO ADDRESS IS LESS THAN FROM ADDRESS	On a TRACK directive, the TO clause specified an address less than that of the FROM clause.	Correct the directive. It might be necessary to use two TRACK directives to achieve the desired effect.
TRAP ADDRESS IS OUTSIDE FIELD LENGTH	An address specified on a directive is not within the job's allocated memory.	Correct the directive.

GLOSSARY

Absolute Load -
A load of an overlay, segment, or COMPASS ABS program. No relocation or satisfaction of externals is needed because this was done when the absolute program was generated.

Basic Load -
A load operation in which all of the object code is loaded into memory at the same time.

Blank Common Block -
A common block into which no data is stored at load time. The first declaration of a blank common block need not be the largest declaration for the common block. Contrast with labeled common block.

Capsule -
A relocatable collection of one or more programs bound together in a special format that allows the programs to be loaded and unloaded dynamically from an executing program by the Fast Dynamic Loading facility.

Common Block -
An area of memory that can be declared by more than one relocatable program and used for storage of shared data.

Control Statement Load -
A load operation that is initiated by control statements encountered within a job stream. Contrast with user call load.

Dymamically Loaded Code -
Code loaded by an explicit request, as needed. The load is initiated by an executing program. Contrast with statically loaded code.

Entry Point -
A location within a program that can be referenced from other programs. Each entry point has a unique name with which it is associated.

Execute Only File -
On NOS, a permanent file created with the M=E option on one of the permanent file statements. The user is granted permission only to execute the file; read or write permission is not granted. In order to protect the contents of such a file when loaded, the loader never writes a load map (even if a map was selected); the loader also restricts the satisfying of externals for execute-only files to system libraries.

External Reference -
A reference in one object program to an entry point in another program.

Fast Dynamic Loading -
A facility that provides fast loading and unloading of specially formatted code, called capsules. The amount of memory required for job execution can be greatly reduced because capsules can be easily loaded and unloaded as needed, freeing memory for other uses.

Fast Overlay Loading -
A facility that provides for generation of an overlay structure with an overlay directory imbedded into the main overlay. It uses the directory for the loading of a higher level overlay.

Field Length -
The number of central memory words assigned to a job.

Global Library Set -
An ordered set of libraries specified on a LIBRARY statement. The libraries remain in effect throughout job execution unless specifically changed by a subsequent LIBRARY statement.

Global Common Block -
In a segmented load, a labeled common block that can be referenced by programs in different segments.

Labeled Common Block -
A common block into which data can be stored at load time. The first program declaring a labeled common block determines the amount of memory allocated. Contrast with blank common block.

Level -
In a segmented load, a division of memory into multiple regions so that loading and unloading of segments can occur independently in the various regions.

Library -
A file created by either LIBGEN (NOS) or EDITLIB (NOS/BE) that contains programs and the tables needed to locate and load the programs.

Library Set -
An ordered set of libraries that the loader searches to satisfy external references. It is composed of the global library set, followed by the local library set, followed by the default system library SYSLIB. Libraries are inserted in the order in which they are specified. Duplicate library names are suppressed.

Linking -
The process of matching external references to entry points of the same names and inserting the addresses of the entry points into the external references.

Loader Statement -
A control statement that begins with one of the keywords: CAPSULE, EXECUTE, GROUP, LDSET, LIBLOAD, LOAD, NOGO, SATISFY, SEGLOAD, or SLOAD. A control statement that begins with a local file name that duplicates any of the above keywords is treated as a name call statement.

Load Map -
A printout showing how memory was allocated by the loader during a load operation.

Load Sequence -
One or more consecutive control statements processed by the loader as a unit. A load sequence can be a single name call statement, or it can consist of loader statements, such as LOAD and LDSET, that are terminated by NOGO, EXECUTE, or a name call statement.

Local Library Set -
An ordered set of libraries defined by LDSET(LIB=) control statements and object directives and used for a single load sequence.

Main Overlay -
An overlay that must remain in memory throughout execution of an overlayed program.

Name Call Statement -
A control statement that begins with either the name of a file (such as LGO) or an entry point in a library (such as FTN). The name call statement causes the file or program to be loaded and executed.

Nucleus -
The NOS/BE system library that contains entry points corresponding to most control statement keywords. It is searched by the loader if the keyword on a name call statement does not name a local file or match any entry point in either the global or local library sets.

Object Directives -
Loader requests encountered within the load input stream in the form of loader tables.

Ovcap -
A special capsule designed for use with overlays. An OVCAP is analagous to a primary overlay in that it must be called into memory by a main overlay, and it can reference entry points and common blocks in the main overlay.

Overlay -
One or more relocatable programs that were relocated and linked together into a single absolute program. It can be a main, primary, or secondary overlay.

Primary Overlay -
A second level overlay that is subordinate to the main overlay. A primary overlay can call its associated secondary overlays and can reference entry points and common blocks in the main overlay.

Program-Initiated Load -
Another term for user call load.

Reduce Mode -
A job execution mode in which the loader automatically sets the field length for executing a program. When a job is not in REDUCE mode, the user must specify the field length.

Relocatable Load -
A load operation in which object programs are placed into memory locations that are not predetermined. Addresses are established and external references satisfied during the load operation.

Relocation -
Placement of object code into central memory in locations that are not predetermined and adjusting the addresses accordingly.

Root Segment -
The main segment of a segmented program that must remain in memory throughout execution of a segmented program.

Satisfying External References -
The process of searching one or more libraries and loading programs that contain entry points matching external references that are currently unsatisfied.

Secondary Overlay -
The third level of overlays. A secondary overlay is called into memory by its associated primary overlay. A secondary overlay can reference entry points and common blocks in both its associated primary overlay and the main overlay.

Segment -
An absolute subdivision of a segmented program that is automatically called into memory as needed (except for the root segment). Different segments can occupy the same memory locations at different times during job execution.

Segmentation -
Dividing a program into sections called segments that can occupy the same storage locations at different times. The root segment must be in memory throughout program execution; all other segments are loaded dynamically during program execution.

Statically Loaded Code -
Code that is loaded into memory once and remains resident throughout job execution. Contrast with dynamically loaded code.

SYSLIB -
The system library containing general purpose subroutines. It is searched by the loader if unsatisfied externals remain after searching the libraries in the global and local library sets. It is not used during capsule and OVCAP generation.

System Library -
A library that is installed as part of the operating system. System library names are maintained in a system table and can be used in the library set if the library name is declared in either a LIBRARY or LDSET(LIB=) statement. (A system library does not have to be attached because it is not a permanent file in the usual sense.) See Library.

Transfer Address -
The address of the entry point to which the loader jumps to begin program execution.

Tree -
A verb allowed in SEGLOAD directives that organizes segments into tree structures.

Tree Structure -
A program structure that, when viewed in time-memory coordinates, resembles a tree.

Unsatisfied External Reference -
An external reference for which no matching entry point was found. The unsatisfied external reference is filled with an address that causes the program to abort if the given instruction is executed.

User Call Load -
A relocatable load that is initiated by user requests from an executing program. The requests are formatted into request tables before the call is issued to the loader. Contrast with control statement load.

User Library -
A library that exists as a local file attached to the job. User libraries can be used in the library set if the library is either created or attached by a job and declared in either a LIBRARY or LDSET(LIB=) statement.

Weak External -
An external reference that is ignored during library searching and cannot cause any other program to be loaded. A weak external is linked, however, if the corresponding entry point is loaded for any other reason.

BINARY TABLES

The binary tables that the loader processes to place code in central memory are discussed in the following subsections. The binary tables consist of both object program tables and request tables.

OBJECT PROGRAM TABLES

Every object program, whether relocatable or absolute, is represented in a file or library as a sequence of tables. The formats of these tables are described in this section. They have no relationship to physical card formats when object programs are punched into binary cards. (Peripheral processor programs and system text overlays are not processed by the loader but are included here because no other manual documents their formats.) The first 60-bit word of each table is a header word with the general format shown in figure D-1.

A relocatable program unit (subprogram) is represented as the following sequence of tables:

1. PRFX table
2. LDSET table (optional)
3. PIDL table
4. ENTR, TEXT, REPL, FILL, LINK, XTEXT, PTEXT XREPL, XFILL, XLINK, SYMBOL and LINE NUMBER tables, any number of each, in any order
5. XFER table (optional)

An absolute central processor program or overlay is represented as the following sequence of tables:

1. PRFX table
2. ASCM, EASCM, ACPM, or EACPM table

A peripheral processor program or overlay is represented as the following sequence of tables:

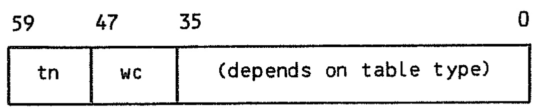
1. PRFX table
2. 6PPM table

A system text overlay is represented as the following sequence of tables:

1. PRFX table
2. ASCM table with level 1=level 2=1 and origin=entry=0

A capsule is represented as the following sequence of tables:

1. PRFX table
2. CAPSULE table



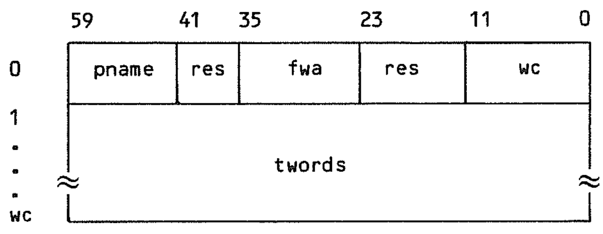
tn	A binary number designating the type of table.
wc	The number of 60-bit words in the table, not counting the header word.

Figure D-1. Header Word Format

A list of the binary tables is shown in table D-1. The formats of these tables are described as follows. The tables are listed in ascending order by table number. In these descriptions, all areas designated as res can be used only by CDC and are presently all zeros. Except where noted, all names are one to seven characters in display code, left-justified with binary zero fill, and can contain any character codes except 55_8 (blank) and 00.

6PPM TABLE

The 6PPM table (figure D-2) contains the memory image of either a peripheral processor program or overlay.



pname	Three letters and/or digits in display code.
res	Reserved for use by CDC.
fwa	The address of the byte in PPU memory into which the first byte of the table header word is to be loaded; the first text byte is loaded at PPU memory address fwa+5.
wc	The number of 60-bit text words; the number of 12-bit bytes is five times wc.
twords	The core image of a peripheral processor or overlay.

Figure D-2. 6PPM Table Format

TABLE D-1. BINARY TABLE TYPES

tn(octal)[†]	Mnemonic	Table Type
aabb[††]	6PPM	6000 peripheral processor (PP) program or overlay; bits 59 through 42 contain three alphanumeric characters in display code
3400	PIDL	Program identification and length
3500	PTEXT	Relocatable text
3600	ENTR	Entry point definitions
3700	XTEXT	Extended relocatable text
4000	TEXT	Relocatable text
4100	XFILL	Extended relocation fill
4200	FILL	Relocation fill
4300	REPL	Replication of text
4400	LINK	External reference linkage
4500	XLINK	Extended external reference linkage
4600	XFER	Transfer point
4700	XREPL	Extended replication of text
5000 [††]	ASCM	Absolute CPU program or overlay
5100 [††]	EASCM	Absolute CPU program with multiple entry points
5300 [††]	ACPM	Absolute program or overlay with ECS data
5400 [††]	EACPM	Absolute program or overlay with ECS data and multiple entry points
5600	SYMBOL	Symbol information
5700	LINE	Line number information and object code addresses
6000[††]	CAPSULE	Relocatable capsule or OVCAP (overlay-capsule)
6600		Reserved for use by installations
6700		Reserved for use by installations
7000	LDSET	Object directive
7700	PRFX	Prefix

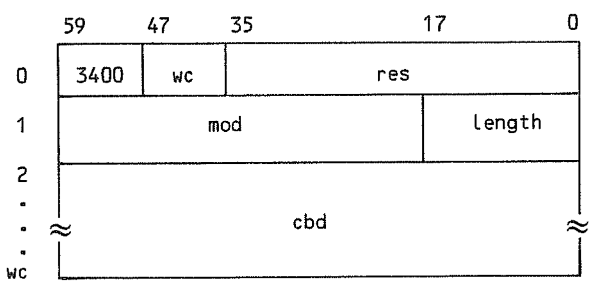
[†]A binary number designating the type of table.

[††]The header word varies from the general format by not containing a word count. This table must be the last or only table of its program or overlay and is terminated by an end-of-record.

PIDL TABLE (3400)

The PIDL table (figure D-3) contains the names and lengths of a relocatable subprogram's storage blocks.

The common block descriptors (figure D-4), if any, contain the name, type, and length of each common block known to the subprogram; there could also be a descriptor for the subprogram's local ECS block, if any. The order in which the descriptors occur is important, because this is the basis for the relocation designators in subsequent tables. A subprogram's PIDL table can contain up to 509 descriptors.



wc The number of 60-bit words in the table, not counting the header word.

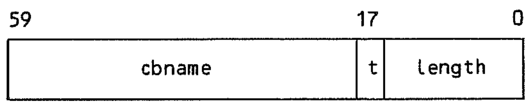
res Reserved for use by CDC.

mod The name of the program module, left-justified with zero fill. A subprogram name can be the same as a common block name and/or an entry point name without confusion.

length The number of 60-bit words in the program module's local central memory block; if this is zero, the block length is determined by the largest address into which subsequent tables load text.

cbd The common block descriptors which contain the name, type, and length of each common block known t: a subprogram. There could also be a descriptor for the program's ECS block.

Figure D-3. PIDL Table Format



cbname The name of the common block.

t The type of common block; t can be the following:

0 A central memory common block

1 An ECS common block

length If t is 0, the block size is length 60-bit words. If t is 1, the block size is 8 x length 60-bit words.

Figure D-4. PIDL Table Common Block Descriptor Format

The subprogram's local ECS block, if present, is represented by a descriptor in which the name is null (42 zero bits) and the type of block (t) is 1. A blank common block is represented by a descriptor in which the name is seven blanks. A subprogram can have two blank common blocks provided one is central memory and the other is ECS. Otherwise, no two common blocks can have the same name. However, a common block name can be the same as a subprogram name and/or an entry point name without confusion. The maximum size of a central memory block is $2^{17}-1$ (131071) 60-bit words; the maximum size of an ECS block is $2^{20}-8$ (1048568) 60-bit words.

PTEXT TABLE (3500)

The PTEXT table (figure D-5) contains text and an address at which it is to be loaded. The PTEXT table can also contain optional replication descriptors.

The text bits are loaded into consecutive memory locations beginning with word s+B, where s is the relative first word address and b is the binary relocation base. The text bits are loaded starting at the first bit in the first word (ffb) that receives textwords, and extends for the text length in bits (tlb). The binary relocation base (b) is determined by the relocation base designator (rb) as shown in table D-2; however, rb cannot be negative (negative program relocation) or 0 (absolute relocation), nor can it refer to a blank common block. Any bits not overwritten in the affected words are preserved.

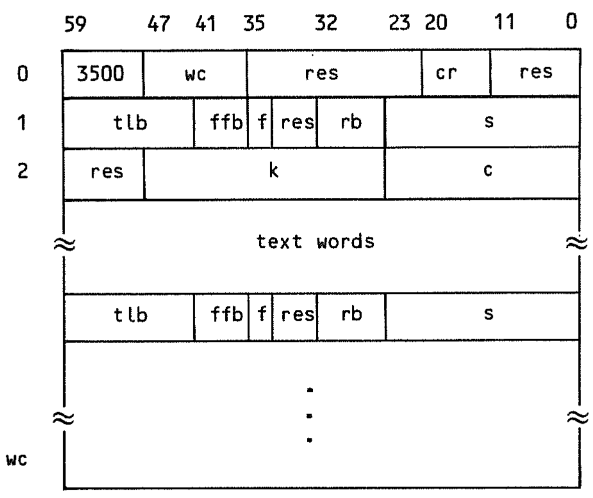
If a replicated PTEXT table is indicated by the flag (f), the increment (k) and the count (c) are processed as in the REPL table.

TABLE D-2. DETERMINATION OF THE BINARY RELOCATION BASE

rb	b
000	0
001	p †
002	-p †
003	c_1 ††
004	c_2 ††
.	.
.	.
.	.
777_8	c_{509}††

†Base address of the subprogram central memory local block.

††Base address of the ith common block listed in the subprogram PIDL table. $1 \leq i \leq$ number of common blocks.



wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
cr	Conditional relocation base designator. The PTEXT table is ignored if cr refers to a common block that was first declared by an earlier subprogram.
tlb	Text length in bits.
ffb	First bit in first word to receive first bit of text words.
f	Flag indicating whether PTEXT table is replicated. Values for f are as follows:

0	text words follow immediately
1	next word is replication indicator, followed immediately by text words

rb	Relocation base designator.
s	Relative first word address; cannot exceed 377777 octal.
k	Increment in bits between replicated copies of text words. If k=0, loader assumes increment is tlb.
c	Replication count. If c=0, loader assumes 1.

Figure D-5. PTEXT Table Format

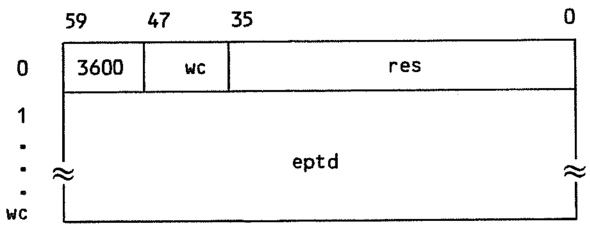
ENTR TABLE (3600)

The ENTR table (figure D-6) declares and defines entry point names in a relocatable subprogram, which can be used to satisfy external references in other subprograms.

Each entry point descriptor (figure D-7) is two words and defines one entry point name. The order in which they occur is immaterial. No two entry points in the same program can have the same name; however, an entry point name can be the same as a subprogram name and/or a common block name without confusion.

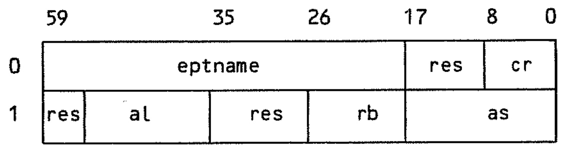
The defined value of the entry point name is a signed binary integer value A+b where:

- A is the relative value of the entry point (al if rb refers to an ECS block; as if rb does not refer to an ECS block) with sign extended.
- b is the binary relocation base designated by rb as shown in table D-2.



wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
eptd	Entry point descriptors. Each is two words long and defines one entry point name.

Figure D-6. ENTR Table Format

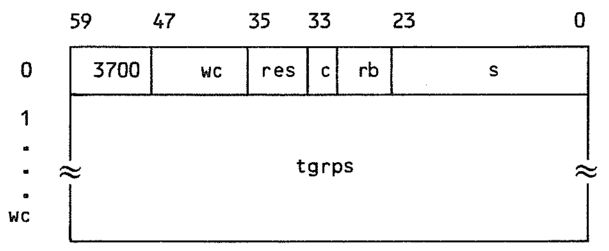


eptname	The entry point name in a relocatable program.
res	Reserved for use by CDC.
cr	A relocation base indicator, in the same format as r. The descriptor is ignored if cr refers to a common block that was first declared by an earlier subprogram.
al	Relative value of the entry point if r refers to an ECS block.
rb	Relocation base designator.
as	Relative value of the entry point if r does not refer to an ECS block.

Figure D-7. ENTR Table Entry Point Descriptor Format

XTEXT TABLE (3700)

The XTEXT table (figure D-8) contains text (instructions and data) and an address at which the text is to be loaded. The XTEXT table can also contain relocation indicators. The XTEXT table allows a relative starting address greater than 377777_8. The text words are loaded into consecutive memory locations beginning with s+b, where s is the relative first word address and b is determined by rb as in table D-2. The relative base designator (rb) cannot be 2 (negative program relocation) and cannot refer to a blank common block. Each text group except possibly the last is 16 words long. The text groups and their locations are the same as described for the TEXT table.



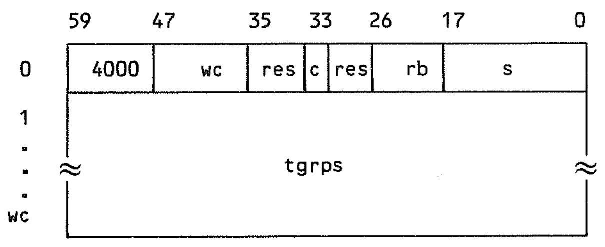
wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
c	The conditional flag; c can have the following values: 0 Load text unconditionally; 1 Ignore this XTEXT table if r refers to a common block that was first declared by an earlier subprogram
rb	Relocation base designator.
s	Relative first word address; cannot exceed 7777777_8.
tgrps	Groups of text words to be loaded. Each group, except possibly the last, is 16 words long.

Figure D-8. XTEXT Table Format

TEXT TABLE (4000)

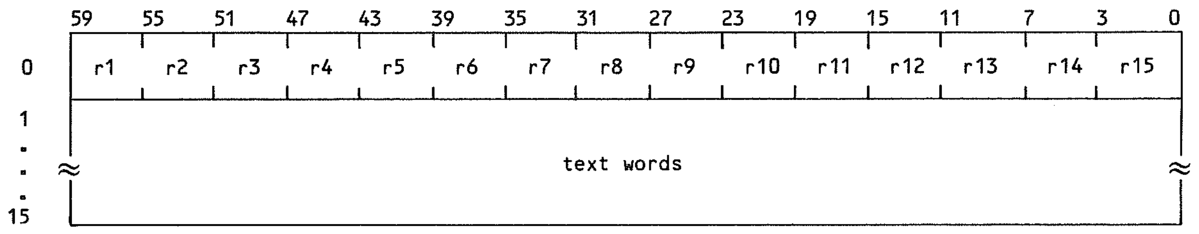
The TEXT table (figure D-9) contains text (instructions and data) and an address at which it is to be loaded. The TEXT table can also contain relocation indicators.

The text words (figure D-10) are loaded into consecutive memory locations beginning with s+b, where s is the relative first word address and b is determined by rb as in table D-2. The relative base designator (rb) cannot be 2 (negative program relocation) and cannot refer to a blank common block. Each text group, except possibly the last, is 16 words long.



wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
c	The conditional flag; c can have the following values: 0 Load text unconditionally; 1 Ignore this TEXT table if r refers to a common block that was first declared by an earlier subprogram
rb	Relocation base designator.
s	Relative first word address; cannot exceed 377777_8.
tgrps	Groups of text words to be loaded. Each group, except possibly the last, is 16 words long.

Figure D-9. TEXT Table Format



r_i Relocation indicator. A 4-bit byte that indicates the kind of relocation that is to be performed for each text word.

Figure D-10. TEXT Table Text Group Format

The first text word contains a 4-bit relocation indicator byte for each of the up to 15 text words in the group. The leftmost byte is applied to the first text word, the next byte to the second text word, and so on. The bits in the relocation indicator bytes are interpreted as follows:

000x	No relocation
10xx	Upper address, positive relocation
11xx	Upper address, negative relocation
010x	Middle address, positive relocation
011x	Middle address, negative relocation
1x10	Upper lower address, positive relocation
1x11	Upper lower address, negative relocation
0010	Lower address, positive relocation
0011	Lower address, negative relocation

Upper, middle, and lower address refer to the three positions that a 30-bit central processor instruction can occupy within a word.

Codes of the form 1x1x specify simultaneous and independent relocation of both the upper and lower address fields.

The address fields referenced are 18-bit fields, as shown in figure D-11. These address fields are the three positions that a 30-bit central processor instruction can occupy within a word.

The relocation is performed by adding the base address of the subprogram's local central memory block to the content of an address field that is assumed to contain a relative address. Relocation using any other base address, or involving address fields differing in length and/or position from those shown in figure D-11, can be accomplished only with the XTEXT and XFILL tables.

A TEXT table can load text into an ECS block, provided the relative starting address, s, does not exceed 377777_8.

XFILL TABLE (4100)

The XFILL table (figure D-12) serves the same purpose as the FILL table, but with most of the restrictions eliminated.

Each relocation descriptor (figure D-13) is one 60-bit word and specifies the relocation of one address field in a previously loaded text word. The values for br are the same as the values for rb shown in table D-2; the relocation quantity has the same values as the values of b also shown in table D-2.

The relocation quantity is added to the content of the address field of the previously loaded text word in bit positions (pos+size-1) through (pos), where pos is the bit position of the low-order bit of the address field in the text word, and size is the address field length in bits. The location of the previously loaded text word is given by the relocation indicator for the text word address (rb) and the relative address of the text word (a). In doing the addition, the relocation quantity is lengthened to 60 bits by sign extension and then shortened to the length of the field to be modified by simple truncation of the bits at the left end. The relocation quantity is then added to the field to be modified as if in a register of the same length as the field to be modified, with end-around carry; that is, the addend and augend are treated as having a sign bit and (size-1) magnitude bits.

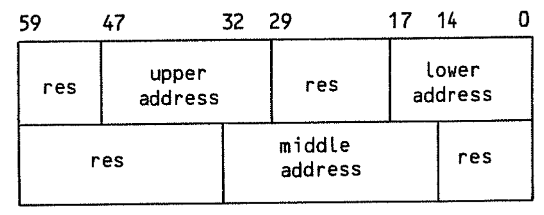
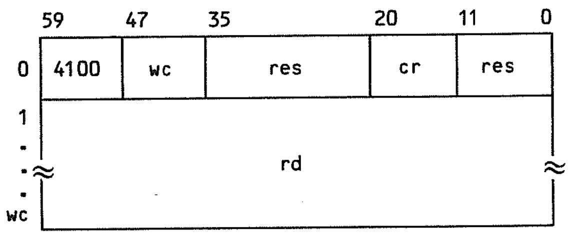
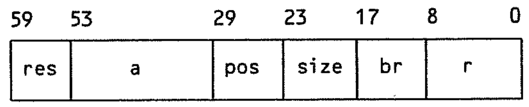


Figure D-11. TEXT Table Address Field Format



wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
cr	The conditional relocation base designator. The XFILL table is ignored if cr refers to a common block that was first declared by an earlier subprogram.
rd	Relocation descriptors; each is one word and specifies the relocation of one address field.

Figure D-12. XFILL Table Format



res	Reserved for use by CDC.
a	Relative address of the text word; can be as large as 7777777_8.
pos	Bit position of the low-order bit of the address field in the text word.
size	The address field length in bits.
br	Base address designator for the relocation quantity.
r	Relocation base designator for the text word address; r cannot be 2 (negative program relocation) and cannot refer to a blank common block.

Figure D-13. XFILL Table Relocation Descriptor Format

Because address fields cannot cross word boundaries, the following restrictions must be satisfied:

- $0 \leq pos \leq 59$
- $1 \leq size \leq 60$
- $1 \leq pos + size \leq 60$

FILL TABLE (4200)

The FILL table (figure D-14) provides for relocating address fields in text words previously loaded with TEXT, XTEXT, and immediate REPL and XREPL tables. During the loading process, FILL tables are saved when encountered and are processed at load completion.

Each relocation sequence consists of a 30-bit header byte (figure D-15) followed by any number of 30-bit trailer bytes (figure D-16). These bytes can be in either the upper or lower half of a word, although the figures show both in the lower half.

The header byte (figure D-15) specifies a base address, which is to be added to the contents of the address fields specified by the trailer bytes. The base address (b) is a signed integer whose value depends on br. The values for br are the same as the values for rb shown in table D-2.

In figure D-16, address a of the text word in which the content of address field p is to be relocated is a+b, where b is determined by rb, as shown in table D-2. However, rb cannot be 2 (negative program relocation) and cannot refer to a blank common block. Relocation of text words having a relative address greater than 377777_8, or of address fields having lengths and positions other than the three normal address fields shown in figure D-16, can be accomplished with the XFILL table only.

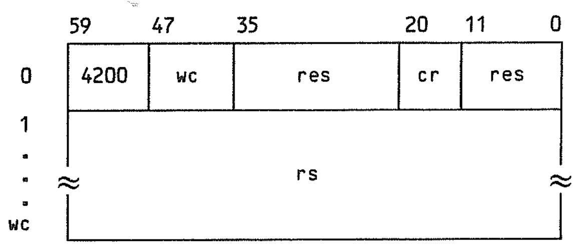
A FILL table can be used to relocate text in ECS blocks, provided the range restrictions described for address a, in the preceding paragraph, are satisfied. The br in the header byte could refer to an ECS block, but this should be used with caution because the address fields in the text words are only 18 bits long. Overflow is ignored.

REPL TABLE (4300)

The REPL table (figure D-17) causes replication of one or more copies of a block of previously loaded text words so that fewer TEXT and XTEXT tables are required.

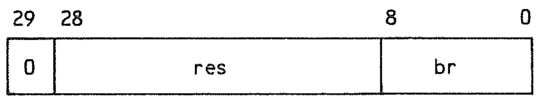
Each replication descriptor (figure D-18) is two words long.

The starting source address (S) is determined by as+BS, where as is the relative source address and BS is the relocation base address. BS depends on the relocation base designator for the source address (rs). BS can have the same values as b in table D-2, rs can have the same values as rb in table D-2. The value for rs cannot be 2 (negative program relocation), nor can rs refer to a blank common block.



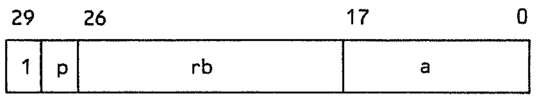
wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
cr	The conditional relocation base designator. The FILL table is ignored if cr refers to a common block that was first declared by an earlier subprogram.
rs	Relocation sequences; each consists of a 30-bit header byte (figure D-15) followed by a 30-bit trailor byte (figure D-16).

Figure D-14. FILL Table Format



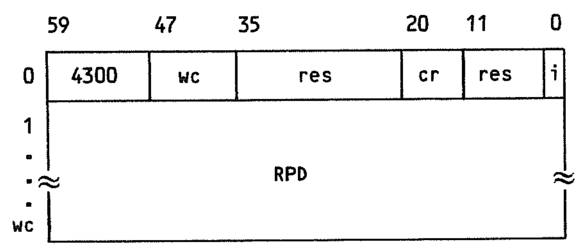
res	Reserved for use by CDC.
br	Relocation base designator.

Figure D-15. FILL Table Header Byte Format



p	The address field to be relocated, designated as follows: 00 = Lower address (bits 17 through 0) 01 = Middle address (bits 32 through 15) 10 = Upper address (bits 47 through 30)
rb	Relocation base designator.
a	Relative address; cannot exceed 377777_8.

Figure D-16. FILL Table Trailer Byte Format



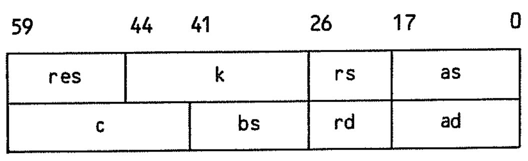
wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
cr	Conditional relocation base designator. The REPL table is ignored if cr refers to a common block that was first declared by an earlier subprogram.
i	The flag that determines when REPL table is processed; the value of i can be the following: 0 The REPL table is saved when encountered and processed when the end of the current load file or library is reached. 1 The REPL table is processed immediately when encountered.
RPD	Replication descriptors; each is two words long and is used to copy a block of previously loaded text.

Figure D-17. REPL Table Format

The starting destination address (D) is determined by ad+BD, where ad is the relative destination address and BD is the relocation base address. BD depends on rd in the same manner as BS depends on rs. If D is 0, D=S+b is assumed.

The loader copies the number of text words determined by the block size (bs) beginning at S to the number of text words (bs) beginning at D. The same text words are then copied to the number of text words (bs) beginning at D plus the destination address increment (k), then to D+2xk, and so on. The loader continues this process until the block has been copied the total number of times designated by the count (c).

The values of k and c cannot exceed 377777_8. The value of b cannot exceed 77777_8. Source and destination fields having relative addresses greater than 377777_8 can be specified only with the XREPL table. The source and destination fields can be in either type of memory (central memory or ECS), provided the above range restrictions are satisfied, but the two fields must be in the same type of memory.



res	Reserved for use by CDC.
k	Destination address increment: k is added to destination address (D) after each copy; if k is 0, the loader uses block size (bs) for increment size.
rs	Relocation base designator for the source address (S).
as	Relative source address; cannot exceed 377777_8.
c	Count; the number of times the block is copied. If c is 0 or 1, the loader makes one copy.
bs	Block size (number of words to be copied) in 60-bit words. If bs is 0 or 1, the loader copies one word.
rd	Relocation base designator for destination address D.
ad	Relative destination address; cannot exceed 377777_8.

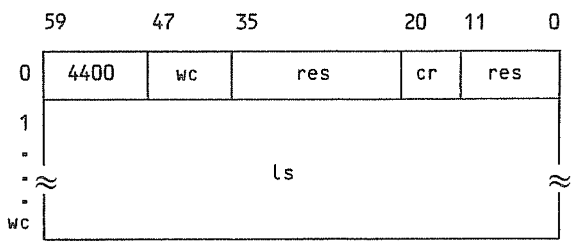
Figure D-18. REPL Table Replication Descriptor Format

LINK TABLE (4400)

The LINK table (figure D-19) provides for external reference linkage. The value of entry points defined in other subprograms is added to address fields in text words loaded with TEXT and XTEXT tables in the present subprogram. During the loading process, LINK tables are saved when encountered and are processed when the end of the current load file or library is reached.

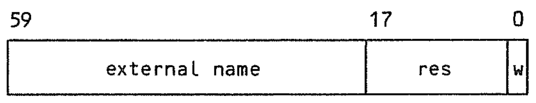
Each linkage sequence consists of a 60-bit header byte (figure D-20) followed by any number of 30-bit trailer bytes. The header byte can begin in the middle of one word and end in the middle of the next word. If the last trailer byte of a sequence ends in the middle of a word, it can be followed by 30 zero bits to fill out the word. These 30 zero bits are ignored rather than being taken as part of the next header byte.

The first character of the name must have a display code octal value 01 through 37; that is, it must be a letter A to Z or a digit 0 to 4, so that the leftmost bit of the header byte is a 0 to distinguish it from trailer bytes. The trailer bytes have the same format and interpretation as in the FILL table (figure D-16). The value of the external name, defined by an ENTR table in some other subprogram, is added to the contents of the address fields specified in the trailer bytes, using 18-bit signed integer arithmetic with overflow ignored.



wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
cr	Conditional relocation base designator. The LINK table is ignored if cr refers to a common block that was first declared by an earlier subprogram.
ls	Linkage sequences; each consists of a 60-bit header byte (figure D-20) followed by any number of 30-bit trailer bytes.

Figure D-19. LINK Table Format



res	Reserved for use by CDC.
w	Weak external flag:
	0 = strong
	1 = weak

Figure D-20. LINK Table Header Byte Format

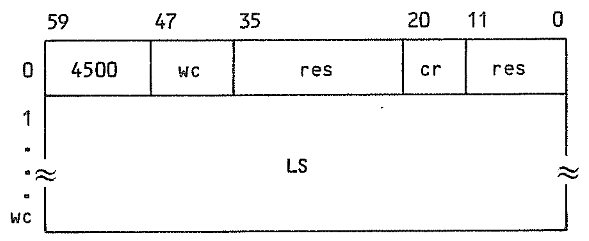
XLINK TABLE (4500)

The XLINK table (figure D-21) serves the same purpose as the LINK table, but with most of the restrictions eliminated.

Each linkage sequence (figure D-22) is one or more words. The all-zero word terminating the linkage sequence can be omitted for the last or only linkage sequence in the XLINK table. The first character of the external name can be any character code except 55_8 (blank) or 00.

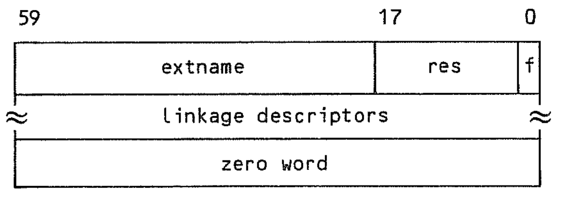
Each linkage descriptor (figure D-23) is one word.

The value of the external name, defined by an ENTR table in some other subprogram, is added to the content of an address field in a previously-loaded text word. In doing the addition, the relocation quantity is, in effect, lengthened to 60 bits by sign extension and then shortened to the length of the field to be modified by simple truncation of the bits at the left end. The relocation quantity is then added to the field to be modified as if in a register of the same length as the field to be modified, with end-around carry; that is, the addend and the augend are treated as having a sign bit and (size-1) magnitude bits.



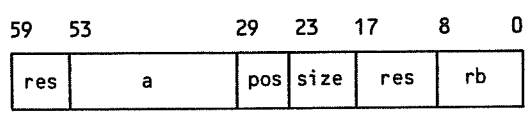
wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
cr	The conditional relocation base designator. The XLINK table is ignored if cr refers to a common block that was first declared by an earlier subprogram.
LS	Linkage sequences composed of one or more linkage descriptors. Each linkage sequence is one or more words long.

Figure D-21. XLINK Table Format



extname	The name of an external, which is defined by an ENTR table in another subprogram.
res	Reserved for use by CDC.
f	The flag which indicates whether the external is weak or strong. Values for f can be as follows:
	0 strong
	1 weak

Figure D-22. XLINK Table Linkage Sequence Format



res	Reserved for use by CDC.
a	Relative address of the text word; can be as large as 777777_8.
pos	Bit position of the low-order bit of the address field in the text word.
size	The address field length in bits.
rb	Relocation base designator for the text word address; rb cannot be 2 (negative program relocation) and cannot refer to a blank common block.

Figure D-23. XLINK Table Linkage Descriptor Format

XFER TABLE (4600)

The XFER table (figure D-24) specifies the transfer point for the program being loaded (the address at which execution is to begin).

The table is ignored if the transfer name is seven blanks. Otherwise, the name must be defined as an entry point by some subprogram, not necessarily the subprogram containing the XFER table.

XREPL TABLE (4700)

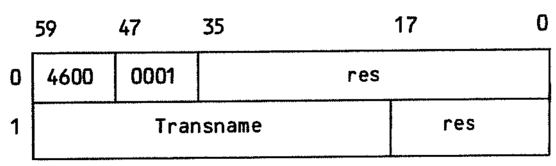
The XREPL table (figure D-25) serves the same purpose as the REPL table, but with most of the restrictions eliminated.

Each replication descriptor (figure D-26) is two words long.

ASCM TABLE (5000)

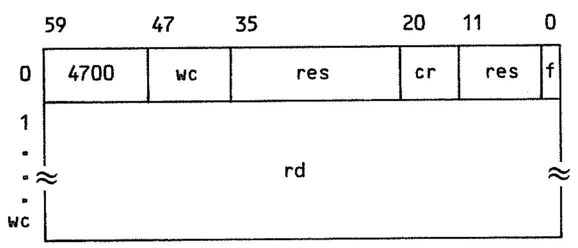
The ASCM table (figure D-27) contains the absolute central memory image of either a central processor program or overlay with one unnamed entry point.

If level 1=level 2=0, the program is a main overlay and must have fwa=100. If level 1 is nonzero and level 2 is zero, the overlay is a primary overlay. If level 1 and level 2 are both nonzero, the overlay is a secondary overlay and is dependent on the primary overlay with the same level 1 value. An exception is system text, which has both fwa and entry set to 0.



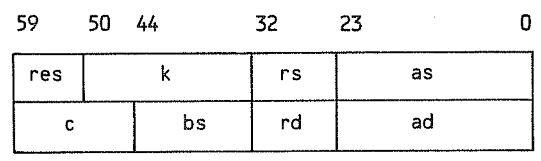
res	Reserved for use by CDC.
Transname	The name of the transfer point of the program being loaded; must have been defined as an entry point by a program.

Figure D-24. XFER Table Format



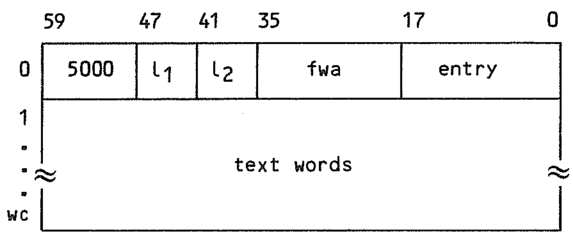
wc	The number of 60-bit words in the table, not counting the header word.
res	Reserved for use by CDC.
cr	Conditional relocation base designator. The XREPL table is ignored if cr refers to a common block that was first declared by an earlier subprogram.
f	The flag that determines when the XREPL table is processed. Values for f can be as follows:
	0 The XREPL table is saved when encountered and processed when the end of the current load file or library is reached.
	1 The XREPL table is processed immediately when encountered.
rd	Replication descriptors; each is two words long.

Figure D-25. XREPL Table Format



res	Reserved for use by CDC.
k	Destination address increment: k is added to destination address (D) after each copy; if k is 0, the loader uses block size (b) for increment size.
rs	Relocation base designator for the source address (S).
as	Relative source address; cannot exceed 7777777_8.
c	Count; the number of times the block is copied. If c is 0 or 1, the loader makes one copy. Cannot exceed 32767.
bs	Block size (number of words to be copied) in 60-bit words. If bs is 0 or 1, the loader copies one word. Cannot exceed 4095.
rd	Relocation base designator for destination address D.
ad	Relative destination address; cannot; exceed 7777777_8.

Figure D-26. XREPL Table Replication Descriptor Format

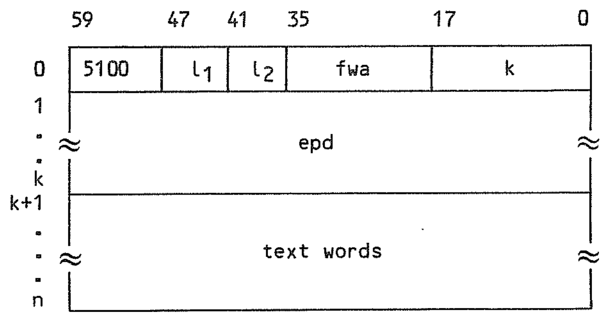


l_1 l_2	Overlay level numbers. If l_1=l_2=0, the program must have fwa=100.
fwa	Address of the word in central memory into which the table header word is to be loaded; the first text word is loaded at central memory address fwa+1.
entry	The address at which execution of the program or overlay is to begin.

Figure D-27. ASCM Table Format

EASCM TABLE (5100)

The EASCM table (figure D-28) contains the absolute central memory image of either a central processor program or overlay, with one or more named entry points.



l_1,l_2	Same as in the ASCM table. If l_1=l_2=0, the same restrictions on fwa apply.
fwa	The address of the word in central memory into which the table header word is to be loaded. The first entry point definition is loaded at central memory address fwa+1, and the first text word at fwa+k+1.
k	The number of entry point definitions.
epd	The entry point definitions; each provides the name and location of an entry point to the overlay.

Figure D-28. EASCM Table Format

If level 1=level 2=0, the same restrictions apply as described for the ASCM table.

Each entry point definition (figure D-29) provides the name and location of an entry point to the overlay.

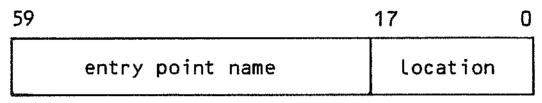


Figure D-29. EASCM Table Entry Point Definition Format

ACPM TABLE (5300)

The ACPM table (figure D-30) contains the absolute central memory image of either a central processor program or overlay, with one unnamed entry point or one or more named entry points. It can also contain an absolute ECS image.

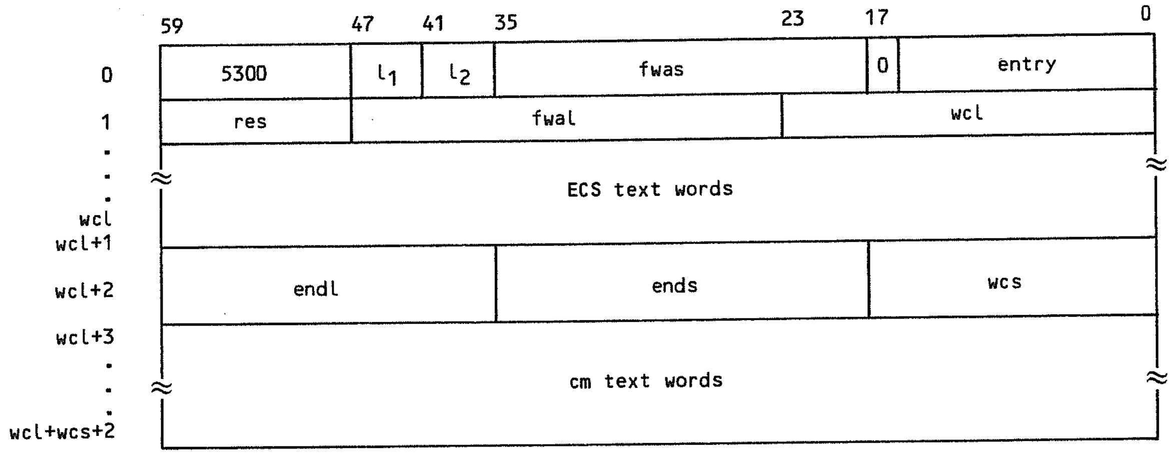
The greatest address+1 of the fixed ECS area used by this program (endl) and the greatest address+1 of the fixed central memory area used by this program (ends) are the origins of the central memory and ECS areas that can be used for dynamic storage allocation. The value for ends must not be less than fwas+k+wcs, where fwas is the address of the word in central memory into which the table header word is to be loaded, k is the number of entry point definitions, and wcs is the number of central memory text words. The value for ends must not exceed the central memory field length at execution time. Similarly, endl must not be less than fwal+wcl, where fwal is the address of the word in ECS into which the first ECS text word (if any) is to be loaded, and wcl is the number of ECS text words. The value for endl must not exceed the ECS field length at execution time.

For a (0,0) overlay, ends and endl contain the highest high address required to load any of the overlays originating from the (0,0) overlay. For non-(0,0) overlays, ends and endl are ignored.

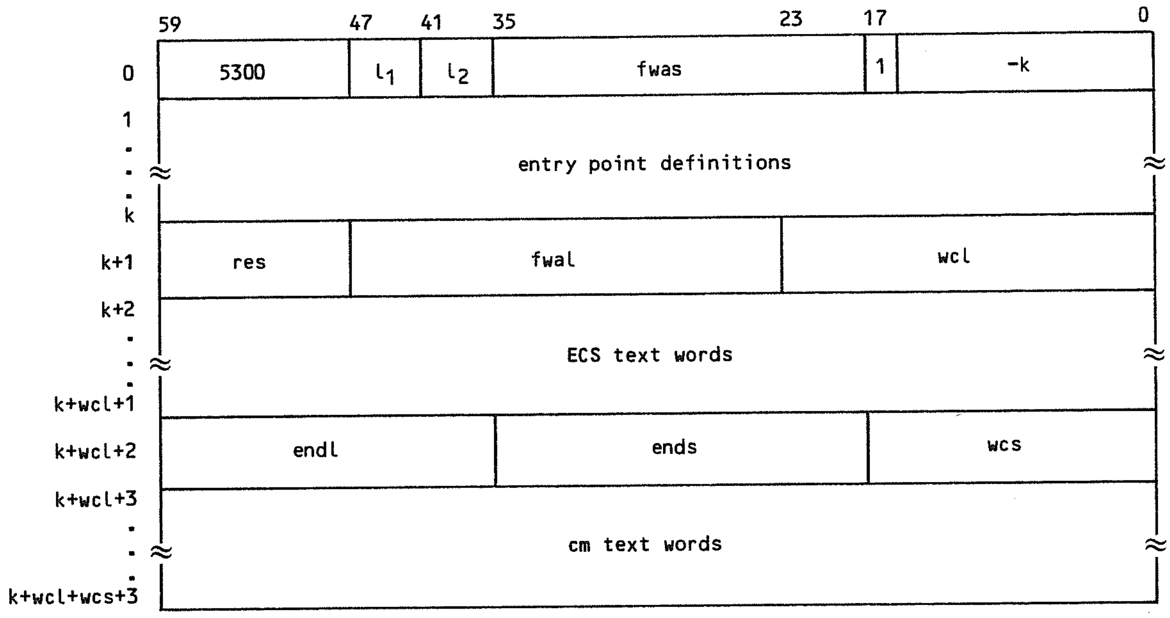
EACPM TABLE (5400)

The EACPM table (figure D-31) contains an absolute central memory program or an overlay and (optionally) an associated ECS image. The loader generates all absolute overlays as EACPM tables.

Format 1 (one unnamed entry point):



Format 2 (one or more named entry points):



$l_1 l_2$	The overlay level number.
fwas	The address of the word in central memory into which the table header word is to be loaded. For format 1, the first central memory text word is loaded at address fwas+1. For format 2, the first entry point definition is loaded at address fwas+1, and the first central memory text word at fwas+k+1. If $l_1+l_2=0$, fwas must not be less than 100_8.

Figure D-30. ACPM Table Format (Sheet 1 of 2)

entry	The address at which execution of the program or overlay is to begin.
res	Reserved for use by CDC.
k	The number of entry point definitions. The complement of k is stored in the table to distinguish format 2 from format 1.
entry point definitions	Same as in the EASCM table.
fwal	The address of the word in ECS into which the first ECS text word (if any) is to be loaded.
wcl	The number of ECS text words; can be zero.
endl	The greatest address+1 of the fixed ECS area used by this program. If endl is 0, endl=fwal+wcl is assumed.
ends	The greatest address+1 of the fixed central memory area used by this program. If ends is 0, ends=fwas+wcs+1 is assumed.
wcs	The number of central memory text words. If wcs is 0, the central memory text words comprise all that remains of the ACPM table, which is terminated by an end-of-record.
cm text words	The absolute central memory image of either a central processor program or overlay.
ECS text words	The absolute ECS memory image of either a central processor program or overlay.

Figure D-30. ACPM Table Format (Sheet 2 of 2)

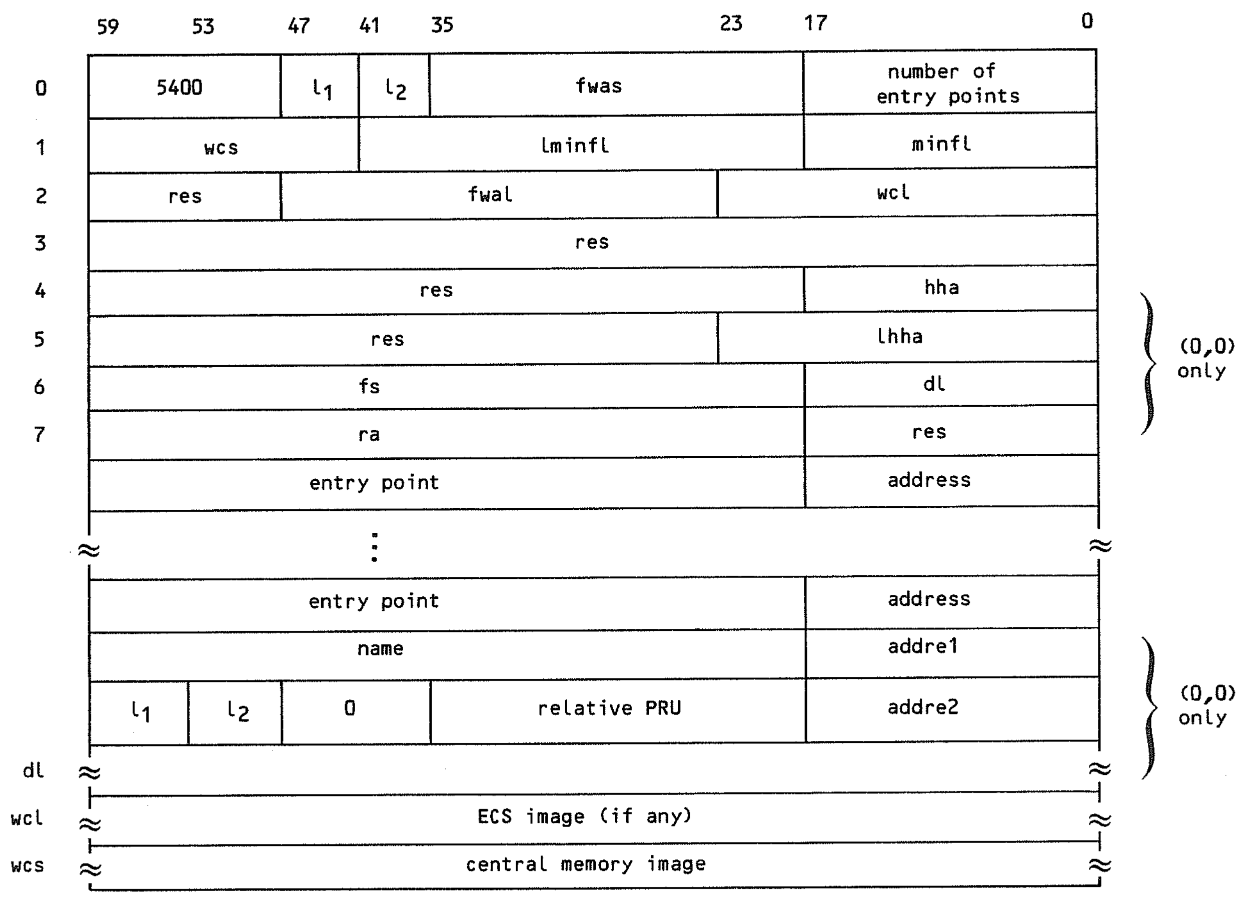


Figure D-31. EACPM Table Format (Sheet 1 of 2)

l_1,l_2	Overlay level. Words 4 through 7 of the table header are present only for level (0,0) overlays.
fwas	The address of the word in central memory into which word 0 of the table header is to be loaded. The entry point list, the FOL directory, and then the central memory image are to be loaded immediately following the header. If $l_1=l_2=0$, fwa must be equal to 100.
number of entry points	The number of entry points in the absolute central memory program or the overlay.
wcs	The number of words in the central memory image.
lminfl	The minimum ECS field length needed to execute the overlay.
minfl	The minimum central memory field length needed to execute the overlay; equivalent to the lwa + 1 of the overlay.
res	Reserved for use by CDC.
fwal	The address of the word in ECS, if any, into which the first word of the ECS image is to be loaded.
wcl	The number of words in the ECS image.
hha	The highest high address for central memory; the minimum field length needed to execute any legal combination of overlays generated as part of this overlay structure. This number is derived from the values of minfl specified for each of the overlays.
lhha	The highest high address for ECS.
fs	The file specification entry set into the 5400 table when a FOL overlay structure (0,0) overlay is loaded.
dl	The fast overlay directory length. Two words are used for each entry.
ra	The random address set into the 5400 table when a FOL overlay structure (0,0) overlay is loaded.
entry points	If the overlay was generated because of OVERLAY directives in the object stream, or because of a NOGO control statement specifying a file name only, the overlay will have a single entry point. Its name will be the same as that of the overlay and its address will be that of the last transfer address encountered. If the overlay was generated because of a NOGO control statement (and no OVERLAY directives) specifying a file name and one or more entry points, the overlay will contain those entry points named on the NOGO statement with their respective addresses.
address	The entry point address.
name	The name of the overlay or OVCAP.
addre1	The address of the first word of the overlay. Zero if an OVCAP.
relative PRU	The PRU address relative to the 0,0 overlay.
addre2	The address of the last word of the overlay plus 1. Length if an OVCAP.

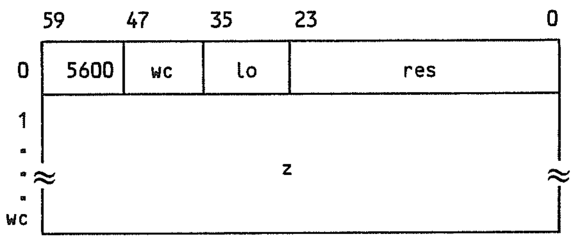
Figure D-31. EACPM Table Format (Sheet 2 of 2)

SYMBOL TABLE (5600)

The SYMBOL table (figure D-32) contains symbol information for the language designated in the header word. The SYMBOL table is used for interactive debugging.

LINE NUMBER TABLE (5700)

The LINE NUMBER table (figure D-33) contains line number information and addresses of object code for the language designated in the header word. The LINE NUMBER table is used for interactive debugging.



wc	The number of 60-bit words in the table, not counting the header word.
lo	Language ordinal, for example: 2=FTN4 4=FTN5 8=BASIC
res	Reserved for use by CDC.
z	Symbol table entry. Each entry is 2 words long. The content of this field is irrelevant to the loader.

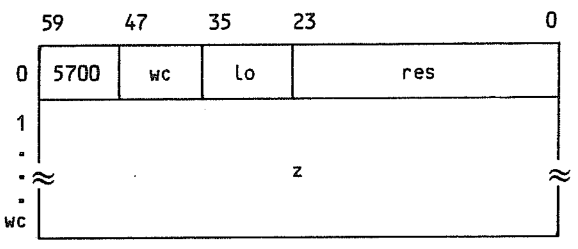
Figure D-32. SYMBOL Table Format

CAPSULE TABLE (6000)

The CAPSULE table (figure D-34) contains relocatable code in a form that allows the code to be quickly loaded and relocated by the Fast Dynamic Loader.

The CAPSULE table is divided into six parts: the header, code image, entry point list, external reference list, reference chains, and the relocation table.

The header (figure D-35) is three words long. FLAG=0 indicates a capsule; FLAG=1 indicates an overlay-capsule.



wc	The number of 60-bit words in the table, not counting the header word.
lo	Language ordinal.
res	Reserved for use by CDC.
z	Line number table entry. The content of this field is irrelevant to the loader.

Figure D-33. LINE NUMBER Table Format

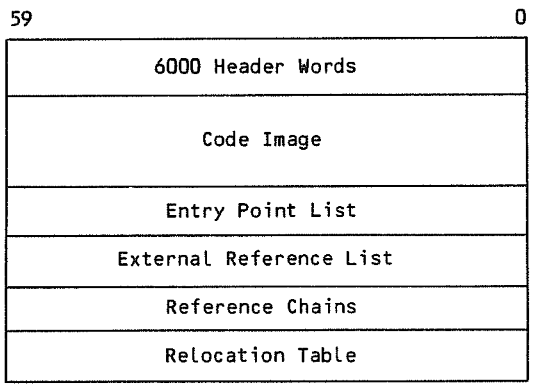
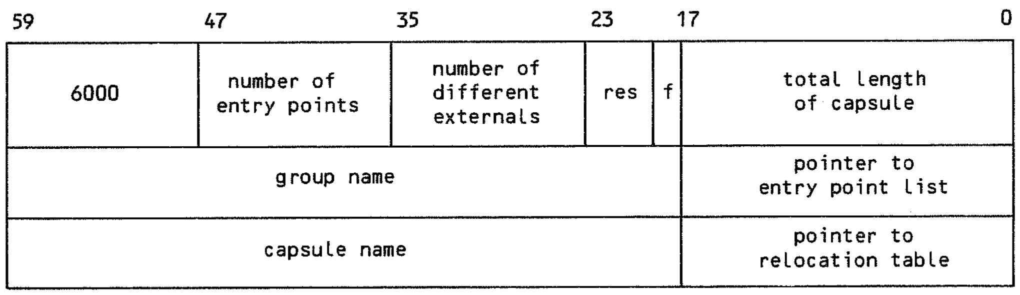


Figure D-34. CAPSULE Table Format



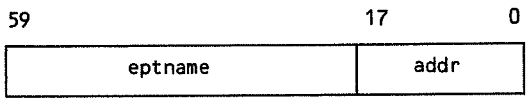
res	Reserved for use by CDC.
f	The flag that indicates the kind of capsule; f can have the following values: 0 Capsule 1 OVCAP (overlay capsule)

The values of the pointers are relative to the start of the 6000 table.

Figure D-35. CAPSULE Table Header Format

The code image contains the actual executable code and is relocated so that the first word of the 6000 header corresponds to zero.

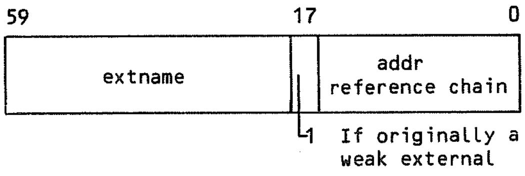
The entry point list (figure D-36) is alphabetical and contains one word for each entry point.



eptname	Entry point name; the entry point names are listed in alphabetic order in the list.
addr	Address of entry point relative to the first word within this table.

Figure D-36. CAPSULE Table Entry Point List Format

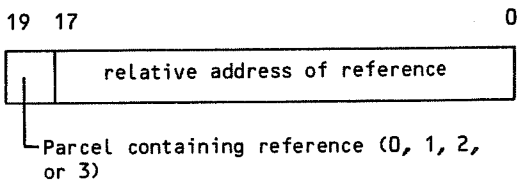
The external reference list (figure D-37) consists of an alphabetical list of external names.



extname	External name; external names are listed in alphabetic order.
addr	Address of reference chain relative to the first word within the table.

Figure D-37. CAPSULE Table External Reference List Format

The reference chain for a particular external consists of a series of 20-bit fields packed three per central memory word. The first is zero for use by the execution-time subroutines. The format of other reference chains is shown in figure D-38.


19
17
0
relative address of reference
Parcel containing reference (0, 1, 2, or 3)

0 = Termination of reference chain

1 = Upper parcel

2 = Middle parcel

3 = Lower parcel

Figure D-38. CAPSULE Table Reference Chain Format

The chain is terminated by an entry with zero in the parcel field.

The relocation table (figure D-39) is similar to the relocation portion of the TEXT table. One 4-bit relocation indicator corresponds to each word of the header and code image. Refer to the discussion of the TEXT table (figure D-10) for a description of the relocation indicator.

LDSET TABLE (7000)

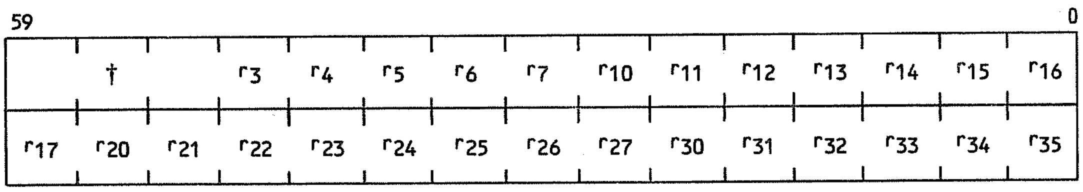
The LDSET table (figure D-40) allows object directives to be incorporated into a relocatable object program in an internal format. The objective directives have the same effect as the corresponding LDSET control statement options.

Several compilers (for example, FTN and COBOL5) and the loader generate LDSET tables automatically. The user is often unaware of the existence of these tables in the program unit.

In the LDSET table, each object directive begins with, or consists of, a header word with the same general format as a table header word. The object directives, and the names of the LDSET directive options to which they correspond, are described below. Any or all of them can be present, in any order.

LIB Option (0010)

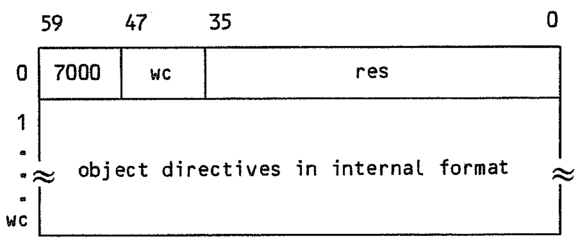
The LIB option (figure D-41) identifies one or more library names comprising the local library set.



r_i The 4-bit relocation indicator.

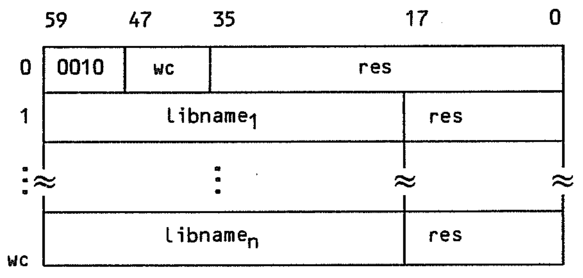
†Zero, indicating that the three header words are not to be relocated.

Figure D-39. CAPSULE Table Relocation Table Format



wc	The number of words in the table, not counting the header word.
res	Reserved for use by CDC.

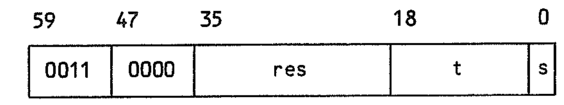
Figure D-40. LDSET Table Format



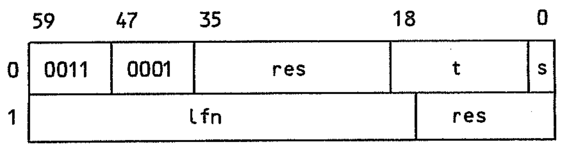
wc	The number of library names listed in the table.
res	Reserved for use by CDC.
libname	The name of the library to be added to the local library set.

Figure D-41. LIB Option Format

Format 1:



Format 2:



res Reserved for use by CDC.

t The map type octal code. These options are equivalent to LDSET(MAP=) options. Any combination of the following can be specified:

1 (bit 1) = Statistics (S)

2 (bit 2) = Blocks (B)

4 (bit 3) = Entry points (E)

10 (bit 4) = Entry point cross-references (X)

s Significance of the t flag:

0 = Ignore t and write the default map type

1 = Write the map type specified by t

lfn The local file that is to receive the map.

Figure D-42. MAP Option Formats

MAP Option (0011)

The MAP option has two formats as shown in figure D-42.

Format 1 of the MAP option specifies the type of map to be written. The map is written on the file OUTPUT.

Format 2 of the MAP option has the same interpretation of the descriptors s and t. The map is written on the specified file.

PRESET/PRESETA Options (0012)

The PRESET option has two formats as shown in figure D-43.

Format 1 of the PRESET option specifies a value to be stored into each word of the memory image that is not set by TEXT, XTEXT, REPL, and XREPL tables. If p is zero, the value is used as is. If p is nonzero (PRESETA), the location of each word is stored in its lower 17 bits if central memory, or lower 24 bits if ECS.

Format 2 is the internal form of LDSET(PRESET=NONE).

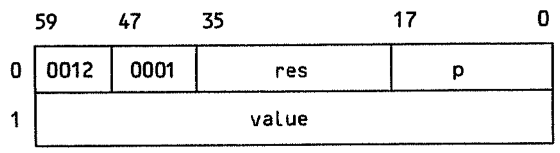
ERR Option (0013)

The ERR option (figure D-44) specifies the error severity that causes the loader to abort.

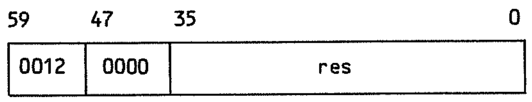
REWIND/NOREWIN Option (0014)

The REWIND/NOREWIN option (figure D-45) specifies the default initial position for all files to be read by the loader.

Format 1:



Format 2:

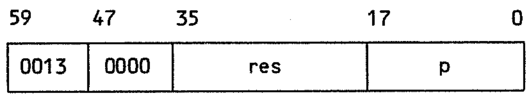


res	Reserved for use by CDC.
p	A parameter which specifies whether the value stored into each word of the core image is used as is:

p=0	the value is used as is.
p≠0	the location of each word is stored in its lower 17 bits if central memory, or lower 24 bits if ECS.

val	The value to which unused memory is set before execution of the loaded program.

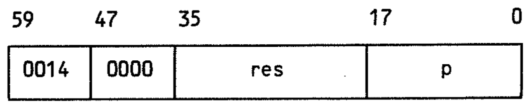
Figure D-43. PRESET/PRESETA Option Formats



res	Reserved for use by CDC.
p	The error severity that causes the loader to abort; p can be the following:

0	all, the program aborts if any loader error is encountered.
1	fatal, the program aborts for fatal and catastrophic loader errors.
2	none, the program aborts for catastrophic loader errors only.

Figure D-44. ERR Option Format



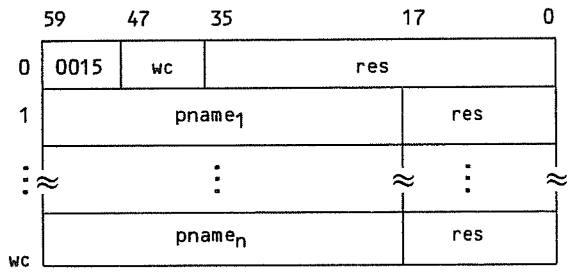
res	Reserved for use by CDC.
p	Default initial position for all files to be loaded; p can be the following:

0	Files are rewound
1	Files are not rewound

Figure D-45. REWIND/NOREWIN Option Format

USEP Option (0015)

The USEP option (figure D-46) identifies one or more object programs that are loaded whether or not they are needed to satisfy external references. The loader loads the programs on the next occasion that it satisfies externals.

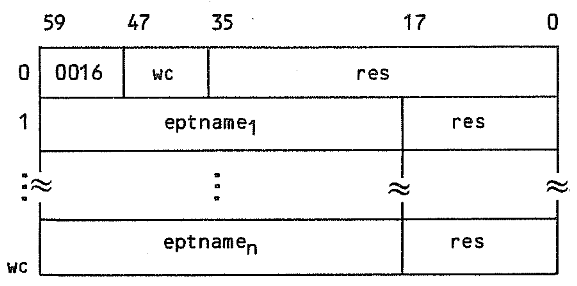


wc	The number of object programs listed in the table.
res	Reserved for use by CDC.
pname	The name of the object program to be loaded.

Figure D-46. USEP Option Format

USE Option (0016)

The USE option (figure D-47) identifies one or more object programs that are loaded to assure that specified entry points are included in the load. The loader loads the programs on the next occasion that it satisfies externals.



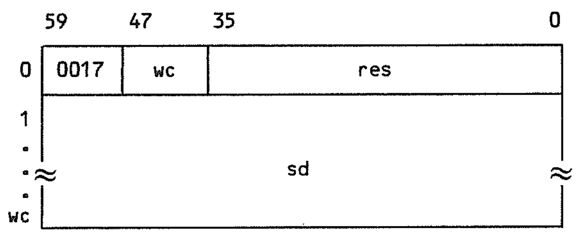
wc	The number of entry names listed in the table.
res	Reserved for use by CDC.
eptname	An entry name that is to be included in the load.

Figure D-47. USE Option Format

SUBST Option (0017)

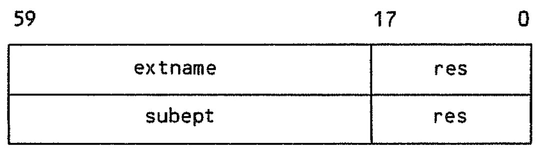
The SUBST option (figure D-48) supplies entry point names to be used in place of external names appearing in all subsequent LINK and XLINK tables.

Each substitution descriptor is two words long and has the format shown in figure D-49.



res	Reserved for use by CDC.
sd	Substitution descriptors, each is two words long.

Figure D-48. SUBST Option Format

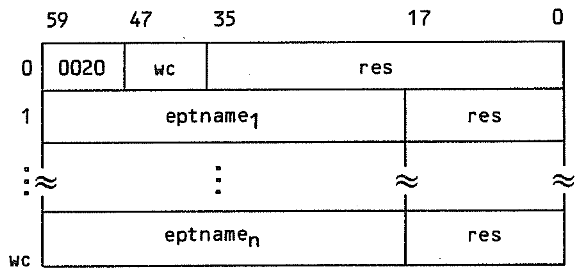


extname	The external name.
res	Reserved for use by CDC.
subept	The substitute entry point name that is to be used in place of text name in subsequent LINK or XLINK tables.

Figure D-49. SUBST Option Substitution Descriptor Format

OMIT Option (0020)

The OMIT option (figure D-50) identifies one or more entry point names that are to remain unsatisfied, whether or not the program containing these names is loaded. The loader processes these names in the same manner that it processes other unsatisfied entry points, but no errors result.

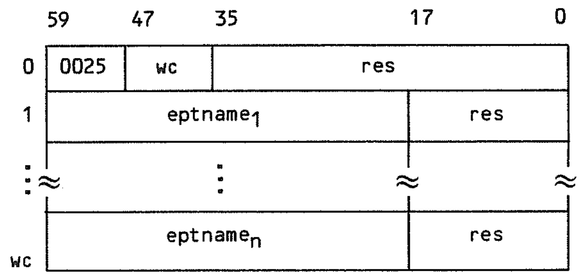


wc	The number of entry points listed in the table.
res	Reserved for use by CDC.
eptname	The entry point name that is to remain unsatisfied.

Figure D-50. OMIT Option Format

EPT Option (0025)

The EPT option (figure D-51) identifies the entry point names of capsules, overlays, and OVCAPS.

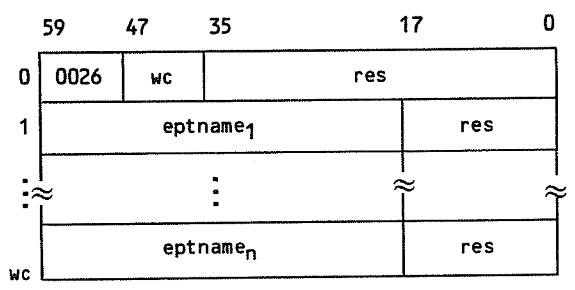


wc	The number of entry points listed in the table.
res	Reserved for use by CDC.
eptname	The entry point name to be used in a capsule, overlay or OVCAP.

Figure D-51. EPT Option Format

NOEPT Option (0026)

The NOEPT option (figure D-52) identifies one or more entry point names that are to be ignored in capsules, overlays, or OVCAPS.

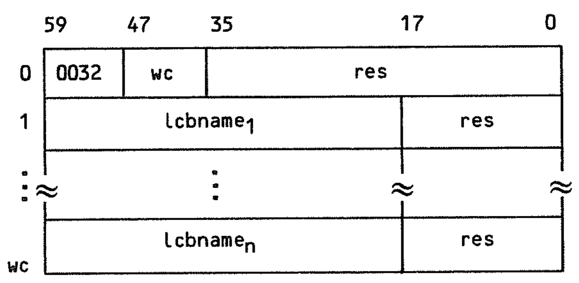


wc	The number of entry points listed in the table.
res	Reserved for use by CDC.
eptname	The entry point names that are not to be used in a capsule, overlay or OVCAP.

Figure D-52. NOEPT Option Format

COMMON Option (0032)

The COMMON option (figure D-53) identifies the labeled common blocks that are to be moved to the nearest common ancestor of all the segments that reference them.

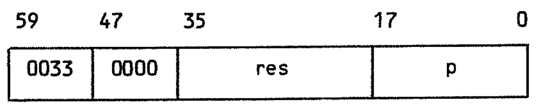


wc	The number of labeled common blocks listed in the table.
res	Reserved for use by CDC.
lcbname	The labeled common block that is to be moved to the nearest common ancestor of the segment that references it.

Figure D-53. COMMON Option Format

PD Option (0033)

The PD option (figure D-54) identifies the print density of the load map.

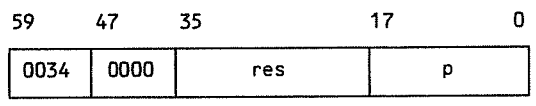


res	Reserved for use by CDC.
p	Print density for load map (6 or 8 lines/inch).

Figure D-54. PD Option Format

PS Option (0034)

The PS option (figure D-55) identifies the page size of the load map.



res	Reserved for use by CDC.
p	Page size for load map (lines/page). Must be at least 10.

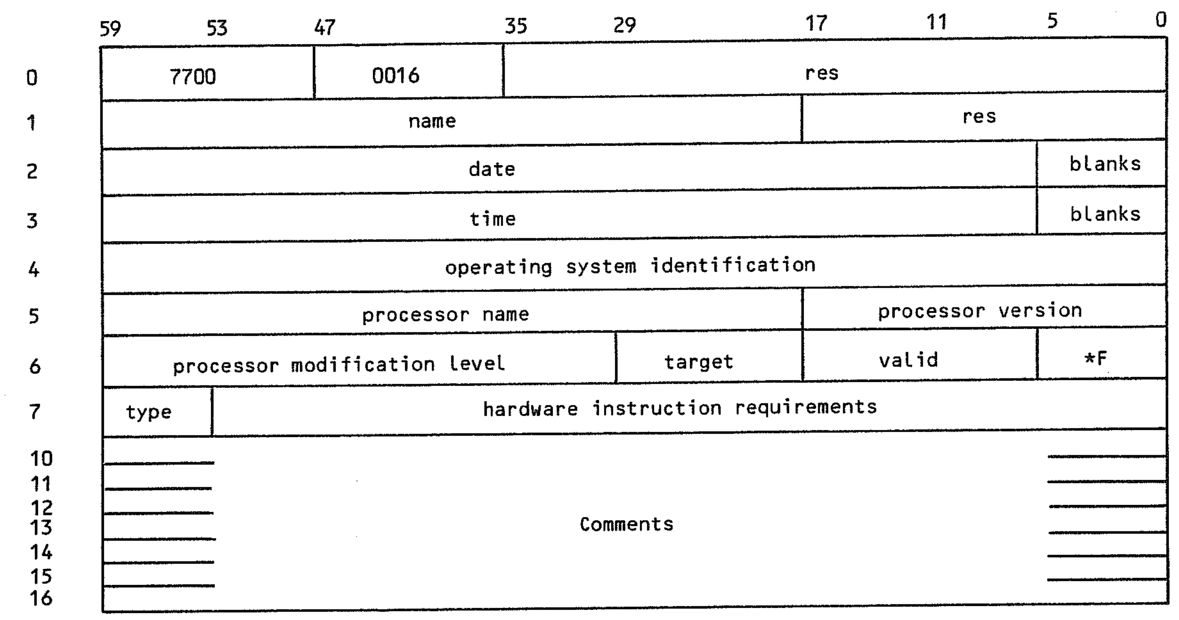
Figure D-55. PS Option Format

PRFX Table (7700)

The PRFX table (figure D-56) identifies and serves as a header for all types of object programs. The table is also used with other types of binary data, such as system texts. The loader and other programs print selected portions of the prefix table, and ITEMIZE prints the entire table.

The word count (bits 36 through 47 of the header word) can be any nonnegative value. 16_8 is standard.

Unused characters in words 2 through 7 should be blank-filled to allow them to be printed.



res	Reserved for use by CDC.
name	The name of the program or subprogram, left-justified with zero fill. For PRFX tables generated by the loader, this is the name of the first program of the overlay; not necessarily the name of the main program (the one at which execution begins)
date	The date the table was generated, left-justified, in the form yy/mm/dd or mm/dd/yy depending on the installation option.
time	The time the table was generated, left-justified, in the form hh.mm.ss.
operating system identification	The name and version number of the operating system under which the table was generated (for example, NOSΔ2.4.2Δ or NOS/BEΔ1.5). (Δ denotes blank.)
processor name and version	The name and version number of the program generating the table (for example, FTNΔΔΔ5.1 or LOADERΔ1.5).
processor modification level	The modification level of the program generating the table, such as a PSR summary number or a Julian date (for example, 647ΔΔor 85310).
target	Two characters indicating the type of processor for which the program is optimized. Recommended values are:

64 = 6400-type CPU

66 = 6600-type CPU

6X = Either of the above

C5 = CYBER 175 CPU

CX = Any of the above

76 = 7600-type CPU

ΔΔ = Any of the above, or not applicable

6P = 6000-type PPU

7P = 7000-type PPU

8P = 180-type PPU

Figure D-56. PRFX Table Format (Sheet 1 of 2)

	The 6200, 6400, 6500, CDC CYBER 70 models 71, 72, 73, and CDC CYBER 170 models 172, 173, 174, 720, and 730 contain one or two 6400-type CPUs and several PPUs. The 6600, and CDC CYBER 70 model 74, and the CDC CYBER 170 models 175, 740, 750, and 760 contain a 6600-type CPU and several 6000-type PPUs. The 6700 and dual-CPU CDC CYBER 70 model 74 contain both a 6400-type CPU and 6600-type CPU, as well as several 6000-type PPUs. The 7600, 7700, and the CDC CYBER 170 model 176 contain one or two 7600-type CPUs.
valid	Two characters indicating the type of processor on which the program can be executed. The recommended values are the same as those listed for the target field.
*F	For COMPASS assemblies, the value of the *F special symbol; blank in all other cases.

type	A letter designating the type of program:
	
	  = Relocatable CPU program or subprogram
	
	A = Absolute CPU program or overlay
	
	P = PPU program or overlay
	
	T = System text

hardware instruction requirements	A sequence of letters, left-justified, designating optional hardware features (if any) required to execute the program:
	
	C = CMU instructions (IM, DM, CC, and CU)
	
	D = Distributive data path
	
	I = Integer multiply instruction (IXi Xj*Xk)
	
	L = ECS instructions (RE and WE)
	
	X = Central and monitor exchange jumps (XJ, MJ, MXN, and MAN)

Comments	Comments to be listed by the loader, ITEMIZE, and other programs; zero if no comments are present.

Figure D-56. PRFX Table Format (Sheet 2 of 2)

REQUEST TABLES

Each user call to the loader must be accompanied by a request table. The request table can be either generated directly by the programmer or generated through a sequence of three or more LDREQ macros. The LDREQ macro and options that can be used to generate a request table are discussed in section 4.

The general format of a request table is shown in figure D-57.

Each request table must begin with a header and end with a zero word. The header can be generated by the BEGIN option of the LDREQ macro, and the zero word can be generated by the END option of the LDREQ macro.

The internal formats of all requests that can make up a request table follow. Areas of the request table figures marked reserved or res are fields that are ignored by the loader and are usually set to zero.

The LIB (0010), MAP (0011), PRESET and PRESETA (0012), USEP (0015), USE (0016), SUBST (0017), OMIT (0020), PD (0033), and PS (0034) options have the same internal format as their corresponding options of the LDSET objective directive.

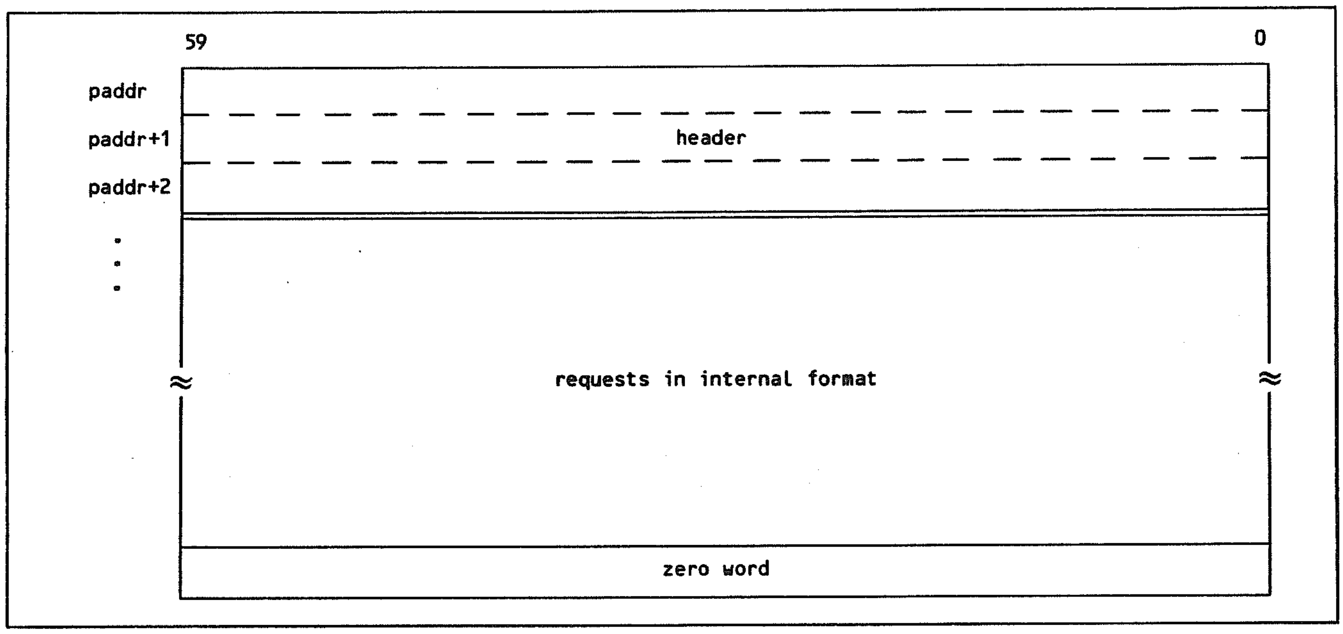


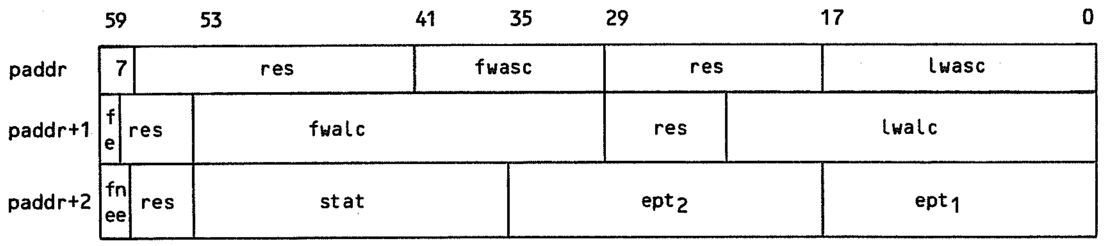
Figure D-57. Request Table Format

BEGIN (7)

BEGIN generates a three-word header, as shown in figure D-58.

The first two words contain information passed on the LDREQ BEGIN macro call. The 7 in paddr+0, bits 59 through 57, identifies the table as being for a current loader rather than the SCOPE 3.3 loader. The CYBER loader recognizes the 3.3 form and converts the call to the new form before processing. The 3.3 form cannot be used if the Common Memory Manager (CMM) parameter is present on the LOADER macro call.

The third word contains reply information returned by the loader upon completion of processing of the request sequence.



paddr	First word address of the loader request table.
res	Reserved for use by CDC.
fwasc	First word address of the central memory loadable area.
lwasc	Last word address + 1 of the central memory loadable area.
fwalc	First word address of the ECS loadable area.
lwalc	Last word address + 1 of the ECS loadable area.
paddr+2	Reply information returned by the loader upon completion of processing the request sequence. The information is as follows:

Bits	Field	Significance
59	fe	Fatal error flag; set to 1 if a fatal error occurs during loading.
58	ne	Nonfatal error flag; set to 1 if one or more nonfatal errors occur during loading.
53 through 36	stat	Status; zero if both fe and ne are zero. If fe is 1, stat contains the code of the fatal error detected. If fe is zero and ne is 1, stat contains the error code for the first nonfatal error detected. (Refer to appendix B for error codes.)
35 through 18	ept_2	Secondary entry address; next-to-last transfer symbol encountered while processing the user call.
17 through 0	ept_1	Primary entry address; last transfer symbol encountered while processing the user call.

stat	Status; zero if both fe and ne are zero.

Figure D-58. BEGIN Option Format

END

The END option (figure D-59) generates a zeroed word.

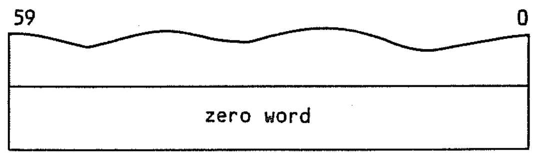
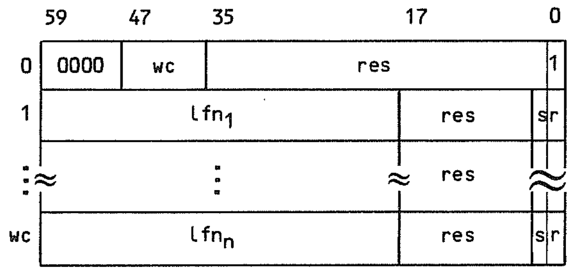


Figure D-59. END Option Format

LOAD Option (0000)

The LOAD option (figure D-60) identifies one or more files that contain object programs to be loaded.

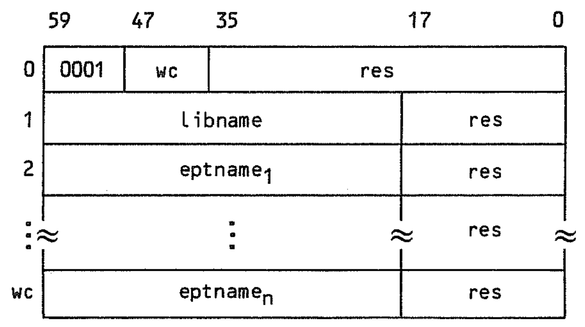


wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
lfn_i	The logical file name of the file to be loaded, left-justified with zero fill.
s	Specification of the rewind flag:
	0 = Rewind indicator absent
	1 = Rewind indicator specified
r	Rewind indicator:
	0 The file is not to be rewound
	1 The file is to be rewound

Figure D-60. LOAD Option Format

LIBLOAD Option (0001)

The LIBLOAD option (figure D-61) identifies one or more programs that are to be loaded from a particular library.



wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
libname	The name of the library, left-justified with zero fill. The library must be either a user library or system library.
$eptname_i$	The entry point name, left-justified with zero fill.

Figure D-61. LIBLOAD Option Format

SLOAD Option (0002)

The SLOAD option (figure D-62) identifies the selected object programs that are to be loaded from a local file.

CMLOAD Option (0003)

The CMLOAD option (figure D-63) identifies the first and last word addresses of an area in central memory that is to be loaded.

ECLOAD Option (0004)

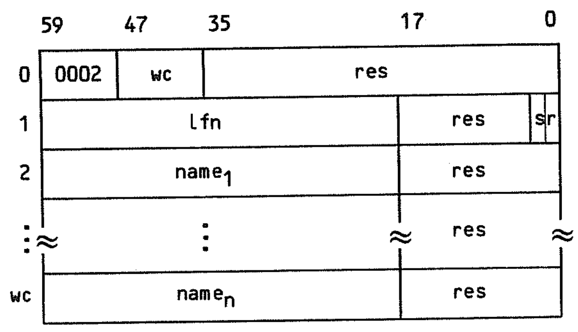
The ECLOAD option (figure D-64) identifies the first and last word addresses of an area in ECS from which the object text is to be entracted for the loading operation.

EXECUTE Option (0005)

The EXECUTE option (figure D-65) identifies the completion of the load. Execution of the loaded program begins at the specified entry point.

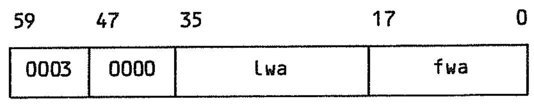
NOGO Option (0006)

The NOGO option (figure D-66) identifies completion of the load, but execution does not take place.



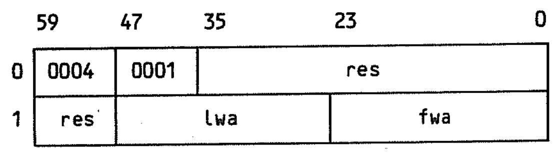
wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
lfn	The file name, left-justified with zero fill.
s	Specification of the rewind flag: 0 = Rewind indicator absent 1 = Rewind indicator specified
r	Rewind indicator: 0 The object program is not rewound 1 The object program is rewound
$name_i$	Names of the object module, left-justified with zero fill.

Figure D-62. SLOAD Option Format



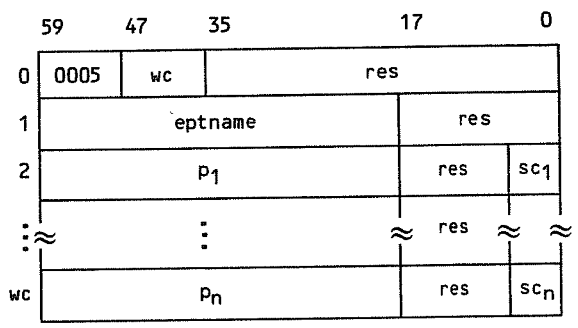
lwa	The last word address of the central memory area to be loaded.
fwa	The first word address of the central memory area to be loaded.

Figure D-63. CMLOAD Option Format



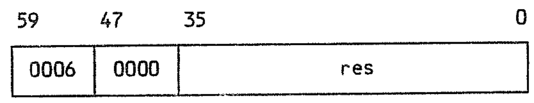
lwa	The last word address of the ECS area from which object text is to be fetched.
fwa	The first word address of the central memory area from which object text is to be fetched.

Figure D-64. ECLOAD Option Format



wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
eptname	The entry point name at which execution is to begin, left-justified with zero fill.
p_i	The execution-time parameter.
sc_i	The parameter separator. The EXECUTE macro treats all separators as commas. The values are: 01 = Comma 17 = End of list

Figure D-65. EXECUTE Option Format

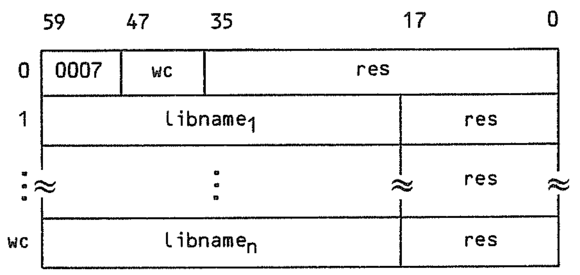


res	Reserved for use by CDC.

Figure D-66. NOGO Option Format

SATISFY Option (0007)

The SATISFY option (figure D-67) identifies one or more libraries to be searched for satisfaction of unsatisfied externals.

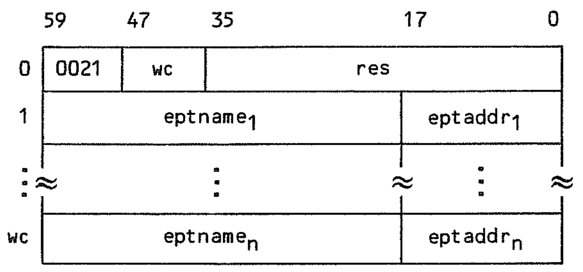


wc	The number of words in the entry not counting the control word. The number is 0 if the request is for use of the current library set.
res	Reserved for use by CDC.
libname$_i$	The library file name, left-justified with zero fill.

Figure D-67. SATISFY Option Format

ENTRY Option (0021)

The ENTRY option (figure D-68) identifies the addresses of entry points (that are currently being loaded or that have been loaded previously) to an executing program.

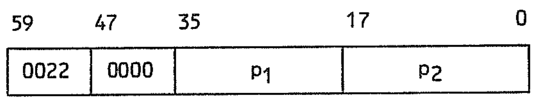


wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
eptname$_i$	The entry point name, left-justified with zero fill.
eptaddr$_i$	Upon completion of request processing, the word is cleared and the entry point address is placed in the least significant bits of the word. If the entry point cannot be found, the word is unmodified. The entry points must be in alphabetical order.

Figure D-68. ENTRY Option Format

DMP Option (0022)

The DMP option (figure D-69) specifies a user request for a dump. When the loader encounters a dump request it issues an RA+1 call to the operating system.



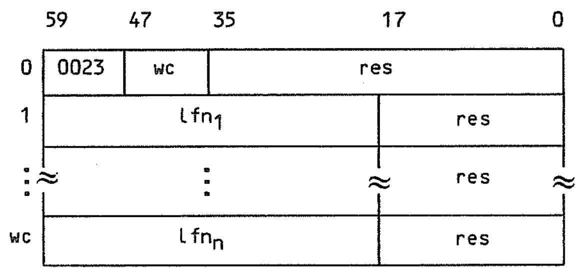
p$_1$	The first dump parameter if two parameters are present; zero if only one parameter is present.
p$_2$	The second parameter if two parameters are present; otherwise, the only parameter.

Refer to the appropriate operating system reference manual for a description of dump parameters.

Figure D-69. DMP Option Format

FILES Option (0023)

The FILES option (figure D-70) ensures that certain library programs, needed by CYBER Record Manager for processing, are loaded.

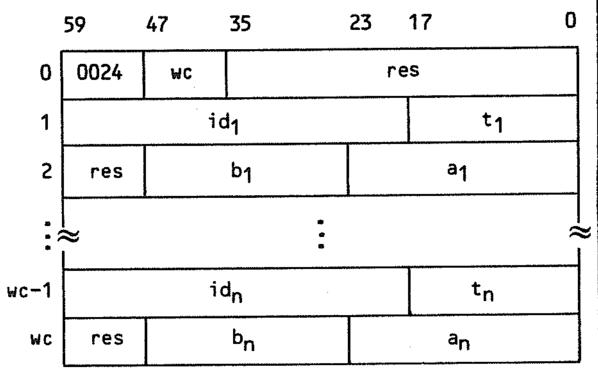


wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
lfn$_i$	The logical file name, left-justified with zero fill.

Figure D-70. FILES Option Format

PASSLOC Option (0024)

The PASSLOC option (figure D-71) identifies addresses supplied by an executing program.



wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
id_i	The name of program block, entry point, or common block.
t_i	Type of name:
b_i	The block length in words; ignored if t_i is 0.
a_i	The address of the entry point, or first word address of the program or common block.

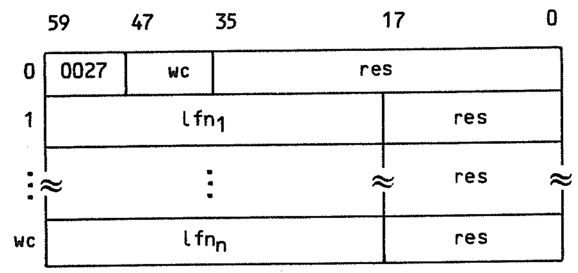
Type of name (t_i):

0	Entry point	A null id_i is illegal
1	Program block	A null id_i is illegal
2	Central memory common block	A null id_i designates a blank common block
3	ECS common block	A null id_i designates a blank common block

Figure D-71. PASSLOC Option Format

STAT Option (0027)

The STAT option (figure D-72) performs the same function as the FILES option.



wc	The number of words in the entry not counting the control word.
res	Reserved for use by CDC.
lfn_i	The logical file name, left-justified with zero fill.

Figure D-72. STAT Option Format

LOAD MAPS

Load maps are illustrated in figures E-1 through E-6. The content of the map depends on the map options as follows:

Map Contents	Map Option
Items 1 through 6, 15 through 21	S
Items 1 through 10, 19 through 24	B
Items 1 through 6, 11	E
Items 1 through 6, 11 through 14, 25	X

In addition, items 26 and 27 appear on listings generated by the TRAP directive processor; items 28 through 35 appear on the execution-time output generated by TRAPPER; and items 36 through 39 appear on capsule generation load maps.

1. Header containing the name of the first program encountered in the load except for segmented loads.

2. Loader update level.

3. fwa and lwa+1 of the area into which text was loaded. For user call loads, lwa+1 is always less than the lwa of the loadable area.

4. Name of the file containing absolute memory image of the load. This line appears if the NOGO (lfn) option was used or if this is a segment load.

5. Name and address where execution begins. If, during a user call load, no name is present, the address is that of the return to the program making the user call.

   List of entry points in 5400 table header.

6. List of all fatal or nonfatal errors encountered. Fatal errors are prefixed by FEnnnn***; nonfatal errors are prefixed NEnnnn///.

7. Labeled common block information. This includes block name, address, and length. If the block is in ECS, the address and length are of the format 40000000_8+v, where v is the actual value.

8. Program block information. This includes name, address, length, file from which obtained, and contents of words 2 through 16_8 of the PRFX (77_8) table, if present.

9. Library indicator: SL, system library; UL, user library.

10. Blank common information.

11. List of entry points and addresses in ascending order of address. Entry points are indicated by an E. The address list for unsatisfied externals contains *WEAK* for weak externals, and *UNSAT* for all other unsatisfied externals.

12. Name of the program containing entry points at left. The program name appears only once for each program.

13. Name of the program containing one or more references to the entry point at left.

14. Absolute addresses of references in the program specified at left. Up to seven references can appear on a line.

15. Total central processor seconds used for the load.

16. Maximum amount of memory required to perform the loading operation.

17. Number of times the managed tables had to be rearranged. This is primarily of interest for performance considerations.

18. Last OVERLAY directive encountered.

19. Image of the OVERLAY directive that starts a new overlay.

20. SEGLOAD directives.

21. Diagram of the tree structure as defined by directives. The symbol * indicates a segment that is the base of a tree or its only member. The symbol _ indicates that a segment branches from the segment at the top of the column from which the symbol ? extends. The symbols _ and ? appear when an ASCII graphic character set is used to print the tree diagram. The symbol ↱ corresponds to the symbol _ and the symbol ↓ corresponds to the symbol ? when a CDC graphic character set is used to print the tree diagram.

22. The horizontal line results from the LEVEL directive partitioning the user's field length. It indicates a new level from which additional trees spring.

23. Segment name.

24. The segment name delimited by parentheses defines a block used by SEGRES to control loading of segments called by this segment.

25. Name of the labeled common block in central memory. If the block was declared on a GLOBAL directive, a status indicator follows the name; otherwise, no status indicator follows the name. The status indicators are:

    G (global) = This block is declared global within this segment

    GS (global saved) = This block is declared global SAVE within this segment

    S (safe) = This global block is assigned to a segment that is always loaded when this segment is loaded

    B (bad) = This global block is assigned to a segment that is incompatible with this segment

    D (doubtful) = This global block is assigned to a segment that might or might not be loaded when this segment is loaded

26. Capsule name.

27. Group name.

28. Length of capsule binary, including supplemental tables written at the end (containing relocation, entry, and external reference information).

29. Indication of capsule entry points (E) and externals (X).

30. Indication of LOCAL SAVE block for the named program.

```
①LOAD MAP - FOREST                                 CYBER LOADER 1.5-650      85/11/11. 13.19.46.      PAGE     1
                                                                ②

③FWA OF THE LOAD           111
 LWA+1 OF THE LOAD       13375

⑤TRANSFER ADDRESS -- FOREST           2057

 PROGRAM ENTRY POINTS --       FOREST          2067


********* ERROR SUMMARY ⑥

    NE4100///  UNSATISFIED EXTERNAL REF -- GRASS


 PROGRAM AND BLOCK ASSIGNMENTS.

⑦BLOCK        ADDRESS  LENGTH   FILE     DATE     PROCSSR VER LEVEL  HARDWARE  COMMENTS

 /GROVE/          111      1
 /PINE/           112    144
 /MAPLE/          256    310
 /HICKORY/        566    454
 /OAK/           1242    620
⑧FOREST          2062     43    LGO    85/11/11 FTN     5.1 650    666X I      PROGRAMOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
 WOODS   -LS  ㉚2125      1
 WOODS           2126     11    LGO    85/11/11 FTN     5.1 650    666X I    SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
 NEEDLES         2137     22    LGO    85/11/11 FTN     5.1 650    666X I    SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
 LEAVES          2161     21    LGO    85/11/11 FTN     5.1 650    666X I    SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
 BRANCH          2202     21    LGO    85/11/11 FTN     5.1 650    666X I    SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
 TRUNK           2223     21    LGO    85/11/11 FTN     5.1 650    666X I    SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
 SYSAID=         2244      1 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              LINK BETWEEN SYS=AID AND INITIALIZATION CODE.
 /FCL.C./        2245     36
 /STP.END/       2303      1
 /Q5.IO./        2304    373         ⑨
 Q5NTRY=         2677    252 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              FCL5 - INITIALIZE FCL5 RUN TIME LIBRARY.
 CMMOVE=         3151    171 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              CHARACTER MOVE AND CONCATENATE
 /FCL=ENT/       3342     75
 COMIO=          3437     42 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              COMMON CODED I/O ROUTINES AND CONSTANTS.
 FCL=FDL         3501     63 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              FCL CAPSULE LOADING
 FECMSK=         3564     41 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              INITIALIZE CONSTANTS.
 FEIFST=         3625      3 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              CONVERTED DATA STORAGE.
 FLTOUT=         3630    323 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              COMMON FLOATING OUTPUT CODE
 /AP.IO./        4153     17
 FORSYS=         4172   2023 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              FORTRAN OBJECT LIBRARY UTILITIES.
 FORUTL=         6215    200 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              FCL MISC. UTILITIES.
 GETFIT=         6415    206 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              GETFIT= - LOCATE A FIT GIVEN A UNIT DESCRIPTOR
 LDOUT=          6623    315 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              LIST DIRECTED OUTPUT FORMATTING
 OUTCOM=         7140    161 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              COMMON OUTPUT CODE
 OUTF=           7321    252 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              LIST DIRECTED OUTPUT CONTROL
 Q5RPV=          7573     14 SL-FTN5LIB 85/09/23 COMPASS 3.6 650              FCL5 - ABORT RECOVERY INITIALIZATION
 CPU.CIO         7607     16 SL-SYSLIB  85/09/23 COMPASS 3.6 642              I/O FUNCTION PROCESSOR.
 CPU.MVE         7625     64 SL-SYSLIB  85/09/23 COMPASS 3.6 642              MOVE BLOCK OF DATA.
 CPU.SYS         7711     40 SL-SYSLIB  85/09/23 COMPASS 3.6 642              PROCESS SYSTEM REQUEST.
 CMF.ALF         7751    175 SL-SYSLIB  85/09/23 COMPASS 3.6 650              CMM V1.1 - ALLOCATE FIXED.
 CMF.CSF        10146      6 SL-SYSLIB  85/09/23 COMPASS 3.6 650              CMM V1.1 - CHANGE SPECS FIXED.
```

Figure E-1. Full Load Map (Sheet 1 of 4)

```
LOAD MAP - FOREST                                   CYBER LOADER 1.5-650      85/11/11. 13.19.46.      PAGE     2

BLOCK         ADDRESS   LENGTH   FILE       DATE     PROCSSR VER LEVEL   HARDWARE   COMMENTS

CMM.FFA         10154       14 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CMM V1.1 - FIXED FREE ALGORITHM.
CMF.FRF         10170       36 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CMM V1.1 - FREE FIXED.
CMF.GSS         10226       22 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CMM V1.1 - GET SUMMARY STATISTICS.
CMM.MEM         10250        7 SL-SYSLIB   85/09/23 COMPASS 3.6 650
CMM.R           10257      212 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CMM V1.1 - RESIDENT SUBROUTINES.
CMF.SLF         10471       22 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CMM V1.1 - SHRINK AT LWA FIXED.
/FDL.COM/       10513       23
FDL.RES         10536      212 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 FAST DYNAMIC LOADER RESIDENT.
FDL.MMI         10750      402 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 FDL MEMORY MANAGER INTERFACE.
CTL$RM          11352      501 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CRM CONTROLLING ROUTINE.
CTL$SKP         12053       57 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CRM CONTROLLER - SKIP PHYSICAL/FILE.
CTL$WR          12132       47 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CRM CONTROLLER - WEOX, REWIND
ERR$RM          12201       25 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CRM ERROR PROCESSOR ENTRY.
LIST$RM         12226       67 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CRM - ALLOCATE SPACE FOR LIST OF FILES
PM$SYS=         12315        5 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 CRM - POST RA+1 REQUEST
RECOVR          12322      362 SL-SYSLIB   85/09/23 COMPASS 3.6 650                 RECOVR - V2.0, USER INTERFACE TO *RPV*.
CPU.CPM         12704        5 SL-SYSLIB   85/08/16 COMPASS 3.6 642                 82/02/26. 85/07/29. CPUREL - CONTROL POINT MANA
CPU.LFM         12711       10 SL-SYSLIB   85/08/16 COMPASS 3.6 642                 82/02/26. 85/07/29. CPUREL - LOCAL FILE MANAGER
⑩ //            12721      454


ENTRY POINTS.

⑪ENTRY      ADDRESS   PROGRAM             REFERENCES

AAM$RL       *WEAK*                     ⑬CTL$RM       ⑭11441
AAM$GJ       *WEAK*                       CTL$RM         11442      11673
AAM$LDK      *WEAK*                       CTL$RM         11443
AAM$PBC      *WEAK*                       CTL$RM         11444
CMM.GOA      *WEAK*                       CMF.ALF         7766       7776
LABL$RM      *WEAK*                       CTL$RM         11400
ABTPMD.      *WEAK*                       FORSYS=         4655
PM$PARD      *WEAK*                       CTL$RM         12004
CLSV$SQ      *WEAK*                       CTL$RM         11363
BACKSR.      *WEAK*                       FORSYS=         6124
OPEN$        *WEAK*                       FCL=ENT         3407
CMM.AGR      *WEAK*                       CTL$RM         11633      11636
CMM.VAF      *WEAK*                       CMF.ALF        10144
CMM.VFA      *WEAK*                       CMM.FFA        10162      10163
LMC.         *WEAK*                       FCL=ENT         3403
CHEK$SQ      *WEAK*                       CTL$RM         11360
RAPLUS1      *WEAK*                       FORSYS=         4676
CTRL$AA      *WEAK*                       CTL$RM         11370
SYS2$        *WEAK*                       FCL=ENT         3422
CLOS$RI      *WEAK*                       FCL=ENT         3350
CLSF$RM      *WEAK*                       CTL$RM         11362
RPVCOD$      *WEAK*                       FCL=ENT         3412
PM$PARI      *WEAK*                       CTL$RM         11776
CMMX$AA      *WEAK*                       CTL$RM         11445
Q2NTPY.      *WEAK*                       GETFIT=         6431
REW$WA       *WEAK*                       CTL$RM         11413
OPEN.        *WEAK*                       FCL=ENT         3410
GET$SQ       *WEAK*                       CTL$RM         11375
GET$WA       *WEAK*                       CTL$RM         11376
```

Figure E-1. Full Load Map (Sheet 2 of 4)

```
LOAD MAP - FOREST                                          CYBER LOADER 1.5-650      85/11/11. 13.19.46.      PAGE      3

ENTRY      ADDRESS   PROGRAM              REFERENCES

GPTM$SQ      *WEAK*                       CTL$RM        11377
FECPPT$      *WEAK*                       FCL=ENT        3371
INQRET.      *WEAK*                       FCL=ENT        3376
FEIERR$      *WEAK*                       FCL=ENT        3373
IOERR$       *WEAK*                       FCL=ENT        3400
PUT$SQ       *WEAK*                       CTL$RM        11407
PUT$WA       *WEAK*                       CTL$RM        11410
TRACEB$      *WEAK*                       FCL=ENT        3425
REW$SQ       *WEAK*                       CTL$RM        11412
ECA.         *WEAK*                       FCL=ENT        3356
FECEIE$      *WEAK*                       FCL=ENT        3362
PMP$LR       *WEAK*                       CTL$RM        11750
GET$S        *WEAK*                       CTL$RM        11374
Q5PMD.       *WEAK*                       FORSYS=        4675
SYSERR$      *WEAK*                       FCL=ENT        3417
COMM$WA      *WEAK*                       CTL$RM        11366
GRASS       *UNSAT*                       FOREST         2107
SKFL$SQ      *WEAK*                       CTL$RM        11432
INQUIR$      *WEAK*                       FCL=ENT        3377
SKIP$SQ      *WEAK*                       CTL$RM        11433
DF$CPM       *WEAK*                       CTL$RM        11371
SKSF$SQ      *WEAK*                       CTL$RM        11435
PUT$S        *WEAK*                       CTL$RM        11406
REPL$SQ      *WEAK*                       CTL$RM        11411
SKBL$SQ      *WEAK*                       CTL$RM        11431
FLEF$RM      *WEAK*                       CTL$RM        11373
FECESE$      *WEAK*                       FCL=ENT        3364
SKSB$SQ      *WEAK*                       CTL$RM        11434
WEOX$SQ      *WEAK*                       CTL$RM        11437
OPEN$RM      *WEAK*                       CTL$RM        11405
FOREST      2067     FOREST ⑫
WOODS       2133     WOODS                FOREST         2073
NEEDLES     2144     NEEDLES              FOREST         2075
LEAVES      2166     LEAVES               FOREST         2077
BRANCH      2207     BRANCH               FOREST         2101
TRUNK       2230     TRUNK                FOREST         2103
SYSAID=     2244     SYSAID=              Q5.IO.         2653
FCL.C.      2245     FORUTL=              FCL=ENT        3360
                                          COMIO=         3500

RANDOM.     2251
D10.        2253
RBF.        2254
UCF.        2255
LOF.FTN     2257
USRKEY.     2260
LINLIM.     2261
ERRLOC.     2301
STP.END     2303     Q5NTRY=              FCL=ENT        3415
Q5.IO.      2304     FORSYS=              FCL=ENT        3411
DXB.        2314     Q5NTRY=
Q5N100.     2455
Q5NTRY.     2654                          Q5RPV=         7600
STA.ALF     2702                          GETFIT=        6540
MMC.        3157     CMMOVE=              FCL=ENT        3405
.                                         COMIO=         3466
.
.
```

Figure E-1. Full Load Map (Sheet 3 of 4)

```
LOAD MAP - FOREST                                  CYBER LOADER 1.5-650       85/11/11. 13.19.46.      PAGE      7

ENTRY      ADDRESS   PROGRAM              REFERENCES

                                          CTL$WR       12136     12163     12164
                                          ERR$RM       12205
RMP$FE       11373
RM$FLB       11440
CMMX.AA      11445
DFCT$RM      11455
RM$CIO       11457                        CTL$WR       12177
RM$RCLA      11471                        CTL$WR       12174
CHWR$RM      11477
MOVE$RM      11505
RM$ABJF      11625                        CTL$SKP      12125
RM$PD        11657
RM$FDC       11663                        CTL$SKP      12064
                                          CTL$WR       12152
RM$FAT       11667                        CTL$WR       12134
RM$LGO       11670                        CTL$SKP      12061     12111     12113     12115     12127
                                          CTL$WR       12137     12163     12165
RM$BID       11676                        ERR$RM       12206
RM$LDC       11703                        FORSYS=       4637
RM$UTC       11732
RM$TKP       11741
RM$LVL       11742
RM$ULJ       11743
CLOF$RM      11752                        FCL=FDL       3502
                                          FORSYS=       4573      5063      5165      5753
OPNM$RM      11755                        FORSYS=       4543      5055      6114
SFIT$RM      11760                        Q5.IO.        2563
                                          FCL=ENT       3414
                                          FORSYS=       6014
GET$RM       11762
RM$P4G       11772
RM$G1        11777
PUT$RM       12001                        FCL=FDL       3504
                                          FORSYS=       4565      5157
                                          OUTF=         7551      7555
RM$EREX      12015
RM$PAR4      12016
RM$P1        12042


SKBF$RM      12053   CTL$SKP
SKBP$RM      12054
SKFF$RM      12055
SKFP$RM      12056
SKFL$RM      12100                        FORSYS=       5121
SKBL$RM      12116
REW$RM       12132   CTL$WR               FORSYS=       5211
.
.
.

      ⑮ .155 CP SECONDS           ⑯ 32300B CM STORAGE USED             ⑰ 15 TABLE MOVES
```

Figure E-1. Full Load Map (Sheet 4 of 4)

```
(1) LOAD MAP - FOREST                              CYBER LOADER 1.5-650       85/11/08. 16.36.14.       PAGE      1
                                                        (2)

(3) FWA OF THE LOAD           111
    LWA+1 OF THE LOAD       13375

(5) TRANSFER ADDRESS -- FOREST          2067

    PROGRAM ENTRY POINTS --       FOREST          2067


********* ERROR SUMMARY

(6) NE4100///  UNSATISFIED EXTERNAL REF -- GRASS


    PROGRAM AND BLOCK ASSIGNMENTS.

(7) BLOCK          ADDRESS    LENGTH     FILE      DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

    /GROVE/           111         1
    /PINE/            112       144
    /MAPLE/           256       310
    /HICKORY/         566       454
    /OAK/            1242       620
(8) FOREST           2062        43     LGO      85/11/08 FTN      5.1 650    666X I        PROGRAMOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
    WOODS   -LS (30) 2125         1
    WOODS            2126        11     LGO      85/11/08 FTN      5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
    NEEDLES          2137        22     LGO      85/11/08 FTN      5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
    LEAVES           2161        21     LGO      85/11/08 FTN      5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
    BRANCH           2202        21     LGO      85/11/08 FTN      5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
    TRUNK            2223        21     LGO      85/11/08 FTN      5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
    SYSAID=          2244         1 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               LINK BETWEEN SYS=AID AND INITIALIZATION CODE.
    /FCL.C./         2245        36
    /STP.END/        2303         1
    /Q5.IO./         2304       373         (9)
    Q5NTRY=          2677       252 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               FCL5 - INITIALIZE FCL5 RUN TIME LIBRARY.
    CHMOVE=          3151       171 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               CHARACTER MOVE AND CONCATENATE
    /FCL=ENT/        3342        75
    COMIO=           3437        42 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               COMMON CODED I/O ROUTINES AND CONSTANTS.
    FCL=FDL          3501        63 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               FCL CAPSULE LOADING
    FECMSK=          3564        41 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               INITIALIZE CONSTANTS.
    FEIFST=          3625         3 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               CONVERTED DATA STORAGE.
    FLTOUT=          3630       323 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               COMMON FLOATING OUTPUT CODE
    /AP.IO./         4153        17
    FORSYS=          4172      2023 SL-FTN5LIB   85/09/23 COMPASS  3.6 650 .             FORTRAN OBJECT LIBRARY UTILITIES.
    FORUTL=          6215       200 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               FCL MISC. UTILITIES.
    GETFIT=          6415       206 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               GETFIT= - LOCATE A FIT GIVEN A UNIT DESCRIPTOR
    LDOUT=           6623       315 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               LIST DIRECTED OUTPUT FORMATTING
    OUTCOM=          7140       161 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               COMMON OUTPUT CODE
    OUTF=            7321       252 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               LIST DIRECTED OUTPUT CONTROL
    Q5RPV=           7573        14 SL-FTN5LIB   85/09/23 COMPASS  3.6 650               FCL5 - ABORT RECOVERY INITIALIZATION
    CPU.CIO          7607        15 SL-SYSLIB    85/09/23 COMPASS  3.6 642               I/O FUNCTION PROCESSOR.
    CPU.MVE          7625        64 SL-SYSLIB    85/09/23 COMPASS  3.6 642               MOVE BLOCK OF DATA.
    CPU.SYS          7711        40 SL-SYSLIB    85/09/23 COMPASS  3.6 642               PROCESS SYSTEM REQUEST.
    CMF.ALF          7751       175 SL-SYSLIB    85/09/23 COMPASS  3.6 650               CMM V1.1 - ALLOCATE FIXED.
    CMF.CSF         10146         6 SL-SYSLIB    85/09/23 COMPASS  3.6 650               CMM V1.1 - CHANGE SPECS FIXED.
```

Figure E-2. Partial Load Map (Sheet 1 of 2)

```
LOAD MAP - FOREST                                    CYBER LOADER 1.5-650     85/11/08. 15.36.14.     PAGE    2

BLOCK        ADDRESS   LENGTH    FILE       DATE     PROCSSR VER LEVEL  HARDWARE   COMMENTS

CMM.FFA       10154      14 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CMM V1.1 - FIXED FREE ALGORITHM.
CMF.FRF       10170      35 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CMM V1.1 - FREE FIXED.
CMF.GSS       10226      22 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CMM V1.1 - GET SUMMARY STATISTICS.
CMM.MEM       10250       7 SL-SYSLIB  85/09/23 COMPASS 3.6 650
CMM.R         10257     212 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CMM V1.1 - RESIDENT SUBROUTINES.
CMF.SLF       10471      22 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CMM V1.1 - SHRINK AT LWA FIXED.
/FDL.COM/     10513      23
FDL.RES       10536     212 SL-SYSLIB  85/09/23 COMPASS 3.6 650                FAST DYNAMIC LOADER RESIDENT.
FDL.MMI       10750     402 SL-SYSLIB  85/09/23 COMPASS 3.6 650                FDL MEMORY MANAGER INTERFACE.
CTL$RM        11352     501 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CRM CONTROLLING ROUTINE.
CTL$SKP       12053      57 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CRM CONTROLLER - SKIP PHYSICAL/FILE.
CTL$WR        12132      47 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CRM CONTROLLER - WEOX, REWIND
ERR$RM        12201      25 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CRM ERROR PROCESSOR ENTRY.
LIST$RM       12226      67 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CRM - ALLOCATE SPACE FOR LIST OF FILES
RM$SYS=       12315       5 SL-SYSLIB  85/09/23 COMPASS 3.6 650                CRM - POST RA+1 REQUEST
RECOVR        12322     362 SL-SYSLIB  85/09/23 COMPASS 3.6 650                RECOVR - V2.0, USER INTERFACE TO *RPV*.
CPU.CPM       12704       5 SL-SYSLIB  85/08/16 COMPASS 3.6 642                82/02/26. 85/07/29. CPUREL - CONTROL POINT MANA
CPU.LFM       12711      10 SL-SYSLIB  85/08/16 COMPASS 3.6 642                82/02/26. 85/07/29. CPUREL - LOCAL FILE MANAGER
(10) //       12721     454


   (15) .116 CP SECONDS      (16) 31200B CM STORAGE USED          (17) 9 TABLE MOVES
```

Figure E-2. Partial Load Map (Sheet 2 of 2)

```
① LOAD MAP - FOREST                                   CYBER LOADER 1.5-650       85/11/11. 15.31.35.       PAGE     1
⑱ OVERLAY(LAND,0,0,OV=5)                                          ②

  ⑲
-------- OVERLAY(LAND,0,0,OV=5)

③ FWA OF THE LOAD              123
  LWA+1 OF THE LOAD          14367

⑤ TRANSFER ADDRESS -- FOREST          2101

  PROGRAM ENTRY POINTS --       FOREST          2101


  PROGRAM AND BLOCK ASSIGNMENTS.

  BLOCK        ADDRESS   LENGTH    FILE      DATE     PROCSSR VER LEVEL  HARDWARE   COMMENTS

  /GROVE/         123        1
⑦ /PINE/          124      144
  /MAPLE/         270      310
  /HICKORY/       600      454
  /OAK/          1254      620
⑧ FOREST         2074       72    LGO     85/11/11 FTN     5.1 650    666X I       PROGRAMOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
  SYSAID=        2166        1 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               LINK BETWEEN SYS=AID AND INITIALIZATION CODE.
  /FCL.C./       2167       36    ⑨
  /STP.END/      2225        1
  /Q5.IO./       2226      373
  Q5NTRY=        2621      252 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               FCL5 - INITIALIZE FCL5 RUN TIME LIBRARY.
  CMMOVE=        3073      171 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               CHARACTER MOVE AND CONCATENATE
  /FCL=ENT/      3264       75
  COMIO=         3361       42 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               COMMON CODED I/O ROUTINES AND CONSTANTS.
  FCL=FCL        3423       63 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               FCL CAPSULE LOADING
  FECMSK=        3506       41 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               INITIALIZE CONSTANTS.
  FEIFST=        3547        3 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               CONVERTED DATA STORAGE.
  FLTOUT=        3552      323 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               COMMON FLOATING OUTPUT CODE
  /AP.IO./       4075       17
  FORSYS=        4114     2023 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               FORTRAN OBJECT LIBRARY UTILITIES.
  FORUTL=        6137      200 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               FCL MISC. UTILITIES.
  GETFIT=        6337      206 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               GETFIT= - LOCATE A FIT GIVEN A UNIT DESCRIPTOR
  LDOUT=         6545      315 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               LIST DIRECTED OUTPUT FORMATTING
  OUTCOM=        7062      161 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               COMMON OUTPUT CODE
  OUTF=          7243      252 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               LIST DIRECTED OUTPUT CONTROL
  OVERLAY        7515      245 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               OVERLAY LOADING ROUTINE.
  Q5RPV=         7762       14 SL-FTN5LIB 85/09/23 COMPASS 3.6 650               FCL5 - ABORT RECOVERY INITIALIZATION
  CPU.CIO        7776       16 SL-SYSLIB  85/09/23 COMPASS 3.6 642               I/O FUNCTION PROCESSOR.
  CPU.MVE       10014       64 SL-SYSLIB  85/09/23 COMPASS 3.6 642               MOVE BLOCK OF DATA.
  CPU.SYS       10100       40 SL-SYSLIB  85/09/23 COMPASS 3.6 642               PROCESS SYSTEM REQUEST.
  CMF.ALF       10140      175 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - ALLOCATE FIXED.
  CMM.CIA       10335      105 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - CHANGE INTERNAL AREA.
  CMF.CSF       10442        6 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - CHANGE SPECS FIXED.
  CMM.FFA       10450       14 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - FIXED FREE ALGORITHM.
  CMF.FRF       10464       36 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - FREE FIXED.
  CMF.GSS       10522       22 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - GET SUMMARY STATISTICS.
  CMF.LDV       10544      274 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - LOAD OVERLAY.
  CMF.LCV       11040       55 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - LOAD OVERLAY VIA FOL.
  CMM.MEM       11115        7 SL-SYSLIB  85/09/23 COMPASS 3.6 650
  CMM.R         11124      212 SL-SYSLIB  85/09/23 COMPASS 3.6 650               CMM V1.1 - RESIDENT SUBROUTINES.
```

Figure E-3. Partial Load Map of Overlay Generation (Sheet 1 of 4)

```
     LOAD MAP - FOREST                                          CYBER LOADER 1.5-650       85/11/11. 15.31.35.       PAGE     2
     OVERLAY(LAND,0,0,OV=5)

     BLOCK          ADDRESS    LENGTH     FILE         DATE        PROCSSR VER LEVEL   HARDWARE    COMMENTS

     CMF.SLF          11336        22 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       CMM V1.1 - SHRINK AT LWA FIXED.
     /FDL.COM/        11360        23
     FDL.RES          11403       212 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       FAST DYNAMIC LOADER RESIDENT.
     FDL.MMI          11615       402 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       FDL MEMORY MANAGER INTERFACE.
     FOL.RES          12217       125 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       FAST OVERLAY LOADER RESIDENT.
     CTL$RM           12344       501 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       CRM CONTROLLING ROUTINE.
     CTL$SKP          13045        57 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       CRM CONTROLLER - SKIP PHYSICAL/FILE.
     CTL$WR           13124        47 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       CRM CONTROLLER - WEOX, REWIND
     ERR$RM           13173        25 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       CRM ERROR PROCESSOR ENTRY.
     LIST$RM          13220        67 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       CRM - ALLOCATE SPACE FOR LIST OF FILES
     RM$SYS=          13307         5 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       CRM - POST RA+1 REQUEST
     RECOVR           13314       362 SL-SYSLIB     85/09/23 COMPASS 3.6 650                       RECOVR - V2.0, USER INTERFACE TO *RPV*.
     CPU.CPM          13676         5 SL-SYSLIB     85/08/16 COMPASS 3.6 642                       82/02/26. 85/07/29. CPUREL - CONTROL POINT MANA
     CPU.LFM          13703        10 SL-SYSLIB     85/08/16 COMPASS 3.6 642                       82/02/26. 85/07/29. CPUREL - LOCAL FILE MANAGER
 (10) //              13713       454



(19)-------- OVERLAY(LAND,1,0)

     FWA OF THE LOAD            14374
     LWA+1 OF THE LOAD          14421

     TRANSFER ADDRESS -- WOODS            14401


     PROGRAM AND BLOCK ASSIGNMENTS.

     BLOCK          ADDRESS    LENGTH     FILE         DATE        PROCSSR VER LEVEL   HARDWARE    COMMENTS

     WOODS            14374        25     LGO       85/11/11 FTN      5.1 650      666X I         PROGRAMOPT=0,ROUND= A/ S/ M/-D       ARG=UNS



(19)-------- OVERLAY(LAND,2,0)

     FWA OF THE LOAD            14374
     LWA+1 OF THE LOAD          14421

     TRANSFER ADDRESS -- NEEDLES          14401


     PROGRAM AND BLOCK ASSIGNMENTS.

     BLOCK          ADDRESS    LENGTH     FILE         DATE        PROCSSR VER LEVEL   HARDWARE    COMMENTS

     NEEDLES          14374        25     LGO       85/11/11 FTN      5.1 650      666X I         PROGRAMOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
```

Figure E-3. Partial Load Map of Overlay Generation (Sheet 2 of 4)

```
         LOAD MAP - FOREST                                      CYBER LOADER 1.5-650     85/11/11. 15.31.35.     PAGE     3
         OVERLAY(LAND,2,1)



(19)---- OVERLAY(LAND,2,1)

         FWA OF THE LOAD          14426
         LWA+1 OF THE LOAD        14453

         TRANSFER ADDRESS -- LEAVES        14433


         PROGRAM AND BLOCK ASSIGNMENTS.

         BLOCK        ADDRESS    LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE  COMMENTS

         LEAVES         14426        25    LGO      85/11/11 FTN      5.1 650    666X I       PROGRAMOPT=0,ROUND= A/ S/ M/-D       ARG=UNS


(19)---- OVERLAY(LAND,3,0)

         FWA OF THE LOAD          14374
         LWA+1 OF THE LOAD        14421

         TRANSFER ADDRESS -- BRANCH        14401


         PROGRAM AND BLOCK ASSIGNMENTS.

         BLOCK        ADDRESS    LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE  COMMENTS

         BRANCH         14374        25    LGO      85/11/11 FTN      5.1 650    666X I       PROGRAMOPT=0,ROUND= A/ S/ M/-D       ARG=UNS


(19)---- OVERLAY(LAND,4,0)

         FWA OF THE LOAD          14374
         LWA+1 OF THE LOAD        14421

         TRANSFER ADDRESS -- TRUNK         14401


         PROGRAM AND BLOCK ASSIGNMENTS.
```

Figure E-3. Partial Load Map of Overlay Generation (Sheet 3 of 4)

```
LOAD MAP - FOREST                                  CYBER LOADER 1.5-650       85/11/11. 15.31.35.       PAGE      4
OVERLAY(LAND,4,0)

BLOCK        ADDRESS    LENGTH     FILE      DATE       PROCSSR VER LEVEL  HARDWARE   COMMENTS

TRUNK          14374        25     LGO       85/11/11 FTN      5.1 650    666X I        PROGRAMOPT=0,ROUND= A/ S/ M/-D          ARG=UNS


   ⑮ .149 CP SECONDS          ⑯ 353008 CM STORAGE USED                ⑰ 15 TABLE MOVES
```

Figure E-3. Partial Load Map of Overlay Generation (Sheet 4 of 4)

```
(1) LOAD MAP - FOREST                                          CYBER LOADER 1.5-650       85/11/11. 15.31.14.      PAGE     1
(18)OVERLAY(BIGTREE,0,0,OV=3)                                          (2)

    (19)
-------- OVERLAY(BIGTREE,0,0,OV=3)
    WRITTEN TO FILE      CAPS

(3) FWA OF THE LOAD            117
    LWA+1 OF THE LOAD        15274

(5) TRANSFER ADDRESS -- FOREST          2075

    PROGRAM ENTRY POINTS --       FOREST          2075


    PROGRAM AND BLOCK ASSIGNMENTS.

    BLOCK        ADDRESS   LENGTH     FILE      DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

    /GROVE/          117       1
(7) /PINE/           120     144
    /MAPLE/          264     310
    /HICKORY/        574     454
    /OAK/           1250     620
(8) FOREST          2070      41     LGO     85/11/11 FTN      5.1 650    666X I       PROGRAMOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
    SYSAID=         2131       1 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                LINK BETWEEN SYS=AID AND INITIALIZATION CODE.
    /FCL.C./        2132      36    (9)
    /STP.END/       2170       1
    /Q5.IC./        2171     373
    Q5NTRY=         2564     252 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                FCL5 - INITIALIZE FCL5 RUN TIME LIBRARY.
    CHMOVE=         3036     171 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                CHARACTER MOVE AND CONCATENATE
    /FCL=ENT/       3227      75
    COMIO=          3324      42 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                COMMON CODED I/O ROUTINES AND CONSTANTS.
    FCL=FDL         3366      63 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                FCL CAPSULE LOADING
    FECMSK=         3451      41 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                INITIALIZE CONSTANTS.
    FEIFST=         3512       3 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                CONVERTED DATA STORAGE.
    FLTOUT=         3515     323 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                COMMON FLOATING OUTPUT CODE
    /AP.IO./        4040      17
    FMTAP=          4057     646 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                CRACK APLIST AND FORMAT FOR KODER/KRAKER.
    FORSYS=         4725    2023 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                FORTRAN OBJECT LIBRARY UTILITIES.
    FORUTL=         6750     200 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                FCL MISC. UTILITIES.
    GETFIT=         7150     206 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                GETFIT= - LOCATE A FIT GIVEN A UNIT DESCRIPTOR
    KODER=          7356     702 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                OUTPUT FORMAT INTERPRETER.
    LDOUT=         10260     315 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                LIST DIRECTED OUTPUT FORMATTING
    OUTC=          10575     242 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                FORMATTED WRITE FORTRAN RECORD.
    OUTCOM=        11037     161 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                COMMON OUTPUT CODE
    OUTF=          11220     252 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                LIST DIRECTED OUTPUT CONTROL
    Q5RPV=         11472      14 SL-FTN5LIB  85/09/23 COMPASS  3.6 650                FCL5 - ABORT RECOVERY INITIALIZATION
    CPU.CIO        11506      16 SL-SYSLIB   85/09/23 COMPASS  3.6 642                I/O FUNCTION PROCESSOR.
    CPU.MVE        11524      64 SL-SYSLIB   85/09/23 COMPASS  3.6 642                MOVE BLOCK OF DATA.
    CPU.SYS        11610      40 SL-SYSLIB   85/09/23 COMPASS  3.6 642                PROCESS SYSTEM REQUEST.
    CMF.ALF        11650     175 SL-SYSLIB   85/09/23 COMPASS  3.6 650                CMM V1.1 - ALLOCATE FIXED.
    CMF.CSF        12045       6 SL-SYSLIB   85/09/23 COMPASS  3.6 650                CMM V1.1 - CHANGE SPECS FIXED.
    CMM.FFA        12053      14 SL-SYSLIB   85/09/23 COMPASS  3.6 650                CMM V1.1 - FIXED FREE ALGORITHM.
    CMF.FRF        12067      36 SL-SYSLIB   85/09/23 COMPASS  3.6 650                CMM V1.1 - FREE FIXED.
    CMF.GSS        12125      22 SL-SYSLIB   85/09/23 COMPASS  3.6 650                CMM V1.1 - GET SUMMARY STATISTICS.
    CMM.MEM        12147       7 SL-SYSLIB   85/09/23 COMPASS  3.6 650
    CMM.R          12156     212 SL-SYSLIB   85/09/23 COMPASS  3.6 650                CMM V1.1 - RESIDENT SUBROUTINES.
```

Figure E-4. Full Load Map of Overlay and OVCAP Generation (Sheet 1 of 5)

```
LOAD MAP - FOREST                                          CYBER LOADER 1.5-650      85/11/11. 15.31.14.     PAGE     2
OVERLAY(BIGTREE,0,0,OV=3)

BLOCK        ADDRESS    LENGTH    FILE       DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

CMF.SLF        12370        22 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  CMM V1.1 - SHRINK AT LWA FIXED.
/FDL.COM/      12412        23
FDL.RES        12435       212 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  FAST DYNAMIC LOADER RESIDENT.
FDL.MMI        12647       402 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  FDL MEMORY MANAGER INTERFACE.
CTL$RM         13251       501 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  CRM CONTROLLING ROUTINE.
CTL$SKP        13752        57 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  CRM CONTROLLER - SKIP PHYSICAL/FILE.
CTL$WR         14031        47 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  CRM CONTROLLER - WEOX, REWIND
ERR$RM         14100        25 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  CRM ERROR PROCESSOR ENTRY.
LIST$RM        14125        67 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  CRM - ALLOCATE SPACE FOR LIST OF FILES
RM$SYS=        14214         5 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  CRM - POST RA+1 REQUEST
RECOVR         14221       362 SL-SYSLIB   85/09/23 COMPASS 3.6 650                  RECOVR - V2.0, USER INTERFACE TO *RPV*.
CPU.CPM        14603         5 SL-SYSLIB   85/08/16 COMPASS 3.6 642                  82/02/26. 85/07/29. CPUREL - CONTROL POINT MANA
CPU.LFM        14610        10 SL-SYSLIB   85/08/16 COMPASS 3.6 642                  82/02/26. 85/07/29. CPUREL - LOCAL FILE MANAGER
⑩ //           14620       454


⑪ ENTRY POINTS.

ENTRY       ADDRESS   PROGRAM              REFERENCES

AAM$BL        *WEAK*                       CTL$RM        13340
AAM$GD        *WEAK*                       CTL$RM        13341      13572
.
.
.                                          CTL$SKP       14027      14030
LIST$RM      14125    LIST$RM              CTL$RM        13251
RM$SYS=      14216    RM$SYS=              CTL$RM        13324      13366      13374
RECOVR       14340    RECOVR
SETUP.       14554                         Q5RPV=        11475
CPM=         14604    CPU.CPM              FORSYS=        5261
                                           CTL$RM        13266
LFM=         14611    CPU.LFM              CTL$RM        13345
```

Figure E-4. Full Load Map of Overlay and OVCAP Generation (Sheet 2 of 5)

```
LOAD MAP - FOREST                                    CYBER LOADER 1.5-650       85/11/11. 15.31.14.      PAGE     9
OVCAP.



-------- OVCAP.

         OVCAP  (26)WOODS  (27)GROUP     FOREST     WRITTEN TO FILE  (4)CAPS

         OVCAP LENGTH --           (28)66


         PROGRAM AND BLOCK ASSIGNMENTS.

         BLOCK        ADDRESS   LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

      (8)WOODS             3+       13    LG02     85/11/11  FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D      ARG=UNS
         DEER             16+       17    LG02     85/11/11  FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D      ARG=UNS
         RABBIT           35+       22    LG02     85/11/11  FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D      ARG=UNS


     (11)ENTRY POINTS.

         ENTRY      ADDRESS   PROGRAM             REFERENCES

         FOREST        2075   FOREST
         SYSAID=       2131   SYSAID=
         FCL.C.        2132   FORUTL=
         .
         .
         .
         LISTSRM      14125   LISTSRM
         RM$SYS=      14216   RM$SYS=
         RECOVR       1434C   RECOVR
         SFTUP.       14554
         CPM=         14604   CPU.CPM
         LFM=         14611   CPU.LFM
```

Figure E-4. Full Load Map of Overlay and OVCAP Generation (Sheet 3 of 5)

```
     LOAD MAP - FORFST                                   CYBER LOADER 1.5-650       85/11/11. 15.31.14.      PAGE    13
     OVCAP.

     ENTRY      ADDRESS   PROGRAM              REFERENCES

(13) WOODS    E       10+  WOODS
     DEER     E       23+  DEER
     RABBIT   E       42+  RABBIT



-------- OVCAP.

     OVCAP      NEEDLES   GROUP      FOREST    WRITTEN TO FILE     CAPS

     OVCAP LENGTH --             45


     PROGRAM AND BLOCK ASSIGNMENTS.

     BLOCK        ADDRESS   LENGTH     FILE     DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

     NEEDLES           3+       13     LG02     85/11/11  FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS
     LEAVES           16+       22     LG02     85/11/11  FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS


     ENTRY POINTS.

     ENTRY      ADDRESS   PROGRAM              REFERENCES

     FOREST       2075    FORFST
     SYSAID=      2131    SYSAID=
     FCL,C.       2132    FORUTL=
     .
     .
     .
     RECOVR      14340    RECOVR
     SETUP.      14554
     CPM=        14604    CPU.CPM
     LFM=        14611    CPU.LFM
(13) NEEDLES E       10+  NEEDLES
     LEAVES  E       23+  LEAVES
```

Figure E-4. Full Load Map of Overlay and OVCAP Generation (Sheet 4 of 5)

```
LOAD MAP - FOREST                                          CYBER LOADER 1.5-650      85/11/11. 15.31.14.     PAGE    17
OVCAP.

-------- OVCAP.

         OVCAP    BRANCH    GROUP    FOREST    WRITTEN TO FILE    CAPS

         OVCAP LENGTH --            133


         PROGRAM AND BLOCK ASSIGNMENTS.

         BLOCK        ADDRESS   LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

         BRANCH            3+       13    LG02     85/11/11  FTN       5.1 650   666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
         TRUNK            16+       42    LG02     85/11/11  FTN       5.1 650   666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
         TWIG             60+       42    LG02     85/11/11  FTN       5.1 650   666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D       ARG=UNS


         ENTRY POINTS.

         ENTRY      ADDRESS   PROGRAM            REFERENCES

         FOREST       2075    FOREST
         SYSAID=      2131    SYSAID=
         FCL.C.       2132    FORUTL=
         .
         .
         .
         LFM=        14611    CPU.LFM
      ⑬ BRANCH   E      10+    BRANCH
         TRUNK    E      23+    TRUNK
         TWIG     E      65+    TWIG

           ⑮ .259 CP SECONDS        ⑯ 37600B CM STORAGE USED          ⑰ 45 TABLE MOVES
```

Figure E-4. Full Load Map of Overlay and OVCAP Generation (Sheet 5 of 5)

```
(1) LOAD MAP - SEGMENTED LOAD.                          CYBER LOADER 1.5-650      85/11/11. 14.10.39.      PAGE     1
                                                                   (2)
(20) SEGLOAD DIRECTIVES.

              GLOBAL  GROVE,PINE,MAPLE,HICKORY,OAK
     LAND     TREE    FOREST-(FAUNA-(DEER,RABBIT))
              LEVEL
     CONE     TREE    TREE-(TRUNK,BRANCH-(LEAVES,NEEDLES))
              END


(21) TREE DIAGRAM.

     *FOREST
      ?
      ?_FAUNA
        ?
        ?_DEER
        ?
        ?_RABBIT

(22) ********************************************************************************

     *TREE
      ?
      ?_TRUNK
      ?
      ?_BRANCH
        ?
        ?_LEAVES
        ?
        ?_NEEDLES


(3)  FWA OF THE LOAD           1022
     LWA+1 OF THE LOAD        14012

(10) WRITTEN TO FILE     ABS

(5)  TRANSFER ADDRESS -- FOREST          3573



-------- SEGMENT - FOREST



(20) PROGRAM AND BLOCK ASSIGNMENTS.

     BLOCK        ADDRESS    LENGTH    FILE     DATE     PROCSSR VER LEVEL  HARDWARE   COMMENTS

(24) (FOREST)        1022         2
     /FCL.C./   GS   1024        36
(25) /Q5.IO./   GS   1062       373
     /STP.END/  GS   1455         1
     /FCL=ENT/  GS   1456        75
     /AP.IO./   GS   1553        17
     /GROVE/    G    1572         1
```

Figure E-5. Partial Load Map of Segment Generation (Sheet 1 of 4)

```
LOAD MAP - SEGMENTED LOAD.                          CYBER LOADER 1.5-650       85/11/11. 14.10.39.      PAGE      2

BLOCK        ADDRESS   LENGTH    FILE      DATE     PROCSSR VER LEVEL  HARDWARE   COMMENTS

/PINE/     G    1573      144
/MAPLE/    G    1737      310
/HICKORY/  G    2247      454
/OAK/      G    2723      620
/FDL.COM/       3543       23
FORLST          3566       33    LGO    (9) 85/11/11 FTN      5.1 650   666X I       PROGRAMOPT=0,ROUND= A/ S/ M/-D       ARG=UNS
(8) SYSAID=     3621        1 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              LINK BETWEEN SYS=AID AND INITIALIZATION CODE.
Q5NTRY=         3622      252 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              FCL5 - INITIALIZE FCL5 RUN TIME LIBRARY.
CHMOVE=         4074      171 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              CHARACTER MOVE AND CONCATENATE
COMIO=          4265       42 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              COMMON CODED I/O ROUTINES AND CONSTANTS.
FCL=FDL         4327       63 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              FCL CAPSULE LOADING
FECMSK=         4412       41 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              INITIALIZE CONSTANTS.
FEIFST=         4453        3 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              CONVERTED DATA STORAGE.
FLTOUT=         4456      323 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              COMMON FLOATING OUTPUT CODE
FORSYS=         5001     2023 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              FORTRAN OBJECT LIBRARY UTILITIES.
FORUTL=         7024      200 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              FCL MISC. UTILITIES.
GETFIT=         7224      206 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              GETFIT= - LOCATE A FIT GIVEN A UNIT DESCRIPTOR
LDOUT=          7432      315 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              LIST DIRECTED OUTPUT FORMATTING
OUTCOM=         7747      161 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              COMMON OUTPUT CODE
OUTF=          10130      252 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              LIST DIRECTED OUTPUT CONTROL
Q5RPV=         10402       14 SL-FTN5LIB  85/09/23 COMPASS 3.6 650              FCL5 - ABORT RECOVERY INITIALIZATION
CPU.CIO        10416       16 SL-SYSLIB   85/09/23 COMPASS 3.6 642              I/O FUNCTION PROCESSOR.
CPU.MVE        10434       64 SL-SYSLIB   85/09/23 COMPASS 3.6 642              MOVE BLOCK OF DATA.
CPU.SYS        10520       40 SL-SYSLIB   85/09/23 COMPASS 3.6 642              PROCESS SYSTEM REQUEST.
CMF.ALF        10560      175 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - ALLOCATE FIXED.
CMF.CSF        10755        6 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - CHANGE SPECS FIXED.
CMM.FFA        10763       14 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - FIXED FREE ALGORITHM.
CMF.FRF        10777       36 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - FREE FIXED.
CMF.GSS        11035       22 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - GET SUMMARY STATISTICS.
CMM.MEM        11057        7 SL-SYSLIB   85/09/23 COMPASS 3.6 650
CMM.R          11066      212 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - RESIDENT SUBROUTINES.
CMF.SLF        11300       22 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - SHRINK AT LWA FIXED.
FDL.RES        11322      212 SL-SYSLIB   85/09/23 COMPASS 3.6 650              FAST DYNAMIC LOADER RESIDENT.
FDL.MMI        11534      402 SL-SYSLIB   85/09/23 COMPASS 3.6 650              FDL MEMORY MANAGER INTERFACE.
CTL$RM         12136      501 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CRM CONTROLLING ROUTINE.
CTL$SKP        12637       57 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CRM CONTROLLER - SKIP PHYSICAL/FILE.
CTL$WR         12716       47 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CRM CONTROLLER - WEOX, REWIND
ERR$RM         12765       25 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CRM ERROR PROCESSOR ENTRY.
LIST$RM        13012       67 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CRM - ALLOCATE SPACE FOR LIST OF FILES
RM$SYS=        13101        5 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CRM - POST RA+1 REQUEST
RECOVR         13106      362 SL-SYSLIB   85/09/23 COMPASS 3.6 650              RECOVR - V2.0, USER INTERFACE TO *RPV*.
CPU.CPM        13470        5 SL-SYSLIB   85/08/16 COMPASS 3.6 642              82/02/26. 85/07/29. CPUREL - CONTROL POINT MANA
CPU.LFM        13475       10 SL-SYSLIB   85/08/16 COMPASS 3.6 642              82/02/26. 85/07/29. CPUREL - LOCAL FILE MANAGER
CMF.SDA        13505       47 SL-SYSLIB   85/09/23 COMPASS 3.6 650              CMM V1.1 - SET DABA.


-------- SEGMENT - FAUNA


PROGRAM AND BLOCK ASSIGNMENTS.
```

Figure E-5. Partial Loap Map of Segment Generation (Sheet 2 of 4)

```
          LOAD MAP - SEGMENTED LOAD.                              CYBER LOADER 1.5-650      85/11/11. 14.10.39.     PAGE     3

          BLOCK         ADDRESS   LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

          /GROVE/   S     1572        1
          (FAUNA)        13554        2
          FAUNA          13556       42     LGO      85/11/11 FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D         ARG=UNS


--------- SEGMENT - DEER


          PROGRAM AND BLOCK ASSIGNMENTS.

          BLOCK         ADDRESS   LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

          (DEER)         13620        0
          DEER           13620       22     LGO      85/11/11 FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D         ARG=UNS


--------- SEGMENT - RABBIT


          PROGRAM AND BLOCK ASSIGNMENTS.

          BLOCK         ADDRESS   LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

          (RABBIT)       13620        0
          RABBIT         13620       22     LGO      85/11/11 FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D         ARG=UNS


--------- SEGMENT - TREE


          PROGRAM AND BLOCK ASSIGNMENTS.

          BLOCK         ADDRESS   LENGTH    FILE     DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

          /PINE/    S     1573      144
          (TREE)         13642        2
          TREE           13644       45     LGO      85/11/11 FTN     5.1 650    666X I     SUBROUTINEOPT=0,ROUND= A/ S/ M/-D         ARG=UNS


--------- SEGMENT - TRUNK


          PROGRAM AND BLOCK ASSIGNMENTS.
```

Figure E-5. Partial Load Map of Segment Generation (Sheet 3 of 4)

```
LOAD MAP - SEGMENTED LOAD.                            CYBER LOADER 1.5-650      85/11/11. 14.10.39.      PAGE     4

BLOCK          ADDRESS    LENGTH     FILE      DATE      PROCSSR VER LEVEL  HARDWARE    COMMENTS

/MAPLE/    S      1737       310
(TRUNK)          13711         0
TRUNK            13711        26     LGO       85/11/11 FTN      5.1 650    666X I      SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS


-------- SEGMENT - BRANCH


         PROGRAM AND BLOCK ASSIGNMENTS.

         BLOCK          ADDRESS    LENGTH     FILE      DATE      PROCSSR VER LEVEL  HARDWARE    COMMENTS

         /HICKORY/ S      2247       454
         (BRANCH)        13711         2
         BRANCH          13713        52     LGO       85/11/11 FTN      5.1 650    666X I      SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS


-------- SEGMENT - LEAVES


         PROGRAM AND BLOCK ASSIGNMENTS.

         BLOCK          ADDRESS    LENGTH     FILE      DATE      PROCSSR VER LEVEL  HARDWARE    COMMENTS

         /OAK/      S     2723       620
         (LEAVES)        13765         0
         LEAVES          13765        25     LGO       85/11/11 FTN      5.1 650    666X I      SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS


-------- SEGMENT - NEEDLES


         PROGRAM AND BLOCK ASSIGNMENTS.

         BLOCK          ADDRESS    LENGTH     FILE      DATE      PROCSSR VER LEVEL  HARDWARE    COMMENTS

         (NEEDLES)       13765         0
         NEEDLES         13765        25     LGO       85/11/11 FTN      5.1 650    666X I      SUBROUTINEOPT=0,ROUND= A/ S/ M/-D        ARG=UNS

  (15) .295 CP SECONDS            (16) 315008 CM STORAGE USED           (17) 11 TABLE MOVES
     NO. OF SEGMENTS+PROGRAMS+BLOCKS USED =    98          MAXIMUM PERMITTED = 8192
```

Figure E-5. Partial Load Map of Segment Generation (Sheet 4 of 4)

```
(1) LOAD MAP - TREE                                        CYBER LOADER 1.5-650      85/11/11. 14.37.14.      PAGE      1
                                                                     (2)

              (26)              (27)
--------- CAPSULE  TREE        GROUP  MYGROUP      WRITTEN TO FILE  CAPS

          CAPSULE LENGTH --         (28) 33


          PROGRAM AND BLOCK ASSIGNMENTS.

          BLOCK        ADDRESS   LENGTH    FILE      DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

      (8) TPEE               3       15    LGO       85/11/11 COMPASS 3.6 650


          ENTRY POINTS.

          ENTRY      ADDRESS   PROGRAM               REFERENCES

          FIT      X (29) *UNSAT*                   TREE            10
     (11) GRASS    X     *UNSAT*               (13) TREE       (14) 17
          PUT$RM   X     *UNSAT*                    TREE            13
          SYS=     X     *UNSAT*                    TREE            16
          TREE     E        7   TREE

--------- CAPSULE   WOODS       GROUP  MYGROUP      WRITTEN TO FILE  CAPS

          CAPSULE LENGTH --            33


          PROGRAM AND BLOCK ASSIGNMENTS.

          BLOCK        ADDRESS   LENGTH    FILE      DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

          WOODS              3       15    LGO       85/11/11 COMPASS 3.6 650


          ENTRY POINTS.

          ENTRY      ADDRESS   PROGRAM               REFERENCES

          FIT      X     *UNSAT*                    WOODS           10
          FLOWER   X     *UNSAT*                    WOODS           17
          PUT$RM   X     *UNSAT*                    WOODS           13
          SYS=     X     *UNSAT*                    WOODS           16
          WOODS    E        7   WOODS

       (15) .009 CP SECONDS          (16) 16630B CM STORAGE USED             (17) 1 TABLE MOVE
```

Figure E-6. Full Load Map of Capsule Generation

INDEX

COMMENT SHEET

MANUAL TITLE: CYBER Loader Version 1 Reference Manual

PUBLICATION NO.: 60429800

REVISION: K

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply _____ No reply necessary

FOLD FOLD

CUT ALONG LINE

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
P.O. BOX 3492
Sunnyvale, California 94088-3492

FOLD FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAPE

LOADER FEATURES SUMMARY

Mnemonic	Code (octal)	Page Numbers	Corresponding Page Numbers		
			LDREQ Macro	Directive Table Formats	
				LDSET Macro	LDSET Table
Loader Control Statement:					
name call		2-1			
LOAD	0000	2-3	4-2	D-25	
LIBLOAD	0001	2-3	4-2	D-25	
SLOAD	0002	2-4	4-2	D-25	
CMLOAD	0003		4-4	D-25	
ECLOAD	0004		4-4	D-25	
EXECUTE	0005	2-4	4-4	D-25	
NOGO	0006	2-5	4-5	D-25	
SATISFY	0007	2-6	4-3	D-27	
LDSET		2-7			
LDSET Parameter:					
LIB	0010	2-8	4-2	D-16	D-16
MAP	0011	2-8	4-2	D-17	D-17
PRESET/PRESETA	0012	2-8	4-2	D-17	D-17
ERR	0013	2-9			D-17
REWIND/NOREWIN	0014	2-9			D-17
USEP	0015	2-9	4-2	D-18	D-18
USE	0016	2-10	4-2	D-18	D-18
SUBST	0017	2-10	4-2	D-19	D-19
OMIT	0020	2-10	4-2	D-19	D-19
ENTRY	0021		4-5	D-27	
DMP	0022		4-5	D-27	
FILES/STAT	0023	2-10	4-2	D-27/28	
PASSLOC	0024		4-5	D-28	
EPT	0025	2-10			D-19
NOEPT	0026	2-10			D-19
COMMON	0032	2-10			D-20
PD	0033	2-10	4-2	D-20	D-20
PS	0034	2-10	4-2	D-20	D-20

LITHO IN U.S.A.