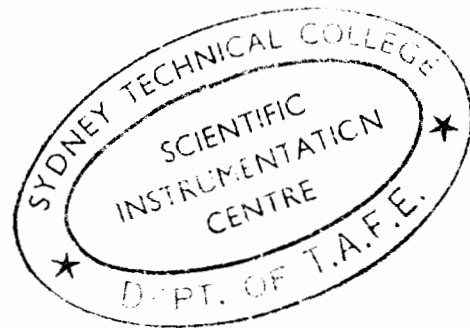




RTE-6/VM Loader

Reference Manual



PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what software manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

First Edition..... Dec 1981
Update 1 Jul 1982

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Preface

This manual describes the procedures and tools used to load programs for execution on the RTE-6/VM Operating System. It provides reference material for the experienced user, as well as descriptive information about the process of loading programs.

The manual includes documentation for the RTE-6/VM loaders, MLLDR and LOADR, and the program segmentation and loading tools, SGMTR, SXREF and INDXR. The LINK loader available in RTE-6/VM is described in a separate manual.

The RTE-6/VM loader MLLDR provides Extended Code Space - a method of loading very large programs with minor or no source code changes, using a Multilevel Segmentation, Load-On-Call scheme. The loader LOADR provides a single-level segmentation method. It can be used to load segmented programs that were developed on previous operating systems that do not provide multilevel segmentation.

Program segmentation is aided by the utility program SGMTR, which helps create a segmentation structure for the program. The utility SXREF, a cross reference listing program, also aids the segmentation process.

The utility INDXR increases the speed of MLLDR by indexing files containing relocatable code.

Manual Organization

References are made throughout the manual to "MLLDR", although, except for segmentation, most of the information applies to both loaders. Chapter 9 describes the differences between MLLDR and LOADR.

The manual is organized into nine chapters and six appendices.

Chapter 1 gives an overview of loader functions for the user who has little experience loading programs.

Chapter 2 describes loader functions in more detail, and includes command descriptions for each function.

Chapter 3 describes Multilevel Segmentation, including related commands and a sample program to demonstrate the segmentation process.

Chapter 4 describes loader operation, including command and code entry, error reporting, and the loader output listing.

Chapter 5 contains command descriptions organized for quick reference.

Chapter 6 describes the utilities SGMTR, SXREF, and INDXR.

Chapter 7 shows memory organization of loaded programs.

Chapter 8 gives procedures recommended for segmenting and loading large programs.

Chapter 9 lists differences between MLLDR and LOADR.

Appendix A contains error messages for MLLDR, LOADR, SGMTR, SXREF, and INDXR.

Appendix B contains diagrams showing relocatable record formats.

Appendix C shows sample RTE-6/VM logical memory configurations.

Appendix D contains source code for a sample program with SGMTR and SXREF output.

Appendix E gives an overview of linking performed by the loaders.

Appendix F contains a table that summarizes COMMON types and uses.

Related Documentation

The following manuals are referenced in this manual:

*** RTE-6/VM Programmer's Reference Manual:**

Describes the features provided by the RTE-6/VM Operating System which allow the user to programmatically use system services and/or control system resources. Detailed descriptions of the calling sequences that invoke system action are provided, with examples in FORTRAN, Pascal, and MACRO.

*** RTE-6/VM Terminal User's Manual:**

Describes the features of the RTE-6/VM Operating System that are available to the user in interactive mode.

*** LINK Relocating Loader Manual (92077-90009):**

Describes the features of the LINK relocating loader available in RTE-6/VM and RTE-A.1.

Preface

This manual describes the procedures and tools used to load programs for execution on the RTE-6/VM Operating System. It provides reference material for the experienced user, as well as descriptive information about the process of loading programs.

The manual includes documentation for the RTE-6/VM loaders, MLLDR and LOADR, and the program segmentation and loading tools, SGMTR, SXREF and INDXR.

The RTE-6/VM loader MLLDR provides Extended Code Space - a method of loading very large programs with minor or no source code changes, using a Multilevel Segmentation, Load-On-Call scheme. The loader LOADR provides a single-level segmentation method. It can be used to load segmented programs that were developed on previous operating systems that do not provide multilevel segmentation.

Program segmentation is aided by the utility program SGMTR, which helps create a segmentation structure for the program. The utility SXREF, a cross reference listing program, also aids the segmentation process.

The utility INDXR increases the speed of MLLDR by indexing files containing relocatable code.

Manual Organization

References are made throughout the manual to "MLLDR", although, except for segmentation, most of the information applies to both loaders. Chapter 9 describes the differences between MLLDR and LOADR.

The manual is organized into nine chapters and six appendices.

Chapter 1 gives an overview of loader functions for the user who has little experience loading programs.

Chapter 2 describes loader functions in more detail, and includes command descriptions for each function.

Chapter 3 describes Multilevel Segmentation, including related commands and a sample program to demonstrate the segmentation process.

Chapter 4 describes loader operation, including command and code entry, error reporting, and the loader output listing.

Chapter 5 contains command descriptions organized for quick reference.

Chapter 6 describes the utilities SGMTR, SXREF, and INDXR.

Chapter 7 shows memory organization of loaded programs.

Chapter 8 gives procedures recommended for segmenting and loading large programs.

Chapter 9 lists differences between MLLDR and LOADR.

Appendix A contains error messages for MLLDR, LOADR, SGMTR, SXREF, and INDXR.

Appendix B contains diagrams showing relocatable record formats.

Appendix C shows sample RTE-6/VM logical memory configurations.

Appendix D contains source code for a sample program with SGMTR and SXREF output.

Appendix E gives an overview of linking performed by the loaders.

Appendix F contains a table that summarizes COMMON types and uses.

Related Documentation

The following manuals are referenced in this manual:

*** RTE-6/VM Programmer's Reference Manual:**

Describes the features provided by the RTE-6/VM Operating System which allow the user to programmatically use system services and/or control system resources. Detailed descriptions of the calling sequences that invoke system action are provided, with examples in FORTRAN, Pascal, and MACRO.

*** RTE-6/VM Terminal User's Manual**

Describes the features of the RTE-6/VM Operating System that are available to the user in interactive mode.

Table of Contents

Chapter 1 Introduction

Program Relocation and Linking	1-1
MLLDR Operation	1-1
Segmentation	1-2
Temporary and Permanent Program Loads	1-2
Memory Allocation	1-3
Profile Option and Debugger	1-3

Chapter 2 Loader Functions

Program Relocation and Linking	2-1
Relocation	2-1
Linking	2-3
Current Page Linking	2-3
Base Page Linking	2-4
Mixed Linking	2-4
Multilevel Segmentation Load-On-Call (MLS-LOC)	2-5
Benefits of MLS-LOC	2-6
Comparison with Segmentation in Previous Operating Sys	2-6
Segmentation Tools	2-7
Program Addition, Replacement, and Deletion	2-7
Temporary Programs	2-8
Permanent Programs	2-8
Adding Permanent Programs	2-9
Replacing Permanent Programs	2-9
Deleting Permanent Programs	2-10
Program Loading Requirements and Restrictions	2-10
ID Segment Requirements	2-11
General Requirements	2-11
Memory Allocation	2-13
Program Types	2-13
Main Programs	2-14
Subroutines	2-15
Program Size	2-15
COMMON Allocation	2-16
Local COMMON	2-16
System COMMON	2-16
Reverse COMMON	2-16
EMA and Shareable EMA	2-17
Virtual Memory (VMA)	2-18
Profile Option	2-20
MLLDR and Debugger Interface	2-20

Chapter 3 Multilevel Segmentation Load-On-Call

MLS Structure	3-1
Overview	3-1
MLS Terminology	3-2
Shared Logical Address Space	3-4
Creating the MLS Tree	3-5
Sample Calling Sequence and Node Tree	3-6
Writing the Command File	3-8
MLS Commands	3-8
Sample Command File	3-11
Sample Trace of MLLDR Processing	3-13
Segmentation Structure Restrictions	3-16
Load-on-Call	3-16
MLS Tree Restrictions	3-16
Maximum Tree Size and Pathlength	3-16
Mixed Disc-Resident and Memory-Resident Nodes	3-17
Duplication of Modules	3-17
Offpath References	3-19
Circular Call Chains and Recursion	3-20
Data Considerations	3-22
Language Considerations	3-22
Pascal	3-22
FORTRAN	3-23

Chapter 4 Operation

Modes of Operation	4-1
Commands and Command File	4-2
Command File	4-5
MLS Programs	4-5
Non-Segmented Programs	4-5
Runstring Operation	4-6
Runstring Parameters	4-6
Runstring Operation Examples	4-11
Error Reporting	4-13
Undefined Externals	4-13
MLS Errors	4-14
Other Errors	4-16
Warning and Information Reporting	4-16
Duplicate Program Name	4-17
Waiting for Resources	4-17
Loadmap	4-18
Default	4-18
EC Option	4-21
LE Option	4-24
Shared Base Page	4-24
Rotating Base Page	4-24

Completion Message	4-29
Profile Output	4-31
Return Parameters for Programmatic Scheduling	4-31
Aborting MLLDR	4-32

Chapter 5 Commands

Abort (AB, or /A)	5-2
Assign (AS,xx)	5-2
Disc Node Specification (D)	5-2
Display Undefined Externals (DI)	5-2
Echo Commands to List Output (EC)	5-3
End of Command Input (EN, EX, or /E)	5-3
Format (FM,format)	5-3
Force Load (FO)	5-4
User Library File (LI,file)	5-4
List Output (LL,namr)	5-4
Set Relocation Base (LO,address or LO,+n)	5-5
Memory Node (M)	5-5
Multiple Search (MS,namr)	5-6
User Library Name (NA,name)	5-6
Opcode (OP,opcode)	5-7
Profile (PF,lu)	5-7
Relocate (RE,namr)	5-7
Save (SA,xx)	5-8
Search (SE,namr)	5-8
Shareable EMA (SH,label)	5-9
Search User Libraries (SL)	5-9
System Library Name (SY,name)	5-10
Size (SZ,xx or SZ,+n)	5-10
Transfer (TR,namr or TR)	5-10
VMA Size (VS,xxxx)	5-11
Working Set Size (WS,xxxx)	5-11
Comment (*)	5-11

Chapter 6 Segmentation and Loading Tools

Segmentation Utility SGMTR	6-1
Function	6-1
Operation	6-1
Runstring Parameters	6-1
Input File	6-2
Output File	6-2
Path Size	6-3
Main Entry Point	6-4
Segmentation Options	6-6
Load Options	6-6
Scratch File	6-7
Aborting SGMTR	6-7
Preparing the Input File	6-7
COMMON	6-7
Merged and Indexed Files	6-11
RPL's	6-11
SGMTR Output	6-12
Sample Program, Terminal Output, and Command File	6-12
Shortening the Command File	6-18
Segmentation Restrictions	6-19
Local Data: Disc Nodes and Duplicated Modules	6-19
Recursion and Offpath References	6-20
Passing Procedures as Parameters	6-20
Summary of SGMTR Errors	6-21
Cross Referencer SXREF	6-23
Operation	6-23
Output File	6-24
Scratch Files	6-24
Aborting SXREF	6-24
Syntax Checking Mode	6-25
Expected Order of Commands	6-26
Cross Reference Mode	6-27
Command File Listing	6-27
Cross Reference Listing	6-28
Sample SXREF Output	6-30
Summary of SXREF Errors	6-39
Relocatable Indexer INDXR	6-41
Operation	6-41
Commands	6-41
CR[EATE],namr	6-42
LI[ST],namr	6-42
IN[DEX],namr	6-42
EX[IT] or EN[D]	6-42
AB[ORT]	6-43
TR[ANSFER],namr	6-43
Examples	6-43

Chapter 7 Memory Allocation

Introduction	7-1
Memory Layout	7-1
Base Page to Root	7-2
Program Nodes	7-2
Root Node Only	7-2
Disc-Resident Nodes Only	7-2
Memory-Resident Nodes Only	7-3
Memory and Disc Nodes	7-3
End of Program to End of Partition	7-8
Dynamic Buffer Area	7-8
VMA/EMA Area	7-8
Memory Overhead	7-9
Multilevel Segmentation	7-9
Profile Option	7-9

Chapter 8 Loading Large Programs

Compilation Considerations	8-1
FORTRAN	8-2
Pascal	8-2
Evaluate the Program	8-3
Segmenting Using SGMTR	8-4
Path Overflow	8-6
Load Time Errors	8-6
Base Page Link Overflow	8-7
Memory Overflow	8-7
Reducing Path Size	8-8
Reducing Links	8-10
Runtime Errors	8-11
Local Data (Disc Nodes and Duplicated Modules)	8-11
Improper Use of COMMON	8-12
Offpath References	8-12
Reducing Load Time	8-12

Chapter 9 Relocating Loader LOADR

Single-Level Segmentation	9-1
Operation	9-3
Runstring	9-3
Command Input	9-3
Commands	9-3
Code Input	9-5
Completion Message	9-6
Memory Layout	9-7
Converting to Multilevel Segmentation	9-9
FORTRAN and Assembly Language	9-9
Pascal	9-9

Appendix A Errors

MLLDR and LOADR Errors	A-1
SGMTR Error Messages	A-12
SXREF Error Messages	A-17
INDXR Error Messages	A-21

Appendix B Relocatable Record Formats

Appendix C RTE-6/VM 32K-Word Logical Memory Configurations

Appendix D Sample Program

Source Code	D-1
SGMTR Output	D-5
SXREF Output	D-8

Appendix E Linking Overview

Appendix F Summary of COMMON Types

List of Illustrations

Figure 3-1.	Sample Program and Calling Sequence.	3-1
Figure 3-2.	Sample Node Tree	3-3
Figure 3-3.	Preorder Node Numbers Reported on the Load Map	3-3
Figure 3-4.	Path Numbers Used for Error Reporting and by the Debugger MLSDB	3-4
Figure 3-5.	Sample Calling Sequence.	3-6
Figure 3-6.	Sample MLS Tree Structure.	3-7
Figure 3-7.	File Organization for Sample Program	3-11
Figure 3-8.	Loader Command File for Sample Program	3-12
Figure 3-9.	Duplication of Modules	3-18
Figure 3-10.	Offpath References	3-21
Figure 4-1.	Loadmap: Default	4-19
Figure 4-2.	Loadmap: EC Option	4-21
Figure 4-3.	Loadmap: LE Option	4-25
Figure 6-1.	Declaring a Main Module in Macro	6-5
Figure 6-2.	FORTRAN 77 Example: Ordinary Labeled COMMON, Block Data Subprogram.	6-9
Figure 6-3.	MACRO Example: Block Data Subprogram	6-10
Figure 6-4.	Shortening the Command File Using EDIT	6-18
Figure 7-1.	Memory Map - Root Only	7-4
Figure 7-2.	Memory Map - Disc-Resident Nodes Only.	7-5
Figure 7-3.	Memory Map - Memory-Resident Nodes Only.	7-6
Figure 7-4.	Memory Map - Memory-Resident and Disc-Resident Nodes.	7-7
Figure 8-1.	Steps to Correct Overflow Conditions	8-5
Figure 9-1.	Memory Map - Single-Level Segmentation	9-8
Figure C-1.	RTE-6/VM 32K-Word Logical Memory Configurations	C-2
Figure E-1.	Base Page Linking.	E-2
Figure E-2.	Current Page Linking	E-3

List of Tables

Table 4-1.	MLLDR Commands	4-3
Table 7-1.	Location of Dynamic Buffer Area.	7-8
Table 9-1.	Summary of LOADR Commands.	9-4
Table F-1.	Summary of COMMON Types.	F-1

Chapter 1

Introduction

This chapter contains an overview of the functions and characteristics of the relocating loader MLLDR. The brief descriptions below are intended to introduce the new user to the capabilities of MLLDR. Chapter 2 describes these functions in more detail.

Program Relocation and Linking

MLLDR converts relocatable code (type 5 files) into a memory-image format module that is ready for execution. It sets up the linkage between the program and any required library files and system routines. Appendix E contains an overview of program linking.

MLLDR Operation

MLLDR reads relocatable code from any FMP file or logical unit. It accepts commands from its runstring, a command file, and interactively from the user's terminal. For Multilevel Segmentation (MLS) programs, a command file must be used. For non-segmented programs, a command file is not required; all commands can be entered interactively or included in the MLLDR runstring. MLLDR can be operated under control of File Manager in batch mode.

Segmentation

When a program's code exceeds the amount of logical address space available, the program must be segmented. Two forms of segmentation are available with RTE-6/VM: multilevel segmentation, performed by MLLDR, and single-level segmentation, performed by LOADR.

A program in multilevel segmentation, load-on-call (MLS-LOC) format has part of its code in logical memory at all times; the rest of the code is in physical memory, on disc, or both. Multilevel segmentation of a program is performed at load time; no changes are made to the program source code. The load-on-call mechanism, set up by MLLDR and performed by the operating system, automatically brings a called subroutine into logical memory when needed at runtime. More information about MLS-LOC is contained in Chapter 2, and Chapter 3 describes it in detail.

A single-level segmentation program has part of its code in logical memory at all times; the rest of the code resides on disc. The program's source code contains requests to the operating system to perform segment overlays. During execution, the program must request a segment overlay when the code in a segment is needed. The single-level segmentation loader, LOADR, is described in Chapter 9. Except for segmentation, its functions and operation are identical to MLLDR.

Temporary and Permanent Program Loads

MLLDR performs "temporary" and "permanent" loads. A temporary program can be stored in a type 6 file using the File Manager "SP,prog" command. Its ID segment can be removed with the "OF,prog,8" command and restored with the File Manager "RP,prog" command. A permanently loaded program's ID segment cannot be removed with the OF command; MLLDR must be used to replace or remove it.

Memory Allocation

MLLDR relocates a program within a 32K-word logical address space according to the memory map for the program's type. It sets up the memory allocation for programs that use local and system COMMON, EMA (including shareable labeled EMA), and virtual memory (VMA).

A shareable EMA partition can be shared by multiple programs. Its size is limited only by the size of physical memory. When a shareable EMA partition is used with a label file, programs that reference a COMMON block by the same label access the same data.

Virtual memory allows programs to declare up to 128M bytes (65536 pages) of data on a per-program basis; each VMA program can have up to 128 Mbytes of virtual memory. VMA allows a program that requires a very large data area (e.g. larger than physical memory) to execute in a small partition. A program's virtual memory area resides in a disc file that is created and managed by the operating system.

Profile Option and Debugger

The profile routine measures the amount of overhead incurred loading segments in an MLS program. The information provided by the profiler can be used to improve the performance of the program. The profiler is most useful for programs with disc-resident segments since the overhead for disc I/O is high.

MLLDR allows the user to specify that the debug routine MLSDB, distributed with the RTE-6/VM operating system, is to be appended to the program being loaded. This routine is used to debug the program using breakpoints, register examination, and other features.

Chapter 2

Loader Functions

This chapter describes the functions of MLLDR, and lists the commands associated with each function. For more detailed information about Multilevel Segmentation, Load-On-Call, refer to Chapter 3.

Program Relocation and Linking

MLLDR converts binary relocatable code into a memory-image format module that is ready for execution. The conversion process consists of linking the program to any required routines contained in library files or in the system library, and "relocating" the program in relation to a logical address. At load time, the user need not know the physical address or partition in which the program will execute.

Relocation

The commands associated with program relocation are summarized below. Chapter 3 describes MLS commands. Chapter 5 contains all command descriptions in alphabetical order.

- RE - Relocate the binary relocatable code that resides in the specified type 5 file or LU.
- LI - The file specified is a user library file.
- SL - Search user library files to satisfy undefined external references.
- SE - Search the specified file, or, if no file specified, search the system library to satisfy undefined external references.
- MS - Search the specified library file multiple times until a search does not resolve any undefined external references.
- SA - Allocate SAVE area (FORTRAN 7X and MACRO).
- LO - Set the relocation base for the next module to the specified address or the specified page boundary.

Loader Functions

MLS Commands:

- M - The following modules are in the specified memory-resident node.
- D - The following modules are in the specified disc-resident node.
- NA - The module containing the specified entry point name is to be found in one of the user libraries.
- SY - The module containing the specified entry point name is to be found in the system library.

A namr for the relocatable code can be specified in the MLLDR runstring, or, namr's can be specified using the RE command interactively or in a command file. FOR MLS programs, the relocatable code must reside in one or more FMP files, not on an LU.

The memory-image format module produced by MLLDR can be executed in any partition that is of the right type and is large enough to accommodate it. The location at which a program in memory-image format appears to begin is a logical address, called the "relocation base". The relocation base is within the logical (32K-word) address space, and the operating system causes each memory partition to appear to be within the first 32K words of memory. For an overview of memory organization under RTE-6/VM, refer to the On-Line Generator manual.

The program type determines the relocation base used by MLLDR. The relocation base can be changed by changing the program's type using the "OP,type" command. A program's relocation base follows any areas (table areas, driver partition, COMMON, etc.) mapped in with the program of that type. Also, space is reserved before the relocation base for saving the X- and Y-Registers, and 8 additional words reserved for use by the operating system. Information is saved in these locations when the program is swapped. The memory maps in Appendix C show the areas mapped in for each program type.

The relocation base can also be changed explicitly using the locate (LO) command. This command moves the base to the specified address or page boundary. It can also be used any time during the relocation process to set the relocation base for a particular subroutine. This is useful for optimizing program linking (linking is described below).

Linking

Three linking methods are provided by MLLDR. (An overview of linking is contained in Appendix E.) The linking method can be specified in the runstring or interactively using the format (FM) command:

FM,linking option

where linking option is one of the following:

CP - Current page linking (default)
BP - Base page linking
MP - Mixed linking

Current Page Linking

This is the default linking method used by MLLDR. When current page linking is specified, current page links are used whenever possible. A base page link is allocated when a current page link cannot be used. External references go direct (no link is used) if the referenced routine or entry point is on the same page as the referencing instruction.

Current page linking is used to reduce the number of base page links and consequently to conserve available words on the base page. Links are put in current page only on the first and last pages of a module, and only referencing instructions on this page can use the current page links. Program page crossings may cause indirect links to be generated on the base page if there are no current page link areas in the same page as the reference instruction. Thus, small modules can often make better use of current page linking than large modules.

Current page linking will probably be the option used most frequently. Base page linking is generally used only when memory space is critical, or when you need to reduce program size by a minimal number of words; (e.g., if a program is slightly too large for the partition in which you want it to run, or if the program is slightly larger than the logical address space available for the program type). Chapter 8 contains more information about using current page linking.

Loader Functions

Base Page Linking

In this linking scheme, only base page links are used. External references are forced to use base page links, even if the referenced routine is on the same page as the referencing instruction. The use of base page links can conserve program space since space in the program is not allocated for links, and the base page area is allocated whether or not it is used for program links. If there are not enough base page locations for the links required by the program, MLLDR issues the error L-OV BSE (base page overflow). This problem may be solved by using current page or mixed linking.

Mixed Linking

This linkage scheme provides a mix of current and base page linking. Current page links are used whenever possible, except for external references. External references are forced to use base page links. Mixed linking is useful in the case where both the base page limit and the program size limit are critical. For a particular program, the CP option might cause a memory overflow error (L-OV MEM), and the BP option might cause a base page overflow error (L-OV BSE). In this case, the MP option might allow the program to load with no errors.

Multilevel Segmentation Load-On-Call (MLS-LOC)

This section gives an overview of the multilevel segmentation, load-on-call (MLS-LOC) capability provided by MLLDR. Detailed information, including command descriptions, is contained in Chapter 3.

The multilevel segmentation, load-on-call capability allows very large programs to be loaded in a hierarchical segmentation structure. Program transportability is maintained since no changes to the source code are needed to segment the program.

A program in multilevel segmentation (MLS) format consists of a main program and two or more segments. The main program must reside in logical memory. A segment consists of subroutines and data, and can be memory-resident or disc-resident. The amount of memory-resident code is limited only by the size of user-available physical memory (the largest mother partition on the system), since memory-resident code is mapped in and out of the logical address space. Each segment can have one or more descendent (son) segments. Thus, the segmentation structure has multiple levels, and can be represented by a tree structure.

Transfer of control between segments (referred to as "nodes" throughout the rest of this manual) occurs via subroutine calls. The load-on-call mechanism, set up by MLLDR and performed by the operating system, automatically makes a called subroutine available for execution. For a routine that is in physical memory but not in a node that is mapped into the program's logical address space, the operating system performs the required mapping operation. For a routine that is part of a disc-resident node, the operating system moves the node from disc into memory and maps it into the program's logical address space. Thus, the LOC function makes segmentation transparent at the program code level; a program's segmentation structure can be changed (for example, to use more or less memory) without changing the program source code.

Benefits of MLS-LOC

The benefits of this method of segmentation are described below. (Limitations are described in Chapter 3.)

1. All of user-available physical memory can be used for program code. A larger amount of memory-resident code can greatly increase program performance.
2. No program source code is used to implement segmentation, and the amount of memory to be used for program execution is determined at load time. This provides ease of program transportability.
3. The load-on-call function is very fast. Approximate times for load-on-call overhead for memory-resident routines are:

455 nanoseconds if the called routine is already mapped.
34 microseconds if the called routine is not already mapped.
4. Multilevel segmentation allows programs to be loaded that, due to program structure, could not be loaded using a single-level segmentation scheme.

Comparison with Segmentation in Previous Operating System

RTE operating systems prior to RTE-6/VM use a single-level segmentation scheme. Segmentation is accomplished by including in the program code requests for the operating system to perform segment overlays. The segment overlay operation is invoked by an EXEC 8 request or by a call to the system routine SEGLD. MLLDR only provides the MLS-LOC method of program segmentation. Segmented programs written under previous versions of RTE can be loaded using the RTE-6/VM single-level segmentation loader LOADR, or they can be converted to use the MLS-LOC segmentation method. The single-level segmentation loader LOADR is documented in Chapter 9 of this manual. Procedures for converting single-level programs to MLS-LOC programs are described in Chapter 8.

Segmentation Tools

Two utilities are provided to aid development of MLS programs. The segmenter utility SGMTR reads a program's relocatable code and produces a segmentation structure for the program in the form of an MLLDR command file. The utility SXREF generates program cross reference listings (for modules and entry points) and checks the validity of MLLDR command files. For cross reference listings that include program variables, the compilers and Macroassembler each have a cross reference option. SGMTR and SXREF are described in Chapter 6.

Program Addition, Replacement, and Deletion

The loader can be used to load temporary programs, add permanent programs, and replace or delete permanent programs. On many systems, modification of permanent system programs is done only by the System Manager. These operations are invoked using the following commands:

- OP,TE - Perform a temporary load of the program. This is the default load type.
- OP,PE - Load the program permanently (only MLLDR can remove it).
- OP,RP - Replace a permanent program with a new version having the same name.
- OP,PU - Delete a permanent program.

The status program WHZAT can be used to list the names of all programs currently loaded on the system, along with their load type and other information. Use the FMGR command "WH,PR".

Temporary Programs

The default operation of MLLDR is to perform a temporary load. A temporary program resides on system scratch tracks allocated by the loader. It can be removed using the "OF,prog,8" command. When operating under Session, temporary programs are removed automatically when the user logs off. A temporary program can be saved, however, using the File Manager SP command. This stores the program in the type 6 file specified in the command. The program can later be restored using the File Manager RP command. This sets up an ID segment for the program, which is then ready to be executed. The File Manager RU command will automatically restore a temporary program from a type 6 file if the program does not already have an ID segment assigned by the RP command.

Permanent Programs

Permanent programs are disc-resident programs loaded during generation, and programs loaded on-line using MLLDR with the "permanent" (OP,PE) option. A permanent program cannot be removed with the OF command; MLLDR must be used to delete or replace it. MLLDR can add, replace, and delete only disc-resident programs; it cannot perform these operations on memory-resident programs.

Loader Functions



Adding Permanent Programs

The "OP,PE" command is used to add a new permanent program to the system. The program is stored on a complete disc track or several contiguous tracks. The new tracks allocated are assigned to the system and are software-protected; the tracks can be accessed only by MLLDR or the system. A blank ID segment is allocated to record the program's memory and disc boundaries, name, type, priority, assigned partition, time values, and other information.

Programs can be permanently loaded as an alternative to loading them during generation. The advantage is that the generation takes less time. Loading online, however, does not make as efficient use of disc space; the generator allocates disc space on a sector basis, the loader allocates on a track basis.

The section "Program Loading Requirements", below, contains a list of conditions which must be met before a program can be permanently loaded.

Replacing Permanent Programs

The "OP,RP" command is used to replace a permanent program. The new program, which must have the same name as the program to be replaced, is relocated onto one complete track or set of contiguous tracks. The ID segment of the old program is updated with information about the new program. The old disc area is deallocated by the system.

Loader Functions

Deleting Permanent Programs

A permanent program which has been loaded online with MLLDR or loaded during generation can be deleted using the "OP,PU" command. (Programs loaded using LOADR must be purged using LOADR, not MLLDR.) The program's ID segment is blanked out and made available for loading another program. The tracks containing the program are released. For programs that were saved using the File Manager SP command, the ID segment is blanked out, but the tracks are not released (this is equivalent to the "OF,prog,8" command). The purge operation cannot be used from batch mode, entered from a command file, or used during program relocation. It can be performed interactively or from the runstring, and it can only be used before any relocation takes place.

When performing the purge operation interactively, the prompt:

```
/MLLDR: PNAME?
```

is output on the terminal in response to the "OP,PU" command. Enter the name of the program to be purged.

Only one program can be purged per execution.

Enter /A to terminate the loader and prevent the purge operation.

The purge operation can be specified in the runstring, as follows:

```
RU,MLLDR:IH,,PROG,,PU
```

The program named PROG is purged from the system. The error L-RP MLS will be issued if an attempt is made to use MLLDR to purge a program that was loaded by LOADR.

Program Loading Requirements and Restrictions

This section describes requirements and restrictions for loading temporary programs and adding, deleting, and replacing permanent programs. If an error occurs during a load, refer to Appendix A for a description of the error and possible corrective actions. This section, and Chapter 4, MLLDR Operation, also contain information about what to do if an error occurs.

Loader Functions

ID Segment Requirements

Each main program on the system is identified by an ID segment. The ID segment contains the program's name, priority, time parameters, main memory and base page addresses, location on disc-resident or, for memory-resident programs, location in memory. For programs that use EMA, an ID segment extension is also used. For more information about the contents and format of ID segments, refer to the Programmer's Reference Manual.

During system generation, the System Manager specifies the number of free ID segments and free ID segment extensions to be included in the system. This determines the number of programs that can be ready for execution at any given time, including memory-resident and both temporary and permanent disc-resident programs. An ID segment must be available for both temporary and permanent loads, except when a permanent program is being replaced.

The status command "WH,PR" can be used to find out how many free ID segments are currently available in the system.

If MLLDR cannot obtain an ID segment for the program it is loading (all ID segments are currently in use), it issues the error L-NO IDS and aborts. If it cannot obtain an ID segment extension for an EMA program, it issues the error L-ID EXT and aborts. If there are not enough ID segments when a user attempts to restore a program from a type 6 file, File Manager issues the error FMGR 014 in response to the RP command. In all of these cases, a program must be removed before the new program can be loaded. Use the OF command to remove temporary programs. To remove permanent programs, use the "OP,PU" command from either MLLDR or LOADR, whichever was used to load the program.

General Requirements

The following restrictions apply when adding, replacing, or deleting a permanent disc-resident program:

- The session capability level of 60 is required for operations on permanent programs.
- The original program MLLDR must be used, not a copy of MLLDR. Use the runstring RU,MLLDR:IH to inhibit copying of the loader.

Below is a list of problems that can occur when loading a temporary or permanent program.

Loader Functions

1. System or reverse COMMON is requested by the program but the program's COMMON length exceeds the length of the COMMON area available.
2. Blank COMMON was not declared in the first relocatable module encountered by the loader, but is declared in a subsequent module.
3. The base page linkages exceed the size of the base page linkage area for disc-resident programs established by the system during generation.
4. The length of the memory-image unit, including current page links, exceeds the address space available for programs of that type, or for the partition the program has been assigned to.
5. Disc space is not available to store the program.
6. An ID segment or extension is not available for the program.
7. Shareable EMA is declared, but the EMA label does not appear in the system.
8. An attempt was made using MLLDR to replace a program that was loaded with LOADR (or vice-versa).
9. On some systems, the disc write-protect switch is kept on. The write-protect switch must be turned off before a program can be loaded.

A program to be replaced or deleted must meet the following requirements:

1. The program must be dormant.
2. It must not currently occupy a partition (for example, the program terminated saving resources).
3. It must not be in the time list.
4. It must have a zero point of suspension.
5. It must have been loaded originally by MLLDR.

Memory Allocation

The loader allocates space in the program's logical address space for the program's code and data. It reserves space for SSGA, Table Areas I and II, and System Driver Area, depending on the program's type. It reserves space for system or local COMMON depending on whether the program requests it or the user requests it using a command to MLLDR.

One page in the logical address space, the base page, is always reserved for storing program links and information recorded when the program is swapped. If current page linking is used, MLLDR allocates space in the program for links.

The loader also sets up the allocation of memory used that is outside of the program's logical address space. This can include EMA, shareable EMA, or virtual memory. Also, in some MLS programs memory-resident nodes are mapped from physical memory into the program's logical address space during execution.

This section describes the MLLDR commands that can be used to control memory allocation. It is divided into the following sections: program types, COMMON allocation, EMA and shareable EMA, and virtual memory. Chapter 7 contains more detailed information about memory allocation, including program links and allocation for program code in MLS format.

Program Types

When a program or subroutine is assembled or compiled, it can be assigned a type, which is stored in the module's NAM record. The type information is used mainly by the On-Line Generator, and is used in some cases by the loader. Refer to the Program Types table in the Programmer's Reference Manual for a description of the various program and subroutine types. Note that the type information in the table is described in terms of how it is used by the On-Line Generator. This section describes how module type information is used by MLLDR.

Loader Functions

Main Programs

For a main program, the default is type 6, Extended Background; the default is not the type contained in the NAM record. MLLDR determines which module is the main program by finding a module whose END record contains a transfer address. The transfer address is the address of where the program is to start execution. In FORTRAN and Pascal, this module contains the "PROGRAM xxxxx" statement. In assembly language, this module contains a label for the transfer address in the operand of the END statement. If multiple assembly modules contain a transfer address in the END statement, the first one encountered by MLLDR is used as the main program module.

The type of a program can be changed from the default type of Extended Background by using the following commands:

OP,type code - change the program's type to the type specified, where type code = RT, BG, LB, or EB.

OP,SS - request SSGA access.

OP,SC - request system COMMON (discussed below).

OP,RC - request reverse COMMON (discussed below).

For programs loaded with SSGA access, or system or reverse COMMON, the default type is type 4, Large Background. COMMON allocation is described in the next section.

The relationship between MLLDR opcodes, NAM types, and program types is shown below.

NAM Type	MLLDR Opcode	Program Type
-----	-----	-----
2	RT	Real-Time
3	BG	Background
4	LB	Large Background
0,6	EB (default)	Extended Background

where NAM Types 2, 3, 4, 6, and 0 are main programs. (NAM Type 5 indicates a single-level segment, which must be loaded using LOADR. MLLDR aborts when it encounters a type 5 module.)

Loader Functions

Characteristics of each program type are:

Type 2 (Real-Time) - Relocated with access to Table Area II. The logical map also includes System Driver Area and Table Area I. When the program is run, the system assigns it to a Real-Time partition if one is available.

Type 3 (Background) - Has the same logical map as type 2. When the program is run, the system assigns it to a Background partition if one is available.

Type 4 (Large Background) - Table Area II and the System Driver Area are not included in the program's address space. This provides a larger logical address space for the program.

Type 6 (Extended Background) - Nothing beyond the base page is included in the program's address space. This provides the largest possible logical address space (31 pages) for the program. NAM Types 0, 2, 3, 4, and 6 default to Type 6 unless system COMMON or SSGA are specified, in which case they default to Type 4, Large Background.

Subroutines

MLLDR handles type 6, 7, and 14 modules as though they were normal subroutines (type 7) to be appended to any program that makes a reference to them. (Note: these types are treated by the On-Line Generator as described in the Program Types table in the Programmer's Reference Manual.)

Program Size

At the end of the load, program size information is reported. The completion message, described in Chapter 4, reports the size of the program's longest path, required partition size, and VMA or EMA sizes.

The size of the program's dynamic buffer area can be set during the load with the loader SZ command, or it can be altered after the program is loaded using the breakmode SZ command (for non-EMA programs). A program's EMA size can be altered after the program is loaded using the breakmode SZ command. A VMA program's VM size and working set size can be set during the load using the loader VS and WS commands, or after the load using the breakmode VS and WS commands.

COMMON Allocation

The "OP,SC" and "OP,RC" commands affect how the loader handles a program that uses blank (unlabeled) COMMON. Labeled COMMON (ordinary, EMA, and SAVE) are loaded according to information contained in the program's relocatable records. Refer to Appendix F for a summary table of COMMON types.

One of three options can be specified at load time when allocating a COMMON area for a program.

Local COMMON

This is the default COMMON type. The corresponding command is "OP,NC". The local COMMON area for a program is allocated starting at the program's load point, immediately before the program. The COMMON area will be swapped together with the program. It must be declared in the first module loaded, and this declaration must be the largest in the program.

System COMMON

Use the "OP,SC" command. This implies a Background or Large Background program with COMMON in the background system COMMON area, or a Real-Time program with COMMON in the Real-Time COMMON area. Real-Time and Background system COMMON areas are established when the system is generated.

Reverse COMMON

Use the "OP,RC" command. This implies a Background program with its COMMON in the Real-Time COMMON area, or a Real-Time program with its COMMON in the Background system COMMON area.

EMA and Shareable EMA

Two commands affect the use of EMA and shareable EMA:

OP,EM - The program is an EMA program.

SH,label - The program's EMA area is to reside in the shareable EMA partition specified by label.

The "OP,EM" command specifies that the program uses EMA. This command is only necessary when loading a multilevel program in which no module in the root declares EMA but a module outside the root does.

The "SH,<label>" command specifies that the program's EMA area is to reside in the shareable EMA partition designated by <label>. The label is assigned to the partition during system generation or reconfiguration. Only one shareable EMA partition can be used per program. Note: The shared memory must be EMA; shared virtual memory is not allowed.

A program can use shareable EMA in one of two ways, depending on how it is loaded. At load time the command "SH,<label>" is used to specify a shareable EMA partition. If a shareable EMA label file with the name <label> exists on LU 2, then this file is used during the load.

The label file contains EMA COMMON block labels and sizes. For programs loaded with a label file, all data references to an EMA COMMON block, where the COMMON block label is contained in the label file, will access the same COMMON block in all programs. The Programmer's Reference Manual contains more information about using shareable EMA.

CAUTION

A shareable EMA label file should be used when the programs to be loaded contain multiple EMA COMMON blocks. This binds each label to the correct address. Declaring the blocks in the same order in all programs does not guarantee that the labels will be bound to the same address in different programs. Also, local EMA should not be used in programs that use shareable EMA, since there is not guarantee that a local EMA variable will be bound to the same address in different programs.

Virtual Memory (VMA)

For programs that require a very large amount of data storage, virtual memory (VMA) can be used. Up to 128M bytes (65536 pages) of virtual memory can be used for data. At the program source code level, there is no difference between EMA and virtual memory. At load time, the user specifies that virtual memory is to be used, and the size of the VM area is specified.

A program's virtual memory area resides in a file, called the "backing store file", that is created and managed by the operating system. (The backing store file can also be managed by the user program. Refer to the Programmer's Reference manual.) At any given time, a portion of virtual memory, called the "working set", resides in physical memory. Virtual memory uses a "demand-paged" scheme; when a program makes a memory reference, the operating system checks if the required page of memory is already in the working set, and, if necessary, moves the page from the backing store file into the working set. The size of virtual memory and the working set are transparent at the program code level.

Note that EMA is a subset of VMA where the working set size equals the VMA size.

Three MLLDR commands affect the program's use of VMA:

- OP,VM - The program uses Virtual Memory with the default environment: VMA size = 8192 pages, working set size = 31 pages.
- VS,xxxxx - Specified that xxxxx + 1 pages of VMA will be available for the program. Default = 8192 pages (xxxxx = 8191).
- WS,xxxx - Specifies the size of the working set. Default = 31 pages.

The "OP,VM" command specifies that the program is a VMA program and sets the default sizes of the working set and backing store file. If the EMA size specified in the program is greater than the default VMA size (8192 pages), then the VMA size will be set to the EMA size. If the EMA size specified in the program is less than the default working set size (31 pages), then the working set size is set to the EMA size.

Loader Functions

The VS and WS commands are used to explicitly set the size of the VMA area and working set. These commands will override the defaults set by the "OP,VM" command.

The command "VS,xxxx" sets the VMA size to $xxxx + 1$ pages, where $31 \leq xxxx \leq 65535$. If the EMA size specified in the program is greater than the specified VMA size, then the VMA size will be set to the EMA size.

Note: If $175777B < \text{VMA size} < 177776B$ (decimal, $64511 < \text{VMA size} < 65534$) then the VMA size will be set to $177777B$ (decimal, 65535). Numbers in this range are reserved for special use by the virtual memory system.

The command "WS,xxxx" sets the working set size to $xxxx$ pages, where $5 \leq xxxx \leq [\text{max partition size} - \text{code size} - 1]$ (for the VMA/EMA page table). If the EMA size specified in the program is smaller than the specified working set size, then the working set size is set to the EMA size.

Note: In certain situations, described above, the VM size and working set size can be set by the loader to values other than the default values (8192 and 31), or the values specified in VM and WS commands. These values are set transparently; they are not reported at the time the VM, WS, and "OP,VM" commands are entered. The VM and working set sizes are reported at the end of the load, and they can be displayed using the breakmode SZ command.

For more information about using virtual memory, refer to the Programmer's Reference Manual. Also, memory allocation performed by MLLDR for VMA/EMA programs is described in Chapter 7.

Profile Option

In order to optimize the MLS structure set up by SGMTR, a profiling routine can be added to MLS-LOC programs. Use the command "PF,lu", where lu is the Logical Unit to which the profile output is to be sent.

Profiling the load-on-call mechanism can vastly improve program performance. The profile reports the number of disc and memory map faults which occurred during a run of the program. (A fault occurs when a call is made to a routine in a node that is not currently mapped.) The node numbers and the number of times each node was called via a fault will be printed at the specified LU. Using these results, the user can then modify the load structure of the program to reduce the number of faults. This is most useful for programs with disc-resident nodes, since memory map faults for memory-resident nodes cause insignificant overhead. The fault overhead for disc-resident nodes, however, is high due to disc I/O time.

The profile option forces normal termination of the program; the program cannot terminate saving resources or serially reusable.

Sample profiler output is shown in Chapter 4.

MLLDR and Debugger Interface

When setting a breakpoint in an MLS-LOC program using MLSDB, the user must supply the node "path number". Path number is defined in the section MLS Terminology in Chapter 3. For more information about the debugger subroutine, refer to the RTE-6/VM Debug Subroutine manual.

Chapter 3

Multilevel Segmentation Load-On-Call

This chapter describes the procedures used to load a program in MLS-LOC format. It describes the commands that are used, and contains an example that shows how to derive an MLS structure for a program, and create a command file to load the program. Information about the memory organization of MLS-LOC programs is contained in Chapter 7, Memory Allocation.

MLS Structure

Overview

Because transfer of control between nodes occurs via subroutine calls, a program's segmentation structure is based mainly on the organization of the program's subroutines. This organization can be expressed in terms of the calling sequence (the sequence in which subroutines are called) starting at the main program level. Figure 3-1 shows a "program" containing four subroutines. On the right is a diagram of the program's calling sequence, derived from the sequence of subroutine calls.

```
PROGRAM MAIN
  CALL SUBA
  CALL SUBB
  END { MAIN }
SUBROUTINE SUBA
  CALL SUBC
  CALL SUBD
  END { SUBA }
SUBROUTINE SUBB
  { Contains no subroutine calls }
  END { SUBB }
SUBROUTINE SUBC
  { Contains no subroutine calls }
  END { SUBC }
SUBROUTINE SUBD
  { Contains no subroutine calls }
  END { SUBD }
```

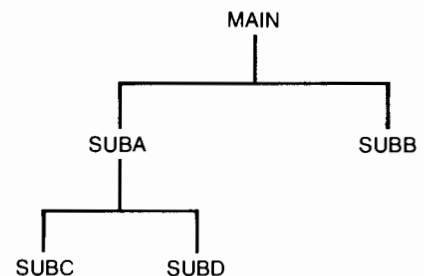


Figure 3-1. Sample Program and Calling Sequence

Multilevel Segmentation Load-On-Call

The principle used in multilevel segmentation is that only one path in the calling sequence needs to be available for execution (in memory and mapped in) at any time. If SUBB, for example, causes the program to be larger than the maximum available logical address space (31K words), then the program must be segmented so that SUBB is not mapped in when the other subroutines are executing.

The advantage of the multilevel segmentation scheme is that, for example, if SUBD also causes the program to be too large (even when SUBB is a separate segment), the program can still be segmented, since SUBC need not be available when SUBD is executing, and vice-versa.

MLS Terminology

This section introduces the terminology used in this manual to describe multilevel segmentation and load-on-call.

A "module" can contain the main program, data, and subroutines. (In this manual, "subroutine" means a FORTRAN subroutine or function, or, in Pascal, a procedure or function. The terms "routine" and "subroutine" are used interchangeably.) In assembly language terms, a module begins with a NAM statement, followed by code, data, or both, followed by an END statement. In terms of the relocatable records processed by the loader, a module begins with a NAM record and ends with an END record.

A "node" is a group of modules. A program's organization of nodes, defined by the user or by the SGMTR utility, is a "node tree", or "MLS tree". Nodes in an MLS tree are named using the M and D commands as shown in Figure 3-2. Node M is the "root" node, which is always memory-resident. Node M is the "parent" of three "son" nodes: M.1, M.2, and D.3. Node M.1 has son nodes M.1.1 and M.1.2. Node M.2 has no son nodes. Node D.3 has sons D.3.1 and D.3.2.

A "leaf" node is a node that does not have any son nodes. There are five leaf nodes in Figure 3-2. "Brother" nodes are nodes that are at the same level in the tree and have the same parent node. Nodes M.1, M.2, and D.3 are brother nodes. Nodes M.1.2 and D.3.1 are not brother nodes.

A "path" is a series of nodes starting at the root. The length of a path can be described in terms of number of nodes or length of code. Length of code can be approximated by the user (or the by SGMTR program), but measuring the length accurately can be difficult due to, for example, space allocated at load time for current page links.

Multilevel Segmentation Load-On-Call

The "depth" of a tree is the length of the longest path.

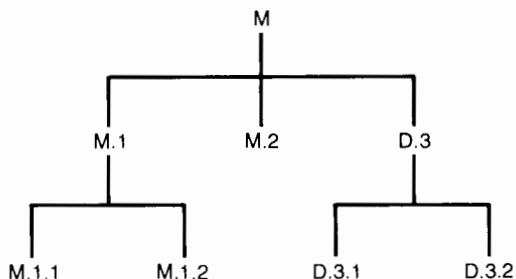
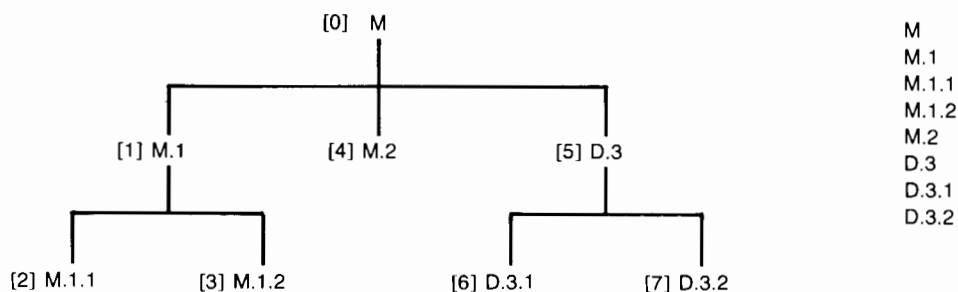


Figure 3-2. Sample Node Tree

MLLDR relocates nodes in "preorder". The nodes are numbered sequentially in preorder starting with the number 0 for the root. These numbers are listed in the loadmap. Figure 3-3 shows an example of preorder tree traversal. The numbers in brackets ([]) show the preorder "node numbers", and the node names are listed in preorder on the right side of the figure.

One use for node numbers is in program profiling. The profiler is appended to the program by using the PF command. It reports the number of node "faults" that occurred during execution of the program. A fault occurs when a routine is called that is not in a node that is already mapped.



8100-32

Figure 3-3. Preorder Node Numbers Reported on the Loadmap

Multilevel Segmentation Load-On-Call

"Path numbers" are used in system error reporting and by the user when setting breakpoints with the debugger MLSDB. For a particular node, the path number is the node number of the leftmost leaf node below that node. The path goes from the root node to the leaf node. The example in Figure 3-4 shows how path numbers correspond to node numbers. Some runtime errors reported by system include the program address at which the error occurred. For MLS programs, the error message also includes LEAF NODE = x, where x is the number of the path that was mapped in when the error occurred.

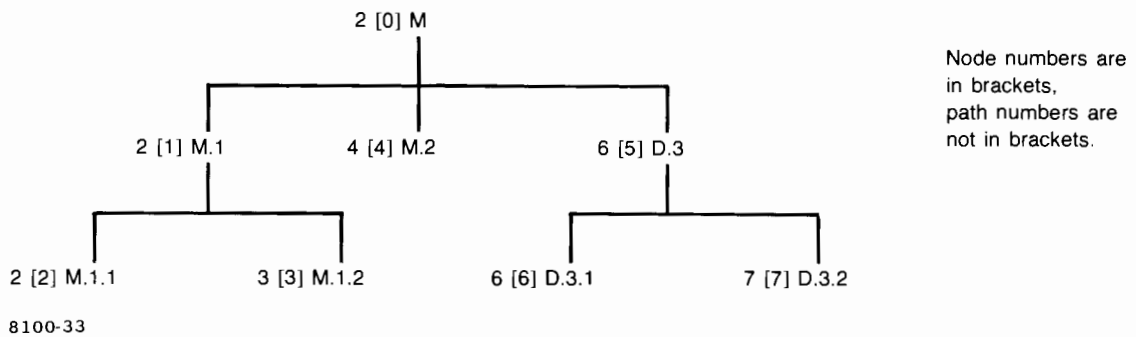


Figure 3-4. Path Numbers Used for Error Reporting and by the Debugger MLSDB

Shared Logical Address Space

The largest possible MLS pathlength is 31K words (not including base page), which is allowed when loading the program as type Extended Background. Brother nodes are relocated in relation to the same starting logical address, or relocation base. In Figure 3-2, for example, nodes M.1, M.2, and D.3 are loaded in relation to the same relocation base, which is determined by the size of the root node M and the program's type.

For very large programs, a broader MLS tree is used. Since brother nodes "share" logical address space, a broader tree can reduce the length of the longest path in the tree. Note that no logical address space is gained when a node has no brother nodes, since no logical address space is shared.

Note about memory allocation: Memory-resident nodes always begin on page boundaries. Memory allocation is described in Chapter 7.

Creating the MLS Tree

The SGMTR utility can be used to create a preliminary command file for the program. SGMTR, however, will not handle all programs, and optimization of the segmentation structure, if necessary, must be performed by the user. The following sections give information about the segmentation process, for use when segmenting manually or when modifying a command file produced by SGMTR.

Usually there are many possible node trees for a program. The program's calling sequence affects the node structure because a routine can call only one level down the tree. The size of the program's longest calling sequence path can also affect the segmentation structure, because steps can be taken, such as duplicating modules, to keep the longest path in the node tree within the 31K limit. Also, the amount of memory available in the system may determine how much of the program can reside in memory-resident nodes.

After the program loads using a preliminary command file (created using SGMTR or manually by the user), the segmentation structure can be modified to optimize the program's performance and use of memory. The situations described below show how segmentation structure for a program might be modified according to program performance needs, or system memory requirements.

- For a program where overhead for disc nodes is not a problem, use the smallest partition allowed by the program's maximum pathlength, and place a large amount of code in disc nodes. This can also be done when memory must be preserved for other programs that must execute concurrently without swapping, or for other uses such as shared EMA.
- For a program that requires high performance (fast response time, for example), place all code, or only critical code, in memory nodes and use as large a partition as needed.
- For a very large program, the goal is simply to load the program in any way possible. Use as many memory or disc nodes as needed. It may be necessary to use disc nodes, since they are not aligned to page boundaries.

These are only a few of the strategies that can be used when creating the MLS structure. Note that SGMTR can be used to create the preliminary command file; the user can then optimize the segmentation structure.

Multilevel Segmentation Load-On-Call

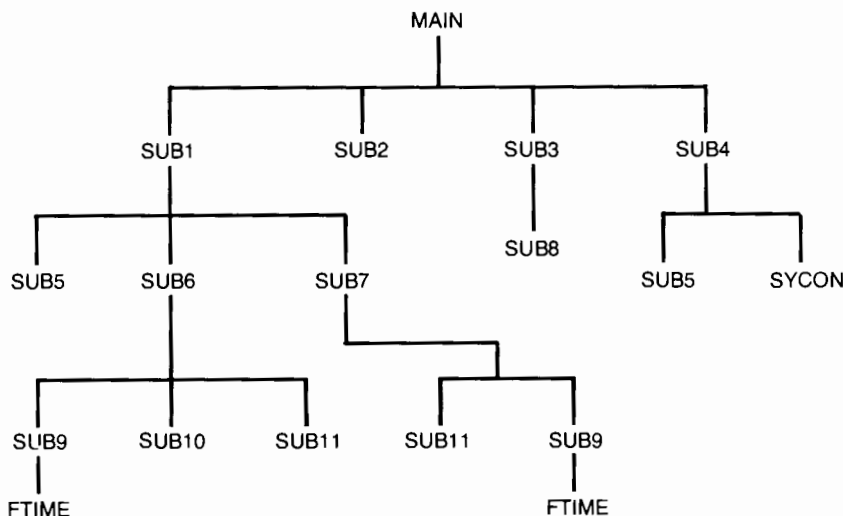
In general, greater program modularity allows a program to be loaded with a shorter maximum pathlength. A program that makes use of many small subroutines has less segmentation restrictions than a program that has fewer, longer subroutines and uses GOTO statements. Separation of code and data also can reduce segmentation restrictions. Because the transfer of control between nodes occurs via subroutine calls, a subroutine cannot be divided; it must be completely contained within one node. Note also that the loader relocates an entire module at a time, which can contain more than one subroutine. When a subroutine's entry point satisfies an undefined external, the entire module containing the subroutine is relocated, including any data and code in the module that is not a part of the subroutine.

More information about segmentation structure is contained in Chapter 8, Loading Large Programs.

Sample Calling Sequence and Node Tree

A node tree can be created by examining the program's calling sequence. The SXREF utility, described in Chapter 6, provides information to help analyze the program's calling sequence and path lengths.

A sample calling sequence is shown in Figure 3-5. Figure 3-6 shows a possible MLS tree. The next section continues this example, showing a command file for this program, followed by a trace of MLLDR processing during the load of the program.



8100-34

Figure 3-5. Sample Calling Sequence

Multilevel Segmentation Load-On-Call

The source for the program whose calling sequence is shown in Figure 3-5 is contained in Appendix D, along with sample SGMTR and SXREF output for the program.

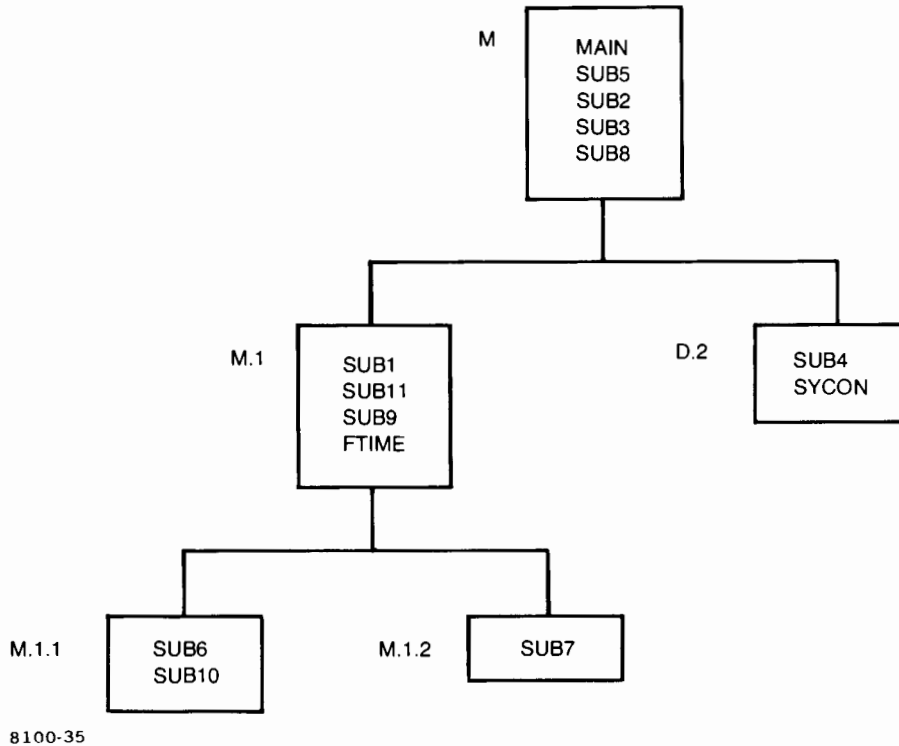


Figure 3-6. Sample MLS Tree Structure

Note: The node tree in Figure 3-6 shows only one of many possible node trees that could be created for a program with the calling sequence shown in Figure 3-5.

Since SUB5 is called by routines in both M.1 and D.2, it was put in the root node. SUB11 was relocated with M.1 since routines in both M.1.1 and M.1.2 call it, and SUB9 was placed in M.1 since routines in M.1.1 and M.1.2 reference it. FTIME was placed in M.1 since routines in M.1.1 and M.1.2 call it (indirectly). It would also have been possible to load separate copies of SUB5, SUB11, SUB9, and FTIME into all the nodes which reference them (subject to the restrictions described later in this chapter in the section "Duplication of Modules").

Writing the Command File

When the node tree has been defined, it is then translated into an MLS command file. The command file can then be checked for errors using SXREF, which performs a quick syntax check for many errors that would abort the load.

MLS Commands

The commands used in the node specification part of the command file, listed below, are used to define the nodes in the segmentation structure, and to specify which modules reside in which nodes.

A command file is required for the node specification and relocation part of the load process. When loading a program in MLS format, MLLDR needs to know what routines are one level below the node currently being relocated. It finds this information by looking ahead in the command file. The loader looks at NA, SY, and RE commands to find an entry point one level down in the command file. For this reason, interactive command entry is not allowed during the node specification stage of the load process.

In the descriptions below, "current node" refers to the node that MLLDR is constructing at the time the command is processed.

M Command

The M command specifies that the following routines until the next M or D command are to be relocated in the same memory-resident node. M by itself specifies the main. M.i where i is a positive integer, specifies a memory-resident node on the first level (where the main is level 0). M.i.j specifies a node on the second level which is a descendent of node M.i. For each new level, another .x is appended to the M and for each new node in that level, x is incremented to x+1. Figure 3-2 shows an example of how node numbers assigned using the M command correspond to the position of the nodes in the MLS tree. Note: Memory nodes require more path space, since they are aligned to page boundaries.

Multilevel Segmentation Load-On-Call

D Command

The D command specifies that the following routines until the next D command are to be relocated in the same disc-resident node. Since the main of a program is always memory-resident, the disc-resident nodes start at level one, i.e., D.i specifies a disc-resident node at the highest level available to disc-resident nodes. D.i.j specifies a disc-resident node on the second level which is a descendent of node D.i. For each new level, another .x is appended to the D.i and for each new node on that level, x is incremented to x+1. Figure 3-2 shows an example of how node numbers assigned using the D command correspond to the position of the nodes in the MLS tree.

LI Command

This command sets up the specified file as a user library file. Up to ten files can be specified as user library files. The SL command causes these files to be searched, and the files are searched automatically at the end of each node's load.

NA Command

The "NA,<name>" specifies that the module containing the entry point named <name> is to be found in one of the user-specified libraries and is to be loaded with the current node. Up to 10 relocatable files can be specified to be user library files using the LI command.

SY Command

The "SY,<name>" specifies that the module containing the entry point named <name> is to be found in the system library and is to be loaded with the current node.

RE Command

The relocate command loads all code in the specified file with the node currently being loaded.

Multilevel Segmentation Load-On-Call

SE Command

The SE command searches the specified file to satisfy undefined externals in the node currently being loaded. If no file is specified, the system library is searched. All modules found that satisfy undefined externals are loaded with the current node. The system library is automatically searched at the end of each node's load.

MS Command

The MS command searches the specified file to satisfy undefined externals in the current node. The file is searched multiple times to satisfy backward references. If a module that satisfied one undefined external causes new undefined externals, then the file is searched again to attempt to resolve the new undefined externals. All modules found that satisfy undefined externals are loaded with the current node.

SL Command

The SL command searches the set of user libraries, specified previously with the LI command, to satisfy undefined externals in the node currently being loaded. All modules found that satisfy undefined externals are loaded with the current node. Each user library is searched multiple times to satisfy backward references. User libraries are automatically searched, if necessary, after each node's load.



Sample Command File

This section continues the example started in Figures 3-5 and 3-6. Figure 3-7 shows the organization of modules in relocatable files. Figure 3-8 shows a command file derived from the node tree and file organization.

FILE NAME:	SUBROUTINES IN FILE:
\$LIB1	SUB8, SUB12
\$LIB2	SUB9, SUB13
%MAIN	MAIN, SUB5
%SUB2	SUB2, SUB3
%SUB1	SUB1
%SUB11	SUB11
%SUB6	SUB6, SUB10
%SUB7	SUB7
%SUB4	SUB4
(system relocatable library)	SYCON, FTIME

Figure 3-7. File Organization for Sample Program

Multilevel Segmentation Load-On-Call

```
LI,$LIB1          ,,$LIB1 is a library file
LI,$LIB2          ,,$LIB2 is a library file
OP,RTSCSS        ,,Realtime, System COMMON, SSGA
*
*
M                ,,,The following are in the main
RE,%MAIN         ,,routines in %MAIN (MAIN and SUB5)
RE,%SUB2         ,,routines in %SUB2 (SUB2 and SUB3)
*
*
M.1             ,,,The following are in node M.1
RE,%SUB1         ,,routines in %SUB1 (SUB1)
RE,%SUB11        ,,routines in %SUB11 (SUB11)
NA,SUB9          ,,SUB9 is in one of the libraries and is
*                ,   loaded with node M.1
SY,FTIME         ,,FTIME is in the system library and is
*                ,   loaded with node M.1
*
M.1.1           ,,,The following are in node M.1.1
RE,%SUB6         ,,routines in %SUB6 (SUB6 and SUB10)
*
*
M.1.2           ,,,The following are in node M.1.2
RE,%SUB7         ,,routines in %SUB7 (SUB7)
*
*
D.2             ,,,The following are in node D.2
RE,%SUB4         ,,routines in %SUB4 (SUB4)
SY,SYCON         ,,SYCON is in the system library and is
*                ,   loaded with node D.2
*
EN              ,,,end of the load
```

Figure 3-8. Loader Command File for Sample Program

Sample Trace of MLLDR Processing

The sample below shows the processing performed by MLLDR for the sample program in Figures 3-5 through 3-8. An understanding of this processing will help in creating command files faster and with fewer errors.

The loader performs preliminary syntax checking on the command file before the load process begins. This syntax checking is described in Chapter 4 in the Error Reporting section.

The loader, using the command file shown in Figure 3-8, would relocate the program as follows:

1. Relocate MAIN and SUB5 since they are in the file %MAIN.
2. Relocate SUB2 and SUB3 since they are in %SUB2.
3. Search one level down to resolve undefined external references. Mark any entry points found. When searching one level down, the loader can find an entry point that is specified in a SY or NA command, and it will detect an entry point in a file specified in an RE command. It does not detect that an entry point will be resolved by a search type command (SE, SL, or MS). When loading the root node M, M.1 and D.1 are searched, and SUB1 and SUB4 are marked.
4. Search the user defined libraries to resolve undefined external references which were not marked in step 3 (SUB8, when loading the root node).
5. If anything loaded, repeat step 4 to resolve backward references.
6. Search the system library and resolve undefined external references which were not marked in step 3.
7. If anything loaded, repeat step 6 to resolve backward references.
8. If any unmarked external reference exists, issue a warning message and prompt for a command. If a command is entered, process that command, repeat this step, and prompt for another command. Legal commands are SE, MS, SL, LI, FO, DI, AB, and /A. The commands "RE,<namr>", "TR,<namr>", /E, EN, EX, M, and D are illegal. When a TR command (with no <namr>) is entered, continue processing the command file.

Multilevel Segmentation Load-On-Call

9. Begin loading node M.1.
10. Relocate SUB1 and SUB11 (files %SUB1 and %SUB11).
11. Mark SUB9 in the symbol table as undefined, to be loaded with node M.1. (SUB9 will be loaded in step 4.)
12. Mark FTIME in the symbol table as undefined, to be loaded with node M.1. (FTIME will be loaded in step 6.)
13. Repeat steps 3 through 8.
14. Begin loading node M.1.1. Relocate SUB6 and SUB10, from file %SUB6.
15. Repeat steps 3 through 8.
16. Load node M.1.2 using the same relocation base as M.1.1. Relocate routine SUB7, from file %SUB7.
17. Repeat steps 3 through 8.
18. Load node D.2 using the same relocation base as M.1. Relocate SUB4, from file %SUB4.
19. Search the system library for SYCON and load it.
20. Repeat steps 3 through 8.

At step 8, unmarked undefined externals remaining after the system and user libraries are searched will cause MLLDR execution to be suspended, and the following messages will be displayed:

```
/MLLDR:UNDEFINED EXTERNALS  
/MLLDR:<list of the node's undefined externals>  
/MLLDR:W-UN EXT  
/MLLDR:
```

Multilevel Segmentation Load-On-Call

A command can be entered in response to the prompt. Note: If the undefined external was specified in an NA command, then the loader will only load the module from a user-defined library. A RE or SE will not load the module. Use the LI command for the file containing the module that will satisfy the undefined external, then use the SL command to load the module.

If the terminal times out, MLLDR will repeat the prompt. MLLDR will repeat the prompt up to 5 times, and then abort if there is still no response.

The force (FO) command can be entered to force load the code and prevent an abort if the undefined externals cannot be satisfied. If the force option was already in effect, then MLLDR would report the undefined externals as above, but would not prompt for commands.

Segmentation Structure Restrictions

This section describes restrictions on how a program can be segmented in MLS format, and guidelines that are useful when segmenting programs.

Load-on-Call

A routine on one level cannot call a subroutine on the same level if it is not in the same node. A routine cannot call a subroutine more than one level down a path. It can call any subroutine up the tree in any node on its path back to the root.

A node can only access a symbol (entry point) in a son node with a JSB instruction (subroutine call). Any other instruction will cause the error L-IL MLS at load time. The SGMTR program detects if a routine contains a symbol that is referenced as data (with a non-JSB instruction), and marks the routine as nonduplicatable. In the resulting MLS command file, the module is placed in a node above all modules that reference it. Refer to the section "Duplication of Modules", below.

MLS Tree Restrictions

Maximum Tree Size and Path Length

The maximum number of nodes in a program is 1024. The maximum number of nodes in any path to a leaf node is 31, including the root node. The maximum length of code in any path from the root to a leaf node is 31K words. This corresponds to the Extended Background program type, which allows the largest possible amount of logical address space for the program's code and data. Since a memory-resident node must start on a page boundary, the minimum size for a memory-resident node is one page.

Mixed Disc-Resident and Memory-Resident Nodes

If disc-resident nodes are mixed with memory-resident nodes, the structure of the tree must be defined as follows. All disc-resident nodes must be on the right hand side of the tree and the disc-resident nodes must start at level one. No memory-resident nodes can be further to the right of the disc-resident nodes and no memory-resident nodes can be descendents of disc-resident nodes.

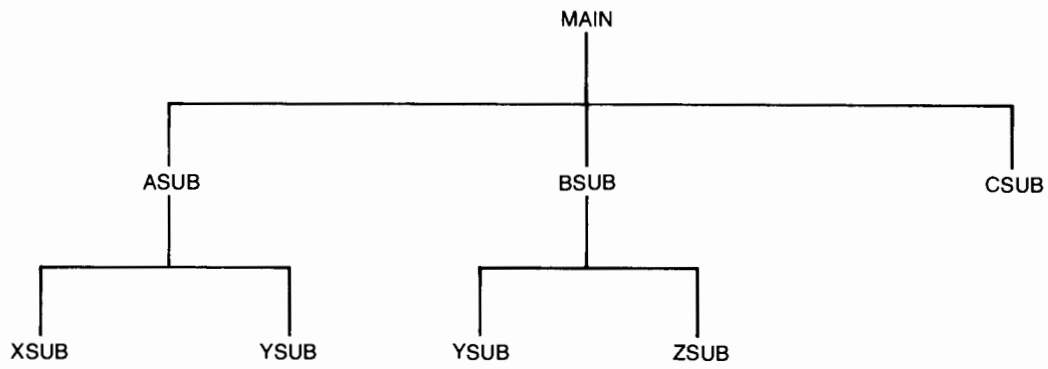
Valid ordering of disc-resident and memory-resident nodes can also be described in terms of the order in which they appear in the MLS command file. All M commands must come first, followed by all D commands. Note that this ordering of commands in the command file is the preorder of the nodes in the MLS tree (see Figure 3-3).

Duplication of Modules

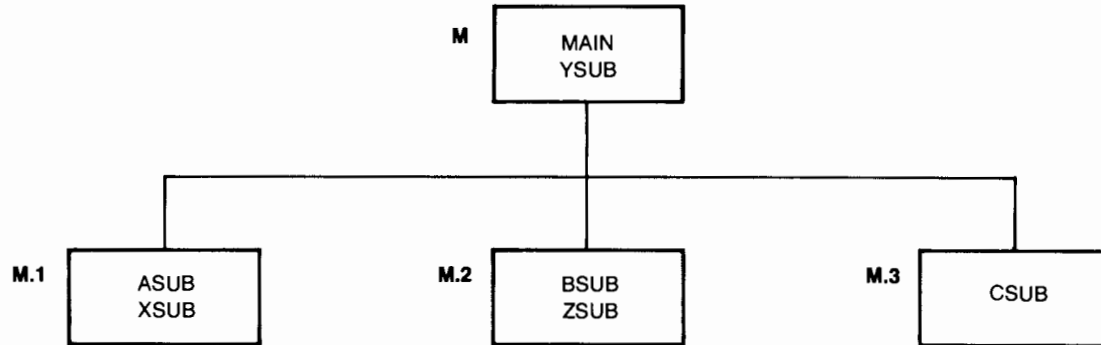
In some cases it is useful to duplicate modules. Consider the calling sequence in Figure 3-9A. Figure 3-9B shows one possible node tree. If the path to M.3 (CSUB) is too long (for example, greater than 31K words), then YSUB could be removed from the main node M, and copies of it placed in nodes M.1 and M.2, as shown in Figure 3-9C. Depending on the size of YSUB and the page alignment of the nodes, this could solve the pathlength problem for node M.3.

Note: Since node M.3 is memory-resident (memory nodes always begin on page boundaries) removing YSUB from the root would decrease the pathlength to M.3 only if it eliminated a page boundary crossing in the root node. Moving YSUB into M.1 and M.2 does not necessarily increase the pathlength to either of these nodes, since YSUB was already in these paths when YSUB was in the main. Moving YSUB, however, could change the pathlengths to M.1 and M.2 depending on the page alignment of the nodes. If, for example, YSUB extends node M.1 beyond a page boundary, then the path to M.1 would be one page larger.

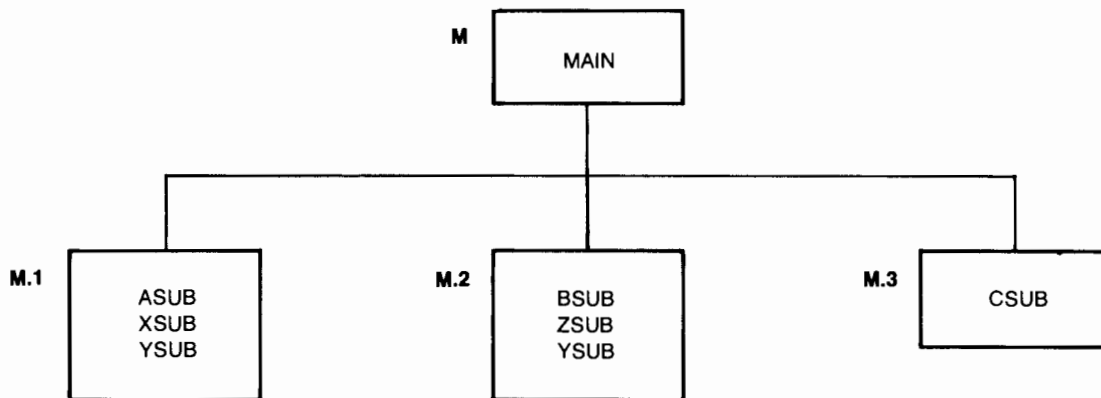
Multilevel Segmentation Load-On-Call



A. Calling sequence



B. YSUB not duplicated



C. YSUB duplicated

8100-17

Figure 3-9. Duplication of Modules

Multilevel Segmentation Load-On-Call

Only under certain circumstances can a module be duplicated. The SGMTR program places one restriction on duplicating modules: if an entry point in a module is referenced as data (by any instruction other than a JSB), then the SGMTR program will not duplicate the module. If, for example, the module contains data that is accessed by one or more subroutines, then all of these subroutines must access the same copy of the module.

Depending on the program, however, there may be other restrictions on duplicating modules. For a routine that depends on local data being preserved from a previous call, for example, only one copy of the module can exist in the MLS tree. In this case there are also considerations regarding disc-resident vs. memory-resident segments. Refer to the section "Data Considerations", below.

Offpath References

An offpath reference occurs when a routine makes a call to a subroutine on the path to the root, and the called subroutine makes a call to another routine down a different path. Execution then cannot resume in the original routine. The result of an attempt to return to the original routine is unpredictable. This is because when the offpath call is made, a new node is mapped in over the logical address space where the original call occurred. When control is passed back to the return address, the original code is no longer mapped in. Runtime errors caused by offpath references are difficult to diagnose.

An offpath reference error can occur when a program that uses recursion or a circular call chain is loaded incorrectly. Figure 3-10A shows a circular calling sequence. Figure 3-10B shows a valid node tree; no offpath references can occur. Figure 3-10C shows an invalid node tree. One calling sequence, for example, that would cause an offpath reference is "A calls B calls C calls A calls x".

The SGMTR utility detects potential offpath references and issues diagnostic information.

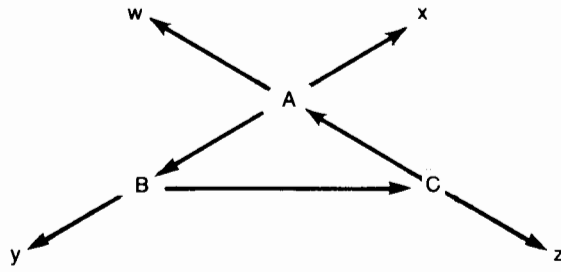
Circular Call Chains and Recursion

Programs that use recursion or circular call chains must be loaded so that no offpath references can occur during program execution. Offpath references will not occur if the recursive routines or routines in the circular call chain reside in the same path along with all their support modules. A routine's support modules are all modules referenced directly or indirectly by the routine.

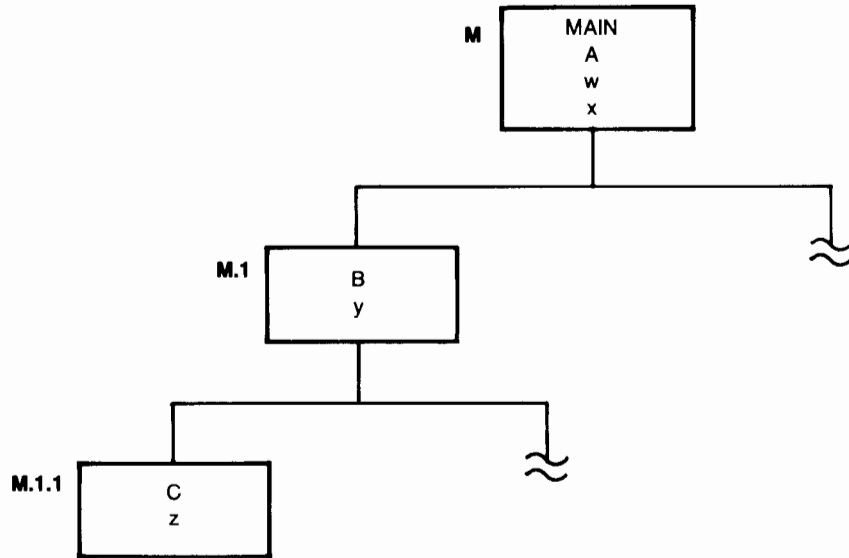
Figure 3-10A shows an example of a circular call chain. Routine A calls B, B calls C, and C calls A. The support modules are routines w, x, y, and z.

A valid MLS tree (one with no potential offpath references) is shown in Figure 3-10B. Figure 3-10C shows an invalid node tree. The routines x and y are not in the same path as the recursive routines. In the calling sequence "A calls B calls C calls A calls x", the call to x is an offpath reference, and execution could not return to routine C.

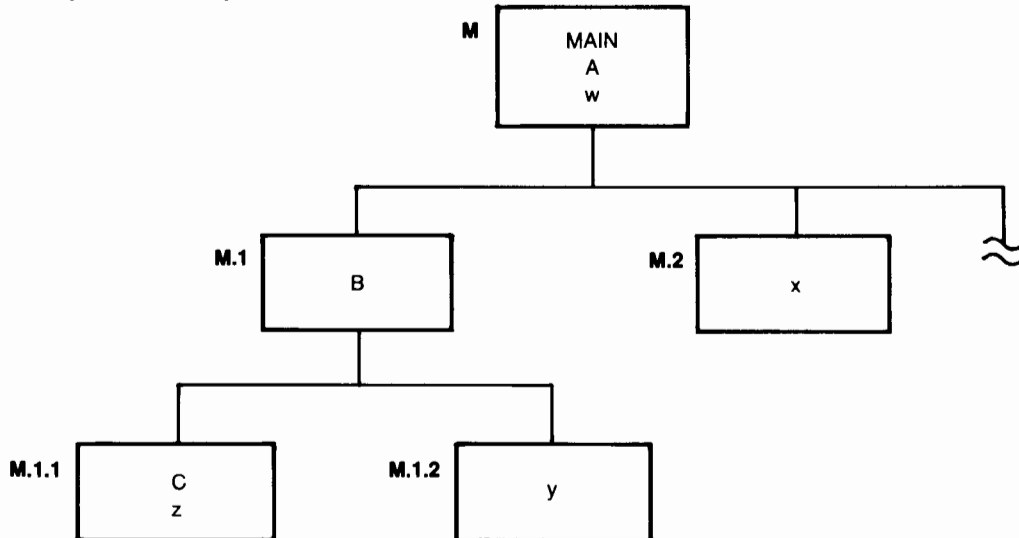
Multilevel Segmentation Load-On-Call



A. Circular calling sequence



B. No potential offpath references



C. Potential offpath references

8100-24

Figure 3-10. Offpath References

Data Considerations

In some cases there are requirements on where data can be placed in the MLS tree.

Disc Nodes - Data values are not preserved between loads of disc nodes, since a new copy of the node is brought in from disc. If data values must be preserved, place the data in a memory node, in EMA or VMA, or, in FORTRAN 77 or MACRO, declare the variable to be a SAVE variable.

This includes data modules, such as COMMON blocks, and subroutines that make use of data values from previous calls.

Duplicated Modules - For a data module that must show consistent values when accessed by more than one routine, the module containing the data must not be duplicated. The data module (a COMMON block, for example) must reside at the same level or higher in the tree than the routines that access it.

COMMON Initialization - A labeled COMMON block cannot be initialized by a block data subprogram below the node in which the COMMON block was declared.

Using SGMTR - When using SGMTR there are requirements for use of COMMON. Refer to Chapter 6.

Language Considerations

Pascal

Use the LIBRARY compiler directive to generate separate level-1 modules.

For recursion, refer to the section, "Circular Call Chains and Recursion", in this chapter.

For programs with memory nodes that use recursion or dynamic variables, specify at least one page of dynamic buffer space, using the "SZ,+n" command, for heap/stack initialization. For programs that do not use recursion or dynamic variables, compile the program using the compiler directives HEAP 0 and RECURSIVE OFF.

Multilevel Segmentation Load-On-Call

FORTRAN

If a subroutine's local data must be preserved between calls, the data must be contained in a memory node, declared to be a SAVE variable, or reside in EMA or VMA. Local data is not preserved between disc node loads.

Data modules, such as labeled COMMON blocks, must not be duplicated, and must be located at the same level or higher in the tree than all routines that reference them.

EMA transparency mode (in FORTRAN 77) is useful when routines are passed both EMA and non-EMA variables. It may be necessary to use this feature if part of the program's data has been moved to EMA to reduce the program's path size.

For information about requirements for use of COMMON when using SGMTR, refer to Chapter 6.

Chapter 4

Operation

This chapter describes how to schedule MLLDR and pass commands and parameters to it, how errors are reported and what to do about them, and how to interpret the information output by MLLDR during and after a program load.

Modes of Operation

MLLDR has three modes of operation, corresponding to the three methods that can be used to pass commands to it:

1. Runstring operation - parameters are passed through the runstring. (This method can be used in Batch mode.)
2. Interactive operation - commands are entered from an interactive device, usually a terminal.
3. Non-interactive operation - commands are supplied in a command file or from a noninteractive LU, such as a minicartridge tape unit. (This method can be used in batch mode.)

There are restrictions on which method can be used in certain situations. When loading an MLS program, for example, the node specification commands, starting with the root node M, must be supplied in a (disc) command file. Also, more than one method might be used in a particular program load. For example, the namr of the relocatable file that contains the main program can be included in the runstring. If all subroutines needed by the program are contained in this file and in the system relocatable library, then no further information is needed. If one or more routines needed by the program, however, are not contained in the specified file, MLLDR issues the message W-UN EXT on the terminal from which it was scheduled. It then prompts the user to enter commands interactively.

Loading from a Logical Unit

For non-segmented programs, relocatable code can be input using the "RU,MLLDR,,LU" runstring, or interactively with the "RE,LU" command. If more than one tape is to be mounted for the load, then the interactive mode must be used and the "RE,LU" command reentered for each tape. For MLS programs, an LU cannot be specified in the runstring input parameter or in RE commands.

Commands and Command File

MLLDR commands can be divided into four categories, as shown in Table 4-1.

Operation

Table 4-1. MLLDR Commands

Group 1: Set conditions for the load. Commands in this group must appear before the first M command in a segmented program, and before any relocate or search commands (commands in Group 2) in a non-segmented program.

- LL Set list namr
- OP Set opcode
 - Program type:
 - BG Background
 - RT Realtime
 - LB Large Background
 - EB Extended Background
 - COMMON type:
 - SC system
 - RC reverse system
 - NC local or no COMMON
 - SS Subsystem Global Area (SSGA)
 - Load type:
 - TE temporary
 - PE permanent
 - RP replacement
 - EMA type:
 - VM virtual memory
 - EM EMA
- FM Set format code
 - DB append debugger
 - LE list entry points and base page linkages
 - NL no listing
 - DC don't copy
 - Linkage type:
 - BP base page
 - CP current page
 - MP mixed
- AS Assign partition
- SZ Set program size
- PF Append profile routine, direct output to specified LU
- SH Set shareable EMA label
- SA Allocate SAVE area
- VS Set VMA size
- WS Set working set size
- LI Specify file as user library file. (In this group when used in a command file loading an MLS program. Can also be used when prompted to enter commands to satisfy undefined externals.)

Operation

Table 4-1. MLLDR Commands (continued)

Group 2: Relocation

RE Relocate code in namr.
LO Set relocation base.
SE Search namr or system library.
MS Multiple search of namr.
LI Specify file as user library file. (In this group when used interactively in response to the undefined EXTERNALS warning message when loading an MLS program. Also in this group when used interactively loading a non-segmented program.)
SL Search user library files.
M Memory-resident node specification.
D Disc-resident node specification.
NA Module is in a user library, load with current node.
SY Module is in the system library, load with current node.

Group 3: Miscellaneous

EC Echo commands to list device
DI Display undefined externals
FO Force load the remainder of the program; ignore undefined externals
TR Transfer to command file
* Comment

Group 4: Termination commands. A command in this group will be the last command to be processed in a load.

EN, EX, /E Finish processing the load
AB, /A Abort the load



Command File

A command file can be created and edited using EDIT/1000, or the segmenter program SGMTR can be used. Syntax for commands in a command file is identical to syntax for commands entered interactively. Refer to Chapter 5 for command syntax.

MLS Programs

A command file (disc file) is required for MLS programs because MLLDR must know what routines are located one level below the node currently being loaded. The command file must contain an M command before any command that causes relocation (relocate and search type commands), or an L-IL MLS error will occur. Commands before the first M command can be interactive, but a transfer must be made to a command file before the first M command.

Non-Segmented Programs

For non-segmented programs, an LU can be used for command entry, and an M command is not required. All commands can be entered interactively, and the code can be input from a file or LU using the RE command. If a relocatable file is specified in the runstring, MLLDR will load all relocatable code contained in the file. Note that commands entered interactively or from a command file will override commands specified in the runstring.

Runstring Operation

Runstring Parameters

Parameters can be included in the MLLDR runstring to specify the following:

1. Command file namr. This parameter must be an FMP file when loading an MLS program.
2. Relocatable code namr. File name of an input file for relocatable code. If the program is not segmented, this parameter can be an LU.
3. List destination namr.
4. An operation code (opcode) that allows Subsystem Global Area (SSGA) access and specification of program type, COMMON type, load type, and EMA type.
5. A format code that allows temporary loads with MLSDB appended.
6. Inclusion of a profiling option.
7. Listing characteristics.
8. Linking characteristics (current, base page, or mixed).

The loader is scheduled for execution with the RU or ON operator command in the format:

```
RU,MLLDR[,command[,input[,list[,opcode[,format[,ptn[,sz  
[profile]]]]]]]]
```

where:

command The command file parameter can be used for MLS programs, and for non-segmented programs when more than one relocatable file is required. The command parameter specifies one of the following:

1. For both MLS programs and non-segmented programs, a command file namr. A file descriptor (disc file) is required for multilevel segmentation specification.

Operation

2. For non-segmented programs, an interactive input device (terminal LU) from which commands are to be entered. When commands are entered interactively on such a device, the /MLLDR: prompt is displayed when the loader is ready for a new command.
3. For non-segmented programs, a non-interactive input device (LU), such as a tape cassette, from which commands are to be entered. If undefined externals remain after the commands have been processed, the loader issues a prompt for commands at the user's terminal.

If this and all other parameters are omitted, command entry defaults to the Logical Unit number of the user's terminal. If running under Batch Mode, the default is logical unit 5.

input The namr of the main program's relocatable code. there is no default for this parameter. A Logical Unit number for the relocatable code is only allowed for non-segmented programs.

list List namr. This corresponds to the interactive "LL,namr" command. The default setting is the user's terminal. In batch mode, the default is Logical Unit 6. If the list device is a non-interactive device, it is locked for the duration of the load. If the device is already locked to another program, the loader displays the message WAITING FOR LIST DEVICE. The loader repeats the LU lock request and is suspended until the device becomes available.

If the list namr is a file, the file must not already exist; the loader must create the file. The only exception to this is if the specified file name has an apostrophe (') as its first character. For example:

'name

In this case, the loader will create the file if it does not exist, or open the file if it does exist.

Refer to the format parameter below for list options.

Operation

opcode Mnemonic operation code. This corresponds to the "OP,opcode" command. Up to three mnemonics can be specified in the opcode parameter in any order with no intervening commas or blanks. If more than three mnemonics are to be used, they can also be specified in the format parameter. A later specification will override an earlier specification. The default is EBNCTE. The parameter defines the program type, COMMON type, load type, whether or not the program requires the Subsystem Global Area (SSGA), EMA or VMA. Note that the program type does not default to the type that is kept in the NAM record, but to type 6 (EB). Opcode mnemonics are:

Program Type	COMMON Type	Load Type	EMA Type
BG	SC	PE	VM
RT	RC	TE (default)	EM
LB	NC (default)	RP	
EB (default)	SS		

where:

BG = Background program
RT = Realtime program
LB = Large Background program
EB = Extended Background program

SC = System COMMON
RC = Reverse system COMMON
NC = No COMMON (or local COMMON)
SS = Provide access to Subsystem Global Area (SSGA).

PE = Permanent program
TE = Temporary program
RP = Replace permanent program (do not also specify PE).

NOTE

Only MLLDR can perform permanent loads or purges of multi-level programs. Use the runstring "RU,MLLDR:IH" to inhibit copying of the loader. A copy of the loader can perform temporary loads, but will be aborted with an L-PE LDR error if an attempt is made to perform a permanent load, purge, or replacement operation.

Operation

VM = Virtual memory program with default VMA environment: working set size = 31 pages, VMA size = 8192 pages. If the EMA size in the relocatable records is greater than 8192, the VMA size is set to the EMA size. If the EMA size is less than 31, the working set size is set to the EMA size.

EM = EMA program. The EM command must be used when loading a multilevel program if the root does not declare EMA but another node does.

format Mnemonic format code. This corresponds to the command "FM,format code". Up to three mnemonics can be specified in any order with no intervening commas or blanks. The default is CP. The format parameter defines the format for the program load operation. In the runstring, this parameter also serves as an extension of the opcode parameter; opcode mnemonics can be specified in the format parameter. The opcode and format parameter mnemonics can be intermixed in any order. A comma must be included as a parameter position marker if the character count within the opcode parameter exceeds six, or subsequent parameters, such as partition, size, or profile are to be specified. Format mnemonics are:

Append Debugger	List Options	Don't Copy	Linking Options
DB	LE NL	DC	MP CP (default) BP

where:

DB = Append the MLSDB subroutine to the program.

LE = Include listing of entry points and base page linkages in the program's loadmap.

NL = Do not output a loadmap.

DC = Don't copy. Copies of this program will not be made. This option should be specified when multiple copies of the program are not desired.

Operation

MP = Mixed page links. Use current page links where possible, except for external references. EXT linkages are forced to base page.

CP = Use current page links where possible, including external references. External references do not use links when possible. The default linking method is CP.

BP = Use base page links only. No current page links are used. External references are forced to use base page links.

partition (ptn) The number of the partition in which the program is to be executed. This corresponds to the "AS,ptn" command. If not specified, the program will execute in any available partition of sufficient size.

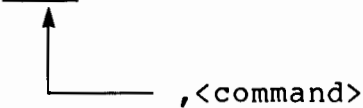
size (sz) Allows a logical address space larger than the program size. This corresponds to the loader "SZ,size" command. This permits use of a dynamic buffer at the end of the program for use as a data array, symbol table space, etc., if required. If the program is an EMA program, the EMA area immediately follows the dynamic buffer area.

profile The logical unit number to which profile output is to be printed. If not specified, no profile is performed.

Runstring Operation Examples

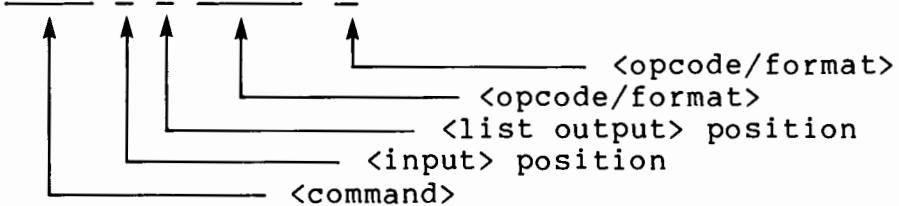
The following examples show use of runstring parameters. Note that all information specified in the runstring can alternately be specified in a command file.

```
:RU,MLLDR,#CMD
```



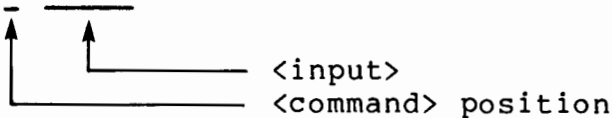
Comments: All commands for the load are contained in the command file. The user is prompted to enter additional commands if undefined externals exist after a node is loaded.

```
:RU,MLLDR,#PROG, , ,RTDBSS,NL
```



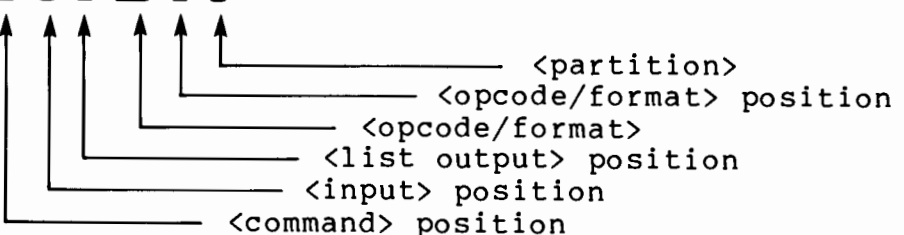
Comments: The user is prompted at the terminal for commands only if undefined externals exist after a node is loaded.

```
:RU,MLLDR, ,%PROG
```



Comments: All relocatable code in %PROG is relocated. The system library is searched and the user is prompted for commands if undefined externals remain. This program must be non-segmented since relocation cannot occur before node specification commands.

```
:RU,MLLDR:IH, , , ,RP, ,7
```



Operation

Comments: The program will replace a permanently loaded program and will be assigned to partition 7. The user will be prompted at the terminal for commands. For a non-segmented program, all commands can be entered interactively. For a segmented program, refer to Table 4-1 for a list of commands that can be entered interactively (commands in Group 1) before transferring to the FMP file containing the node specification commands.

```
:RU,MLLDR:IH, ,%PROG, ,PE
```

<opcode/format>
<list output> position
<input>
<command> position

Comments: A non-segmented program will be loaded permanently. The user is prompted for commands if undefined externals exist after the system library is searched.

```
:RU,MLLDR,#PROG, ,'PROG,LEDB, , , ,6
```

<profile>
<size> position
<partition> position
<opcode/format> position
<opcode/format>
<list output>
<input> position
<command>

Comments: Load the program with debugger appended, and list entry points and base page linkages in the loadmap file. Output the loadmap to the file 'PROG, and profile information about program execution to LU 6.

Error Reporting

Error codes and descriptions are contained in Appendix A.

All loader errors are reported to the list device. If the loader was initiated from an interactive device which is different than the list device, all errors are also reported to the interactive device. The list device can be specified in the runstring, interactively with the LL command, or it can be defaulted. The default list device is the user's terminal. If running under Batch, the default is LU 6.

Error codes are displayed on the list device in the following form:

```
/MLLDR:<error code>
```

All error codes are prefixed by an L character. All warning codes are prefixed by a W character.

Undefined Externals

During the load process, undefined externals may exist after loading a node or non-segmented program. At the end of each node, all nodes one level down are searched, and all user-specified libraries and the system library are searched. If undefined externals still exist, the following warning messages are issued and the user is prompted to enter commands:

```
/MLLDR: UNDEFINED EXT  
/MLLDR: <list of the undefined externals>  
/MLLDR: W-UN EXT  
/MLLDR
```

When running under Batch mode, the following is printed after the list of the undefined externals and the load is aborted:

```
/MLLDR: L-UN EXT  
/MLLDR: ABORTED
```

During normal operation (force option not in effect, and not running under Batch) the loader suspends and issues a prompt. If the interactive device times out five times after prompts are issued, the loader will abort. The suspend and prompt allow the user to enter commands to satisfy the undefined externals.

Operation

The interactive device will then be prompted at the completion of each command until the TR command is entered. The display (DI) command can be used to list undefined externals. Note that for MLS programs, the list only shows undefined externals for the current node. A search type command (SE, MS, or SL) can be entered to satisfy the undefined externals except in the case described in the note below.

The TR command (with no namr parameter) transfers control back to the command file (or LU) which was being processed when the undefined external occurred. MLLDR resumes processing with the command following the command that caused the loader to go interactive. A "TR,namr" command is not legal in this mode, nor is an "RE,namr", any M command, or any D command. When loading an MLS program, /E, EX, and EN are also not legal at this point. Legal commands are SE, SL, MS, LI, FO, DI, AB, and /A.

Note about search type (SE, MS, and SL) commands: Searches only satisfy undefined externals for the current node. A search in one node will not satisfy undefined externals of a previous node; searches are "local" to the node currently being constructed. If a node expects an undefined external to be satisfied in a son node, then the son node must have a relocate of the module containing the entry point, or an NA or SY command containing the entry point name.

Note: If an undefined external occurred in an NA command, an SE of the file containing the external will not cause the external to be loaded. To satisfy the undefined external, use the LI command to specify that the appropriate file is a library file, and then use SL to search libraries. This must be done since the NA command indicates that the load was to come from a library.

If one or more undefined externals in a node cannot be satisfied, the abort can be prevented by using the force (FO) command. When the force option is in effect, MLLDR will complete processing the program, and the undefined externals will not be satisfied. The result is unpredictable if an unresolved entry point is accessed during program execution.

MLS Errors

The loader performs the following preliminary checks on the command file before beginning the load process:

1. Command syntax is correct.
2. Command order is correct.

Operation

3. Referenced files are accessible.
4. Nodes are defined in preorder.
5. Disc-resident node definitions follow memory-resident node definitions.
6. The first disc-resident node definition occurs as a son of the root node.
7. The root node is a memory-resident node.
8. The total number of nodes is no more than 1024.
9. The maximum node depth is no more than 31.

If the loader detects an error during preliminary command file checking, it gives the number of the command line that contains the error:

```
/MLLDR: ERROR IN COMMAND LINE # <line #>  
/MLLDR: <error code>
```

Preliminary syntax checking can result in the following errors:

```
L-IL DRN   Disc-resident node out of order.  
  
L-NO RSG   No root node found or relocation occurs before  
           root specification.  
  
L-IL MLS   This error is issued for all other errors detected  
           during the preliminary checks listed above, and in  
           the case described below.
```

An L-IL MLS error also occurs when an instruction other than a JSB (a subroutine call) is used to access a symbol in a son node. This error occurs during the load, not during preliminary command file checking. In this case, the following information is given:

```
/MLLDR: <16-bit non-JSB instruction>  
/MLLDR: <address of non-JSB instruction>  
/MLLDR: <entry point name being referenced>  
/MLLDR: L-IL MLS
```

Other Errors

For some error codes, such as L-DU ENT and L-SS ENT, the name of the program module and the entry point name of the subroutine being relocated are displayed prior to the error code display line, as follows:

```
/MLLDR:<module name>  
/MLLDR:<entry point name>  
/MLLDR:<error code>
```

For some non-recoverable error conditions, MLLDR aborts execution after displaying the error report as follows:

```
/MLLDR:<error code>  
/MLLDR:MLLDR ABORTED
```

Warning and Information Reporting

For error conditions from which recovery is possible, MLLDR issues warning messages. Warning messages are prefixed by a W character. The possible warnings are:

W-UN EXT	Undefined externals (described in the previous section).
W-DU PGM	Duplicate program (see below).
W-IL CMD	Illegal command.
W-RQ PGS	Program too large for partition.
W-IN CAP	Insufficient user capability.
W-SV MIX	Mixing SAVE and non-SAVE.
W-WS EMA	Working Set size too large.

For more detailed information, refer to Appendix A, Errors and Warnings. For information about recovering from the undefined externals condition, refer to the section, "Error Reporting", in this chapter.

Operation

Duplicate Program Name

If a new program is loaded with the same name as a main program or a program segment that already has an ID segment, the following message is displayed:

```
/MLLDR: DUPLICATE PROG NAME -<nnnnn>  
/MLLDR: W-DU PGM
```

where nnnnn is the name of the program that already exists. The loader automatically attempts to create a unique name for the new program by replacing the first two characters of the new program's name with period characters (..). If successful, the load process continues, and when the load is completed, the following messages are displayed:

```
/MLLDR: <..nnn> READY AT <time and date>  
/MLLDR: $END
```

where ..nnn is the modified program name.

If a program named ..nnn already exists, the loader aborts after the following messages are displayed:

```
/MLLDR: DUPLICATE PROG NAME -<nnnnn>  
/MLLDR: W-DU PGM  
/MLLDR: DUPLICATE PROG NAME -<..nnn>  
/MLLDR: L-DU PGM  
/MLLDR: MLLDR ABORTED
```

One of the existing programs must be removed using the OF command before the new program can be loaded.

Waiting for Resources

The loader issues these information messages under the circumstances described:

WAITING FOR DISC SPACE - The loader is waiting for a system disc track allocation request. The loader goes into state 5 until enough tracks are available.

WAITING FOR LIST DEVICE - The list device is locked to another program. The loader is waiting on an LU lock request. It goes into state 3 until the LU lock is removed.

Loadmap

Sample loadmaps are shown in Figures 4-1 through 4-3 for the default listing, listing with commands echoed, and with entry points and base page links. The source and command file for the program used in the examples are contained in Appendix D. Comments have been removed from the command file used in these examples.

The program output shown in this section is a sample only. Output on various systems will differ according to system configuration and current software revision.

Default

Figure 4-1 shows a sample default MLLDR loadmap. At the start of each node the node name and the ordinal (preorder) node number are output. For each module the module name, its logical start and end addresses, and comment information from the NAM record are output. The address space in which the module was loaded is shown in the form:

```
<module name>      xxxxx - yyyyy
```

The start address, in octal, is xxxxx. The end address is yyyyy. The first address shown in the loadmap is referred to as the program's "load point". For any module, if the end address is lower than the start address (for example, "2012 - 2011"), the module was empty. If there is space between the end of one module and the start of the next module, the space has been set aside by the loader for current page links. For example,

```
ROUTINE1           10000 10200
ROUTINE2           10210 10300
```

shows that the locations between 10200 and 10210 are available for current page links. The amount of space that will be needed for current page links for the module is estimated by the loader before loading the module.

COMMON and SAVE areas are reported as shown below:

Operation



For local blank COMMON, the following is output at the start of the main or root node:

```
COM          xxxxx yyyyy
```

For the SAVE area specified in the SAVE command, the following is output after local blank COMMON, if used:

```
SAVE          xxxxx yyyyy
```

When a local SAVE area is allocated, it is reported as follows:

```
LOCAL SAVE AREA xxxxx yyyyy
```

Named COMMON (when declared with the default ALLOCATE method) and SAVE named COMMON are listed when loaded:

```
NAMED COMMON    xxxxx yyyyy
SAVE NAMED COM  xxxxx yyyyy

M
NODE           0

MAIN           2012  3026  EXAMPLE: CHAPTER 4, APPENDIX D

$LOC$         3027  3100  92084-1X415 REV.2121 810723
.RRGR         3101  3135  92084-1X419 REV.2121 810723

D.1
NODE           1

SUB1           3162  3476

D.1.1
NODE           2

SUB6           3521  4347

D.1.1.1
NODE           3

SUB9           4372  4706

FTIME         4707  5200  92084-1X032 REV.2121 780731
```

Figure 4-1. Loadmap: Default

Operation

D.1.1.2							
NODE	4						
SUB11		4372	4703				
D.1.2							
NODE	5						
SUB7		3521	4034				
D.1.2.1							
NODE	6						
SUB9		4057	4373				
FTIME		4374	4665	92084-1X032	REV.2121	780731	
D.1.2.2							
NODE	7						
SUB11		4057	4370				
D.2							
NODE	8						
SUB2		3162	4006				
SUB4		4007	4326				
SUB8		4327	4330				
SYCON		4331	4347	92084-1X047	REV.2121	780921	

3 PAGE LONGEST PATH
3 PAGE PARTITION REQUIRED

LINKS:BP PROGRAM:EB LOAD:TE COMMON:NC

/MLLDR:MAIN READY AT 2:54 PM SAT., 24 OCT., 1981
/MLLDR:\$END

Figure 4-1. Loadmap: Default (Cont.)

EC Option

Figure 4-2 shows the same loadmap as Figure 4-1, except the EC (echo commands) option has been turned on. Note: For a particular node, the next M or D command indicates to MLLDR the end of the current node. This command is echoed to the listing, followed by other information for the current node.

```

LI, $EXMPL
OP, BP
M
M
NODE          0

NA, MAIN
NA, SUB5
NA, TBUFF
SY, $LOC$
SY, $LOD$
SY, .RRGR
SY, .SVRG
D.1
MAIN          2012  3026  EXAMPLE: CHAPTER 4, APPENDIX D
$LOC$        3027  3100  92084-1X415 REV.2121 810723
.RRGR        3101  3135  92084-1X419 REV.2121 810723

D.1
NODE          1

NA, SUB1
D.1.1
SUB1          3162  3476

D.1.1
NODE          2

NA, SUB10
NA, SUB6
D.1.1.1
SUB6          3521  4347

```

Figure 4-2. Loadmap: EC Option

Operation

D.1.1.1 NODE	3					
NA,SUB9 D.1.1.2 SUB9		4372	4706			
FTIME		4707	5200	92084-1X032	REV.2121	780731
D.1.1.2 NODE	4					
NA,SUB11 D.1.2 SUB11		4372	4703			
D.1.2 NODE	5					
NA,SUB7 D.1.2.1 SUB7		3521	4034			
D.1.2.1 NODE	6					
NA,SUB9 D.1.2.2 SUB9		4057	4373			
FTIME		4374	4665	92084-1X032	REV.2121	780731
D.1.2.2 NODE	7					
NA,SUB11 D.2 SUB11		4057	4370			

Figure 4-2. Loadmap: EC Option (Cont.)

Operation

```
D.2
NODE      8

NA,SUB2
NA,SUB3
NA,SUB4
END
SUB2      3162  4006
SUB4      4007  4326
SUB8      4327  4330

SYCON      4331  4347  92084-1X047 REV.2121 780921
```

```
3 PAGE LONGEST PATH
3 PAGE PARTITION REQUIRED
```

```
LINKS:BP   PROGRAM:EB   LOAD:TE   COMMON:NC
```

```
/MLLDR:MAIN   READY AT  2:56 PM   SAT., 24   OCT., 1981
/MLLDR:$END
```

Figure 4-2. Loadmap: EC Option (Cont.)

LE Option

The LE option lists entry points and locations of base page links. Figure 4-3 shows the same loadmap as Figure 4-1, but with the LE option on.

After all modules in a node are loaded, entry points are listed with their locations.

After each module, the location of base page links used by the module is reported as follows:

```
BP LINKAGE  xxxx - yyyy
```

where xxxx and yyyy are starting and ending base page locations in octal. The base page link area begins at location 2. If the ending address for a module is less than the starting address (for example "2 - 1") no base page links have been used. Note: A node may use base page links for the load-on-call mechanism. These links are not explicitly reported on the loadmap.

Shared Base Page

Adjacent brother leaf nodes can share the same base page. In the sample loadmap (Figure 4-3) nodes D.1.1.1 and D.1.1.2 are adjacent brother leaf nodes. The path to D.1.1 uses base page links 2 - 21. (Locations 17 - 21 were allocated for the load-on-call mechanism. They are not reported in node D.1.1 of the loadmap.) Node D.1.1.1 uses one link at location 22 (the links for the path are reported 2 - 22). Node D.1.1.2 is then loaded. It uses the links in the path to D.1.1 (2 - 21), but new links for node D.1.1.2 start at location 23, after the links used by D.1.1.1.

Rotating Base Page

A base page cannot be shared when the next node is not a leaf node that is a brother of the current node, or when the next node is a brother leaf node but there are not enough links remaining for the next node. In this case the base page is "rotated". This consists of creating a new base page for the next node. All links in the path to the new node are included, and allocation of new links starts after those links. When a new base page is created for a node, it is written out (to disc) with the node. During program execution, when the node is loaded the base page is brought in with the node and overlays the current base page.

Operation

In the loadmap in Figure 4-3, for example, suppose there were not enough links remaining for node D.1.1.2. A new base page would be created for that node. It would contain the links in the path to D.1.1 (2 - 21). No links beyond location 21 in the old base page are needed in the new base page since they are used by different paths. New links, for node D.1.1.2, are created starting at location 22.

```

M
NODE          0

MAIN          2012  3026  EXAMPLE: CHAPTER 4, APPENDIX D
  BP LINKAGE  2 -    1

$LOC$        3027  3100  92084-1X415 REV.2121 810723
  BP LINKAGE  2 -    1

.RRGR        3101  3135  92084-1X419 REV.2121 810723
  BP LINKAGE  2 -    3

ENTRY POINTS

*.CNOD              2011
*$LOC$              3027
*MAIN                2012
*SUB5                3025
*TBUFF              3006
*$L0D$              3071
*.RRGR              3122
*.SVRG              3101
*EXEC                1606
*$LOC                105241

D.1
NODE          1

SUB1          3162  3476
  BP LINKAGE  2 -   12

ENTRY POINTS

*SUB1              3162
  
```

Figure 4-3. Loadmap: LE Option

Operation

D.1.1
NODE 2

SUB6 3521 4347
BP LINKAGE 2 - 16

ENTRY POINTS

*SUB6 3521
*SUB10 4036

D.1.1.1
NODE 3

SUB9 4372 4706
BP LINKAGE 2 - 21

FTIME 4707 5200 92084-1X032 REV.2121 780731
BP LINKAGE 2 - 22

ENTRY POINTS

*SUB9 4372
*FTIME 4722

D.1.1.2
NODE 4

SUB11 4372 4703
BP LINKAGE 2 - 21 23 - 23

ENTRY POINTS

*SUB11 4372

Figure 4-3. Loadmap: LE Option (Cont.)

Operation

D.1.2

NODE 5

SUB7 3521 4034
BP LINKAGE 2 - 15

ENTRY POINTS

*SUB7 3521

D.1.2.1

NODE 6

SUB9 4057 4373
BP LINKAGE 2 - 20

FTIME 4374 4665 92084-1X032 REV.2121 780731
BP LINKAGE 2 - 21

ENTRY POINTS

*SUB9 4057
*FTIME 4407

D.1.2.2

NODE 7

SUB11 4057 4370
BP LINKAGE 2 - 20 22 - 22

ENTRY POINTS

*SUB11 4057

Figure 4-3. Loadmap: LE Option (Cont.)

Operation

```
D.2
NODE      8

SUB2      3162  4006
  BP LINKAGE  2 -  12

SUB4      4007  4326
  BP LINKAGE  2 -  14

SUB8      4327  4330
  BP LINKAGE  2 -  15

SYCON     4331  4347  92084-1X047 REV.2121 780921
  BP LINKAGE  2 -  16

ENTRY POINTS

*SUB2      3162
*SUB3      3474
*SUB4      4007
*SUB8      4327
*SYCON     4333
*.ENTR     105223
*XLUEX     1607
```

3 PAGE LONGEST PATH
3 PAGE PARTITION REQUIRED

```
LINKS:BP    PROGRAM:EB    LOAD:TE    COMMON:NC

/MLLDR:MAIN  READY AT  2:58 PM  SAT., 24  OCT., 1981
/MLLDR:SEND
```

Figure 4-3. Loadmap: LE Option (Cont)

Completion Message

Following the successful relocation of a program, the loader terminates with messages shown below according to whether the program uses EMA, VMA, or neither.

The format of the completion message is as follows:

For all programs,

```
ww PAGE LONGEST PATH
xx PAGE PARTITION REQUIRED
```

followed by,

For EMA programs:

```
DEFAULT EMA or yy PAGE EMA
zz PAGE MSEG
```

For VMA programs:

```
yy LAST PAGE OF VMA
zz PAGE MSEG
nn PAGE WORKING SET
```

followed by:

```
LINKS:pp PROGRAM:ss LOAD:tt COMMON:uuuu vv qq SHARE:rr
/MLLDR:name READY AT HR:MIN AM/PM day, date, year

/MLLDR:$END
```

where:

- ww = The number of pages occupied by the longest path in the program (includes base page).
- xx = Size in pages of the partition required by the program.
- yy = For EMA programs, the EMA size in pages. For VMA programs, the last page of VMA (VMA size in pages is yy + 1).
- zz = For EMA or VMA programs, the MSEG size in pages.
- nn = For VMA programs only, the working set size in pages.

Operation

pp = CP, MP, or BP for current page, mixed page, or base page, depending on which option was used. Default is CP.

ss = Program type, RT, BG, LB, or EB. Default is EB.

tt = TE for temporary program. PE for permanent program. RP for replacing a permanent program. Default is TE.

uuuu = SC, RC, or NC for system COMMON, reverse system COMMON, or no system COMMON, respectively, followed by SS for Subsystem Global Area (SSGA) access or blanks if SSGA is not used.

vv = DB if DEBUG appended, PF if profile option requested.

qq = FO if force load, blank if not.

rr = Label identifying shareable EMA area. If shareable EMA is not used, SHARE:rr will not be printed.

name = Name of the main program, which is ready to run when the loader terminates.

Profile Output

After the program terminates, profile information is output on the specified LU as shown below. For each node the following message is printed:

```
NODE# <node number> FAULTED <number of faults> TIMES
```

Return Parameters for Programmatic Scheduling

When the loader completes a load or terminates unsuccessfully, it returns five words of information about the load to the program that scheduled it. MLLDR returns the information to the scheduling program's ID segment via the PRTN system subroutine. The information can then be obtained using the system subroutine RMPAR.

When the loader is run from File Manager, for example, FMGR picks up the information in parameters 1P, 2P, 3P (this is also the FMGR 10G), 4P, and 5P. A successful load results in the following:

```
1P,2P,3P = program name  
4P,5P = 4 spaces
```

If the load was unsuccessful, the following information is returned:

```
1P,2P,3P = 6 character mnemonic error code  
4P = L-  
5P = 0
```

If the load was aborted by the user with the AB, /A or system BR command, the following values are returned:

```
1P,2P,3P = 77 XXX  
4P = L-  
5P = 0
```

Aborting MLLDR

The abort command, AB or /A, can be entered at any time in response to an MLLDR prompt.

At times, the user may wish to abort the load while the load is going on (i.e. when the abort command cannot be entered because the loader has not prompted for a command). In this case, use the system break (BR) command, which causes the loader to terminate in an orderly manner. The command should be used in the form:

```
S=xx COMMAND ?BR,MLLxx
```

where xx is the number of the user's session. When performing a permanent load, purge, or permanent replacement, use the break command in the form:

```
S=xx COMMAND ?BR,MLLDR
```

The system OF command should not be used, since files being accessed by the loader may be left open.

When the loader is aborted using the abort or system break command, it issues the message:

```
/MLLDR: MLLDR ABORTED
```

Chapter 5

Commands

Commands are listed alphabetically. For a list of commands by category, refer to Chapter 4. For more information about command functions, refer to Chapter 2 and 3.

Each command is described as follows:

1. A word or phrase indicating the function of the command.
2. Legal forms of the command. Lower case is used to indicate parameters entered by the user.
3. Command description.

NOTE

Specification of the commands listed below must precede specification of any relocate or search type command if the commands are coming from an LU, or they must precede the root definition if they are in an MLS command file. Otherwise, the command will be ignored if entered from an interactive device, or cause errors if entered from a file. Note that runstring command parameters will be overridden by any commands subsequently entered from a command file or interactively.

AS	LI	PF	SZ
EC	LL	SA	VS
FM	OP	SH	WS

Abort

AB
/A

Aborts the loader immediately. A clean termination of the load operation is performed. All files in use are closed.

Assign

AS,xx

(see Note)

Assigns the program being loaded to partition xx.

Disc Node Specification

D

Disc-resident node specification. All routines until the next D command are to be relocated in the disc-resident node specified in this command. Since the main of a program is always memory-resident, the disc-resident nodes start at level one. The format of the D command is as follows: D.i, where i is a positive integer, specifies a disc-resident node at the highest level available to disc-resident nodes. D.i.j specifies a disc-resident node on the second level which is a descendent of node D.i. For each new level, another .x is appended, and for each new node on that level, x is incremented to x+1.

Display Undefined Externals

DI

Prints a list of undefined externals on the list device, or, in the interactive mode, on the interactive scheduling device. Note that the undefined externals listed are those referenced by the node currently being loaded. When using LOADR, the "DI,AL" form of the command lists undefined externals in both the main and the current segment.

Echo Commands to List Output

EC

(see Note)

Causes the input commands from a file to be echoed on the list device as they are encountered. This is useful for debugging loader command files. The command is ignored if the commands are coming from an LU.

End of Command Input

EN

EX

/E

End of command input. Signals the loader to exit command mode and finish the load. If undefined externals exist at this time, an automatic search of the system library is performed. If undefined externals still remain, the user is prompted for commands (when force option not in effect and not running under Batch).

Format

FM,format

(see Note)

Specifies a format parameter, where "format" is as defined as follows:

DB Append debugger

LE List entry points and base page linkages on loadmap

NL No listing

DC Don't copy

Linkage type:

BP Base page

CP Current page

MP Mixed

Format codes specified interactively or in a command file will override format codes specified in the runstring. Also, for both runstring and interactive entry of format codes, a format specification will override a previous format specification.

Force Load

FO

Force load the remainder of the program. Undefined externals will be ignored. The FORCE option, once set, remains in effect throughout the load. The result is unpredictable if an unsatisfied external is referenced during program execution. This is because the address of an unsatisfied external is set to zero by the loader.

Library File

LI,file
(see Note)

This sets up the specified file as a library file. Up to 10 files can be specified. The SL command causes these files to be searched. These files also will be searched automatically at the end of every load, and at the end of each node in an MLS program load. Before continuing to the next library file, a file is searched multiple times, until no undefined externals are satisfied by a search of that file.

List Output

LL,namr
(see Note)

Specifies the file or LU that the listing is to go to. If a file name is specified, the file must not already exist unless its name begins with an apostrophe ('). If the listing is sent to a non-interactive LU, the LU is locked for the duration of the load.

Set Relocation Base

LO,address

LO,+n (MLLDR only)

Changes the relocation base of the next module to be relocated to the specified address. To specify the address in octal, place a 'B' after the last digit (i.e., 46000B). This command is useful for aligning modules to page boundaries to conserve program links. If +n is specified, the relocation base is moved to the nth subsequent page boundary. The command "LO,+0" causes the relocation base to be moved to the next page boundary. Note: Commands that set conditions for the load must occur before the LO command.

Memory Node

M

Memory-resident node specification. All routines until the next M or D command are to be relocated in the memory-resident node specified in this command. The format of the M command is as follows: M by itself specifies the root node. M.i, where i is a positive integer, specifies a memory-resident node on level 1, (where the main, or root node, is level 0). M.i.j specifies a node on level 2 which is a descendent of node M.i. For each new level, another .x is appended, and for each new node at that level, x is incremented to x+1.

Note: Memory nodes require more space than disc nodes since they are aligned to page boundaries.

Multiple Search

MS,namr

Searches namr for undefined externals. If no namr is specified, the system library is searched. The file is searched multiple times to ensure that backward references are satisfied. That is, if a routine is loaded that satisfies an undefined external, and that routine in turn causes more undefined externals, then the file is searched again to attempt to load the new external routines.

No search type commands (MS, SL, SE) can occur between the first NA or SY command in the node and the last NA or SY command in the node. In a given node, search commands can occur before the first NA or SY command or after the last NA or SY command specified for the node.

Note about search type (SE, MS, and SL) commands: Searches only satisfy undefined externals for the current node. A search in one node will not satisfy undefined externals of a previous node; searches are "local" to the node currently being constructed. If a node expects an undefined external to be satisfied in a son node, then the son node must have a relocate of the module containing the entry point, or an NA or SY command containing the entry point name.

See also SE and SL commands.

User Library Name

NA,name

The specified entry point name is to be found in one of the libraries declared in an LI command. The module containing name is loaded with the the current node. In Pascal or FORTRAN, name is a subroutine or program name. In assembly language, it is any entry point name.

No search type commands (MS, SL, SE) can occur between the first NA or SY command in the node and the last NA or SY command in the node. In a given node, search commands can occur before the first NA or SY command or after the last NA or SY command specified for the node.

Opcode

OP,opcode

(see Note)

Specifies an opcode parameter where opcode is as defined below. The default setting is EBNCTE.

Program type:

BG Background
RT Realtime
LB Large Background
EB Extended Background

COMMON type:

SC System
RC Reverse system
NC Local or no COMMON
SS Subsystem Global Area (SSGA)

Load type:

TE Temporary
PE Permanent
RP Replacement

VMA/EMA type:

VM Virtual Memory
EM EMA

Profile

PF,lu

(see Note)

Specifies that the profiling subroutine is to be appended to the program, and LU is the logical unit to which the profile output is to be printed.

Relocate

RE,namr

Loads the relocatable code contained in namr as part of the node currently being constructed. The namr specified can be a program or subroutine. For MLS programs, namr must not be an LU. An RE command cannot precede the first M command when loading an MLS program.

Save

SA,xx

(see Note)

Save command. Reserves xx words of local SAVE area for FORTRAN 77 and MACRO SAVE statement use. Sufficient space must be requested for all SAVE areas in the entire program.

Search

SE,namr

Searches the file namr for undefined externals. If no namr is specified, the system library is searched. Only the first two characters of this command need be specified for a single-pass search of the named file. If more than two characters are used in the command; that is, "SExxxx,namr" instead of "SE,namr", the file is searched multiple times to ensure that backward references are satisfied (this is the same as the MS command). The "SE,namr" form is faster but will not satisfy backward references.

No search type commands (MS, SL, SE) can occur between the first NA or SY command in the node and the last NA or SY command in the node. In a given node, search commands can occur before the first NA or SY command or after the last NA or SY command specified for the node.

Note about search type (SE, MS, and SL) commands: Searches only satisfy undefined externals for the current node. A search in one node will not satisfy undefined externals of a previous node; searches are "local" to the node currently being constructed. If a node expects an undefined external to be satisfied in a son node, then the son node must have a relocate of the module containing the entry point, or an NA or SY command containing the entry point name.

See also MS and SL commands.

Shareable EMA

SH,label
(see Note)

EMA area of the program is to be in a shareable EMA partition. The shareable EMA partition in the system is identified by label. Also, if a shareable EMA label file with the name label exists on LU 2, then the program will use shareable EMA label file.

Search User Libraries

SL

Search all files previously specified with the library (LI) command. For each file, a multiple search is performed. That is, if a routine loaded from one of the files satisfies an undefined external, and the routine in turn causes more undefined externals, then the library file is searched again.

No search type commands (MS, SL, SE) can occur between the first NA or SY command in the node and the last NA or SY command in the node. In a given node, search commands can occur before the first NA or SY command or after the last NA or SY command specified for the node.

Note about search type (SE, MS, and SL) commands: Searches only satisfy undefined externals for the current node. A search in one node will not satisfy undefined externals of a previous node; searches are "local" to the node currently being constructed. If a node expects an undefined external to be satisfied in a son node, then the son node must have a relocate of the module containing the entry point, or an NA or SY command containing the entry point name.

See also SE and MS commands.

System Library Name

SY,name

The specified entry point name is to be found in the system library. The module containing name is to be loaded with the current node.

No search type commands (MS, SL, SE) can occur between the first NA or SY command in the node and the last NA or SY command in the node. In a given node, search commands can occur before the first NA or SY command or after the last NA or SY command specified for the node.

Size

SZ,xx

SZ,+n (MLLDR only)

(see Note)

This command allows the user to request more memory for the program than the actual program code requires. The extra space is called dynamic buffer area. xx is the number of pages of memory for the program and dynamic buffer area. For EMA programs, the EMA area will immediately follow the dynamic buffer area. Note that this dynamic buffer area can be changed on-line for non-EMA programs with the SZ operator command. If +n is specified, n more pages than the program requires is set as the required program size.

Transfer

TR,namr

TR

Transfer to namr for subsequent MLLDR commands. TR from a command file to a command file or LU is not allowed. If namr is not specified, return to the command file that was suspended when undefined external was encountered. Control goes to the command following the command that was being processed when the undefined external was encountered.

VMA Size

VS,xxxxx
(see Note)

The program is a VMA program, and xxxxx specifies the last page of VMA, where VMA pages are numbered starting at zero. Thus, the program has a maximum VMA size of xxxxx + 1 pages, where $31 \leq \text{xxxxx} \leq 65535$. The command "VS,1000", for example, specifies that the program has a maximum VMA size of 1001 pages. This command will override the default size set by the "OP,VM" command. If the EMA size specified in the relocatable records is greater than the VMA size, then the VMA size will be set to the EMA size. If $175777B < \text{VS size} < 177776B$ (decimal, $64511 < \text{VS} < 65534$), then the VS size will be set to 177777B (decimal 65536). Numbers in this range are reserved for use by the Virtual Memory system.

Working Set Size

WS,xxxx
(see Note)

Working Set size. The program is a VM program with a required working set size of xxxx pages, where $5 \leq \text{xxxx} \leq [\text{max partition size} - \text{code size}]$. This command overrides the default WS size set by the "OP,VM" command. Note: Shareable Virtual Memory is not allowed; you cannot specify SH and VS, WS, or VM.

Comment

*

Denotes a comment line when entered as the first character of an entry line. The loader ignores the entire line. Comments can also follow a command and be in the same entry line as the command, providing two commas appear in the line. For example:

```
RE,PROGA      ,LOAD PROGRAM NAMED PROGA
SE           ,,SEARCH THE SYSTEM LIBRARY
DI           ,,DISPLAY UNSATISFIED EXT REFS
```


Chapter 6

Segmentation and Loading Tools

Segmentation Utility SGMTR

Function

SGMTR is a utility for generating preliminary loader command files for the Multi-Level Segmentation Loader MLLDR. SGMTR analyzes a program's relocatables in terms of module to module references and module sizes and creates an MLS loader command file as output.

The resulting command file may not be optimal in terms of either program size or execution time. However, it gives the user a tool for loading large programs with less time spent segmenting the program than if the command file was written entirely by the user. Once the initial command file has been generated, the user can improve it by editing the commented loader command file.

More information about using SGMTR to help load large programs is contained in Chapter 8.

Operation

Runstring Parameters

SGMTR accepts the following runstring:

```
RU,SGMTR,input[,output[,pathsize[,mainent[,segoptions  
[ ,loadoptions]]]]]
```

where

input File containing merged relocatables for the program to be segmented.

output Name of loader command file to be created. Default value is the session terminal.

Segmentation and Loading Tools

pathsize Maximum number of pages allowed in a path. Default value is 31 decimal.

mainent Main entry point of the program. Default is the first module in the input file containing an entry point.

segoptions Use either "M" or "D" for memory-resident or disc-resident nodes. Default is memory-resident nodes. The "A" option will cause an NA or SY command to be generated for every module in the program. Default operation is described below. To use more than one option, include them with no intervening blanks or commas.

loadoptions Use either "VM" or "EM" for VMA or EMA. The default is "EM" if EMA use is detected in the main module of the program. Use either "PF" or "DB" for a load with profiler or debugger. To use more than one option, include them with no intervening blanks or commas.

Input File

The input file should contain all the relocatables and libraries needed to load the program except the system library routines. SGMTR will search the system library after searching the input file to resolve undefined external references. For more information, refer to the section "Preparing the Input File".

Output File

If the output file already exists then the user will be asked:

```
THE FOLLOWING FILE ALREADY EXISTS. OK TO OVERLAY IT? <file name>
ENTER YES/NO/EXIT :
```

If the user enters "YES" then SGMTR continues and overlays the old file. If the user enters "EXIT" or a cntrl/D then SGMTR terminates. If the user enters "NO" then the user will be asked:

```
ENTER NEW FILENAME:
```

If the first two characters of the filename are "EX", or a cntrl/D is entered, then SGMTR will terminate. Otherwise the new filename is read and the check for existence is applied again.

Segmentation and Loading Tools

Path Size

A path size from 1 to 31 pages can be specified. The maximum size for EMA/VMA programs is 29. This allows 2 pages at the end of the logical address space, required for use of EMA or VMA.

Note: Memory-resident nodes are aligned to page boundaries. Specifying disc nodes in SGMTR's runstring can help reduce the program's required path size.

If SGMTR cannot fit the program into the specified path size, then the SGMTR error PATH OVERFLOW will occur. The error will appear in the command file in the node that caused the overflow. When SGMTR encounters a path overflow, it stops segmenting the code beyond the point of the overflow. It places all of this code in a single leaf node, and continues segmenting the rest of the program. Thus, more than one path overflow can occur during one run of SGMTR.

If a program is loaded with a command file that contains path overflow errors, either a memory overflow error will occur, or the program will load successfully but will require a larger path size than was specified in SGMTR's runstring.

In order to correct the path overflow error, specify a larger path size and run SGMTR again. If a larger path cannot be used, the SGMTR output will indicate the critical path(s) of the program in terms of size, and aid the user in correcting the path size problem. SXREF can help analyze the command file.

Refer to Chapter 8 for more information about reducing path size.

Note: Using a small path usually results in more duplication of modules than when using a large path. This increases the total program size. If the program uses memory-resident nodes, using a smaller path can increase the size of the partition required to run the program.

Segmentation and Loading Tools

Main Entry Point

The main module of the program is specified using the main entry point parameter. The default main module is the first module in the input file that contains an entry point. The main module must be specified by an entry point since SGMTR emits an NA command in the command file to load the main module.

This parameter can also be used to tell SGMTR to segment a subtree of the program; see the section "Non-Main Modules", below.

FORTTRAN

The main module of a FORTRAN program contains an entry point that matches the name of the program. Every FORTRAN subroutine or function contains an entry point which matches the name of the subroutine or function.

Pascal

The main module of a Pascal program contains an entry point that matches the name of the program. Note: Pascal programs must be compiled with the RTE-6/VM version of Pascal.

The LIBRARY compiler option causes the compiler to create a separate module for each procedure and function, and each module contains an entry point matching the name of the procedure or function. This option must be used when compiling programs that will be loaded using multilevel segmentation.

Segmentation and Loading Tools

MACRO

For MACRO programs, any entry point in the main module can be used. An entry point must exist in the main routine so that SGMTR can emit a "NA" MLLDR command to load the module in the root node.

The main module of a MACRO program should be type 2, 3, 4, or 6, and a transfer address should be declared. Figure 6-1 shows an example.



```
MACRO,Q,C
    NAM PROG,3 ;Declare module to be type 3, a main module
    ENT PROG
PROG  NOP      ;Main entry point as defined by transfer address
    ...
    END PROG  ;Declare a transfer address for the loader
```

Figure 6-1. Declaring a Main Module in MACRO

Non-Main Modules

A non-main module can be specified in the main entry point parameter for purposes of segmenting a subtree or portion of the program. In this case, do not include the main module of the program in the input file since it may be referenced by subroutines. This would force the entire program to be segmented. In general, to insure that only the subtree is segmented, remove all program code not in the subtree from the input file.

Note that declaring a non-main module to be used as the main will work for the purpose of segmenting, but the program subtree may not load. The module specified as the "main", for example, will probably not contain a transfer address, and undefined externals may exist.

Segmentation and Loading Tools

Segmentation Options

Use either "M" or "D" to produce memory-resident or disc-resident nodes. The resulting command file can later be edited to change the type of nodes. Memory nodes can be changed to disc nodes, but changing disc nodes to memory nodes may cause a memory overflow when the program is loaded. MLLDR aligns the start of each memory-resident node to a page boundary. Specifying "M" causes SGMTR to take this into account when segmenting the program. A path loaded using memory nodes will usually be longer than the same path loaded using disc nodes.

The "A" option causes SGMTR to emit an NA or SY command for every module referenced in the program. The default is not to emit NA and SY commands for leaf node modules that would be loaded automatically by MLLDR's default searches. The default ("A" option not used) results in a shorter command file and a faster load. The "A" option is useful for showing all the routines used in the program without doing a load.

Load Options

Options specified in this parameter tell SGMTR to allow for the loadtime overhead in the root node caused by the options.

Specify EMA or VMA using "EM" or "VM". The default is "EM" if EMA use is detected in the program's main module. Note that the system routines that handle VMA are larger than those that handle EMA, so a VMA program requires more code in the root node.

Use "PF" or "DB" for a load with the profiler or debugger. Only one can be used; if both are specified, an error will be reported. SGMTR includes the overhead for the profiler or debugger in its pathsize calculations. Note: SGMTR does not count all overhead for the profile option. The amount of spaces not counted is approximately (3 + number of nodes) words.

Note that SGMTR forces the use of base page linking because it cannot predict the effect of current page link allocation on path size. If a base page link overflow occurs, try changing the command file to use mixed or current page links.

Segmentation and Loading Tools

Scratch File

SGMTR uses a temporary scratch file during execution. It is created as a type 3 file, 64 blocks long, in the first cartridge of the user's cartridge list. It is purged before SGMTR terminates.

Aborting SGMTR

To abort a run of SGMTR, use the system BR command. This ensures that all files used by SGMTR are closed, and that the scratch file is purged. The command file being created is not purged.

Preparing the Input File

The input file must be prepared as described in this section. The main considerations are the use of COMMON and the order of relocatable modules in the input file.

COMMON

If the program being segmented uses COMMON, then certain requirements must be met in order for SGMTR to properly segment the program and to avoid load-time errors. For blank COMMON, the largest declaration must appear in the first module loaded (the main module). The three types of labeled COMMON are EMA, SAVE, and "ordinary" (non-EMA, non-SAVE). SGMTR requirements for the use of labeled COMMON are described below.

Segmentation and Loading Tools

Non-EMA Labeled COMMON - Block Data Subprograms

For each "ordinary" (non-EMA, non-SAVE) labeled COMMON block that is used in the main, declare the block using the noallocate option (when using FORTRAN 77). Create a block data subprogram that contains all of the program's ordinary labeled COMMON blocks. For each COMMON block, declare the largest size and, when using FORTRAN 77, use the noallocate option. The block data subprogram(s) must follow the main in the input file if blank COMMON is used.

Figure 6-2 shows a sample FORTRAN 77 program using this method. Figure 6-3 shows a similar program written in MACRO.

Use of the noallocate option (with FORTRAN 77) allows SGMTR to place COMMON blocks in specific nodes. The relocatable code for the block data subprogram, when used with the noallocate option as shown in Figure 6-2, contains a separate module for each COMMON block, and each module contains an entry point matching the name of the COMMON block. The entry points are needed so that SGMTR can emit an NA command to allocate space for each block in the appropriate place in the MLS tree. When MLLDR searches the library file to load modules specified in NA commands, the COMMON block module is loaded (space is allocated for the COMMON block in the current node).

Note that if a COMMON block is placed in a disc-resident node, then its values may be redefined after control is returned to that node. Use SAVE labeled COMMON for variables that must retain their values between subroutine calls.

Segmentation and Loading Tools

FTN7X

C Ordinary labeled COMMON used in the main are declared
C in the main using noallocate option

```
$ALIAS /XXX/, NOALLOCATE
PROGRAM EXMPL (3,90) Example using ordinary COMMON
COMMON /XXX/ BUFF(100)
```

```
·
·
·
```

```
CALL SUB1
```

```
·
·
·
```

```
END
```

C

C Declare all the program's ordinary labeled COMMON blocks
C in a block data subprogram using noallocate

```
$ALIAS /XXX/, NOALLOCATE
$ALIAS /YYY/, NOALLOCATE
BLOCK DATA SUBPROGRAM
COMMON /XXX/ BUFF1(100)
COMMON /YYY/ BUFF2(200)
DATA BUFF1/1,2,3,97*0/
DATA BUFF2/200*0/
END
```

C

C No requirements in subroutines; noallocate is not necessary

```
SUBROUTINE SUB1
COMMON /YYY/ BUFF2(200)
```

```
·
·
```

```
END
```

Figure 6-2. FORTRAN 77 Example: Ordinary Labeled COMMON,
Block Data Subprogram

Segmentation and Loading Tools

```
MACRO
    NAM EXMPL,3
    ENT EXMPL
    EXT XXX,YYY
EXMPL NOP
    .
    .
    END EXMPL

    NAM XXX,1000B   Type 1000B indicates a data module
    ENT XXX
XXX  DEC 1,2,3
    BSS 97
    END

    NAM YYY,1000B   Type 1000B indicates a data module
    ENT YYY
YYY  BSS 200
    END

    NAM SUB1,7
    ENT SUB1
    EXT YYY
    .
    .
    END
```

Figure 6-3. MACRO Example: Block Data Subprogram

EMA and SAVE Labeled COMMON

All EMA and SAVE labeled COMMON blocks must be declared in the main with their largest sizes.

Labeled COMMON Block Types and Sizes

Each labeled COMMON block should be declared with the same type and size in all occurrences of the block in the program. This helps prevent errors at load time. The FORTRAN INCLUDE feature is an easy way to declare consistent types and sizes for all occurrences of each COMMON block.

SGMTR reports type and size mismatches on the user's terminal and in the command file.

Segmentation and Loading Tools

Merged and Indexed Files

The input file should contain all the program and library relocatables required to load the program except for system library routines. SGMTR will search the system library to satisfy undefined externals after searching the input file.

The MERGE utility can be used to combine all of the program's relocatable files (including library files) into one file. MLLDR will run faster if this file is indexed using the utility INDXR.

NOTE: Do not merge indexed files together. MLLDR only recognizes the first index in the merged file; no other entry points in the file will be found.

The input file for SGMTR should be organized as follows. Compile program code in the following order, or merge separate program relocatable files in the following order:

1. The main module of the program.
2. Block data subprograms (for ordinary labeled COMMON).
3. The remainder of the program (subroutines, etc.).

Then use merge to append all required library files to the program file.

RPL's

If the program uses user-defined RPL's, then a module should be created which defines all the RPL symbols with their values. This module should be contained in the input file, but the RPL symbols will not show up in the command file since the NA command cannot reference a RPL symbol. Therefore, the user should edit the command file, inserting an RE (relocate) command in the root to load the module containing the RPL definitions.

SGMTR Output

The program output shown in this section is a sample only. Output on various systems will differ according to system configuration and current software revision.

Sample Program, Terminal Output, and Command File

Below is the source for a sample program. Explanation of the information in the command file follows the command file example.

```

MACRO,Q
    NAM MAINE,3
    ENT MAINE, SUBR6
    EXT SUBR2, SUBR5, EXEC
MAINE NOP
    JSB SUBR2
    JSB EXEC
    DEF *+2
    DEF SIX
SIX DEC 6
SUBR6 NOP
    JSB SUBR5
    JMP SUBR6,I
    BSS 1000
    END MAINE

    NAM SUBR2,7
    ENT SUBR2, OFFPA
    EXT SUBR3, SUBR4
SUBR2 NOP
    JSB SUBR3
    JMP SUBR2,I
OFFPA NOP                ; this entry point will force
                        ; an offpath reference

    JSB SUBR4
    JMP OFFPA,I
    BSS 1000
    END

    NAM SUBR3,7
    ENT SUBR3
    EXT OFFPA
    EXT NODUP            ; here is a reference to a
                        ; non-duplicatable module

```

Segmentation and Loading Tools

```
SUBR3  NOP
      JSB OFFPA      ; start of offpath reference
      LDA NODUP      ; forces module NODUP to be
                   ; nonduplicatable
      JMP SUBR3,I
      BSS 1000
      END
      NAM SUBR4,7
      ENT SUBR4
      EXT NODUP
SUBR4  NOP
      LDA NODUP
      JMP SUBR4,I
      BSS 1000
      END

      NAM SUBR5,7
      ENT SUBR5
      EXT UNDEF
SUBR5  NOP
      LDA UNDEF      ; force an undefined external warning
      JMP SUBR5,I
      BSS 1000
      END

      NAM NODUP,7    ; referenced as data
      ENT NODUP      ; must not be duplicated.
NODUP  NOP
      BSS 10
      END
```


Segmentation and Loading Tools

Using the runstring

```
:RU,SGMTR,%X,OUTPUT,4,MAINE,M
```

the text below would appear on the user's terminal. This text also appears in the command file. A description of this text and the command file follows the sample command file.

```
:RU,SGMTR,%X,OUTPUT,4,MAINE,M
SGMTR: MODULE #      1  MAINE
SGMTR: MODULE #      2  SUBR2
SGMTR: MODULE #      3  SUBR3
SGMTR: MODULE #      4  SUBR4
SGMTR: MODULE #      5  SUBR5
SGMTR: MODULE #      6  NODUP                ;   referenced as data
SGMTR: MODULE #      7  $LOC$                92084-1X415 REV.2121 810723
SGMTR: MODULE #      8  .RRGR                92084-1X419 REV.2121 810723
SGMTR: MODULE LOAD COMPLETED
SGMTR: UNSATISFIED EXTERNALS IN MODULE:SUBR5
  UNDEF
SGMTR: COMMON BLOCKS ARE:
SGMTR: SEGMENTING... TOTAL PROGRAM SIZE IN DECIMAL                5107
SGMTR: WARNING! PATH LIMIT OVERFLOW!
SGMTR: WARNING! PATH LIMIT OVERFLOW!
SGMTR: OFFPATH REFERENCE CHECK
SGMTR: OFFPATH REFERENCE ERROR IN WHICH THE MODULE:
  SUBR3
  IN NODE ORDINAL          2
  REFERENCES IN A PREDECESSOR NODE THE MODULE:
  SUBR2
  WHICH COULD POTENTIALLY CAUSE AN OFFPATH REFERENCE
SGMTR:          5 NODES CREATED
SGMTR: CREATING OUTPUT LISTING
SGMTR: LISTING NODE: M
SGMTR: LISTING NODE: M.1
SGMTR: LISTING NODE: M.1.1
SGMTR: ...PATH LIMIT OVERFLOW!
SGMTR: LISTING NODE: M.1.2
SGMTR: ...PATH LIMIT OVERFLOW!
SGMTR: LISTING NODE: M.2
SGMTR: SGMTR COMPLETED!
```

Segmentation and Loading Tools

Here is the resulting command file:

```

* RU,SGMTR,%X,OUTPUT,4,MAINE,M
* 10:39 AM THU., 29 OCT., 1981
LI, %X
*SGMTR: MODULE #      1  MAINE
OP,BP
*SGMTR: MODULE #      2  SUBR2
*SGMTR: MODULE #      3  SUBR3
*SGMTR: MODULE #      4  SUBR4
*SGMTR: MODULE #      5  SUBR5
*SGMTR: MODULE #      6  NODUP           ;   referenced as data
*SGMTR: MODULE #      7  $LOC$         92084-1X415 REV.2121 810723
*SGMTR: MODULE #      8  .RRGR         92084-1X419 REV.2121 810723
*SGMTR: MODULE LOAD COMPLETED
*SGMTR: UNSATISFIED EXTERNALS IN MODULE:SUBR5
* UNDEF
*SGMTR: COMMON BLOCKS ARE:
*TOTAL PROGRAM SIZE IN DECIMAL          5107
*SGMTR: WARNING! PATH LIMIT OVERFLOW!
*SGMTR: WARNING! PATH LIMIT OVERFLOW!
*SGMTR: OFFPATH REFERENCE CHECK
*SGMTR: OFFPATH REFERENCE ERROR IN WHICH THE MODULE:
* SUBR3
* IN NODE ORDINAL          2
* REFERENCES IN A PREDECESSOR NODE THE MODULE:
* SUBR2
* WHICH COULD POTENTIALLY CAUSE AN OFFPATH REFERENCE
*SGMTR:          5 NODES CREATED
*SGMTR: CREATING OUTPUT LISTING
*****
*
M
*
* NODE SIZE IS          2048 WORDS (          4000B)
*****
*   MAINE
* MODULE SIZE IS   1009 WORDS (   1761B)
NA,MAINE
NA,SUBR6
*
*   $LOC$          92084-1X415 REV.2121 810723
* MODULE SIZE IS   42 WORDS (    52B)
SY,$LOC$
SY,$LOD$
* THE PREVIOUS MODULE MAY NOT BE DUPLICATED

```

Segmentation and Loading Tools

```
*
*   .RRGR           92084-1X419 REV.2121 810723
* MODULE SIZE IS   29 WORDS (   35B)
SY,.RRGR
SY,.SVRG
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
*
*****
*
M.1
*
* NODE SIZE IS           2048 WORDS (       4000B)
*****
*   SUBR2
* MODULE SIZE IS   1006 WORDS (   1756B)
NA,OFFPA
NA,SUBR2
*
*   NODUP           ;   referenced as data
* MODULE SIZE IS   11 WORDS (    13B)
NA,NODUP
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
* THE PREVIOUS MODULE MAY NOT BE DUPLICATED
*
*****
*
M.1.1
*
* NODE SIZE IS           1024 WORDS (       2000B)
*SGMTR: ...PATH LIMIT OVERFLOW!
*****
*   SUBR3
* MODULE SIZE IS   1004 WORDS (   1754B)
NA,SUBR3
*
*****
*
M.1.2
*
* NODE SIZE IS           1024 WORDS (       2000B)
*SGMTR: ...PATH LIMIT OVERFLOW!
*****
*   SUBR4
* MODULE SIZE IS   1003 WORDS (   1753B)
NA,SUBR4
*
```

Segmentation and Loading Tools

```
*****
*
M.2
*
* NODE SIZE IS          1024 WORDS (          2000B)
*****
*   SUBR5
* MODULE SIZE IS  1003 WORDS (  1753B)
NA,SUBR5
*
END
```

The first two lines are comments that show the runstring SGMTR read and the time of the segmentation.

The next line specifies the input file as the global library file for MLLDR. Modules referenced by entry point parameters in NA commands will be searched for in this file.

Then the main module with its NAM and comment string are listed, followed by the "OP,BP" loader command, which forces use of base page links. The other modules are loaded, and the end of this phase is noted by the MODULE LOAD COMPLETED message.

This is followed by unsatisfied external messages. In the example, the external symbol UNDEF is used in the module SUBR5, which listed with its NAM and comment string.

COMMON blocks are then listed with their label, type, and size.

The total program size is then listed in decimal.

Offpath reference errors are then listed. The first module listed references a module in the root node which references a module off the path of the first module. The NAM and comment strings of the two modules are listed. Offpath references cause runtime errors. For more information refer to Chapter 3.

The total number of nodes created is reported.

Each node is then listed in preorder. The node command is listed followed by the size of the node, in decimal and octal, rounded up to a page boundary if it is a memory-resident node.

Then, for each module of the node, the NAM and comment string are listed followed by the module's size and the entry points of the module.

Segmentation and Loading Tools

User library modules are referenced via NA commands. System library modules are referenced via SY commands. If the module is ever referenced as data, then the comment:

```
THIS MODULE MAY NOT BE DUPLICATED
```

is listed. If the module references no other modules then the comment:

```
THIS MODULE REFERENCES NO OTHER MODULES
```

is listed.

The loader command file is terminated with an END command.

Shortening the Command File

Do not use the "A" segmentation option. The default is to list a subset of each leaf node's modules. The subset will be enough to cause all of a node's modules to be loaded by means of MLLDR's searches of global and system libraries at the end of a node's processing. Those searches will load modules to satisfy the unsatisfied references of the subset of modules.

Use EDIT as shown in Figure 6-4 to remove comment lines and leading blanks from the output file. This will make the output about one third its former size and make MLLDR run faster.

```
:RU,EDIT,#PROG      edit the command file
/SEREON             turn on regular expression option
/l$X/ //Q           remove spaces quietly
/l$D/^[A-Z]/AQ      delete all comments quietly
/EC #PROG2          create a second output file
:
```

Figure 6-4. Shortening the Command File Using EDIT

A second output file is created (instead of modifying the original copy) since the comments in the original file may be useful later.

Segmentation Restrictions

Some restrictions on MLS segmentation structure are described here. SGMTR does not detect all violations of these restrictions. For both command files created by SGMTR and by the user, the user must ensure that the command file represents a valid MLS segmentation structure. More information on segmentation structure restrictions is contained in Chapter 3.

Local Data: Disc Nodes and Duplicated Modules

Special consideration must be given for MLS programs in which routines are coded such that values from one call must be preserved for use in a subsequent call.

In memory-resident nodes, routines retain local variable values from previous calls even after the node has been mapped out and back in. When a routine in a disc-resident node is called, however, a new copy of the node may be brought in from disc if the path was not already mapped. In this case, local variables do not retain values from previous calls. Note that this is also true for COMMON blocks in disc-resident nodes.

The ANSI FORTRAN 77 standard does not support local variables and data except in the case of SAVE variables. This means that unless a variable is declared to be a SAVE variable, then the value of that variable need not reflect changes caused by prior calls to the routine containing it.

Note also, SGMTR will duplicate modules unless an entry point is referenced as data by some other module in the merged relocatable file. A data reference is anything other than a direct JSB to the entry point, A LDA or a DEF, for example, is a data reference. In FORTRAN, different copies of a module will not contain consistent values for a local variable unless that variable is declared to be a SAVE variable.

Recursion and Offpath References

In a recursive program, subroutines can call themselves or other modules which might indirectly call back to the original routine. Refer to Chapter 3 for an example. Recursion can cause offpath references, which may cause runtime errors that are hard to diagnose. Recursion can be used only if modules are positioned in the MLS tree as described in Chapter 3.

SGMTR reports where offpath references may occur.

Note that if the main module is called by one of its support routines, no offpath error will be generated since SGMTR assumes that no module ever calls the main module as a subroutine.

Passing Procedures as Parameters

In some programs, routine names are passed in parameter lists. (FORTRAN and Pascal allow this.) SGMTR detects data references to those procedures and marks them as nonduplicatable. They are placed, along with all their support modules, in a node above all nodes containing modules that reference them. A routine's support modules are all modules that are referenced, directly or indirectly, by the routine.

Organizing the MLS tree in this manner may cause path overflow errors in the subtree of the node containing the routines that are passed as parameters. In this case it may be necessary to change the code so that data is passed rather than routine names. In a FORTRAN program, for example, numeric values could be passed instead of procedure names. The numeric values could be used in a computed goto statement, transferring to a statement that calls the appropriate routine.

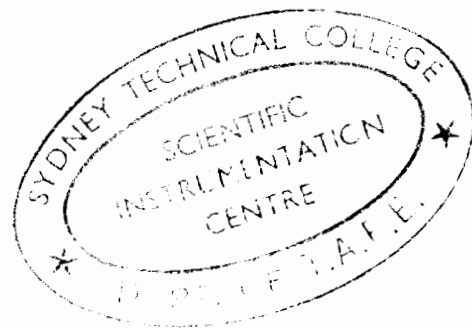
Note: SGMTR will not duplicate a routine that is passed as a parameter. However, if the routine is duplicatable, then the command file can be modified by the user so that the routine is duplicated. In some cases this may help solve pathsize problems.

Summary of SGMTR Errors

SGMTR error messages are listed here by category. Detailed error descriptions are in Appendix A.

Runstring messages:

SGMTR: BAD RUN STRING
SGMTR: NO INPUT FILE GIVEN
SGMTR: PATH LIMIT NOT INTEGER
SGMTR: PATH LIMIT OUT OF RANGE (1-31)
SGMTR: UNRECOGNIZED NODE TYPE
SGMTR: BAD TYPE SPECIFIED FOR FILE: <NAMR>
SGMTR: BAD SIZE SPECIFIED FOR FILE: <NAMR>
SGMTR: CANNOT CREATE FILE: <NAMR>
SGMTR: ILLEGAL LU: <NAMR>
SGMTR: MAIN ROUTINE MISSING
SGMTR: WARNING! MAIN ROUTINE HAS NO TRANSFER ADDRESS



Overflow messages:

SGMTR: SYMBOL TABLE OVERFLOW
SGMTR: EMA SPACE OVERFLOW
SGMTR: STACK OVERFLOW
SGMTR: MODULE STACK OVERFLOW
SGMTR: NODE STACK OVERFLOW
SGMTR: NODE TREE OVERFLOW

Symbol Table Messages:

SGMTR: BLANK COMMON SIZE ERROR
SGMTR: WARNING! LAST ROUTINE IS A SEGMENT
SGMTR: UNSATISFIED EXTERNALS IN MODULE: <NAM AND COMMENT>
<EXTERNAL> <EXTERNAL> <EXTERNAL> <EXTERNAL>
SGMTR: DUPLICATE ENTRY POINT SYMBOLS : <SYMBOL>
SGMTR: ENTRY POINT MATCHES RPL SYMBOL: <SYMBOL>
SGMTR: DUPLICATE RPL SYMBOLS : <SYMBOL>
SGMTR: RPL SYMBOL MATCHES COMMON : <SYMBOL>
SGMTR: ENTRY POINT MATCHES COMMON : <SYMBOL>
SGMTR: COMMON BLOCK TYPES DONT MATCH : <SYMBOL>
SGMTR: COMMON BLOCK SIZES DONT MATCH : <SYMBOL> <SIZE>
SGMTR: SAVE COMMON BLOCK NOT IN MAIN : <SYMBOL> <SIZE>
SGMTR: EMA COMMON BLOCK NOT IN MAIN : <SYMBOL> <SIZE>
SGMTR: BLOCK DATA MISSING FOR COMMON : <SYMBOL> <SIZE>

Segmentation and Loading Tools

Path Limit and Offpath Reference Messages:

```
SGMTR: WARNING! PATH LIMIT OVERFLOW!  
SGMTR: ...PATH LIMIT OVERFLOW!  
SGMTR: OFFPATH REFERENCE IN WHICH THE MODULE:  
      <NAM AND COMMENT>  
      IN NODE ORDINAL <ORDINAL>  
      IN A PREDECESSOR NODE REFERENCES THE MODULE:  
      <NAM AND COMMENT>  
      WHICH COULD POTENTIALLY CAUSE AN OFFPATH REFERENCE
```

File Manager and Termination Messages:

```
SGMTR: FMGRXXX ERROR WITH FILE: <NAMR>  
SGMTR: SEGMENT SGMT<X> CANNOT BE LOADED  
SGMTR: TERMINATING...  
SGMTR: COMPLETED!
```

Cross Referencer SXREF

SXREF creates a cross reference listing for a program according to its MLLDR command file. It also performs syntax checking on the command file.

Operation

SXREF accepts the runstring:

```
RU,SXREF,command[,list]
```

where:

command <namr> of the MLLDR command file, (must be a disc file).

list <namr> of the output listing file.

The user chooses between two modes of operation in SXREF's runstring.

If no output filename is specified in the runstring, then SXREF performs syntax and error checking on the command file. If an output file is specified, then SXREF also creates a cross reference listing.

Segmentation and Loading Tools

Output File

The output file can be defined as type 3, 4, 8, or higher. The size can be defined as one block or more.

If a file with the specified name already exists, the user is prompted:

```
THE FOLLOWING FILE ALREADY EXISTS. OK TO OVERLAY IT?  
OUTPUT:::3:24  
ENTER YES/NO/EXIT :
```

If the user enters "YES", then SXREF purges the old file and creates another. If the user enters "EXIT" or a cntrl/D character, then SXREF will terminate.

If the user enters "NO", then the user is prompted:

```
ENTER NEW FILENAME:
```

If the first two characters of the filename are "EX", or a cntrl/D character is entered, then SXREF will terminate. Otherwise the new filename is taken and the existence check is applied again.

Scratch Files

SXREF uses two scratch files during its execution. Their filenames begin with the characters "SX" and are followed by four digits. They are purged before SXREF terminates.

Aborting SXREF

SXREF can be cleanly aborted by setting the break bit using the system BR command. This will cause SXREF to terminate, closing any open files and purging any scratch files created during the run.

Syntax Checking Mode

The following error checks are made on the command file. Violations are listed on the user's terminal and the output listing if one is requested:

1. Command syntax.
2. Command order.
3. Referenced files are accessible.
4. Nodes are defined in preorder.
5. Disc-resident node definitions follow memory-resident node definitions.
6. The first disc-resident node definition occurs as a son of the root node.
7. The root node is a memory-resident node.
8. Total number of nodes is no more than 1024.
9. Maximum node depth is no more than 31.

If the user requests a cross reference listing then the following error checks are made and violations are listed on the users terminal and on the output listing:

10. Duplicate entry points.
11. Undefined entry points.
12. Potential offpath references.

SXREF does not detect all errors that MLLDR will detect. A nonexhaustive list follows:

1. Base page link overflow.
2. Module type. MLLDR will not accept type 5 (segment) modules.
3. Data references into son nodes.
4. Path size overflow versus size of available partitions.

Segmentation and Loading Tools

Expected Order of Commands

These commands must precede the definition of the root node (if any):

LL	SH	LI
AS	SA	FM, (BB, LE, NL, DC, BP, CP, MP)
SZ	VS	OP, (BG, RT, LB, EB, SC, RC, NC, SS, TE, PE, RP, VM, EM)
PF	WS	

The root node definition M need not be used; SXREF will operate on non-segmented programs. To use SXREF on a non-segmented program, or on a program for which no command file exists, a command file must be created which contains relocate (RE) commands for all the program code and search commands for the required libraries. This will allow SXREF to create a cross reference listing for the relocatable code.

The following commands can be used after the root node definition.

FO	SE
DI	MS
LO	SL
M.	NA
D.	SY
RE	

In any particular node, an SE or SL command cannot be used between the first and last occurrence of any NA or SY commands in that node.

An MLLDR command file can be terminated by any of these commands:

```
/A
AB
/E
EX
EN
<eof> end of file
```

A comment can appear anywhere within a command file:

```
* <comment line>
```

The following are legal loader commands but they are ignored by SXREF.

```
TR
The opcode PU
```

Segmentation and Loading Tools

After the syntax check of the command file has been completed, SXREF ignores all but the following commands in the command file. These commands are used in the cross reference work of the later passes if the user requested a cross reference listing by specifying an output file parameter in the runstring.

M	MS
D	SL
RE	NA
SE	SY

and any of the 'end' commands

Cross Reference Mode

Command File Listing

SXREF initially outputs an indented command file listing to outline the node definitions if SXREF is run in cross reference listing mode. A sample program, command file, and SXREF output are shown in the section Sample SXREF Output. Another example of SXREF output is contained in Appendix D.

Segmentation and Loading Tools

Cross Reference Listing

If SXREF finds syntax or command order errors in the command file during the first pass then execution will terminate at the end of that pass. Otherwise SXREF will generate a cross reference listing of the program defined by the loader command file. For each node, SXREF skips a page and prints the node ordinal (preorder) number and node definition command at the top of the node listing in the format:

```
NODE ORDINAL NUMBER   ###   M.1.2.3
-----
```

Nodes contain one or more modules. Information about each module is listed. This information includes the module name, comment field, subroutine type, size of program, local EMA space, base page space, local save space, unnamed or blank COMMON space, pure code space, EMA symbol and EMA space in pages, mseg page size, the symbol and size for any save, named, or EMA labeled COMMON areas, all entry point and external symbols, and the transfer address value as defined in the END record. All values are listed in decimal. The following format is used:

```
*MODULE RECORD* [subroutine name] [comment field]
TYPE           :#####
PROGRAM        :##### LOCAL EMA :#####
BASE PAGE:     :#### LOCAL SAVE:#####
COMMON         :##### PURE CODE :#####
EMA            :<sym> PAGE SIZE :#####
MSEG SIZE:     ##

SAVE COMMON   : [ COMMON label ] SIZE:      #####
NAMED COMMON  : [ COMMON label ] SIZE:      #####
EMA COMMON    : [ COMMON label ] SIZE:#####

ENTRY PTS:    [ entry point symbol ] [ ... ] [ ... ]

EXTERNALS:    [ external symbol   ] [ ... ] [ ... ]

END           :#####
```

Segmentation and Loading Tools

After all the modules of the node have been listed, SXREF skips a page and lists cross reference information. The first section lists entry points with their host module and the modules which reference them. The referencing modules are tagged with the ordinal number of the node which hosts them. The second section lists modules with the modules which reference them tagged with their host node's ordinal number.

```
ENTRY POINTS      ! DEFINED IN MODULE:! REFERENCED BY:
-----
[entry point]    [hostmodule]      [refingmod]:[ordn]
                  [refingmod]:[ordn]
[entry point]    [hostmodule]      [refingmod]:[ordn]
                  [refingmod]:[ordn]
MODULE            ! REFERENCED BY:
-----
[modulename]     [refingmod]:[ordn] [refingmod]:[ordn]
[modulename]     [refingmod]:[ordn] [refingmod]:[ordn]
                  [refingmod]:[ordn]
```

Then a listing is made of the externals which are satisfied outside of the current node with the ordinal number of the satisfying node. The tag indicates if the satisfying entry point resides in a predecessor or son node of the current node.

```
EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNAL          ! ORDINAL NUMBER ! PRED/SUCC
-----
[externalsymb1]  #####                PRED
[externalsymb1]  #####                SUCC
```

Then a scan is made for undefined externals. If none are found the message "...NO UNDEFS..." is listed. If any are found then they are listed with their host module. Unsatisfied external references are listed in the following manner:

```
UNDEFS           ! REFERENCED BY:
-----
[externalsymb1]  [hostmodule]
[externalsymb1]  [hostmodule]
```


Sample SXREF Output

This section shows a sample program, command file, and SXREF output. Another example of SXREF output is shown in Appendix D.

The program output shown in this section is a sample only. Output on various systems will differ according to system configuration and current software revision

Below is the sample program:

```

MACRO,Q
    NAM MAINE,3
    ENT MAINE, SUBR6
    EXT SUBR2, SUBR5, EXEC
MAINE NOP
    JSB SUBR2
    JSB EXEC
    DEF *+2
    DEF SIX
SIX    DEC 6
SUBR6  NOP
    JSB SUBR5
    JMP SUBR6,I
    BSS 1000
    END MAINE

    NAM SUBR2,7
    ENT SUBR2, OFFPA
    EXT SUBR3, SUBR4
SUBR2  NOP
    JSB SUBR3
    JMP SUBR2,I
OFFPA  NOP                ; this entry point will force
                        ; an offpath reference

    JSB SUBR4
    JMP OFFPA,I
    BSS 1000
    END

    NAM SUBR3,7
    ENT SUBR3
    EXT OFFPA
    EXT NODUP            ; here is a reference to a
                        ; non-duplicatable module

```

Segmentation and Loading Tools

```
SUBR3  NOP
      JSB OFFPA      ; start of offpath reference
      LDA NODUP      ; forces module NODUP to be
                   ; nonduplicatable

      JMP SUBR3,I
      BSS 1000
      END
      NAM SUBR4,7
      ENT SUBR4
      EXT NODUP

SUBR4  NOP
      LDA NODUP
      JMP SUBR4,I
      BSS 1000
      END

      NAM SUBR5,7
      ENT SUBR5
      EXT UNDEF

SUBR5  NOP
      LDA UNDEF      ; force an undefined external warning
      JMP SUBR5,I
      BSS 1000
      END

      NAM NODUP,7    ; referenced as data
      ENT NODUP      ; must not be duplicated

NODUP  NOP
      BSS 10
      END
```



Segmentation and Loading Tools

The command file for the program is:

```
LI, %X
OP,BP
M
NA,MAINE
NA,SUBR6
SY,$LOC$
SY,$LOD$
SY,.RRGR
SY,.SVRG
M.1
NA,OFFPA
NA,SUBR2
NA,NODUP
M.1.1
NA,SUBR3
M.1.2
NA,SUBR4
M.2
NA,SUBR5
END
```

When SXREF is run on the above command file, SXREF prints the following on the user's terminal:

```
:RU,SXREF,#M6EXS,"M6EX
SXREF: UNDEFINED ENTRY POINT: .CNOD          IN NODE          0
SXREF: UNDEFINED ENTRY POINT: UNDEF         IN NODE          4
SXREF: POSSIBLE OFFPATH REFERENCE VIA EXTERNAL SUBR4
OF A MODULE IN NODE      1
TO AN ENTRY POINT IN NODE      3
ORIGINAL CALLER IS SUBR3
IN NODE:      2
SXREF: COMPLETED!
:
```

Below is the command file listing for the program, followed by the cross reference listing. Lines too long for indentation are marked with an asterisk ("*") and are truncated when necessary.

Segmentation and Loading Tools

NODE LOADER
 ORDINAL COMMAND

```

-----
      LI, %X
      OP,BP

0 M
  NA,MAINE
  NA,SUBR6
  SY,$LOC$
  SY,$LOD$
  SY,.RRGR
  SY,.SVRG

1  M.1
  NA,OFFPA
  NA,SUBR2
  NA,NODUP

2  M.1.1
  NA,SUBR3

3  M.1.2
  NA,SUBR4

4  M.2
  NA,SUBR5
  END

SXREF: UNDEFINED ENTRY POINT:  .CNOD                    IN NODE            0
SXREF: UNDEFINED ENTRY POINT:  UNDEF                   IN NODE            4
SXREF: POSSIBLE OFFPATH REFERENCE VIA EXTERNAL SUBR4
OF A MODULE IN NODE            1
TO AN ENTRY POINT IN NODE            3
ORIGINAL CALLER IS SUBR3
IN NODE:                        2
  
```

Segmentation and Loading Tools

Below is the cross reference listing for the program.

NODE ORDINAL NUMBER 0 M

MODULE RECORD MAINE

TYPE : 3
PROGRAM : 1009 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: MAINE SUBR6

EXTERNALS: EXEC SUBR2 SUBR5

END : 0

MODULE RECORD \$LOC\$ 92084-1X415 REV.2121 810723

TYPE : 7
PROGRAM : 42 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: \$LOC\$ \$L0D\$

EXTERNALS: .SVRG .RRGR \$LOC
 .CNOD EXEC

END : 0

MODULE RECORD .RRGR 92084-1X419 REV.2121 810723

TYPE : 7
PROGRAM : 29 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: .RRGR .SVRG

END : 0

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

MAINE	MAINE
SUBR6	MAINE

Segmentation and Loading Tools

```

$LOC$           $LOC$
$LOD$           $LOC$
.RRGR           .RRGR           $LOC$           :   0
.SVRG           .RRGR           $LOC$           :   0
    
```

MODULE ! REFERENCED BY:

```

MAINE
$LOC$
.RRGR           $LOC$           :   0
    
```

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
 EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

```

SUBR2           1           SUCC
SUBR5           4           SUCC
    
```

UNDEFS ! REFERENCED BY:

```

.CNOD           $LOC$
NODE ORDINAL NUMBER       1 M.1
    
```

MODULE RECORD SUBR2

```

TYPE           :   7
PROGRAM        : 1006 LOCAL EMA :           0
BASE PAGE     :   0 LOCAL SAVE:           0
COMMON        :   0 PURE CODE :           0
    
```

ENTRY PTS: OFFPA SUBR2

EXTERNALS: SUBR3 SUBR4

END : 1542

MODULE RECORD NODUP ; referenced as data

```

TYPE           :   7
PROGRAM        :   11 LOCAL EMA :           0
BASE PAGE     :   0 LOCAL SAVE:           0
COMMON        :   0 PURE CODE :           0
    
```

ENTRY PTS: NODUP

END : 257

ENTRY POINTS ! DEFINED IN MODULE: ! REFERENCED BY:

Segmentation and Loading Tools

```

OFFPA          SUBR2          SUBR3          : 2
SUBR2          SUBR2          MAINE          : 0
NODUP          NODUP          SUBR3          : 2
                                   SUBR4          : 3
    
```

MODULE ! REFERENCED BY:

```

-----
SUBR2          MAINE          : 0          SUBR3          : 2
NODUP          SUBR3          : 2          SUBR4          : 3
    
```

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

```

-----
SUBR3          2          SUCC
SUBR4          3          SUCC
    
```

UNDEFS ! REFERENCED BY:

```

-----
.....NO UNDEFS .....
NODE ORDINAL NUMBER      2  M.1.1
-----
    
```

MODULE RECORD SUBR3

```

TYPE          : 7
PROGRAM       : 1004 LOCAL EMA : 0
BASE PAGE    : 0 LOCAL SAVE: 0
COMMON       : 0 PURE CODE : 0
    
```

ENTRY PTS: SUBR3

EXTERNALS: NODUP OFFPA

END : 1028

ENTRY POINTS ! DEFINED IN MODULE: ! REFERENCED BY:

```

-----
SUBR3          SUBR3          SUBR2          : 1
    
```

MODULE ! REFERENCED BY:

```

-----
SUBR3          SUBR2          : 1
    
```

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

```

-----
NODUP          1          PRED
OFFPA          1          PRED
    
```

UNDEFS ! REFERENCED BY:

Segmentation and Loading Tools

.....NO UNDEFS
NODE ORDINAL NUMBER 3 M.1.2

MODULE RECORD SUBR4
TYPE : 7
PROGRAM : 1003 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: SUBR4

EXTERNALS: NODUP

END : 771

* * * * *

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

SUBR4 SUBR4 SUBR2 : 1

MODULE ! REFERENCED BY:

SUBR4 SUBR2 : 1

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

NODUP 1 PRED

UNDEFS ! REFERENCED BY:

.....NO UNDEFS
NODE ORDINAL NUMBER 4 M.2

MODULE RECORD SUBR5
TYPE : 7
PROGRAM : 1003 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: SUBR5

EXTERNALS: UNDEF

END : 771

* * * * *

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

Segmentation and Loading Tools

SUBR5 SUBR5 MAINE : 0

MODULE ! REFERENCED BY:

SUBR5 MAINE : 0

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

UNDEFS ! REFERENCED BY:

UNDEF SUBR5



Summary of SXREF Errors

The following is a list of SXREF's error messages. These messages appear on the user's terminal and not the output listing. Detailed error descriptions are contained in Appendix A.

SXREF: BAD RUN STRING
SXREF: NO COMMAND FILENAME GIVEN

These messages apply to the output filename, if any.

SXREF: BAD TYPE SPECIFIED FOR FILE: <FILE>
SXREF: BAD SIZE SPECIFIED FOR FILE: <FILE>

These messages apply to a given line of the command file.

SXREF: LINE XXXXXX WRONG PARAMETER TYPE
SXREF: LINE XXXXXX COMMAND FILE READ ERROR
SXREF: LINE XXXXXX COULD NOT BE PARSED
SXREF: LINE XXXXXX COMMAND OUT OF ORDER
SXREF: LINE XXXXXX REFERENCED FILE COULD NOT BE OPENED
SXREF: LINE XXXXXX EITHER A DISC RESIDENT NODE
PRECEDES A MEMORY RESIDENT NODE
OR THE FIRST DISC RESIDENT NODE
IS NOT A SON OF THE ROOT NODE
SXREF: LINE XXXXXX NODE DEFINITION OUT OF ORDER
SXREF: LINE XXXXXX ROOT NODE DEFINITION MISSING
SXREF: LINE XXXXXX BAD FIRST SON NODE DEFINITION
SXREF: LINE XXXXXX BAD BROTHER NODE DEFINITION
SXREF: LINE XXXXXX BAD PREDECESSOR NODE DEFINITION

The following are warnings:

SXREF: NODE DEPTH OVER 31 DEEP
SXREF: NODE COUNT OVER 1024

The following messages will only be seen if SXREF is run in the mode which generates a cross-reference listing.

SXREF: MORE THAN 128 UNIQUE FILES ARE REFERENCED
SXREF: MORE THAN 10 GLOBAL FILES ARE DEFINED

Segmentation and Loading Tools

Duplicate or undefined entry points, or offpath references are flagged with warning messages:

```
SXREF: DUPLICATE ENTRY POINT: XXXXXXXXXXXXXXXXXXXX IN NODE  XXXXXX
      AND XXXXXX
SXREF: UNDEFINED ENTRY POINT: XXXXXXXXXXXXXXXXXXXX IN NODE  XXXXXX
SXREF: UNDEFINED ENTRY POINT: XXXXXXXXXXXXXXXXXXXX
      REFERENCED BY NA COMMAND IN NODE: XXXXXX
SXREF: UNDEFINED ENTRY POINT: XXXXXXXXXXXXXXXXXXXX
      REFERENCED BY SY COMMAND IN NODE: XXXXXX
SXREF: POSSIBLE OFFPATH REFERENCE VIA EXTERNAL XXXXXXXXXXXXXXXXXXXX
      OF A MODULE IN NODE XXXXXX
      TO AN ENTRY POINT IN NODE XXXXXX
ORIGINAL CALLER IS <NAM>
      IN NODE: XXXXXX
```

The following are miscellaneous messages which may appear during either run mode.

```
SXREF: SEGMENT XXXXXX COULD NOT BE LOADED
SXREF: FMGR XXX ERROR WITH FILE: <FILE>
SXREF: MORE DYNAMIC MEMORY SPACE NEEDED
SXREF: TERMINATING...
SXREF: COMPLETED!
```

Relocatable Indexer INDXR

The utility INDXR takes as input one or more relocatable (type 5) files and creates an indexed relocatable file. Indexing relocatable files increases the speed of loader search operations. The index consists of a directory of locations of NAM and ENT symbols in the file.

CAUTION

Do not use the MERGE utility to merge files that have already been indexed. The file should be indexed after it has been merged, or use INDXR to perform the merge operation.

INDXR error messages are in Appendix A.

Operation

INDXR can be operated interactively, or commands can come from a file.

The runstring is:

```
:RU,INDXR[,command]
```

If the utility is run in interactive mode it prompts the user for command input with;

```
/INDXR:
```

Commands

There are six INDXR commands that can be entered to build an indexed relocatable library. These commands are LIST, TRansfer, CReate, INdex, ENd(or EXit), and ABort.

Segmentation and Loading Tools

CR[EATE],namr

This command creates a file with attributes as specified in "namr". It is this file which is the indexed relocatable library file. Note: This command must precede any use of the INdEX command to merge relocatables into the library. CReate can be used once per invocation of the utility, since only one library can be built at a time. If a file size is not specified, then the default size of 24 blocks is used. A file size of -1 causes all available space on cartridge to be used, with the file being truncated to the actual area used when it is closed. If the file already exists then INDXR will ask the user if the file is to be overlaid with new data. When commands are coming from a transfer file/lu, then the INDXR will abort if the user chooses not to overlay the file. A type 0 file cannot be specified as the indexed file.

LI[ST],namr

The LIst command is used to alter the current list file or lu. When the utility is first invoked, LU 1 is the default list device. By using the LIst command this can be changed to another lu, or output can be directed to a file. Note that if the file specified does not exist, then one will be created. If the file is not type 3 or 4 then an error message is issued and the command ignored.

IN[DEX],namr

Copies the file specified by "namr" to the relocatable library file. As the file is copied all NAM and ENT symbols are placed in a directory. Note that a file name must be given and not an logical unit.

EX[IT] or EN[D]

Causes the indexed relocatable library file and the current list LU/file to be closed.

Segmentation and Loading Tools

AB[ORT]

Aborts the utility. If a file has been created with the CREATE command then it is erased. Also closes the list LU/file.

TR[ANSFER],namr

Causes control to be passed to a transfer file. Commands will be taken from the transfer file as if they were being read from a terminal. Only one level of transfer may be invoked, so a TRansfer command may not be contained in a transfer file.

Examples

Create an indexed relocatable library called %SLIB containing the type 5 modules %MAIN, %SUBA, and %SUBB.

```
:RUN,INDXR
/INDXR: CREATE,%SLIB           Create the indexed rel. library.
/INDXR: IN,%MAIN              Merge relocatable %MAIN using INdx.
/INDXR: MAIN
/INDXR: INDEX,%SUBA
/INDXR: SUBA
/INDXR: I,%SUBB               Invalid INdx command.
/INDXR: ??
/INDXR: IN,%SUBB
/INDXR: SUBB
/INDXR: END
      INDXR DONE
```

Create an indexed relocatable library called %FARM containing the relocatable files %PIG, %COW, %DOG, and the set of merged relocatable files in %CAT. A transfer file called *GROW contains commands for INDXR.

```
:LIST,*GROW

*GROW T=00004 IS ON CR LC USING 00001 BLKS R=0000
0001 CREATE,%FARM
0002 INDEX,%PIG
0003 INDEX,%COW
0004 INDEX,%DOG
0005 INDEX,%CAT
0006 END
```

Segmentation and Loading Tools

```
:RU,INDXR,*GROW
/INDXR: PIG
/INDXR: SUB1
/INDXR: SUB2
/INDXR: SUBX
/INDXR: SUBY
/INDXR: SUBZ
  INDXR DONE
```

No prompt is issued for command input when a transfer file is used.

Create the same library as above except this time send the listing to LU 6 (line printer).

```
:LI,*GROW
```

```
*GROW T=00004 IS ON CR LC USING 00001 BLKS R=0000
0001 LIST,6
0002 CREATE,%FARM
0003 INDEX,%PIG
0004 INDEX,%COW
0005 INDEX,%DOG
0006 INDEX,%CAT
0007 END
```

```
:RU,INDXR,*GROW
  INDXR DONE
```

All output now goes to the line printer (LU 6).

Chapter 7

Memory Allocation

Introduction

This chapter describes program memory layout and contains memory maps for various MLS program formats. Logical memory maps for user programs, contained in Appendix C, are referenced. This chapter also provides information about overhead for MLS and the profile option.

Understanding of MLLDR processing can help the user reduce the maximum pathlength and required partition size of the loaded program. This is important when loading a very large program, or when a program of any size must be loaded to run in the smallest partition possible. Examples of these situations are:

- A large program will not load with the given MLS structure (near the end of the load, for example, an L-OV MEM error occurs).
- A program must be loaded to fit in a small partition.
- After modifications to a program are made (for example, moving a section of code from one node to another) the loaded program is now a page larger than before.

After the user has determined the MLS structure (created a command file), MLLDR operation can still affect the size of the loaded program. This is especially apparent in the third situation above. Chapter 8, Loading Large Programs, describes methods of avoiding these problems.

Memory Layout

Physical and logical memory layout is shown in Figures 7-1 through 7-4. In the logical memory maps, one area is shown for system areas mapped in. The memory maps in Appendix C show the system areas in more detail and how the load points are set for various program types.

Base Page to Root

Base page links are allocated starting with location 2 (addresses 0 and 1 are used to access the A- and B-Registers). The program starts on the second physical page beginning with 10 locations reserved for use by the operating system, followed by a local COMMON area, if used, followed by a SAVE area, if used. These areas are immediately (not necessarily starting on a page boundary) followed by the root node. The beginning of the root is not aligned to a page boundary.

More information about base page link allocation is given in Chapter 4 in the section Loadmap ("LE Option").

Program Nodes

The layout of memory after the root node depends on whether the program contains memory nodes, disc nodes, both, or no other nodes besides the root. In all cases the nodes are organized in preorder, the order in which they were relocated by MLLDR, which is the order in which nodes were specified in the command file.

Root Node Only

For a program with only a root (a non-segmented program), the root is followed immediately (not necessarily on a page boundary) by the dynamic buffer area, if used. See Figure 7-1.

Disc-Resident Nodes Only

If the program contains only disc-resident nodes (Figure 7-2), then the root is followed by a space large enough to hold the longest path in the program. Disc-resident nodes are not aligned on page boundaries, and the root node is followed immediately (not necessarily starting on a page boundary) by the disc-resident node area. Disc-resident nodes are aligned to sector boundaries when the program is written to disc by MLLDR.

Memory Allocation

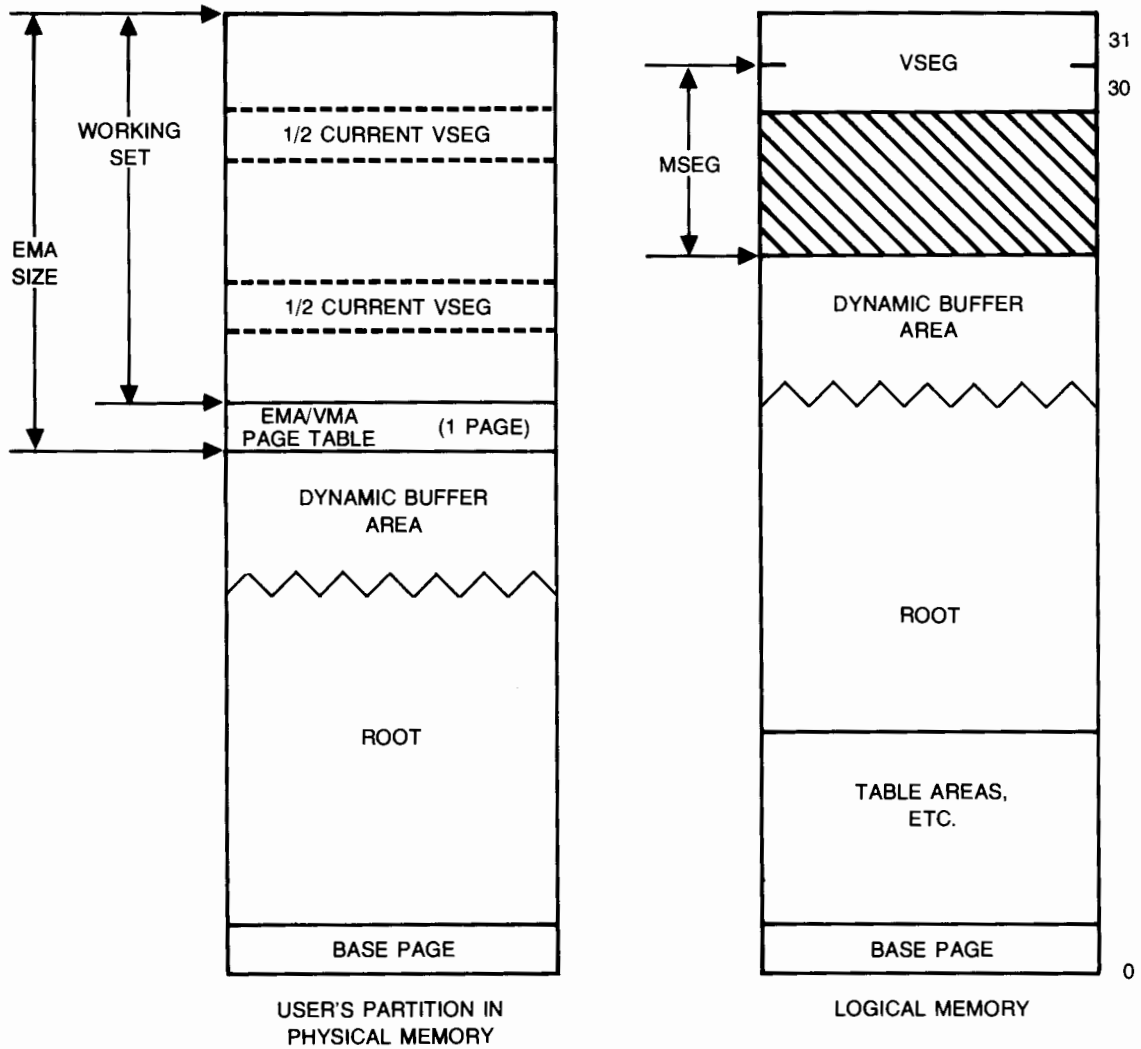
Memory-Resident Nodes Only

If the program contains only memory-resident nodes (Figure 7-3), the root is followed by a space large enough for all nodes in the program. Memory-resident nodes always begin and end on page boundaries (the root node is an exception in the case described above for programs containing disc-resident nodes only). Thus, the smallest possible size for a memory node is one page. Note that when a memory node ends several words beyond a page boundary, the remainder of the page is not used, and the next node starts on the following page.

Memory and Disc Nodes

If the program contains both memory-resident and disc-resident nodes (Figure 7-4), then the root is followed by all memory-resident nodes. Each memory-resident node is aligned to a page boundary. The memory-resident node area is followed by a space large enough for the longest disc-resident path. This space begins on the page boundary following the last memory-resident node. Thus, the beginning of the first disc node (the leftmost disc node that is a son of the root) is aligned on the page boundary following the root when loaded into memory during execution. The rest of the disc nodes are not aligned to page boundaries when loaded into memory during execution. Disc-resident nodes are aligned to sector boundaries when they are written to disc during loading by MLLDR.

Memory Allocation



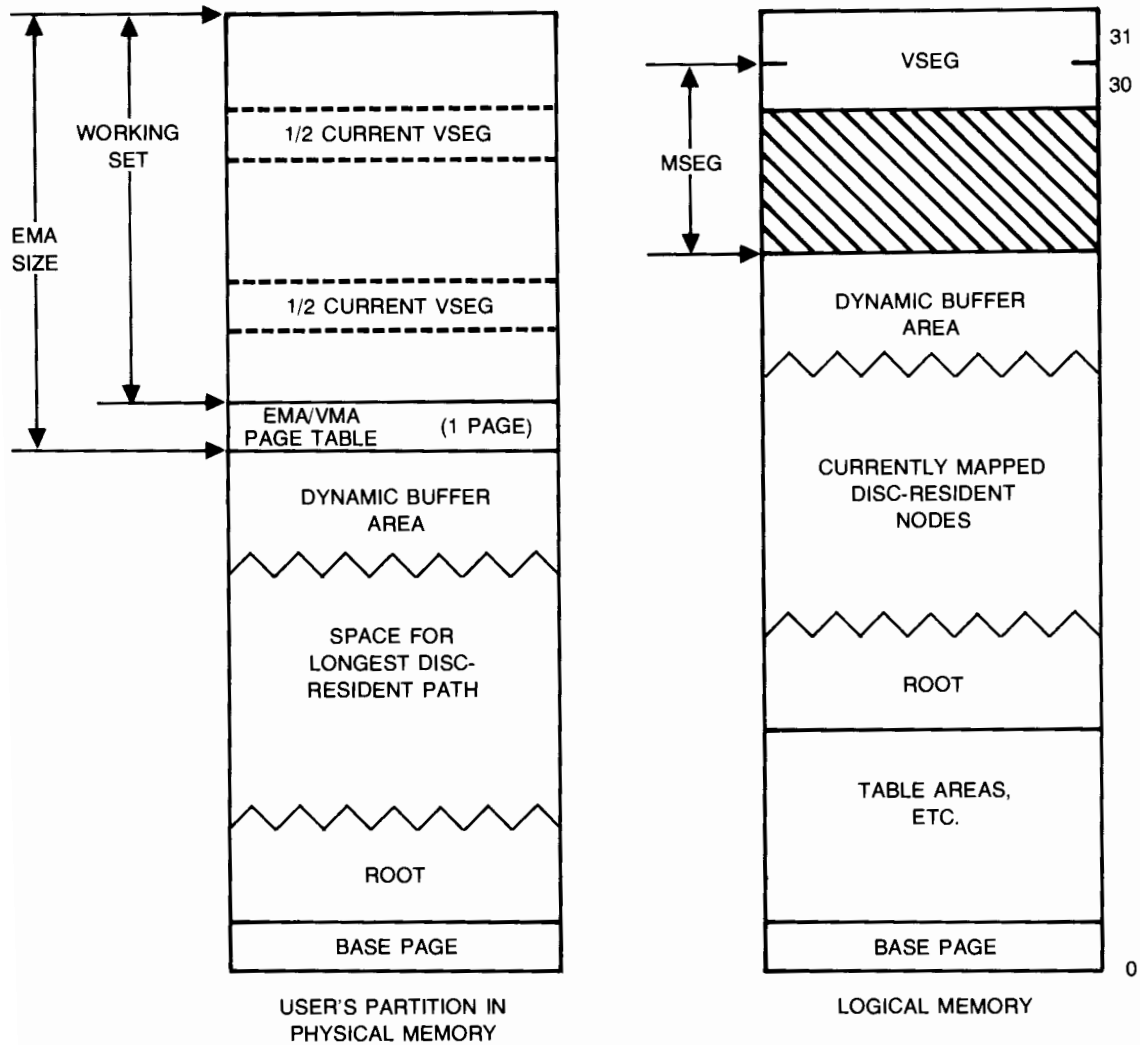
ALL HORIZONTAL STRAIGHT LINES ARE PAGE BOUNDARIES.

WAVY LINES ARE NOT NECESSARILY PAGE BOUNDARIES.

8100-60

Figure 7-1. Memory Map - Root Only

Memory Allocation



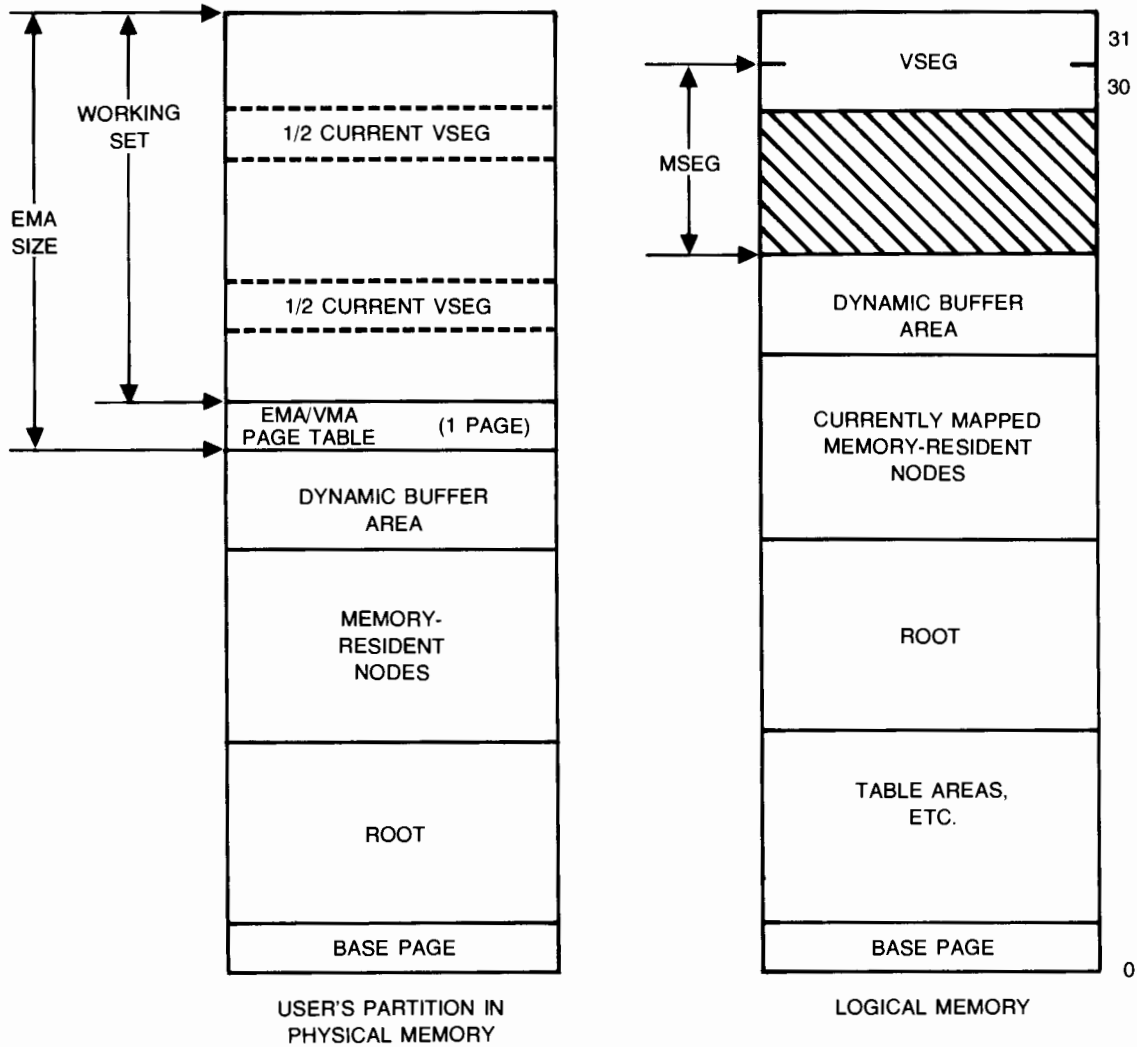
ALL HORIZONTAL STRAIGHT LINES ARE PAGE BOUNDARIES.

WAVY LINES ARE NOT NECESSARILY PAGE BOUNDARIES.

8100-61

Figure 7-2. Memory Map - Disc-Resident Nodes Only

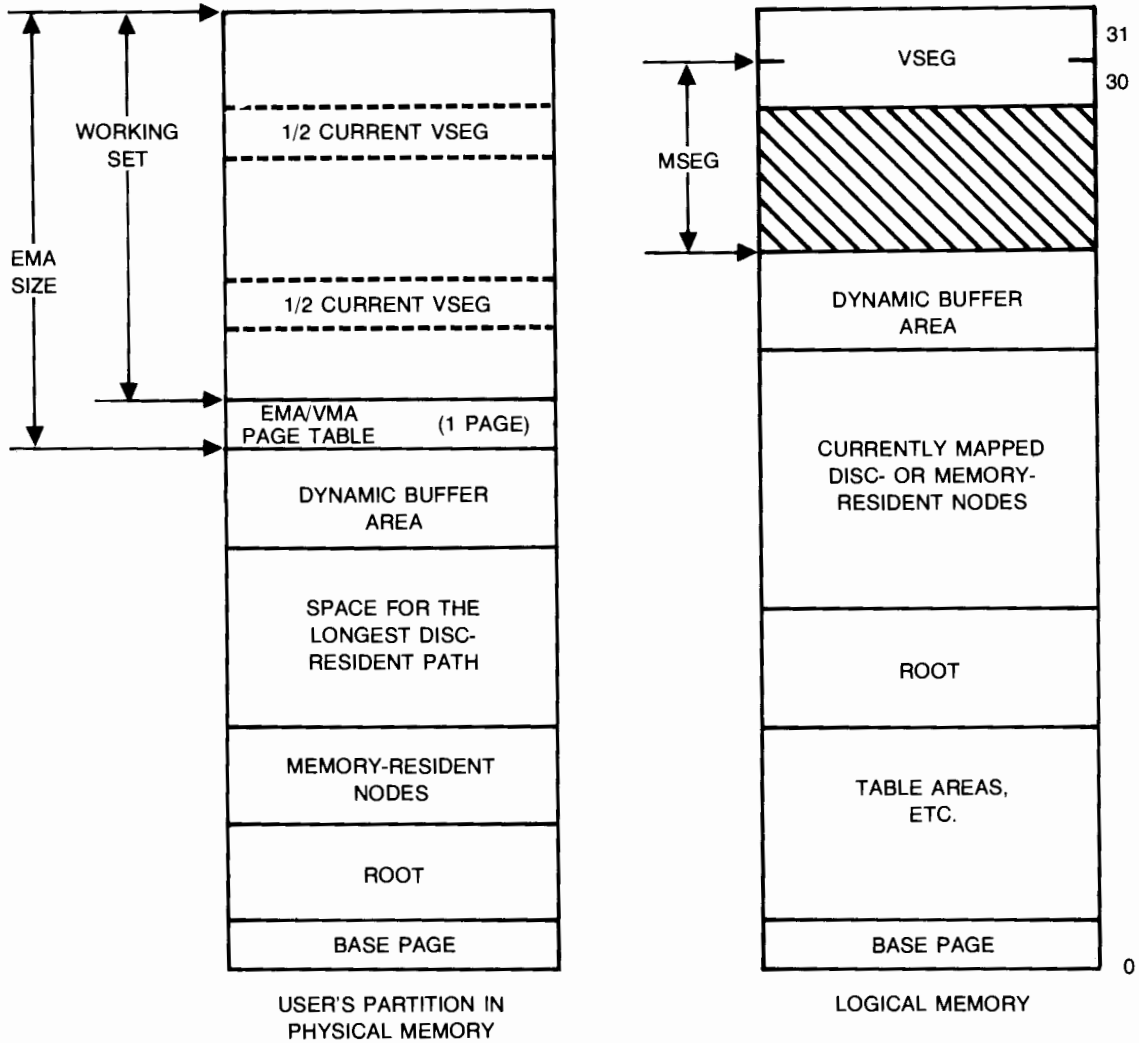
Memory Allocation



ALL HORIZONTAL STRAIGHT LINES ARE PAGE BOUNDARIES.
8100-62

Figure 7-3. Memory Map - Memory-Resident Nodes Only

Memory Allocation



ALL HORIZONTAL STRAIGHT LINES ARE PAGE BOUNDARIES.

8100-63

Figure 7-4. Memory Map - Memory-Resident and Disc-Resident Nodes

End of Program to End of Partition

Dynamic Buffer Area

The dynamic buffer area begins on either the next word or next page boundary following the space reserved for the program's longest path. Whether the dynamic buffer area is page-aligned depends on the type of nodes (disc-resident vs. memory-resident) used in the program. Refer to Table 7-1.

Table 7-1. Location of Dynamic Buffer Area

Nodes Used:	Dynamic Buffer Area Aligned to Page Boundary:
Root Only	No
Disc Only	No
Memory Only	Yes
Memory and Disc	Yes

VMA/EMA Area

VMA/EMA organization is the same regardless of the organization of nodes. Refer to Figure 7-1.

Memory organization is the same for both VMA and EMA programs. EMA is a subset of VMA in which the VMA size is equal to the working set size. The VMA/EMA page table, one page long, starts on the page boundary following the node area or dynamic buffer area, if used. The page table is used for both VMA and EMA programs. It does not occupy space in the program's logical address space. (The VMA/EMA system, however, maps the page table over a page of the program's map during execution when necessary.)

Following the page table is the working set or EMA, which extends to the length which was defaulted or specified in the program (EMA) or load (VMA). The largest size possible for this area is the size allowed by the largest partition in the system. If the partition size is larger than the current largest partition on the system, then the loader reports the warning W-RQ PGS, and the system can be reconfigured with a partition large enough to run the program. If the program could never fit, even after adding the maximum possible amount of memory and reconfiguring, then the loader reports the error L-RQ PGS.

Memory Allocation

MSEG is shown in the logical memory maps in Figures 7-1 thru 7-4. For both EMA and VMA programs, MSEG size is reported in the loader's completion message. This size is also reported by the system SZ command. MSEG size is calculated by the loader as follows:

MSEG size (in pages) = 31 - highest code page

Memory Overhead

Multilevel Segmentation

Use of multilevel segmentation adds a small amount of memory overhead to the program. When segmenting a very large program it may be helpful to know how much overhead is added to each node.

For each node the overhead for MLS is approximately:

8 words for each son node +
1 word for each routine called that is in a son node.

Profile Option

The profile option adds a small amount of overhead to the root node. Overhead for the profile option is approximately:

60-word subroutine added to the root +
(3 + number of nodes) words in the root.

Chapter 8

Loading Large Programs

The purpose of this chapter is to provide information that is helpful when loading very large programs using multilevel segmentation. The goal of the procedure suggested here is to load the program within the largest logical address space allowed by the system, and to avoid runtime errors that can be caused by improper segmentation. Some of the steps described here are useful when the user has limited knowledge of the program, as in the case when the program is being converted to run on an HP/1000 from another system.

It is recommended that SGMTR be used to create the initial command file. For many programs this is much faster than creating the segmentation structure manually. The user should be familiar with the operation of SGMTR, described in Chapter 6.

Compilation Considerations

Described below are steps that should be taken at compile-time for MLS loading of programs written in high-level languages. General information about programming practices that affect segmentation is contained in Chapter 3, "Language Considerations", and Chapter 6, "SGMTR - Segmentation Restrictions".

Note: For very large programs, placing separate compilation units in separate files will reduce the time required for edit and compiling.

FORTRAN

If the program uses COMMON, compile the program and organize SGMTR'd input file as described in Chapter 6, "SGMTR - Preparing the Input File". The main requirement is that all non-EMA, non-SAVE labeled COMMON blocks must be declared in a block data subprogram with the noallocate option.

If part of the program's data is in EMA, then FORTRAN's EMA transparency mode will be useful for routines that are passed both EMA and non-EMA parameters. However, this will increase the size of the code and may increase the path size necessary for loading the program.

Pascal

Use the LIBRARY option for all compilation units. This causes the compiler to create a separate relocatable module for each level-1 routine. (Pascal program must be compiled with the RTE-6/VM version of Pascal).

Pascal programs use dynamic buffer space for Heap 1 variables and for local data of recursive routines. The amount of dynamic buffer space required should be taken into account when specifying a path size to SGMTR, and the command "SZ,n", where n is the required number of pages, must be inserted at the beginning of the command file produced by SGMTR. For programs that contain memory nodes, at least one page of dynamic buffer space must be provided. For a program with only disc nodes, dynamic buffer space (less than one page) is automatically available to the program. For programs that do not use recursion or dynamic variables, use the compiler directives HEAP 0 and RECURSIVE OFF so that space is not required for HEAP/STACK initialization.

Evaluate the Program

If the user is not familiar with the structure of the program being segmented (the program, for example, is being moved from another computer), it may be helpful to obtain the information outlined below before attempting segmentation.

The number of routines - If there are more than 1000 routines, SGMTR may not have enough room in its symbol table to segment the program. The maximum number of modules that SGMTR can handle is 5000. If SGMTR reports a symbol table overflow, it may be necessary to run SGMTR on subtrees of the program separately, and segment manually at the higher levels.

The size of the code - Examine the compilation listing or SXREF output. Programs with many small subroutines are easier to segment than programs with large subroutines, since these may cause pathlength problems.

The size of data - If the data space required is 3K words or more, perhaps the data should be moved to EMA before segmentation is attempted.

Does the program use recursion or contain offpath references? There are segmentation structure restrictions in this case that will probably increase pathsize. Command files produced by SGMTR must be examined to ensure that these restrictions are not violated; "potential offpath reference" warnings, reported by SGMTR, should be eliminated. Refer to Chapter 3, "Segmentation Structure Restrictions". Also, Chapter 6 describes how SGMTR handles recursion and offpath references.

Are procedures passed as parameters? This programming practice has two effects on segmentation. SGMTR does not duplicate these routines, and these routines, along with all of their support routines, must reside in the same node or higher in the tree than the routine they are passed to. This can result in very large paths.

Segmenting Using SGMTR

The function of SGMTR is to create a preliminary command file and to quickly identify aspects of the program that affect its "loadability". For large programs, this method is faster than if the user creates the preliminary command file manually. The initial output from SGMTR will indicate whether the program has large paths that will make segmentation difficult, and it reports conditions that can cause load-time and run-time errors, such as potential offpath references and COMMON block type and size mismatches. The user must verify that restrictions on segmentation structure, described in this chapter and in Chapter 3, are not violated.

Prepare the input file for SGMTR as described in Chapter 6, creating a block data subprogram if necessary. For the first run of SGMTR, use one of the runstrings shown below. This gives SGMTR the best chance of success.

For EMA/VMA programs, use

```
RU,SGMTR,input,output,29,main,D
```

For non-EMA/VMA programs, use

```
RU,SGMTR,input,output,31,main,D
```

Specifying the largest possible pathlength gives SGMTR the best chance of success, and the pathsize can be reduced later if the first run is successful.

Selecting disc-resident nodes allows more nodes to occupy a path than memory-resident nodes, since memory nodes are aligned to page boundaries.

Allowing the default of EMA rather than using the VM option will allow the input a better chance to segment, since the VMA option requires more code overhead in the root node. (On the other hand, a VMA program can be run in a smaller partition.)

The profile and debug options should not be used in the first run of SGMTR; they add to path size.

If SGMTR creates the command file with no errors or warnings, and the program loads with no errors, then SGMTR can be run again, using a smaller path, memory-resident nodes, and other options to study the effect on program size, number of nodes, etc.

Loading Large Programs

If the command file contains errors or warnings, they should be corrected and SGMTR should be run again. Information about correcting path overflow errors is given below.

Errors related to pathlength that are most likely to occur during the segmentation and load process are the SGMTR error PATH OVERFLOW, and the MLLDR errors L-OV MEM (memory overflow) and L-OV BSE (base page overflow). The base page overflow error is included because correcting it by using mixed or current page links can affect path size. Figure 8-1 shows how these problems are interrelated. For many programs, the sequence of steps shown in Figure 8-1, and the information below, will help the user overcome common problems.

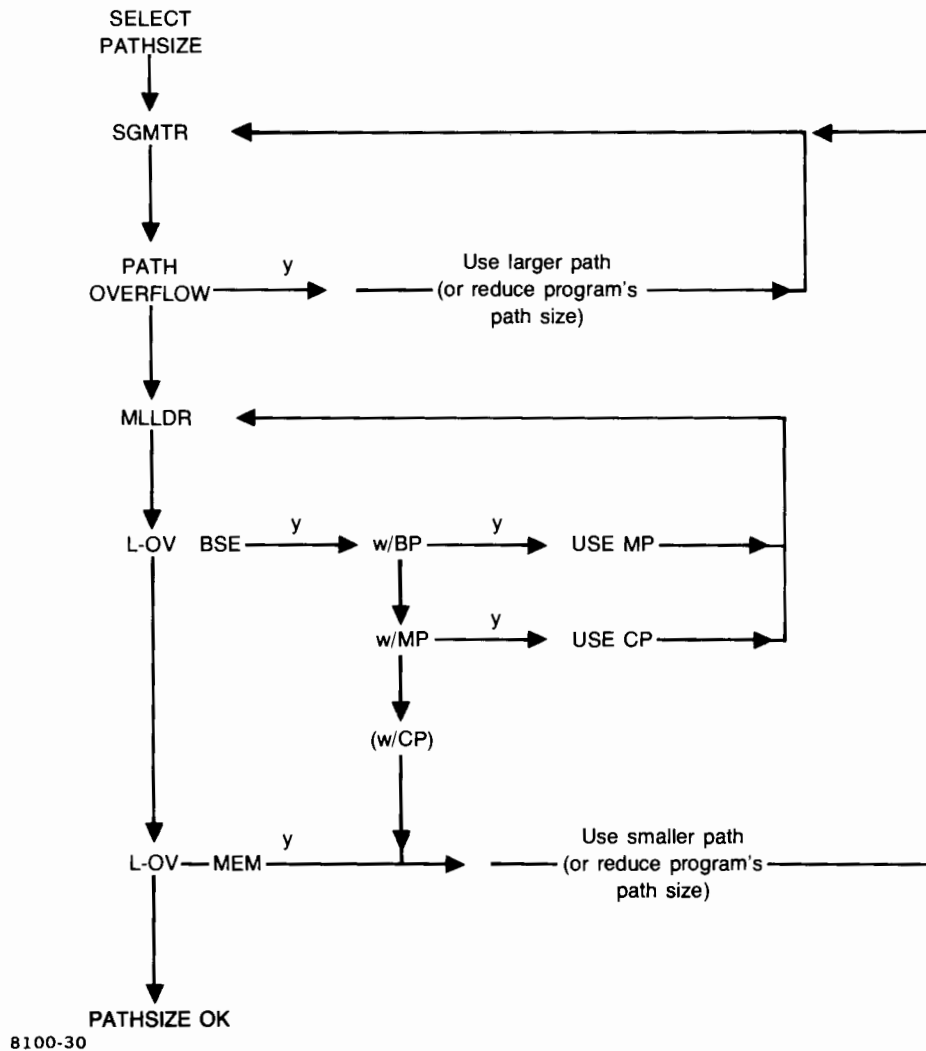


Figure 8-1. Steps to Correct Overflow Conditions

Loading Large Programs

Path Overflow

Examine the command file to determine the main cause of the path overflow. There may be, for example, one very large module or data area that caused the overflow. If the problem was not in the path from the root to where the overflow occurred, then examine the leaf node created after the overflow. A large module here, or collection of smaller modules, may have caused the overflow.

If more detailed path structure information is needed about the code in the leaf node following the overflow, then SGMTR can be run on this subtree of the program. This will identify the longest path in the subtree. Refer to the section "Main Entry Point" under the heading "Non-Main Type Modules" for information about segmenting a subtree. Another method is to generate a cross-reference listing using SXREF, and identify the longest path by examining the listing.

The section "Reducing Path Size", below, describes steps that can be taken to correct path overflow errors.

Note: In rare cases, a command file with a path overflow error will load without requiring a larger logical address space than was specified in SGMTR's runstring. The program may load, for example, when a memory overflow was expected. This is because SGMTR may overestimate the small amount of code overhead required for the load-on-call mechanism. The overestimation occurs when SGMTR joins brother leaf nodes in an attempt to decrease the total program size. The overestimation may amount to a few ten's of words.

Load Time Errors

If the SGMTR output contains errors or warnings, load-time errors may occur. Undefined externals, path overflow messages, SAVE or EMA COMMON blocks not declared in the main, missing block data subprograms for named COMMON, and mismatched COMMON block types or sizes are all sufficient to cause a load to fail.

If SGMTR creates the command file without reporting any errors or warnings, then load-time errors can still occur. The most common load-time errors that can occur in this situation are described below.



Base Page Link Overflow

SGMTR specifies base page links in the command file it creates. If a base page link overflow occurs, change the command file to use mixed links. If a base page overflow still occurs, try current page links.

Another option is to segment the program specifying a smaller path. This may move routines that create links into another path (where they may use a different base page) reducing the number of links in the path where the overflow occurred.

Refer to the section "Reducing Links", below.

Memory Overflow

One cause of this error is if the user changes the command file to use mixed or current page links due to a base page link overflow. If current page links were used, try using mixed links. If this does not solve the problem, try segmenting specifying a smaller path, or reduce the size of longest path in the program.

The amount of space allocated at load time for current page links depends mainly on module size and page alignment of modules. Refer to the section "Reducing Links", below.

Use of the profile (PF) option can also cause memory overflow. When the profile option is used, MLLDR inserts in the root a table that is used to record the number of faults for each node. SGMTR cannot estimate the size of this table when it creates the command file. If the table is too large, it can cause a memory overflow error. The length of the profile table is approximately 3 + number of nodes in the program. Determine the size of the table for the program and add a buffer of that length into the main module. Run SGMTR again on the modified program. Then run MLLDR using the new command file and the original input file. (Note: the new command file may have more nodes than the original, so more space may be required for the profile table.)

Note: MLS and profile option overhead are described in Chapter 7.

Reducing Path Size

The following are steps that can be taken to reduce the program's path size:

1. Segment using disc nodes. Memory nodes require a larger path since they are aligned to page boundaries. Remember that memory-resident nodes cannot follow any disc-resident node in the command file.
2. If memory-resident nodes are used, edit the command file so that some paths (the longest paths) become disc-resident.
3. Place data so that it does not increase path size.
 - a. Put COMMON in EMA if the length of COMMON is greater than 3K words. Use of EMA requires 2 pages of path space, so more than 2 pages must be placed in EMA to gain path space for the program. The additional page is to allow for overhead for accessing EMA variables. The amount of overhead varies among different programs.
 - b. If less than 2 to 3 pages of data resides in EMA, either move it out of EMA, or move more data into EMA. The effect of this depends on the amount of overhead in the program for accessing EMA variables.
 - c. If VMA is not required, use EMA. (Don't use the "VM" option when running SGMTR). VMA requires about one extra page of code in the root node.
 - d. Move non-EMA, non-SAVE labeled COMMON blocks down to the lowest common predecessor of the nodes that access the COMMON blocks. This reduces the size of the paths that do not access the COMMON blocks. (SGMTR does this.)
 - e. Move data out of SAVE space where possible (SAVE space is allocated in the root). The data can be moved to a labeled COMMON block in a memory-resident path. (Follow the procedures for use of COMMON, described in the section "Preparing the Input File", so that the block will not be duplicated and will be placed by SGMTR in the proper node.) Another alternative is to move the data to EMA.

Loading Large Programs

4. If memory nodes are being used, move routines that reference no other modules (these will be in leaf nodes in command files produced by SGMTR) into unused space at the end of memory nodes. Choose routines that are in the path that is too long. If routines are moved into a memory-resident node without extending the size of the node over the next page boundary, then the size of that node will not be increased. Note: enough space must be left for code added at load time for the load-on-call mechanism. Refer to Chapter 7, "MLS Overhead".
5. Change the code for a nonduplicatable module so that the module can be duplicated. If routines are being passed as parameters, the program can be changed so that data values are passed instead. For assembly programs, move the referenced data into a separate module. This allows the new module (containing only code) to be duplicated, possibly shortening one or more paths.
6. For programs written in high-level languages, there may be compiler directives that can be used to reduce the size of program code. Refer to the appropriate language manual.
7. Break routines in the largest path into smaller routines. This can move code out of the longest path and into other, smaller paths.

Reducing Links

Allocation of links at load time depends on such factors as module size, the order in which modules are loaded, page alignment of modules, and the type of linking specified to the loader. When a base page link overflow occurs, it is not always possible to use mixed or current page links because this may cause a memory overflow error. In this case steps can sometimes be taken to reduce the number of links required by the program.

The main point to remember is that instructions can directly access only data located on the current page or base page. Thus, for example, if a module is loaded so that it is split across a page boundary, then many links may be required for that module. The worst case would be if many symbols were defined at the beginning of the module, and references to those symbols were made later in the module (on the next page), or vice-versa. All of these references would require links.

Eliminating the type of situation described above can greatly reduce the number of links required by the program, possibly solving the link overflow (or, for current page links, memory overflow) condition. Use the LE (list entry points and base page links) option to find one or more modules that are using many links. Then try using the LO command to align the start of the module to the nearest page boundary. It may be possible to reorder modules so that small modules fill the space before the page boundary. Try loading the program again and examine the loadmap to see how many links were eliminated.

Runtime Errors

Improper segmentation can cause runtime errors. Most of the conditions that cause these errors are detected by SGMTR, and should be corrected before loading the program. Note, however, that a routine's use of local data cannot be detected by SGMTR.

Local Data (Disc Nodes and Duplicated Modules)

If the program contains routines that are written to make use of local data values preserved from previous calls to those routines, then runtime errors can occur if these routines reside in disc-resident nodes or if they are duplicated in the MLS tree. Note also that a COMMON block located in a disc node will not retain its values when a new copy is brought in from disc.

SGMTR cannot detect routines that require data values to be preserved between calls, so it may duplicate these routines. Also, when disc nodes are specified in SGMTR's runstring, all modules except the root are placed in disc nodes.

For COMMON blocks (labeled, Non-EMA, Non-SAVE) that must retain their values throughout execution of the program, modify the command file, if necessary, so that the COMMON blocks are not located in disc nodes.

For routines that require that local data be preserved between calls, there are several ways to ensure that the program executes properly. In FORTRAN, the variables can be declared to be SAVE variables. Another solution is to place the routines in a memory path, and not duplicate the routines. In assembly programs, the data and code can be split into separate modules; SGMTR will not duplicate the data modules. If disc nodes are specified, the data modules must be moved into memory nodes.

For more information see the sections "Data Considerations" and "Language Considerations" in Chapter 3.

Improper Use of COMMON

Mismatches of COMMON block types and sizes, and labeled COMMON without a block data subprogram can cause runtime errors. SGMTR detects these conditions and issues error messages.

Offpath References

An offpath reference error occurs when a routine terminates and control is passed to a location in the wrong path in the tree. The result of this is "runaway control", which is unpredictable (a MP or DM error could occur, for example), so offpath reference errors are hard to diagnose.

Potential offpath references, usually caused by recursion, are detected by SGMTR and SXREF. The command file should be modified so that no offpath references can occur. The recursive routine and all of its support routines must reside in the same path. When indirect recursion is used, all routines that are used recursively, along with all their support routines, must reside in the same path. See the example in Chapter 3, "Segmentation Structure Restrictions". SXREF can be used to verify that all potential offpath references have been eliminated.

Reducing Load Time

MLLDR will run faster if the following steps are taken.

1. Index library files using INDXR. The load will go faster when using one indexed merged library file rather than several separate files that are not indexed.
2. Do not use SGMTR's "A" option. This reduces the time it takes MLLDR to search son nodes to satisfy undefined externals.
3. Remove comments from the command file. This reduces the time it takes for MLLDR to search son nodes.
4. Modify the command file to use REs instead of NAs. Library searches take a significant amount of time.

Chapter 9

Relocating Loader LOADR

The functions and operation of LOADR are the same as MLLDR with the exceptions described in this chapter. Single-level segmentation is described here, as well as the differences in operation (command entry and code input) for LOADR. The section Memory Allocation shows the memory layout for single-level segmentation programs loaded using LOADR. LOADR functions not related to segmentation are described in Chapter 2. Use Chapter 5 for reference to find information about a particular command. For reference information about LOADR errors, refer to Appendix A. The error descriptions in Appendix A apply to both loaders except when otherwise noted.

Single-Level Segmentation

The relocating loader LOADR performs the same functions as the MLS loader MLLDR with the exception of segmentation; LOADR performs single-level segmentation. In a program using single-level segmentation, the main program includes EXEC 8 segment load calls or calls to the system routine SEGLD, specifying the name of the segment. Pascal programs include calls to the Pascal library routine @SGLD.

The segment load call brings the segment in from disc, overlaying the current contents of the segment area of the program. Only the main program can request a segment load. For programs that call EXEC 8 or the system routine SEGLD, after the segment is brought into memory, control is passed to the first statement in the segment. For Pascal programs, control is returned to the statement following the segment load request, and routines in the segment can then be called.

Relocating Loader LOADR

When a program using single-level segmentation is compiled or assembled, a module with NAM type 5 is generated for each segment. The name of the segment is contained in the NAM record. When LOADR encounters a type 5 module, it finishes processing the main (or current segment) searching user libraries and the system library to satisfy undefined external references. It then begins loading the segment. The relocation base for the segment area of the program is set at this point, and no more modules will be relocated in the main program area. Thus, all unsatisfied external references from the main must be satisfied by subsequent modules loaded with segments. Undefined external references in a segment must be satisfied before proceeding to the next segment.

After the load is complete, LOADR sets up an ID segment for the main, and one short ID segment for each of the segments. To save the program use the File Manager SP command for the main and each of the segments. This stores the main and each segment in a separate type 6 (memory image format) file. Each file is given the name of the main or segment it contains. Note that when the user logs off, File Manager only removes ID segments for main programs associated with the user's session; short ID segments for program segments are not removed.

During program execution, when a segment overlay request is made, the system routine SEGLD looks for the short ID segment corresponding to the segment that is to be loaded. If the short ID is found, the segment is overlaid in the program's segment area of memory. If the short ID is not found, then SEGLD schedules the short ID segment handler T5IDM, which attempts to find the segment's type 6 file and create a short ID segment.

More information about using single-level segmentation is contained in the Programmer's Reference Manual.

Operation

The main differences in operation for LOADR are command entry, code entry, and the completion message. These differences are described here. Other aspects of LOADR operation are the same as for MLLDR and are described in Chapter 4.

Runstring

The runstring contains the same parameter options as MLLDR except the last parameter, the profile option. The format of the runstring is as follows:

```
RU,LOADR,[command[,input[,list[,opcode[,format[,ptn[,sz]]]]]]]
```

For all programs loaded using LOADR, the command and input parameters can be any namr (LU or file). Note: The default opcode setting is BGNCTE. The default format setting is BP.

For detailed information about use of runstring parameters, refer to Chapter 4.

Command Input

When using LOADR, all commands can be entered interactively or from an LU. A command file is not required.

Commands

A summary table of LOADR commands is presented in Table 9-1. Refer to Chapter 5 for information about the format and function of each command. Note: The default opcode setting is BGNCTE. The default format setting is BP.

Relocating Loader LOADR

Table 9-1. Summary of LOADR Commands

Group 1: Set conditions for the load.

LL Set list namr
OP Set opcode - default is BGNCTE
Program type:
BG Background
RT Realtime
LB Large Background
EB Extended Background
COMMON type:
SC system
RC reverse system
NC local or no COMMON
SS Subsystem Global Area (SSGA)
Load type:
TE temporary
PE permanent
RP replacement
EMA type:
VM virtual memory
EM EMA
FM Set format code - default is BP
DB append debugger
LE list entry points and base page linkages
NL no listing
DC don't copy
Linkage type:
BP base page
CP current page
MP mixed
AS Assign partition
SZ Set program size
Note: the "SZ,+n" form of the command is not allowed by LOADR. The result of this form, if used, is the same as "SZ,n".
SH Set shareable EMA label
SA Allocate SAVE area
VS Set VMA size
WS Set working set size
LI Specify file as user library file

Relocating Loader LOADR

Group 2: Relocation

RE Relocate code in namr
LO Set relocation base
Note: the "LO,+n" form is not allowed by
LOADR. The result of this form, if
used, is the same as "LO,n".
SE Search namr or system library
MS Multiple search of namr
LI Specify file as user library file
SL Search user library files

Group 3: Miscellaneous

EC Echo commands to list device
DI Display undefined externals
FO Force load the remainder of the program;
ignore undefined externals
TR Transfer to command file
* Comment

Group 4: Termination commands. A command in this group will be the last command to be processed in a load.

EN, EX, /E Finish processing the load
AB, /A Abort the load

Code Input

When loading using LOADR, all code can be input from an LU. (When using MLLDR the command "RE,<lu>" is not legal from a disc command file.) Relocate and search commands can specify an LU, and an LU for code input can be specified in the runstring. Note however that a multiple search command (MS, or SEA, for example) will only perform a single search when reading from an LU.

Completion Message

The format of the LOADR completion message is as follows:

For all programs,

```
ww PAGES RELOCATED
xx PAGES REQUIRED
```

followed by,

For EMA programs:

```
DEFAULT EMA or yy PAGES EMA
zz PAGES MSEG
```

For VMA programs:

```
yy LAST PAGE OF VMA
zz PAGES MSEG
nn PAGES WORKING SET
```

followed by:

```
LINKS:pp PROGRAM:ss LOAD:tt COMMON:uuuu vv qq SHARE:rr
/LOADR:name READY AT HR:MIN AM/PM day, date, year
/LOADR:$END
```

where:

- ww = The number of pages occupied by the program's longest path, determined by the size of the longest segment. Includes base page.
- xx = Size in pages of the partition required by the program.
- yy = For EMA programs, the EMA size in pages. For VMA programs, the last page of VMA (VMA size in pages is yy + 1).
- zz = For EMA or VMA programs, the MSEG size in pages.
- nn = For VMA programs only, the working set size in pages.
- pp = CP, MP, or BP for current page, mixed page, or base page, depending on which option was used. Default is BP.

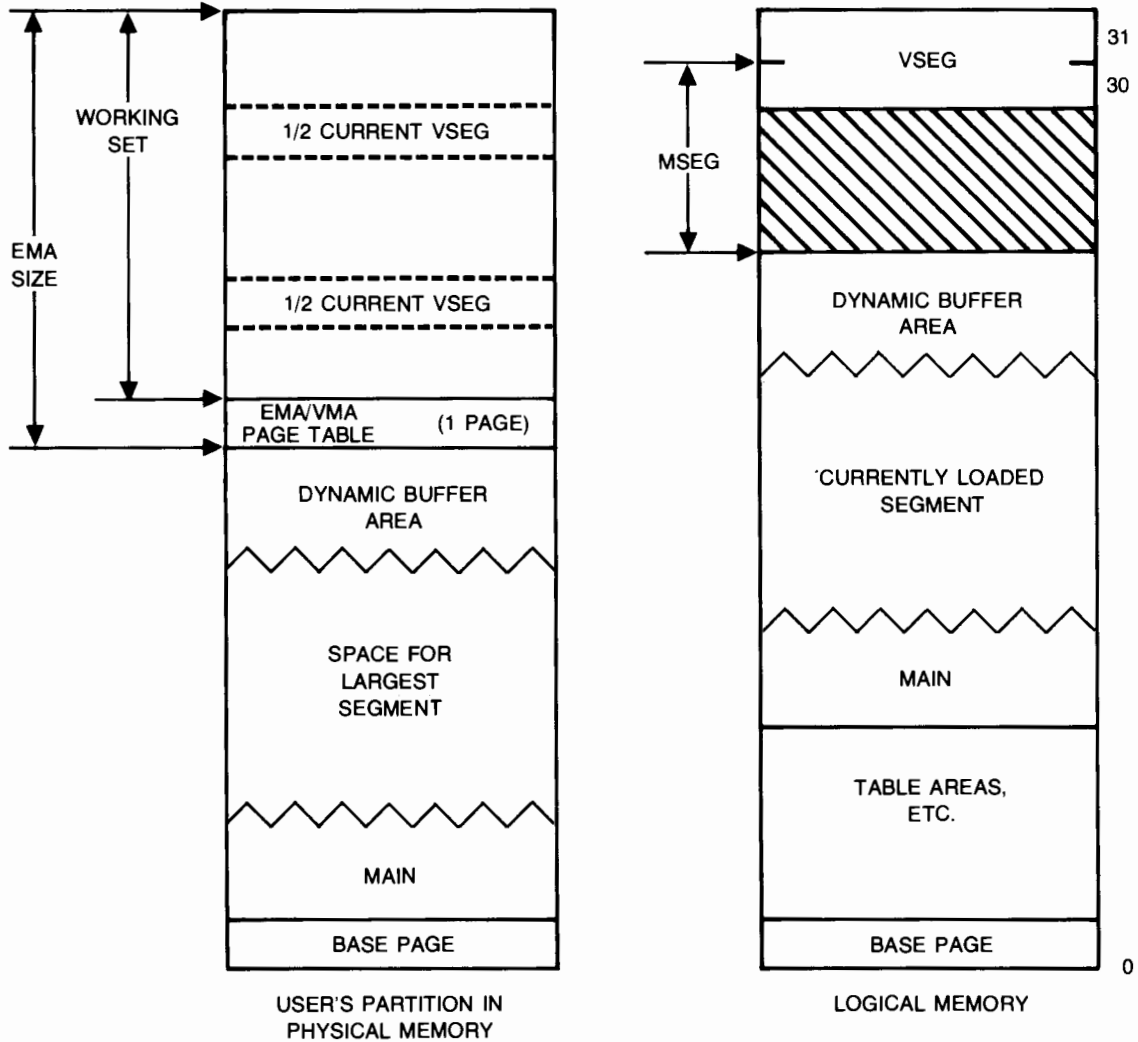
Relocating Loader LOADR

- ss = Program type, RT, BG, LB, or EB. Default is BG.
- tt = TE for temporary program. PE for permanent program. RP for replacing a permanent program. Default is TE.
- uuuu = SC, RC, or NC for system COMMON, reverse system COMMON, or no system COMMON, respectively, followed by SS for Subsystem Global Area (SSGA) access or blanks if SSGA is not used.
- vv = DB if DEBUG appended.
- qq = FO if force load, blank if not.
- rr = Label identifying shareable EMA area. If shareable EMA is not used, SHARE:rr will not be printed.
- name = Name of the main program, which is ready to run when the loader terminates.

Memory Layout

In the memory organization for a segmented program loaded using LOADR a segment overlay area is reserved between the main program and the dynamic buffer area (if used). The size of the overlay area is the size of the largest segment. See Figure 9-1 for a diagram of memory organization for LOADR.

Relocating Loader LOADR



ALL HORIZONTAL STRAIGHT LINES ARE PAGE BOUNDARIES.

WAVY LINES ARE NOT NECESSARILY PAGE BOUNDARIES.

8100-29

Figure 9-1. Memory Map - Single-Level Segmentation

Converting to Multilevel Segmentation

In some cases the user may want to convert a program from single-level segmentation to multilevel segmentation. Advantages of multilevel segmentation are described in Chapter 2.

FORTRAN and Assembly Language

Convert program segments (type 5 modules) into subroutines, and convert their respective EXEC 8 or SEGLD calls into subroutine calls. Note: Data references to a son node are not allowed.

Pascal

Delete the \$SEGMENT compiler directive from all segments, and move all code (procedure and function declarations) from the segments into the declaration section of the main program. Be sure that global declarations for segments match global declaration for the main. Delete all segment load calls from the main program. Use the \$LIBRARY compiler directive. The program can now be compiled in one unit and segmented in MLS format.

Appendix A

Errors

MLLDR and LOADR Errors

" "

L-CK SUM

THIS IS A CHECKSUM ERROR. A FILE SPECIFIED TO THE LOADER THAT DID NOT CONTAIN RELOCATABLE FORMAT CODE. A TYPICAL MISTAKE IS SPECIFYING THE SOURCE FILE NAME INSTEAD OF THE BINARY FILE NAME. IF THE FILE SPECIFIED WAS CORRECT, THAT FILE HAS BEEN OVERLAID OR CORRUPTED. PURGE THAT FILE, RECOMPILE THE ORIGINAL SOURCE AND TRY AGAIN.

" "

L-CM BLK

THIS IS A COMMON BLOCK ERROR. THIS ERROR OCCURS IF THE LARGEST BLANK COMMON DECLARATION OF A PROGRAM DOES NOT APPEAR IN THE FIRST MODULE OF THE PROGRAM LOADED. PROGRAMS THAT USE BLANK COMMON MUST DECLARE THAT COMMON IN THE FIRST ROUTINE LOADED AND THAT COMMON DECLARATION MUST BE THE LARGEST ENCOUNTERED IN THE LOAD. THIS ERROR IS ALSO GENERATED IF THE AMOUNT OF COMMON REQUIRED EXCEEDS THAT AVAILABLE. ALSO, A MODULE CONTAINING BLANK COMMON CANNOT BE LOADED AFTER AN

" "

L-CO RES

YOU TRIED TO REPLACE OR PURGE A MEMORY-RESIDENT PROGRAM. THIS IS ILLEGAL.

" "

L-DU ENT

DUPLICATE ENTRY POINT. GENERALLY THIS OCCURS WHEN THE SAME SUBROUTINE WAS LOADED TWICE. ALTERNATELY YOU NAMED A SUBROUTINE WITH THE SAME NAME (ENT IN ASMB) USED SOMEWHERE ELSE WITHIN THE PROGRAM. CONFUSION SOMETIMES OCCURS WITH SEGMENTED PROGRAMS. A SUBROUTINE LOADED WITH THE MAIN MUST NOT BE AGAIN LOADED WITH A SEGMENT. LOOK AT THE LOAD MAP FOR THE LOAD. DID YOU TRY TO LOAD THE SUBROUTINE WITH A SEGMENT WHERE THAT SUBROUTINE WAS ALREADY LOADED WITH THE MAIN? THE LOAD MAP WILL LIST ALL SUBROUTINES LOADED WITH THE MAIN. IN AN MLS PROGRAM THIS ERROR OCCURS WHEN THE SAME ROUTINE IS LOADED TWICE IN THE SAME PATH. THIS ERROR ALSO OCCURS FOR MLS PROGRAMS WHERE THE MAIN CONTAINS A COMMON BLOCK DECLARED ALLOCATE, AND A SON CONTAINS A NA,<COMMON BLOCK NAME>, AND THE COMMON BLOCK IS DECLARED NO-ALLOCATE IN THE SON.

Errors

" "

L-DU PGM

THIS PROBLEM RESULTS WHEN YOU TRY TO LOAD THE SAME PROGRAM SEVERAL TIMES BUT DO NOT GET RID OF THE EARLIER LOADS. FOR EXAMPLE, YOU LOADED A PROGRAM CALLED XXXXX AND FOR SOME REASON LOADED THE SAME PROGRAM AGAIN. IN THIS CASE THE LOADER WARNED YOU WITH A W-DU PGM WARNING MESSAGE AND THEN RENAMED YOUR PROGRAM TO ..XXX. YOU HAVE NOT LOADED A PROGRAM WITH THE SAME NAME A THIRD TIME. THE LOADER WILL NOT RENAME THE PROGRAM AGAIN. TO REMOVE THE TWO COPIES OF THE PROGRAM USE:

```
:OF,XXXXX,8
:OF,..XXX,8
```

AND START THE LOAD OVER AGAIN.

" "

L-EX CPY

ATTEMPT TO REPLACE OR PURGE A PROGRAM WHERE COPIES OF THAT PROGRAM EXISTED. IT IS NOT POSSIBLE TO REPLACE OR PURGE A PROGRAM FROM THE SYSTEM IF COPIES OF THAT PROGRAM EXIST. THE PROBLEM HERE IS THAT OTHER COPIES OF THE SAME PROGRAM EXIST AND MAY BE IN USE. THE PROPER COURSE HERE IS TO DO AN 'OF,PROG,8' ON ALL THE PROGRAMS LISTED AS COPIES. THIS WILL GET RID OF THOSE PROGRAMS SO THAT YOU CAN PERFORM THE PROGRAM PURGE OR REPLACE. NOTE THAT THIS PROCESS SHOULD ONLY BE DONE BY THE SYSTEM MANAGER.

" "

L-ID EXT

NO ID EXTENSIONS AVAILABLE FOR THE EMA PROGRAM. YOU MUST FREE UP SOME ID EXTENSIONS BEFORE THE EMA PROGRAM CAN BE SUCCESSFULLY LOADED.

" "

L-IL ALC

EXTERNAL REFERENCES TO NAMED COMMON WHICH APPEAR IN AN ALLOCATE RECORD ARE NOT ALLOWED BEFORE THE ALLOCATE RECORD OCCURS.

" "

L-IL CMD

ATTEMPT TO PURGE A PROGRAM UNDER BATCH OR ATTEMPT TO USE THE PU COMMAND WITHIN A LOADER COMMAND FILE. THE PU COMMAND IS NOT ALLOWED IN A LOADER COMMAND FILE.

" "

L-IL DRN

ILLEGAL DISC-RESIDENT NODE SPECIFICATION. DISC-RESIDENT NODES MUST START AT THE FIRST NODE BELOW THE ROOT. ALSO, ALL MEMORY-RESIDENT NODES MUST OCCUR BEFORE ALL DISC-RESIDENT NODES IN PREORDER.

Errors

" "

L-IL EMA

TRIED TO USE SHAREABLE EMA WITH THE OLD EMA RELOCATABLE FORMAT. THIS WILL NOT WORK WHEN A SHAREABLE EMA LABEL FILE EXISTS ON LU 2. ANOTHER CAUSE IS MIXING OLD AND NEW EMA RELOCATABLES.

" "

L-IL MLS

THIS ERROR OCCURS DURING PRELIMINARY CHECKING OF THE COMMAND FILE. THE CHECKS THAT ARE PERFORMED ARE:

1. COMMAND SYNTAX IS CORRECT.
2. COMMAND ORDER IS CORRECT.
3. REFERENCED FILES ARE ACCESSABLE.
4. NODES ARE DEFINED IN PREORDER.
5. DISC-RESIDENT NODE DEFINITIONS FOLLOW MEMORY-RESIDENT NODE DEFINITIONS.
6. THE FIRST DISC-RESIDENT NODE DEFINITION OCCURS AS A SON OF THE ROOT NODE.
7. THE ROOT NODE IS A MEMORY-RESIDENT NODE.
8. THE TOTAL NUMBER OF NODES IS NO MORE THAN 1024.
9. THE MAXIMUM NODE DEPTH IS NO MORE THAN 31.

AN L-IL MLS ERROR ALSO OCCURS WHEN AN INSTRUCTION OTHER THAN A JSB (A SUBROUTINE CALL) IS USED TO ACCESS A SYMBOL IN A SON NODE. THIS ERROR OCCURS DURING THE LOAD, NOT DURING PRELIMINARY COMMAND FILE CHECKING. THIS CONDITION IS REPORTED AS FOLLOWS:

```
/MLLDR: <16-BIT NON-JSB INSTRUCTION>  
/MLLDR: <ADDRESS OF NON-JSB INSTRUCTION>  
/MLLDR: <ENTRY POINT NAME BEING REFERENCED>  
/MLLDR: L-IL MLS
```

" "

L-IL PRM

THE RUNSTRING OR A COMMAND IN A COMMAND FILE CONTAINED AN ERROR.

" "

L-IL PTN

YOU SPECIFIED A PARTITION IN THE LOAD OF THE PROGRAM, HOWEVER, THAT PARTITION DOES NOT EXIST OR HAS BEEN DOWNED DUE TO A PARITY ERROR. TRY AGAIN, THIS TIME SPECIFY A PARTITION THAT EXISTS OR DON'T SPECIFY ANY PARTITION AT ALL.

" "

L-IL REC

THE LOADER FOUND A RECORD THAT WAS NOT A NAM, ENT, EXT, DBL, EMA, GEN, LOD, END RECORD, OR EXTENDED RECORD. THE CHECKSUM WAS OK BUT THE RECORD WAS NOT IDENTIFIED. WAS THE FILE SPECIFIED A RELOCATABLE FILE? TRY RECOMPILING AND LOADING.

Errors

""

L-IL REL

THE COMPILER PRODUCED AN ILLEGAL RECORD. ONE OF THE FOLLOWING OCCURRED: THE NUMBER OF ENTRIES SPECIFIED IN AN ENT OR EXT RECORD WAS ZERO. THE NUMBER OF INSTRUCTION WORDS SPECIFIED IN A DBL RECORD WAS ZERO. A RELOCATABLE INDICATOR IN A DBL RECORD WAS SEVEN. A DBL RECORD WAS PRODUCED THAT REFERENCED AN EXTERNAL BUT THAT EXTERNAL WAS NOT IN ANY OF THE EXT RECORDS. SOMETHING OTHER THAN AN IDR RECORD WAS FOUND IN AN INDEXED LIBRARY'S DIRECTORY. RECOMPILE AND TRY AGAIN. THIS COULD BE A COMPILER BUG.

""

L-IL RPL

TRIED TO DO A JSB TO A USER-SPECIFIED RPL IN A SON NODE. FOR EXAMPLE, NODE M.1 HAS A JSB Q AND NODE M.1.1 HAS A NA,Q OR RE,%Q WHERE Q IS AN RPL.

""

L-IL SCB

ILLEGAL SESSION CONTROL BLOCK VALUE (NEGATIVE CAPABILITY LEVEL).

""

L-IL SEG

ILLEGAL SEGMENT SPECIFICATION. YOU TRIED TO LOAD A MODULE WITH A TYPE 5 NAM USING MLLDR.

""

L-IN CAP

ATTEMPT TO LOAD, PURGE, OR REPLACE A PERMANENTLY LOADED PROGRAM WITHOUT HAVING A SESSION CAPABILITY LEVEL HIGH ENOUGH TO PERFORM THIS FUNCTION. THESE FUNCTIONS REQUIRE A CAPABILITY LEVEL OF 60 OR HIGHER.

""

L-LM LIB

THE LIMIT ON THE NUMBER OF LIBRARIES SPECIFIED BY THE 'LI' COMMAND HAS BEEN EXCEEDED. YOU MAY SPECIFY 10 LIBRARIES. INSTEAD OF SPECIFYING ANOTHER LIBRARY YOU CAN SPECIFICALLY DO AN 'SE' OF THE FILE.

""

L-ML BDT

MULTIPLE BLOCK DATA SUBPROGRAM. ATTEMPTED TO INITIALIZE THE SAME AREA MORE THAN ONCE.

""

L-ML EMA

ILLEGAL EMA DECLARATION. TWO DIFFERENT EMA LABELS WERE USED, OR THE EMA DECLARATION WAS NOT MADE IN THE MAIN OF A PROGRAM AND THAT MAIN LOADED FIRST, OR AN EMA LABEL WAS ALSO DECLARED AS AN ENTRY POINT IN ANOTHER MODULE. THE EMA DECLARATION MUST BE IN THE MAIN OF THE PROGRAM AND THAT MAIN MUST BE THE FIRST MODULE LOADED. THE EMA STATEMENT MUST BE IN ANY SEGMENT OR SUBROUTINE REFERENCING ANY ELEMENT IN EMA.

Errors

" "

L-NO IDS

NOT ENOUGH ID SEGMENTS TO FINISH THE LOAD. YOUR SYSTEM HAS RUN OUT OF ID SEGMENTS. CALL THE SYSTEM MANAGER TO FREE UP SOME ID SEGMENTS. HE WILL PROBABLY USE THE OFF COMMAND TO PURGE SOME PROGRAMS FROM THE SYSTEM.

" "

L-NO RSG

NO ROOT SEGMENT SPECIFIED. AN MLSLOC PROGRAM MUST HAVE A MEMORY RESIDENT ROOT SPECIFIED BY THE COMMAND M. CHECK YOUR LOADER COMMAND FILE TO MAKE SURE THAT THE FIRST NODE SPECIFICATION IS M AND THAT THE M OCCURS BEFORE ANY RELOCATION OCCURS.

" "

L-NO SNP

NOTIFY YOUR SYSTEM MANAGER. THE LOADER COULD NOT FIND THE FILE \$SYENT.

" "

L-OV BSE

BASE PAGE OVERFLOW. THIS PROGRAM HAS USED TOO MANY BASE PAGE LINKS. IF THE CP (OR MP) OPTION WAS NOT USED, TRY USING IT TO PUT LINKS ON THE CURRENT PAGE INSTEAD OF ALL ON THE BASE PAGE. IF THE CP OPTION WAS USED, RELOAD THE PROGRAM BUT THIS TIME SPECIFY THE 'OP, LE' OPTION. THIS WILL LIST ALL ENTRY POINTS AND THE BASE PAGE LINKAGES. THIS LOAD WILL ALSO FAIL, HOWEVER, NOW YOU KNOW WHICH MODULES ARE USING UP ALL THE LINKS. BY USING THE LO,XXXXX COMMAND AND ALIGNING THOSE MODULES TO PAGE BOUNDARIES THE LINKAGE NEEDS CAN BE REDUCED. ALTERNATELY, YOU MAY WISH TO REARRANGE THE LOADING ORDER OF YOUR SUBROUTINES. THIS MAY IMPROVE (OR MAKE WORSE) THE LINKAGE NEEDS OF YOUR PROGRAM.

" "

L-OV DSK

YOUR PROGRAM EXCEEDS THE MAXIMUM DISC SPACE ALLOWED A PROGRAM. IT OCCUPIES MORE 128-WORD SECTORS THAN CAN BE RECORDED IN ONE WORD OF THE ID SEGMENT.

" "

L-OV FIX

THIS IS A FIXUP TABLE OVERFLOW. THE LOADER NEEDS MORE ROOM FOR ITS INTERNAL SYMBOL TABLE AND FIXUP TABLE. SINCE LOADR IS A TYPE 4 PROGRAM IT CAN BE MADE AS LARGE AS THE LARGEST NORMAL BACKGROUND PARTITION. SINCE MLLDR IS A TYPE 6 PROGRAM, IT CAN BE SIZED TO 32 PAGES. TO GIVE THE LOADER MORE ROOM USE THE 'SZ' OPERATOR COMMAND:

```
        *SZ,MLLDR,XX
OR FROM *SZ,LOADR,XX
FMGR,
        :SYSZ,MLLDR,XX
OR :SYSZ,LOADR,XX
```

Errors

CONSULT THE TERMINAL USER'S REFERENCE MANUAL FOR MORE INFORMATION ON THE 'SZ' COMMAND. IF THE SZ COMMAND DOES NOT SOLVE THE PROBLEM, THEN TRY USING THE LOADER 'SE' COMMAND AFTER EVERY LOADER 'RE' COMMAND. THIS WILL REDUCE SPACE NEEDED FOR FIXUPS. IN ADDITION TRY LOADING A NUMBER OF SUBROUTINES (STILL DOING 'SE') BEFORE THE MAIN OF THE PROGRAM.

""

L-OV MEM

THE RELOCATION ADDRESS HAS EXCEEDED 77777B, 77777B - MSEG, OR THE SIZE YOU SPECIFIED USING SZ,N. THE SIZE OF THE CODE LOADED SO FAR EXCEEDS THE MAX SIZE THAT YOU SPECIFIED OR EXCEEDS THE LARGEST POSSIBLE SIZE FOR A PROGRAM. THE MAXIMUM SIZE FOR EXTENDED BACKGROUND (EB) NON-VMA/EMA PROGRAMS IS 32K WORDS (INCLUDING BASE PAGE), AND 30K FOR EB VMA/EMA PROGRAMS. THE MAXIMUM SIZE FOR LARGE BACKGROUND (LB) NON-VMA/EMA PROGRAMS IS 28K WORDS, AND 26K FOR VMA/EMA PROGRAMS. CONSULT THE GENERATION MAP FOR THE MAXIMUM SIZE OF REALTIME AND BACKGROUND PROGRAMS. IF YOUR PROGRAM IS TOO LARGE FOR ITS TYPE, TRY THE FOLLOWING:

1. IF POSSIBLE, CHANGE THE PROGRAM'S TYPE. FOR EXAMPLE, IF THE PROGRAM IS NOT TYPE 6 (EXTENDED BACKGROUND), MADE IT TYPE 6 BY USING THE 'OP,EB' COMMAND.
2. IF YOU SPECIFIED A SIZE, THEN DON'T SPECIFY A SIZE. THE LOADER WILL DO ALL IT CAN TO MAKE YOUR PROGRAM FIT.
3. SEGMENT THE PROGRAM.
4. IF THE PROGRAM IS ALREADY SEGMENTED, THEN TRY TO REDUCE THE SIZE OF THE LONGEST PATH (FOR MLS PROGRAMS) OR THE LONGEST SEGMENT (FOR TYPE 5 SEGMENTED PROGRAMS).
5. CHECK IF THERE ARE ANY DATA DECLARATIONS THAT CAN BE MOVED TO VMA/EMA.

""

L-OV PTN

THE SIZE SPECIFIED USING THE SZ COMMAND WAS TOO LARGE FOR THE PROGRAM'S TYPE. FOR THE SZ,N FORMAT OF THE SIZE COMMAND, THE LOADER ADDS THE REQUESTED NUMBER OF PAGES TO THE RELOCATION BASE FOR THE TYPE OF THE PROGRAM BEING LOADED. FOR THE SZ,+N FORMAT OF THE SIZE COMMAND, THE LOADER SUMS THE RELOCATION BASE, CODE SIZE, MSEG (IF USED), AND THE REQUESTED NUMBER OF PAGES. IN BOTH CASES, THE RESULT MUST BE 32K WORDS OR LESS. SPECIFY A SMALLER SIZE, OR TRY LOADING THE PROGRAM AS LARGE BACKGROUND OR EXTENDED BACKGROUND.

""

L-OV RBP

OVERFLOW OF ROTATING BASE PAGE (MLLDR ONLY). A MODULE TRIED TO RELOCATE ON BASE PAGE AND THE ORB INSTRUCTION COULD NOT BE PROCESSED WITHOUT CAUSING BASE PAGE TO ROTATE. TRY MOVING THE MODULE.

""

Errors

L-OV SAV

OVERFLOWED SAVE AREA. THE AMOUNT OF SPACE RESERVED BY THE SA COMMAND WAS INSUFFICIENT FOR ALL SAVE AREAS OF ALL MODULES OF THIS PROGRAM. RELOAD THE PROGRAM, SPECIFYING THE REQUIRED SIZE OF THE SAVE AREA.

""

L-OV SNP

OVERFLOWED SNAP FILE \$SYENT. TRY RUNNING THE LOADER USING "RU,LOADR,-1,-1" TO CREATE THE \$SYENT FILE.

""

L-OV SYM

THIS IS A SYMBOL TABLE OVERFLOW. THE LOADER NEEDS MORE ROOM FOR ITS INTERNAL SYMBOL TABLE AND FIX UP TABLE. SINCE LOADR IS A TYPE 4 PROGRAM, IT CAN BE MADE AS LARGE AS THE LARGEST NORMAL BACKGROUND PARTITION. SINCE MLLDR IS A TYPE 6 PROGRAM, IT CAN BE SIZED TO 32 PAGES. TO GIVE THE LOADER MORE ROOM, USE THE 'SZ' OPERATOR COMMAND. THAT IS,

*SZ,MLLDR,XX XX = # OF PAGES

OR *SZ,LOADR,XX

OR FROM FMGR,

:SYSZ,MLLDR,XX

:SYSZ,LOADR,XX

BY INCREASING THE SPACE FOR THE LOADER, THE L-OV SYM PROBLEM SHOULD BE SOLVED. CONSULT THE RTE-6/VM TERMINAL USER'S REFERENCE MANUAL FOR MORE INFORMATION ON THE 'SZ' COMMAND. IF THE SZ COMMAND DOES NOT SOLVE THE PROBLEM, THEN TRY USING THE LOADER 'SE' COMMAND AFTER EVERY LOADER 'RE' COMMAND. THIS WILL REDUCE SPACE NEEDED FOR FIXUPS. IN ADDITION TO USING THE 'SE' COMMAND AFTER EVERY 'RE' COMMAND, TRY LOADING A NUMBER OF YOUR SUBROUTINES (STILL DOING 'SE') BEFORE THE MAIN OF THE PROGRAM.

""

L-PE LDR

YOU TRIED TO DO A PURGE, REPLACE, OR PERMANENT LOAD WITH A COPY OF THE LOADER. RUN THE ORIGINAL LOADER (NOT A COPY): RU,MLLDR:IH OR RU,LOADR:IH.

""

L-RE SEQ

RECORD OUT OF SEQUENCE. THE LOADER WAS RELOCATING AND ENCOUNTERED RECORDS IN THE WRONG ORDER OR AN END RECORD WAS MISSING. GENERALLY THIS ERROR OCCURS WHEN RELOCATING FROM A TAPE, AND THE TAPE IS INCORRECTLY POSITIONED. IF THE RELOCATION WAS FROM A FILE, THE FILE IS PROBABLY CORRUPT. RECOMPILE THE SOURCE AND TRY AGAIN.

Errors

""

L-RF EMA

ATTEMPT TO ACCESS AN EMA EXTERNAL WITH OFFSET OR INDIRECT. IF THIS IS A FORTRAN PROGRAM YOU PROBABLY FORGOT TO PUT THE \$EMA STATEMENT IN A SUBROUTINE THAT ACCESSED AN EMA ELEMENT. IF THE PROGRAM WAS WRITTEN IN ASSEMBLY LANGUAGE, USE THE H-P SUPPLIED ROUTINES .IMAP AND .EMIO TO MAP IN THE ARRAYS AND THEN INDEX INTO THE ARRAY VIA THE ADDRESS RETURNED, NOT VIA A REFERENCE TO THE EMA LABEL.

""

L-RP CPY

ATTEMPT TO REPLACE A COPIED PROGRAM. YOU TRIED TO DO A PROGRAM REPLACE ON A PROGRAM THAT WAS A COPY OF ANOTHER PROGRAM. REPLACEMENT OPERATIONS CAN ONLY BE DONE ON THE ORIGINAL PROGRAM NOT THE COPIED PROGRAM. EITHER SPECIFY THE ORIGINAL PROGRAM, OR MAKE SURE THE NEW PROGRAM NAME IS THE SAME AS THE NAME OF THE ORIGINAL PROGRAM. EDIT THE SOURCE OF YOUR PROGRAM AND MAKE SURE THE NAME IS THE ORIGINAL PROGRAM NAME.

""

L-RP MLS

YOU TRIED TO USE MLLDR TO REPLACE A PROGRAM THAT WAS LOADED BY LOADR OR VICE-VERSA. ANOTHER CAUSE IS TRYING TO REPLACE OR PURGE A MEMORY-RESIDENT PROGRAM USING MLLDR.

""

L-RP PGM

YOU TRIED TO REPLACE OR PURGE A PERMANENT PROGRAM THAT HAS TERMINATED SERIALLY REUSABLE, SAVING RESOURCES, OR WAS OPERATOR SUSPENDED. THAT IS, THE PROGRAM STILL OWNS A PARTITION. REMOVE THE PROGRAM USING THE OF COMMAND AND REPEAT THE PURGE OR REPLACE OPERATION.

""

L-RQ PGS

YOU DID BOTH A SZ AND AS AND THE SIZE IS LARGER THAN THE PARTITION. THE NUMBER OF PAGES THAT YOU SPECIFIED IN THE LOAD OF THE PROGRAM EXCEEDS THAT NUMBER OF PAGES IN THE PARTITION YOU SPECIFIED. EITHER SPECIFY A DIFFERENT PARTITION OR NO PARTITION AT ALL. ANOTHER POSSIBILITY IS THAT THE PROGRAM COULD NEVER FIT INTO A PARTITION, EVEN AFTER MEMORY RECONFIGURATION WITH THE LARGEST POSSIBLE AMOUNT OF PHYSICAL MEMORY. THIS ERROR ALSO OCCURS IF A NEGATIVE SIZE IS SPECIFIED IN THE RUNSTRING.

""

L-SH EMA

A SHAREABLE EMA LABEL FILE DID NOT HAVE THE CONTROL COMMAND \$SHEMA STARTING IN THE FIRST COLUMN IN THE FIRST LINE, OR THERE WAS ANOTHER ERROR IN THE FILE, SUCH AS A LABEL LONGER THAN 16 CHARACTERS. NOTE: IF THE LABEL SPECIFIED IN THE SH COMMAND, IS NOT FOUND IN THE SYSTEM, THEN THE LOADER RESPONDS WITH ?? IF THE COMMAND WAS ENTERED INTERACTIVELY, OR WITH L-IL PRM IF THE COMMAND WAS IN A COMMAND FILE.

Update 1

Errors

" "

L-SH PTN

A PROGRAM CANNOT BE ASSIGNED TO A SHAREABLE EMA PARTITION, A SUBPARTITION OF A SHAREABLE EMA PARTITION, OR A MOTHER PARTITION WHICH HAS ANY SHAREABLE EMA SUBPARTITIONS.

" "

L-SS ENT

YOU ATTEMPTED TO ACCESS AN SSGA ENTRY POINT BUT YOU DID NOT ASK FOR SSGA AT THE BEGINNING OF THE LOAD. RELOAD THE PROGRAM BUT THIS TIME DO AN 'OP,SS' AT THE BEGINNING OF THE LOAD. THE TWO NAMES REPORTED ARE THE MODULE IN WHICH THE REFERENCE WAS MADE AND THE ENTRY POINT BEING REFERENCED.

" "

L-SZ ALC

ALLOCATE SIZE ERROR. POSSIBLE CAUSES ARE:

1. THE FIRST ALLOCATE RECORD WITH THIS NAME HAS TO SPECIFY THE MAXIMUM SIZE. AN ALLOCATE RECORD IN THIS MODULE HAS A SIZE LARGER THAN THE FIRST OCCURRENCE OF THE RECORD.
2. A COMMON BLOCK WAS DECLARED NON-SAVE, AND WAS DECLARED SAVE IN A MODULE LOADED LATER.
3. THE FIRST OCCURRENCE OF THIS NAME WAS EMA TYPE, AND THE NAME OF OCCURS IN THIS MODULE WITH A NON-EMA TYPE, OR VISE-VERSA.

" "

L-SZ EMA

EMA SIZE IS GREATER THAN 1K PAGES AND VMA IS NOT BEING USED. THIS ERROR CAN ALSO OCCUR IF THE SECOND OF TWO MODULES THAT HAVE THE SAME NAME HAS A LARGER LOCAL EMA SIZE THAN THE FIRST MODULE.

" "

L-TR ADD

NO TRANSFER ADDRESS. ONLY SUBROUTINES WERE LOADED. THE LOADER COUNT NOT TELL WHICH MODULE OF THE PROGRAM WAS THE MAIN AND WHICH ONES WERE SUBROUTINES. IF THE PROGRAM WAS WRITTEN IN FORTRAN NO MODULES WERE FOUND THAT CONTAINED THE 'PROGRAM XXXXX' STATEMENT. IF THE PROGRAM WAS WRITTEN IN ASMB YOU PROBABLY FORGOT TO PUT A LABEL ON THE END STATEMENT. IN ASMB THE MAIN OF A SEGMENT OR OF A PROGRAM IS DIFFERENTIATED FROM SUBROUTINES BY PLACING THE LABEL OF WHERE THE PROGRAM OR SEGMENT IS TO START EXECUTION AS THE OPERAND OF THE END STATEMENT. IF MULTIPLE ROUTINES HAVE LABELS ON THE END STATEMENT, THE FIRST ONE ENCOUNTERED IS USED AS THE MAIN OF THE PROGRAM. IF A MULTI LEVEL PROGRAM WAS LOADED, THE TRANSFER ADDRESS MUST BE IN THE ROOT. THAT IS, THE MAIN MUST BE IN THE ROOT.

Errors

" "

L-UN EXT

UNDEFINED EXTERNALS EXIST WHICH PROHIBITS THE LOAD FROM COMPLETING. AN UNDEFINED EXTERNAL IS A REFERENCE MADE BY THE ROUTINE YOU ARE LOADING TO ANOTHER ROUTINE. FOR EXAMPLE, IF YOUR FORTRAN PROGRAM HAD THE FOLLOWING CODE:

```
CALL XYZ(I,J,K)
```

THEN THE SUBROUTINE XYZ WOULD BE AN EXTERNAL. THE PROBLEM YOU HAVE IS THAT YOU LOADED THE ROUTINE THAT CONTAINED THE CALL TO XYZ BUT YOU DIDN'T LOAD THE XYZ SUBROUTINE ITSELF. XYZ IS THE UNDEFINED EXTERNAL. THE PROPER COURSE HERE IS TO RELOAD YOUR PROGRAM BUT THIS TIME DON'T FORGET TO LOAD THE ROUTINES LISTED WHEN THE LOADER ABORTED THE LAST TIME YOU TRIED TO LOAD THE PROGRAM. ONE LAST POINT. IT IS POSSIBLE TO FORCE LOAD A PROGRAM OR SEGMENTS THAT HAVE UNDEFINED EXTERNALS. THIS IS DONE WITH THE LOADER 'FORCE' COMMAND. HOWEVER, IF YOU FORCE LOAD THE PROGRAM IT IS YOUR RESPONSIBILITY TO MAKE SURE THAT THE LINE OF CODE THAT REFERENCES THE EXTERNAL IS NEVER EXECUTED. THAT IS, MAKE SURE THAT THE CALL TO XYZ IS NOT EXECUTED OR YOUR PROGRAM WILL PROBABLY BE ABORTED WITH A DM OR MP ERROR.

" "

L-VM EMA

TRIED TO USE SHAREABLE VMA. SHAREABLE VIRTUAL MEMORY IS NOT SUPPORTED.

" "

L-VS EMA

THE SPECIFIED VMA SIZE IS ILLEGAL.

" "

W-DU PGM

DUPLICATE PROGRAM NAME. NO ACTION IS REQUIRED. PROGRAM XXXXX IS RENAMED ..XXX AUTOMATICALLY.

" "

W-IL CMD

ATTEMPTED TO RELOCATE A MODULE OR TRANSFER TO A COMMAND FILE WHILE DOING SPECIAL PROCESSING WHEN UNDEFINED EXTERNALS EXIST. AT THIS TIME, RE,XXXXX OR TR,YYYYY, OR ANY M OR D COMMAND ARE ILLEGAL COMMANDS. IN THE CASE OF AN MLS PROGRAM, /E, EN, AND EX ARE ALSO ILLEGAL AT THIS TIME. LEGAL COMMANDS ARE SE, SL, LI, FO, DI, AB, AND /A.

" "

W-IN CAP

DUE TO INSUFFICIENT USER CAPABILITY, THE PROGRAM PRIORITY WAS CHANGED TO 99. THE MESSAGE IS:

```
/MLLDR: XXXXX SET TO PRIORITY 99
```

```
/MLLDR: W-IN CAP
```



" "

W-RQ PGS

THE REQUIRED NUMBER OF PAGES IS TOO LARGE. ONE OF THE FOLLOWING HAS OCCURRED:

1. THE PROGRAM WAS ASSIGNED TO A PARTITION AND THE PARTITION IS NOT LARGE ENOUGH. REASSIGN THE PROGRAM TO A PARTITION THAT IS LARGE ENOUGH.
2. THE SIZE SPECIFIED WAS LARGER THAN ANY EXISTING PARTITION. CHANGE THE SIZE OR RECONFIGURE MEMORY, CREATING A PARTITION THAT IS LARGE ENOUGH.
3. NO SIZE WAS SPECIFIED, BUT THE PROGRAM IS TOO LARGE FOR ANY EXISTING PARTITION. RECONFIGURE MEMORY, CREATING A PARTITION THAT IS LARGE ENOUGH.
4. SHAREABLE EMA WAS USED, AND THE SHAREABLE EMA PARTITION IS NOT LARGE ENOUGH FOR THE PROGRAM'S EMA.

YOU CAN LOOK AT THE LOADMAP TO CHECK IF IT WAS THE PROGRAM CODE SIZE, EMA SIZE, OR WORKING SET SIZE WHICH CAUSED THE PROBLEM AND TRY A SYSTEM SZ COMMAND TO RESIZE THE PROGRAM.

" "

W-SV MIX

MIXING SAVE NAMED COMMON WITH NAMED COMMON. A NAMED COMMON BLOCK MATCHES THE NAME OF A PREVIOUSLY LOADED SAVE NAMED COMMON BLOCK.

" "

W-UN EXT

UNDEFINED EXTERNALS EXIST AND THE LOADER WAS INITIATED FROM AN INTERACTIVE DEVICE. WHEN THE PROMPT IS ISSUED, DO AN SE OF THE LIBRARY CONTAINING THE REQUIRED SUBROUTINE. IF THE ENTRY POINT WAS SPECIFIED IN AN "NA" COMMAND, USE "LI,<FILE>" ON THE FILE CONTAINING THE MODULE, FOLLOWED BY AN "SL" COMMAND.

" "

W-WS EMA

THE SPECIFIED WORKING SET SIZE IS TOO LARGE. IF USING INTERACTIVE MODE ENTER A SMALLER WORKING SET SIZE. OTHERWISE, YOU CAN USE THE SYSTEM WS COMMAND TO CHANGE THE SIZE OF THE WORKING SET.

SGMTR Error Messages

- SGMTR: BAD RUNSTRING
SGMTR did not get a runstring of the sort it expected. Refer to the chapter on the runstring for a description.
- SGMTR: NO INPUT FILE GIVEN
SGMTR needs the input filename in the runstring. Refer to the chapter on the runstring for a description.
- SGMTR: PATH LIMIT NOT INTEGER
The path limit parameter did not parse as an integer.
- SGMTR: PATH LIMIT OUT OF RANGE (1-31)
The path limit was out of the range of one to 31 pages. EMA programs should not be segmented into paths of more than 29 pages to allow two pages for the EMA window at the end of the partition. Non EMA programs have up to 31 pages available for a path.
- SGMTR: UNRECOGNIZED NODE TYPE
The letter "M" or "D" should be used to select memory-resident or disc-resident nodes for generation. This error occurs if the letter is not an "M" or "D", or if both "M" and "D" are specified. Memory-resident nodes can be switched in microseconds where disc-resident nodes require milliseconds, but disc nodes require less program space since memory-resident nodes begin on page boundaries. Also, disc-resident nodes may re-initialize local data when they are switched back. Memory-resident nodes retain local data changes.
- SGMTR: BAD TYPE SPECIFIED FOR FILE: <NAMR>
File types of 3, 4, or 8 or more are accepted.
- SGMTR: BAD SIZE SPECIFIED FOR FILE: <NAMR>
File sizes of -1, or 1 or more are accepted.
- SGMTR: CANNOT CREATE FILE: <NAMR>
The file cannot be created. The cartridge does not have sufficient space to allocate for the file.
- SGMTR: ILLEGAL LU: <NAMR>
The output file is specified as an illegal session LU. Check your session list with the SL command for legal LU numbers.

Errors

SGMTR: MAIN ROUTINE MISSING

The given main entry point cannot be found in any of the input modules, or the default of using the first module with an entry point failed because no input file modules had entry points.

SGMTR: WARNING! MAIN ROUTINE HAS NO TRANSFER ADDRESS

The main routine has no transfer address defined in the end record. MLLDR will not be able to determine where the program should begin execution. If a load is attempted with such a main routine then it will abort.

SGMTR: SYMBOL TABLE OVERFLOW

The symbol table overflowed. It has a capacity for about 5000 six character symbols. Try segmenting a subtree or portion of the program a piece at a time and join the results later.

SGMTR: EMA SPACE OVERFLOW

The EMA space used by SGMTR does not have capacity to hold all the necessary information on the program's modules. Try segmenting a subtree or portion of the program a piece at a time and join the results later.

SGMTR: STACK OVERFLOW

Too many modules exist for the amount of dynamic memory available. More dynamic memory space may be needed. If possible, reload SGMTR specifying a larger dynamic buffer area.

SGMTR: MODULE STACK OVERFLOW

The dynamic memory space is insufficient to hold the information necessary for the offpath reference check. If possible, reload SGMTR specifying a larger dynamic buffer area.

SGMTR: NODE STACK OVERFLOW

More than 100 nodes exist in a path.

SGMTR: NODE TREE OVERFLOW

The EMA space is insufficient to hold the nodes under construction. Try segmenting a subtree or portion of the program a piece at a time and join the results later.

SGMTR: BLANK COMMON SIZE ERROR

The las routine loaded contained a blank or unnamed COMMON area of size greater than the main routine of the program. This routine will cause a load time error if a load is attempted.

Errors

- SGMTR: WARNING! LAST ROUTINE IS A SEGMENT
The las routine loaded is a type 5 module or a segment program. If a load is attempted with the output of SGMTR then it will abort. If the routine is written in FORTRAN, convert the program statement to a subroutine. If the routine is written in ASMB or MACRO, convert the type given in the NAM statement to 7 (for other values consult the language manual). Edit the other program routines and convert any EXEC 8 or SEGLD calls into subroutine calls.
- SGMTR: UNSATISFIED EXTERNALS IN MODULE: <NAM AND COMMENT>
<EXTERNAL> <EXTERNAL> <EXTERNAL> <EXTERNAL>
No entry points can be found in the input file or the system library which match the externals of the module.
- SGMTR: DUPLICATE ENTRY POINT SYMBOLS : <SYMBOL>
This error is preceded by the message:

SGMTR: FIRST FOUND IN MODULE (module #)

The last routine loaded contains an entry point which matches an entry point of the routine loaded earlier. SGMTR cannot tell which entry point is to be used to satisfy external references.
- SGMTR: ENTRY POINT MATCHES RPL SYMBOL: <SYMBOL>
This is like a duplicate entry point error In this case an entry point matches a RPL symbol.
- SGMTR: DUPLICATE RPL SYMBOLS : <SYMBOL>
This is like a duplicate entry point error. In this case two rpl symbols have been seen of different values.
- SGMTR: RPL SYMBOL MATCHES COMMON : <SYMBOL>
RPL symbols cannot be used to replace or initialize labeled COMMON.
- SGMTR: ENTRY POINT MATCHES COMMON : <SYMBOL>
An entry point matches a labeled SAVE or labeled EMA COMMON block. Entry points may only match ordinary named COMMON blocks.
- SGMTR: COMMON BLOCK TYPES DONT MATCH : <SYMBOL>
A labeled COMMON block has been declared in two different ways out of the choices of SAVE, EMA or ordinary named COMMON.

Errors

- SGMTR: COMMON BLOCK SIZES DONT MATCH : <SYMBOL> <SIZE>
The last routine to be loaded declared a labeled COMMON block of a size which does not match the size declared in an earlier loaded routine. If the smaller of the two appears first in the output command file a load time error will result.
- SGMTR: SAVE COMMON BLOCK NOT IN MAIN : <SYMBOL> <SIZE>
The last routine to be loaded declared a labeled SAVE COMMON block which was not declared in the main routine. SGMTR requires all SAVE and EMA COMMON blocks to be declared in the main routine of the program.
- SGMTR: EMA COMMON BLOCK NOT IN MAIN : <SYMBOL> <SIZE>
The last routine to be loaded declared a labeled EMA COMMON block which was not declared in the main routine. SGMTR requires all SAVE and EMA COMMON blocks to be declared in the main routine of the program.
- SGMTR: BLOCK DATA MISSING FOR COMMON : <SYMBOL> <SIZE>
A block data subprogram or MACRO routine with an entry point matching the label of the labeled COMMON block cannot be found. SGMTR requires an entry point to be available in the input file for every ordinary named COMMON block in the program. Block data subprograms written in FTN7X should be compiled with the noallocate option for those labeled COMMON blocks.
- SGMTR: WARNING! PATH LIMIT OVERFLOW!
This message is emitted during segmentation. SGMTR cannot segment the program to fit in the given path limit. Try a larger path limit. If your attempt was made with memory-resident nodes, try again with disc-resident nodes as they are smaller. It may be that the program will load anyway because SGMTR may estimate thunk overhead to be more than it is. This may be the case for memory-resident nodes where the path that overflows the limit contains a node which crosses a page boundary due to the thunk overhead. It may also be the case for disc-resident nodes where the node size for a node in the offending path is overestimated due to thunks.
- SGMTR: ...PATH LIMIT OVERFLOW!
This message is emitted during the listing of the output. It has the same meaning as the previous message. It labels the node responsible for the warning.

Errors

SGMTR: OFFPATH REFERENCE IN WHICH THE MODULE:
<NAM AND COMMENT>

IN NODE ORDINAL <ORDINAL>

IN A PREDECESSOR NODE REFERENCES THE MODULE:
<NAM AND COMMENT>

WHICH COULD POTENTIALLY CAUSE AN OFFPATH REFERENCE

The first module listed references the second module listed. The second module lies in a higher node than the first module occupies. The second module also references modules in a path other than the first module occupies. Therefore it is possible for an offpath reference error to occur during execution. Note well, it is assumed the main module of the program is never called by any other module in the program. No offpath reference check is applied to the main module.

SGMTR: FMGRXXX ERROR WITH FILE: <NAMR>

A file manager error took place with the given namr. If SYSLIB appears then an error took place with the system library.

SGMTR: SEGMENT SGMT<X> CANNOT BE LOADED

One of SGMTR's segments could not be found to be loaded. SGMTR's segments are named SGMT1, SGMT2, SGMT3, SGMT4, SGMT5, and SGMT6.

SGMTR: TERMINATING...

SGMTR terminated with errors.

SGMTR: COMPLETED!

SGMTR finished successfully.

SXREF Error Messages

SXREF Errors abort processing except for messages noted as warning messages.

SXREF: BAD RUNSTRING

No command string could be found using an EXEC 14 request.

SXREF: NO COMMAND FILENAME GIVEN

No parameter for the command file could be found in the run string or the parameter was not recognized as a filename.

SXREF: BAD TYPE SPECIFIED FOR FILE: <FILE>

The parameter for the output cross-reference listing file was not type 3, 4, 8, or higher.

SXREF: BAD SIZE SPECIFIED FOR FILE: <FILE>

The parameter for the output cross-reference listing file did not have a size of -1, 1, or more.

SXREF: LINE XXXXXX WRONG PARAMETER TYPE

The command expects a filename if an lu was given or the command expects an lu or number if a filename or symbol was given.

SXREF: LINE XXXXXX COMMAND FILE READ ERROR

Error occurred while reading the command file.

SXREF: LINE XXXXXX COULD NOT BE PARSED

The command at the given line number is not recognizable as a valid loader command.

SXREF: LINE XXXXXX COMMAND OUT OF ORDER

The command occurring at the given line number is out of order. Order of commands expected by SXREF is documented with SXREF.

Errors

SXREF: LINE XXXXXX REFERENCED FILE COULD NOT BE OPENED
The command referenced a file which may not exist or which may be locked by a security code or open to another program.

SXREF: NODE DEPTH OVER 31 DEEP
WARNING message only. This error will not cause an abort. The current node depth is over 31. This is impossible to load if the path is made of memory-resident nodes and may not be readable due to line length limits of 80 characters.

SXREF: NODE COUNT OVER 1024
WARNING message only. This error will not cause an abort. This is a one time message and will not reoccur. This many nodes may create space problems on disc if a load is attempted.

SXREF: LINE XXXXXX EITHER A DISC RESIDENT NODE
PRECEDES A MEMORY RESIDENT NODE
OR THE FIRST DISC RESIDENT NODE
IS NOT A SON OF THE ROOT NODE
The first occurrence of a disc-resident node must be as a son of the root node. Thereafter only disc-resident nodes may be used.

SXREF: LINE XXXXXX NODE DEFINITION OUT OF ORDER
The node definition at the given line number is out of order. The definition is more than one level deeper than the preceding node definition.

SXREF: LINE XXXXXX ROOT NODE DEFINITION MISSING
The root node definition command "M" should be used before the command given in the line where the error was found.

SXREF: LINE XXXXXX BAD FIRST SON NODE DEFINITION
The definition is one level deeper than the preceding node definition but the level number is not a "1" or the other level numbers do not match up.

SXREF: LINE XXXXXX BAD BROTHER NODE DEFINITION
The definition is the same level as the preceding node definition but does not match on corresponding level numbers or is not one greater in the last level number.

Errors

SXREF: LINE XXXXXX BAD PREDECESSOR NODE DEFINITION
The definition given is for a node at a higher level than the preceding node definition but does not match up in corresponding level numbers.

SXREF: MORE THAN 128 UNIQUE FILES ARE REFERENCED
There is an internal limit on the number of files which may be referenced and that limit is 128.

SXREF: MORE THAN 10 GLOBAL FILES ARE DEFINED
The loader will accept no more than 10 global user library files.

SXREF: DUPLICATE ENTRY POINT: XXXXXXXXXXXXXXXX IN NODE XXXXXX
AND XXXXXX
WARNING message only. SXREF will not abort.
The given entry point is duplicated in the nodes reported. Nodes are reported by their preorder number.

SXREF: UNDEFINED ENTRY POINT: XXXXXXXXXXXXXXXX IN NODE XXXXXX
WARNING message only. SXREF will not abort.
A module containing the given entry point cannot be found and is needed in the ordinal node.

SXREF: UNDEFINED ENTRY POINT: XXXXXXXXXXXXXXXX
REFERENCED BY NA COMMAND IN NODE: XXXXXX
WARNING message only. SXREF will not abort.
A module containing the given entry point cannot be found in the global libraries and is needed in the ordinal node.

SXREF: UNDEFINED ENTRY POINT: XXXXXXXXXXXXXXXX
REFERENCED BY SY COMMAND IN NODE: XXXXXX
WARNING message only. SXREF will not abort.
A module containing the given entry point cannot be found in the system library and is needed in the ordinal node.

Errors

SXREF: POSSIBLE OFFPATH REFERENCE VIA EXTERNAL XXXXXXXXXXXXXXXXXXXX
OF A MODULE IN NODE XXXXXX
TO AN ENTRY POINT IN NODE XXXXXX
ORIGINAL CALLER IS <NAM>
IN NODE: XXXXXX

WARNING message only. SXREF will not abort.
SXREF has detected a potential offpath reference error. If the original caller module indirectly references a module occurring in a predecessor node of the original module's node which references a module lying in a node off the path defined by the original module's node, then an offpath reference could occur during a run of the program.

Reference chains are not traced through the main module of the program because it is assumed that the main module is never called by any of its subroutines. The default main module is the first module loaded into the root node with a non-zero transfer address.

SXREF: SEGMENT XXXXXX COULD NOT BE LOADED
The given segment could not be found for loading.

SXREF: FMGR XXX ERROR WITH FILE: <FILE>
A file manager error occurred with the given file.

SXREF: MORE DYNAMIC MEMORY SPACE NEEDED
SXREF makes use of the space occurring at the end of the partition. This message implies the program needs more space for buffers and symbol table. Use the SZ command on SXREF to size it up or assign it to a larger partition.

SXREF: TERMINATING...
Either the user requested or an error caused an early abort of the run.

SXREF: COMPLETED!
SXREF completed its operation successfully.

INDXR Error Messages

??

An invalid command was entered, try again.

FMGR xxx

A FMP error has occurred. When a FMGR type error occurs giving an error message of this form then the FMGR error is placed in the SCB for the HELP program to examine. Some common errors and their responses are:

-11= DCB not open. Probably an attempt was made to use the INDEX command before a library file had been created with the CREATE command. Use the CREATE command to create a file for the indexed relocatable library.

-6 = Nonexistent file. The file specified in the INDEX command does not exist. Check to see that the proper name was specified.

-2 = Duplicate file name. When the CREATE command was used to create a library file, a file with the same name was found. Use a different name for the indexed relocatable library file.

*** CAUTION ! INDEXED FILE ALREADY EXISTS, OVERLAY(Y OR N) ?

INDXR has found in response to a "CR" command that the indexed file specified already exists. This message will be output and a response read from the LU that scheduled the program. Enter a 'Y' if the file specified is to be overlaid with new data, enter a 'N' otherwise.

*** INPUT ERROR, INDXR ABORTED ***

An invalid transfer file was specified in the run string. A transfer file cannot be a type 1 or 2 file. Check the transfer file.

*** ONLY ONE LEVEL OF TRANSFER ALLOWED ***

Only one transfer file may be opened for command input to this utility. This includes run-string transfer files and LUS. Restructure the transfer file/LU so that only one level is utilized.

Errors

*** INDEXED FILE ALREADY EXISTS, REQUEST IGNORED ***

An attempt was made to use the CREATE command more than once. The command is ignored and the utility continues to use the file specified with the first CREATE command.

*** FILE MUST BE TYPE 5, REQUEST IGNORED ***

The file specified in the INDEX command was not a type 5 (relocatable) file. Only a relocatable module, or library can be merged into an indexed relocatable library. Check to make sure that the correct filename was entered.

*** CHECKSUM ERROR, INDXR ABORTED ***

While reading records from the file specified by the last INDEX command, a checksum error was detected in one of the records. The file may be corrupt.

*** DIRECTORY TOO LARGE, INDXR ABORTED ***

The INDXR creates a scratch file to build the directory index in. This file is created on the first cartridge in the cartridge list. If this file creates more than 255 extents or fills up the remainder of the cartridge, then this error is returned. Pack the first cartridge in the list or allocate an empty cartridge as the first cartridge in the list to be used by the scratch file.

*** SCRATCH FILE CREATE OVERFLOW ***

The INDXR has tried to create a scratch file with a name in the range from DIRA to DIRZ, but has found that all these files already exist. The INDXR must have a scratch file to build a directory for the indexed file. To correct this error erase at least one file in the above range and rerun the INDXR.

*** LIST FILE MUST BE TYPE 3 OR 4 ***

An already existing FMP file used as a list file must be a type 3 or 4. If the file is opened and found to be another type then this error is issued. Note that a list file created by INDXR is always created as a type 4.

*** LIST FILE/LU ALREADY OPEN, REQUEST IGNORED ***

The list file may only be opened/created once. If the LI[ST] command is invoked more than once then this error is issued.

Appendix B

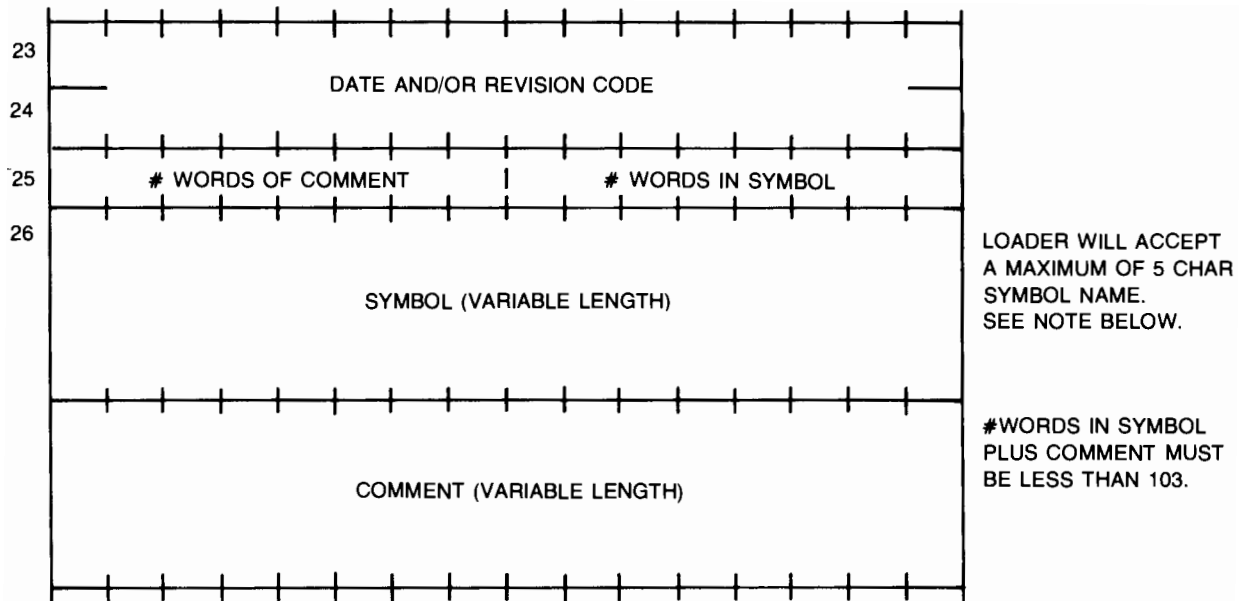
Relocatable Record Formats

NOTE

Hewlett-Packard reserves the right to modify the form of these data structures without notice to the customer.

EXTENDED NAM RECORD

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	EXPLANATION
1	RECORD LENGTH												////////////////////				REC LENGTH <128 WORDS
2	IDENT =7				SUB-IDENT =1				OFFSET OF SIZ WRD								
3	CHECKSUM																CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS EXCEPT WORDS 1 AND 3
4	LOCAL EMA SIZE																SIZE IS IN WORDS
5	SAVE SIZE																SIZE IS IN WORDS
6	PROGRAM SIZE																SIZE IS IN WORDS
7	0	PROGRAM SIZE															SIZE IS IN WORDS
8	BASE PAGE SIZE																SIZE IS IN WORDS
9	COMMON SIZE																SIZE IS IN WORDS
10	PROGRAM TYPE																
11	PRIORITY																
12	RESOLUTION CODE																
13	EXECUTION MULTIPLE																
14	HOURS																
15	MINUTES																
16	SECONDS																
17	10'S OF MILLISECONDS																
18	<RESERVED>																
19	YEAR OF COMPILATION																
20	JULIAN DAY								HOUR								
21	MINUTES								SECONDS								
22	<RESERVED>																



///// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

NOTE

All Operating systems allow space for only five characters in main-program and segment names.

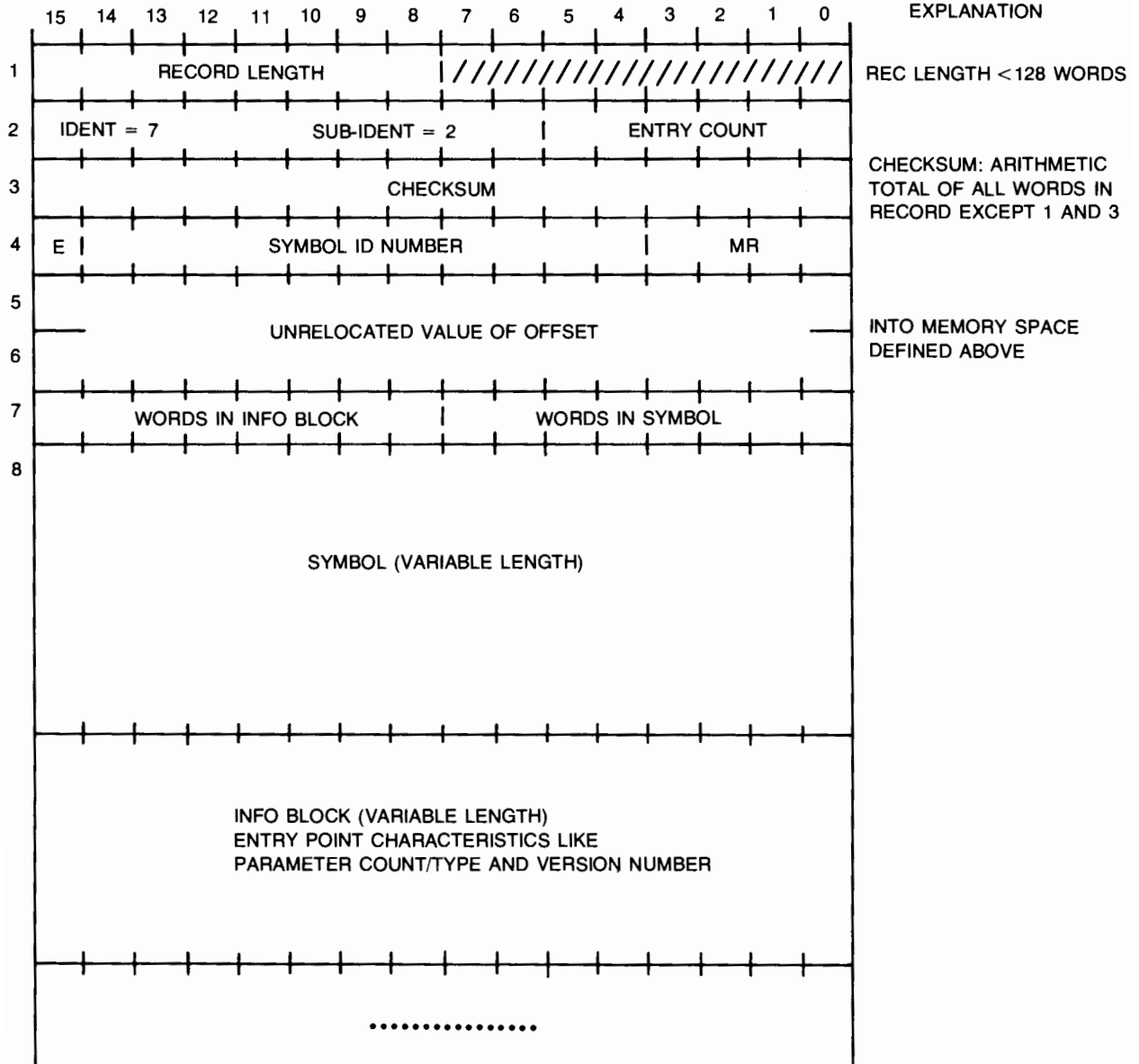
The loaders for RTE-6/VM and RTE-XL will handle symbols longer than five characters.

The loaders for RTE-4B, RTE-4E and earlier operating systems will not handle the longer symbols.

All system generators are limited to five-character symbols, module names and entry points.

Use of the OLDRE Utility is recommended to ensure conformance to these standards.

EXTENDED ENT RECORD

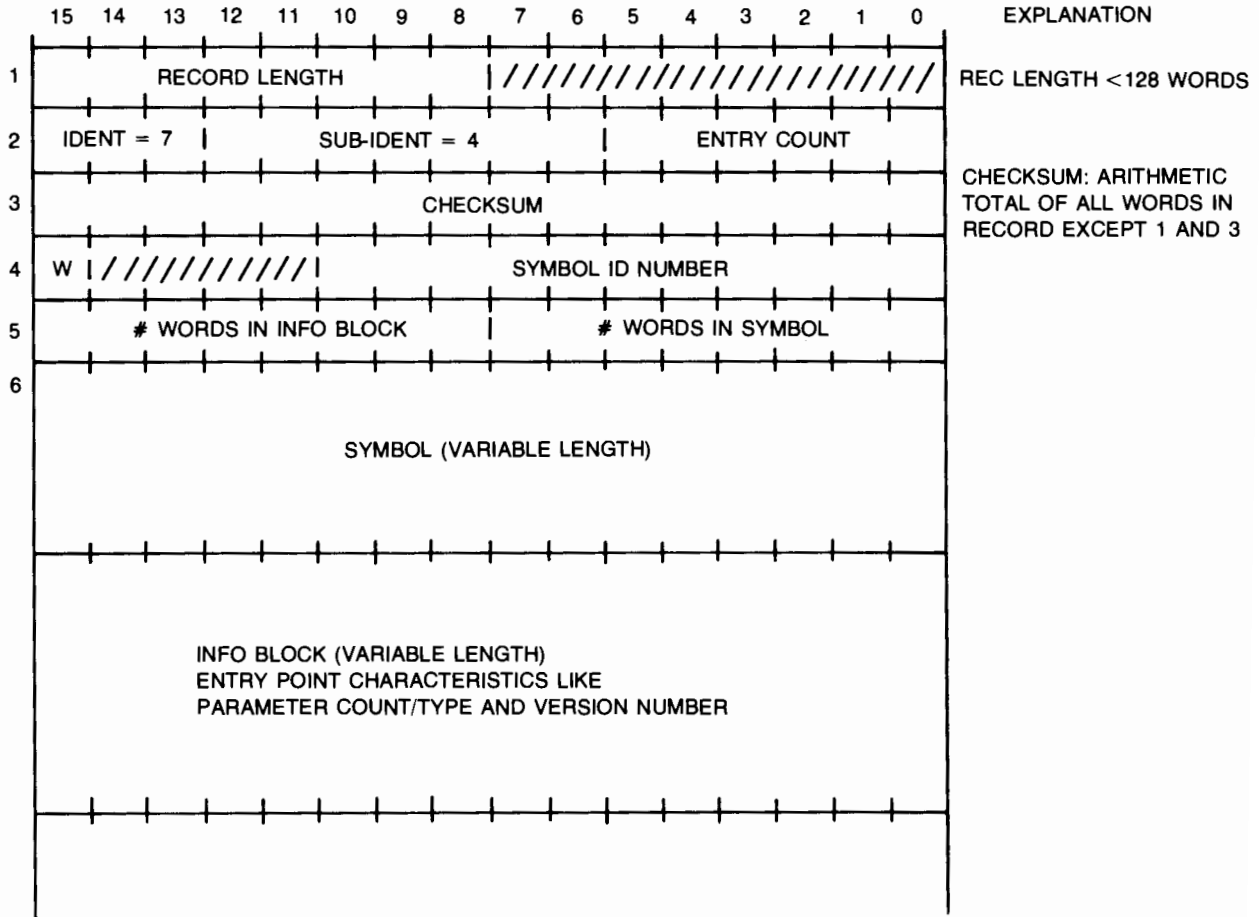


///// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

E = 1, IF EMA
= 0, IF MR

MR IS MEMORY SPACE
= 0, IF ABSOLUTE SPACE
= 1, IF PROGRAM RELOCATABLE SPACE
= 2, IF BASE PAGE RELOCATABLE SPACE
= 3, IF COMMON RELOCATABLE SPACE
= 4, <RESERVED>
= 5, IF EMA SPACE
= 6, IF SAVE AREA SPACE

EXTENDED EXT RECORD

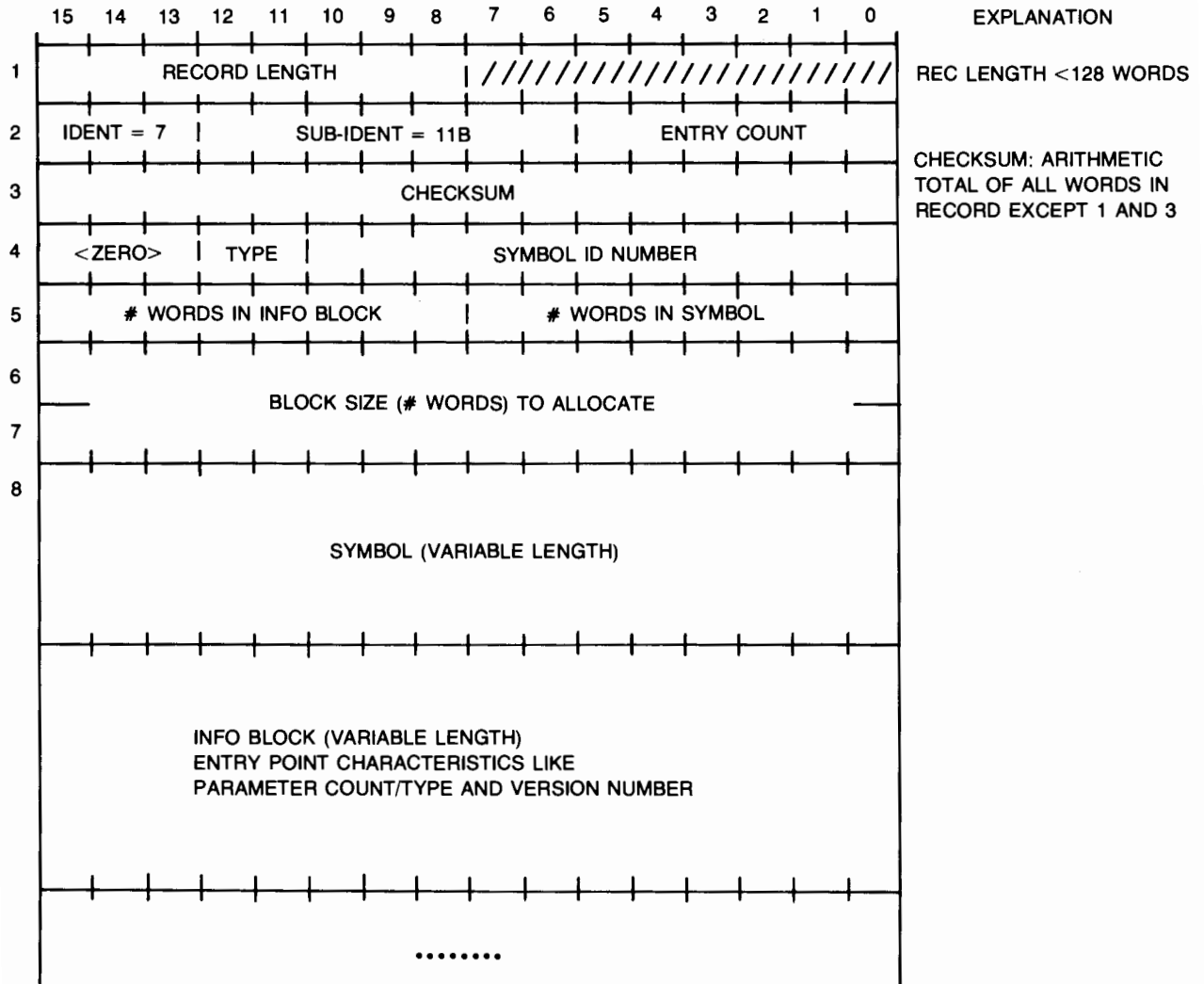


///// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

W = 1, IF WEAK EXTERNAL
= 0, IF REGULAR EXTERNAL



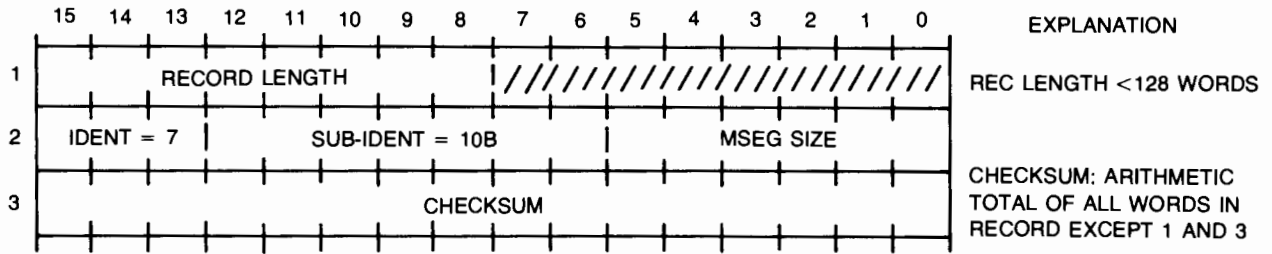
ALLOCATE RECORD



///// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

TYPE = 0, IF NAMED COMMON (PROGRAM ALLOCATE)
 1, IF NAMED SAVE COMMON (SAVE ALLOCATE)
 2, IF NAMED EMA COMMON (EMA ALLOCATE)

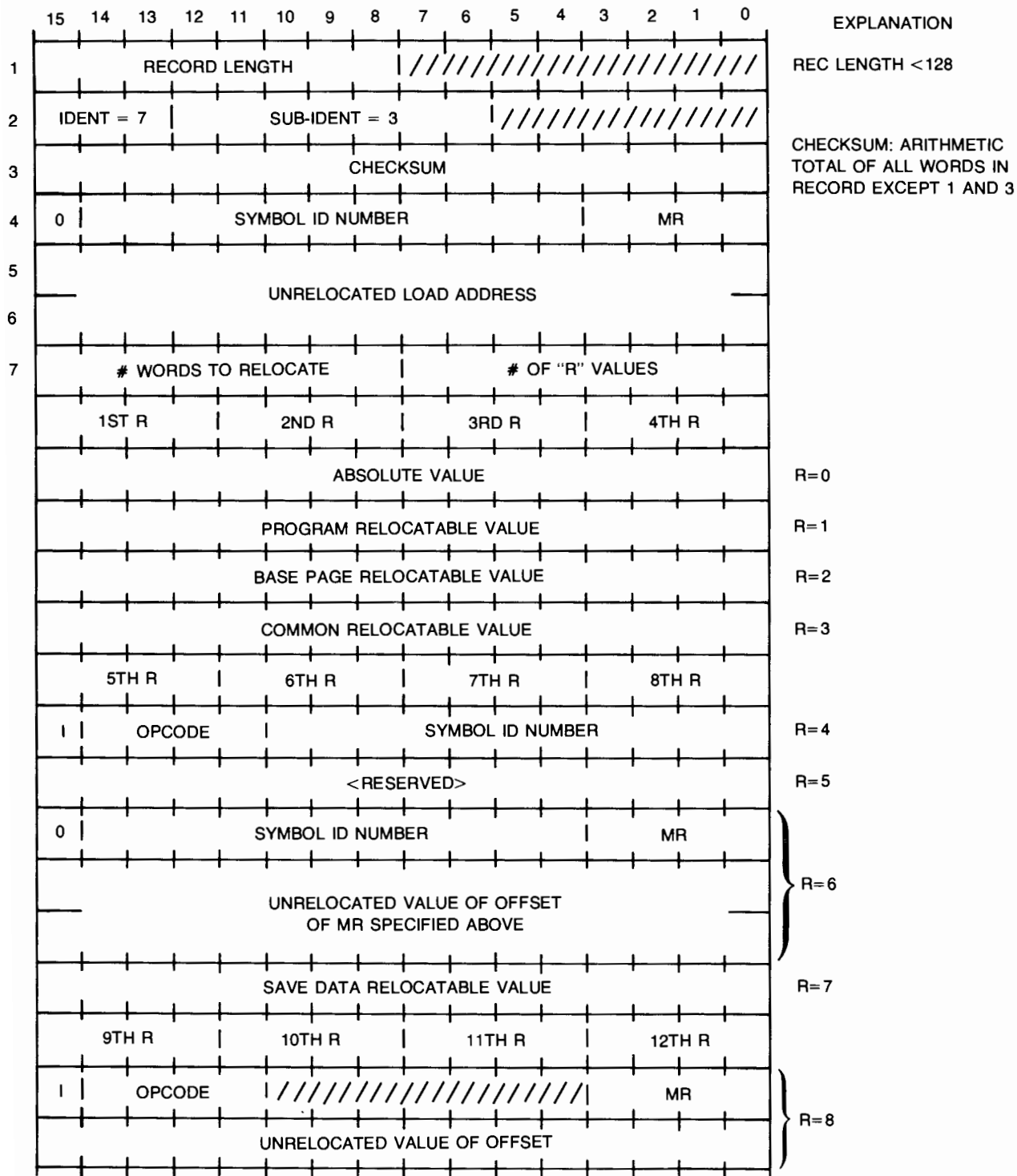
MSEG RECORD



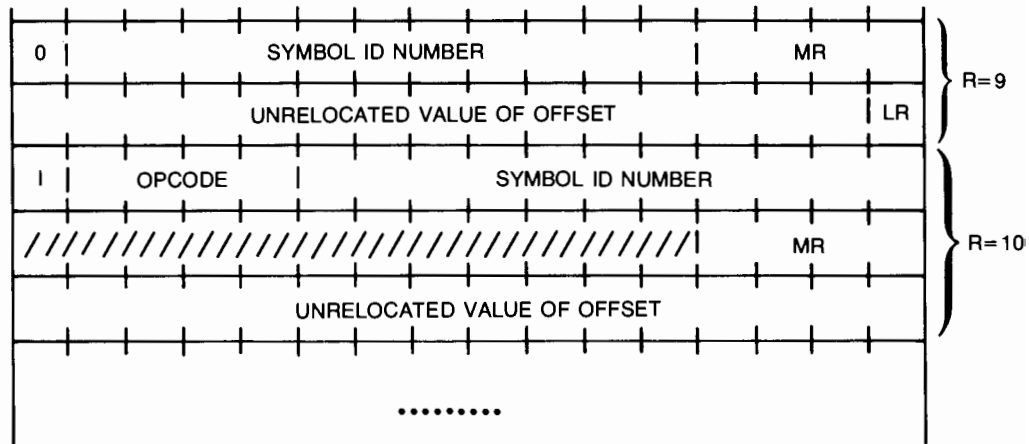
////// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

MSEG SIZE IN PAGES, 1 ≤ MSEG SIZE ≤ 32

EXTENDED DBL RECORD



EXTENDED DBL RECORD (continued)

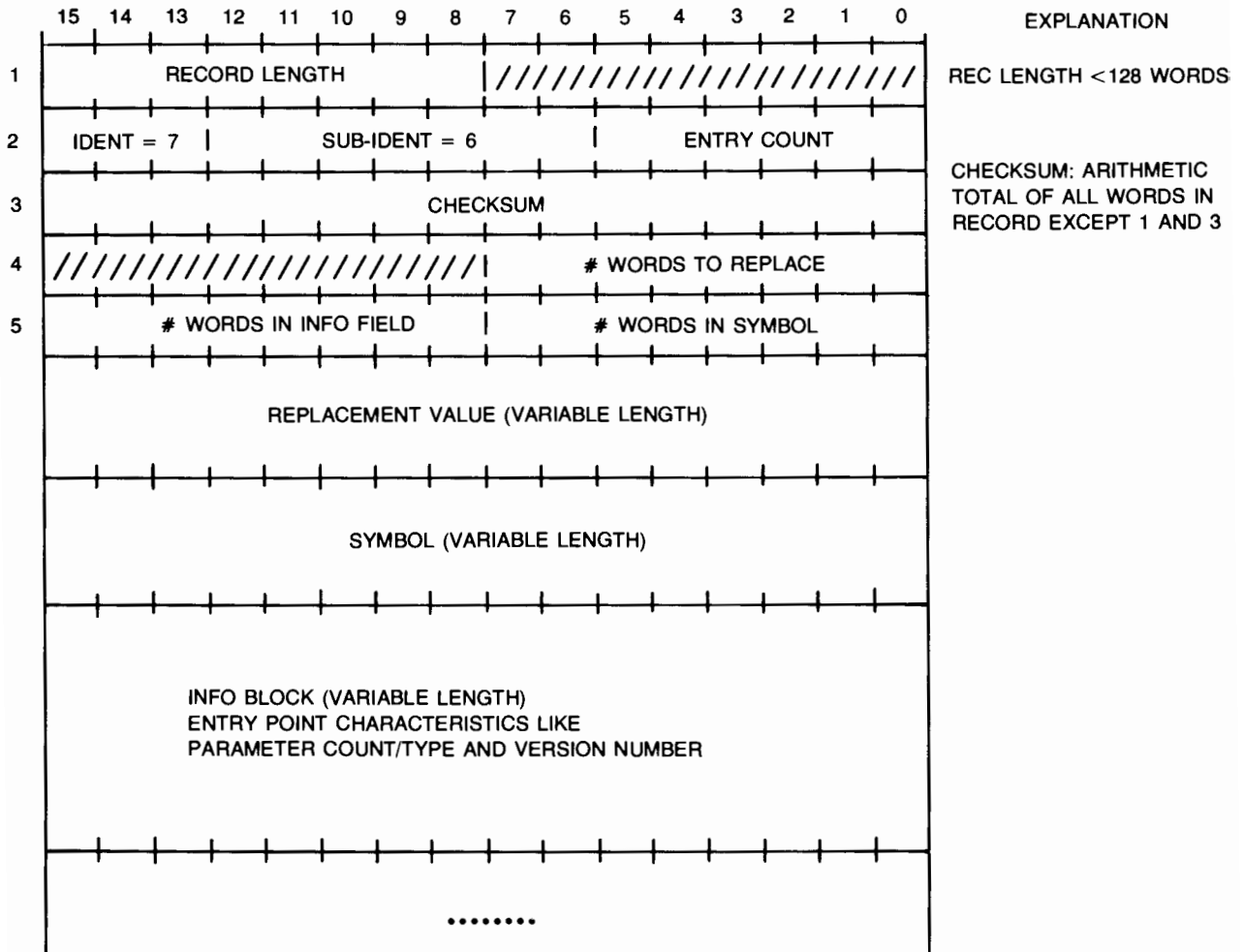


////// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

MR IS MEMORY SPACE

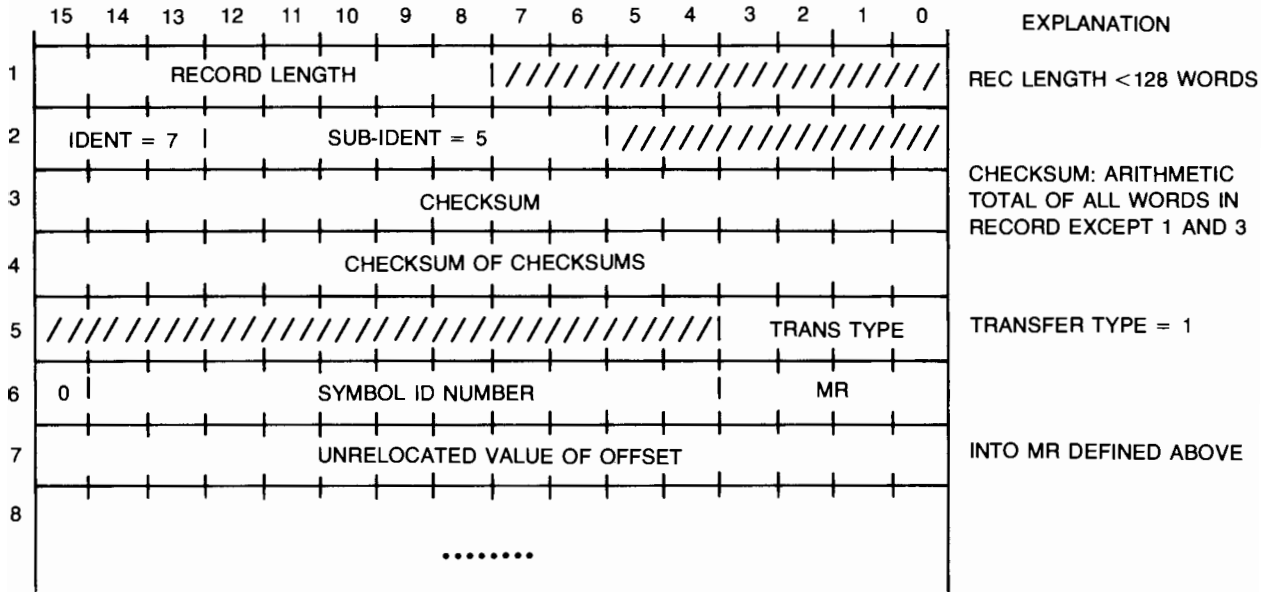
- = 0, IF ABSOLUTE SPACE
- = 1, IF PROGRAM RELOCATABLE SPACE
- = 2, IF BASE PAGE RELOCATABLE SPACE
- = 3, IF COMMON RELOCATABLE SPACE
- = 4, <RESERVED>
- = 5, IF EMA SPACE
- = 6, IF SAVE AREA SPACE

RPL RECORD



////////// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

EXTENDED END RECORD

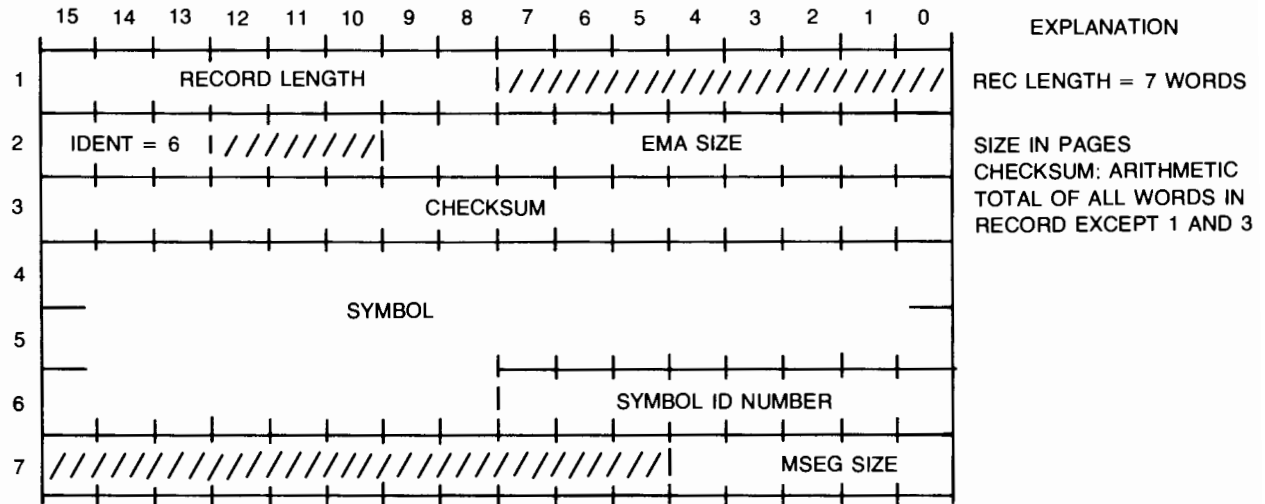


///// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

MR IS MEMORY SPACE

- = 0, IF ABSOLUTE SPACE
- = 1, IF PROGRAM RELOCATABLE SPACE
- = 2, IF BASE PAGE RELOCATABLE SPACE
- = 3, IF COMMON RELOCATABLE SPACE
- = 4, <RESERVED>
- = 5, IF EMA SPACE
- = 6, IF SAVE AREA SPACE

EMA RECORD

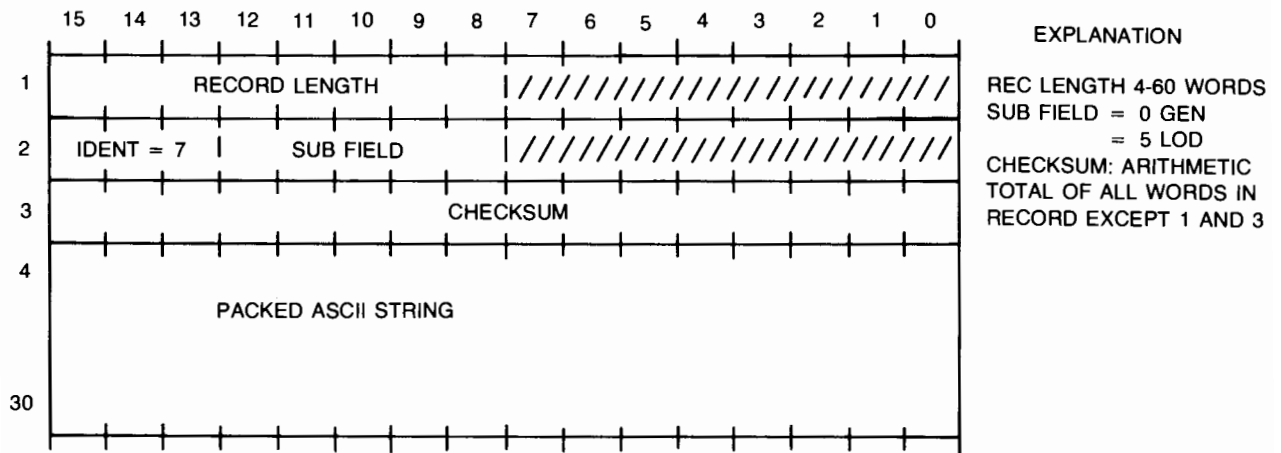


EXPLANATION
 REC LENGTH = 7 WORDS
 SIZE IN PAGES
 CHECKSUM: ARITHMETIC
 TOTAL OF ALL WORDS IN
 RECORD EXCEPT 1 AND 3

///// MEANS ZERO-FILLED WHEN RECORD IS GENERATED.

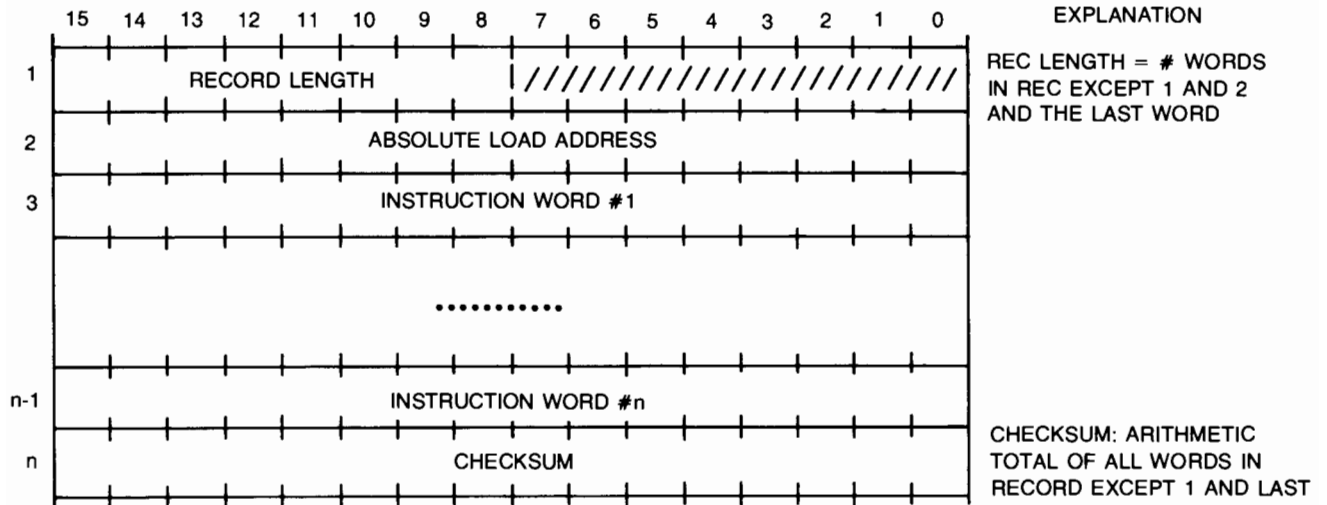
MSEG SIZE IS IN PAGES, $1 \leq \text{MSEG SIZE} \leq 32$

LOADER/GENERATOR INFORMATION RECORD



EXPLANATION
 REC LENGTH 4-60 WORDS
 SUB FIELD = 0 GEN
 = 5 LOD
 CHECKSUM: ARITHMETIC
 TOTAL OF ALL WORDS IN
 RECORD EXCEPT 1 AND 3

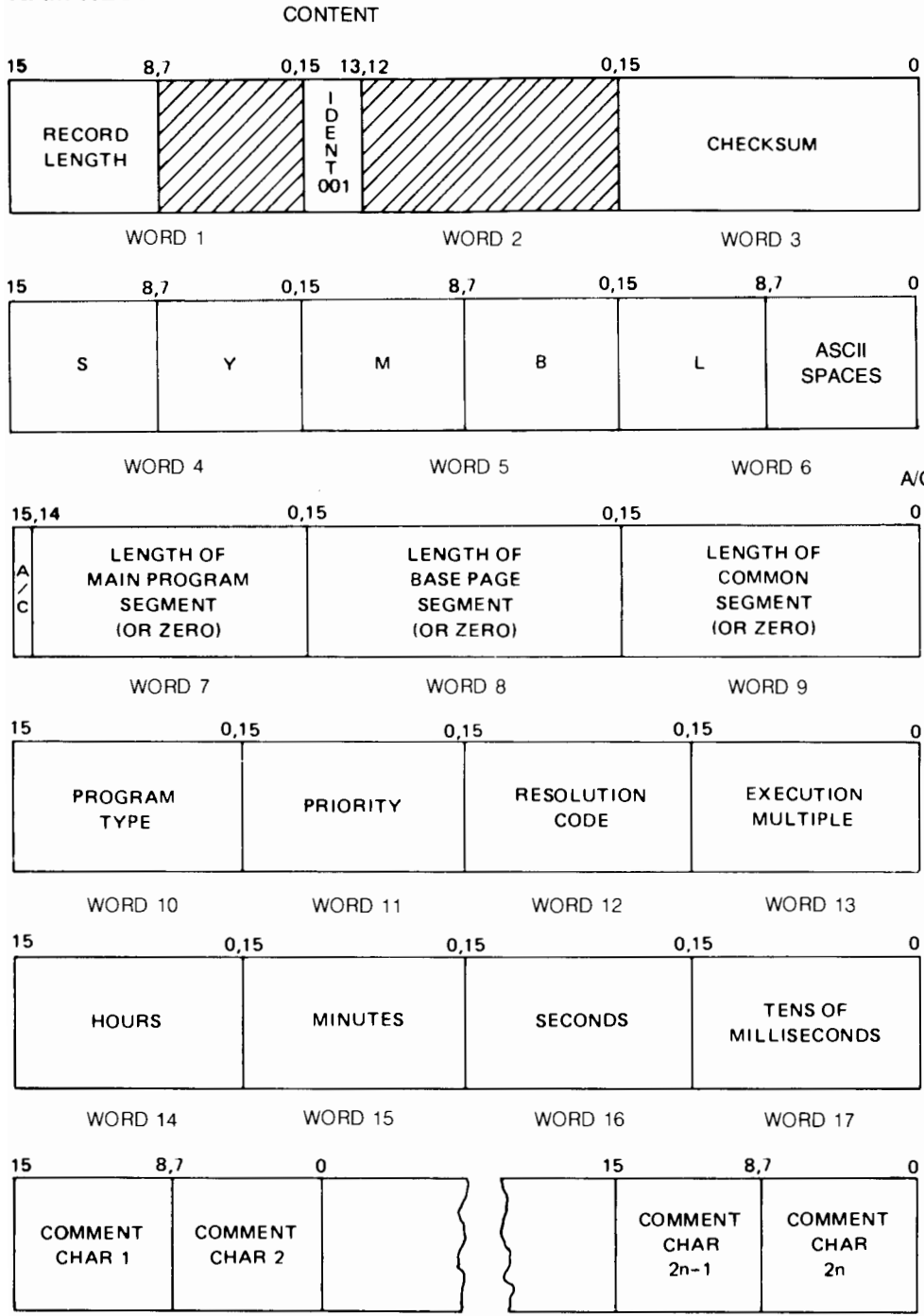
ABSOLUTE FORMAT



ABSOLUTE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW.

INSTRUCTION WORDS: ABSOLUTE INSTRUCTIONS OR DATA.

NAM RECORD



EXPLANATION

RECORD LENGTH = 9-60 WORDS

IDENT = 001

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS IN RECORD EXCLUDING WORDS 1 AND 3.

SYMBL: FIVE CHARACTER NAME OF PROGRAM

A/C: BINARY TAPE PRECESSION

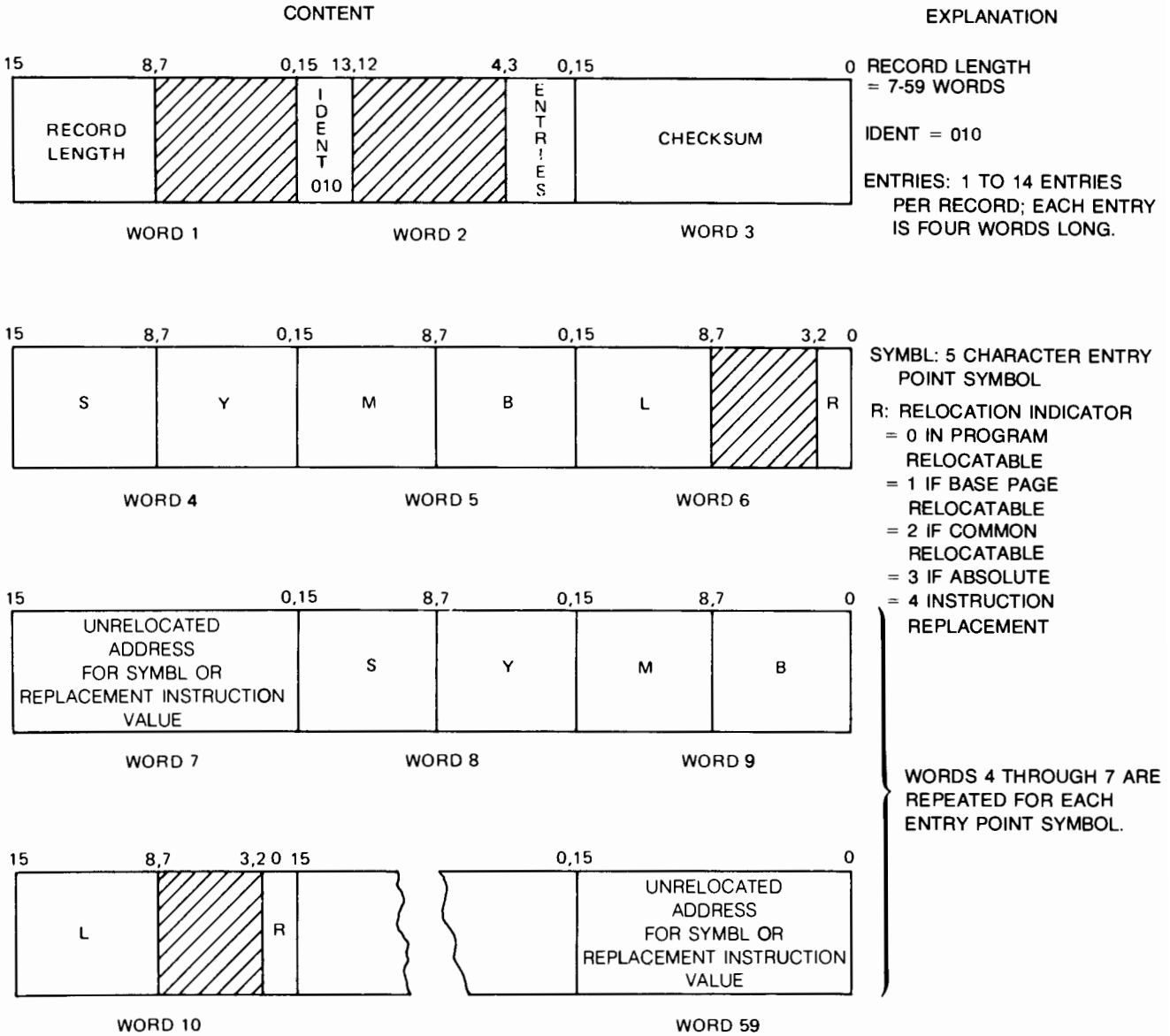
= 0 IF ASSEMBLER PRODUCED OR LENGTH IS EXACT.

= 1 IF COMPILER PRODUCED AND LENGTH IS UNKNOWN.

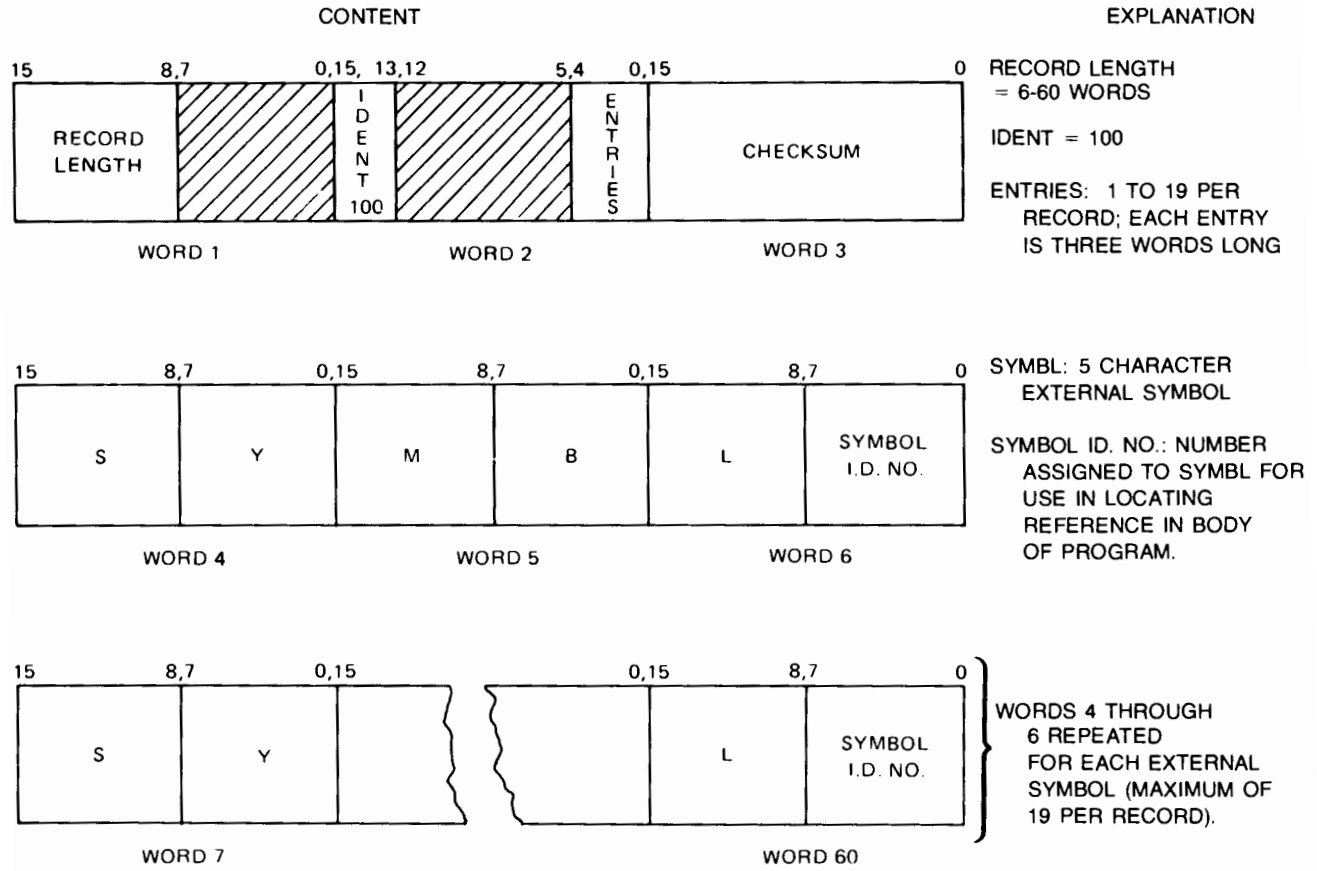
HATCH-MARKED AREAS SHOULD BE ZERO-FILLED WHEN THE RECORDS ARE GENERATED

WORD *n*
(*n* ≤ 60)

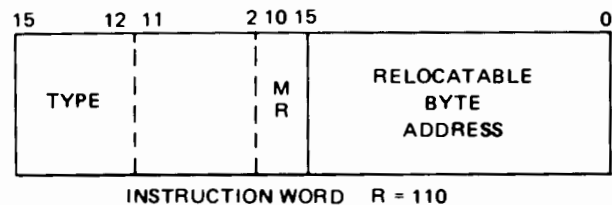
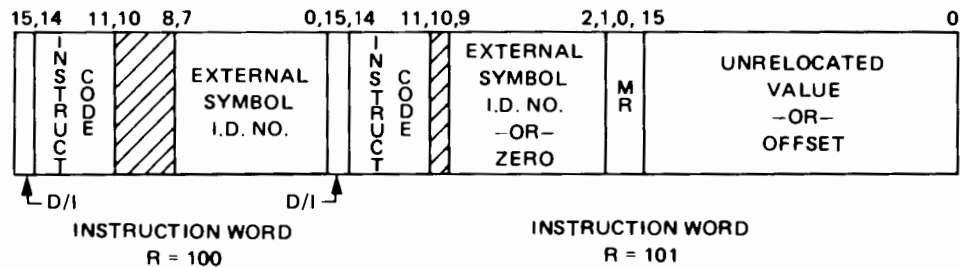
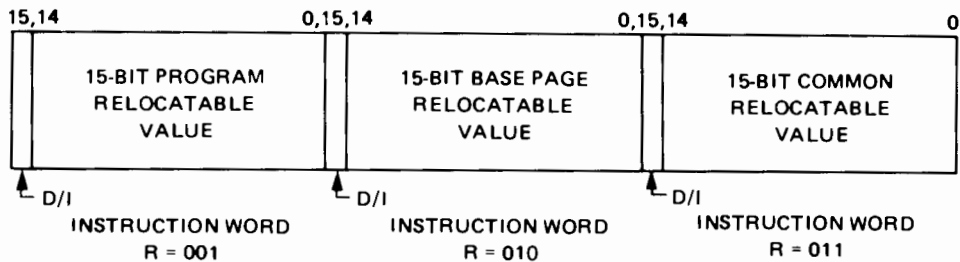
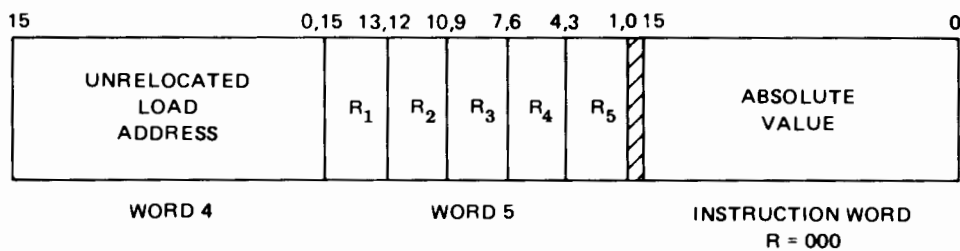
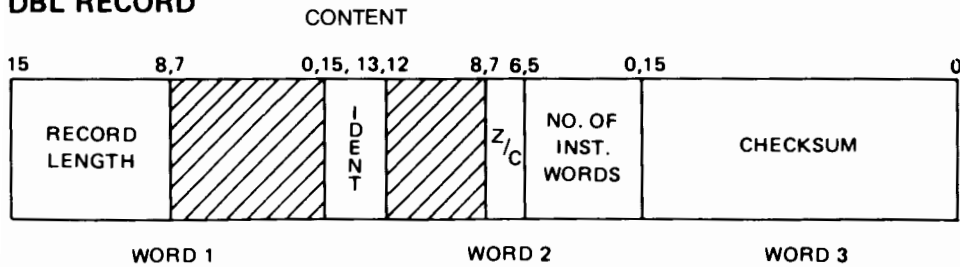
ENT RECORD



EXT RECORD



DBL RECORD



EXPLANATION

RECORD LENGTH = 6-60 WORDS

IDENT = 011

Z/C: RELOCATION OF LOAD ADDRESS

= 0 FOR BASE PAGE

= 1 FOR PROGRAM

= 2 FOR ABSOLUTE

= 3 FOR COMMON

NO. OF INST. WORDS: 1 TO 45 LOADABLE INSTRUCTION WORDS PER RECORD

RELOCATABLE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW;

R's: RELOCATION INDICATORS:

000 = ABSOLUTE

001 = 15-BIT PROGRAM RELOCATABLE

010 = 15-BIT BASE PAGE RELOCATABLE

011 = 15-BIT COMMON RELOCATABLE

100 = EXTERNAL REFERENCE

101 = MEMORY REFERENCE

110 = BYTE ADDRESS

R₁ IS RELOCATION INDICATOR FOR INSTRUCTION WORD₁; R₂, FOR INSTRUCTION WORD₂; ETC.

D/I: INDIRECT ADDRESSING

0 = DIRECT

1 = INDIRECT

MEMORY REFERENCE INSTRUCTIONS USE TWO WORDS, WITHIN THE TWO-WORD GROUP "MR" INDICATES RELOCATABILITY OF OPERAND SPECIFIED IN SECOND WORDS:

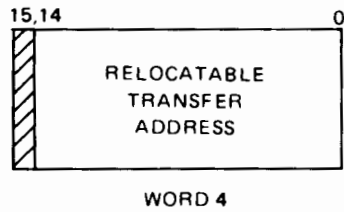
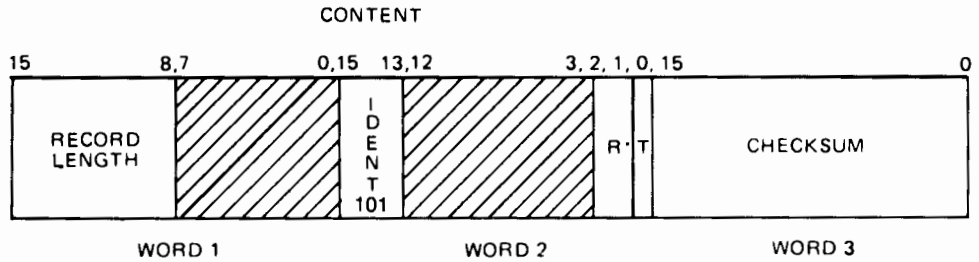
00 = PROGRAM RELOCATABLE

01 = BASE PAGE RELOCATABLE

10 = COMMON RELOCATABLE

11 = ABSOLUTE

END RECORD



EXPLANATION

RECORD LENGTH = 4 WORDS
IDENT = 101

R: RELOCATION INDICATOR FOR TRANSFER ADDRESS

- = 0 IF PROGRAM RELOCATABLE
- = 1 IF BASE PAGE RELOCATABLE
- = 2 IF COMMON RELOCATABLE
- = 3 IF ABSOLUTE

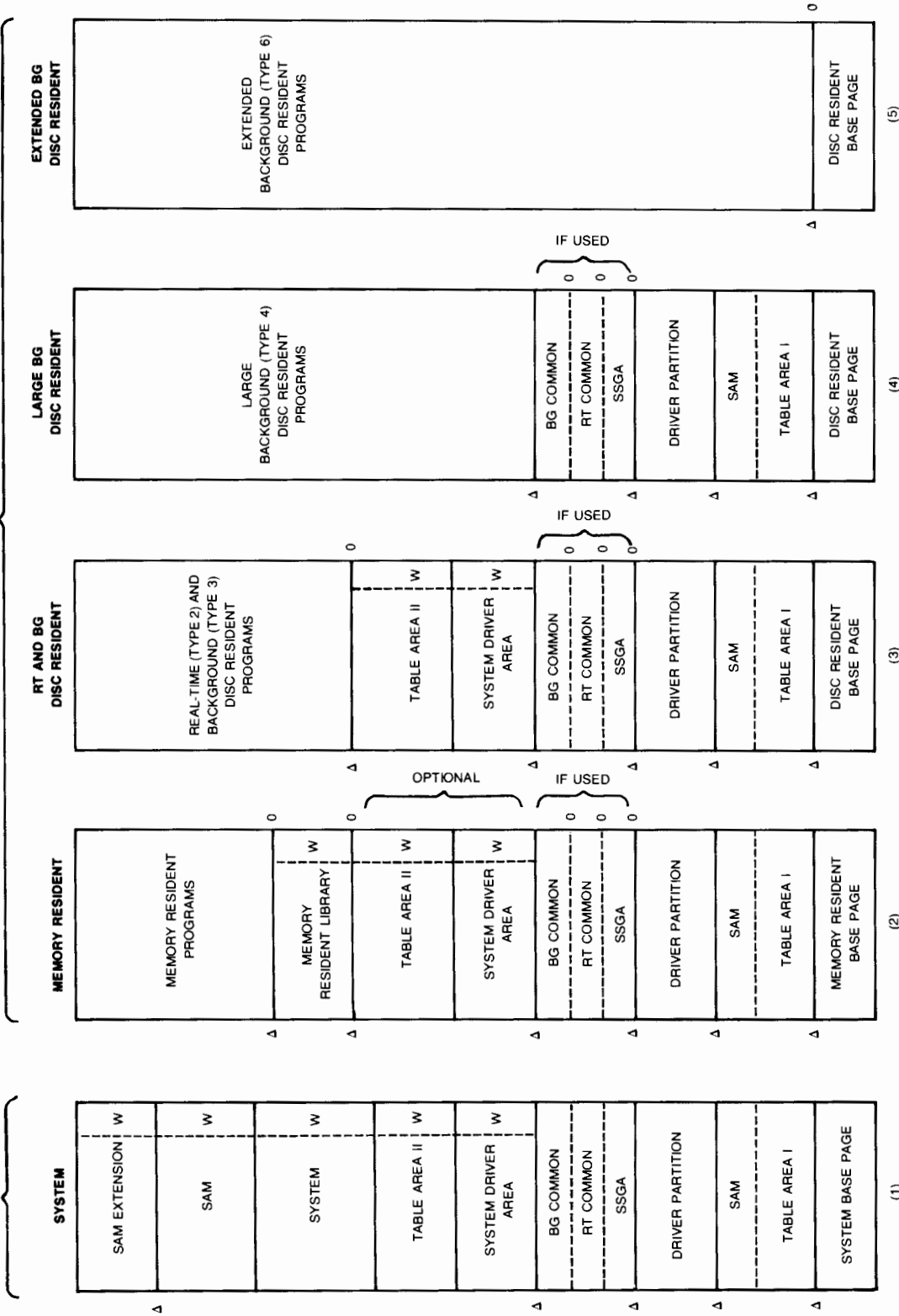
T: TRANSFER ADDRESS INDICATOR

- = 0 IF NO TRANSFER ADDRESS IN RECORD
- = 1 IF TRANSFER ADDRESS PRESENT

Appendix C
RTE-6/VM 32K-Word Logical Memory
Configurations

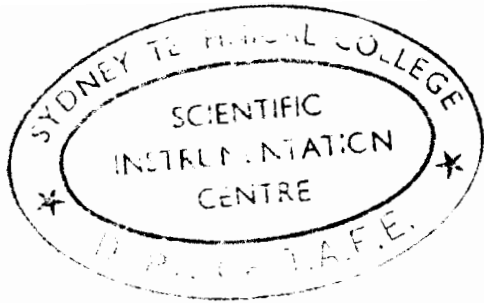
DESCRIBED BY
SYSTEM MAP

FOUR POSSIBLE CONFIGURATIONS DESCRIBED
BY USER MAP



(1)
Δ = PAGE BOUNDARIES
W = WRITE PROTECT
0 = MEMORY PROTECT FENCE SETTINGS

8100-31 Figure C-1 RTE-6/VM 32K-word Logical Memory Configurations



Appendix D Sample Program

This Appendix present a sample program, with source code, SGMTR output, and SXREF output. A loadmap for this program is shown in Chapter 4.

The program output shown in this section is a sample only. Output on various systems will differ according to system configuration and current software revision.

Source Code

```
MACRO,L
* Source for file %MAIN
    NAM MAIN,3,90          EXAMPLE: CHAPTER 4, APPENDIX D
    EXT SUB1,SUB2,SUB3,SUB4,EXEC
    ENT MAIN,SUB5,TBUFF
MAIN  JSB SUB1
      JSB SUB2
      JSB SUB3
      JSB SUB4
      JSB EXEC
      DEF *+2
      DEF SIX
SIX   DEC 6
      BSS 500
TBUFF BSS 15
SUB5  NOP
      JMP SUB5,I
      END MAIN

MACRO,L
* Source for file %SUB1
    NAM SUB1,7
    EXT SUB5,SUB6,SUB7
    ENT SUB1
SUB1  NOP
      JSB SUB5
      JSB SUB6
      JSB SUB7
      JMP SUB1,I
      BSS 200
      END
```

Sample Program

```
MACRO,L
* Source for file %SUB2
  NAM SUB2,7
  EXT SUB8
  ENT SUB2,SUB3
SUB2 NOP
  JMP SUB2,I
  BSS 200
SUB3 NOP
  JSB SUB8
  JMP SUB3,I
  BSS 200
  END

MACRO,L
* Source for file %SUB4
  NAM SUB4,7
  EXT SUB5,SYCON,TBUFF
  ENT SUB4
SUB4 NOP
  JSB SUB5
  JSB SYCON
  DEF *+3
  DEF TBUFF
  DEF .15
  JMP SUB4,I
.15 DEC 15
  BSS 200
  END

MACRO,L
* Source for file %SUB6
  NAM SUB6,7
  EXT SUB9,SUB11
  ENT SUB6,SUB10
SUB6 NOP
  JSB SUB9
  JSB SUB10
  JSB SUB11
  JMP SUB6,I
  BSS 200
SUB10 NOP
  JMP SUB10,I
  BSS 200
  END
```

Sample Program

```
MACRO,L
* Source for file %SUB7
  NAM SUB7,7
  EXT SUB9,SUB11
  ENT SUB7
SUB7 NOP
  JSB SUB9
  JSB SUB11
  JMP SUB7,I
  BSS 200
  END
```

```
MACRO,L
* Source for file %SUB11
  NAM SUB11,7
  ENT SUB11
SUB11 NOP
  JMP SUB11,I
  BSS 200
  END
```

```
MACRO,L
* Source for %LIB1
  NAM SUB8,7
  ENT SUB8
SUB8 NOP
  JMP SUB8,I
  END
  NAM SUB12,7
  ENT SUB12
  BSS 200
SUB12 NOP
  JMP SUB12,I
  BSS 200
  END
```



Sample Program

```
MACRO,L
* Source for %LIB2
  NAM SUB9,7
  ENT SUB9
  EXT FTIME,TBUFF
SUB9  NOP
      JSB FTIME
      DEF *+1
      DEF TBUFF
      JMP SUB9,I
      BSS 200
      END
      NAM SUB13,7
      ENT SUB13
      BSS 200
SUB13 NOP
      JMP SUB13,I
      BSS 200
      END
```

Sample Program

SGMTR Output

Below is the SGMTR output for the preceding program. (All relocatable files were merged into the file \$EXMPL.)

```
* RU,SGMTR,$EXMPL,#EXMPL::JT,2,MAIN,D
* 1:10 PM SAT., 24 OCT., 1981
LI, $EXMPL
*SGMTR: MODULE #      1  MAIN          EXAMPLE: CHAPTER 4, APPENDIX D
OP,BP
*SGMTR: MODULE #      2  SUB1
*SGMTR: MODULE #      3  SUB2
*SGMTR: MODULE #      4  SUB4
*SGMTR: MODULE #      5  SUB6
*SGMTR: MODULE #      6  SUB7
*SGMTR: MODULE #      7  SUB11
*SGMTR: MODULE #      8  SUB8
*SGMTR: MODULE #      9  SUB9
*SGMTR: MODULE #     10  FTIME          92084-1X032 REV.2121 780731
*SGMTR: MODULE #     11  SYCON          92084-1X047 REV.2121 780921
*SGMTR: MODULE #     12  $LOC$          92084-1X415 REV.2121 810723
*SGMTR: MODULE #     13  .RRGR          92084-1X419 REV.2121 810723
*SGMTR: MODULE LOAD COMPLETED
*SGMTR: COMMON BLOCKS ARE:
*TOTAL PROGRAM SIZE IN DECIMAL          2635
*SGMTR: WARNING! PATH LIMIT OVERFLOW!
*SGMTR: OFFPATH REFERENCE CHECK
*SGMTR:          9 NODES CREATED
*SGMTR: CREATING OUTPUT LISTING
*****
*
M
*
* NODE SIZE IS          1024 WORDS (          2000B)
*****
*   MAIN          EXAMPLE: CHAPTER 4, APPENDIX D
* MODULE SIZE IS     525 WORDS (  1015B)
NA,MAIN
NA,SUB5
NA,TBUFF
* THE PREVIOUS MODULE MAY NOT BE DUPLICATED
*
*   $LOC$          92084-1X415 REV.2121 810723
* MODULE SIZE IS     42 WORDS (    52B)
SY,$LOC$
SY,$LOD$
```

Sample Program

```
* THE PREVIOUS MODULE MAY NOT BE DUPLICATED
*
*   .RRGR           92084-1X419 REV.2121 810723
* MODULE SIZE IS   29 WORDS (    35B)
SY,.RRGR
SY,.SVRG
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
*
*****
*
D.1
*
* NODE SIZE IS           224 WORDS (          340B)
*****
*   SUB1
* MODULE SIZE IS   205 WORDS (    315B)
NA,SUB1
*
*****
*
D.1.1
*
* NODE SIZE IS           425 WORDS (          651B)
*****
*   SUB6
* MODULE SIZE IS   407 WORDS (    627B)
NA,SUB10
NA,SUB6
*
*****
*
D.1.1.1
*
* NODE SIZE IS           393 WORDS (          611B)
*SGMTR: ...PATH LIMIT OVERFLOW!
*****
*   SUB9
* MODULE SIZE IS   205 WORDS (    315B)
NA,SUB9
*
*****
*
D.1.1.2
*
* NODE SIZE IS           204 WORDS (          314B)
*****
*   SUB11
* MODULE SIZE IS   202 WORDS (    312B)
NA,SUB11
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
*
```

Sample Program

```
*****
*
D.1.2
*
* NODE SIZE IS          229 WORDS (          345B)
*****
*   SUB7
* MODULE SIZE IS      204 WORDS (   314B)
NA,SUB7
*
*****
*
D.1.2.1
*
* NODE SIZE IS          393 WORDS (          611B)
*****
*   SUB9
* MODULE SIZE IS      205 WORDS (   315B)
NA,SUB9
*
*****
*
D.1.2.2
*
* NODE SIZE IS          204 WORDS (          314B)
*****
*   SUB11
* MODULE SIZE IS      202 WORDS (   312B)
NA,SUB11
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
*
*****
*
D.2
*
* NODE SIZE IS          654 WORDS (        1216B)
*****
*   SUB2
* MODULE SIZE IS      405 WORDS (   625B)
NA,SUB2
NA,SUB3
*
*   SUB4
* MODULE SIZE IS      208 WORDS (   320B)
NA,SUB4
*
END
```


Sample Program

SXREF Output

Below is SXREF output for sample program using the preceding command file created by SGMTR.

NODE LOADER
ORDINAL COMMAND

```
* RU,SGMTR,$EXMPL,#EXMPL::JT,2,MAIN,D
* 1:10 PM SAT., 24 OCT., 1981
LI, $EXMPL
* *SGMTR: MODULE #      1  MAIN  EXAMPLE: CHAPTER 4, APPENDIX D
  OP,BP
*SGMTR: MODULE #      2  SUB1
*SGMTR: MODULE #      3  SUB2
*SGMTR: MODULE #      4  SUB4
*SGMTR: MODULE #      5  SUB6
*SGMTR: MODULE #      6  SUB7
*SGMTR: MODULE #      7  SUB11
*SGMTR: MODULE #      8  SUB8
*SGMTR: MODULE #      9  SUB9
* *SGMTR: MODULE #     10  FTIME  92084-1X032 REV.2121 780731
* *SGMTR: MODULE #     11  SYCON  92084-1X047 REV.2121 780921
* *SGMTR: MODULE #     12  $LOC$  92084-1X415 REV.2121 810723
* *SGMTR: MODULE #     13  .RRGR  92084-1X419 REV.2121 810723
*SGMTR: MODULE LOAD COMPLETED
*SGMTR: COMMON BLOCKS ARE:
*TOTAL PROGRAM SIZE IN DECIMAL          2635
*SGMTR: WARNING! PATH LIMIT OVERFLOW!
*SGMTR: OFFPATH REFERENCE CHECK
*SGMTR:          9 NODES CREATED
*SGMTR: CREATING OUTPUT LISTING
*****
*
0  M
*
* NODE SIZE IS          1024 WORDS (          2000B)
*****
*   MAIN                      EXAMPLE: CHAPTER 4, APPENDIX D
* MODULE SIZE IS      525 WORDS (  1015B)
NA,MAIN
NA,SUB5
NA,TBUFF
* THE PREVIOUS MODULE MAY NOT BE DUPLICATED
*
```

Sample Program

```
* $LOC$          92084-1X415 REV.2121 810723
* MODULE SIZE IS 42 WORDS ( 52B)
SY,$LOC$
SY,$LOD$
* THE PREVIOUS MODULE MAY NOT BE DUPLICATED
*
* .RRGR          92084-1X419 REV.2121 810723
* MODULE SIZE IS 29 WORDS ( 35B)
SY,.RRGR
SY,.SVRG
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
*
*****
*
```

- 1 D.1
*
* NODE SIZE IS 224 WORDS (340B)

* SUB1
* MODULE SIZE IS 205 WORDS (315B)
NA,SUB1
*

*

- 2 D.1.1
*
* NODE SIZE IS 425 WORDS (651B)

* SUB6
* MODULE SIZE IS 407 WORDS (627B)
NA,SUB10
NA,SUB6
*

*

- 3 D.1.1.1
*
* NODE SIZE IS 393 WORDS (611B)
*SGMTR: ...PATH LIMIT OVERFLOW!

* SUB9
* MODULE SIZE IS 205 WORDS (315B)
NA,SUB9
*

*

- 4 D.1.1.2

Sample Program

```
*
* NODE SIZE IS          204 WORDS (          314B)
*****
*   SUB11
* MODULE SIZE IS      202 WORDS (   312B)
NA,SUB11
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
*
*****
*

5   D.1.2
*
* NODE SIZE IS          229 WORDS (          345B)
*****
*   SUB7
* MODULE SIZE IS      204 WORDS (   314B)
NA,SUB7
*
*****
*

6   D.1.2.1
*
* NODE SIZE IS          393 WORDS (          611B)
*****
*   SUB9
* MODULE SIZE IS      205 WORDS (   315B)
NA,SUB9
*
*****
*

7   D.1.2.2
*
* NODE SIZE IS          204 WORDS (          314B)
*****
*   SUB11
* MODULE SIZE IS      202 WORDS (   312B)
NA,SUB11
* THE PREVIOUS MODULE REFERENCES NO OTHER MODULES
*
*****
*

8   D.2
*
* NODE SIZE IS          654 WORDS (         1216B)
*****
*   SUB2
* MODULE SIZE IS      405 WORDS (   625B)
```

Sample Program

```

NA,SUB2
NA,SUB3
*
*   SUB4
* MODULE SIZE IS   208 WORDS (   320B)
NA,SUB4
*
END
    
```

```

SXREF: UNDEFINED ENTRY POINT: .CNOD           IN NODE       0
      NODE ORDINAL NUMBER      0           M
    
```

```

*MODULE RECORD* MAIN           EXAMPLE: CHAPTER 4, APPENDIX D
TYPE      :      3
PROGRAM   :    525 LOCAL EMA :           0
BASE PAGE:      0 LOCAL SAVE:      0
COMMON    :      0 PURE CODE :      0

ENTRY PTS:  MAIN              SUB5              TBUFF

EXTERNALS:  EXEC              SUB1              SUB2
            SUB3              SUB4

END        :      0
    
```

```

*MODULE RECORD* $LOC$          92084-1X415 REV.2121 810723
TYPE      :      7
PROGRAM   :    42 LOCAL EMA :           0
BASE PAGE:      0 LOCAL SAVE:      0
COMMON    :      0 PURE CODE :      0

ENTRY PTS:  $LOC$            $LOD$

EXTERNALS:  .SVRG            .RRGR            $LOC
            .CNOD            EXEC

END        :      0
    
```

```

*MODULE RECORD* .RRGR          92084-1X419 REV.2121 810723
TYPE      :      7
PROGRAM   :    29 LOCAL EMA :           0
BASE PAGE:      0 LOCAL SAVE:      0
COMMON    :      0 PURE CODE :      0

ENTRY PTS:  .RRGR            .SVRG
    
```

Sample Program

END : 0

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

```
-----
MAIN          MAIN
SUB5          MAIN          SUB1          : 1  SUB4          : 8
TBUFF        MAIN          SUB9          : 3  SUB9          : 6
              SUB4          : 8
$LOC$        $LOC$
$LOD$        $LOC$
.RRGR        .RRGR        $LOC$          : 0
.SVRG        .RRGR        $LOC$          : 0
```

MODULE ! REFERENCED BY:

```
-----
MAIN          SUB1          : 1  SUB9          : 3
              SUB9          : 6  SUB4          : 8
$LOC$
.RRGR        $LOC$          : 0
```

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

```
-----
SUB2          7          SUCC
SUB3          7          SUCC
SUB4          7          SUCC
```

UNDEFS ! REFERENCED BY:

```
-----
.CNOD          $LOC$
NODE ORDINAL NUMBER 1 D.1
```

MODULE RECORD SUB1

```
TYPE : 7
PROGRAM : 205 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0
```

ENTRY PTS: SUB1

EXTERNALS: SUB5 SUB6 SUB7

END : 1285

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

```
-----
SUB1          SUB1          MAIN          : 0
```

Sample Program

MODULE ! REFERENCED BY:

 SUB1 MAIN : 0

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
 EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

 SUB5 0 PRED
 SUB6 0 PRED
 SUB7 3 SUCC



UNDEFS ! REFERENCED BY:

NO UNDEFS
 NODE ORDINAL NUMBER 2 D.1.1

MODULE RECORD SUB6

TYPE : 7
 PROGRAM : 407 LOCAL EMA : 0
 BASE PAGE: 0 LOCAL SAVE: 0
 COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB10 SUB6

EXTERNALS: SUB11 SUB9

END : 514

ENTRY POINTS ! DEFINED IN MODULE: ! REFERENCED BY:

 SUB10 SUB6
 SUB6 SUB6 SUB1 : 1

MODULE ! REFERENCED BY:

 SUB6 SUB1 : 1

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
 EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

 SUB11 1 PRED
 SUB9 0 PRED

UNDEFS ! REFERENCED BY:

NO UNDEFS
 NODE ORDINAL NUMBER 3 D.1.1.1

Sample Program

MODULE RECORD SUB9
TYPE : 7
PROGRAM : 205 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB9

EXTERNALS: FTIME TBUFF

END : 1285

MODULE RECORD FTIME 92084-1X032 REV.2121 780731
TYPE : 7
PROGRAM : 186 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: FTIME

EXTERNALS: EXEC

END : 0

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

SUB9 SUB9 SUB6 : 2
FTIME FTIME SUB9 : 3

MODULE ! REFERENCED BY:

SUB9 SUB6 : 2
FTIME SUB9 : 3

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

TBUFF 0 PRED

UNDEFS ! REFERENCED BY:

.....NO UNDEFS
NODE ORDINAL NUMBER 4 D.1.1.2

MODULE RECORD SUB11
TYPE : 7

Sample Program

PROGRAM : 202 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB11

END : 514

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

SUB11 SUB11 SUB6 : 2

MODULE ! REFERENCED BY:

SUB11 SUB6 : 2

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

UNDEFS ! REFERENCED BY:

.....NO UNDEFS
NODE ORDINAL NUMBER 5 D.1.2

MODULE RECORD SUB7

TYPE : 7
PROGRAM : 204 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB7

EXTERNALS: SUB11 SUB9

END : 1028

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

SUB7 SUB7 SUB1 : 1

MODULE ! REFERENCED BY:

SUB7 SUB1 : 1

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

Sample Program

SUB11 1 PRED
 SUB9 0 PRED

UNDEFS ! REFERENCED BY:

NO UNDEFS
 NODE ORDINAL NUMBER 6 D.1.2.1

MODULE RECORD SUB9

TYPE : 7
 PROGRAM : 205 LOCAL EMA : 0
 BASE PAGE : 0 LOCAL SAVE: 0
 COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB9

EXTERNALS: FTIME TBUFF

END : 1285

MODULE RECORD FTIME 92084-1X032 REV.2121 780731

TYPE : 7
 PROGRAM : 186 LOCAL EMA : 0
 BASE PAGE : 0 LOCAL SAVE: 0
 COMMON : 0 PURE CODE : 0

ENTRY PTS: FTIME

EXTERNALS: EXEC

END : 0

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

 SUB9 SUB9 SUB7 : 5
 FTIME FTIME SUB9 : 6

MODULE ! REFERENCED BY:

 SUB9 SUB7 : 5
 FTIME SUB9 : 6

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
 EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

TBUFF 0 PRED

Sample Program

UNDEFS ! REFERENCED BY:

.....NO UNDEFS
NODE ORDINAL NUMBER 7 D.1.2.2

MODULE RECORD SUB11

TYPE : 7
PROGRAM : 202 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB11

END : 514

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

SUB11 SUB11 SUB7 : 5

MODULE ! REFERENCED BY:

SUB11 SUB7 : 5

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

UNDEFS ! REFERENCED BY:

.....NO UNDEFS
NODE ORDINAL NUMBER 8 D.2

MODULE RECORD SUB2

TYPE : 7
PROGRAM : 405 LOCAL EMA : 0
BASE PAGE: 0 LOCAL SAVE: 0
COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB2 SUB3

EXTERNALS: SUB8

END : 771

MODULE RECORD SUB4

TYPE : 7

Sample Program

PROGRAM : 208 LOCAL EMA : 0
 BASE PAGE: 0 LOCAL SAVE: 0
 COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB4

EXTERNALS: SUB5 SYCON TBUFF

END : 2056

MODULE RECORD SUB8

TYPE : 7
 PROGRAM : 2 LOCAL EMA : 0
 BASE PAGE: 0 LOCAL SAVE: 0
 COMMON : 0 PURE CODE : 0

ENTRY PTS: SUB8

END : 514

MODULE RECORD SYCON 92084-1X047 REV.2121 780921

TYPE : 7
 PROGRAM : 15 LOCAL EMA : 0
 BASE PAGE: 0 LOCAL SAVE: 0
 COMMON : 0 PURE CODE : 0

ENTRY PTS: SYCON

EXTERNALS: .ENTR XLUEX

END : 0

ENTRY POINTS ! DEFINED IN MODULE:! REFERENCED BY:

SUB2	SUB2	MAIN	: 0
SUB3	SUB2	MAIN	: 0
SUB4	SUB4	MAIN	: 0
SUB8	SUB8	SUB2	: 8
SYCON	SYCON	SUB4	: 8

MODULE ! REFERENCED BY:

SUB2	MAIN	: 0	MAIN	: 0
SUB4	MAIN	: 0		
SUB8	SUB2	: 8		

Sample Program

SYCON SUB4 : 8

EXTERNALS SATISFIED OUTSIDE OF CURRENT NODE
EXTERNALS ! ORDINAL NUMBER ! PRED/SUCC

SUB5 0 PRED
TBUFF 0 PRED

UNDEFS ! REFERENCED BY:

.....NO UNDEFS

Appendix E

Linking Overview

Due to the addressing architecture of HP 1000 computers, instructions can only directly access data located in the current instruction page or in the program's base page (page 0). To access areas outside these two regions, programs must make use of locations in the current page or base page containing the address of the desired data or subroutine. These locations are called links. Programs make indirect references through links to access data or instructions outside their directly addressable area. The loader will automatically create links in two circumstances:

- * References to external entry points. MLLDR will create a link for each external entry point referenced in a program, except in two cases:
 1. The external resides on the same page as the reference instruction and current page linking (described below) is being used.
 2. The reference uses a DEF to an external with an initial offset.

In the two cases above, the references are direct (no link is used).

- * References to data and instructions located outside the current page. Since the relocation of programs in memory depends on many factors (e.g., program type, size, COMMON, etc.) it would be very difficult for a program to make provisions for linking and still make efficient use of memory. Therefore, MLLDR allocates a link whenever an instruction makes a direct reference outside the current page.

In all cases where links are generated, the referencing instruction is modified to make an indirect reference through the link. An example of base page linking is shown in Figure E-1. In this example, two instructions reference areas outside their pages. When MLLDR detects this condition, it will allocate base page links and modify the instructions to use the links.

Linking Overview

When using current page linking, links are allocated on the same page as the referencing instruction. An example is shown in Figure E-2. Current page links are allocated in two areas: immediately preceding the module and immediately following the module, where a module begins with a NAM record and ends with an END record.

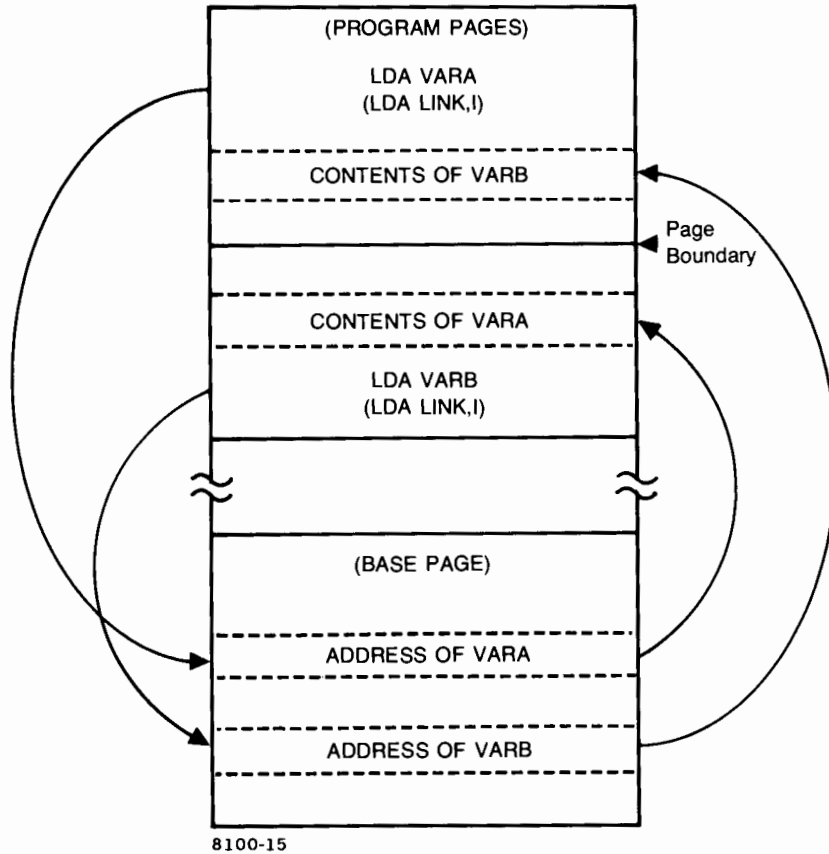


Figure E-1. Base Page Linking

Linking Overview

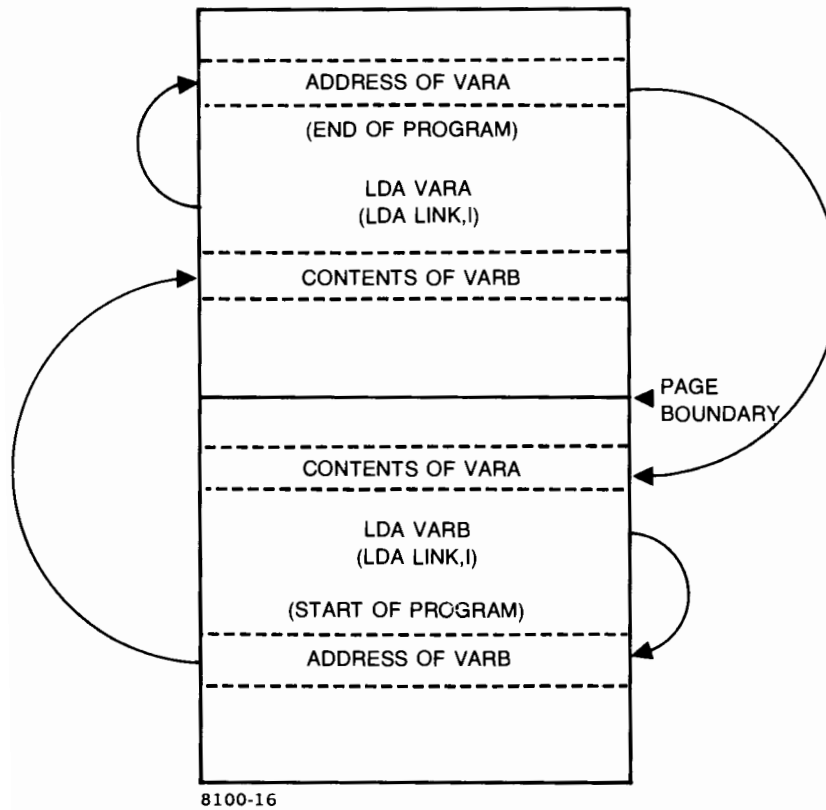


Figure E-2. Current Page Linking

Appendix F

Summary of COMMON Types

Table F-1. Summary of COMMON Types

NAME	TREATED BY RTE	LOADED TYPE	LOCATION	COMMANDS REQUIREMENTS	ACCESS BY	PROTECTION	RECOMMENDED
Blank COMMON	As COMMON	Local	Beginning of program space	OP, NC (Default)	Loaded program Segments and Subroutine only	Protected from all other programs	Normal Use
		Back-ground	System area	OP, SC OP, RC	Any program using same area	Protected from programs not using RT, BG and SSGA	Program-to-Program Communication or special drivers
		Real Time	System area	OP, RC OP, SC	Any program using same area	Protected from programs not using RT or SSGA	Same as above
Labeled COMMON	As Allocate or External Variable	Local	Anywhere in program space	(Default) 4X: REL, block data 7X: optionally, REL, block data	Loaded program Segments and Subroutines only	Protected from all other programs	Normal use
		System	SSGA	OP, SS 7X: no allocate, no block data	Any program using SSGA	Protected from programs not using SSGA	Only when absolutely necessary

Other Forms of Labeled COMMON

EMA resides in EMA area, or set at load time to reside in VMA (VM command) or in a shareable EMA partition (SH command). In FORTRAN 77 this area can contain multiple labeled COMMON blocks and program variables (local EMA). An EMA labeled COMMON block must be typed (as EMA) consistently throughout the program.

SAVE resides with the main program (not in segments or nodes). A SAVE labeled COMMON block should be typed (as SAVE) consistently throughout the program. The SAVE area can contain multiple labeled COMMON blocks and program variables (local SAVE). The loader SA command must be used to reserve sufficient space for all SAVE variables and SAVE labeled COMMON blocks in the program.

Index

@SGLD, 9-1

A

AB (abort), 5-2

aborting,

 INDXR, 6-42

 LOADR, 4-32

 MLLDR, 4-32

 SGMTR, 6-7

 SXREF, 6-24

ALLOCATE - see also NOALLOCATE

ALLOCATE, 4-19

AS (assign), 5-2

Assembly, 9-9

B

Background program, 2-15

backing store file, 2-18

base page link overflow, 6-6, 8-5, 8-7

base page links, 2-3, 2-4, 4-10, 4-24, 6-6, 7-2, E-2

base page,

 rotating, 4-24

 shared, 4-24

BG - see Background program

blank COMMON, see COMMON

block data subprogram, 6-8, 6-9, 6-10

breakpoint, 2-20

brother node, 3-2

C

calling sequence, 3-1

 sample, 3-6

characteristics of program types, 2-15

circular call chain - see also recursion

circular call chain, 3-19, 3-20

code input, 9-5

command file listing (SXREF), 6-27

command file, 3-8, 4-2, 4-5, 6-1

 sample, 3-11, 6-15, D-5

 shortening, 6-18

 syntax checking (SXREF), 6-23, 6-25

 writing, 3-8

- command summary, 4-3
- commands and command file, 4-2
 - MLS programs, 4-5
 - non-segmented programs, 4-5
- commands, INDXR, 6-41
 - AB, 6-42
 - CR, 6-42
 - EX, 6-42
 - IN, 6-42
 - LI, 6-42
 - TR, 6-43
- commands, MLLDR and LOADR, 5-1, 9-3
 - * (comment), 5-11
 - AB (abort), 5-2
 - AS (assign), 5-2
 - D (disc node specification), 5-2
 - DI (display undefined externals), 5-2
 - EC (echo commands to list output), 5-3
 - EN, EX, /E (end of command input), 5-3
 - FM (format), 5-3
 - FO (force load), 5-4
 - LI (user library file), 5-4
 - LL (list output), 5-4
 - LO (set relocation base), 5-5
 - M (memory node), 5-5
 - MS (multiple search), 5-6
 - NA (user library name), 5-6
 - OP (opcode), 5-7
 - PF (profile), 5-7
 - RE (relocate), 5-7
 - SA (save), 5-8
 - SE (search), 5-8
 - SH (shareable EMA), 5-9
 - SL (search user libraries), 5-9
 - SY (system library name), 5-10
 - SZ (size), 5-10
 - TR (transfer), 5-10
 - VS (VMA size), 5-11
 - WS (working set size), 5-11
- comment (*), 5-11
- COMMON, 2-16, 4-19, 4-30, 6-17, 7-2, 8-11
 - blank, 2-16
 - block types and sizes (SGMTR), 6-10
 - EMA, F-1
 - improper use of, 8-12
 - initialization, 3-22
 - labeled, 6-7, 6-9, 6-10
 - local, 2-16
 - named, 4-19
 - named SAVE, 4-19
 - ordinary, 6-8
 - requirements for SGMTR, 6-7

- reverse, 2-16
- SAVE, F-1
- summary of types, F-1
- system, 2-16
- type and size mismatches, 6-10
- unlabeled, 2-16

compilation considerations, 8-1
completion message, 4-29, 9-6
creating the MLS tree, 3-5
cross referencer, see SXREF
current node, 3-8, 4-21
current page links, 2-3, 3-2, 4-10, 4-18, E-3

D

D (disc node specification), 3-9, 5-2
data considerations, 3-22
data reference, 6-19, 8-9, 9-9
debugger, 1-3, 3-4

- force load, 4-30
- interface with MLLDR, 2-20

deleting permanent programs, 2-10
deletion, commands, 2-7
depth, 3-3
DI (display undefined externals), 5-2
disc nodes, 3-22, 6-6, 6-8, 6-19, 7-2, 8-11
disc-resident nodes, and memory-resident nodes, mixed, 3-17
duplicate program name, 4-17
duplication of modules, 3-17, 3-22, 6-3, 6-18, 6-19, 6-20, 8-9, 8-11
dynamic buffer area - see also SZ command
dynamic buffer area, 2-15, 7-8, 8-2

E

EB - see Extended Background
EC (echo commands to list output), 4-21, 5-3
EMA, 7-8, 70-1

- and Shareable EMA, 2-17
- labeled COMMON, 6-10
- size, 2-15

empty module, 4-18
EN, EX, or /E (end of command input), 5-3
end of partition, 7-8
end of program, 7-8
END record, 3-2, 6-28
END statement, 3-2, 6-5
ENT record, 6-41
entry point, 3-9

- FORTTRAN, 6-4
- MACRO, 6-5

- main, 6-4
- non-main, 6-5
- Pascal, 6-4
- errors,
 - base page link overflow, 6-6
 - duplicate program name, 4-17
 - expected order of commands, 5-1, 6-26
 - INDXR, A-21
 - LOADR, A-1
 - memory overflow, 6-3, 6-6, 7-1
 - MLLDR, A-1
 - multilevel segmentation, 4-14
 - offpath reference, 6-20, 6-25, 8-12
 - other, 4-16
 - path overflow, 6-3, 6-20
 - reporting, 4-13
 - runtime, 8-11
 - SGMTR, A-12
 - summary of SGMTR, 6-21
 - summary of SXREF, 6-39
 - SXREF, A-17
 - syntax checking mode (SXREF), 6-25
 - waiting for resources, 4-17
 - warnings, 4-16, 7-8
- EXEC 8, 2-6, 9-1
- Extended Background program, 2-15, 3-4
- extended code space, 0-3

F

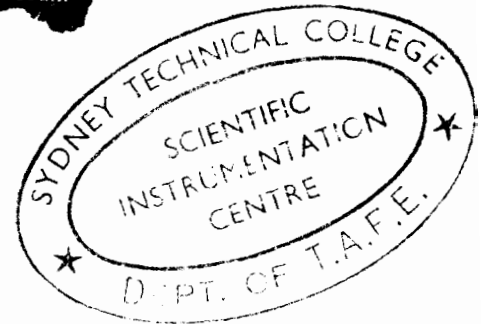
- fault, 3-3, 4-31
 - node, 2-20
 - overhead, 2-20
- FM (format), 5-3
 - default, 5-3
- FO (force load), 5-4
- force command, 4-14
- force load, 3-15, 4-30
- FORTTRAN, 3-23, 6-4, 6-8, 6-9, 6-10, 6-19, 6-20, 8-2, 8-11, 9-9, 70-1

I

- ID segment, 2-8, 2-9, 2-10, 9-2
 - requirements for program loading, 2-11
- INDXR, 6-41
 - commands, 6-41
 - error messages, A-21
 - examples, 6-43
 - operation, 6-41

Index-4

L



labeled COMMON - see COMMON
 language considerations, 3-22, 8-1
 Large Background program, 2-15
 LB - see Large Background program
 LE option, 4-24
 leaf node, 3-2
 LI (user library file), 3-9, 5-4
 linkage type, 5-3
 linking - see also base page linking
 linking - see also current page linking
 linking, 2-3
 base page, 2-3, 2-4
 current page, 2-3
 mixed, 2-3, 2-4
 overview, E-1
 reported in completion message, 4-29, 9-6
 list output, 5-4
 LO (set relocation base), 5-5
 load point, 4-18
 load-on-call, 1-2, 2-5, 2-20, 3-1, 3-16
 loading from a logical unit, 4-2, 9-5
 loading large programs - see also reducing load time
 loading large programs - see also SGMTR
 loading large programs, 6-1, 8-1, 8-4, 8-5
 base page link overflow, 8-7
 evaluate the program, 8-3
 load time errors, 8-6
 reducing links, 8-10
 reducing load time, 6-18, 8-12
 reducing path size, 8-8
 runtime errors, 8-11
 segmenting using SGMTR, 8-4
 loadmap,
 default, 4-18
 EC option, 4-21, 8-10
 LE option, 4-24
 sample, 4-18
 LOADR errors, A-1
 LOADR, 5-2, 9-1
 code input, 9-5
 command input, 9-3
 commands, 9-3
 completion message, 9-6
 converting to multilevel segmentation, 9-9
 functions, 1-1, 2-1
 memory layout, 9-7
 operation, 9-3
 runstring, 9-3

- single-level segmentation, 9-1
- summary of commands, 9-4
- LOC - see load-on-call
- local COMMON, see COMMON
- local data, 6-19, 8-11
- local SAVE area, 4-19
- logical address space - see also longest path
- logical address space - see also memory maps
- logical address space, 1-3
- logical unit, loading from, 4-2
- longest path, 2-15, 3-3, 3-16, 4-29

M

- M (memory node), 3-8, 5-5
- MACRO, 6-5, 6-8, 6-10
- main entry point, 6-2
- main programs, 2-14
- maximum pathlength and tree size, 3-16
- memory allocation, 1-3, 2-13, 7-1
 - base page to root, 7-2
 - LOADR, 9-7
 - NAM record, 2-13
 - program nodes, 7-2
 - program nodes, disc-resident only, 7-2
 - program nodes, memory and disc nodes, 7-3
 - program nodes, memory-resident only, 7-3
 - program nodes, root node only, 7-2
 - program types, 2-13
 - VMA/EMA area, 7-8
 - VMA/EMA page table, 7-8
- memory maps, 7-1, 9-7, C-1
- memory nodes, 6-6
- memory overflow, 6-3, 6-6, 7-1, 8-3, 8-5, 8-7
- memory overhead - see overhead
- memory usage criteria, 3-5
- memory-image format, 2-2
- memory-resident and disc-resident nodes, mixed, 3-17
- MERGE utility, 6-41
- merged and indexed files, 6-11
- mixed linking, 2-3, 2-4, 4-10, 6-6
- MLLDR,
 - errors, A-1
 - interface with debugger, 2-20
 - modes of operation, 4-1
 - operation overview, 1-1
 - processing sample trace, 3-13
 - runstring operation, 4-6
 - runstring operation examples, 4-11

- MLLDR, runstring parameters, 4-6
 - command, 4-6
 - format, 4-9
 - input, 4-7
 - list, 4-7
 - opcode, 4-8
 - partition, 4-10
 - profile, 4-10
 - size, 4-10
- MLS - see multilevel segmentation
- MLS commands, 3-8
 - D (disc node), 3-9
 - LI (user library file), 3-9
 - M (memory node), 3-8
 - MS (multiple search), 3-10
 - NA (user library name), 3-9
 - RE (relocate), 3-9
 - SE (search), 3-10
 - SL (search user libraries), 3-10
 - SY (system library name), 3-9
- MLS programs, 4-5
- MLS tree, 3-2
 - creating, 3-5
 - restrictions, 3-16
- MLS-LOC, see multilevel segmentation, load-on-call
- MLSDB - see debugger
- module, 3-2
- MS (multiple search), 3-10, 5-6
- MSEG size, 4-29, 7-9
- multilevel segmentation - see also MLLDR
- multilevel segmentation, 3-1, 7-9
 - benefits of, 2-6
 - command file, 3-11
 - converting to, 9-9
 - data reference, 3-16
 - duplication of modules, 3-17
 - errors, 4-14
 - load-on-call, 2-5
 - maximum pathlength, 3-16
 - maximum tree size, 3-16
 - overview, 3-1
 - reducing path size, 8-8
 - restrictions, 3-16, 6-19, 6-20
 - restrictions on duplicating modules, 3-19
 - structure, 3-1
 - terminology, 3-2
 - tree structure, 3-7
 - writing the command file, 3-8

N

NA (user library name), 3-9, 5-6
NAM record, 2-13, 6-17, 6-41
NAM statement, 3-2
NAM type 5, 9-2
named COMMON - see COMMON
noallocate option, 6-8, 6-9
noallocate option - see also ALLOCATE
node, 3-2
node fault, 2-20, 3-3, 4-31
node numbers, 3-3
node ordinal - see preorder
node tree, 3-2
 sample, 3-6
non-EMA labeled COMMON - see COMMON
non-main modules, 6-5
non-segmented programs, 4-5

O

offpath reference, 3-19, 3-20, 6-17, 6-20, 6-22, 6-25, 8-3, 8-12
OP (opcode), 5-7
 default, 5-7
order of commands, expected, 6-26
overhead,
 debugger, 6-6
 EMA, 6-6
 fault, 2-20
 links, 8-10
 memory, 7-9
 multilevel segmentation, 7-9
 profile option, 6-6, 7-9
 virtual memory, 8-8
 VMA, 6-6

P

page alignment, 3-8, 3-17, 5-5, 6-3, 6-6, 7-3, 7-8, 8-10
partition, 5-2
PARTITION REQUIRED, 4-29, 6-3, 7-1
Pascal, 3-22, 6-4, 8-2, 9-1, 9-9
passing procedures as parameters, 6-20
path, 3-2
path numbers, 2-20, 3-4
path overflow, 6-3, 6-20, 8-3, 8-6
path size, 6-2, 6-3
pathlength and tree size, maximum, 3-16

- permanent loads, 4-8
- permanent program, 1-2, 4-30
- PF (profile), 5-7
- potential offpath references - see offpath reference
- preliminary command file, 3-5, 6-1, 8-4
- preorder, 3-3, 4-18, 6-17, 6-28
- profile option, 1-3, 2-20, 4-30, 7-9
- profile output, 4-31
- program addition, commands, 2-7
- program loading,
 - adding permanent, 2-9
 - deleting permanent, 2-10
 - permanent, 2-8
 - replacing permanent, 2-9
 - requirements and restrictions, 2-10
 - temporary, 2-8
- program relocation and linking, 1-1, 2-1
- program size, 2-15
- program termination, 2-20
- program types, 2-13, 4-30
 - Background, 2-15
 - Extended Background, 2-15, 3-4
 - Large Background, 2-15
 - Real-Time, 2-15
- programmatic scheduling, return parameters for, 4-31

R

- RE (relocate), 3-9, 5-7
- Real-Time program, 2-15
- recursion and offpath references, 3-19, 3-20, 6-20
- reducing links, 8-10
- reducing load time, 6-18, 8-12
- reducing path size, 8-8
- relocatable code, 1-1
- relocatable indexer, see INDXR
- relocate command, 5-1
- relocating loader LOADR, 9-1
- relocation base, 2-2, 3-4
- relocation base - see also load point
- relocation commands, 2-1
 - LI (user library file), 2-1
 - LO (set relocation base), 2-1
 - MS (multiple search), 2-1
 - RE (relocate), 2-1
 - SA (save area), 2-1
 - SE (search), 2-1
 - SL (search user library), 2-1
- replacing permanent programs, 2-7, 2-9
- required partition size, 2-15
- requirements for program loading, 2-11

- return parameters for programmatic scheduling, 4-31
- reverse COMMON - see COMMON
- RMPAR, 4-31
- root node, 3-2
- root node only, 7-2
- rotating base page, 4-24
- routine, 3-2
- RPL's, 6-11
- RT - see Real-Time
- runstring parameters, SGMTR, 6-1

S

- SA (save), 5-8
- sample calling sequence and node tree, 3-6
- sample command file, 3-11
- sample program,
 - loadmaps, 4-18, 4-21, 4-25
 - SGMTR output, D-5
 - source code, D-1
 - SXREF output, D-8
- sample trace of MLLDR processing, 3-13
- SAVE, 70-1
 - area, 4-19, 7-2
 - labeled COMMON, 6-10
 - loader command, 5-8
 - variable, 6-19
- saving resources termination, 2-20
- SE (search), 3-10, 5-1, 5-8
- search commands, 4-14, 9-5
 - MS (multiple search), 5-6
 - SL (search user libraries), 5-9
- search processing, 3-13, 4-14, 5-4, 5-6, 6-6, 6-8
- search processing (LOADR), 9-2
- SEGLD, 2-6, 9-1
- segmentation - see also multilevel segmentation
- segmentation utility SGMTR, 6-1
- segmentation,
 - and loading tools, 2-7, 6-1
 - in previous operating systems, 2-6
 - options, 6-6
 - restrictions, 3-16, 6-19
- segmenter utility - see SGMTR
- serially reusable termination, 2-20
- SGMTR - see also loading large programs
- SGMTR, 3-5, 6-1, 8-1, 8-4
 - aborting, 6-7
 - error messages, A-12
 - errors, summary, 6-21
 - first run of SGMTR, 8-4
 - function, 6-1

- indexed and merged files, 6-11
- merged and indexed files, 6-11
- offpath reference message, 6-22
- operation, 6-1
- output, 6-12
- passing procedures as parameters, 6-20
- preparing the input file, 6-7
- reducing path size, 8-8
- requirements for COMMON, 6-7
- RPL's, 6-11
- sample output, D-5
- scratch file, 6-7
- segmentation restrictions, 6-19
- segmenting a subtree, 6-5
- shortening the command file, 6-18
- symbol table overflow, 8-3
- SGMTR, runstring parameters, 6-1
 - input, 6-1, 6-2
 - load options, 6-2, 6-6
 - main entry point, 6-2, 6-4
 - output, 6-1, 6-2
 - path size, 6-2, 6-3
 - segmentation options, 6-2, 6-6
- SH (shareable EMA), 5-9
- shareable EMA, 1-3, 2-17, 4-30
 - label file, 2-17
 - partition, 2-17
- shared base page, 4-24
- shared logical address space, 3-4
- shortening the command file, 6-18
- single-level segmentation, 1-2, 9-1
- SL (search user libraries), 3-10, 5-9
- son node, 3-2
- subroutines, 2-15, 3-2
- SXREF, 6-23
 - aborting, 6-24
 - cross reference mode, 6-27
 - error messages, A-17
 - limitations, 6-25
 - listing, 6-28
 - operation, 6-23
 - output file, 6-24
 - runstring parameters, 6-23
 - sample output, 6-30, D-8
 - scratch files, 6-24
 - summary of errors, 6-39
 - syntax checking mode, 6-23, 6-25
- SY (system library name), 3-9, 5-10
- syntax checking mode (SXREF), 6-23, 6-25
- system COMMON - see COMMON
- SZ (size), 5-10

T

T5IDM, 9-2
temporary program, 1-2, 4-30
termination, command input, 5-2, 5-3
termination, program, 2-20
TR (transfer), 5-10
transfer address, 6-28
transfer of control, subroutines, 3-1
tree size and pathlength, maximum, 3-16
tree structure, sample, 3-7
type 2 program, 2-15
type 3 program, 2-15
type 4 program, 2-15
type 5 file, 1-1
type 5 module, 9-2
type 6 file, 2-8, 9-2
type 6 program, 2-15

U

undefined external reference, 3-15, 4-13, 4-14, 5-2, 6-17, 6-29
unlabeled COMMON - see COMMON
utilities, 6-1

V

virtual memory (VMA), 1-3, 2-18, 4-29, 5-11, 7-8, 8-8
 commands, 2-18
 default environment, 2-18
 size, 2-15
VMA - see virtual memory
VS (VMA size), 5-11

W

waiting for disc space, 4-17
waiting for list device, 4-17
waiting for resources, 4-17
warning and information reporting, 4-16
warning W-RQ PGS, 7-8
warnings (LOADR and MLLDR), A-10
 duplicate program name, 4-17
 reporting, 4-16
 undefined externals, 4-13
working set, 2-18, 4-29, 7-8
write-protect switch, 2-12
WS (working set size), 5-11

