IBM System/3 Overlay Linkage Editor Reference Manual

Program Numbers:

5702-SC1	(Models 8 and 10)
5703-SC1	(Model 6)
5704-SC1	(Model 15)
5704-SC2	(Model 15)
5705-SC1	(Model 12)

GC21-7561-5
File No. S3-31

Preface

The Overlay Linkage Editor is a part of the IBM System/3 Model 15 System Control Program (Program Number 5704-SC1), and is a separately orderable feature of the IBM System/3 Model 10 Disk System (Feature 6026/6027), IBM System/3 Model 12 (Feature 6026/6027), and the IBM System/3 Model 6 (Feature 6011/6012). This manual provides reference information for programmers using the Overlay Linkage Editor capabilities of System/3. This manual is intended for experienced programmers who plan to link-edit their own object modules rather than have the language translators (assemblers and compilers) do the link-editing.

Note: In this publication there are some references to support of 64K bytes of main storage. A System/3 Model 10 with a 64K processing unit is available only as an RPQ. Your IBM Marketing Representative can provide information about this.

System/3 Model 8

The System/3 Model 8 is supported by System/3 Model 10 Disk System control programming and program products. The facilities described in this publication for the Model 10 are also applicable to the Model 8, although the Model 8 is not referenced. It should be noted that not all devices and features which are available on the Model 10 are available on the Model 8. Therefore, Model 8 users should be familiar with the contents of *IBM System/3 Model 8 Introduction,* GC21-5114.

Sixth Edition (September 1978)

This is a major revision of, and obsoletes GC21-7561-4 and technical newsletters GN21-5406 and GN21-5585. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

This edition applies to the System/3 program versions listed below and to all subsequent versions and modifications until otherwise indicated in new editions or technical newsletters.

Version	Modifications	Program Number	Feature Number	Model
15	0	5702-SC1	6026, 6027	8, 10
15	0	5703-SC1	6010, 6011	4, 6
6	0	5704-SC1	6033, 6070, 6071	15A, 15B, 15C
3	0	5704-SC2	6011,6012	15D
4	0	5705-SC1	6070, 6071	12

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, refer to the latest *IBM System/3 Bibliography,* GC20-8080, for the editions that are applicable and current.

Use this publication only for the purposes stated in the *Preface.*

Publications are not stocked at the address below. Requestes for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Address your comments about this publication to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Contents

Introduction

Linkage editor processing is necessary following the assembly or compilation of any program. The output of a language translator (assembler or compiler) is called an object module (see Figure 1). An object module cannot be run as a program until it is link-edited into a load module. Object modules and load modules can reside on cards or in the object library on disk (see Figure 2).

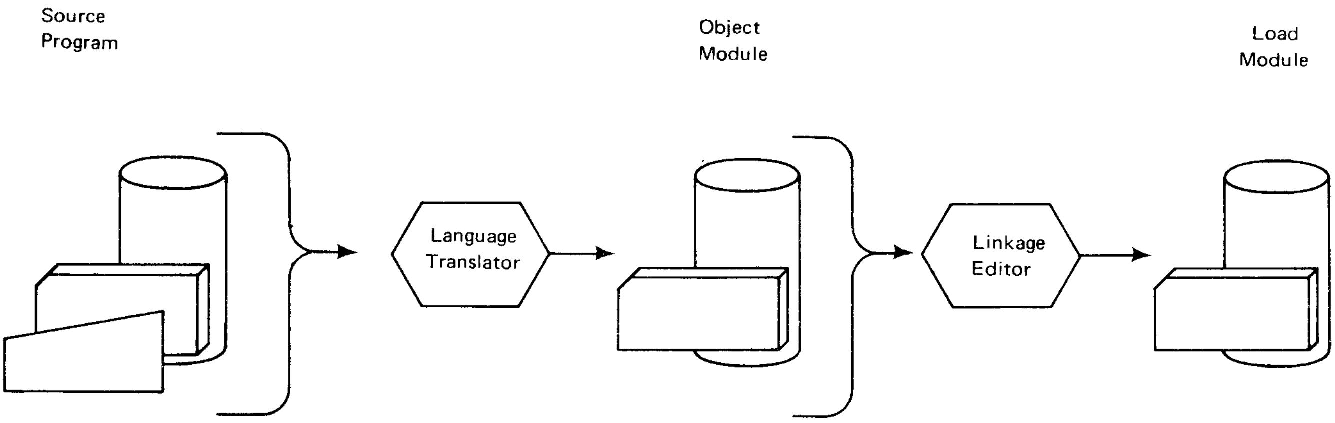
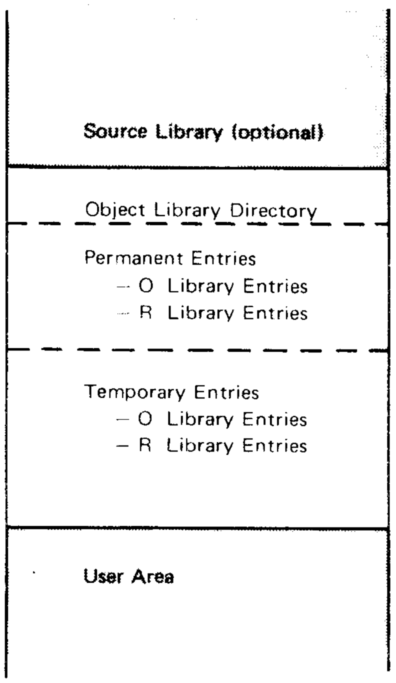


Figure 1. Preparing a Source Program for Execution



The O library entries are load modules. They are loaded by the LOAD OCL statement.

The R library entries are object modules that must be link-edited into a load module before they can be loaded.

Figure 2. Format of Object Library

The Overlay Linkage Editor provides a compiler entry and a user entry. The compiler entry provides the following functions:

- Punches into cards and/or catalogs into the object library on a disk the output object module of a language translator such as IBM System/3 FORTRAN, COBOL, Basic Assembler, and Model 15 RPG II.
- Link-edits the output of language translators such as IBM System/3 FORTRAN, COBOL and Model 15 RPG II into a load module and punches the load module into cards and/or catalogs it into an object library on disk. The assigning of modules to overlay segments is determined automatically by the Overlay Linkage Editor.

The user entry allows the user to link-edit IBM System/3 Basic Assembler object modules and object modules built by other language translators into load modules. The user can influence the determination of overlays himself or he can allow the Overlay Linkage Editor to determine the overlay structure. The load modules can be punched into cards and/or cataloged into an object library on disk.

SYSTEM CONFIGURATION

For information concerning the minimum system configuration for Overlay Linkage Editor and additional devices supported. see one of the following publications, as appropriate for your System/3 model:

- *IBM System/3 Models 4, 6, 8, 10, and 12 System Generation Reference Manual,* GC21-5126
- *IBM System/3 Model 15 System Generation Reference Manual,* GC21-7616
- *IBM System/3 Model 4 Introduction,* GC21-5146
- *IBM System/3 Model 6 Introduction,* GA21-9122
- *IBM System/3 Model 8 Introduction,* GC21-5114
- *IBM System/3 Model 12 Introduction,* GC21-5116
- *IBM System/3 Model 15 Introduction,* GC21-5094

PRIMARY STORAGE REQUIREMENTS

The primary storage requirements for the execution of the Overlay Linkage Editor are as follows:

System/3 Model	*Main Storage*
Models 4 and 6	7K
Models 8 and 10	7K
Model 12	8K
Model 15	10K

SECONDARY STORAGE REQUIREMENTS

The Overlay Linkage Editor requires 10 tracks in the object library. For execution, work space must be available as follows:

			3340	
	5444	5445	5444 Simulation Area	Main Data Area
Models 4 and 6	X			
Models 8 and 10	X			
Model 12			X	
Model 15	X	X	X	X

This space can be specified by the user or allocated by the Overlay Linkage Editor (see index entry: *OCL statements*).

ERROR HALTS

Halts are issued with system halt messages on SYSLOG for error conditions. If the log is off, a second level halt is issued to fully define the error condition (Models 6, 10 Disk System, and 12).

CHANGES IN LOAD MODULE SIZE

Changes made to the Overlay Linkage Editor from release to release may cause change in the size of the output load module. For example, a program that fits in 4K on one release may not fit in 4K on the next release.

Using the Overlay Linkage Editor

This section describes the compiler and user entries to the Overlay Linkage Editor and the storage map printed by the Overlay Linkage Editor to inform the user of the structure of his program. The input object modules used by both entries of the Overlay Linkage Editor are described in Appendix B.

COMPILER ENTRY

The compiler entry to the Overlay Linkage Editor is used by language translators to punch and/or catalog their output object modules (object modules are described in Appendix B). Language translators, such as IBM System/3 FORTRAN, COBOL, and RPG II (Model 15 only), can also specify link-editing. The Overlay Linkage Editor then link-edits the object module into a load module and punches and/or catalogs the load module.

When the user compiles an object module and immediately link-edits it into a load module via the compiler entry, he can influence the determination of overlays only by specifying the category of the object modules on the compiler input. For the Overlay Linkage Editor method of determining overlay structure, see index entry: *determining overlay modules.*

USER ENTRY

To use the Overlay Linkage Editor, the user must supply:

- Operation Control Language (OCL) statements
- Overlay Linkage Editor Control statements
- Modules to be linked (described in Appendix B)

OCL Statements

The following OCL statements are examples of loading the Overlay Linkage Editor via the user entry:

Model 10 Disk System, Model 12, and Model 15

```
// LOAD $OLINK,unit (unit can be R1,F1,R2, or F2)

// FILE NAME-$SOURCE,  . . . \  (These two FILE state-
                             |  ments are optional and
// FILE NAME-$WORK,    . . . |  are standard FILE state-
                             /  ments used by the
// RUN                          compilers.)
```

Model 6

```
010 LOAD     NAME-     $OLINK
011          UNIT-     (R1,F1,R2, or F2)
020 FILE     NAME-     $SOURCE\
 •                             |
 •                             | (These two FILE
 •                             | statements are
030 FILE     NAME-     $WORK   | optional and are
 •                             | standard FILE
 •                             | statements used by
 •                             / the compilers.)
MODIFY

RUN
```

The Overlay Linkage Editor requires from 10 to 30 tracks of disk space. FILE statements should be supplied for large programs (25K or more), or if the partition size that the Overlay Linkage Editor is currently in, is 25K or more. If the two FILE statements are supplied, they must be the same as the standard FILE statements used by the compilers.

The Overlay Linkage Editor will find disk space if FILE statements are not supplied. Space will be assigned on F1 if there is a minimum of 10 tracks available (even though 10 tracks may not be sufficient for a large program). If the minimum space is not available on F1, space will be assigned on R1.

You may save time if you supply FILE statements to place the files optimally(see Appendix C, *Performance Improvements*); for the Model 15, files should be placed on a 5445 disk drive (or 3340 main data area) for best performance. FILE statements can also be supplied to ensure that the Overlay Linkage Editor has adequate work space to complete the link-edit.

The OCL statements can be entered from the system input device or called from the procedure library.

Control Statements

Overlay Linkage Editor control statements can be entered from the system input device or from the procedure library. The types of control statements are:

1. PHASE statement (optional).
2. OPTIONS statement (optional).
3. INCLUDE statements and/or object modules in card form (required). The first object module encountered (either in card form or indicated on an INCLUDE card) will be the mainline routine.
4. GROUP statement (optional).
5. CATEGORY statement (optional).
6. EQUATE statement (optional).
7. END statement (required).

Control Statement Summary

Use	Control Statements
Define Load Module*	// PHASE NAME–name,UNIT–code,PUNCH– $\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$,RETAIN– $\left\{\begin{matrix}\underline{\text{T}}\\ \text{P}\\ \text{R}\end{matrix}\right\}$,LINKADD– $\left\{\begin{matrix}\underline{\text{S}}\\ \text{S+X'aaaa'}\\ \text{X'aaaa'}\end{matrix}\right\}$,RLD– $\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$
Define Environment	// OPTIONS UPACK–code,CORE–annK,LEVEL–nnn,ENTRY–label, ATTR–xxx,MAP– $\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\\ \text{XREF}\\ \text{MSG}\end{matrix}\right\}$
Define Object Modules	// INCLUDE NAME–name (or 'name,name, . . . name'),UNIT–code
Group Object Modules Together in Storage	// GROUP NAME–name (or 'name,name, . . . name'),AREA–USER
Change Category of Object Module	// CATEGORY NAME–name (or 'name,name, . . . name'),VALUE–nnn
Equate Module Names	// EQUATE OLDNAME–name (or 'name,name, . . . name'),NEWNAME–name (or 'name,name, . . . name')
End of Control Statements	// END

*RLD– $\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}$ applies only to Model 15

Parameter Summary

The following is a discussion of the parameters for each of the control statements. When there is a default value for a parameter, the default value is underlined.

PHASE Statement

The PHASE statement specifies the name and destination of the load module. If the PHASE statement is omitted, the load module is assigned the same name as the mainline routine (see index entry: *INCLUDE statement)* and is cataloged as a temporary entry in the object library of the program pack.

// PHASE NAME-name,UNIT-code,PUNCH- $\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$,RETAIN- $\left\{\begin{matrix}\underline{\text{T}}\\ \text{P}\\ \text{R}\end{matrix}\right\}$,LINKADD- $\left\{\begin{matrix}\underline{\text{S}}\\ \text{S+X'aaaa'}\\ \text{X'aaaa'}\end{matrix}\right\}$,RLD- $\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$

NAME-name	The name that the load module has in the object library directory. If the NAME parameter is not supplied, the load module assumes the NAME of the mainline routine. The name can be from one to six characters long and can contain any combination of System/3 characters except blanks, commas, quotes, or periods. The first character must be alphabetic.
UNIT-code	Disk where the load module is placed. Possible codes are R1, F1, R2, and F2. If neither the UNIT parameter nor the PUNCH parameter is specified, the load module is put on the program pack.
PUNCH- $\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$	Specifies whether to punch the load module. If not supplied, the default is NO. The load module can be both punched into cards and put in the object library by specifying both UNIT and PUNCH parameters.
RETAIN- $\left\{\begin{matrix}\underline{\text{T}}\\ \text{P}\\ \text{R}\end{matrix}\right\}$	Specifies whether the load module is to be cataloged as a temporary or permanent entry in the object library directory. RETAIN-R means replace an existing entry with the same name. The RETAIN type of the new module is P. If no entry exists with the same name, the new entry is added with a permanent designation (P). If RETAIN is not specified, T is the default. If RETAIN-P or R is specified, all previous temporary modules are deleted from the library. If this parameter is specified with PUNCH-YES, the retain code is specified on the COPY card that is punched with the module.
LINKADD- $\left\{\begin{matrix}\underline{\text{S}}\\ \text{S+X'aaaa'}\\ \text{X'aaaa'}\end{matrix}\right\}$	Specifies the link-edit start address, which is the address assigned to the first byte of the link-edited load module. On the Model 15, if S is specified, the start address is X'4000'. An absolute address can be specified by coding X'aaaa', where aaaa is an absolute address. S+X'aaaa' is coded to specify the end of the supervisor plus an absolute address. If this parameter is not coded, S is assumed. If the start address plus the number of bytes in the program exceeds X'FFFF', the program is link-edited to start at X'0000'. (The System Control Program (Program Number 5704-SC2) is link-edited to the largest possible multiple of 2K.) This parameter does not affect the loading of the load module (see index entry: *link-edit start addresses*).
RLD- $\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$	Specifies whether a program will be produced with Text-Relocation Directory records (RLDs). If this parameter is not supplied, YES is the default. This parameter applies only to the Model 15.

OPTIONS Statement

The OPTIONS statement describes the load module and specifies the location of user object modules and the type of linkage editor output. If the entire OPTIONS statement or any of the parameters are omitted, the defaults given are used.

```
                                                                YES
// OPTIONS UPACK–code,CORE–annK,LEVEL–nnn,ENTRY–label,ATTR–xxx,MAP– NO
                                                                XREF
```

UPACK–code	Disk where user modules to be link-edited can be found. If UPACK is not specified, the linkage editor looks for the user modules on the pack that the Overlay Linkage Editor is on. This keyword is used when the Overlay Linkage Editor performs AUTOLINK.

CORE–annK	Storage size the load module has available for execution. If specified, the directory entry contains this size even though the actual size required by the load module is less. If not specified, the current partition size is used to determine when overlays are required and the directory entry contains the actual load module size. This parameter is needed only if the partition size at execution will be different from what it is at link-edit time.
	a = increments of 1/4 K
	Q = 1/4 or 256 bytes
	H = 1/2 or 512 bytes
	T = 3/4 or 768 bytes
	0 = zero bytes
	nn = 1K increments
	Example: Q04K = 1/4K + 4K = 256 + 4096 = 4352 bytes

LEVEL–nnn	Number that is placed in the level entry in the object directory entry. Different modification levels of load modules can be assigned different level values. The maximum value for nnn is 255. Default value is 001, except for load modules generated by the following Model 15D compilers: 5704-RG2, RPG II = 253[1] 5704-CB2, COBOL = 254[1] 5704-FO2, FORTRAN = 255[1]
ENTRY–label	An entry point or module NAME of an included module. Default is the entry point of the mainline routine.

[1] If a compiler requested R-module is link-edited using the level parameter of another compiler, erroneous time/date information will be printed in the module directory listing.

If a level value of 000 through 252 is used, the time/date information will not be printed in the module directory listing.

ATTR—xxx or 'xxx,xxx, . . . xxx'	Attributes of the module being link-edited. If ATTR is not specified, no attributes are assigned. xxx = INQ —Inquiry. This program can be run in either program level and dedicates the use of the Inquiry key (PA1 key on Model 15) to its program level (normally used to start processing). IEV —Inquiry Evoking. This program can run on a dedicated system or in level 1 of a DPF system. In a DPF system the Inquiry key (or the ROLLOUT command in the System Control Program [Program Number 5704-SC11) is dedicated to the IEV program in level 1. The Inquiry key is normally used to cause the IEV program to be rolled out to allow another program to run. Using the System Control Program (Program Number 5704-SC1), the IEV program can run in either partition, but can be rolled out only when running in partition 1. This is not supported by the System Control Program (Program Number 5704-SC2). DED—Dedicated. In a DPF system, this program must run with the other program level inactive. This is not supported by the System Control Program (Program Number 5704-SC2). SRQ—Source Required. This program requires the allocation of the $WORK and $SOURCE files. $SOURCE must be filled either from the system input device or a source library. Any program with the SRQ attribute will be loaded, and relocated, to the normal load point plus 10 bytes. DFM—Deferred Mount. This program accepts mounting of packs during its execution. SID —SYSIN Dedication. The system input device must be dedicated to this program. The device is released at end of job. CPR —Checkpoint Restart Program. DSR —Direct Source Read. This program can have a // COMPILE statement and a no-source-required attribute. SYSIN dedication can also be released by program and not have the source-required attribute. The program accesses the source library itself. MRO—Memory Resident Overlay REMAP Program (Model 15). When specified, the program executes the segments in the resident area itself. MOV—Memory Resident Overlay MOVE Program (Model 15). When specified, the program retains the segment in the resident area but executes the segment in the conventional overlay fetch area. *Notes:* 1. CPR and IEV are mutually exclusive on all models. 2. INQ and IEV are mutually exclusive on Models 6, 8, 10, and 12. 3. MRO, MOV, IEV, and CPR are mutually exclusive on Model 15. 4. A checkpoint/restart program cannot have external buffers (5704-SC2). If ATTR is not specified, no attributes are assigned.
MAP— $\left\{\begin{array}{l}\underline{\text{YES}}\\ \text{NO}\\ \text{XREF}\\ \text{MSG}\end{array}\right\}$	Type of printer output during link-edit: YES = A storage map and messages are printed. If MAP is not specified, YES is assumed. NO = No storage map or messages are printed. XREF = A storage map, cross-reference list, and messages are printed. MSG = Only messages are printed.

INCLUDE Statement

The INCLUDE statement specifies which object modules are to be included in the load module. Multiple module names may be submitted on one INCLUDE statement. The first object module named on an INCLUDE statement or read from the SYSIN device is the mainline routine. If a module name is not found on the UNIT specified, a halt 12 will result. By taking a zero option, the program will do a find on the next module name in the statement. Either an INCLUDE statement or an object module in card form must be supplied as input to the Overlay Linkage Editor.

// INCLUDE NAME-name, UNIT-code	
NAME–name or NAME–'name, name, . . . name'	Name(s) of the object module(s) to be included in this program.
UNIT–code	Disk unit where object module is located. If omitted, will default to the program pack.

GROUP Statement

The GROUP statement can be used to specify a number of object modules that the user wishes to group together in storage. The user may design his own overlay structure, based on his knowledge of the object modules being link-edited, to obtain more efficient loading of overlay segments. These modules are put into the same overlay segment or partly in an overlay segment and partly in the root segment.

The first module named in a GROUP statement should be referenced by a module that is not in the group (see index entry: *grouping modules*).

By specifying AREA–USER, you can also use the GROUP statement to assign co-resident overlay modules to the user overlay area (see index entry: *overlay area*), thereby possibly reducing main storage size.

The GROUP statement is optional; if it's not supplied, the Overlay Linkage Editor designs the overlay structure (see index entry: *overlays*).

// GROUP NAME–name, AREA–USER	
NAME–name or NAME–'name, name, . . . name'	The name of a module that should be assigned to the user area or the names of the object modules that must all be in storage at the same time.
AREA–USER	If the modules named in this statement are assigned to an overlay, they will be assigned to the user overlay area. If a list of names ('name,name, . . .') is used with AREA–USER, the named modules are grouped in the user overlay area. To force multiple modules to the user area without grouping them together, specify each module name on a separate GROUP statement.

CATEGORY Statement

The CATEGORY statement is used to temporarily change the category value (priority) of object modules. Because the priority of an object module influences the placement of the module into an overlay, the user can delegate the module to a different segment by changing the category value. The category value of the module is changed only for this link-edit.

CAUTION
A program failure may result from changing the priority of a system module ($$xxxx module) or a compiler object module. Compiler object modules are modules (usually $xxxxx) that are included in the load module built from the compiler-generated object text. Modules for the Binary Synchronous Communications Adapter ($$BSxx modules) must remain at category 0 and cannot be overlaid.

Modules containing DTFs can be overlaid only if the associated file is closed prior to returning to the caller.

// CATEGORY NAME–name,VALUE–nnn	
NAME–name or NAME–'name, name, . . name'	Name(s) of module(s) for which the category value (priority) is to be changed for this link-edit.
VALUE–nnn	The new category value: 0 = A module with this category value cannot be overlaid. It is always placed in the root segment. 1-7 = These category values are generally reserved for system modules. These modules can be overlaid if necessary to satisfy main storage size. Modules with these category values may only call modules with the same category value or category 0 modules. IBM system modules have the following general category values: 2 – Disk I/O 3 – Tape I/O 4 – Arithmetic 6 – Unit Record I/O Category values 1, 5, and 7 have no special meaning but are available for use. 8-124 = These category values assign overlay priorities. The lower the number, the less likely that the module will be overlaid. 125 = For the Model 15, if ATTR-MRO or ATTR-MOV has been specified, category value 125 allows the module to be overlaid but *does not* allow that overlay segment to be a candidate for memory residence. 126 = For the Model 15, this category value assigns a special overlay priority. In any overlay program, routines of category 126 will be given first consideration for re-inclusion in the root area (non-overlay core). Generally, category 126 routines should be RPG mainline routines. 127 = This value will be treated the same as a category value of 0 (zero). It will be displayed on the core map as category 0, not category 127, except when it is assigned by a CATEGORY statement. 128 = This value specifies that the module must be aligned on a 256-byte boundary. Value 128 can be used with any lower category value. This is done by adding the lower value to 128. For example, you can specify that a module have category value of 8 and be aligned on a 256-byte boundary by specifying a category value of 136 (8 + 128) on the CATEGORY statement. Category 128 indicates a category 0 module aligned on a 256-byte boundary. If an input module contains a category value of 128, the module wil be aligned on a 256-byte boundary even if a CATEGORY statement assigns a category value of less than 128. The boundary-align attribute cannot be changed by a CATEGORY statement, but the non-boundary-align (under 128) portion can be changed for this link-edit.

END Statement

An END statement indicates the end of the Overlay Linkage Editor input and must follow the control statements and/or object modules read from the SYSIN device.

```
     // END
```

EQUATE Statement

The EQUATE statement is used to make a temporary change to a reference to a module name or entry point. References to a module name or entry point specified in the OLDNAME parameter are resolved to the module name or entry point in the NEWNAME parameter. If a list of names is entered, the OLDNAME entries have a one-to-one relationship to the NEWNAME entries. The first OLDNAME is resolved to the first NEWNAME, the second to the second, etc.

Each list must contain the same number of names. If a name is used as an OLDNAME more than once, it is resolved to the first NEWNAME it matches. Only one level of equating is done. Consider the following statements.

```
// EQUATE OLDNAME-ABLE,NEWNAME-BAKER
// EQUATE OLDNAME-BAKER,NEWNAME-SAM
```

These statements would cause references to ABLE to be resolved to BAKER and references to BAKER to be resolved to SAM. References to ABLE would not be resolved to SAM.

If two modules are equated and their entry points are also referenced, the entry points also must be equated.

```
// EQUATE OLDNAME- { name                      } ,NEWNAME- { name                      }
                   { 'name,name, . . . name'   }           { 'name,name, . . . name'   }
```

OLDNAME-name or OLDNAME-'name, . . . name'	The module name or entry point now referenced in the program.
NEWNAME-name or NEWNAME-'name, . . . name'	The module name or entry point that will replace the referenced name or entry point in the program.

STORAGE MAP

A storage map is printed unless MAP–NO is specified on the OPTIONS statement. The system date is printed following the title line. The headings on the map are: Start Address, Overlay Number and Overlay Area, Category, Name and Entry (for module name and entry points), Code Length Hexadecimal, Code Length Decimal, and Referenced By (only if a cross-reference list included). The Overlay Area heading lists the area each overlay is loaded into: U for user area, S for system area, and C for co-resident area.

If the category of a module is changed, both the old and new category values are printed. The format is: old, new.

If a module is included in two or more overlays, it appears on the map in two or more places. If MAP–XREF is specified on the OPTIONS statement, a cross-reference list is also printed. This list contains modules that have external reference ESLs to the module names or entry points.

At the end of the storage map, the total storage used is given in decimal, and the start control address is given in hexadecimal. If the program uses overlays, the non-overlay storage size is also printed. The storage size of an overlay program is always in increments of 256 bytes. The non-overlay storage size is the exact number of bytes in the load module.

The storage map can be omitted to save link-edit time (see Appendix C, *Performance Improvements*).

OVERLAY AREAS

Main storage for an object program with overlays may be divided into four areas; root, user, system, and co-resident. (see Figure 3) Not all programs will need all four areas. The storage map indicates which overlay area each overlay segment is loaded into and the start address of each overlay area. See the storage maps printed with the examples.

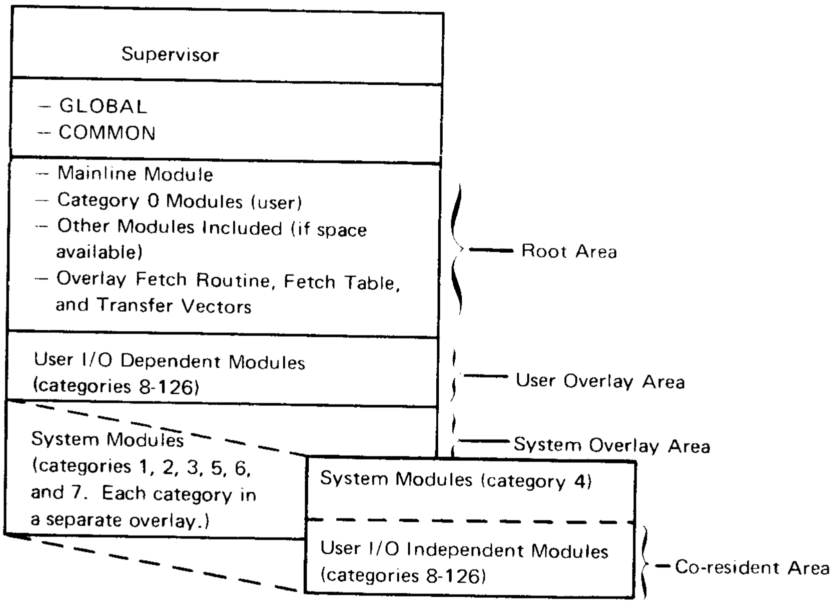

Supervisor
– GLOBAL
– COMMON
– Mainline Module
– Category 0 Modules (user)
– Other Modules Included (if space available)
– Overlay Fetch Routine, Fetch Table, and Transfer Vectors
Root Area
User I/O Dependent Modules (categories 8-126)
User Overlay Area
System Modules (categories 1, 2, 3, 5, 6, and 7. Each category in a separate overlay.)
System Overlay Area
System Modules (category 4)
User I/O Independent Modules (categories 8-126)
Co-resident Area

Figure 3. Overlay Areas (Non-DPF System)

Root Area

The root area of an overlay program contains the parts of the program that are never overlaid (see Figure 3). The root area always contains the mainline module, overlay fetch routine, fetch table, and transfer vectors. The remaining parts of the root depend on the program being linked.

User Overlay Area

The user overlay area contains user modules that call system I/O modules. Each overlay segment loaded into the user overlay area can contain modules of different category values.

If the COBOL segmentation feature has been used, the COBOL segments appear as overlays in the user overlay area. The presence of COBOL segments forces any non-COBOL modules that normally would have been assigned to the user area to the root area (category 0).

System Overlay Area

A system overlay segment contains system modules with the same category value. Each system overlay segment is independent of other system overlay segments. System modules are assigned to overlay segments solely by category value. A system module can only call another module with the same category or a category 0 module.

Co-resident Area

The co-resident area is actually a part of the system overlay area (see Figure 3). The system arithmetic overlay segment (category 4) is sometimes smaller than the system overlay area. If it is smaller, the remaining space is the co-resident area and can be used to load user modules that are I/O-independent (do not call system I/O modules). If the core requirements of category 4 plus the co-resident area is greater than the size of the system overlay area, category 4 modules will be re-included in the root area until all category 4 routines are in non-overlay core, or until the category 4 plus co-resident area will fit into the system overlay area. A module can be moved from this area to the user area by grouping it with an I/O-dependent user module or by specifying AREA-USER on the GROUP statement.

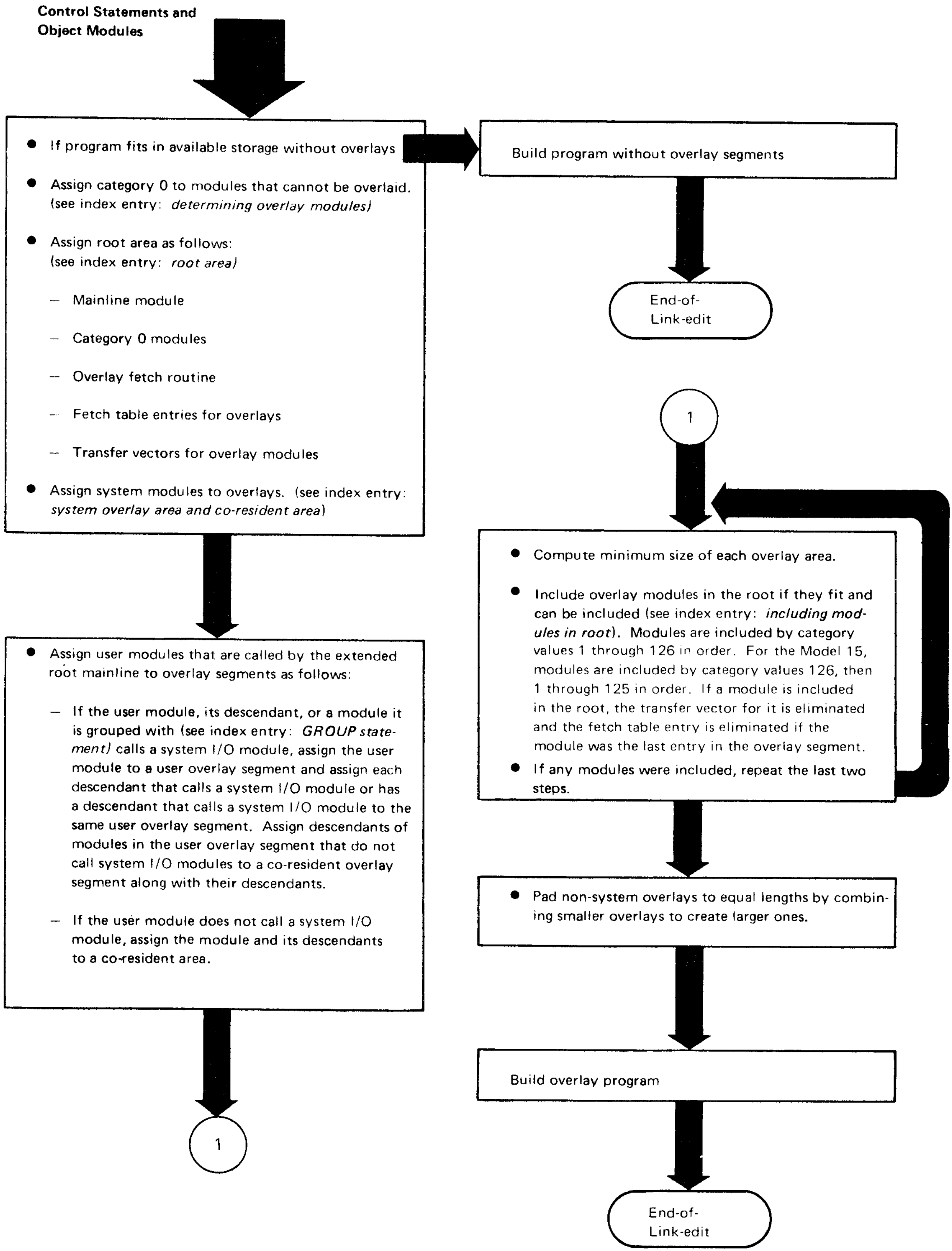
On the Model 15, routines of category 126 will be given first consideration for re-inclusion in the root area (non-overlay core). Generally, category 126 routines should be RPG mainline routines.

ASSIGNING OVERLAYS

The Overlay Linkage Editor attempts to fit all modules of an object program into the specified storage size without overlays. If this cannot be done, the Overlay Linkage Editor assigns some modules to overlay segments. Figure 4 shows the Overlay Linkage Editor method of assigning modules to overlay segments. The maximum number of overlay segments in a program is 254. The first module encountered (either on an INCLUDE statement or as an object module read from the input device) is the mainline routine and thus part of the root. The extended root mainline includes the mainline and all its descendants with each string of descendants being terminated when a non-category 0 module is encountered. A descendant is a module called by another module. The root is in main storage at all times and is never overlaid. The amount of main storage available determines the amount of code placed into overlay segments. If the load module does not fit in the main storage size specified and generating overlay segments would not enable it to fit better in storage, overlay segments are not generated.

Through the user entry you can use the GROUP statement to specify module groupings (see index entry: *grouping modules*) and use the CATEGORY statement to change the category of a module. You originally established the category of a module by specifying options to the compiler or assembler.

The Overlay Linkage Editor generates an overlay fetch routine, fetch table, and transfer vectors for programs with overlay segments and includes them in the root segment. The generated code is 116 bytes (127 bytes if ATTR-MRO is specified) plus 7 bytes for each overlay segment and 11 bytes for each overlay segment entry point that has a transfer vector. During execution of the object program, the overlay fetch routine is called when an overlay segment is needed. The overlay fetch routine checks to see if the segment is already in main storage. If it is, the segment is not reloaded. This saves the time needed to load the segment.



- If program fits in available storage without overlays
- Assign category 0 to modules that cannot be overlaid. (see index entry: *determining overlay modules)*
- Assign root area as follows: (see index entry: *root area)*
  - Mainline module
  - Category 0 modules
  - Overlay fetch routine
  - Fetch table entries for overlays
  - Transfer vectors for overlay modules
- Assign system modules to overlays. (see index entry: *system overlay area and co-resident area*)

Build program without overlay segments

- Assign user modules that are called by the extended root mainline to overlay segments as follows:
  - If the user module, its descendant, or a module it is grouped with (see index entry: *GROUP statement)* calls a system I/O module, assign the user module to a user overlay segment and assign each descendant that calls a system I/O module or has a descendant that calls a system I/O module to the same user overlay segment. Assign descendants of modules in the user overlay segment that do not call system I/O modules to a co-resident overlay segment along with their descendants.
  - If the user module does not call a system I/O module, assign the module and its descendants to a co-resident area.

- Compute minimum size of each overlay area.
- Include overlay modules in the root if they fit and can be included (see index entry: *including modules in root*). Modules are included by category values 1 through 126 in order. For the Model 15, modules are included by category values 126, then 1 through 125 in order. If a module is included in the root, the transfer vector for it is eliminated and the fetch table entry is eliminated if the module was the last entry in the overlay segment.
- If any modules were included, repeat the last two steps.

- Pad non-system overlays to equal lengths by combining smaller overlays to create larger ones.

Build overlay program

Figure 4. Overview of the Overlay Linkage Editor Method of Assigning Overlays

Determining Which Modules Can Be Overlaid

The Overlay Linkage Editor considers a module capable of being overlaid if the category of the module is non-zero and if the module is a direct descendant of (called by) the mainline routine (the first module named on the INCLUDE statement or read from the system input device) or descends from the mainline routine through only category 0 modules. A, C, G, and H in Figure 5 meet these requirements and can be overlaid.

A module that calls a module of an extended mainline routine (B and E are examples of extended mainline in the shaded portion of Figure 5) can be overlaid only if the module called has no direct or indirect call to an overlayable module. C in Figure 5 is overlayable since it calls E, and E does not call an overlayable module. If E called an overlayable module, C would have to be included in root.

A module called by an overlay module can itself be overlaid (module F in Figure 5).

Modules that do not qualify for overlay segments are assigned to the root segment. Module C in Figure 6 is assigned to the root segment because it appears twice in the program. Modules C and F in Figure 7 are assigned to overlay segments because each appears only once in the program, even though they do not meet the normal criteria for overlay modules.

Link-Edit Start Addresses

If LINKADD is not coded on the PHASE statement, the program is link-edited to start at the end of the supervisor (or at a fixed address for the Model 15).

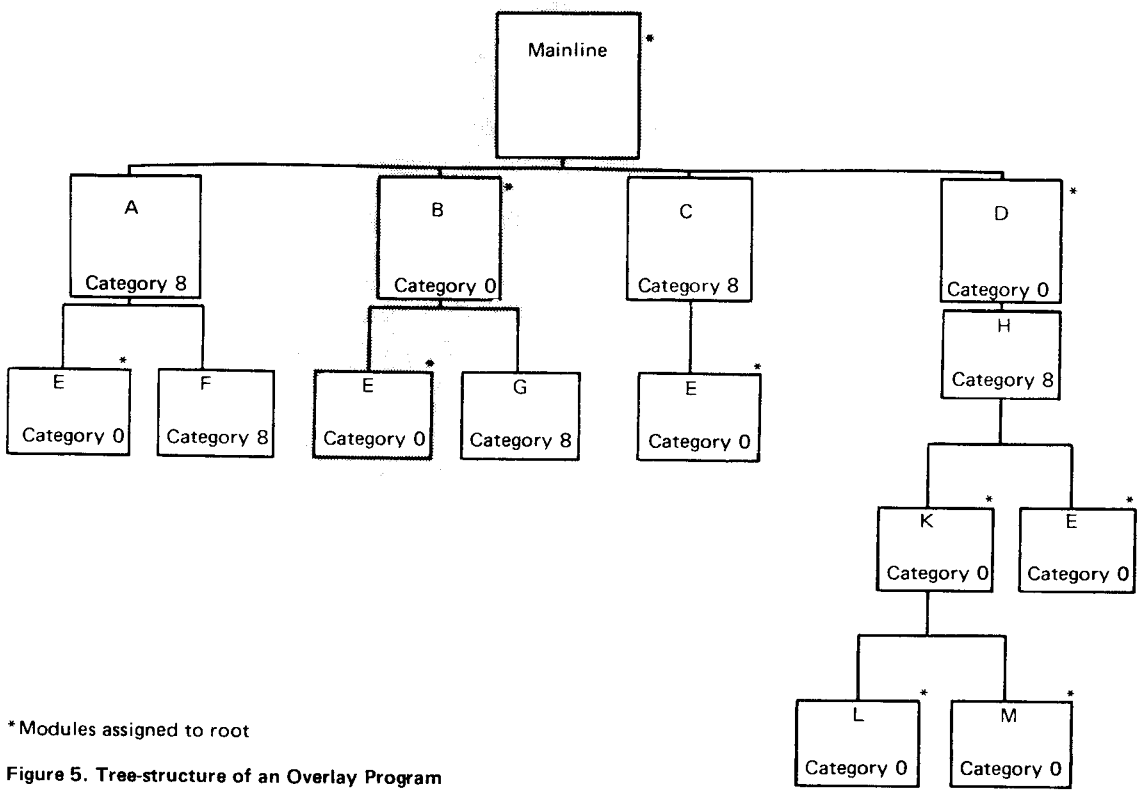
If the start address plus the length of the program exceeds 64K, the program is link-edited to start at X‘0000’. Using the System Control Program (Program Number 5704-SC2), the object program is linked-edited to the largest multiple of 2K, which allows the end address to be less than 64K.

The link-edit start address does not affect the load address. A program run on other than a Model 10 or Model 12 DPF system is always loaded at the end of the supervisor, no matter what address it is link-edited to. A PARTITION statement can be used in the OCL jobstream on a Model 10 or Model 12 DPF system to define the load address for programs run in level 2 (see the PARTITION statement in the *IBM System/3 Model 10 Disk System Control Programming Reference Manual,* GC21-7512, or in the *IBM System/3 Model 12 System Control Programming Reference Manual,* GC21-5130). If a PARTITION statement is not used, a program run in level 2 is loaded at the end of main storage.

Severe throughput degradation results if relocation is necessary for overlay programs. They should be link-edited to start at the load address. To determine the load address for overlay programs which are to run in level 2 of a Model 10 or Model 12 DPF system when a PARTITION statement is not used, subtract the program size from the main storage size. For example, you can calculate the link-edit start address of a 9728-byte program on a 24K system as follows:

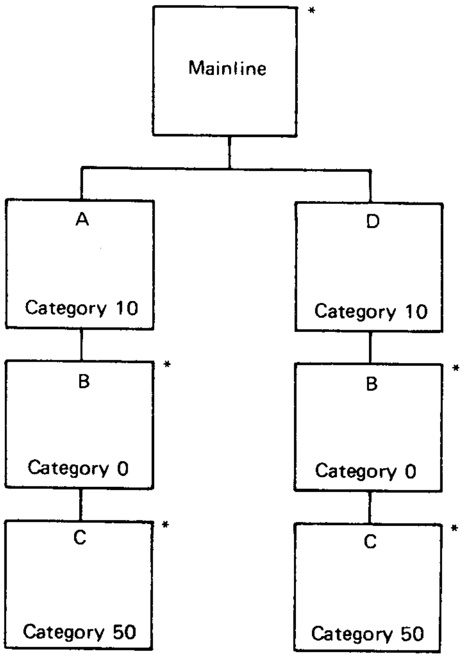
6000	–	24K converted to hexadecimal.
-2600	–	Program size in hexadecimal. (All overlay program sizes are stated in sector (256-byte) increments on the Overlay Linkage Editor storage map. Non-overlay programs must be rounded up to the next even sector.)
3A00		

The sample program just mentioned should be link-edited to start at X‘3A00’ by specifying LINKADD–X‘3A00’ on the PHASE statement.



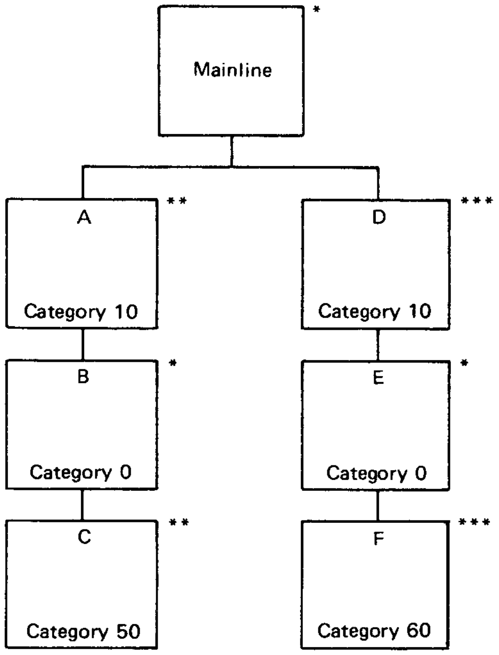
*Modules assigned to root

Figure 5. Tree-structure of an Overlay Program



*Modules assigned to root

Figure 6. User Modules Assigned to the Root Because they Cannot be Overlaid



*Modules assigned to root
**Modules in first user overlay
***Modules in second user overlay

Figure 7. Normal Root Modules Assigned to Overlays

Load Module Entry Point

The entry point of a load module can be changed by using the ENTRY parameter on the OPTIONS statement. The entry point can be changed to an overlay segment. If this is done, the actual entry point will be to the overlay fetch routine to load the overlay segment. The entry point of load modules that reference common areas must be the first byte of the module.

Overlay Area Size

The Overlay Linkage Editor assigns the smallest overlay areas possible. The user can increase the size of the overlay areas, and thereby possibly decrease the number of overlays, by using the GROUP statement to group modules into one large overlay. The Overlay Linkage Editor then automatically increases the size of the other overlays to take advantage of the increased area. This reduces the number of overlays.

Including Overlay Modules in the Root

After the Overlay Linkage Editor has assigned all modules to either the root segment or to overlay segments, any overlay modules that can be included in the root segment without exceeding the user specified main storage size are included. A module can be included if it meets one of the following criteria:

- The module calls no other module.
- The module is a user module and calls another user module but the called user module appears in only one overlay segment.
- The module is a system module called from a user module and all other system modules with the same category, not called by user modules, have already been included in the root segment.

Using the GROUP Statement

The GROUP statement is entered via the user entry to specify module grouping and/or overlay area assignment. The sequence of module names within the GROUP statement is important. The module of a group that is referenced from outside the group should be the first module named on the GROUP statement.

Figure 8 shows the modules referenced in the following GROUP statements. To group modules A, B, and C in one overlay and D, E, and C in another overlay, the correct GROUP statements are:

```
// GROUP NAME–'A,B,C'
// GROUP NAME–'D,E,C'
```

Modules A, B, C, D, and E would be assigned to only one overlay if the sequence of module names in the GROUP statements were as follows:

```
// GROUP NAME–'C,A,B'
// GROUP NAME–'C,E,D'
```

The GROUP statement can also be used to assign overlays to the user area. To assign groups AB and DE to the user overlay area, use the following GROUP statements:

```
// GROUP NAME–'A,B',AREA–USER
// GROUP NAME–'D,E',AREA–USER
```

Module C would be assigned to the co-resident area. This method reduces the size of the user area, saves secondary storage (module C appears only once), and may speed up execution of the program (module C must be loaded only once).

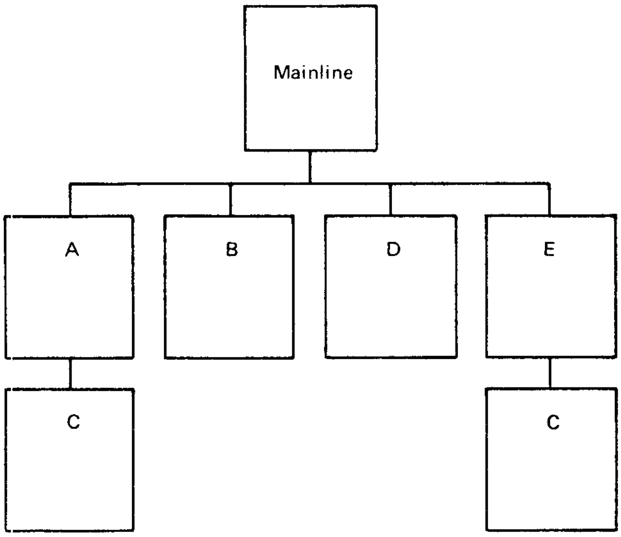


Figure 8. Tree-structure of Sample Program

MEMORY RESIDENT OVERLAYS (MODEL 15 ONLY)

Memory resident overlays (see Figure 9) is a technique designed to increase the performance of large overlay programs by allowing certain overlay segments to remain in primary storage after the initial segment fetch. The two types of memory resident overlay programs are MOVE and REMAP, which differ as follows:

- When ATTR-MOV is specified in the OPTIONS statement (MOVE technique), the program retains the segment in the resident area but executes the segment in the conventional overlay fetch area (see Figure 10).
- When ATTR-MRO is specified in the OPTIONS statement (REMAP technique), the program executes the segments in the resident area itself (see Figure 11).

To use the memory resident overlays feature, the feature must be selected as an option during system generation. However, object programs may be link-edited with this attribute on any system.

The overlay fetch routine generated for the MOVE technique is identical to the fetch routine generated for conventional overlay programs.

The CATEGORY statement controls which overlay segments are candidates for memory resident overlays. Any overlay segment containing a category 125 module is *not* a candidate for memory residence.

With the two memory resident overlay techniques (MOVE and REMAP), programs larger than 48K can reside in primary storage throughout execution if the partition is large enough and if the program can be link-edited within 48K. These techniques may improve performance for overlay programs that require a large number of overlay fetches.

The REMAP technique requires that the overlay segments be link-edited to 2K boundaries. These overlays are loaded on 2K boundaries at execution time. The MOVE technique does not have this restriction. For large overlay segments, REMAP will generally execute faster than MOVE.

Throughput degradation for memory resident overlays programs with RLDs will not be as severe as for conventional overlay programs because each resident overlay segment will be relocated only the first time it is fetched from disk.

Supervisor Support for Memory Resident Overlays

Supervisor support for memory resident overlays provides exactly the same function as the supervisor loader but with the new resident overlay techniques. It exists as a separate module ($@RLOD) and uses the entire partition to retain overlay segments after the initial segment fetch (for example, partitions greater than 48K).

To take advantage of the supervisor support for memory resident overlays, the module must be link-edited with one of the OPTIONS parameters (ATTR-MOV or ATTR-MRO) or compiled using the ATTR parameter of the OCL COMPILE statement.

An object program that is not compiled with one of the memory resident overlays options can be executed on a system that supports memory resident overlays. Similarly, an object program that is compiled with one of the memory resident overlays options can be executed on a system that does not support memory resident overlays.

If a program is link-edited with a memory resident overlays option but it does not require overlays, the option is not used and no diagnostics or halts occur.

Using Memory Resident Overlay with Communications Control Programming

A program running under the communications control program (CCP) can take advantage of the memory resident overlay techniques if you specify additional storage for the program at CCP start-up. The additional storage is specified on the PROGRAM statement of the CCP assignment set. For more information, see the *IBM System/3 Model 15 Communications Control Program System Reference Manual,* GC21-7620.

Memory Resideht Overlay Program Execution

Each time an overlay fetch is necessary, the supervisor loader checks to see if there is room for the segment in the resident area that begins just past the end of the last overlay area. If enough room exists, the segment is loaded into the resident area and the corresponding overlay fetch table entry is modified to show the physical location of the segment.

Those segments not having enough room in the resident overlay area will remain on disk to be fetched into the overlay fetch area as required.

For a general schematic of a memory resident overlays program see Figure 9.

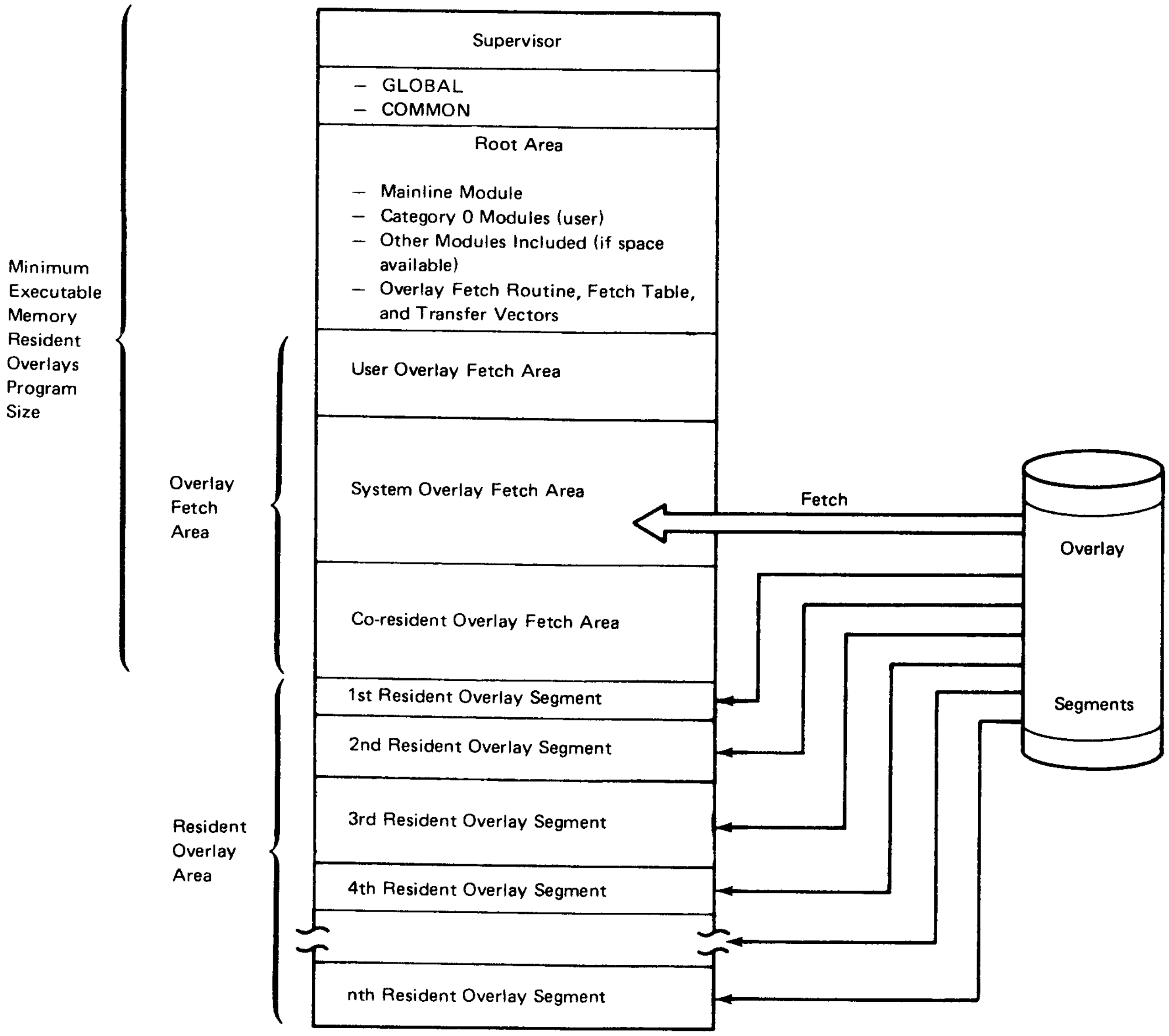

Supervisor
– GLOBAL
– COMMON
Root Area
– Mainline Module
– Category 0 Modules (user)
– Other Modules Included (if space available)
– Overlay Fetch Routine, Fetch Table, and Transfer Vectors
User Overlay Fetch Area
System Overlay Fetch Area
Co-resident Overlay Fetch Area
1st Resident Overlay Segment
2nd Resident Overlay Segment
3rd Resident Overlay Segment
4th Resident Overlay Segment
nth Resident Overlay Segment
Minimum Executable Memory Resident Overlays Program Size
Overlay Fetch Area
Resident Overlay Area
Fetch
Overlay
Segments

Figure 9. Schematic of the Memory Resident Overlays Program

Execution of the MOVE Technique

When ATTR-MOV is specified in the OPTIONS statement, the segment is retained in the resident area but executed in the overlay fetch area. Figure 10 shows a move of the 3rd resident overlay segment into the system overlay fetch area. This move replaces the disk I/O previously needed.

MOVE technique considerations are:

1. The ATTR-MOV memory resident overlays program link-edits into the same amount of storage as a conventional overlay program, but the overlays do not have to begin on 2K boundaries (see *Example 12*).

2. The MOVE technique requires more processing unit cycles than the REMAP technique because data movement is more time-consuming than adjusting address translate registers.

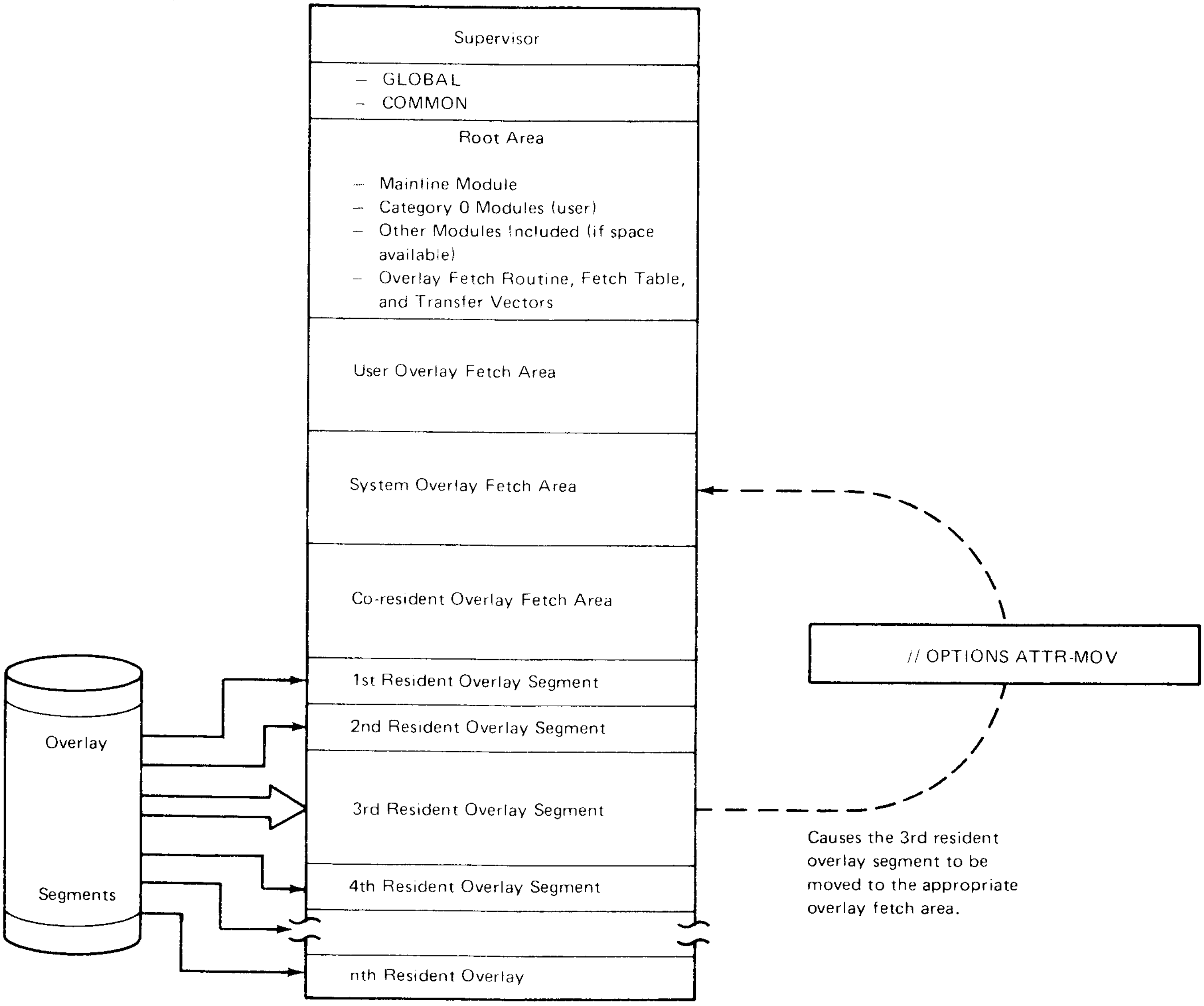

Supervisor
– GLOBAL
– COMMON
Root Area
– Mainline Module
– Category 0 Modules (user)
– Other Modules Included (if space available)
– Overlay Fetch Routine, Fetch Table, and Transfer Vectors
User Overlay Fetch Area
System Overlay Fetch Area
Co-resident Overlay Fetch Area
1st Resident Overlay Segment
2nd Resident Overlay Segment
3rd Resident Overlay Segment
4th Resident Overlay Segment
nth Resident Overlay
Overlay
Segments
// OPTIONS ATTR-MOV
Causes the 3rd resident overlay segment to be moved to the appropriate overlay fetch area.

Figure 10. Schematic Execution of the Memory Resident Overlay MOVE Technique

Execution of the REMAP Technique

Each time a resident overlay segment fetch is requested by the overlay fetch routine, the appropriate address translate registers (ATRs) are adjusted to logically include the segment in translated storage. No disk I/O is required. Figure 11 illustrates translated storage during execution.

REMAP technique considerations are:

1. Segments must start on 2K boundaries.

2. A program may require more execution storage than in the past (must link-edit into 48K).

3. A REMAP program can have fewer overlay segments than a MOVE program and may require more storage to execute in.

4. The link-edit start address must be a multiple of 2K if the overlay program has RLDs; otherwise, the link-edit start address will be rounded down to a 2K boundary.

5. A FORTRAN program using the INVOKE feature cannot be used with a memory-resident overlay.

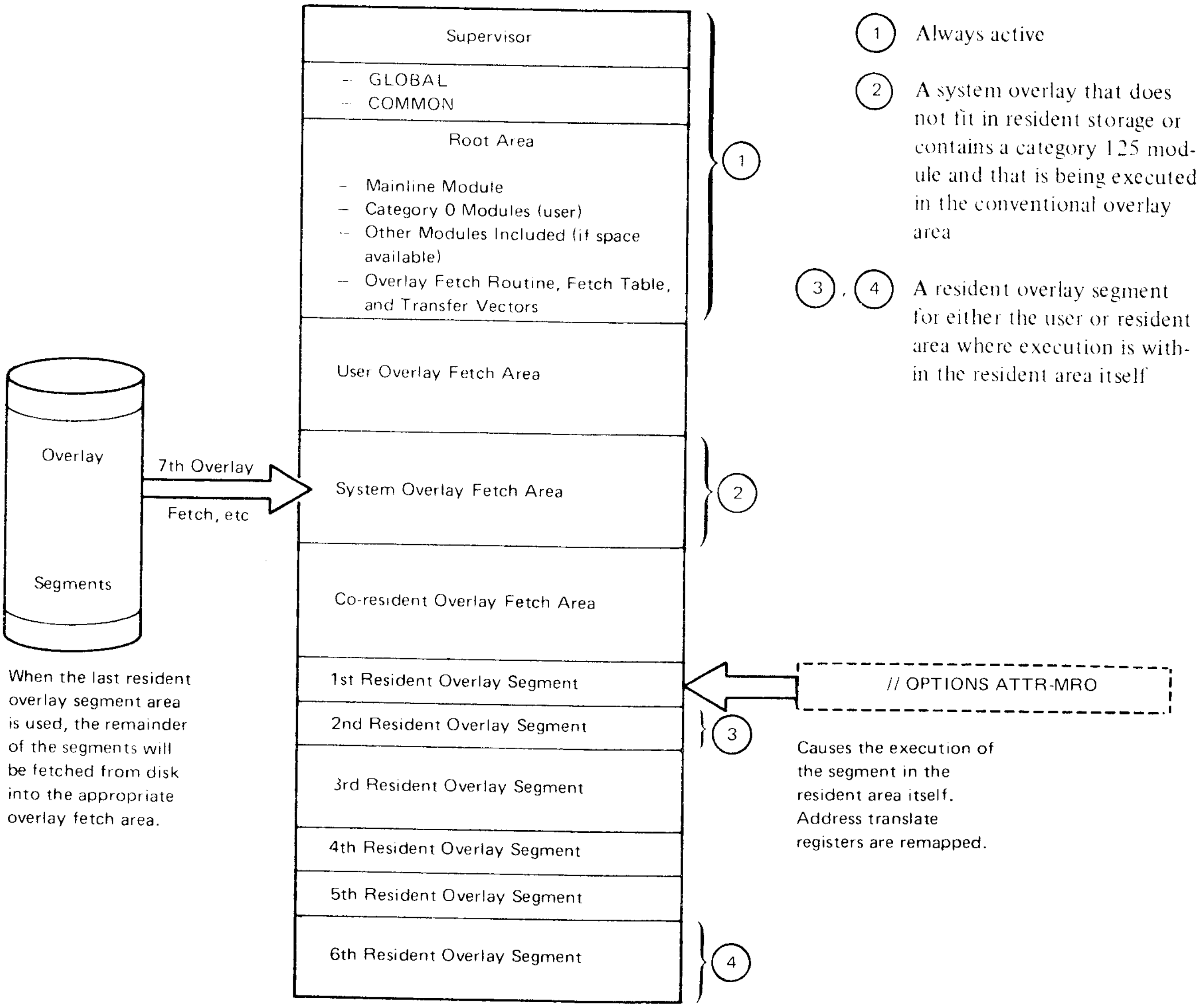
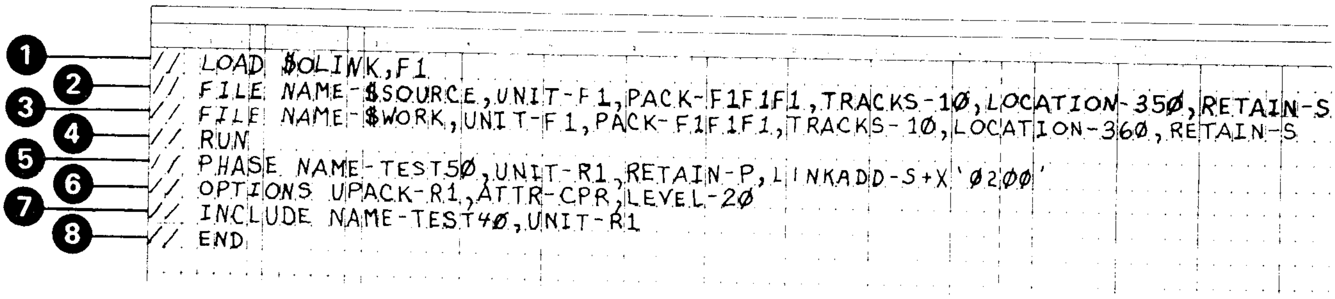

Supervisor
– GLOBAL
– COMMON
Root Area
– Mainline Module
– Category 0 Modules (user)
– Other Modules Included (if space available)
– Overlay Fetch Routine, Fetch Table, and Transfer Vectors
User Overlay Fetch Area
System Overlay Fetch Area
Co-resident Overlay Fetch Area
1st Resident Overlay Segment
2nd Resident Overlay Segment
3rd Resident Overlay Segment
4th Resident Overlay Segment
5th Resident Overlay Segment
6th Resident Overlay Segment
Overlay
Segments
7th Overlay
Fetch, etc
1
2
3
4
// OPTIONS ATTR-MRO
1 Always active
2 A system overlay that does not fit in resident storage or contains a category 125 module and that is being executed in the conventional overlay area
3, 4 A resident overlay segment for either the user or resident area where execution is within the resident area itself
Causes the execution of the segment in the resident area itself. Address translate registers are remapped.
When the last resident overlay segment area is used, the remainder of the segments will be fetched from disk into the appropriate overlay fetch area.

Figure 11. Schematic Execution of the Memory-Resident Overlay REMAP Technique and Overlay Fetch Routine

EXAMPLES 1 THROUGH 5

These five examples show the OCL statements and Overlay Linkage Editor control statements used to link-edit five programs. The notes with each example explain the purpose of the control statements in each job.

Example 1



```
// LOAD $OLINK,F1
// FILE NAME-$SOURCE,UNIT-F1,PACK-F1F1F1,TRACKS-1Ø,LOCATION-35Ø,RETAIN-S
// FILE NAME-$WORK,UNIT-F1,PACK-F1F1F1,TRACKS-1Ø,LOCATION-36Ø,RETAIN-S
// RUN
// PHASE NAME-TEST5Ø,UNIT-R1,RETAIN-P,LINKADD-S+X'Ø2ØØ'
// OPTIONS UPACK-R1,ATTR-CPR,LEVEL-2Ø
// INCLUDE NAME-TEST4Ø,UNIT-R1
// END
```

Notes on Example 1

1. The Overlay Linkage Editor is loaded from the fixed pack on drive 1.

2. The $SOURCE file of ten tracks is allocated on the fixed pack on drive 1 starting at track number 350. This is a scratch file.

3. The $WORK file of ten tracks is allocated on the fixed pack on drive 1 starting at track number 360. This is a scratch file.

4. Execution of $OLINK (last OCL card) is started.

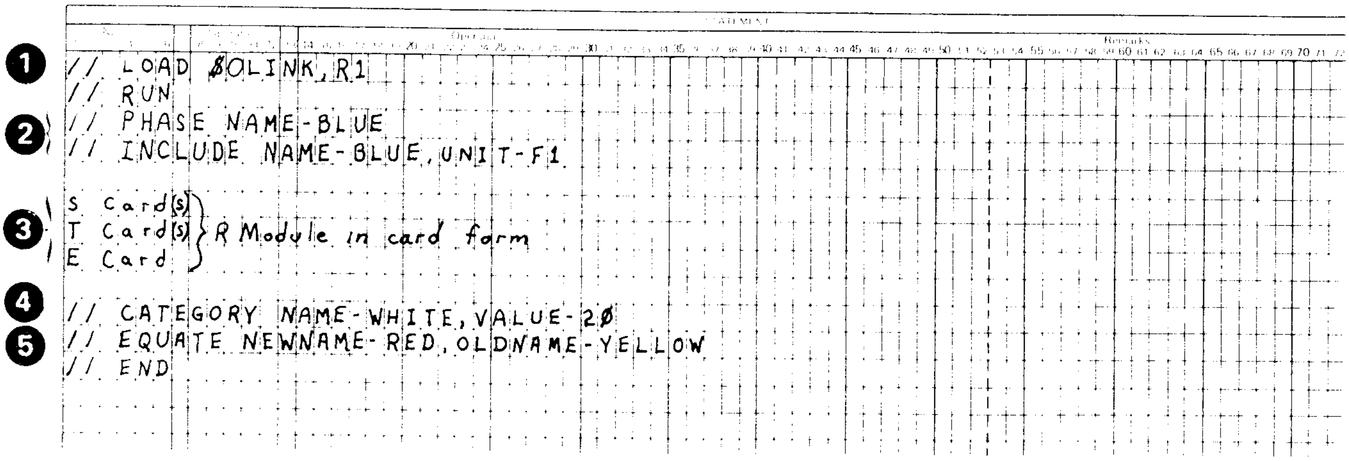
5. The output load module is generated as a permanent entry in the object library of the removable pack on drive 1 under the name TEST50. The start address is the end of the supervisor plus X'0200'.

6. User modules can be found on drive R1. This program is a checkpoint restart program. The level number is 20.

7. The mainline routine for this program is an object module in the object library of the removable pack on drive 1 under the name TEST40.

8. End of the Overlay Linkage Editor input. Automatic overlays are generated if needed

Example 2



```
// LOAD $OLINK,R1
// RUN
// PHASE NAME-BLUE
// INCLUDE NAME-BLUE,UNIT-F1

S Card(s) \
T Card(s)  } R Module in card form
E Card    /

// CATEGORY NAME-WHITE,VALUE-2Ø
// EQUATE NEWNAME-RED,OLDNAME-YELLOW
// END
```

Notes on Example 2

1. The Overlay Linkage Editor is loaded from the removable pack on drive 1. This pack is now the program pack. Because no file statements are given, the Overlay Linkage Editor finds from 10 to 30 tracks of work space on either F1 or R1.

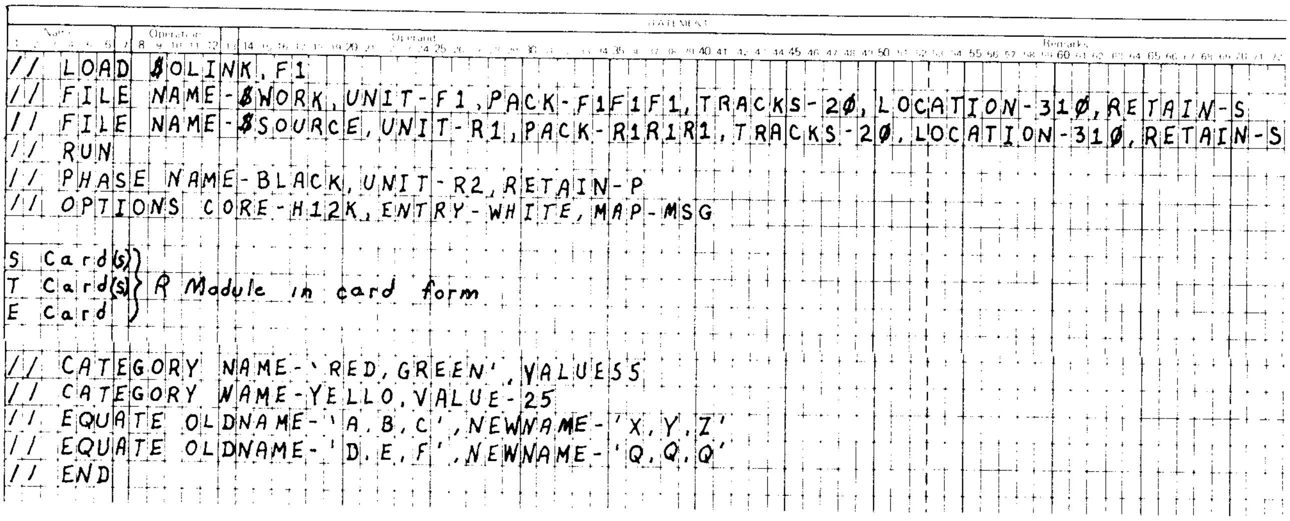
2. The output load module is a temporary entry in the object library of the program pack (see note 1) under the name BLUE. Look for user modules on the program pack because no OPTIONS card is given. The mainline routine for this program is BLUE and is located on F1.

3. One of the modules required for this job is in card form. The name of the module is picked up from the module name ESL entry on the S card.

4. For this link-edit the category of module WHITE is changed to a value of 20. WHITE is included by AUTOLINK.

5. The references to YELLOW in the object module are replaced by RED in the load module.

Example 3



```
// LOAD $OLINK,F1
// FILE NAME-$WORK,UNIT-F1,PACK-F1F1F1,TRACKS-2Ø,LOCATION-31Ø,RETAIN-S
// FILE NAME-$SOURCE,UNIT-R1,PACK-R1R1R1,TRACKS-2Ø,LOCATION-31Ø,RETAIN-S
// RUN
// PHASE NAME-BLACK,UNIT-R2,RETAIN-P
// OPTIONS CORE-H12K,ENTRY-WHITE,MAP-MSG

S Card(s) )
T Card(s) } R Module in card form
E Card    )

// CATEGORY NAME-'RED,GREEN',VALUE55
// CATEGORY NAME-YELLO,VALUE-25
// EQUATE OLDNAME-'A,B,C',NEWNAME-'X,Y,Z'
// EQUATE OLDNAME-'D,E,F',NEWNAME-'Q,Q,Q'
// END
```

Notes on Example 3

1. $WORK and $SOURCE are allocated on the two packs on drive 1. $WORK will be right on top of $SOURCE.

2. The object code is constructed so that the program can run in a 12.5K partition. The entry point WHITE is the start control address. Only messages are printed.

3. The mainline routine for this program is in card form. Therefore, no INCLUDE card is used for it. If a mainline is in card form it must be the first card deck read.

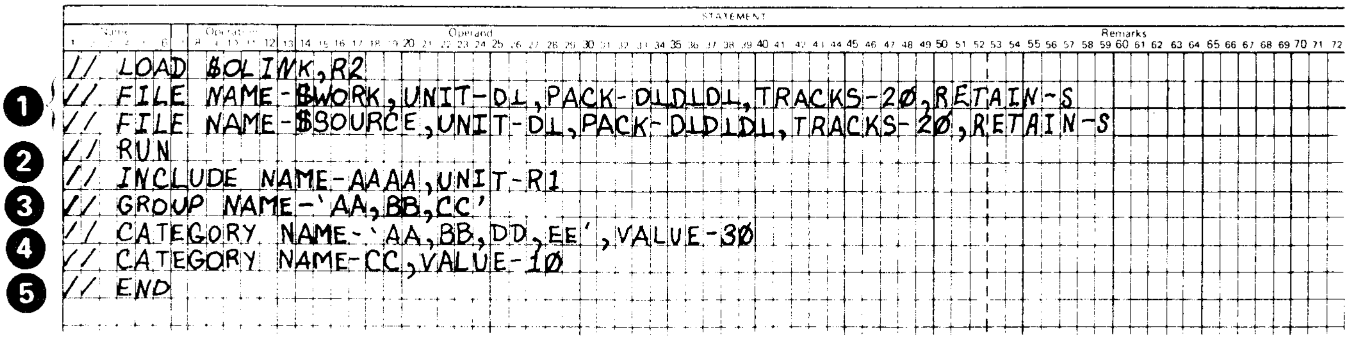
4. For this link edit the category of both modules RED and GREEN is changed to a value of 55.

5. If more than one module category value is to be changed, more than one CATEGORY card can be used.

6. The references to module names or entry points A, B, and C are replaced by X, Y, and Z, respectively.

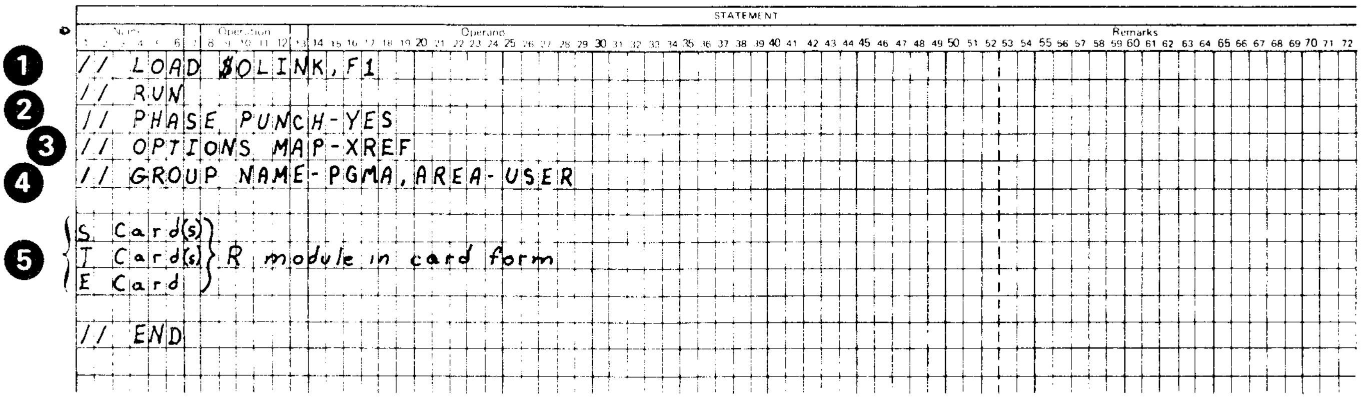
7. The references to module names or entry points D, E, and F are replaced by Q.

Example 4 (Model 15 only)



```
   // LOAD $OLINK,R2
1  // FILE NAME-$WORK,UNIT-D1,PACK-D1D1D1,TRACKS-2Ø,RETAIN-S
   // FILE NAME-$SOURCE,UNIT-D1,PACK-D1D1D1,TRACKS-2Ø,RETAIN-S
2  // RUN
   // INCLUDE NAME-AAAA,UNIT-R1
3  // GROUP NAME-'AA,BB,CC'
4  // CATEGORY NAME-'AA,BB,DD,EE',VALUE-3Ø
   // CATEGORY NAME-CC,VALUE-1Ø
5  // END
```

Example 5



```
1  // LOAD $OLINK,F1
2  // RUN
   // PHASE PUNCH-YES
3  // OPTIONS MAP-XREF
4  // GROUP NAME-PGMA,AREA-USER

   S Card(s) }
5  T Card(s) } R module in card form
   E Card    }

   // END
```

Notes on Example 4

1. $WORK and $SOURCE are allocated on a 5445.
2. A temporary entry is cataloged in the object library on the program pack under the name of the module included.
3. The overlays, if necessary, are constructed so that AA, BB, and CC are in main storage at the same time.
4. The category value of subroutines AA, BB, DD, and EE is 30 for this link edit.
5. Routines in the same group do not need to have the same category value. By giving the module a lower category value, its chance of being in the root segment increases.

Notes on Example 5

1. The work area is assigned by the linkage editor.
2. A load module is punched and no library entry is made.
3. A cross-reference list is printed on the storage map.
4. If PGMA is assigned to an overlay, it will appear in the user overlay area.
5. The only thing needed is an R module. The defaults are taken.

EXAMPLES 6 AND 7

These two examples of the same program show how the overlay structure of a load module can be changed by varying the input control statements. Both examples include the input control statements, the storage map printed by the Overlay Linkage Editor, and a graphic representation of the overlay structure. Figure 12 shows the calling sequence of the modules within the program.

Example 6

Two overlay load points are shown on the storage map (START ADDRESS heading). INIT (co-resident area) has the same load point as the system overlay area because it has no references to system modules and no category 4 modules appear in the program. A reference to a category 4 module does not disqualify a module from the co-resident area.

MULT4 and DIV4 are assigned to the root area by the Overlay Linkage Editor because of their low category values and small size. FINAL is assigned to the root area despite its high category value because it can be placed there without causing the program to exceed its main storage size. Normally GET6 or PUT6 would be included in the root segment before FINAL because of their lower category values, but the Overlay Linkage Editor does not include any system modules that are called by user modules until all system modules of the same category that are called only by other system modules are in the root segment. In this example, $$LPRT and $$MFRD will not fit.

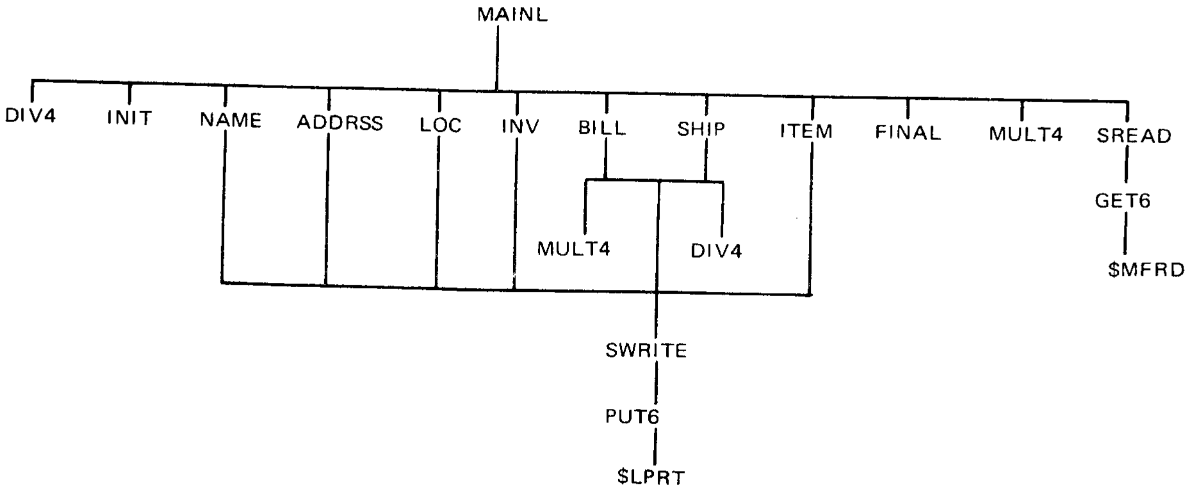
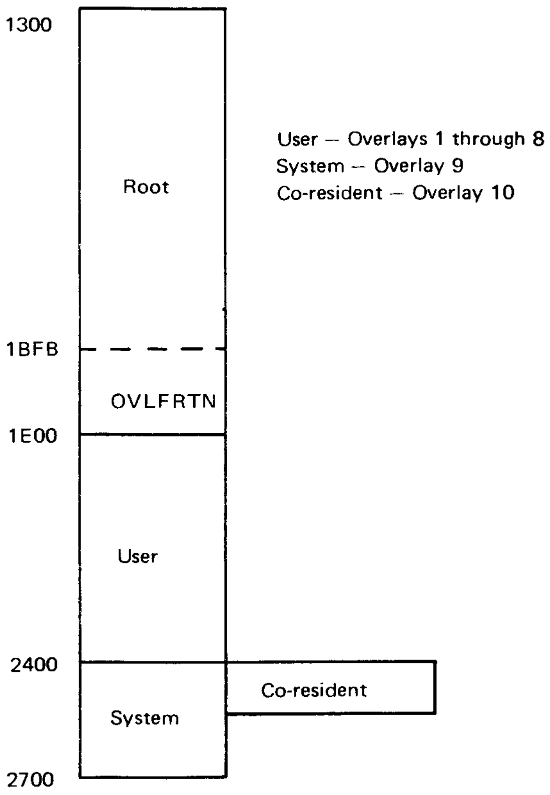


Figure 12. Calling Tree-structure of Modules Linked in Examples 6-7

Example 6

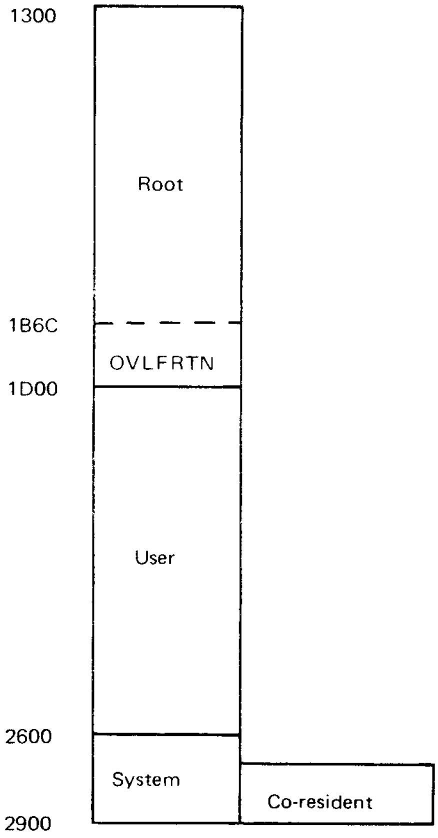


```
// LOAD $OLINK,F1
// FILE NAME-$SOURCE,UNIT-F1,PACK-F1F1F1,RETAIN-S,TRACKS-5
// FILE NAME-$WORK,UNIT-F1,PACK-F1F1F1,RETAIN-S,TRACKS-5
// RUN
// PHASE NAME-OLSMPL,UNIT-F1
// OPTIONS MAP-XREF,CORE-5K
// CATEGORY NAME-'MULT4,DIV4',VALUE-4
// CATEGORY NAME-'GET6,PUT6',VALUE-6
// CATEGORY NAME-SWRITE,VALUE-136
// CATEGORY NAME-SREAD,VALUE-138
// CATEGORY NAME-NAME,VALUE-20
// CATEGORY NAME-ADDRSS,VALUE-21
// CATEGORY NAME-LOC,VALUE-22
// CATEGORY NAME-INV,VALUE-23
// CATEGORY NAME-BILL,VALUE-24
// CATEGORY NAME-SHIP,VALUE-25
// CATEGORY NAME-ITEM,VALUE-26
// CATEGORY NAME-'INIT,FINAL',VALUE-90
// END
```

```
             OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST      11/23/71


 START     OVERLAY    CATEGORY  NAME AND      CODE LENGTH      REFERENCED
ADDRESS  NUMBER AREA              ENTRY   HEXADECIMAL DECIMAL     BY

 1300                    0      MAINL         082D     2093
 1B20                   0,4     MULT4         003F       63    MAINL  BILL
 1B6C                   0,4     DIV4          0041       65    MAINL  SHIP
 1BAD                   0,90    FINAL         004E       78    MAINL
 1BFB                           OVLFRTN       0180      384
 1E00        1    U     0,20    NAME          0320      800    MAINL
 2200        1    U     0,136   SWRITE        0192      402    NAME   ADDRSS LOC    INV    BILL
                                                               SHIP   ITEM
 1E00        2    U     0,21    ADDRSS        0302      770    MAINL
 2200        2    U     0,136   SWRITE        0192      402    NAME   ADDRSS LOC    INV    BILL
                                                               SHIP   ITEM
 1E00        3    U     0,22    LOC           028A      650    MAINL
 2100        3    U     0,136   SWRITE        0192      402    NAME   ADDRSS LOC    INV    BILL
                                                               SHIP   ITEM
 1E00        4    U     0,23    INV           02E4      740    MAINL
 2100        4    U     0,136   SWRITE        0192      402    NAME   ADDRSS LOC    INV    BILL
                                                               SHIP   ITEM
 1E00        5    U     0,24    BILL          02CA      714    MAINL
 2100        5    U     0,136   SWRITE        0192      402    NAME   ADDRSS LOC    INV    BILL
                                                               SHIP   ITEM
 1E00        6    U     0,25    SHIP          0306      774    MAINL
 2200        6    U     0,136   SWRITE        0192      402    NAME   ADDRSS LOC    INV    BILL
                                                               SHIP   ITEM
 1E00        7    U     0,26    ITEM          0296      662    MAINL
 2100        7    U     0,136   SWRITE        0192      402    NAME   ADDRSS LOC    INV    BILL
                                                               SHIP   ITEM
 1E00        8    U     0,138   SREAD         01FA      506    MAINL
 2400        9    S     0,6     GET6          0044       68    SREAD
 2444        9    S     0,6     PUT6          003A       58    SWRITE
 247E        9    S      6      $$LPRT        00FB      251    PUT6
 2579        9    S      6      $$MFRD        0145      325    GET6
 2400       10    C     0,90    INIT          00D0      208    MAINL

OL100 I  THE TOTAL CORE USED BY OLSMPL IS  5120 DECIMAL
OL101 I  THE START CONTROL ADDRESS OF THIS MODULE IS 1300.
OL102 I  THE NON-OVERLAY CORE SIZE IS      9466 DECIMAL
OL104 I  TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS   55
         NAME-OLSMPL,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-O
```

Example 7

This example shows three overlay load points. Overlays 1, 2, 3, and 4 are in the user area. Overlays 5 and 6 are in the system area, and overlay 7 is in the co-resident area (user module in the system area). GROUP statements were entered to increase the size of the user overlay area.



```
// LOAD $OLINK,F1
// FILE NAME-$SOURCE,UNIT-F1,PACK-F1F1F1,RETAIN-S,TRACKS-5
// FILE NAME-$WORK,UNIT-F1,PACK-F1F1F1,RETAIN-S,TRACKS-5
// RUN
// PHASE NAME-OLSMPL,UNIT-F1
// OPTIONS MAP-XREF,CORE-15K
// CATEGORY NAME-'MULT4,DIV4',VALUE-4
// CATEGORY NAME-'GET6,PUT6',VALUE-6
// CATEGORY NAME-SWRITE,VALUE-136
// CATEGORY NAME-SREAD,VALUE-138
// CATEGORY NAME-NAME,VALUE-20
// CATEGORY NAME-ADDRSS,VALUE-21
// CATEGORY NAME-LOC,VALUE-22
// CATEGORY NAME-INV,VALUE-23
// CATEGORY NAME-BILL,VALUE-24
// CATEGORY NAME-SHIP,VALUE-25
// CATEGORY NAME-ITEM,VALUE-26
// CATEGORY NAME-'INIT,FINAL',VALUE-90
// GROUP NAME-'NAME,LOC'
// GROUP NAME-'ADDRSS,INV'
// GROUP NAME-'BILL,SHIP'
// END
```

```
                 OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST       11/23/71

 START     OVERLAY    CATEGORY  NAME AND       CODE LENGTH       REFERENCED
ADDRESS  NUMBER AREA              ENTRY    HEXADECIMAL DECIMAL      BY

 1300                      0     MAINL          082D     2093
 1B2D                    0,4     MULT4          003F       63    MAINL   BILL
 1B6C                            OVLFRTN        0160      352
 1D00       1     U     0,20     NAME           0320      800    MAINL
 2100       1     U     0,136    SWRITE         0192      402    NAME    ADDRSS LOC     INV     BILL
                                                                 SHIP    ITEM
 2292       1     U     0,22     LOC            028A      650    MAINL
 1D00       2     U     0,21     ADDRSS         0302      770    MAINL
 2100       2     U     0,136    SWRITE         0192      402    NAME    ADDRSS LOC     INV     BILL
                                                                 SHIP    ITEM
 2292       2     U     0,23     INV            02E4      740    MAINL
 1D00       3     U     0,24     BILL           02CA      714    MAINL
 2000       3     U     0,136    SWRITE         0192      402    NAME    ADDRSS LOC     INV     BILL
                                                                 SHIP    ITEM
 2192       3     U     0,25     SHIP           0306      774    MAINL
 1D00       4     U     0,26     ITEM           0296      662    MAINL
 2000       4     U     0,136    SWRITE         0192      402    NAME    ADDRSS LOC     INV     BILL
                                                                 SHIP    ITEM
 2200       4     U     0,138    SREAD          01FA      506    MAINL
 2600       5     S     0,4      DIV4           0041       65    MAINL   SHIP
 2600       6     S     0,6      GET6           0044       68    SREAD
 2644       6     S     0,6      PUT6           003A       58    SWRITE
 2671       6     S      6       $$LPRT         00FB      251    PUT6
 2779       6     S      6       $$MFRD         0145      325    GET6
 2700       7     C     0,90     INIT           00D0      208    MAINL
 27D0       7     C     0,90     FINAL          004E       78    MAINL

OL100 I   THE TOTAL CORE USED BY OLSMPL IS  5632 DECIMAL
OL101 I   THE START CONTROL ADDRESS OF THIS MODULE IS 1300.
OL102 I   THE NON-OVERLAY CORE SIZE IS     9466 DECIMAL
OL104 I   TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS   51
          NAME-OLSMPL,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-O
```

EXAMPLES 8 THROUGH 11

These four examples show how the overlay structure of a program can be changed by varying the input control statements. The changes result from varying the category values of modules and varying the main storage size. All four examples show the input control statements, the storage map printed by the Overlay Linkage Editor, and a graphic representation of the storage map. Figure 13 shows the tree structure of the program.

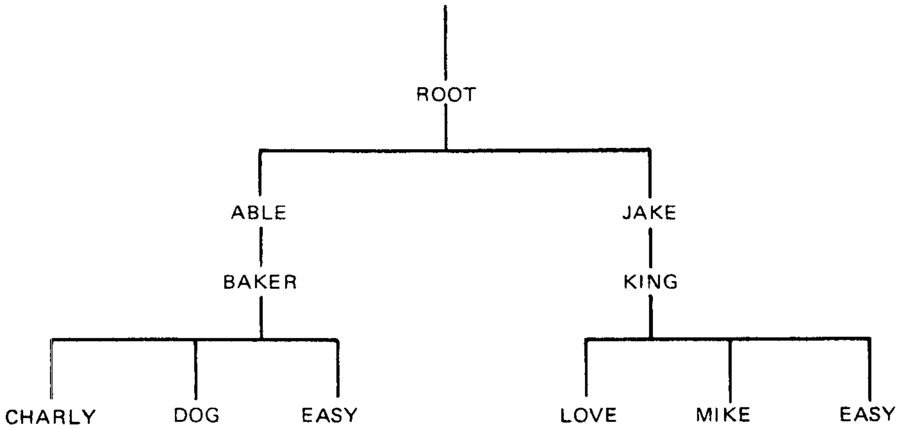
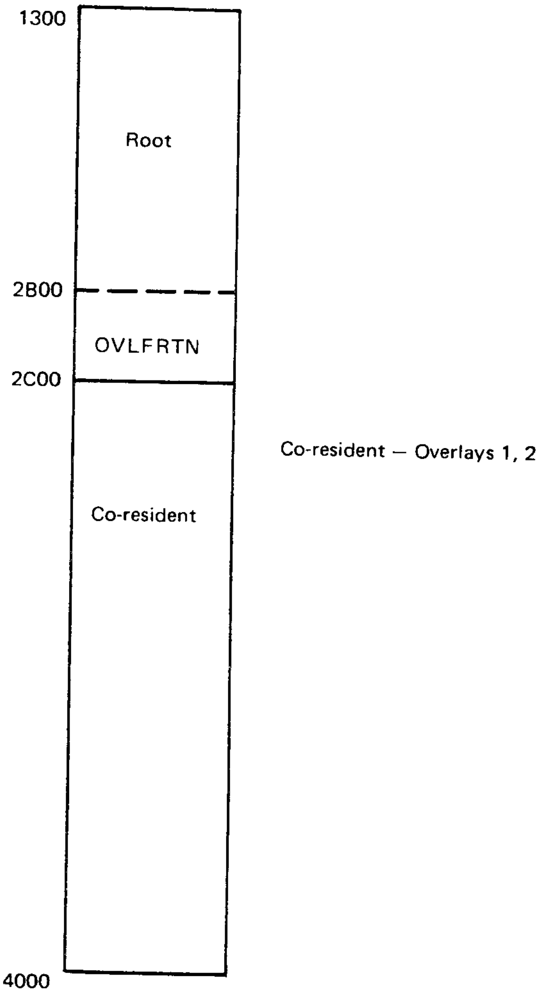


Figure 13. Calling Tree-structure of Modules Linked in Examples 8-11

Example 8

All modules except ROOT, BAKER, and KING are given overlay category values. Because no system modules (category values 1-7) are present, only one overlay area is assigned. Any module assigned to an overlay segment must be assigned to the same segments as its descendants. Because ABLE calls CHARLY and DOG (through BAKER), these three modules are assigned to one segment. Likewise, JAKE calls LOVE and MIKE (through KING) and is assigned, with them, to the second overlay segment. EASY would have been assigned to both segments, but space was available in the root so EASY was included in the root segment.

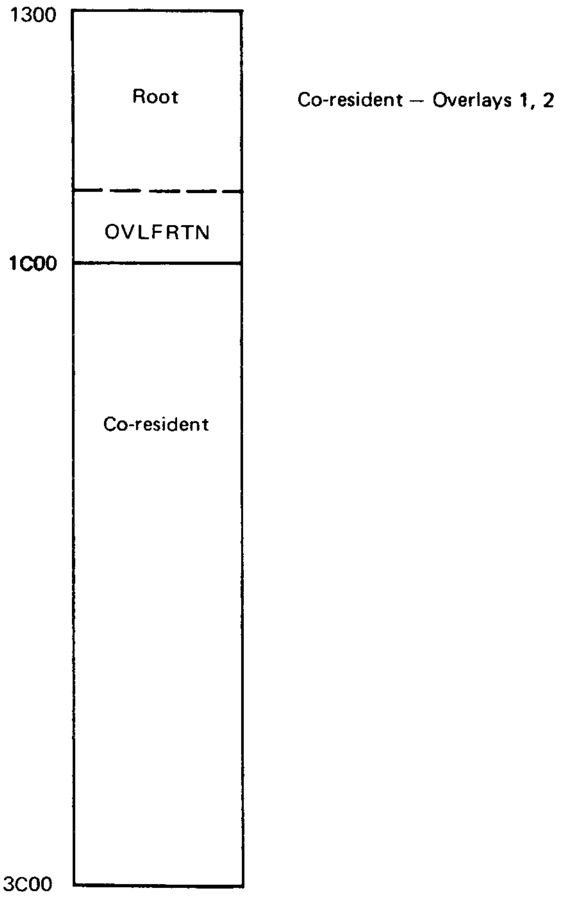


```
// CALL OLINK,R1
XX LOG PRINTER
XX NOHALT
XX PUNCH MFCU2
XX LOAD $OLINK,F1
XX FILE NAME-$SOURCE,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX FILE   NAME-$WORK,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX RUN
// RUN
// PHASE NAME-RIZZI,UNIT-F1
// OPTIONS MAP-XREF,UPACK-R1,CORE-T11K
// INCLUDE NAME-ROOT,UNIT-R1
// CATEGORY NAME-'ABLE,JAKE',VALUE-8
// END
```

```
        OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST       07/15/71


 START     OVERLAY    CATEGORY  NAME AND       CODE LENGTH         REFERENCED
ADDRESS  NUMBER AREA              ENTRY    HEXADECIMAL DECIMAL        BY

 1300                    0     ROOT             0800     2048
 130A                            ROOT
 1800                    0     BAKER            0400     1024     ABLE
 1805                            BAKER                            ABLE
 1F00                    0     KING             0400     1024     JAKF
 2008                            KING                             JAKE
 2300                   37     EASY             0800     2048     BAKER  KING
 250A                            EASY                             BAKER  KING
 2800                          OVLFRTN          00C4      196
 2C00       1     C     0,8    ABLE             0328      808     ROOT
 2F0A                            ABLE                             ROOT
 2F28       1     C     37     CHARLY           0800     2048     BAKER
 2F34                            CHARLY                           BAKER
 3728       1     C     37     DOG              0800     2048     BAKER
 3A31                            DOG                              BAKER
 2C00       2     C     0,8    JAKE             0400     1024     ROOT
 2C0A                            JAKE                             ROOT
 3000       2     C     37     LOVE             0800     2048     KING
 310C                            LOVE                             KING
 3800       2     C     37     MIKE             0800     2048     KING
 380C                            MIKE                             KING

OL100 I   THE TOTAL CORE USED BY RIZZI  IS 11520 DECIMAL
OL101 I   THE START CONTROL ADDRESS OF THIS MODULE IS 130A.
OL102 I   THE NON-OVERLAY CORE SIZE IS     16168 DECIMAL
OL104 I   TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS    67
          NAME-RIZZI ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-O
```

Example 9

All modules have an overlay category value (non-zero). Because there are no system category values (1-7), only one overlay area is assigned by the Overlay Linkage Editor. Only two overlay segments are possible because each calling module must be in the same segment as its descendants. Module EASY could be given a category value of 0 so it would be placed in the root rather than in both segments.

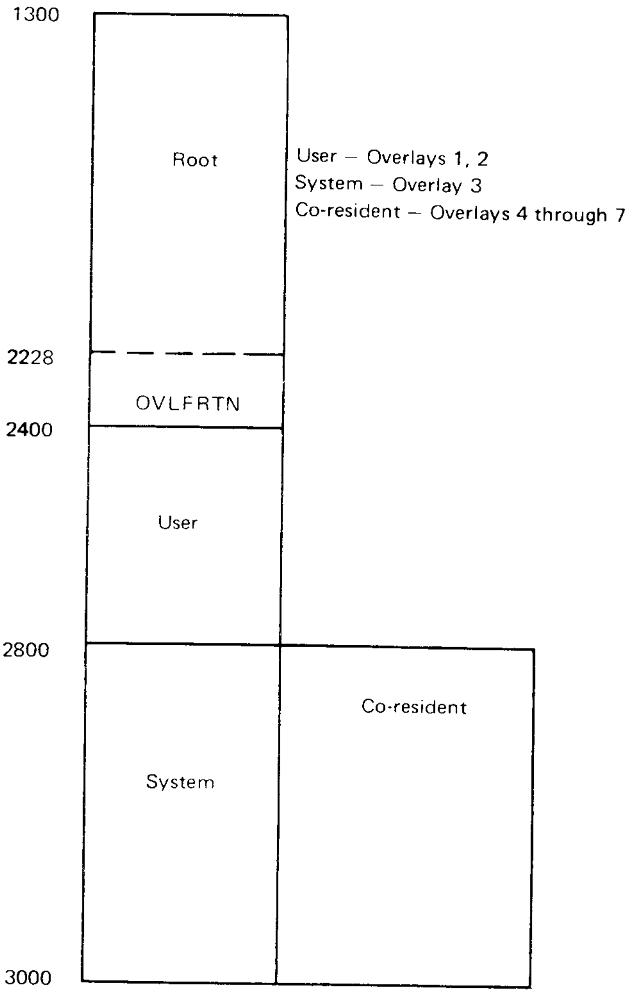


```
// CALL OLINK,R1
XX LOG PRINTER
XX NOHALT
XX PUNCH MFCU2
XX LOAD $OLINK,F1
XX FILE NAME-$SOURCE,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX FILE   NAME-$WORK,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX RUN
// RUN
// PHASE NAME-RIZ,UNIT-F1
// OPTIONS MAP-XREF,UPACK-R1,CORE-H10K
// INCLUDE NAME-ROOT,UNIT-R1
// CATEGORY NAME-'BAKER,KING',VALUE-8
// CATEGORY NAME-'ABLE,JAKE',VALUE-8
// END
```

```
              OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST       07/15/71


 START     OVERLAY    CATEGORY  NAME AND       CODE LENGTH       REFERENCED
ADDRESS  NUMBER AREA              ENTRY    HEXADECIMAL DECIMAL       BY

 1300                    0      ROOT           0800     2048
 130A                             ROOT
 1800                           OVLFRTN        00F0      240
 1C00       1     C     0,8     ABLE           0328      808    ROOT
 1F0A                             ABLE                          ROOT
 1F28       1     C     0,8     BAKER          0400     1024    ABLE
 1F2D                             BAKER                         ABLE
 2328       1     C     37      CHARLY         0800     2048    BAKER
 2334                             CHARLY                        BAKER
 2828       1     C     37      DOG            0800     2048    BAKER
 2E31                             DOG                           BAKER
 3328       1     C     37      EASY           0800     2048    BAKER   KING
 3532                             EASY                          BAKER   KING
 1C00       2     C     0,8     JAKE           0400     1024    ROOT
 1C0A                             JAKE                          ROOT
 2000       2     C     0,8     KING           0400     1024    JAKE
 210B                             KING                          JAKE
 2400       2     C     37      LOVE           0800     2048    KING
 250C                             LOVE                          KING
 2C00       2     C     37      MIKE           0800     2048    KING
 2C0C                             MIKE                          KING
 3400       2     C     37      EASY           0800     2048    BAKER   KING
 360A                             EASY                          BAKER   KING

OL100 I   THE TOTAL CORE USED BY RIZ     IS 10496 DECIMAL
OL101 I   THE START CONTROL ADDRESS OF THIS MODULE IS 130A.
OL102 I   THE NON-OVERLAY CORE SIZE IS    16168 DECIMAL
OL104 I   TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS   75
          NAME-RIZ   ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-O
```

Example 10

Module EASY is assigned a category value of 2. Because the Overlay Linkage Editor assumes that categories 1, 2, 3, 5, 6, and 7 are system I/O modules, modules BAKER and KING are I/O dependent and are assigned to user overlay segments. The remaining four modules are I/O independent and are assigned to co-resident overlay segments.

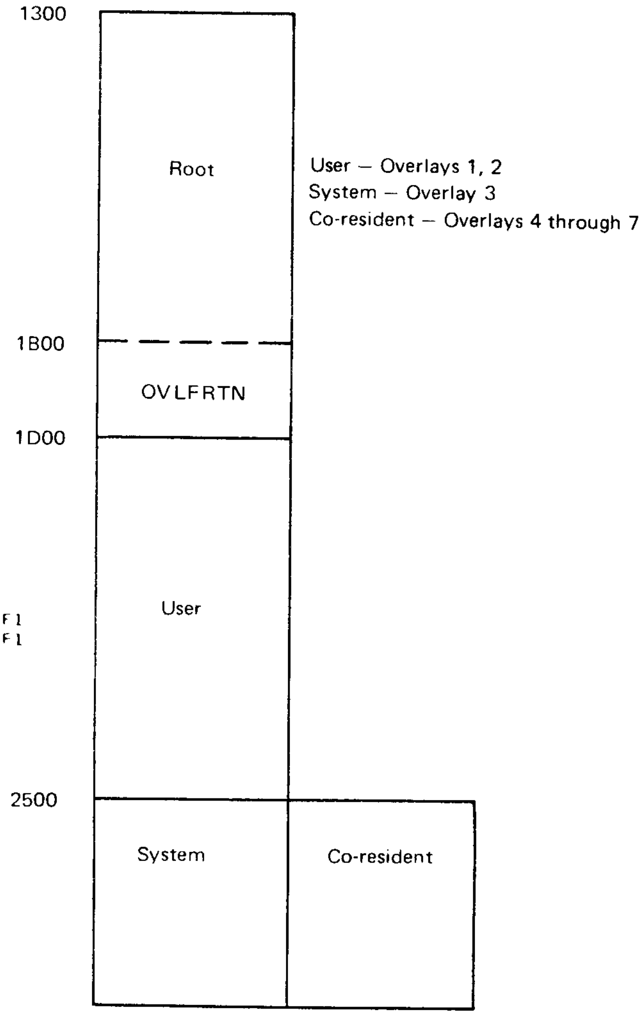


```
// CALL OLINK,R1
XX LOG PRINTER
XX NOHALT
XX PUNCH MFCU2
XX LOAD $OLINK,F1
XX FILE NAME-$SOURCE,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX FILE    NAME-$WORK,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX RUN
// RUN
// PHASE NAME-EV2,UNIT-F1
// OPTIONS MAP-XREF,CORE-H7K,UPACK-R1
// INCLUDE NAME-ROOT,UNIT-R1
// CATEGORY NAME-'BAKER,KING',VALUE-8
// CATEGORY NAME-EASY,VALUE-2
// END
```

```
              OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST          07/15/71


 START     OVERLAY    CATEGORY  NAME AND      CODE LENGTH        REFERENCED
ADDRESS  NUMBER AREA              ENTRY    HEXADECIMAL DECIMAL       BY

 1300                    0      ROOT             0800    2048
 130A                             ROOT
 1800                    0      ABLE             0328     808    ROOT
 1E0A                             ABLE                           ROOT
 1E28                    0      JAKE             0400    1024    ROOT
 1E32                             JAKE                           ROOT
 2228                           OVLFRTN          00F2     242
 2400       1      U    0,8     BAKER            0400    1024    ABLE
 2405                             BAKER                          ABLE
 2400       2      U    0,8     KING            `0400    1024    JAKE
 250B                             KING                           JAKE
 2800       3      S   37,2     EASY             0800    2048    BAKER   KING
 2A0A                             EASY                           BAKER   KING
 2800       4      C   37       CHARLY           0800    2048    BAKER
 280C                             CHARLY                         BAKER
 2800       5      C   37       DOG              0800    2048    BAKER
 2809                             DOG                            BAKER
 2800       6      C   37       LOVE             0800    2048    KING
 290C                             LOVE                           KING
 2800       7      C   37       MIKE             0800    2048    KING
 280C                             MIKE                           KING

OL100 I  THE TOTAL CORE USED BY EV2     IS  7424 DECIMAL
OL101 I  THE START CONTROL ADDRESS OF THIS MODULE IS 130A.
OL102 I  THE NON-OVERLAY CORE SIZE IS    16168 DECIMAL
OL104 I  TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS   72
         NAME-EV2    ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-0
```

Example 11

All modules except ROOT have an overlay category (nonzero). Because BAKER and KING call a system module (EASY), they are assigned to user overlay segments. Modules that call BAKER and KING (ABLE and JAKE) are put into the same overlay segment as the modules they call. Modules that do not call system modules are assigned to the co-resident area.



```
// CALL OLINK,R1
XX LOG PRINTER
XX NOHALT
XX PUNCH MFCU2
XX LOAD $OLINK,F1
XX FILE NAME-$SOURCE,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX FILE    NAME-$WORK,RETAIN-S,TRACKS-25,PACK-F1F1F1,UNIT-F1
XX RUN
// RUN
// PHASE NAME-EV28,UNIT-F1
// OPTIONS OPACK-R1,MAP-XREF,CORE-16K
// INCLUDE NAME-ROOT,UNIT-R1
// CATEGORY NAME-'ABLE,BAKER,JAKE,KING',VALUE-8
// CATEGORY NAME-EASY,VALUE-2
// END
```

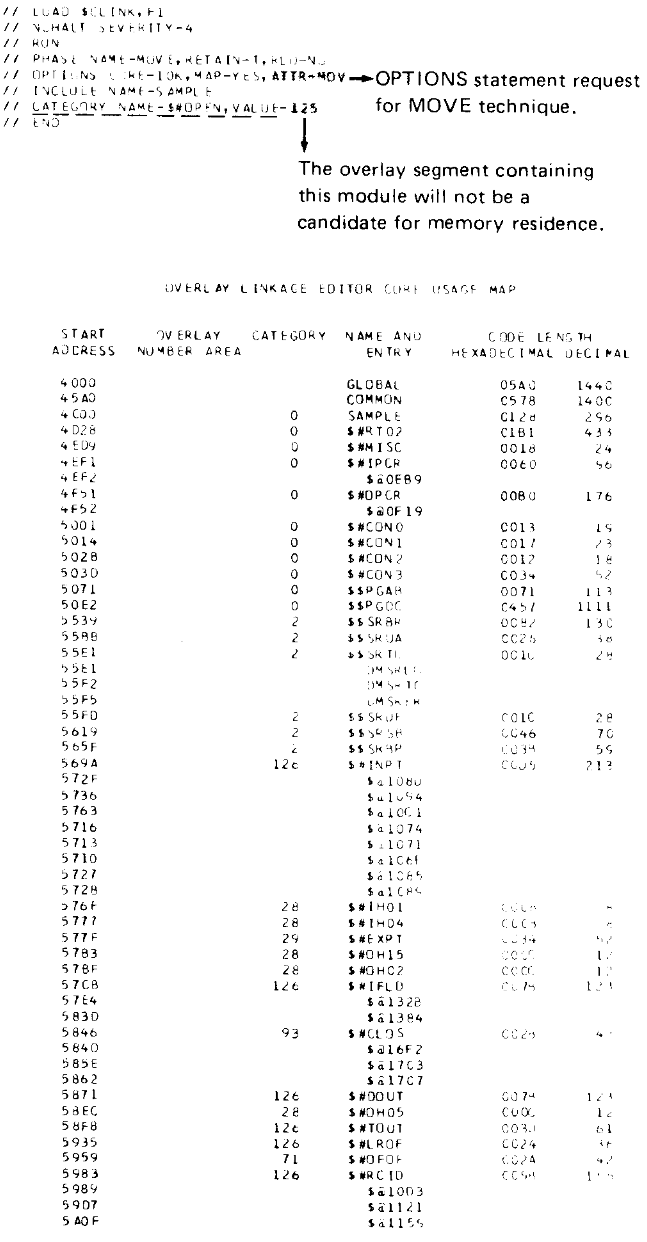
```
              OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST        07/15/71


 START      OVERLAY    CATEGORY  NAME AND       CODE LENGTH        REFERENCED
ADDRESS  NUMBER AREA               ENTRY    HEXADECIMAL DECIMAL       BY

 1300                      0     ROOT           0800     2048
 130A                              ROOT
 1B00                            OVLFRTN        0108      264
 1D00       1     U       0,8    ABLE           0328      808    ROOT
 200A                              ABLE                          ROOT
 2028       1     U       0,8    BAKER          0400     1024    ABLE
 202D                              BAKER                         ABLE
 1D00       2     U       0,8    JAKE           0400     1024    ROOT
 1D0A                              JAKE                          ROOT
 2100       2     U       0,8    KING           0400     1024    JAKE
 2208                              KING                          JAKE
 2500       3     S      37,2    EASY           0800     2048    BAKER   KING
 270A                              EASY                          BAKER   KING
 2500       4     C       37     CHARLY         0800     2048    BAKER
 250C                              CHARLY                        BAKER
 2500       5     C       37     DOG            0800     2048    BAKER
 2809                              DOG                           BAKER
 2500       6     C       37     LOVE           0800     2048    KING
 260C                              LOVE                          KING
 2500       7     C       37     MIKE           0800     2048    KING
 250C                              MIKE                          KING

OL100 I   THE TOTAL CORE USED BY EV28    IS  6656 DECIMAL
OL101 I   THE START CONTROL ADDRESS OF THIS MODULE IS 130A.
OL102 I   THE NON-OVERLAY CORE SIZE IS    16168 DECIMAL
OL104 I   TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS   72
          NAME-EV28  ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-O
```

EXAMPLES 12 AND 13

These two examples show the same RPG program link-edited with both memory resident overlay techniques. Note that the ATTR-MRO program requires 2K more to execute in than the ATTR-MOV program.

Example 12

This example shows a link-edit of an RPG program requesting the memory resident overlay MOVE technique (ATTR-MOV). This technique can also be requested by using the COMPILE statement at compile time. The link-edit map for this technique looks exactly like a conventional link-edit of the program.

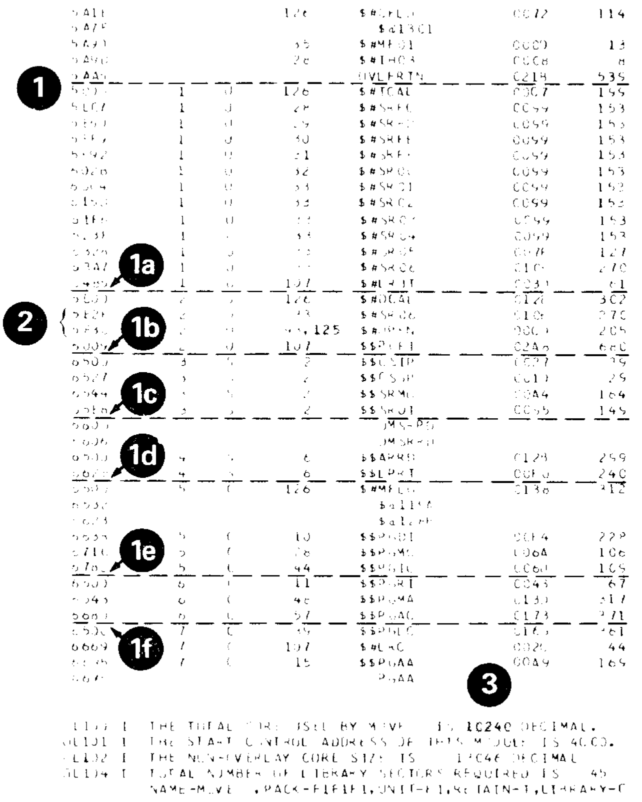


```
// LOAD $OLINK,F1
// NOHALT SEVERITY-4
// RUN
// PHASE NAME-MOVE,RETAIN-T,RLD-NO
// OPTIONS CORE-10K,MAP-YES,ATTR-MOV
// INCLUDE NAME-SAMPLE
// CATEGORY NAME-$#OPEN,VALUE-125
// END
```

OPTIONS statement request for MOVE technique.

The overlay segment containing this module will not be a candidate for memory residence.

```
                      OVERLAY LINKAGE EDITOR CORE USAGE MAP

 START      OVERLAY      CATEGORY   NAME AND        CODE LENGTH
ADDRESS   NUMBER AREA                ENTRY      HEXADECIMAL DECIMAL

 4000                               GLOBAL           05A0      1440
 45A0                               COMMON           0578      1400
 4C00                        0      SAMPLE           0128       296
 4D28                        0      $#RT02           01B1       433
 4ED9                        0      $#MISC           0018        24
 4EF1                        0      $#IPCR           0060        96
 4EF2                                 $a0E89
 4F51                        0      $#OPCR           00B0       176
 4F52                                 $a0F19
 5001                        0      $#CON0           0013        19
 5014                        0      $#CON1           0017        23
 5028                        0      $#CON2           0012        18
 503D                        0      $#CON3           0034        52
 5071                        0      $$PGAB           0071       113
 50E2                        0      $$PGDC           0457      1111
 5539                        2      $$SRBR           0082       130
 55BB                        2      $$SRUA           0026        38
 55E1                        2      $$SRTC           001C        28
 55E1                                 DMSRTC
 55F2                                 DMSRTC
 55F5                                 DMSRTR
 55FD                        2      $$SRUF           001C        28
 5619                        2      $$SRSB           0046        70
 565F                        2      $$SR9P           003B        59
 569A                      126      $#INPT           00D5       213
 572F                                 $a108D
 5736                                 $a1094
 5763                                 $a10C1
 5716                                 $a1074
 5713                                 $a1071
 5710                                 $a106F
 5727                                 $a1085
 572B                                 $a10B9
 576F                       28      $#IH01           0008         8
 5777                       28      $#IH04           0008         8
 577F                       29      $#EXPT           0034        52
 57B3                       28      $#OH15           000C        12
 57BF                       28      $#OH02           000C        12
 57CB                      126      $#IFLD           007B       123
 57E4                                 $a1328
 583D                                 $a1384
 5846                       93      $#CLOS           002B        43
 5840                                 $a16F2
 585E                                 $a17C3
 5862                                 $a17C7
 5871                      126      $#OOUT           007B       123
 58EC                       28      $#OH05           000C        12
 58F8                      126      $#TOUT           003D        61
 5935                      126      $#LROF           0024        36
 5959                       71      $#OFOF           002A        42
 5983                      126      $#RCID           009B       155
 5989                                 $a1003
 5907                                 $a1121
 5A0F                                 $a1155
```



```
5A11                       126    $#CFLO           0072     114
5A7F                                $a13C1
5A91                        55    $#MF01           000D      13
5A9E                        28    $#IH03           0008       8
5AA6                                OVLFRTN        0218     535
5C00         1      0      126    $#TCAL           00C7     199
5CC7         1      0       28    $#SRFC           0099     153
5D60         1      0       29    $#SR[illegible]  0099     153
5DF9         1      0       30    $#SRFF           0099     153
5E92         1      0       31    $#SRF[illegible] 0099     153
5F2B         1      0       32    $#SR00           0099     153
5FC4         1      0       33    $#SR01           0099     153
605D         1      0       33    $#SR02           0099     153
60F6         1      0       [illegible] $#SR03     0099     153
618F         1      0       34    $#SR04           0099     153
6228         1      0       [illegible] $#SR05     007F     127
62A7         1      0       [illegible] $#SR06     010E     270
63B5         1      0      107    $#LRIT           003D      61
5C00         2      0      126    $#OCAL           012E     302
5D2E         2      0       43    $#SR06           010E     270
5E3C         2      0   [illegible],125 $#OPEN     00CD     205
5F09         2      0      107    $$PCFI           02A8     680
5C00         3      0        2    $$CSTP           0027      39
5C27         3      0        2    $$CSSP           001D      29
5C44         3      0        2    $$SRMC           00A4     164
5CE8         3      0        2    $$SRUT           0095     149
5C00                                JMS-PG
5C06                                JMSRHI
5C00         4      0        6    $$ARRD           0103     259
5C2[illegible] 4    0        6    $$LPRT           00F0     240
5C00         5      0      126    $#MFLC           0138     312
5C3C                                $a11FA
5C73                                $a12FF
5D3[illegible] 5    0       10    $$PGDI           00E4     228
5E1C         5      0       28    $$PGMC           006A     106
5E86         5      0       44    $$PGIC           0069     105
5C00         6      0       11    $$PGRT           0043      67
5C43         6      0       46    $$PGMA           013D     317
5D80         6      0       57    $$PGAC           0173     371
5C00         7      0       49    $$PGLC           0169     361
5D69         7      0      107    $#LRC            002C      44
5D95         7      0       15    $$PGAA           00A9     169
5D9F                                PGAA

OL100 I  THE TOTAL CORE USED BY MOVE      IS 10240 DECIMAL.
OL101 I  THE START CONTROL ADDRESS OF THIS MODULE IS 4000.
OL102 I  THE NON-OVERLAY CORE SIZE IS      [illegible] DECIMAL
OL104 I  TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS    45
         NAME-MOVE  ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-C
```

1 Segments do not have to start on 2K boundaries (1a to 1f).

2 This overlay segment will not be a candidate for resident storage because of the category 125 module.

3 Core used is less than REMAP.

Example 13

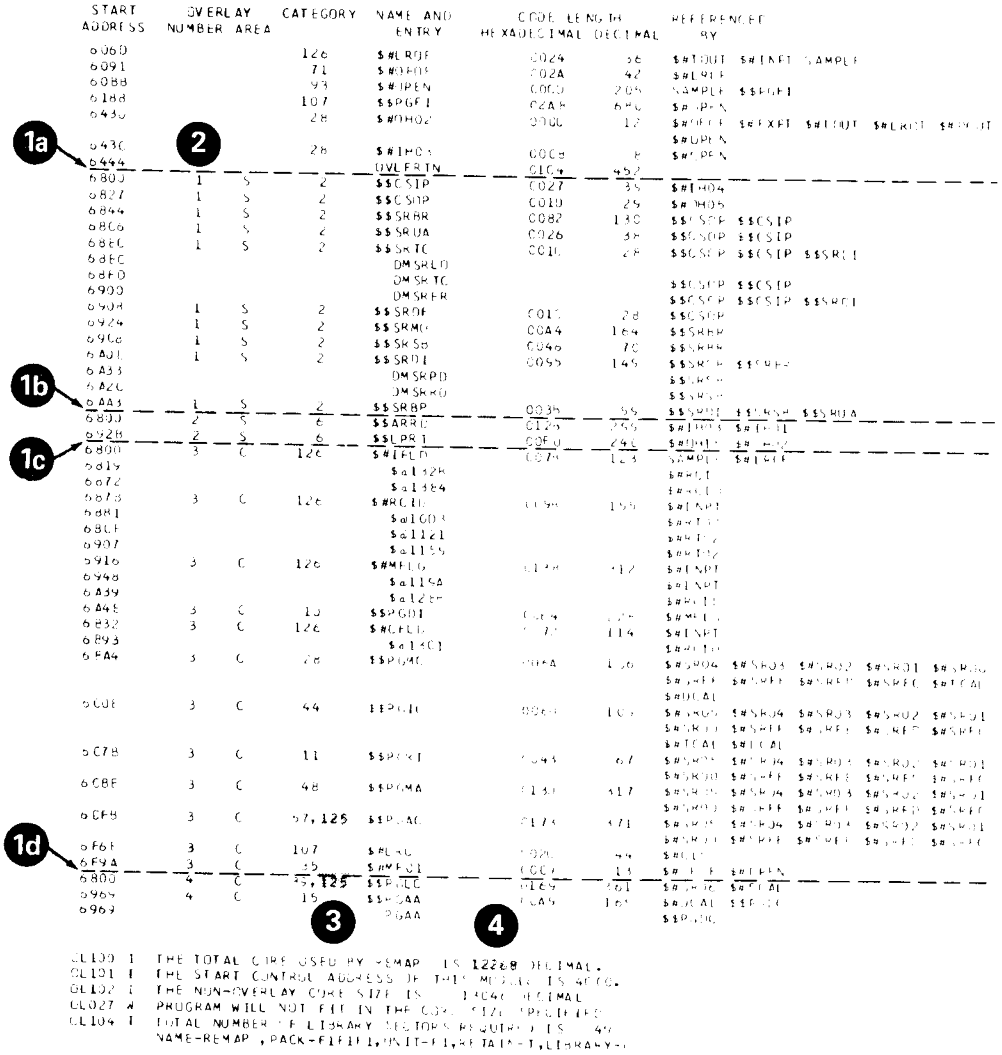
This example shows a link-edit of the same RPG program used for Example 12 requesting the memory resident overlay REMAP technique (ATTR-MRO).

```
// LOAD $CLINK,F1
// HALT
// RUN
// CATEGORY NAME-$$PCLC,$$PGACD,VALUE-125
// OPTIONS ATTR-MRO,MAP-XREF,CORE-10K
// PHASE NAME-REMAP,UNIT-F1,RETAIN-T
// INCLUDE NAME-SAMPLE,UNIT-F1
// END
```

OPTIONS statement request for REMAP technique.

Overlay segments containing these modules will not be candidates for memory residence.

```
               OVERLAY LINKAGE EDITOR CORE USAGE MAP AND CROSS REFERENCE LIST

START      OVERLAY     CATEGORY  NAME AND       CODE LENGTH        REFERENCED
ADDRESS   NUMBER AREA              ENTRY    HEXADECIMAL DECIMAL       BY

4000                            GLOBAL         C5A0     1440
45A0                            COMMON         C578     1400
4C00                        0   SAMPLE         C128      296   $#SR06  $#SR05  $#SR04  $#SR03  $#SR02
                                                               $#SR01  $#SR0U  $#SRFF  $#SRFE  $#SRFD
                                                               $#SRFC  $#MF01  $#CFOF  $#EXPT  $#TOUT
                                                               $#LROT  $#DOUT  $#CLOS  $#OPEN  $#DCAL
                                                               $#CFLD  $#MFLG  $#RCID  $#INPT  $#OPCR
                                                               $#IPCR  $#RT02
4028                        0   $#RT02         0181      433   SAMPLE  $#SR06  $#SR05  $#SR04  $#SR03
                                                               $#SR02  $#SR01  $#SR00  $#SRFF  $#SRFE
                                                               $#SRFD  $#SRFC  $#CH15  $#OFOF  $#EXPT
                                                               $#TOUT  $#LRCF  $#DOUT  $#OPEN  $#DCAL
                                                               $#IFLD  $#CH05  $#CH02
4E09                        0   $#MISC         0018       24   $#RT02  SAMPLE  $#CFLD  $#MFLG
4EE1                        0   $#IPCR         C060       96   $#OPEN  $#INPT
4EF2                              $äCEB9                       $#IH04  $#IH03  $#IH01
4F51                        0   $#OPCR         0080      176   $#LROF  $#TCUT  $#DOUT  $#CLOS  $#OPEN
                                                               $#TCAL  $#LCAL
4F52                              $äCF19                       $#CH15  $#CH05  $#CH02
5001                        0   $#CONC         0013       19   $#DCUT
5014                        0   $#CON1         CC17       23   $#TOUT
5028                        0   $#CON2         C012       18   $#LROT  $#CLCS
5030                        0   $#CON3         C034       52   $#MF01  $#LRCF  $#CFCF  $#EXPT  $#TOUT
                                                               $#LROT  $#DCUT  $#CLOS  $#OPEN  $#TCAL
                                                               $#DCAL
5071                        0   $$PGAB         C071      113   $#INPT  $#CPCR  $#IPCR
50E2                        0   $$PGDC         0457     1111   $#SR06  $#SR05  $#SR04  $#SR03  $#SR02
                                                               $#SR01  $#SP00  $#SRFF  $#SRFE  $#SRFC
                                                               $#SRFC
5534                      126   $#INPT         00D5      213   SAMPLE  $#DCUT
55C1                              $ä1C8D                       $#RCID
5505                              $ä1094                       $#RCID
5602                              $ä10C1                       $#RCID
5585                              $ä1C74                       $#MFLG
5582                              $ä1071                       $#MFLG
55AF                              $ä10EE                       $#MFLG
55C5                              $ä1C85                       $#CFLD
55CA                              $ä1089                       $#CFLD
5608                       28   $#IH01         C008        8   $#INPT
5616                       28   $#IH04         0008        8   $#INPT
561E                      126   $#TCAL         COC7      199   $#INPT  SAMPLE
56E5                       28   $#SRFC         0099      153   $#TCAL
577E                       29   $#SRFD         0099      153   $#TCAL
5817                       30   $#SRFE         0099      153   $#TCAL
5880                       31   $#SRFF         0099      153   $#TCAL
5949                       32   $#SR0C         0099      153   $#TCAL
59E2                       33   $#SR01         0099      153   $#TCAL
5A78                       33   $#SR02         C099      153   $#TCAL
5B14                       33   $#SR03         C099      153   $#TCAL
5BAD                       33   $#SR04         0099      153   $#TCAL
5C46                       33   $#SR05         007F      127   $#TCAL
5CC5                      126   $#DCAL         012E      302   $#IFLD  SAMPLE
5CF3                       33   $#SR06         010E      270   $#TCAL  $#DCAL
5F01                       29   $#EXPT         C034       52   $#SR06  $#SR05  $#SR04  $#SR03  $#SR02
                                                               $#SR01  $#SR00  $#SRFF  $#SRFE  $#SRFD
                                                               $#SRFC
5F35                       28   $#OH15         COCC       12   $#SR06  $#SR05  $#SR04  $#SR03  $#SR02
                                                               $#SR01  $#SR00  $#SRFF  $#SRFE  $#SRFD
                                                               $#SRFC
5F41                       93   $#CLOS         002B       43   SAMPLE
5F48                              $ä16F2                       $#TCAL
5F57                              $ä17C3                       $#TOUT
5F5D                              $ä17C7                       $#LROF
5F6C                      107   $#LROT         C03D       61   $#CLOS
5FA9                      126   $#DOUT         C07B      123   $#OPEN  $#DCAL  SAMPLE
6024                       28   $#OHC5         COOC       12   $#DOUT
6030                      126   $#TOUT         C030       61   $#CLOS  $#TCAL  SAMPLE
```


1a
2
1b
1c
1d
3
4

```
 START     OVERLAY     CATEGORY  NAME AND       CODE LENGTH         REFERENCED
ADDRESS  NUMBER AREA             ENTRY    HEXADECIMAL DECIMAL        BY
 6060                   126     $#LROF        0024       36     $#TOUT  $#INPT  SAMPLE
 6091                    71     $#OFOF        002A       42     $#LROF
 60BB                    93     $#OPEN        00CD      205     SAMPLE  $$PGFI
 618B                   107     $$PGFI        02A8      680     $#OPEN
 6430                    28     $#OHO2        000C       12     $#OFOF  $#EXIT  $#TOUT  $#LROF  $#[illegible]
                                                                $#OPEN
 643C                    28     $#IHO[illegible]  0008    8     $#OPEN
 6444                           OVLFRTN       01C4      452
 6800      1     S        2     $$CSIP        0027       39     $#IHO4
 6827      1     S        2     $$CSOP        001D       29     $#OHO5
 6844      1     S        2     $$SRBR        0082      130     $$CSOP  $$CSIP
 68C6      1     S        2     $$SRUA        0026       38     $$CSOP  $$CSIP
 68EC      1     S        2     $$SRTC        001C       28     $$CSOP  $$CSIP  $$SRCI
 68EC                             DMSRLO
 68FD                             DMSRTC                        $$CSOP  $$CSIP
 6900                             DMSRER                        $$CSOP  $$CSIP  $$SRCI
 6908      1     S        2     $$SROF        001C       28     $$CSOP
 6924      1     S        2     $$SRMO        00A4      164     $$SRBR
 69C8      1     S        2     $$SRSB        0046       70     $$SRBR
 6A0E      1     S        2     $$SROI        0095      149     $$SRCP  $$SRBR
 6A33                             DMSRPD                        $$SRCP
 6A2C                             DMSRRO                        $$SRSP
 6AA3      1     S        2     $$SRBP        0035       55     $$SROI  $$SRSP  $$SRUA
 6800      2     S        6     $$ARRC        0125      293     $#IHO3  $#IHO1
 692B      2     S        6     $$LPRT        00F0      240     $#OHO1  $#OHO2
 6800      3     C      126     $#IFLD        0078      123     SAMPLE  $#IREF
 6819                             $a1328                        $#RCI
 6872                             $a1384                        $#RCI
 6873      3     C      126     $#RCID        0098      155     $#INPT
 6881                             $a1008                        $#RTI
 68CF                             $a1121                        $#RTO2
 6907                             $a1155                        $#RTO2
 6916      3     C      126     $#MFLG        0138      312     $#INPT
 6948                             $a119A                        $#INPT
 6A39                             $a12EF                        $#RCI
 6A4E      3     C       10     $$PGDI        0034       52     $#MFLG
 6B32      3     C      126     $#CFLD        0072      114     $#INPT
 6B93                             $a13C1                        $#RCI
 6FA4      3     C       28     $$PGMC        00EA      1[illegible]6   $#SRO4  $#SRO3  $#SRO2  $#SRO1  $#SROO
                                                                $#SRFF  $#SRFE  $#SREF  $#SREC  $#TCAL
                                                                $#DCAL
 6C0E      3     C       44     $$PGIO        006[illegible]  10[illegible]  $#SRO5  $#SRO4  $#SRO3  $#SRO2  $#SRO1
                                                                $#SROO  $#SRFF  $#SRFE  $#SREF  $#SREE
                                                                $#TCAL  $#DCAL
 6C7B      3     C       11     $$PGXI        0043       67     $#SRO5  $#SRO4  $#SRO3  $#SRO2  $#SRO1
                                                                $#SROO  $#SRFF  $#SRFE  $#SREF  $#SREE
 6CBF      3     C       48     $$PGMA        0130      417     $#SRO5  $#SRO4  $#SRO3  $#SRO2  $#SRO1
                                                                $#SROO  $#SRFF  $#SRFE  $#SREF  $#SREE
 6CF8      3     C   57,125     $$PGAC        0175      471     $#SRO5  $#SRO4  $#SRO3  $#SRO2  $#SRO1
                                                                $#SROO  $#SRFF  $#SRFE  $#SREF  $#SREE
 6F6E      3     C      107     $#LRCC        0020       44     $#CLC
 6F9A      3     C       35     $#MFO1        000D       13     $#[illegible]F  $#IPEN
 6800      4     C   35,125     $$PGLC        0169      361     $#SROO  $#TCAL
 6969      4     C       15     $$PGAA        00A5      165     $#DCAL  $$PGLC
 6969                             PGAA                          $$PGLC

 OL100  I   THE TOTAL CORE USED BY REMAP   IS 12288 DECIMAL.
 OL101  I   THE START CONTROL ADDRESS OF THIS MODULE IS 40C0.
 OL102  I   THE NON-OVERLAY CORE SIZE IS      13C40 DECIMAL
 OL027  W   PROGRAM WILL NOT FIT IN THE CORE SIZE SPECIFIED
 OL104  I   TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS    49
            NAME-REMAP ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-[illegible]
```

1. Overlay segments begin on 2K boundaries (1a to 1d).

2. Four overlay segments in this example.

3. Overlays 3 and 4 are not candidates for memory residence because of the CATEGORY 125 module.

4. REMAP uses more core than MOVE (Example 12).

Appendix A. Messages

There are three classes of messages: informational, warning, and terminal. The informational messages are indicated by an I in print position 7. These messages are printed without halts. Warning messages are indicated by a W in print position 7. A 'P halt with options 0 and 3 is issued with warning messages. Terminal messages have a T in position 7 and are issued with a 'P halt with a 3 option. If two or more warning messages are given, the halt indicates the first message. If a warning message and a terminal message are printed, the halt applies to the terminal error.

MESSAGE			DESCRIPTION
OL016	W	ENTRY POINT NAME ON OPTIONS CARD WAS NOT FOUND.	The label given as the entry point on the OPTIONS statement (ENTRY–label) was not one of the entry points of the object modules. If option 0 is taken, the entry point of the mainline routine is used. The name on the OPTIONS card should match one of the names on the storage map.
OL020	T	TEXT OUT OF SEQUENCE IN MODULE WITH BEGIN ADDRESS nnnn IN OVERLAY nnn.	The object text is out of sequence. An ORG instruction has caused code to overlay other code.
OL021	T	MODULE namexx HAS INVALID ESL nnnnnn.	The object module named has an invalid ESL in an S–type record. If using Basic Assembler, you may have specified an invalid EXTRN subtype. Contact your IBM Field Engineering program support representative.
OL022	T	INVALID RLD IN namexx AT TEXT RECORD ADDRESS nnnn.	The object module named has a bad text record. The error record has the nnnn address in bytes 3 and 4. Contact your IBM Field Engineering program support representative.
OL023	T	UNRESOLVED EXTRN IN MODULE WITH BEGIN ADDRESS nnnn IN OVERLAY nnn.	There is an unresolved EXTRN to an entry point. Probable user error.
OL024	T	EXTERNAL BUFFERS GREATER THAN 64K.	64K is the maximum allowed for external buffers. Reduce buffer size and recompile.
OL025	T	MODULE NAMED namexx NOT FOUND	The name printed in the message was not found by AUTOLINK. If the name begins with $, only the program pack was searched. Otherwise both the user pack (if specified) and the program pack were searched. To correct the error, copy the module to the correct library or change the EXTRN to the correct module.
OL026	T	PROGRAM WILL NOT FIT IN THE MAXIMUM CORE SIZE.	The maximum storage size is 61-1/4K. (64K minus the minimum supervisor) Change the module sizes or categories to allow a different overlay structure.

MESSAGE	DESCRIPTION
OL027 W PROGRAM WILL NOT FIT IN THE CORE SIZE SPECIFIED.	Even with overlays, the program will not fit in storage. The storage size is either the core size specified on the OPTIONS statement (CORE–annK), the core size specified to the compiler, or the default of the current partition size. If more main storage is not available for execution, change the module sizes or categories to allow a different overlay structure. If this message is issued and no overlays are indicated on the storage map, overlay segments were not generated for one or more of the following reasons: 1. Overlays would not have provided a storage advantage. 2. All of the object modules had category 0. 3. Only one overlay segment was available for an overlay area.
OL031 W MODULE NAMED namexx WAS NOT REFERENCED BY AN EXTERN.	A module was read from the system input device (READER) or was referenced on an INCLUDE statement but was not referenced by an EXTRN in the object module. An EXTRN must reference the module name for the Overlay Linkage Editor to determine the program structure.
OL032 W MAINLINE MODULE NAME IN GROUP OR CATEGORY CARD	The mainline module name has no meaning in a GROUP or CATEGORY statement. The name should be removed from the statement.
OL033 W A MODULE IN A GROUP CARD HAS CATEGORY VALUE 0-7	The module with the 0-7 category value is ignored when grouping modules. If the user wants the module in a specific overlay segment, he must supply a CATEGORY statement with a value of 8-126, in addition to the GROUP statement. If no module named in a GROUP statement has a category value of 0-7, this message may result from a module being forced to category 0 by the linkage editor. This could be the mainline module, a non-segment in a segmented COBOL program, a zero-length module, or a module called from a system module (message OL035). Categories 127, 128, and 255 are treated as category 0.
OL034 W MODULE NAME namexx IN CATEGORY OR GROUP CARD NOT REFERENCED BY OBJECT PROGRAM.	The name in the CATEGORY or GROUP statement is not referenced by any of the included or AUTO-LINKED modules. The name should be removed or the correct name determined from the storage map. This message may also appear if a module is named twice in CATEGORY assignments.
OL035 T SYSTEM AREA MODULE namexx CATEGORY n CALLS MODULE namexx.	A module with category 1-7 can call only modules with the same category or category 0. The category of one of the modules must be changed.

MESSAGE		DESCRIPTION
OL036 W	MODULE NAME OR ENTRY POINT namexx HAS A DUPLICATE.	This message can occur for two reasons: 1. If the name is on the core usage map twice, the program contains duplicate entry points or module names. If the duplicate entry points or module names are not referenced, the program can be executed. If the duplicate entry points or module names are referenced, the references may be resolved to the wrong name and the program will not execute correctly. Therefore, the object module should be recreated to eliminate the duplicates. 2. If the name appears only once, the module was included more than once via the SYSIN device or INCLUDE statement. The duplicate modules are dropped. The duplicate deck or INCLUDE card can be removed.
OL038 T	MODULE namexx HAS INVALID ESL NUMBER IN TEXT WITH LOAD ADDRESS nnnn	The object module named has an invalid ESL number in a T–type record. The error record has the nnnn address in bytes 3 and 4. Contact your IBM Field Engineering program support representative.
OL042 T	ENTRY POINT IS NOT RELATIVE ZERO IN A MODULE WITH COMMON.	The entry point must be the first byte of the module because the start control address on the header card is used to indicate the load point of the module. The entry point must be changed by either recreating the module or using the ENTRY parameter on the OPTIONS statement.
OL100 I	THE TOTAL CORE USED BY namexx IS nnnnn DECIMAL.	The module named requires the amount of main storage given by nnnnn to execute.
OL101 I	THE START CONTROL ADDRESS OF THIS MODULE IS xxxx.	The entry point of the root segment is specified by xxxx.
OL102 I	THE NON–OVERLAY CORE SIZE IS nnnnnnnn DECIMAL.	The amount of main storage this program needs to execute without overlays is nnnnnnnn.
OL103 I	TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS nnnn NAME–namexx,PACK–packxx, UNIT–nn,RETAIN–r,LIBRARY–R, CATEGORY–nnn	This message is issued when the compiler entry is used to catalog an object module.
OL104 I	TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS nnnn NAME–namexx,PACK–packxx, UNIT–nn,RETAIN–r,LIBRARY–O	This describes the load module cataloged into the object library.

MESSAGE	DESCRIPTION
OL105 I THE CODE LENGTH OF namexx IS nnnn DECIMAL.	Describes the number of bytes in the R module cataloged to the library or punched. This size does not include: 1. Bytes reserved for COMMON. 2. Bytes bypassed for boundary alignment. 3. Bytes used by routines referenced only by EXTRNs.
OL106 I THE TOTAL CORE USED FOR EXTERNAL BUFFERS BY–namexx IS nnnnn DECIMAL.	Total main storage used for external buffers.
OL107 I THE PARTITION SIZE REQUIRED TO EXECUTE THIS PROGRAM IS xxK.	The partition size is determined by adding the values specified in messages OL100 and OL106 and rounding up to an even multiple of 2K.

Appendix B. Object Modules

The Overlay Linkage Editor accepts object modules for link-editing from either disk or card input. Object modules contain four types of records which must be in this order:

H–type = Header Record (optional)
S–type = External Symbol List (ESL) Records
T–type = Text-Relocation Dictionary (RLD) Records
E–type = End Record

Header Record

A header record is the first record of an object module and contains information describing the module.

External Symbol List (ESL) Records

An S–type record contains up to five ESL fields. ESL fields define areas within an object module and contain external references to other modules. The Overlay Linkage Editor accepts the following types of ESL fields:

- Module name
- Entry point
- External reference

External reference (EXTRN) ESLs are divided into seven subtypes. These subtypes are:

- External reference to a module name (EXTRN subtype 0)
- External reference to an entry point (EXTRN subtype 128)
- Weak external reference to a module name (EXTRN subtype 3)
- Weak external reference to an entry point (EXTRN subtype 131)
- GLOBAL COMMON (EXTRN subtype 4)
- LOCAL COMMON (EXTRN subtype 5)
- Conditional external reference to a module name (EXTRN subtype 6)
- External buffers (EXTRN subtype 7)–Program Number 5704-SC2 only

Module Name

This ESL field provides the symbolic name, start address, length in hexadecimal, and category value of the object module.

Entry Point

This ESL field provides the entry point name in the module and the address of the entry point in hexadecimal.

External Reference (EXTRN subtypes 0 and 128)

This ESL field specifies a symbol that is defined as a module name (subtype 0) or entry point (subtype 128) in another module. The external reference to a module name must be to the cataloged module name. The Overlay Linkage Editor searches the system input device (for user entry) or $WORK file (for compiler entry) to find a module. If the module is not found in the system input device or the $WORK file, AUTOLINK is performed. AUTOLINK means that the Overlay Linkage Editor searches the R. entries in the object libraries to resolve all unresolved external references to module names. External references to entry points are not resolved by AUTOLINK.

External references to module names are resolved in two ways depending on the first character of the module name.

For module names that begin with $, only the program pack directory is searched. A special AUTOLINK is performed for MFCU modules. External references are resolved to the single MFCU module that includes all the functions required by the load module. For example, if one object module has an external reference to a MFCU punch module and another object module has an external reference to a MFCU read module, the single MFCU read-punch module is linked. However, if a specific MFCU module is entered via the user entry (either on an INCLUDE statement or as an R module in the system input device), all other MFCU modules referenced by external references are linked into the load module.

Note: This special AUTOLINK does not apply to MFCM modules.

For any other module name that begins with a valid character, the order of search is:

1. The user pack (if specified)

2. The program pack

If the user pack and the program pack are the same physical pack, the pack is searched only once.

Weak External Reference (EXTRN subtypes 3 and 131)

The function of the weak external reference is the same as for the external reference except no AUTOLINK is performed. If the Overlay Linkage Editor cannot resolve the referenced name, the weak external reference is ignored and remains unresolved.

GLOBAL COMMON Area (EXTRN subtype 4)

This ESL record specifies a space allocation for a GLOBAL COMMON area. This area is allocated at the start of the program level. The size of the area is the size of the largest COMMON area encountered. This area is saved across INVOKE (one FORTRAN program calling another and transferring control to the called program) if the called program contains the GLOBAL COMMON ESL. The Overlay Linkage Editor sets the program common attribute in the load module.

LOCAL COMMON Area (EXTRN subtype 5)

The Overlay Linkage Editor allocates an area of main storage for the COMMON area either at the beginning of the program level or immediately following the storage reserved for the GLOBAL COMMON. This area is used by modules within the same program and is not saved across INVOKE.

Conditional External Reference (EXTRN subtype 6)

The function of the conditional external reference is the same as for the external reference (in other words, AUTOLINK is performed), except that if the Overlay Linkage Editor cannot resolve (find) the referenced name, the conditional external reference is ignored and remains unresolved.

External Buffers (EXTRN subtype 7) – Program Number 5704-SC2 Only

The Overlay Linkage Editor accumulates the length of the required external buffers and places that length in the object library directory entry for the program being cataloged.

Text–Relocation Dictionary (RLD) Records

T–type records contain the object code of modules to be link-edited. T–type records also contain the information needed to make the text relocatable. The load addresses on the text records must be in ascending order, and the text must not overlap from one text record to the next.

Each record is 64 bytes long in the following format:

Byte	*Contents*
0	T (denotes text–RLD record).
1	Length minus 1 of object text contained in the record.
2-3	Address of the rightmost byte of object text in the record.
4-63	Object text begins in byte 4; 1-byte or 3-byte RLD (relocation dictionary) entries are inserted beginning in byte 63 from right to left. Unused bytes (at least one) between text and RLD contain X'00'. RLD points to right end of address (displaced from beginning of text).

One-byte RLD

Each 1-byte RLD entry contains the following:

Bit	*Meaning*
0	0=RLD points to the rightmost byte of an address within this module. 1=RLD points to an EXTRN.
1	0=1-byte RLD.
2-7	Displacement from the leftmost text byte in the record. Displacement count starts with 00.

Three-byte RLD

Three-byte RLDs are generated by the compiler for external references when a displacement from an external symbol is specified in a source statement. These RLDs are re quired to support programs referencing a common data area which may be considered external to all included modules. Each 3-byte RLD contains the following:

Byte	*Bit*	*Meaning*
1-2 (leftmost)	all	Relative EXTRN ESL count so name of the EXTRN can be found. Relative ESL count starts with 0001.
3 (rightmost)	0	1=RLD points to an EXTRN with a known displacement.
	1	1=3-byte RLD.
	2-7	Displacement from the leftmost text byte in the record.

Three-byte RLDs are processed like 1-byte RLDs except that the base address is the address defined in the ESL entry corresponding to the relative EXTRN count.

End Record

An E-type record must be the last record of an object module.

This page intentionally left blank

Appendix C: Performance Improvements

You can reduce the time required to link-edit a program by using one or more of the following procedures:

1. Do not request a cross-reference list. For many programs the time saved may be small, but for programs with many module names and entry points, along with many references to these module names and entry points, significant time saving can result. The time saved is not only the amount of time needed to print the list, but also the additional time needed during the link-edit to save all the information on disk.

2. Locate the work files, $WORK and $SOURCE, optimally.

   a. On a multi-drive system, the program pack and user pack (if one is used) should not be on the same drive as the $WORK and $SOURCE files.

   b. On a single drive system, the work files should be located at the beginning of the library. This can be done by using the Library Maintenance program ALLOCATE statement to first allocate the source library to 30 tracks and then deallocate it to 0 tracks. The 30 tracks are then available at the start of the library and can be used for the work files by specifying the LOCATION parameter on the FILE statements.

   c. On a single drive system, if the work files cannot be located at the beginning of the library (as in step b), they should be located opposite the object library on the other disk drive. For example, if the object library is located at tracks 28 or 37 on F1, the work files should be located at tracks 28 to 37 on R1.

   d. For the Model 15, files should be placed on a 5445 disk drive for best performance.

   e. If steps a, b, c, and/or d are not used to optimally locate the work files, no FILE statements should be provided. The linkage editor finds the work space needed.

3. Load the Overlay Linkage Editor from a program pack containing only the Compiler, the Overlay Linkage Editor, and the R modules to be link-edited. This is more efficient than loading the Overlay Linkage Editor from the system pack.

4. Place the object modules (R modules) that will be used in the link-edit ahead of the load modules in the object library. This would be most effective with step 2b.

5. Have the linkage editor locate the needed object modules via AUTOLINK rather than by you supplying multiple INCLUDE cards.

6. Use the memory resident overlay techniques to increase the performance of large overlay programs. Which technique to use depends on the impact that the 2K boundary restriction has on the program. Because adjusting ATRs takes less time than moving the data (especially for large overlay segments), REMAP object modules generally produce a larger execution storage size and fit fewer segments into the resident area of the partition, but generally execute in less time than MOVE object modules.

This page intentionally left blank

Glossary

The following terms are defined as they are used in this manual. If you do not find the term you are looking for, refer to the index or to the IBM *Data Processing Glossary*, GC20-1699.

AUTOLINK. Process whereby the Overlay Linkage Editor searches the object library for object modules to resolve all unresolved module name external references.

COMMON. An area at the beginning of a partition or following a GLOBAL area.

conditional external reference. An external reference that causes AUTOLINK to be performed. However, if the module named by the conditional external reference is not found, no error message is printed and the conditional external reference is treated as a weak external reference.

descendant. In a caller-called relationship between two modules, the called module is the descendant.

external reference. (1) A reference to a symbol that is defined as an external name in another module. (2) An external symbol that is defined in another module; that which is defined in the assembler language by an EXTRN statement, and is resolved during linkage editing. See also weak external reference.

fetch routine. The routine to find the overlay on disk and load it to storage.

fetch table. The parameter needed to load a single overlay.

GLOBAL. An area of main storage at the beginning of a partition.

INVOKE. Process where one load module calls and transfers control to another load module.

load module. The output of the linkage editor; a program in a format suitable for loading into main storage for execution.

mainline. The first module encountered when link-editing. This module is always in the root segment.

memory-resident overlay. The technique used to increase the performance of large overlay programs by allowing certain overlay segments to remain in primary storage after the initial segment fetch.

MRO. Memory resident overlay.

O module. A load module.

object library. An area on disk that contains load modules and object modules.

object module. A module that is the output of an assembler or compiler and is input to the linkage editor.

object program. A fully compiled or assembled and link-edited program that is ready to be loaded into main storage.

overlay. (1) The technique of repeatedly using the same blocks of internal storage during different stages of a program. When one module is no longer needed in storage, another module can replace all or part of it. (2) A program segment or phase that is loaded into main storage. It replaces all or part of a previously loaded segment.

overlay module. A load module that has been divided into overlay segments, and that has been provided by the Overlay Linkage Editor with information that enables the Overlay Fetch Routine to implement the desired loading of segments when requested.

overlay program. A program in which certain control sections can use the same storage locations at different times during execution.

overlay region. A continuous area of main storage in which segments can be loaded independently of other regions.

overlay segment. See *segment.*

program pack. The disk pack from which the Overlay Linkage Editor program is loaded.

root segment. That segment of an overlay program that remains in main storage at all times during the execution of the overlay program; the first segment in an overlay program. A root segment cannot be overlaid.

R module. An object module.

segment. A part of a computer program divided into parts such that the program can be executed without the entire program being in main storage at any one time.

system pack. The disk pack from which the system is loaded (IPL pack).

transfer vector. The linkage to each entry point in an overlay that allows the overlay to be loaded to storage before control is passed to the entry point.

user pack. A disk pack that contains object modules to be link-edited.

weak external reference. An external reference that does not have to be resolved during linkage editing. If it is not resolved, it appears as though its value was resolved to zero.

Index

GC21-7561-5

IBM System/3 Overlay Linkage Editor (File No. S3-31) Printed in U.S.A. GC21-7561-5

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
Atlanta, Georgia 30301
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)