

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)

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.

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.

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.

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. Requests 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.

© Copyright International Business Machines Corporation 1971, 1972, 1973, 1974, 1974, 1975, 1978

| | | | |
|---|----|---|----|
| INTRODUCTION | 1 | APPENDIX A. MESSAGES | 38 |
| System Configuration | 3 | APPENDIX B. OBJECT MODULES | 43 |
| Primary Storage Requirements | 3 | Header Record | 43 |
| Secondary Storage Requirements | 3 | External Symbol List (ESL) Records | 43 |
| Error Halts | 3 | Text-Relocation Directory (RLD) Records | 44 |
| Changes in Load Module Size | 3 | End Record | 45 |
| USING THE OVERLAY LINKAGE EDITOR | 4 | APPENDIX C. PERFORMANCE IMPROVEMENTS | 47 |
| Compiler Entry | 4 | GLOSSARY | 49 |
| User Entry | 4 | INDEX | 51 |
| OCL Statements | 4 | | |
| Control Statements | 5 | | |
| Parameter Summary | 6 | | |
| Storage Map | 12 | | |
| OVERLAYS | 13 | | |
| Overlay Areas | 13 | | |
| Root Area | 13 | | |
| User Overlay Area | 13 | | |
| System Overlay Area | 14 | | |
| Co-resident Area | 14 | | |
| Assigning Overlays | 14 | | |
| Control Statements and Object Modules | 15 | | |
| Determining Which Modules Can Be Overlaid | 16 | | |
| Link-Edit Start Addresses | 16 | | |
| Load Module Entry Point | 18 | | |
| Overlay Area Size | 18 | | |
| Including Overlay Areas in the Root | 18 | | |
| Using the Group Statement | 18 | | |
| Memory Resident Overlays (Model 15 Only) | 19 | | |
| Supervisor Support for Memory Resident Overlays | 19 | | |
| Using Memory Resident Overlay with Communications | | | |
| Control Programming | 19 | | |
| Memory Resident Overlay Program Execution | 20 | | |
| Execution of the MOVE Technique | 21 | | |
| Execution of the REMAP Technique | 22 | | |
| EXAMPLES | 23 | | |
| Examples 1 through 5 | 23 | | |
| Examples 6 and 7 | 27 | | |
| Example 6 (Figure 12) | 27 | | |
| Example 7 | 29 | | |
| Examples 8 through 11 | 30 | | |
| Example 8 | 31 | | |
| Example 9 | 32 | | |
| Example 10 | 33 | | |
| Example 11 | 34 | | |
| Examples 12 and 13 | 35 | | |
| Example 12 | 35 | | |
| Example 13 | 36 | | |

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).

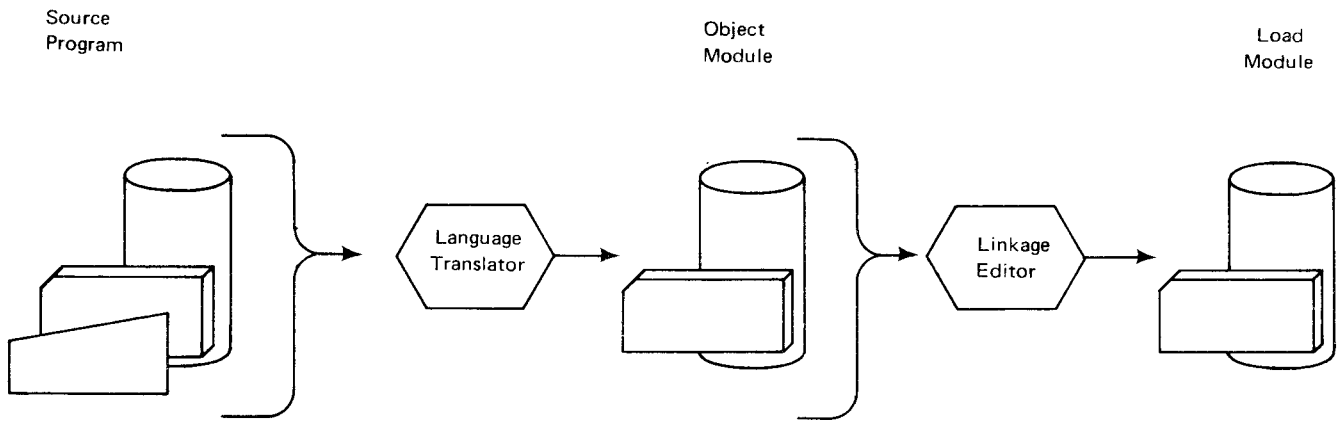


Figure 1. Preparing a Source Program for Execution

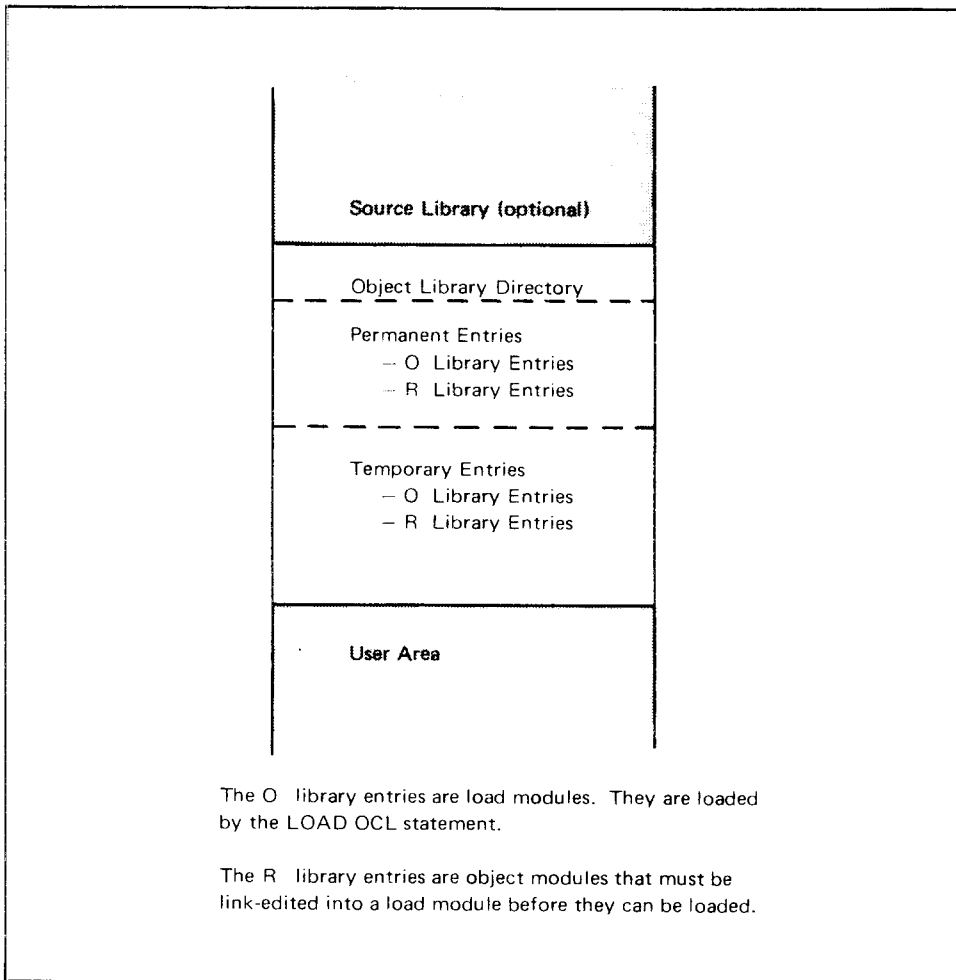


Figure 2. Format of Object Library

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:

| <i>System/3 Model</i> | <i>Main Storage</i> |
|-----------------------|---------------------|
| 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:

| | 5444 | 5445 | 3340 | |
|-----------------|------|------|-------------------------|----------------|
| | | | 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--$SOURCE, . . . } (These two FILE state-
// FILE NAME--$WORK, . . . }  ments are optional and
// RUN                          are standard FILE state-
                                ments used by the
                                compilers.)

Model 6

010 LOAD   NAME--  $OLINK
011        UNIT--  (R1,F1,R2, or F2)
020 FILE   NAME--  $SOURCE
.
.
.
030 FILE   NAME--  $WORK
.
.
.
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- $\left\{ \begin{array}{c} \text{YES} \\ \text{NO} \end{array} \right\}$,RETAIN- $\left\{ \begin{array}{c} \text{T} \\ \text{P} \\ \text{R} \end{array} \right\}$,LINKADD- $\left\{ \begin{array}{c} \text{S} \\ \text{S+X'aaaa'} \\ \text{X'aaaa'} \end{array} \right\}$,RLD- $\left\{ \begin{array}{c} \text{YES} \\ \text{NO} \end{array} \right\}$ |
| Define Environment | // OPTIONS UPACK--code,CORE--annK,LEVEL--nnn,ENTRY--label,ATTR--xxx,MAP- $\left\{ \begin{array}{c} \text{YES} \\ \text{NO} \\ \text{XREF} \\ \text{MSG} \end{array} \right\}$ |
| 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- $\left\{ \begin{array}{c} \text{YES} \\ \text{NO} \end{array} \right\}$ 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-- $\left\{ \begin{array}{c} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$,RETAIN-- $\left\{ \begin{array}{c} \text{T} \\ \text{P} \\ \text{R} \end{array} \right\}$,LINKADD-- $\left\{ \begin{array}{c} \text{S} \\ \text{S+X'aaaa'} \\ \text{X'aaaa'} \end{array} \right\}$,RLD-- $\left\{ \begin{array}{c} \underline{\text{YES}} \\ \text{NO} \end{array} \right\}$ | |
|--|---|
| 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-- $\left\{ \begin{array}{c} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ | 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-- $\left\{ \begin{array}{c} \text{T} \\ \text{P} \\ \text{R} \end{array} \right\}$ | 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-- $\left\{ \begin{array}{c} \text{S} \\ \text{S+X'aaaa'} \\ \text{X'aaaa'} \end{array} \right\}$ | 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: <i>link-edit start addresses</i>). |
| RLD-- $\left\{ \begin{array}{c} \underline{\text{YES}} \\ \text{NO} \end{array} \right\}$ | 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.

| | | | | | | | | | | | |
|--|---|-------------------------------|------------------|---|------------------|-----------------|---|------------------|-------------------|---|------------------|
| <pre>// OPTIONS UPACK--code,CORE--annK,LEVEL--nnn,ENTRY--label,ATTR--xxx,MAP--</pre> | | <pre>YES } NO } XREF }</pre> | | | | | | | | | |
| 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 | <p>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.</p> <p>a = increments of 1/4 K Q = 1/4 or 256 bytes H = 1/2 or 512 bytes T = 3/4 or 768 bytes O = zero bytes</p> <p>nn = 1K increments</p> <p>Example: Q04K = 1/4K + 4K = 256 + 4096 = 4352 bytes</p> | | | | | | | | | | |
| LEVEL--nnn | <p>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:</p> <table border="0"> <tr> <td>5704-RG2, RPG II</td> <td>=</td> <td>253¹</td> </tr> <tr> <td>5704-CB2, COBOL</td> <td>=</td> <td>254¹</td> </tr> <tr> <td>5704-FO2, FORTRAN</td> <td>=</td> <td>255¹</td> </tr> </table> | | 5704-RG2, RPG II | = | 253 ¹ | 5704-CB2, COBOL | = | 254 ¹ | 5704-FO2, FORTRAN | = | 255 ¹ |
| 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. | | | | | | | | | | |

¹If 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-SC1]) 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— { YES
NO
XREF
MSG }

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 Sxxxxx) that are included in the load module built from the compiler-generated object text. Modules for the Binary Synchronous Communications Adapter (\$SBSxx 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 | <p>The new category value:</p> <p>0 = A module with this category value cannot be overlaid. It is always placed in the root segment.</p> <p>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:</p> <p>2 -- Disk I/O 3 -- Tape I/O 4 -- Arithmetic 6 -- Unit Record I/O</p> <p>Category values 1, 5, and 7 have no special meaning but are available for use.</p> <p>8-124 = These category values assign overlay priorities. The lower the number, the less likely that the module will be overlaid.</p> <p>125 = For the Model 15, if ATTR-MRO or ATTR-MOV has been specified, category value 125 allows the module to be overlaid but <i>does not</i> allow that overlay segment to be a candidate for memory residence.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>If an input module contains a category value of 128, the module will 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.</p> |

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.

| | |
|--|---|
| <pre>// EQUATE OLDNAME-- { name 'name,name,... name' } ,NEWNAME-- { name 'name,name,... name' }</pre> | |
| 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.

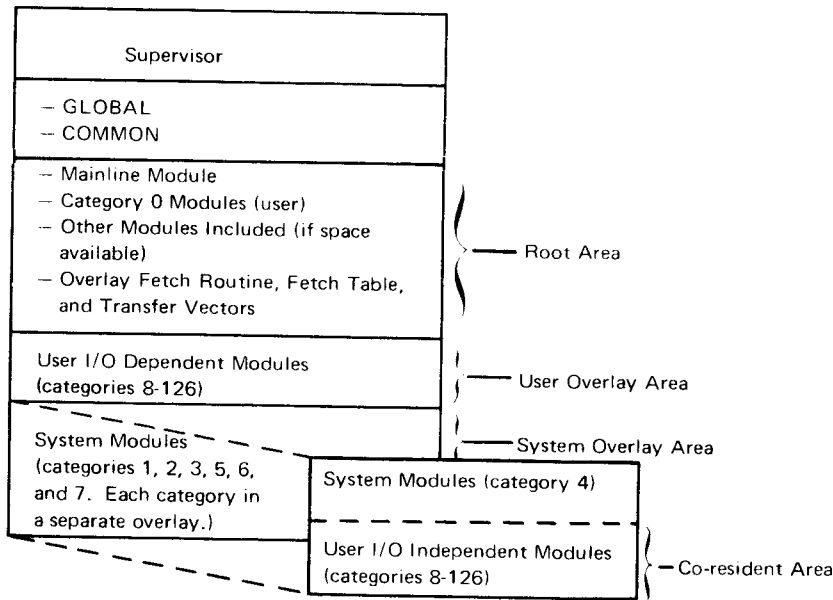


Figure 3. Overlay Areas (Non-DPF System)

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.

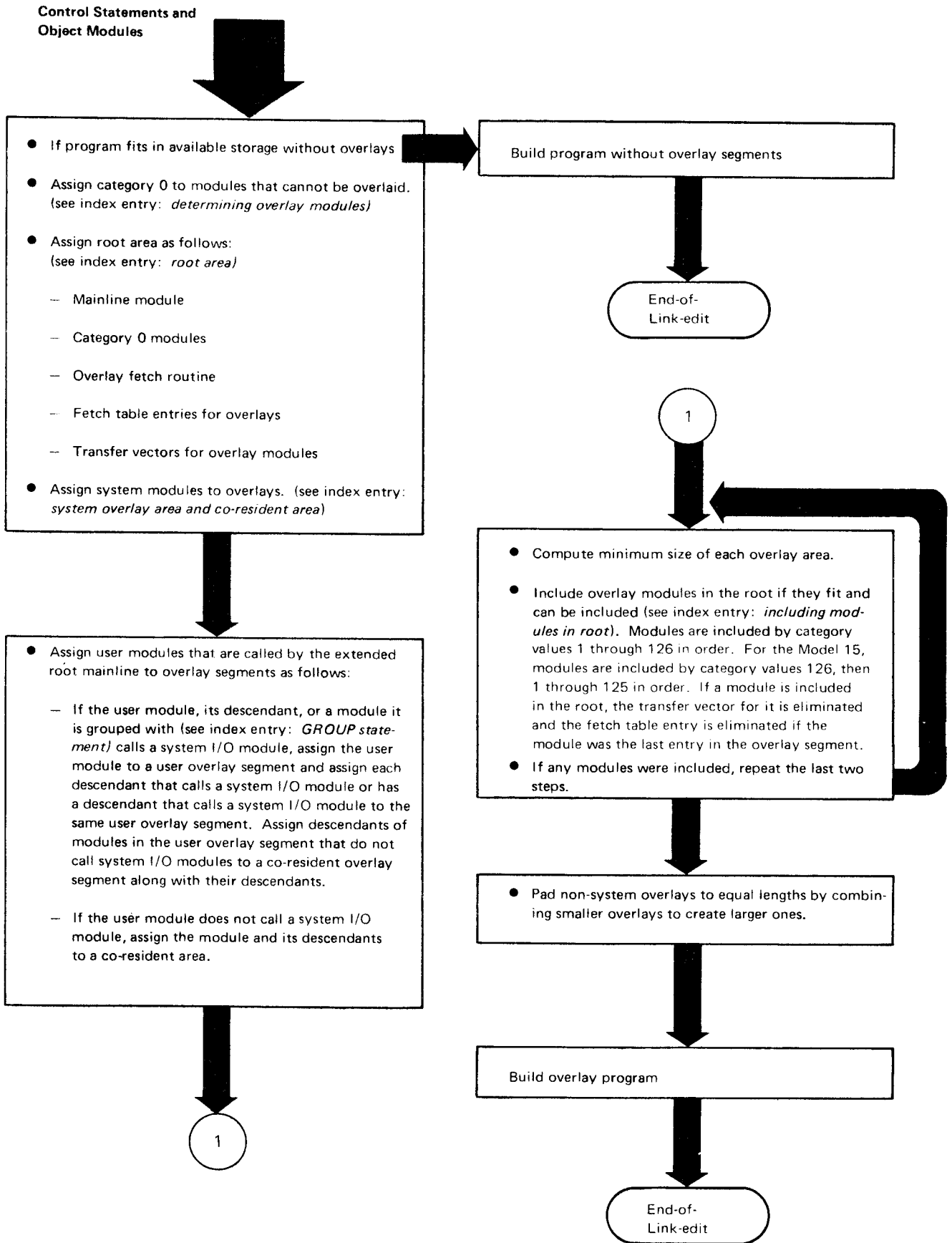


Figure 4. Overview of the Overlay Linkage Editor Method of Assigning Overlays

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. |
| <u>-2600</u> | — | 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.

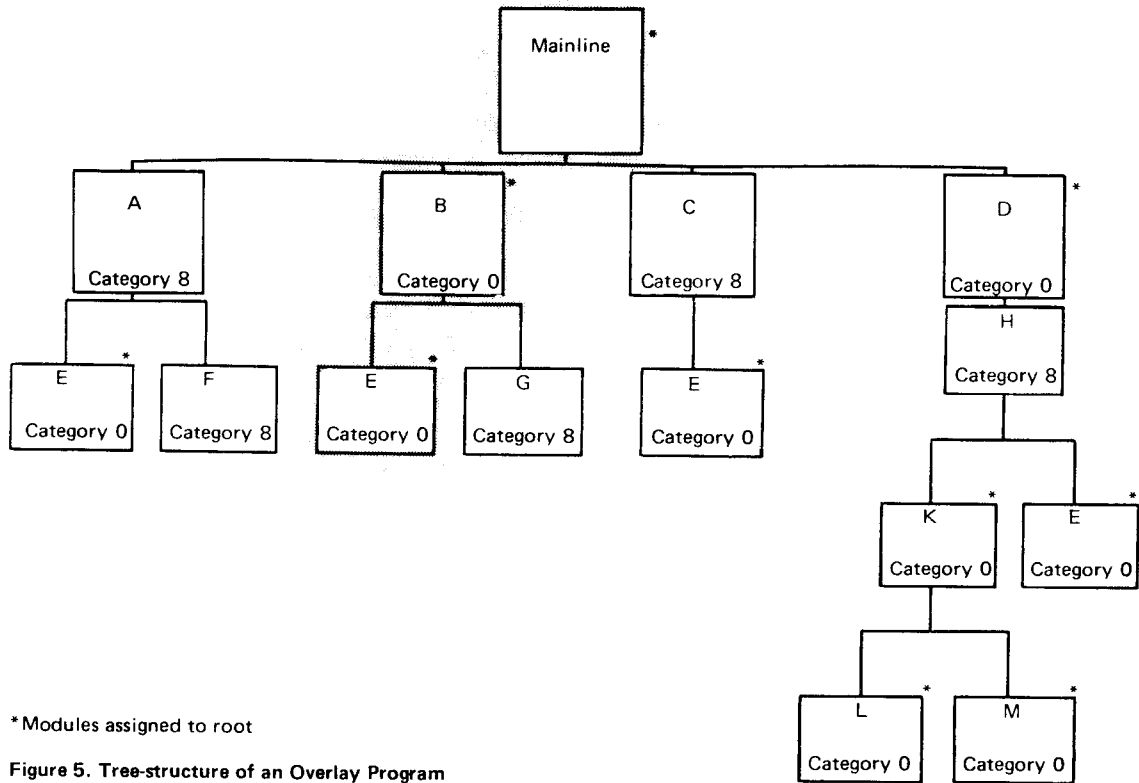


Figure 5. Tree-structure of an Overlay Program

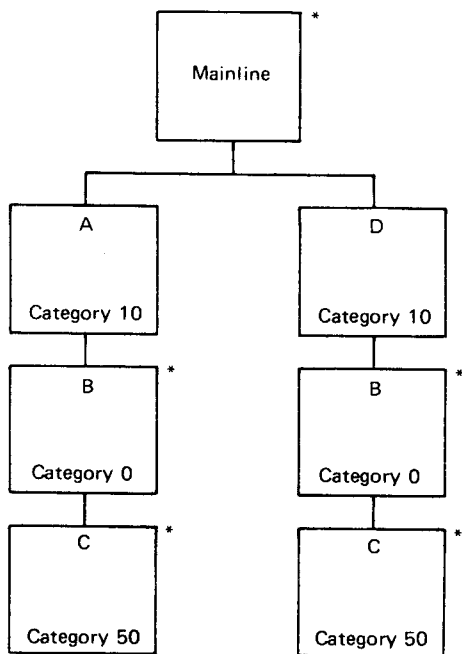


Figure 6. User Modules Assigned to the Root Because they Cannot be Overlaid

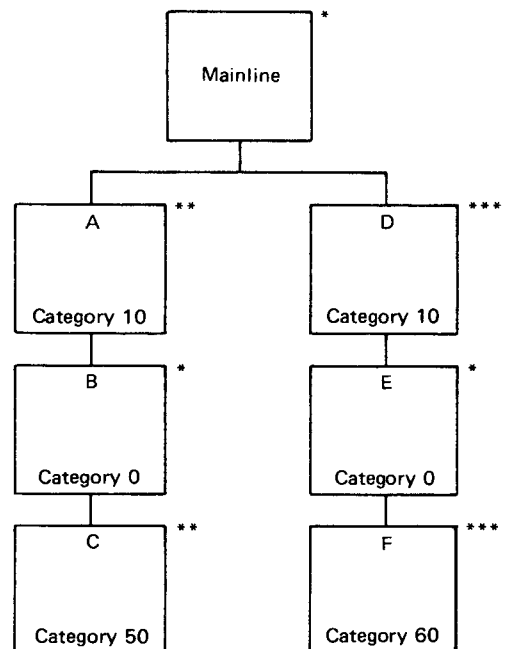


Figure 7. Normal Root Modules Assigned to Overlays

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).

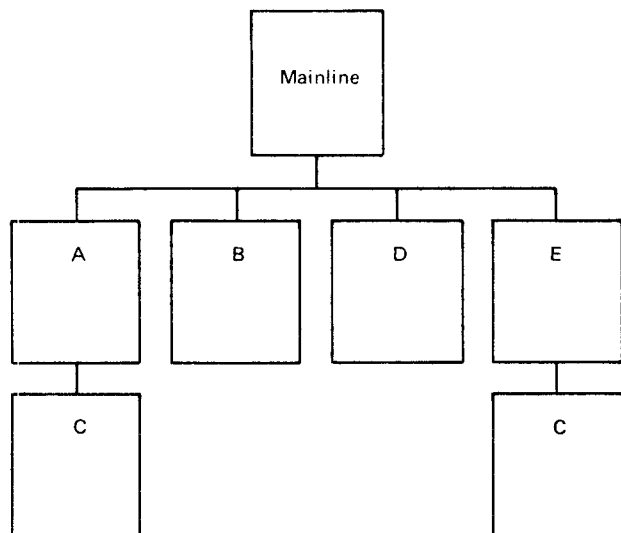


Figure 8. Tree-structure of Sample Program

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 Resident 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.

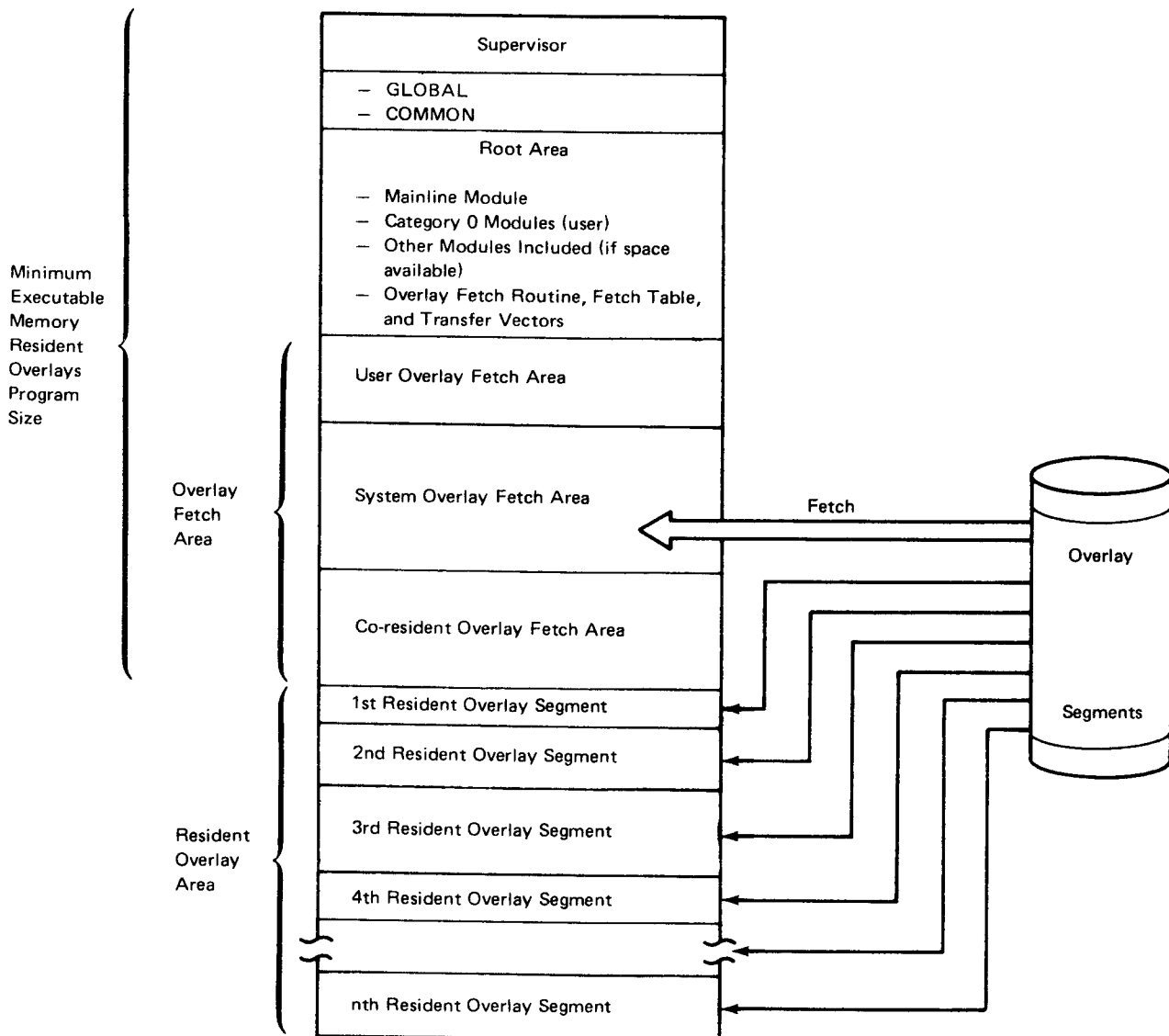


Figure 9. Schematic of the Memory Resident Overlays Program

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.

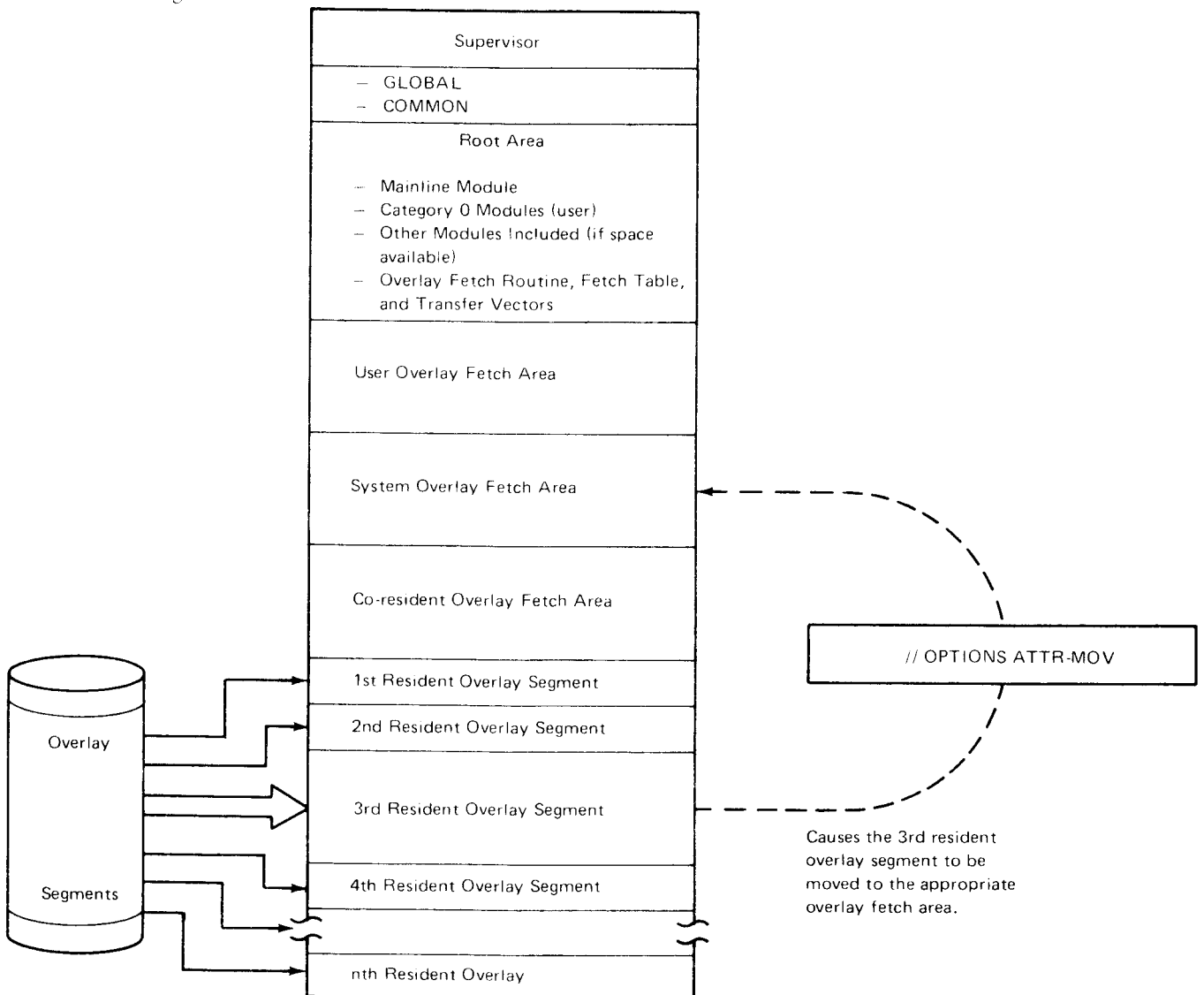


Figure 10. Schematic Execution of the Memory Resident Overlay MOVE Technique

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.

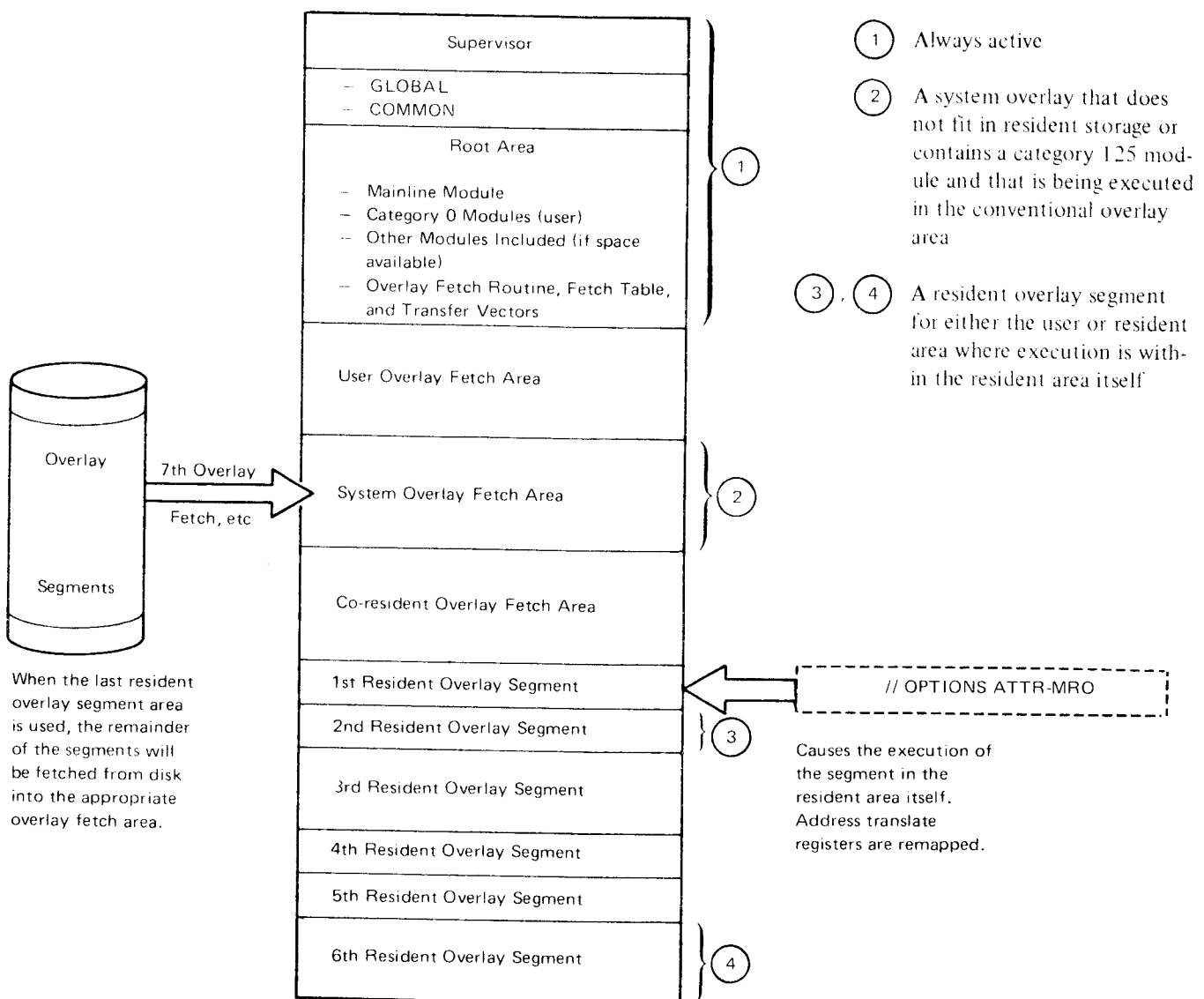


Figure 11. Schematic Execution of the Memory-Resident Overlay REMAP Technique and Overlay Fetch Routine

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

| | |
|---|--|
| 1 | LOAD \$OLINK,F1 |
| 2 | FILE NAME=\$SOURCE,UNIT-F1,PACK-F1F1F1,TRACKS-10,LOCATION-350,RETAIN-S |
| 3 | FILE NAME=\$WORK,UNIT-F1,PACK-F1F1F1,TRACKS-10,LOCATION-360,RETAIN-S |
| 4 | RUN |
| 5 | PHASE NAME-TEST50,UNIT-R1,RETAIN-P,LINKADD-S+X'0200' |
| 6 | OPTIONS UPACK-R1,ATTR-CPR,LEVEL-20 |
| 7 | INCLUDE NAME-TEST40,UNIT-R1 |
| 8 | END |

Notes on Example 1

- | | |
|--|---|
| <p>1 The Overlay Linkage Editor is loaded from the fixed pack on drive I.</p> <p>2 The \$SOURCE file of ten tracks is allocated on the fixed pack on drive I starting at track number 350. This is a scratch file.</p> <p>3 The \$WORK file of ten tracks is allocated on the fixed pack on drive I starting at track number 360. This is a scratch file.</p> <p>4 Execution of \$OLINK (last OCL card) is started.</p> <p>5 The output load module is generated as a permanent entry in the object library of the removable pack on drive I under the name TEST50. The start address is the end of the supervisor plus X'0200'.</p> | <p>6 User modules can be found on drive R1. This program is a checkpoint restart program. The level number is 20.</p> <p>7 The mainline routine for this program is an object module in the object library of the removable pack on drive I under the name TEST40.</p> <p>8 End of the Overlay Linkage Editor input. Automatic overlays are generated if needed</p> |
|--|---|

Example 2

| Line | Code | Text | Remarks |
|------|-------------------|-----------------------------|---------|
| 1 | // LOAD | AUTOLINK, R1 | |
| | // RUN | | |
| 2 | // PHASE NAME- | BLUE | |
| | // INCLUDE NAME- | BLUE, UNIT-F1 | |
| 3 | S Card(s) | } R Module in card form | |
| | T Card(s) | | |
| | E Card | | |
| 4 | // CATEGORY NAME- | WHITE, VALUE-20 | |
| 5 | // 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

| Line | Operator | Statement |
|------|----------------|---|
| 1 | LOAD | \$OLINK, F1 |
| 2 | FILE NAME | \$WORK, UNIT-F1, PACK-F1F1F1, TRACKS-20, LOCATION-310, RETAIN-S |
| 3 | FILE NAME | \$SOURCE, UNIT-R1, PACK-R1R1R1, TRACKS-20, LOCATION-310, RETAIN-S |
| 4 | PHASE NAME | BLACK, UNIT-R2, RETAIN-P |
| 5 | OPTIONS | CORE-H12K, ENTRY-WHITE, MAP-MSG |
| 6 | S Card(s) | R Module in card form |
| 7 | T Card(s) | |
| 8 | E Card | |
| 9 | CATEGORY NAME | 'RED, GREEN', VALUE=55 |
| 10 | CATEGORY NAME | 'YELLOW', VALUE=25 |
| 11 | EQUATE OLDNAME | 'A, B, C', NEWNAME-'X, Y, Z' |
| 12 | EQUATE OLDNAME | 'D, E, F', NEWNAME-'Q, Q, Q' |
| 13 | END | |

Notes on Example 3

- 1 SWORK 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)

| Line | Number | Operator | Operand | Remarks |
|------|--------|---------------|--|---------|
| 1 | | LOAD | \$OLINK,R2 | |
| 2 | | FILE NAME | =BWORK,UNIT-DL,PACK-DLDDL,TRACKS-20,RETAIN-S | |
| 3 | | FILE NAME | =SSOURCE,UNIT-DL,PACK-DLDDL,TRACKS-20,RETAIN-S | |
| 4 | | RUN | | |
| 5 | | INCLUDE NAME | =AAAA,UNIT-R1 | |
| 6 | | GROUP NAME | =AA,BB,CC | |
| 7 | | CATEGORY NAME | =AA,BB,DD,EE,VALUE-30 | |
| 8 | | CATEGORY NAME | =CC,VALUE-10 | |
| 9 | | END | | |

Example 5

| Line | Number | Operator | Operand | Remarks |
|------|--------|--|-----------------|---------|
| 1 | | LOAD | \$OLINK,F1 | |
| 2 | | RUN | | |
| 3 | | PHASE PUNCH | =YES | |
| 4 | | OPTIONS MAP | =XREF | |
| 5 | | GROUP NAME | =PGMA,AREA-USER | |
| 6 | | {S Card(s)} {T Card(s)} } R module in card form {E Card} | | |
| 7 | | END | | |

Notes on Example 4

- 1 SWORK and SSOURCE 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.

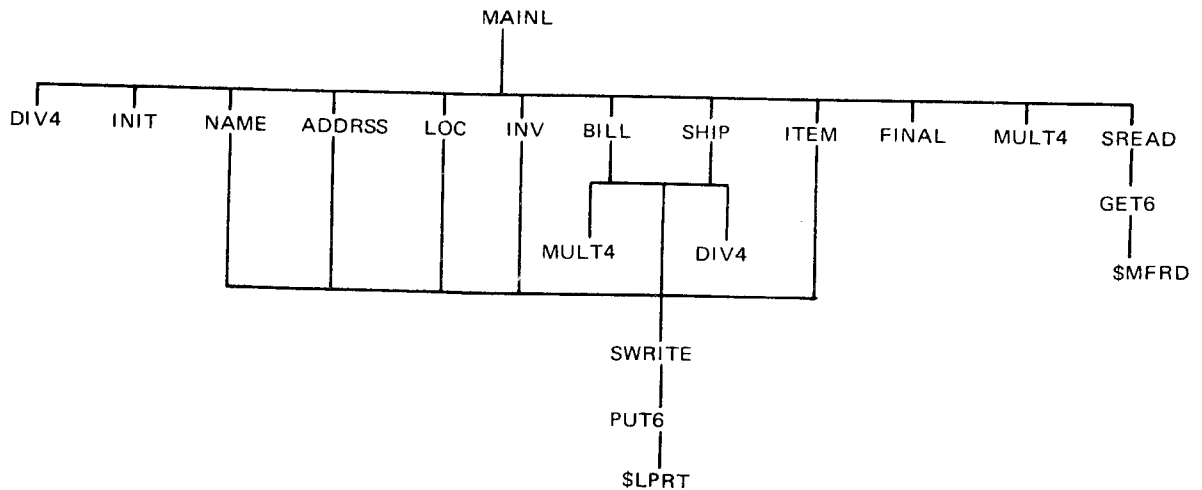
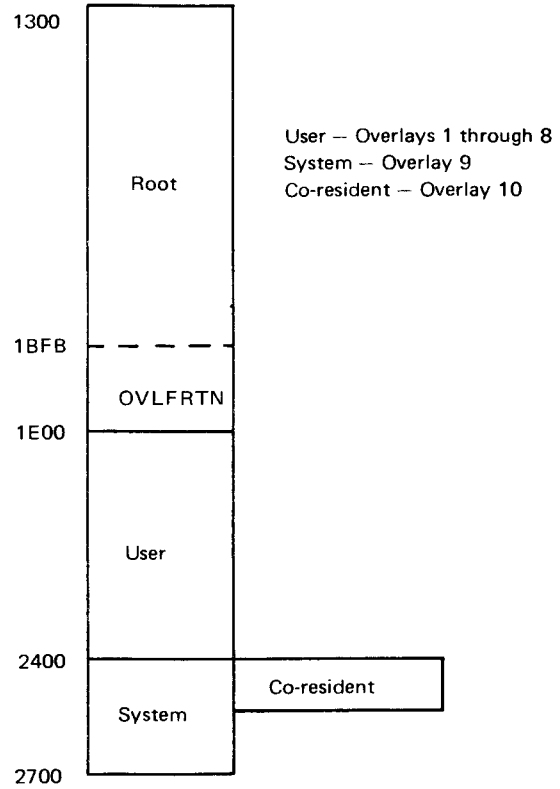


Figure 12. Calling Tree-structure of Modules Linked in Examples 6-7

Example 6

```
// LOAD $OINLK,F1
// FILE NAME=$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

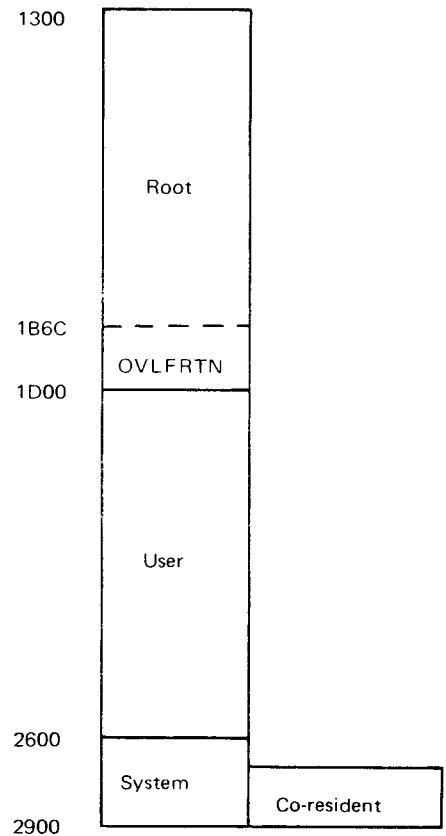
| START ADDRESS | OVERLAY NUMBER | AREA | CATEGORY | NAME AND ENTRY | CODE HEXADECIMAL | LENGTH DECIMAL | REFERENCED BY |
|---------------|----------------|------|----------|----------------|------------------|----------------|------------------------------------|
| 1300 | | | 0 | MAINL | 082D | 2093 | |
| 182D | | | 0,4 | MULT4 | 003F | 63 | MAINL BILL |
| 186C | | | 0,4 | DIV4 | 0041 | 65 | MAINL SHIP |
| 18AD | | | 0,90 | FINAL | 004E | 78 | MAINL |
| 18FB | | | | OVLFRTN | 0180 | 384 | |
| 1F00 | 1 | U | 0,20 | NAME | 0320 | 800 | MAINL |
| 2700 | 1 | U | 0,136 | SWRITE | 0192 | 402 | NAME ADDRSS LOC INV BILL SHIP ITEM |
| 1E00 | 2 | U | 0,21 | ADDRSS | 0302 | 770 | MAINL |
| 2700 | 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 |
| 1F00 | 5 | U | 0,24 | BILL | 02CA | 714 | MAINL |
| 2100 | 5 | U | 0,136 | SWRITE | 0192 | 402 | NAME ADDRSS LOC INV BILL SHIP ITEM |
| 1F00 | 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 |
| 1F00 | 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 | \$SLPRT | 00FB | 251 | PUT6 |
| 2579 | 9 | S | 6 | \$MFRD | 0145 | 325 | GET6 |
| 2400 | 10 | C | 0,90 | INIT | 0000 | 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=0
```


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 $M INK,F1
// FILE NAME=$SOURCE,UNIT=F1,PACK=FF1F1,RETAIN=S,TRACKS=5
// FILE NAME=$WORK,UNIT=F1,PACK=FF1F1,RETAIN=S,TRACKS=5
// ROM
// PHASE NAME=OLSMPL,UNIT=F1
// OPTIONS MAP=XREF,COPI=05*
// CATEGORY NAME=MODULE,VALUE=4
// CATEGORY NAME=GET6,PUT6*,VALUE=5
// CATEGORY NAME=WRITE,VALUE=13
// CATEGORY NAME=SREAD,VALUE=136
// 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 ADDRESS | OVERLAY NUMBER | AREA | CATEGORY | NAME AND ENTRY | CODE HEXADECIMAL | LENGTH DECIMAL | REFERENCED BY |
|---------------|----------------|------|----------|----------------|------------------|----------------|------------------------------------|
| 1300 | | | 0 | MAINL | 0820 | 2093 | |
| 1820 | | | 0,4 | MODULE | 003F | 63 | MAINL BILL |
| 1860 | | | | OVLFRTN | 0160 | 352 | |
| 1000 | 1 | U | 0,20 | NAME | 0320 | 800 | MAINL |
| 2100 | 1 | U | 0,136 | WRITE | 0192 | 402 | NAME ADDRSS LOC INV BILL SHIP ITEM |
| 2292 | 1 | U | 0,22 | LOC | 028A | 650 | MAINL |
| 1000 | 2 | U | 0,21 | ADDRSS | 0302 | 770 | MAINL |
| 2100 | 2 | U | 0,136 | WRITE | 0192 | 402 | NAME ADDRSS LOC INV BILL SHIP ITEM |
| 2292 | 2 | U | 0,23 | INV | 02E4 | 740 | MAINL |
| 1000 | 3 | U | 0,24 | BILL | 02CA | 714 | MAINL |
| 2000 | 3 | U | 0,136 | WRITE | 0192 | 402 | NAME ADDRSS LOC INV BILL SHIP ITEM |
| 2192 | 3 | U | 0,25 | SHIP | 0306 | 774 | MAINL |
| 1000 | 4 | U | 0,26 | ITEM | 0296 | 662 | MAINL |
| 2000 | 4 | U | 0,136 | WRITE | 0192 | 402 | NAME ADDRSS LOC INV BILL SHIP ITEM |
| 2200 | 4 | U | 0,136 | SREAD | 01FA | 506 | MAINL |
| 2600 | 5 | S | 0,4 | GET6 | 0041 | 65 | MAINL SHIP |
| 2600 | 6 | S | 0,6 | PUT6 | 0044 | 68 | SREAD |
| 2644 | 6 | S | 0,6 | WRITE | 003A | 58 | WRITE |
| 2671 | 6 | S | 6 | \$\$\$PRF | 00FB | 251 | PUT6 |
| 2779 | 6 | S | 6 | \$\$\$MERD | 0145 | 325 | GET6 |
| 2700 | 7 | C | 0,90 | INIT | 0000 | 208 | MAINL |
| 2700 | 7 | C | 0,90 | FINAL | 004E | 78 | MAINL |

```
DL100 1 THE TOTAL CORE USED BY OLSMPL IS 5632 DECIMAL
DL101 1 THE START CONTROL ADDRESS OF THIS MODULE IS 1300.
DL102 1 THE NON-OVERLAY CORE SIZE IS 9466 DECIMAL
DL104 1 TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS 51
NAME=OLSMPL,PACK=FF1F1,UNIT=F1,RETAIN=T,LIBRARY=0
```

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.

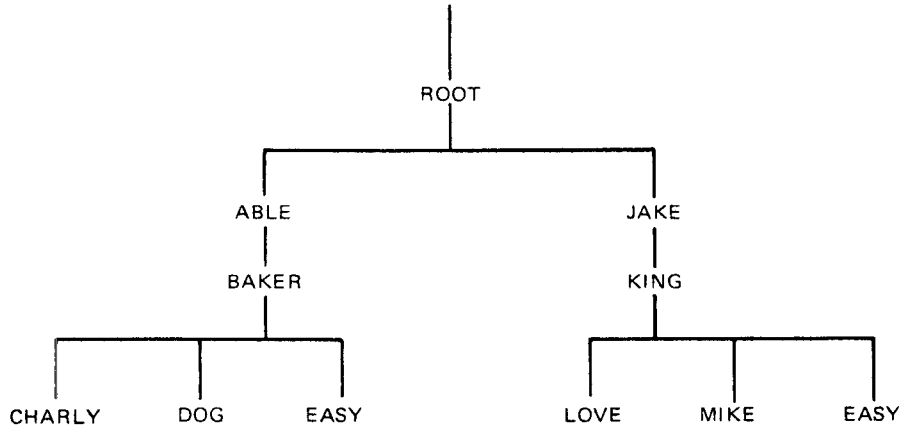
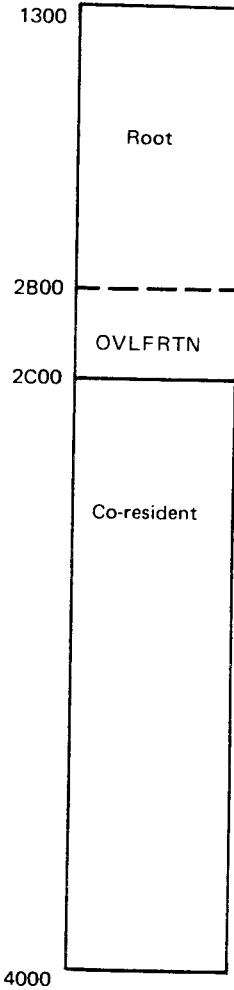


Figure 13. Calling Tree-structure of Modules Linked in Examples 8-11

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-B
// END
```



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

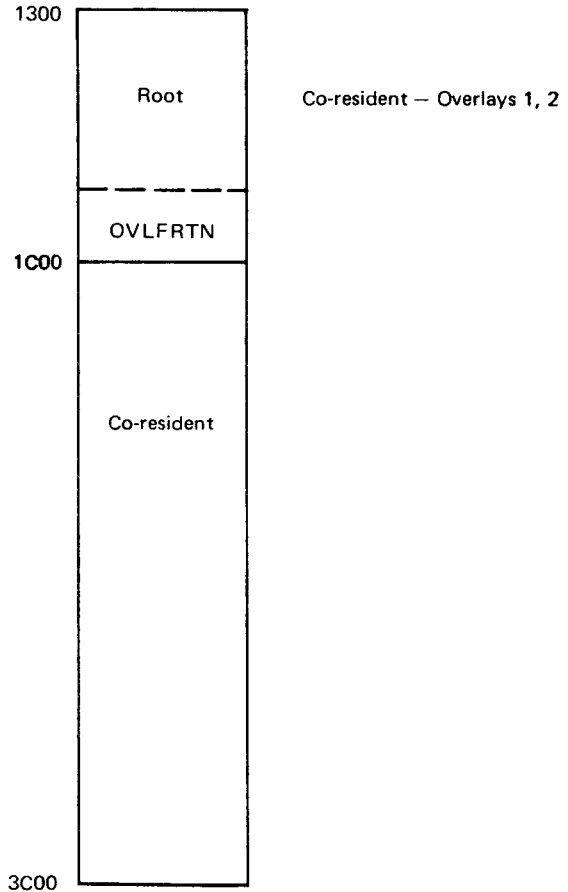
| START ADDRESS | OVERLAY NUMBER | AREA | CATEGORY | NAME AND ENTRY | CODE LENGTH HEXADECIMAL | CODE LENGTH DECIMAL | REFERENCED BY |
|---------------|----------------|------|----------|----------------|-------------------------|---------------------|---------------|
| 1300 | | | 0 | ROOT | 0800 | 2048 | |
| 130A | | | | ROOT | | | |
| 1800 | | | 0 | BAKER | 0400 | 1024 | ABLE |
| 1805 | | | | BAKER | | | ABLE |
| 1F00 | | | 0 | KING | 0400 | 1024 | JAKE |
| 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 |

```
DL100 I THE TOTAL CORE USED BY RIZZI IS 11520 DECIMAL
DL101 I THE START CONTROL ADDRESS OF THIS MODULE IS 130A.
DL102 I THE NON-OVERLAY CORE SIZE IS 16168 DECIMAL
DL104 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 NUHALT
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=HIGH
// 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

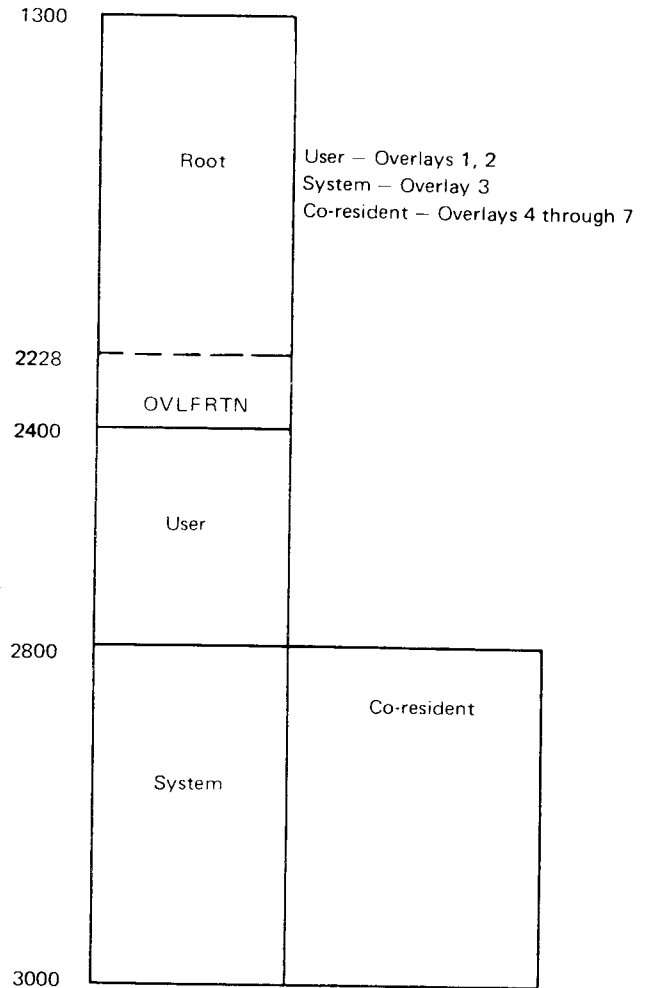
| START ADDRESS | OVERLAY NUMBER | AREA | CATEGORY | NAME AND ENTRY | CODE LENGTH HEXADECIMAL | CODE LENGTH DECIMAL | REFERENCED BY |
|---------------|----------------|------|----------|----------------|-------------------------|---------------------|---------------|
| 1300 | | | 0 | ROOT | 0800 | 2048 | |
| 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 |
| 2128 | 1 | C | 37 | CHARLY | 0800 | 2048 | BAKER |
| 2334 | | | | CHARLY | | | BAKER |
| 2B28 | 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-D
```

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 MFC02
XX LOAD $OLINK,F1
XX FILE NAME=$SOURCE,RETAIN=S,TRACKS=25,PACK=FI F1F1,UNIT=F1
XX FILE NAME=$WORK,RETAIN=S,TRACKS=25,PACK=FI F1F1,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=3
// CATEGORY NAME=EASY,VALUE=2
// END
```



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

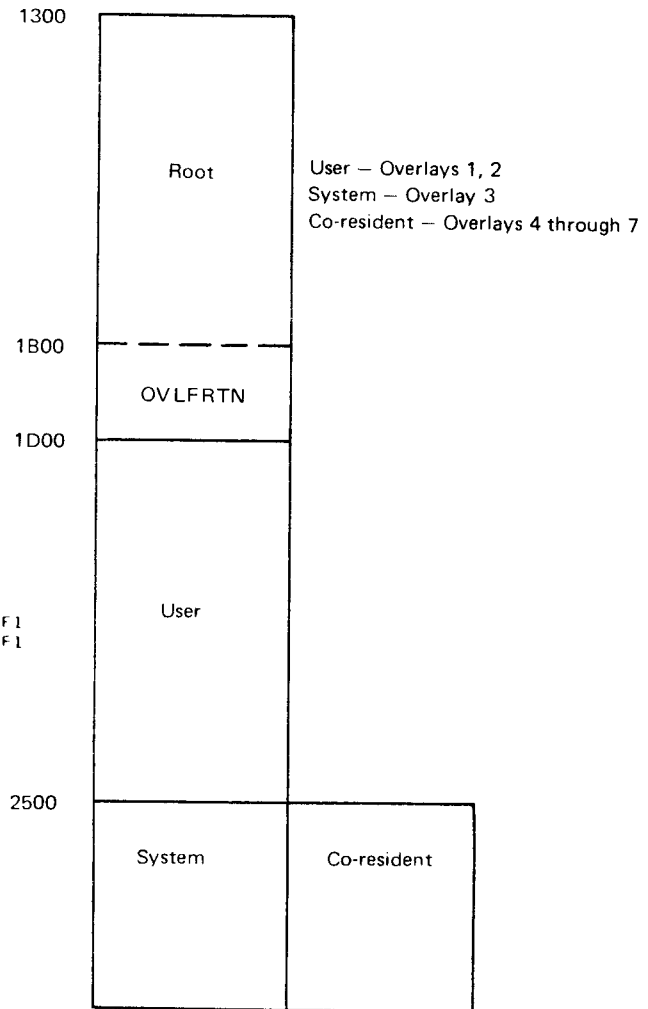
| START ADDRESS | OVERLAY | | CATEGORY | NAME AND ENTRY | CODE LENGTH | | REFERENCED BY |
|---------------|---------|------|----------|----------------|-------------|---------|---------------|
| | NUMBER | AREA | | | HEXADECIMAL | DECIMAL | |
| 1300 | | | 0 | ROOT | 0300 | 2048 | |
| 130A | | | | ROOT | | | |
| 1800 | | | 0 | ABLE | 0328 | 408 | 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 |
| 2508 | | | | 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=FI F1F1,UNIT=F1,RETAIN=T,LIBRARY=0
```

Example 11

All modules except ROOT have an overlay category (non-zero). 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 DLINK,R1
XX LOG PRINTER
XX NOHALT
XX PUNCH MF02
XX LOAD DLINK,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 UPACK-R1,MAP-XREF,CORE-T6K
// INCLUDE NAME=ROOT,UNIT-R1
// CATEGORY NAME='ABLE,BAKER,JAKE,KING',VALUE-R
// CATEGORY NAME=EASY,VALUE-2
// END
```



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

| START ADDRESS | OVERLAY NUMBER | AREA | CATEGORY | NAME AND ENTRY | CODE LENGTH HEXADECIMAL | DECIMAL | REFERENCED BY |
|---------------|----------------|------|----------|----------------|-------------------------|---------|---------------|
| 1300 | | | 0 | ROOT | 0800 | 2048 | |
| 130A | | | | ROOT | | | |
| 1800 | | | | OVLFRN | 0108 | 264 | |
| 1900 | 1 | U | 0,8 | ABLE | 0328 | 808 | ROOT |
| 200A | | | | ABLE | | | ROOT |
| 2028 | 1 | U | 0,8 | BAKER | 0400 | 1024 | ABLE |
| 202D | | | | BAKER | | | ABLE |
| 1000 | 2 | U | 0,8 | JAKE | 0400 | 1024 | ROOT |
| 100A | | | | 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 |

```
DL100 I THE TOTAL CORE USED BY EV28 IS 6656 DECIMAL
DL101 I THE START CONTROL ADDRESS OF THIS MODULE IS 130A.
DL102 I THE NON-OVERLAY CORE SIZE IS 14168 DECIMAL
DL104 I TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS 72
NAME=EV28 ,PACK-F1F1F1,UNIT-F1,RETAIN-T,LIBRARY-0
```

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.

```
// LOAD $OLINK,F1
// VCHALT SEVERITY=4
// PHASE NAME=MOV,RETAIN=1,RELOC=
// OPTIONS PACK=10K,MAP=YES,ATTR=MOV
// INCLUDE NAME=SAMPLE
// CATEGORY NAME=$#DPPN,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 ADDRESS | OVERLAY NUMBER | CATEGORY AREA | NAME AND ENTRY | CODE LENGTH | |
|---------------|----------------|---------------|----------------|--------------|---------|
| | | | | HEXADECIIMAL | DECIMAL |
| 4000 | | | GLOBAL | 05A0 | 1440 |
| 45A0 | | | COMMON | C578 | 1400 |
| 4C00 | | | SAMPLE | C128 | 256 |
| 4D28 | 0 | | \$#RTO2 | C181 | 433 |
| 4E09 | 0 | | \$#MISC | 0018 | 24 |
| 4F1 | 0 | | \$#IPCR | 0060 | 96 |
| 4FF2 | | | \$#OEB9 | | |
| 4F51 | 0 | | \$#OPCR | 00B0 | 176 |
| 4F52 | | | \$#OF19 | | |
| 5001 | 0 | | \$#CON0 | 0013 | 19 |
| 5014 | 0 | | \$#CON1 | 0017 | 23 |
| 5028 | 0 | | \$#CON2 | 0017 | 18 |
| 503D | 0 | | \$#CON3 | 0034 | 52 |
| 5071 | 0 | | \$#PGAR | 0071 | 113 |
| 50E2 | 0 | | \$#PGOC | C457 | 1111 |
| 5339 | 2 | | \$#SRBK | 0087 | 130 |
| 538B | 2 | | \$#SRJA | 0026 | 38 |
| 55E1 | 2 | | \$#SRTE | 001C | 28 |
| 55E1 | | | DM\$R1: | | |
| 55F2 | | | DM\$R1C | | |
| 55F5 | | | DM\$R1R | | |
| 55FD | 2 | | \$#SRUF | 001C | 28 |
| 5619 | 2 | | \$#SRSP | 0046 | 70 |
| 565F | 2 | | \$#SR4P | 0034 | 56 |
| 569A | 126 | | \$#INPT | 0069 | 213 |
| 572F | | | \$#10B0 | | |
| 5736 | | | \$#1094 | | |
| 5763 | | | \$#10C1 | | |
| 5716 | | | \$#1074 | | |
| 5713 | | | \$#1071 | | |
| 5710 | | | \$#106F | | |
| 5727 | | | \$#1065 | | |
| 5728 | | | \$#1064 | | |
| 576F | 28 | | \$#TH01 | 0008 | 8 |
| 5777 | 28 | | \$#TH04 | 0005 | 5 |
| 577F | 29 | | \$#EXPT | 0034 | 50 |
| 5783 | 28 | | \$#WH15 | 002F | 10 |
| 578F | 28 | | \$#WH02 | 0001 | 10 |
| 57C8 | 126 | | \$#IFLD | 0075 | 123 |
| 57E4 | | | \$#132B | | |
| 583D | | | \$#1384 | | |
| 5846 | 93 | | \$#GL95 | 0024 | 4 |
| 584D | | | \$#16F2 | | |
| 585E | | | \$#17C3 | | |
| 5862 | | | \$#17C7 | | |
| 5871 | 126 | | \$#DOUT | 0074 | 124 |
| 58EC | 28 | | \$#WH05 | 0001 | 10 |
| 58F8 | 126 | | \$#TOUT | 0037 | 61 |
| 5935 | 126 | | \$#LROF | 0024 | 36 |
| 5959 | 71 | | \$#OF0F | 002A | 42 |
| 5983 | 126 | | \$#RC10 | 0058 | 118 |
| 5989 | | | \$#1003 | | |
| 59D7 | | | \$#1121 | | |
| 5A0F | | | \$#1154 | | |

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.

| | | | | | |
|------|---|---|---------|------|-----|
| 5411 | | | \$#PLU1 | 0012 | 114 |
| 547F | | | \$#1301 | | |
| 5493 | | | \$#M31 | 0009 | 18 |
| 5496 | | | \$#1004 | 0008 | 8 |
| 54A0 | | | OVERLAY | 0214 | 530 |
| 5507 | 1 | 0 | \$#TCAL | 0007 | 155 |
| 5507 | 1 | 0 | \$#SAFE | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR01 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SAFE | 0059 | 153 |
| 5507 | 1 | 0 | \$#SAFE | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR05 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR01 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR02 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR07 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR04 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR06 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR06 | 0059 | 153 |
| 5507 | 1 | 0 | \$#SR11 | 0059 | 153 |
| 5507 | 2 | 0 | \$#TCAL | 0120 | 307 |
| 5507 | 2 | 0 | \$#SR06 | 0100 | 270 |
| 5507 | 2 | 0 | \$#SR04 | 0009 | 208 |
| 5507 | 2 | 0 | \$#SAFE | 0009 | 660 |
| 5507 | 3 | 0 | \$#CONP | 0077 | 76 |
| 5507 | 3 | 0 | \$#SAFE | 0013 | 29 |
| 5507 | 3 | 0 | \$#SRML | 0004 | 164 |
| 5507 | 3 | 0 | \$#SAFE | 0009 | 195 |
| 5507 | | | DM\$R07 | | |
| 5507 | 4 | 0 | \$#APMS | 0124 | 259 |
| 5507 | 4 | 0 | \$#P04T | 0009 | 240 |
| 5507 | 5 | 0 | \$#M116 | 0136 | 312 |
| 5507 | | | \$#1114 | | |
| 5507 | | | \$#1114 | | |
| 5507 | 5 | 0 | \$#R01 | 0014 | 229 |
| 5507 | 5 | 0 | \$#R05 | 006A | 106 |
| 5507 | 5 | 0 | \$#R01 | 0061 | 165 |
| 5507 | 5 | 0 | \$#R01 | 0043 | 87 |
| 5507 | 6 | 0 | \$#R04A | 0131 | 317 |
| 5507 | 6 | 0 | \$#R0AC | 0123 | 321 |
| 5507 | 7 | 0 | \$#R000 | 0170 | 421 |
| 5507 | 7 | 0 | \$#R0C | 0021 | 44 |
| 5507 | 7 | 0 | \$#R0AA | 0049 | 164 |
| 5507 | | | DATA | | |

0117 I THE TOTAL CORE USED BY MOV IS 10240 DECIMAL.
0101 I THE START CONTROL ADDRESS OF THIS MODULE IS 4000.
0102 I THE NUMBER OF CORE SECTORS IS 10046 DECIMAL.
0114 I TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS 45.
NAME=MOVE,PACK=FIELD,UNIT=FIELD,RETAIN=1,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 $SPGLC, $SPGACD, 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 ADDRESS | OVERLAY NUMBER | OVERLAY AREA | CATEGORY | NAME AND ENTRY | CORE HEXADECIMAL | LENGTH DECIMAL | REFERENCED BY |
|---------------|----------------|--------------|----------|----------------------|------------------|----------------|--|
| 4000 | | | 0 | GLOBAL COMMON SAMPLE | C5A0 | 1440 | |
| 4000 | | | 0 | | C570 | 1400 | |
| 4000 | | | 0 | | C128 | 296 | \$#SRO6 \$#SRO5 \$#SRO4 \$#SRO3 \$#SRO2 \$#SRO1 \$#SRO0 \$#SRFF \$#SRFE \$#SRFD \$#SRFC \$#SRO1 \$#CFDF \$#EXPT \$#TDUT \$#LRPT \$#TDUT \$#CLCS \$#OPEN \$#DCAL \$#CFLD \$#MFLG \$#CFCD \$#INPT \$#DPCR \$#IPCR \$#RTO2 |
| 4028 | | | 0 | \$#RTO2 | 01B1 | 433 | SAMPLE \$#SRO6 \$#SRO5 \$#SRO4 \$#SRO3 \$#SRO2 \$#SRO1 \$#SRO0 \$#SRFF \$#SRFE \$#SRFD \$#SRFC \$#CH15 \$#DIBF \$#EXPT \$#TDUT \$#LRFF \$#TDUT \$#OPEN \$#DCAL \$#IPLD \$#CH05 \$#CH02 |
| 4089 | | | 0 | \$#MISC | 0018 | 24 | \$#RTO2 SAMPLE \$#MFLG |
| 40F1 | | | 0 | \$#IPCR | C060 | 96 | \$#OPEN \$#IPLD \$#CFLD |
| 40F2 | | | 0 | \$#CH09 | | | \$#IH04 \$#IH03 \$#IH01 |
| 40F3 | | | 0 | \$#DPCR | 00B0 | 176 | \$#LROF \$#TDUT \$#TDUT \$#CLCS \$#OPEN \$#TCAL \$#LCLAL \$#CH15 \$#CH05 \$#CH02 |
| 40F4 | | | 0 | \$#CH19 | | | \$#TDUT |
| 40F5 | | | 0 | \$#CON0 | 0013 | 16 | |
| 40F6 | | | 0 | \$#CON1 | 0017 | 23 | \$#TDUT |
| 40F7 | | | 0 | \$#CON2 | 0012 | 18 | \$#LROF \$#CLCS |
| 40F8 | | | 0 | \$#CON3 | 0034 | 52 | \$#MFO1 \$#LRCF \$#CFDF \$#EXPT \$#TDUT \$#LROF \$#DCUT \$#CLCS \$#OPEN \$#TCAL \$#DCAL |
| 40F9 | | | 0 | \$#PGAB | 0071 | 112 | \$#INPT \$#DPCR \$#IPCR |
| 40FA | | | 0 | \$#PGDC | 0457 | 1111 | \$#SRO6 \$#SRO5 \$#SRO4 \$#SRO3 \$#SRO2 \$#SRO1 \$#SRO0 \$#SRFF \$#SRFE \$#SRFD \$#SRFC |
| 40FB | 126 | | 0 | \$#INPT | 0005 | 213 | SAMPLE \$#DCUT |
| 40FC | | | | \$#1080 | | | \$#RCID |
| 40FD | | | | \$#1094 | | | \$#RCID |
| 40FE | | | | \$#10C1 | | | \$#RCID |
| 40FF | | | | \$#1074 | | | \$#MFLG |
| 4100 | | | | \$#1071 | | | \$#MFLG |
| 4101 | | | | \$#107E | | | \$#MFLG |
| 4102 | | | | \$#1085 | | | \$#CFLD |
| 4103 | | | | \$#1089 | | | \$#CFLD |
| 4104 | 28 | | | \$#IH01 | 0008 | 8 | \$#INPT |
| 4105 | 28 | | | \$#IH04 | 0008 | 8 | \$#INPT |
| 4106 | 126 | | | \$#TCAL | 00C7 | 199 | \$#INPT SAMPLE |
| 4107 | 28 | | | \$#SRFC | 0099 | 153 | \$#TCAL |
| 4108 | 29 | | | \$#SRFD | 0099 | 153 | \$#TCAL |
| 4109 | 30 | | | \$#SRFE | 0099 | 153 | \$#TCAL |
| 410A | 31 | | | \$#SRFF | 0099 | 153 | \$#TCAL |
| 410B | 32 | | | \$#SRO0 | 0099 | 153 | \$#TCAL |
| 410C | 33 | | | \$#SRO1 | 0099 | 153 | \$#TCAL |
| 410D | 33 | | | \$#SRO2 | 0099 | 153 | \$#TCAL |
| 410E | 33 | | | \$#SRO3 | 0099 | 153 | \$#TCAL |
| 410F | 33 | | | \$#SRO4 | 0099 | 153 | \$#TCAL |
| 4110 | 33 | | | \$#SRO5 | 0099 | 153 | \$#TCAL |
| 4111 | 33 | | | \$#SRO6 | 0099 | 153 | \$#TCAL |
| 4112 | 126 | | | \$#DCAL | 012E | 302 | \$#IFLD SAMPLE |
| 4113 | 33 | | | \$#SRO6 | 010E | 270 | \$#TCAL \$#CCAL |
| 4114 | 29 | | | \$#EXPT | 0034 | 52 | \$#SRO6 \$#SRO5 \$#SRO4 \$#SRO3 \$#SRO2 \$#SRO1 \$#SRO0 \$#SRFF \$#SRFE \$#SRFD \$#SRFC |
| 4115 | 28 | | | \$#CH15 | 000C | 12 | \$#SRO6 \$#SRO5 \$#SRO4 \$#SRO3 \$#SRO2 \$#SRO1 \$#SRO0 \$#SRFF \$#SRFE \$#SRFD \$#SRFC |
| 4116 | 43 | | | \$#CLOS | 002B | 43 | SAMPLE |
| 4117 | | | | \$#16F2 | | | \$#TCAL |
| 4118 | | | | \$#17C3 | | | \$#TDUT |
| 4119 | | | | \$#17C7 | | | \$#LROF |
| 411A | 107 | | | \$#LROF | 003D | 61 | \$#CLOS |
| 411B | 176 | | | \$#DOUT | 007B | 123 | \$#OPEN \$#DCAL SAMPLE |
| 411C | 28 | | | \$#CH05 | 000C | 12 | \$#DOUT |
| 411D | 126 | | | \$#TDUT | 0030 | 61 | \$#CLOS \$#TCAL SAMPLE |

| START ADDRESS | OVERLAY NUMBER | AREA | CATEGORY | NAME AND ENTRY | CODE LENGTH HEX DECIMAL | DECIMAL | REFERENCE BY |
|---------------|----------------|------|----------|----------------|----------------------------|---------|---|
| 0060 | | | 126 | \$WROF | 0024 | 36 | \$WROF \$WROF \$WROF |
| 0091 | | | 71 | \$WROF | 002A | 42 | \$WROF |
| 0088 | | | 93 | \$WROF | 00C0 | 208 | \$WROF |
| 0188 | | | 107 | \$WROF | 02A8 | 640 | \$WROF |
| 0430 | | | 28 | \$WROF | 070C | 172 | \$WROF \$WROF \$WROF \$WROF \$WROF |
| 043C | | | 28 | \$WROF | 00C8 | 8 | \$WROF |
| 0444 | 2 | | | OVERLAY | 01C4 | 452 | |
| 0800 | 1 | S | 2 | \$SCLIP | 0027 | 35 | \$WROF |
| 0827 | 1 | S | 2 | \$SCLIP | 0019 | 25 | \$WROF |
| 0844 | 1 | S | 2 | \$SRRR | 0082 | 130 | \$SRRR \$SCLIP |
| 08C6 | 1 | S | 6 | \$SRLA | 0026 | 38 | \$SCLIP \$SCLIP |
| 08E6 | 1 | S | 2 | \$SRUC | 001E | 30 | \$SCLIP \$SCLIP \$SRRR |
| 08FC | | | | DMSRLO | | | |
| 08FD | | | | DMSRUC | | | \$SCLIP \$SCLIP |
| 0908 | | | | DMSRER | | | \$SCLIP \$SCLIP \$SRRR |
| 0924 | 1 | S | 2 | \$SRDF | 0017 | 28 | \$SCLIP |
| 0940 | 1 | S | 2 | \$SRME | 00A4 | 164 | \$SRRR |
| 09C6 | 1 | S | 2 | \$SRSE | 0096 | 150 | \$SRRR |
| 0A01 | 1 | S | 2 | \$SRDI | 0055 | 145 | \$SRRR \$SRRR |
| 0A33 | | | | DMSRPD | | | \$SRRR |
| 0A4C | | | | DMSRKD | | | \$SRRR |
| 0A54 | 1 | S | 2 | \$SRBP | 0035 | 55 | \$SRRR \$SRRR \$SRRR |
| 0A73 | 2 | S | 6 | \$SRRR | 0125 | 295 | \$SRRR \$SRRR |
| 0A28 | 2 | S | 6 | \$SRRR | 00F0 | 240 | \$SRRR \$SRRR |
| 0B00 | 3 | C | 126 | \$WROF | 0075 | 117 | \$WROF \$WROF |
| 0B19 | | | | \$WROF | | | \$WROF |
| 0B7C | | | | \$WROF | | | \$WROF |
| 0B7D | | | | \$WROF | | | \$WROF |
| 0B81 | 3 | C | 126 | \$WROF | 0198 | 150 | \$WROF |
| 0BCE | | | | \$WROF | | | \$WROF |
| 0907 | | | | \$WROF | | | \$WROF |
| 0916 | | | | \$WROF | | | \$WROF |
| 0948 | 3 | C | 126 | \$WROF | 0198 | 150 | \$WROF |
| 0A39 | | | | \$WROF | | | \$WROF |
| 0A45 | | | | \$WROF | | | \$WROF |
| 0A53 | 3 | C | 126 | \$WROF | 0198 | 150 | \$WROF |
| 0A93 | 3 | C | 126 | \$WROF | 0198 | 150 | \$WROF |
| 0A94 | 3 | C | 28 | \$WROF | 009A | 156 | \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF |
| 0C0E | 3 | C | 44 | \$WROF | 00E8 | 104 | \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF |
| 0C78 | 3 | C | 11 | \$WROF | 0043 | 67 | \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF |
| 0C8E | 3 | C | 48 | \$WROF | 0130 | 112 | \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF |
| 0C95 | 3 | C | 57, 125 | \$WROF | 0178 | 170 | \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF |
| 0F61 | 3 | C | 107 | \$WROF | 0326 | 79 | \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF |
| 0F9A | 3 | C | 15 | \$WROF | 00C4 | 100 | \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF \$WROF |
| 0800 | 4 | C | 15, 125 | \$WROF | 01E5 | 261 | \$WROF \$WROF |
| 0967 | 4 | C | 15 | \$WROF | 00A5 | 165 | \$WROF \$WROF |
| 0968 | | | | \$WROF | | | \$WROF |

```

GL100 I THE TOTAL CORE USED BY REMAP IS 12280 DECIMAL.
GL101 I THE START CONTROL ADDRESS OF THE MODULE IS 40C0.
GL102 I THE NON-OVERLAY CORE SIZE IS 14064 DECIMAL.
GL103 I PROGRAM WILL NOT FIT IN THE CORE SIZE SPECIFIED.
GL104 I TOTAL NUMBER OF LIBRARY MODULES LOADED IS 49.
NAME=REMAP, PACK=FILE1, UNIT=FILE1A, LIBRARY=

```

- 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 nnnnn. | 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. | <p>Even with overlays, the program will not fit in storage. The storage size is either the core size specified on the <code>OPTIONS</code> statement (<code>CORE=annK</code>), 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:</p> <ol style="list-style-type: none"> 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 <code>namexx</code> WAS NOT REFERENCED BY AN EXTRN. | A module was read from the system input device (READER) or was referenced on an <code>INCLUDE</code> statement but was not referenced by an <code>EXTRN</code> in the object module. An <code>EXTRN</code> 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 <code>GROUP</code> or <code>CATEGORY</code> 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 <code>CATEGORY</code> statement with a value of 8-126, in addition to the <code>GROUP</code> statement. If no module named in a <code>GROUP</code> 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 <code>namexx</code> IN CATEGORY OR GROUP CARD NOT REFERENCED BY OBJECT PROGRAM. | <p>The name in the <code>CATEGORY</code> or <code>GROUP</code> statement is not referenced by any of the included or <code>AUTO-LINKED</code> modules. The name should be removed or the correct name determined from the storage map.</p> <p>This message may also appear if a module is named twice in <code>CATEGORY</code> assignments.</p> |
| OL035 | T SYSTEM AREA MODULE <code>namexx</code> CATEGORY <code>n</code> CALLS MODULE <code>namexx</code> . | 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. | <p>This message can occur for two reasons:</p> <ol style="list-style-type: none"> 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: <ol style="list-style-type: none"> 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. |

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 5704SC2 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:

| <i>Byte</i> | <i>Contents</i> |
|-------------|---|
| 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:

| <i>Bit</i> | <i>Meaning</i> |
|------------|---|
| 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 required to support programs referencing a common data area which may be considered external to all included modules. Each 3-byte RLD contains the following:

| <i>Byte</i> | <i>Bit</i> | <i>Meaning</i> |
|----------------|------------|--|
| 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

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

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.

- address, link-edit start 16
 - address, overlay program load 16
 - alignment, modules on a boundary 10
 - allocating work files (see examples)
 - AREA-USER parameter
 - co-resident area 14
 - GROUP statement 9
 - arithmetic overlay segment, system 14
 - assignment, modules to overlays 9
 - assignment, modules to root segment 16
 - assignment, overlays 14
 - ATTR parameter, OPTIONS statement 8
 - (see example 1)
 - attributes of load module 8
 - AUTOLINK 43
-
- boundary alignment of modules 10
 - BSCA modules, category value change 9
 - buffers, external (5704-SC2 only) 44
-
- CATEGORY statement 5, 9, 19
 - category value
 - change 9
 - (see examples 6-11)
 - system modules 14
 - Category, storage map heading 12
 - CCP, memory resident overlay 19
 - change category value 9
 - (see examples 6-11)
 - changing overlay structure (see examples 6-11)
 - checkpoint restart program 8
 - (see example 1)
 - COBOL, user overlay area 14
 - Code Length Decimal, storage map heading 12
 - Code Length Hexadecimal, storage map heading 12
 - common areas
 - GLOBAL 13, 44
 - LOCAL 13, 44
 - communications control program, memory
 - resident overlay 19
 - compiler entry
 - description 4
 - functions 2
 - compiler object modules 9
 - conditional external reference, ESL
 - record 44
 - configuration, system 3
 - control statements
 - description 5
 - examples 23
 - CORE parameter, OPTIONS statement 7
 - co-resident area 14
 - (see examples 6-11)
 - cross-reference list
 - MAP parameter 8
 - storage map 12
-
- descendant, mainline 14
 - design, overlay structure 9
 - destination of load module, PHASE
 - statement 6
 - determining overlay modules 16
 - disk space, linkage editor 4
-
- end record, object module 45
 - END statement 5, 11
 - ENTRY parameter
 - (see example 3)
 - OPTIONS statement 7
 - entry point
 - ESL records 43
 - load modules 18
 - (see ENTRY parameter)
 - entry, compiler 4
 - Entry, storage map heading 12
 - entry, user 4
 - EQUATE statement 5, 11
 - error halts 3
 - error messages 38
 - ESL (external symbol list) records 43

- examples
 - example 1, OCL and control statements 23
 - example 2, OCL and control statements 24
 - example 3, OCL and control statements 25
 - example 4, OCL and control statements (Model 15 only) 26
 - example 5, OCL and control statements 26
 - example 6, overlay structure of load module 27
 - example 7, overlay structure of load module 29
 - example 8, overlay structure of program 31
 - example 9, overlay structure of program 32
 - example 10, overlay structure of program 33
 - example 11, overlay structure of program 34
 - example 12, MOVE technique (Model 15 only) 35
 - example 13, REMAP technique (Model 15 only) 36
- extended root mainline 14
- external buffers (5704-SC2 only) 44
- external reference, ESL records 43
- external symbol list records 43
- EXTRN (see ESL records)

- fetch routine, overlay (see overlay fetch routine)
- fetch table
 - generation 14
 - root area 13
- FILE statements 4
- FORTRAN 22, 44

- generation, overlays
 - memory resident overlays 19
 - overlay fetch routine 14
- GLOBAL COMMON area 13, 20, 21, 22
- GROUP statement 5, 9
 - (see examples 4 and 7)
 - description 18
 - system modules 14
- grouping modules 18

- halts, error 3
- header record 43

- I/O dependent modules, user overlay areas 13
- I/O independent modules, co-resident area 14
- improvements, performance 47
- INCLUDE statement 5
 - description 9
 - mainline routine 14

- including modules in root 18
- increase size of user area (see example 7)
- informational messages 38
- INVOKE feature
 - GLOBAL COMMON 44
 - LOCAL COMMON 44
 - memory resident overlays 22

- language translator 1
- last control statement (see END statement)
- LEVEL parameter
 - (see example 1)
 - OPTIONS statement 7
- library format, object 2
- link-edit start addresses 16
- LINKADD parameter, PHASE statement 6, 16
- linkage editor
 - control statements 5
 - functions 2
 - loading 4
 - processing 1
- load address, overlay programs 16
- load module
 - control statements 5
 - destination 6
 - entry point 18
 - (see ENTRY parameter)
 - introduction 1
 - name 6
 - RETAIN type 6
 - size change 3, 14
 - (see CORE parameter)
 - storage map 12
 - storage size 7
- loading linkage editor 4
- LOCAL COMMON area 13, 20, 21, 22
- location of user modules (see UPACK parameter)

- mainline routine
 - assignment 9
 - card form 14
 - (see example 3)
 - entry point (see ENTRY parameter)
 - INCLUDE statement (see examples 1, 2, and 4)
 - overlay assignment 14
 - root area 13
- MAP parameter, OPTIONS statement 8
- memory resident overlay
 - Communications Control Programming 19
 - description 19
 - examples 35, 36
 - MOVE technique 21
 - performance considerations 47
 - program execution 20
 - REMAP technique 22
 - supervisor support 19

- messages 38
- MFCU modules, special AUTOLINK 43
- modification levels (see LEVEL parameter)
- module
 - assignment to overlay 9
 - boundary alignment 10
 - movement 9, 14
 - name, ESL record 43
- module, load (see load module)
- module, object (see object module)
- module, overlay (see overlay module)
- MOVE technique, memory resident overlays 19

- NAME parameter
 - CATEGORY statement 10
 - GROUP statement 9
 - INCLUDE statement 9
 - PHASE statement 6
- name, load module
 - PHASE statement 6
- name, module, ESL record 43
- Name, storage map heading 12
- NEWNAME parameter, EQUATE statement 11
- non-overlay storage size 12

- object library format 2
- object module
 - card form from SYSIN 9, 14
(see examples 2, 3, and 4)
 - control statements 5
 - description 43
 - GROUP statement 9
 - introduction 1
- OCL statements
 - description 4
 - examples 23
- OLDNAME parameter, EQUATE statement 11
- one-byte RLD 44
- OPTIONS statement 5
 - description 7
 - memory resident overlays 19
- output during link-edit, printer 8
- overlay area
 - co-resident area 14
 - resident 20
 - root area 13
 - system overlay segment 14
 - user overlay area 13
- Overlay Area, storage map heading 12
- overlay assignment method 15
- overlay determination 16
 - memory resident considerations
 - MOVE technique 19
 - REMAP technique 19
 - overlay fetch routine
 - entry point 18
 - memory resident overlay 20
 - root area 13
 - size 14, 22
 - overlay modules in root, including 18
 - Overlay Number, storage map heading 12
 - overlay programs, load address 16
 - overlay segments, system 14
 - overlay storage size 12
 - overlay structure
 - change (see examples 6-11)
 - design 9
 - overlays
 - fetch routine 14, 20, 22
 - memory resident 19
 - overriding priority (see CATEGORY statement)
- parameter summary 6
- performance improvements 47
- permanent load modules 6
- PHASE statement 5
 - description 6
- primary storage requirements 3
- printer output during link-edit 8
- priority (see category value)
- processing, linkage editor 1
- program failure, category value change 9
- program modification levels (see LEVEL parameter)
- program, source 1
- PUNCH parameter, PHASE statement 6
- punching object module (see PUNCH parameter)

- Referenced By, storage map heading 12
- Relocation Dictionary Records 44
- REMAP technique, memory resident overlay 19
- resident overlay area 20
- resident overlay program execution, memory 20
- RETAIN parameter, PHASE statement 6
- RLD parameter, PHASE statement 6
- RLD records 44
- root area, contents 13, 20, 21, 22
(see examples 6-13)
- root mainline, extended 14
- root segment, module assignment 16
- routine, mainline (see mainline routine)

- secondary storage requirements 3
- segments, loading overlay 14, 20, 22
- size, overlay fetch routine 14, 19
- source program 1
- start address, link-edit 16
- Start Address, storage map heading 12
- statements, control 5
- statements, OCL 4
- storage map
 - contents 12
 - MAP parameter 8
- storage requirements
 - primary 3
 - secondary 3
- storage size, load module 7
- storage size, non-overlay 12
- storage size, overlay 12
- subtype, external references 43
- symbol list records, external 43
- SYSLOG 3
- system configuration 3
- system modules
 - category value 10
 - system overlay area 14
- system modules, category value change 9
- system overlay area 14
 - (see examples 6, 7, 10-13)
 - not used (see examples 8 and 9)
- system overlay fetch area 20, 21, 22
 - (see examples 6, 7, 10-13)
 - not used (see also examples 8, 9)

- techniques, memory resident overlay
 - MOVE 19
 - REMAP 19
- temporary load modules 6
- terminal messages 38
- text-RLD record 44
- three-byte RLD 45
- transfer vectors 13
- translator, language 1

- UNIT parameter
 - INCLUDE statement 9
 - PHASE statement 6
- UPACK parameter, OPTIONS statement 7
- user entry
 - description 4
 - functions 2
- user overlay area 13
 - (see examples 6, 7, 10-13)
 - not used (see examples 8 and 9)
 - size 18
- user overlay fetch area 20, 21, 22
 - (see examples 6, 7, 10-13)
 - not used (see examples 8, 9)

- VALUE parameter, CATEGORY statement 10
- varying overlay structure (see examples 6-11)
- vectors, transfer (see transfer vectors)

- warning messages 38
- weak external references, ESL record 44



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)