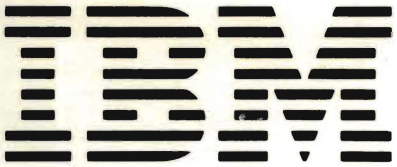


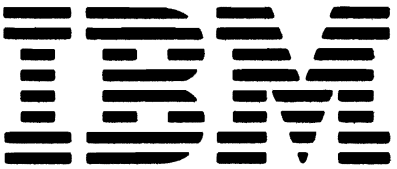
SC21-7707-0

File No. S34-36

IBM System/34
Overlay Linkage Editor
Reference Manual

Program Number 5726-SS1

OVERLAY LINKAGE EDITOR



SC21-7707-0

File No. S34-36

IBM System/34
Overlay Linkage Editor
Reference Manual

Program Number 5726-SS1

OVERLAY LINKAGE EDITOR

Preface

The Overlay Linkage Editor is a part of the IBM System/34 System Support Program Product (Program Number 5726-SS1). This manual is intended for experienced programmers who plan to link-edit their own object modules rather than have the language processors (assembler and compilers) do the link-editing.

This publication contains:

- Introductory information regarding system configuration and storage requirements
- General information about input, output, and use of the Overlay Linkage Editor
- Specific information about the contents of the various overlay areas and how the user can design an overlay
- Examples of how the Overlay Linkage Editor can be used

Related Publications

These publications contain information that further describes topics discussed in this manual:

- *IBM System/34 Basic Assembler and Macro Processor Reference Manual*, SC21-7705
- *IBM System/34 System Support Reference Manual*, SC21-5155
- *IBM System/34 System Data Areas and Diagnostic Aids Handbook*, LY21-0049
- *IBM System/34 System Support Program Logic Manual: System*, LY21-0050
- *IBM System/34 FORTRAN IV Reference Manual*, SC21-7706

First Edition (October 1977)

This edition applies to release 2, modification 0 of the IBM System/34 System Support Program Product (Program Number 5726-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest *IBM System/34 Bibliography*, GH30-0231, for the editions that are applicable and current.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. Use this publication only for the purpose stated in the *Preface*.

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

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. Comments become the property of IBM.

Contents

Chapter 1. Introduction

Overlay linkage editor processing is necessary following the assembly or compilation of any program except an RPG program, which uses the RPG linkage editor. The output of a language processor (assembler or compiler) is called an object module (see Figure 1-1). An object module cannot be run as a program until it is link-edited into a load module. Object modules and load modules reside in the library on disk as subroutine members and load members respectively.

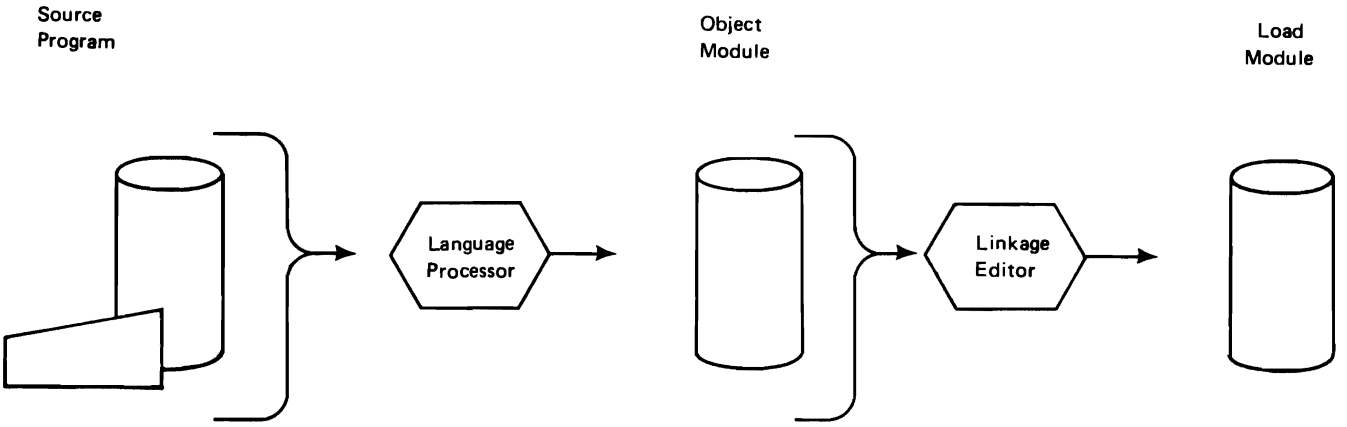


Figure 1-1. Preparing a Source Program for Execution

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

- Catalogs the output object module of a language processor, such as IBM System/34 Assembler, into the library on disk as a subroutine member.
- Link-edits the output object module of a language processor into a load module and catalogs it into the library on disk as a load member. The assigning of modules to overlay segments is determined automatically by the Overlay Linkage Editor.

The user entry allows the programmer to link-edit object modules into load modules. The programmer can influence the determination of overlays himself or he can allow the Overlay Linkage Editor to determine the overlay structure. The load modules are cataloged into the library on disk as a load member.

Note: For further information concerning library organization, see the *System Support Reference Manual.*

SYSTEM CONFIGURATION

The Overlay Linkage Editor is a part of the IBM System/34 System Support Program Product and runs on all models of the IBM System/34.

PRIMARY STORAGE REQUIREMENTS

The primary storage requirement for the execution of the Overlay Linkage Editor is 14K bytes of main storage.

SECONDARY STORAGE REQUIREMENTS

The Overlay Linkage Editor requires 21 library blocks. For execution, work space must be available on the disk; this space can be specified by the programmer or allocated by the Overlay Linkage Editor (see index entry: *OCL statements*).

ERROR HALTS

Halts are issued with error messages on the system logging device for error conditions.

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. If this type of change increases the size of a load module so it no longer fits in available main storage, a message is issued and it will be necessary to build a new load module by rerunning the Overlay Linkage Editor.

Chapter 2. Using the Overlay Linkage Editor

This chapter describes the compiler and user entries to the Overlay Linkage Editor and the storage map that is printed by the Overlay Linkage Editor and is used by the programmer to determine the structure of a program. The requirements for the object modules processed 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 processors to catalog their output object modules (object modules are described in Appendix B). Language processors can also specify link-editing. The Overlay Linkage Editor then link-edits the object module into a load module and catalogs the load module into the library, if requested.

When the programmer 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 this entry to the Overlay Linkage Editor, the programmer must supply:

- Operation Control Language (OCL) statements
- Overlay Linkage Editor control statements

OCL Statements

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

```
// LOAD #OLINK
// FILE NAME-$WORK,RETAIN-S,BLOCKS-40
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-40
// RUN
// MODULE NAME-CCCCCC,LIBRNAME-#LIBRARY
// OPTIONS ATTR-DED,MRTMAX-0,SUBLIB-#LIBRARY
// PHASE NAME-CCCCCC,LIBRNAME-#LIBRARY,RETAIN-P
// END
```

The LOAD statement loads the Overlay Linkage Editor into main storage from disk.

The FILE statements are optional. The Overlay Linkage Editor will find disk space if FILE statements are not supplied. Space will be assigned if there is a minimum of 24 blocks available (even though 24 blocks may not be sufficient for a large program). FILE statements should be supplied for large programs (25K or more) to ensure that the Overlay Linkage Editor has adequate work space to complete the link-editing. Also, the Overlay Linkage Editor uses all available main storage for the file buffers; therefore, the file must be equal to or greater than the region size. If the two FILE statements are supplied, they must be the same as the standard FILE statements used by the compiler.

The RUN statement starts the execution of the Overlay Linkage Editor.

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

Control Statements

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

1. PHASE statement: Optional, none or one allowed.
2. OPTIONS statement: Optional, none or one allowed.
3. MODULE statements: Required, one or more than one allowed.
4. GROUP statement: Optional, none, one, or more than one allowed.
5. CATEGORY statement: Optional, none, one, or more than one allowed.
6. EQUATE statement: Optional, none, one, or more than one allowed.
7. END statement: Required, only one allowed.

Control Statement Summary

Use	Control Statement
Define load module	// PHASE NAME–name, LIBRNAME–name, RETAIN–{P / R}, LINKADD–{X'oooo' / X'aaaa'}, RLD–{YES / NO}
Define environment	// OPTIONS STORE–annK, LEVEL–nnn, ENTRY–label, ATTR–{xxx / 'xxx, xxx, ...xxx'}, MAP–{YES / NO / XREF / MSG}, SUBLIB–{name / 'name1, name2'}, MRTMAX–nnn
Define object modules	// MODULE NAME–{name / 'name, name, . . . name'}, LIBRNAME–name
Group object modules together in storage	// GROUP NAME–{name / 'name, name, . . . name'}, AREA–USER
Change category (priority) of object module	// CATEGORY NAME–{name / 'name, name, . . . name'}, VALUE–nnn
Equate module names	// EQUATE OLDNAME–{name / 'name, name, . . . name'}, NEWNAME–{name / 'name, name, . . . name'}
End of control statements	// END

Parameter Descriptions

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 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: *MODULE statement*).

// PHASE NAME–name, LIBRNAME–name, RETAIN–{P / R},

LINKADD–{X'oooo' / X'aaaa'}, RLD–{YES / NO}

NAME: The name that the load module will have in the library directory. If the NAME parameter is not supplied, the load module will have the name of the mainline routine. The name can be from one to six characters long. The first character must be alphabetic and can contain any combination of System/34 characters except blanks, commas, quotes, or periods.

LIBRNAME: The name of the library where the load module is to be placed. The library name can be from one to eight characters long. If this parameter is omitted, the load module is placed in the system library.

RETAIN: Specifies whether or not the operator is notified of an existing load member in the library specified by LIBRNAME or the system library. RETAIN–R means the load module is to replace an existing member with the same name and the operator is not notified that a duplicate entry existed. If RETAIN–P is specified, and a member exists with the same name, a message is issued that requires an operator response before replacing an existing member with a new member. If RETAIN is not specified, P is the default.

LINKADD: Specifies the link-edit start address, which is the relocatable address assigned to the first byte of the link-edited load module. This address must be a multiple of eight if RLD–NO (on the PHASE statement) or ATTR–COM (on the OPTIONS statement) is specified. If this parameter is omitted, the link-edit address is X'0000'. If the start address plus the number of bytes in the program exceeds X'FFFF' (64K), the program is link-edited to start at X'0000'.

RLD: Specifies whether the program will be produced with Text-Relocation Directory records (RLDs). If this parameter is omitted, YES is the default (see index entry: *RLD records*).

OPTIONS Statement

The OPTIONS statement describes the load module and specifies the type of linkage editor output. If the entire OPTIONS statement or any of the parameters are omitted, the system uses the defaults given in the following parameter explanations.

// OPTIONS STORE–annK, LEVEL–nnn, ENTRY–label, ATTR–{xxx / 'xxx, xxx, ...xxx'}

, MAP–{YES / NO / XREF / MSG}, SUBLIB–{name / 'name1, name2'}, MRTMAX–nnn

STORE: The storage size the load module has available for execution. If specified, the library directory entry contains this size even though the actual size required by the load module is less. If not specified, the available user main storage is used to determine when overlays are required and the library directory entry contains the actual load module size.

a = 1/4 increments expressed as:
Q = 1/4 or 256 bytes
H = 1/2 or 512 bytes
T = 3/4 or 768 bytes
0 = zero bytes

nn = 1K increments, expressed as a two-digit decimal number

Example: Q04K = 1/4K + 4K = 256 + 4096 = 4352 bytes

LEVEL: The number that is placed in the level entry in the library directory. Different modification levels of load modules can be assigned different level values. The maximum value for nnn is 255. The default is 001.

ENTRY: An entry point or module name of an included module. The label must be six characters or less. The default is the entry point of the mainline routine.

ATTR: Attributes of the module being link-edited. If ATTR is not specified, no attributes are assigned. The possible values are:

COM (program common supported): This module requires the calculation of a new load address at load time. This ensures that the new main storage load address is beyond the program's own common region.

DED (dedicated module): This module must execute alone. No other tasks can be executed.

LSC (load only from system console): This module cannot be loaded from a display station.

NEP (never-ending program): This module is a long-running program and any resources allocated to the program are not available to other programs. The never-ending program attribute can be assigned to both multiple requestor terminal programs and single requestor terminal programs. This attribute can also be assigned to programs executed from the input queue. In addition, the never-ending program attribute assigned to a multiple requestor terminal program allows the program to do an ACCEPT INPUT operation without an outstanding INVITE.

NEX (not executable): This module is loaded by another module and cannot be loaded by a LOAD OCL statement.

Note: If NEX is specified, none of the other attributes may be specified.

NIQ (noninquirable module): This module cannot be interrupted by using the inquiry key.

NSW (nonswappable module): This module is never interrupted or put to disk. This attribute may reduce system throughput.

SIS (scientific instruction set microcode): This module requires the scientific instruction set microcode for execution.

SRQ (source required): This module requires the allocation of the $WORK and $SOURCE files. $SOURCE must be filled either from the system input device or a source library.

UCS: This module reads utility control statements.

MAP: 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.

SUBLIB: The name of the library that contains the user subroutines to be link-edited. If two names are entered, name1 library is searched first, followed by name2 if necessary. The library name can be from one to eight characters long. If this parameter is omitted or if the subroutines are not found in the library or libraries named, the Overlay Linkage Editor searches the system library for the user modules. The Overlay Linkage Editor uses this parameter when performing AUTOLINK.

MRTMAX: The maximum number of active requesting display stations that can be allocated to the program. The maximum value for nnn is 255. This parameter indicates that the module can be used as a multiple requestor terminal program. If zero is specified or if MRTMAX is omitted, the program is not considered a multiple requestor terminal program.

MODULE Statement

The MODULE statement specifies which object modules are to be included in the load module. Multiple module names may be submitted on one MODULE statement; the mainline routine is the first object module named. If a module name is not found in the library, an error message is displayed; if you take a zero option, the Overlay Linkage Editor will do a find on the next module name in the statement.

CAUTION
Programs that include instructions that decrement the location counter cannot be link-edited on System/34. An object module that includes a decremented ORG statement causes a terminal error at link-edit time.

// MODULE NAME– $\left\{\begin{matrix}\text{name} \\ \text{'name,name,...name'}\end{matrix}\right\}$,LIBRNAME–name

NAME: The names of the object modules to be included in this program. Each name must be six characters or less. The maximum number of characters that may be entered is 60 (including the commas and the two single quotes).

LIBRNAME: The name of the library where the object modules are located. The library name can be from one to eight characters long. Only one LIBRNAME parameter is allowed with each MODULE statement. If this parameter is omitted, the system library is the default.

GROUP Statement

The GROUP statement specifies a number of object modules that the programmer wishes to group together in storage. The programmer may design an 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 coresident 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 is not supplied, the Overlay Linkage Editor designs the overlay structure (see index entry: *overlays*).

// GROUP NAME– { name / 'name,name,. . . name' } ,AREA–USER

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. Each name must be six characters or less. The maximum number of characters that may be entered is 60 (including the commas and the two single quotes).

AREA: If the module named in this statement is assigned to an overlay, it will be assigned to the user overlay area. If a list of names ('name,name, . . .') precedes 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.

For more detailed information on grouping, See *Using the GROUP Statement*, in Chapter 3.

CATEGORY Statement

The CATEGORY statement temporarily changes the category value (priority) of an object module. Because the priority of an object module influences the placement of the module in an overlay segment, the category value of the module is changed only for this link-editing.

Notes:
1. Modules in the same overlay do not need to have the same category value.
2. For a discussion of all the factors that determine overlay assignments, see *Assigning Overlays* in Chapter 3.

// CATEGORY NAME–{name / 'name,name,...name'},VALUE–nnn

NAME: Names of modules for which the category value (priority) is to be changed for this link-editing. Each name must be six characters or less. The maximum number of characters that can be entered is 60 (including the commas and the two single quotes).

VALUE: The new category value:

0 A module with this category value cannot be overlaid. It is always placed in the root segment.

1-7 These category values are generally reserved for system modules. These modules can be overlaid if necessary to satisfy main storage size. Modules with these category values may call only modules with the same category value or category 0 modules. IBM system modules have the following general category values:

Category Value	**System Module**
2	Disk I/O
4	Arithmetic
5	I/O control modules
6	Display I/O

Category values 1, 3, and 7 have no assignment and are available for use.

8-125 These category values assign overlay priorities. The lower the number, the less likely that the module will be overlaid.

126 This category value assigns a special overlay priority. In any overlay program, routines of category 126 will be given first consideration for reinclusion in the root area (nonoverlay main storage).

127 This value is treated the same as a category value of 0 (zero). It is displayed on the storage map as category 0, not category 127, except when it is assigned by a CATEGORY statement.

128 This value specifies that the module must be aligned on a 256-byte boundary. Value 128 can be used with any lower category value. This is done by adding the lower value to 128. For example, you can specify that a module have a category value of 8 and be aligned on a 256-byte boundary by specifying a category value of 136 (8 + 128) on the CATEGORY statement. Category 128 indicates a category 0 module aligned on a 256-byte boundary.

If an input module contains a category value of 128, the module 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 a module not automatically aligned on a boundary (that is, with a priority less than 128) can be aligned for this link-edit.

CAUTION
If a module has input/output buffers, the buffers *must* be aligned on an 8-byte boundary. Even if the compiler indicates that the buffers are aligned on 8-byte boundaries, this does not ensure that the buffers will be aligned after link-edit.

Assuming that you have aligned the input/output buffers properly in your module (subroutine), there are two ways to ensure that the alignment will not be altered by the link-edit:

- Assign all input/output buffers to the mainline routine.
- Use a category value of 128 to align the module to a 256-byte boundary.

All overlays are aligned on a 256-byte boundary, therefore, if the module is the first module in an overlay, it will be on a 256-byte boundary.

EQUATE Statement

The EQUATE statement allows 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, and so on.

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

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

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

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

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

OLDNAME: The module name or entry point now referenced in the program. The name must be six characters or less. The maximum number of characters that may be entered is 60 (including the commas and the two single quotes).

NEWNAME: The module name or entry point that will replace the referenced name or entry point in the program. The name must be six characters or less. The maximum number of characters that may be entered is 60 (including the commas and the two single quotes).

END Statement

An END statement indicates the end of the Overlay Linkage Editor input and must follow the control statements read from the keyboard or procedure member.

```
// END
```

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 is included). The Overlay Area names the area into which each overlay is loaded: U for user area, S for system area, and C for coresident 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 nonoverlay storage size is also printed. The storage size of an overlay program is always in increments of 256 bytes. The nonoverlay storage size is the exact number of bytes in the load module.

The storage map can be omitted to save link-edit time.

(See examples 6 through 11 for sample output.)

Chapter 3. Overlays

OVERLAY AREAS

Main storage for an object program with overlays may be divided into four areas: root, user, system, and coresident (see Figure 3-1). 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 in Chapter 4.

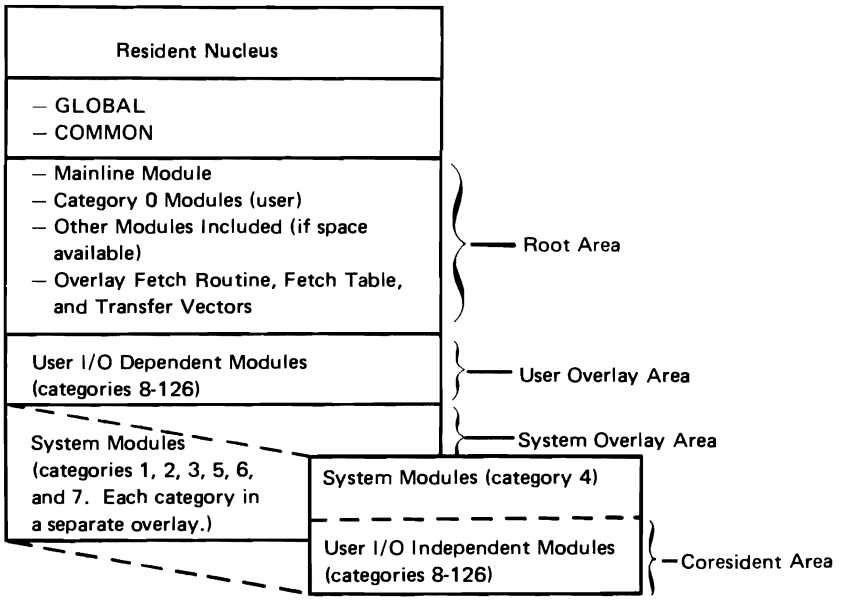


Figure 3-1. Overlay Areas

Root Area

The root area of an overlay program contains the parts of the program that are never overlaid (see Figure 3-1). 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 customer modules that call system I/O modules. An overlay segment loaded into the user overlay area can contain modules of different category values.

System Overlay Area

A system overlay segment contains system modules with the same category value, and all included system modules with that category value are assigned to that overlay segment. Each system overlay segment is independent of other system overlay segments; that is, a system module can call only another module with the same category or a module with category 0.

Coresident Area

The coresident area is actually a part of the system overlay area (see Figure 3-1). The system arithmetic overlay segment (category 4) is sometimes smaller than the system overlay area. If it is smaller, the remaining space is the coresident area and can be used to load user modules that are I/O-independent (do not call system I/O modules). If the main storage requirement of category 4 plus the coresident area is greater than the size of the system overlay area, category 4 modules are reincluded in the root area until all category 4 routines are in nonoverlay main storage, or until the category 4 plus coresident area can fit into the system overlay area. An I/O-independent 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.

Routines of category 126 can be given first consideration for reinclusion in the root area (nonoverlay main storage).

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 3-2 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 (on a MODULE statement) 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 nonzero category 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 (in the STORE parameter of the OPTIONS statement) and generating overlay segments would not enable it to fit better in storage, overlay segments are not generated, and a message is issued.

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.

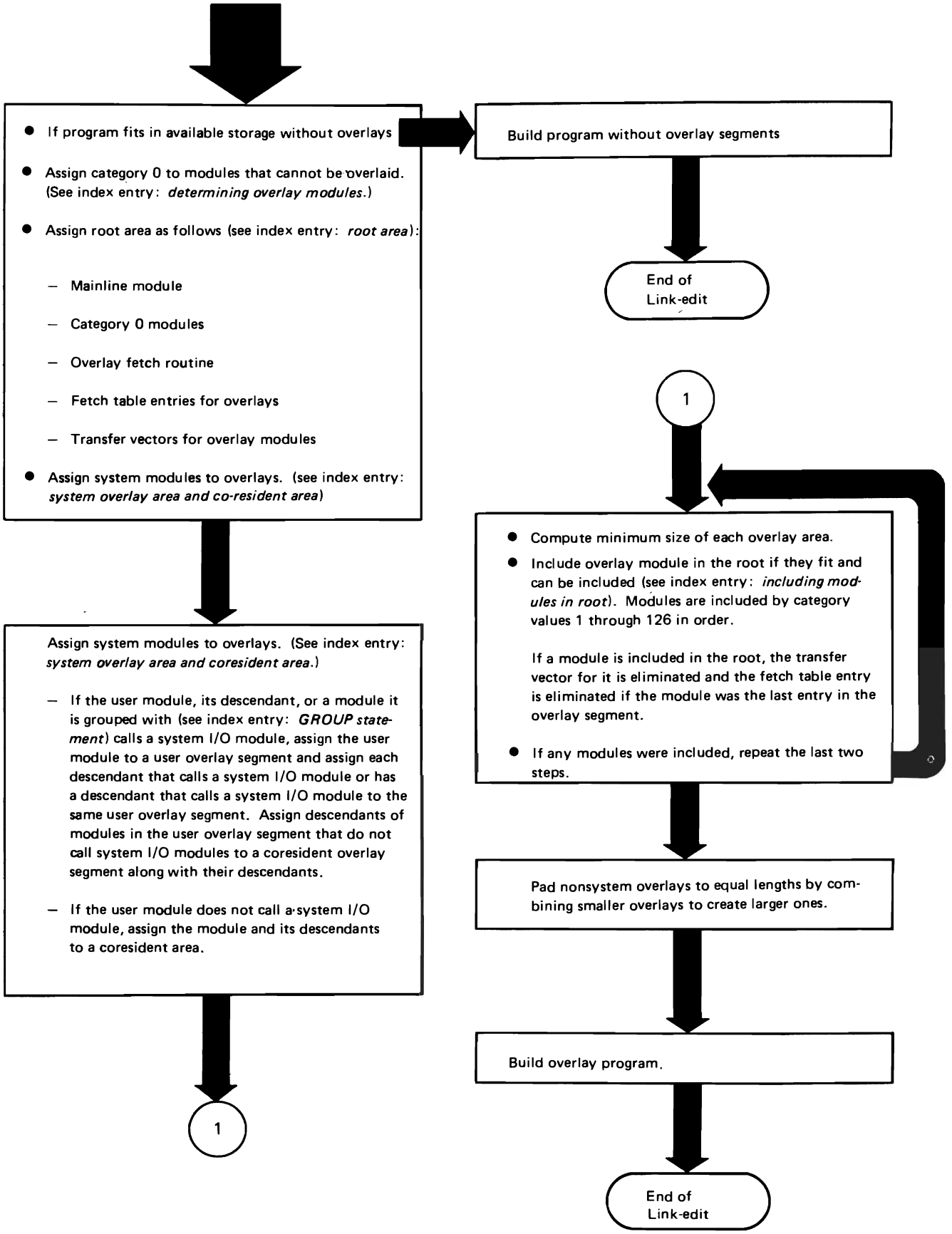


Figure 3-2. Overview of the Overlay Linkage Editor Method of Assigning Overlays

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 121 bytes, 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.

Determining Which Modules Can Be Overlaid

The Overlay Linkage Editor considers a module capable of being overlaid if the category of the module is nonzero and if the module is a direct descendant of (called by) the mainline routine (the first module named on the MODULE statement) or descends from the mainline routine through only category 0 modules. A, C, G, and H in Figure 3-3 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 3-3) can be overlaid only if the module called has no direct or indirect call to an overlayable module. C in Figure 3-3 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 the root.

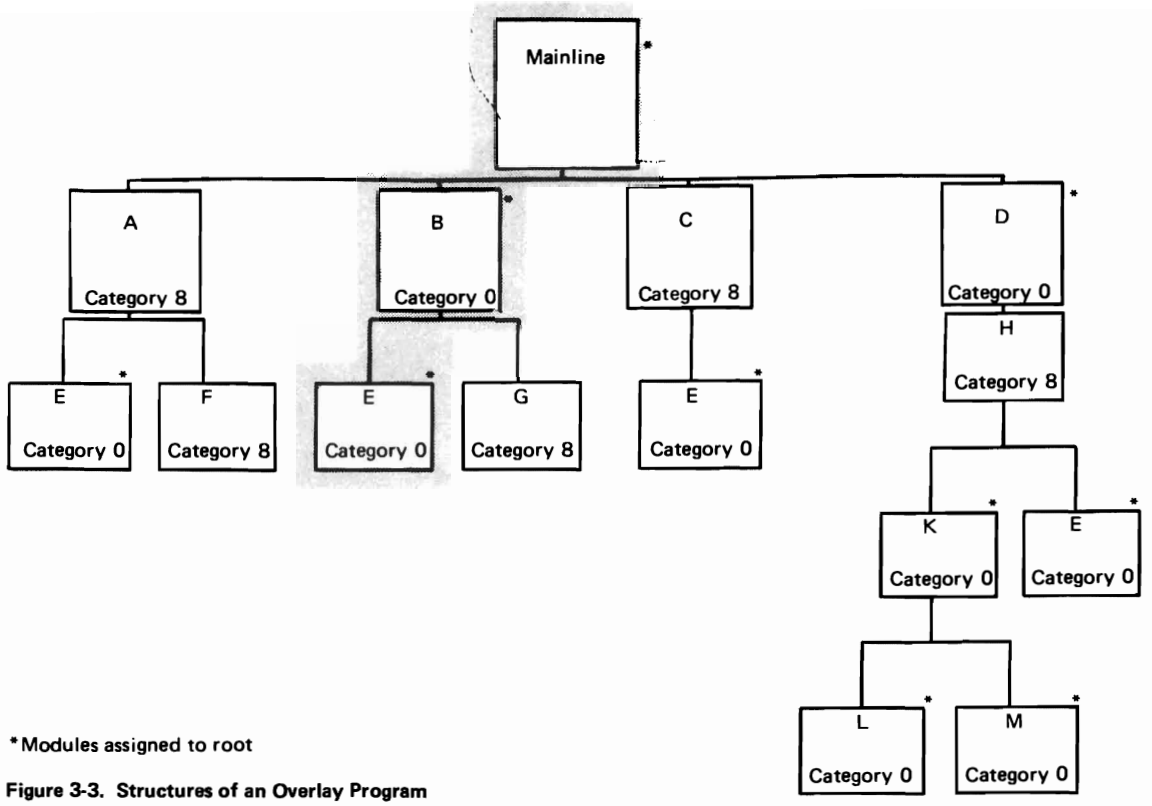
A module called by an overlay module can itself be overlaid (module F in Figure 3-3).

Modules that do not qualify for overlay segments are assigned to the root segment. Module C in Figure 3-4 is assigned to the root segment because it appears twice in the program. Modules C and F in Figure 3-5 are assigned to overlay segments because each appears only once in the program, even though they do not meet the requirements mentioned above for overlay modules.

Including Overlay Modules in the Root

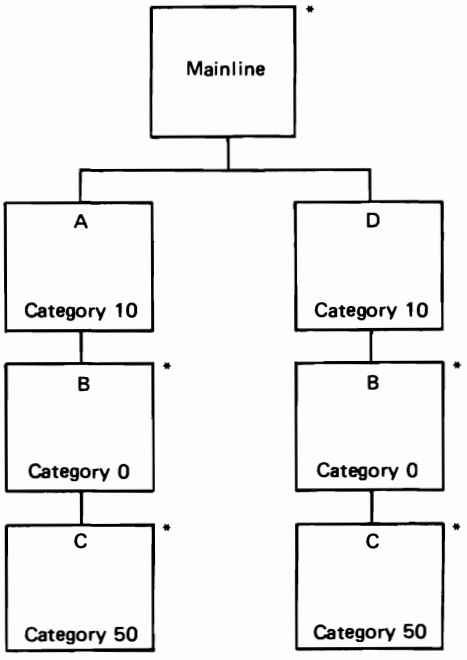
After the Overlay Linkage Editor has assigned all modules to either the root segment or to overlay segments, any overlay modules that can be included in the root segment without exceeding the user-specified main storage size are included. A module can be included if it meets one of the following criteria:

- 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.



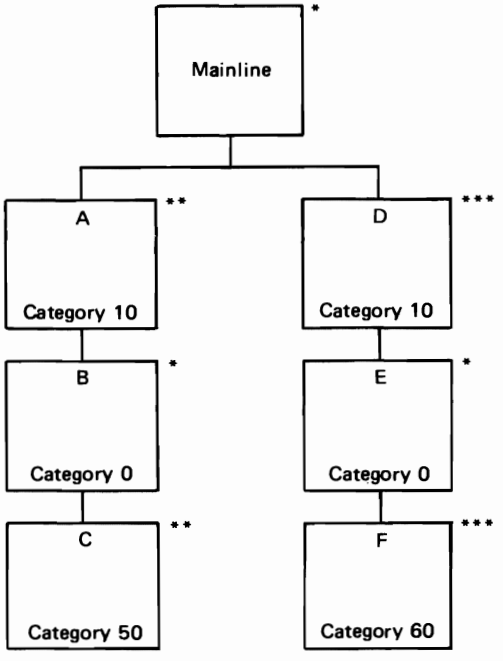
*Modules assigned to root

Figure 3-3. Structures of an Overlay Program



*Modules assigned to root

Figure 3-4. User Modules Assigned to the Root Because they Cannot be Overlaid



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

Figure 3-5. Normal Root Modules Assigned to Overlays

Link-Edit Start Addresses

If LINKADD is not coded on the PHASE statement, the program is link-edited to start at X'0000'. If the start address plus the length of the program exceeds 64K, the program is link-edited to start at X'0000'.

The link-edit start address does not affect the load address.

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 a load module that references a common area must be the first byte of the module.

Overlay Area Size

The Overlay Linkage Editor assigns the smallest overlay areas possible. The programmer can increase the size of the overlay area, 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 sizes of the other overlays to take advantage of the increased area. This reduces the number of overlays.

Using the GROUP Statement

The GROUP statement specifies 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 must be the first module named on the GROUP statement.

Figure 3-6 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 coresident 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).

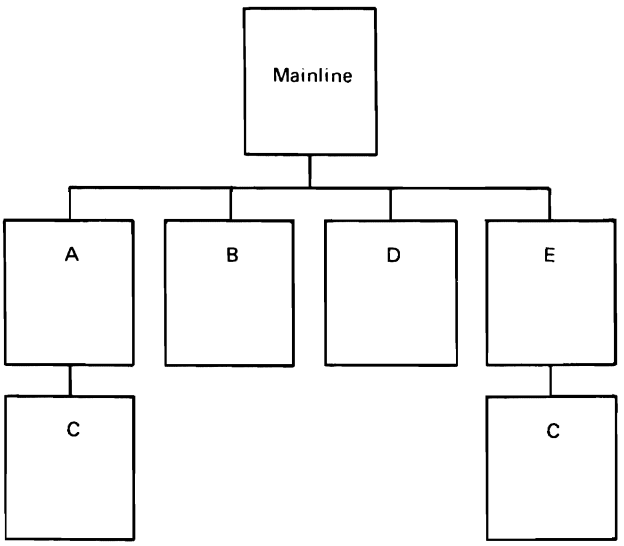


Figure 3-6. Structure of Sample Program

Chapter 4. Examples

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 statements in each job.

Example 1

```
1 // LOAD #OLINK
2 // FILE NAME-$SOURCE,BLOCKS-20,RETAIN-S
3 // FILE NAME-$WORK,BLOCKS-20,RETAIN-S
4 // RUN
5 // PHASE NAME-TEST50,RETAIN-P,LINKADD-X'0A00',LIBRNAME-TEST
6 // OPTIONS ATTR-SRQ,LEVEL-20,SUBLIB-TEST,MRTMAX-007
7 // MODULE NAME-TEST40,LIBRNAME-TEST
8 // END
```

1. The Overlay Linkage Editor is loaded from the disk.

2. The $SOURCE file of 20 blocks is allocated on the disk. This is a scratch file.

3. The $WORK file of 20 blocks is allocated on the disk. This is a scratch file.

4. Execution of #OLINK (last OCL statement) is started.

5. The output load module is generated as a load member in the library named TEST under the name TEST50. If a load member already exists with the same name, a message will be issued. The start address is X'0A00'.

6. This program (TEST50) requires $WORK and $SOURCE files. The level number is 20. Subroutine members to be link-edited reside in the library named TEST. The maximum number of requestor terminals that can access the program at one time is seven.

7. The mainline routine for this program is an object module (subroutine member) in the library named TEST under the name TEST40.

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

Example 2

```
1  // LOAD #OLINK
   // RUN
2  // PHASE NAME-BLUE,RETAIN-R
3  // MODULE NAME-BLUE
4  // CATEGORY NAME-WHITE,VALUE-20
5  // EQUATE NEWNAME-RED,OLDNAME-YELLOW
   // END
```

1 The Overlay Linkage Editor is loaded from the disk. Because no file statements are given, the Overlay Linkage Editor finds from 24 to 72 blocks of work space on disk for each file ($WORK and $SOURCE).

2 The output load module is generated as a load member in the system library under the name BLUE. If a load member already exists with the same name, it will be replaced and no message will be issued.

3 The mainline routine for this program is BLUE and is a subroutine member of the system library.

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

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

Example 3

```
1  // LOAD #OLINK
1  // FILE NAME-$WORK,BLOCKS-40,RETAIN-S
   // FILE NAME-$SOURCE,BLOCKS-40,RETAIN-S
   // RUN
2  // PHASE NAME-BLACK,RETAIN-P
   // OPTIONS STORE-H12K,ENTRY-WHITE,MAP-MSG,SUBLIB-DEPT300
3  // MODULE NAME-GRAY,LIBRNAME-DEPT100
4  // CATEGORY NAME-'RED,GREEN',VALUE-55
5  // CATEGORY NAME-YELLOW,VALUE-25
6  // EQUATE OLDNAME-'A,B,C',NEWNAME-'X,Y,Z'
7  // EQUATE OLDNAME-'D,E,F',NEWNAME-'Q,Q,Q'
   // END
```

1 $WORK and $SOURCE are allocated 40 blocks each on the disk and are scratch files.

2 The object code is constructed so that the program can run in 12.5K of main storage. The entry point WHITE is the start control address. Only messages are printed. Subroutine members to be link-edited reside in the library named DEPT300.

3 The mainline routine for this program is GRAY, which is a subroutine member of the library named DEPT100.

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's category value is to be changed, more than one CATEGORY statement 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

```
// LOAD #OLINK
// FILE NAME-$WORK,BLOCKS-40,RETAIN-S
// FILE NAME-$SOURCE,BLOCKS-40,RETAIN-S
// RUN
1 // MODULE NAME-AAAA
2 // GROUP NAME-'AA,BB,CC'
3 // CATEGORY NAME-'AA,BB,DD,EE',VALUE-30
4 // CATEGORY NAME-CC,VALUE-10
// END
```

1 The mainline routine for this program is AAAA, which is a subroutine member of the system library. The program will be cataloged as a load member in the library under the name AAAA. If a load member AAAA already exists, a message will be issued.

2 The overlays, if necessary, are constructed so that AA, BB, and CC are in main storage at the same time.

3 The category value of subroutines AA, BB, DD, and EE is 30 for this link-edit.

4 Routines in the same overlay 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.

Example 5

```
1 // LOAD #OLINK
1 // RUN
2 // PHASE NAME-JIM
3 // OPTIONS MAP-XREF,SUBLIB-'VOLK,MAN'
4 // GROUP NAME-TOM,AREA-USER
5 // MODULE NAME-SUE,LIBRNAME-BUD
  // END
```

1 The Overlay Linkage Editor is loaded from the disk. Because no file statements are given, the Overlay Linkage Editor finds from 24 to 72 blocks of work space on disk for each file ($WORK and $SOURCE).

2 The program is cataloged as a load member in the system library under the name JIM.

3 Prints a cross-reference list, storage map, and messages. Subroutine members are found in libraries named VOLK and MAN. VOLK has the most subroutine members, and because it is entered first, it is searched before library MAN. Subroutine members not found in VOLK or MAN will be searched for in the system library.

4 If TOM is assigned to an overlay, it will appear in the user overlay area.

5 The mainline routine for this program is named SUE, and is a subroutine member of the library named BUD.

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 4-1 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 (coresident 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 coresident 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. However, 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.

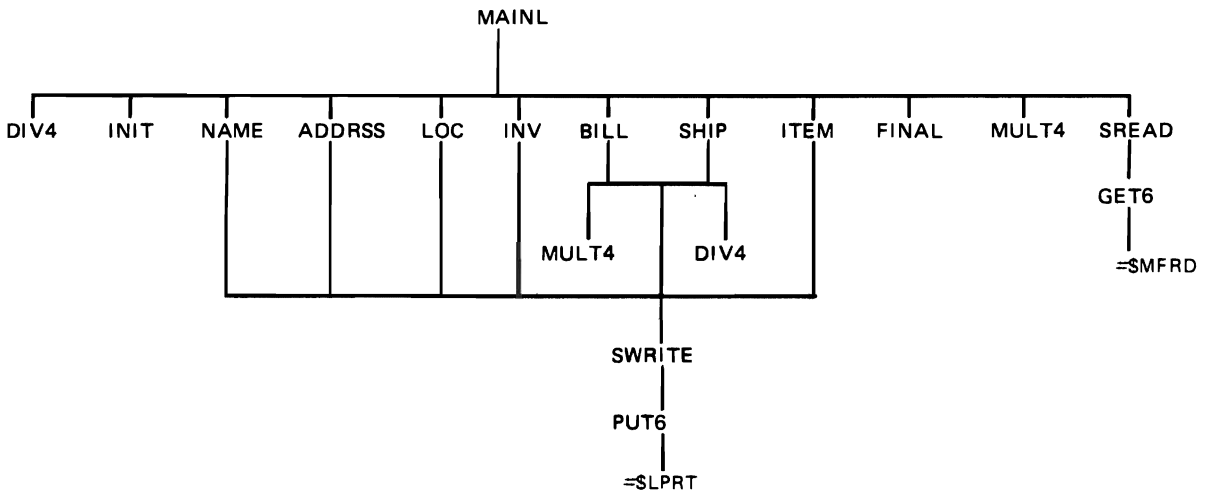
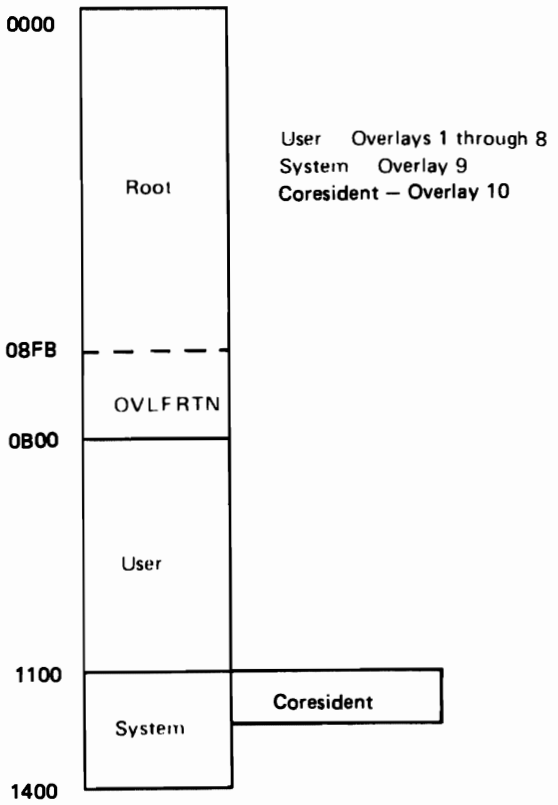


Figure 4-1. Calling Sequence of Modules Link-Edited in Examples 6 and 7

Example 6 (continued)

```
// LOAD #OLINK
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-10
// FILE NAME-$WORK,RETAIN-S,BLOCKS-10
// RUN
// PHASE NAME-OLSMPL
// OPTIONS MAP-XREF,STORE-5K
// MODULE NAME-MAINL
// 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 STORAGE USAGE MAP AND CROSS REFERENCE LIST

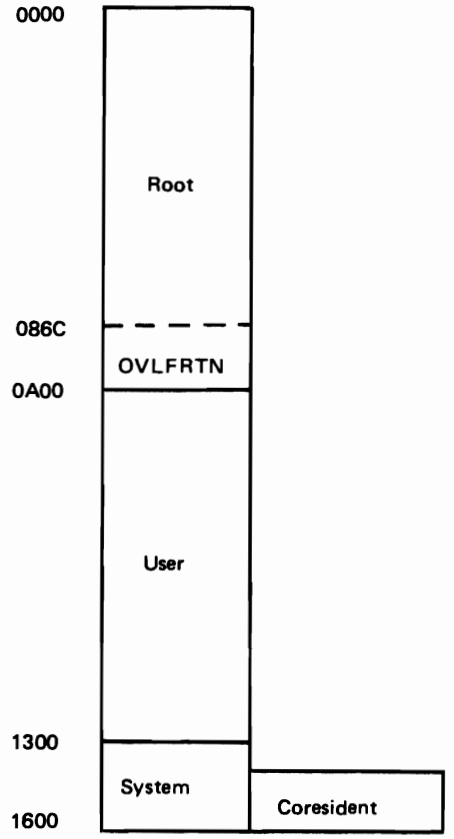
START ADDRESS	OVERLAY NUMBER	AREA	CATEGORY	NAME AND ENTRY	CODE LENGTH HEXADECIMAL	DECIMAL	REFERENCED BY
0000			0	MAINL	082D	2093	
082D			0,4	DIV4	0041	65	MAINL SHIP
086E			0,4	MULT4	003F	63	BILL MAINL
08AD			0,90	FINAL	004E	78	MAINL
08FB				OVLFRTN	0185	389	
0B00	1	U	0,20	NAME	0320	800	MAINL
0F00	1	U	0,136	SWRITE	0192	402	ITEM SHIP BILL INV LOC ADDRSS NAME
0B00	2	U	0,21	ADDRSS	0302	770	MAINL
0F00	2	U	0,136	SWRITE	0192	402	ITEM SHIP BILL INV LOC ADDRSS NAME
0B00	3	U	0,22	LOC	028A	650	MAINL
0E00	3	U	0,136	SWRITE	0192	402	ITEM SHIP BILL INV LOC ADDRSS NAME
0B00	4	U	0,23	INV	02E4	740	MAINL
0E00	4	U	0,136	SWRITE	0192	402	ITEM SHIP BILL INV LOC ADDRSS NAME
0B00	5	U	0,24	BILL	02CA	714	MAINL
0E00	5	U	0,136	SWRITE	0192	402	ITEM SHIP BILL INV LOC ADDRSS NAME
0B00	6	U	0,25	SHIP	0306	774	MAINL
0F00	6	U	0,136	SWRITE	0192	402	ITEM SHIP BILL INV LOC ADDRSS NAME
0B00	7	U	0,26	ITEM	0296	662	MAINL
0E00	7	U	0,136	SWRITE	0192	402	ITEM SHIP BILL INV LOC ADDRSS NAME
0B00	8	U	0,138	SREAD	01FA	506	MAINL
1100	9	S	0,6	GET6	0044	68	SREAD
1144	9	S	0,6	PUT6	003A	58	SWRITE
117E	9	S	6	$$MFRD	0145	325	GET6
12C3	9	S	6	$$LPRT	00FB	251	PUT6
1100	10	C	0,90	INIT	00D0	208	MAINL

```
SYS-3130 I OLSMPL MODULE'S MAIN STORAGE SIZE IS
           5120 DECIMAL
SYS-3131 I 0004 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3132 I THE NONOVERLAY MAIN STORAGE SIZE IS
           9466 DECIMAL
SYS-3134 I OLSMPL MODULE IS CATALOGED AS A LOAD MEMBER
           #LIBRARY IS THE LIBRARY NAME
                55 TOTAL NUMBER OF LIBRARY SECTORS
```

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 coresident area (user module in the system area). GROUP statements were entered to increase the size of the user overlay area.

```
// LOAD #OLINK
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-10
// FILE NAME-$WORK,RETAIN-S,BLOCKS-10
// RUN
// PHASE NAME-OLSMPL
// OPTIONS MAP-XREF,STORE-H5K
// MODULE NAME-MAINL
// CATEGORY NAME-'MULT4,DIV4',VALUE-4
// CATEGORY NAME-'GET6,PUT6',VALUE-6
// CATEGORY NAME-SWRITE,VALUE-136
// CATEGORY NAME-SREAD,VALUE-138
// CATEGORY NAME-NAME,VALUE-20
// CATEGORY NAME-ADDRSS,VALUE-21
// CATEGORY NAME-LOC,VALUE-22
// CATEGORY NAME-INV,VALUE-23
// CATEGORY NAME-BILL,VALUE-24
// CATEGORY NAME-SHIP,VALUE-25
// CATEGORY NAME-ITEM,VALUE-26
// CATEGORY NAME-'INIT,FINAL',VALUE-90
// GROUP NAME-'NAME,LOC'
// GROUP NAME-'ADDRSS,INV'
// GROUP NAME-'BILL,SHIP'
// END
```



OVERLAY LINKAGE EDITOR STORAGE USAGE MAP AND CROSS REFERENCE LIST

START ADDRESS	OVERLAY NUMBER	AREA	CATEGORY	NAME AND ENTRY	CODE LENGTH HEXADECIMAL	DECIMAL	REFERENCED BY				
0000			0	MAINL	082D	2093					
082D			0,4	MULT4	003F	63	BILL	MAINL			
086C				OVLFRTN	0165	357					
0A00	1	U	0,20	NAME	0320	800	MAINL				
0E00	1	U	0,136	SWRITE	0192	402	ITEM	SHIP	BILL	INV	LOC
							ADDRSS	NAME			
0F92	1	U	0,22	LOC	028A	650	MAINL				
0A00	2	U	0,21	ADDRSS	0302	770	MAINL				
0E00	2	U	0,136	SWRITE	0192	402	ITEM	SHIP	BILL	INV	LOC
							ADDRSS	NAME			
0F92	2	U	0,23	INV	02E4	740	MAINL				
0A00	3	U	0,24	BILL	02CA	714	MAINL				
0D00	3	U	0,136	SWRITE	0192	402	ITEM	SHIP	BILL	INV	LOC
							ADDRSS	NAME			
0E92	3	U	0,25	SHIP	0306	774	MAINL				
0A00	4	U	0,26	ITEM	0296	662	MAINL				
0D00	4	U	0,136	SWRITE	0192	402	ITEM	SHIP	BILL	INV	LOC
							ADDRSS	NAME			
0F00	4	U	0,138	SREAD	01FA	506	MAINL				
1300	5	S	0,4	DIV4	0041	65	MAINL	SHIP			
1300	6	S	0,6	GET6	0044	68	SREAD				
1344	6	S	0,6	PUT6	003A	58	SWRITE				
137E	6	S	6	$$MFRD	0145	325	GET6				
14C3	6	S	6	$$LPRT	00FB	251	PUT6				
1400	7	C	0,90	INIT	00D0	208	MAINL				
14D0	7	C	0,90	FINAL	004E	78	MAINL				

```
SYS-3130 I OLSMPL MODULE'S MAIN STORAGE SIZE IS
           5632 DECIMAL
SYS-3131 I 0004 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3132 I THE NONOVERLAY MAIN STORAGE SIZE IS
           9466 DECIMAL
SYS-3134 I OLSMPL MODULE IS CATALOGED AS A LOAD MEMBER
           #LIBRARY IS THE LIBRARY NAME
               51 TOTAL NUMBER OF LIBRARY SECTORS
```

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 4-2 shows the calling sequence of the modules within the program.

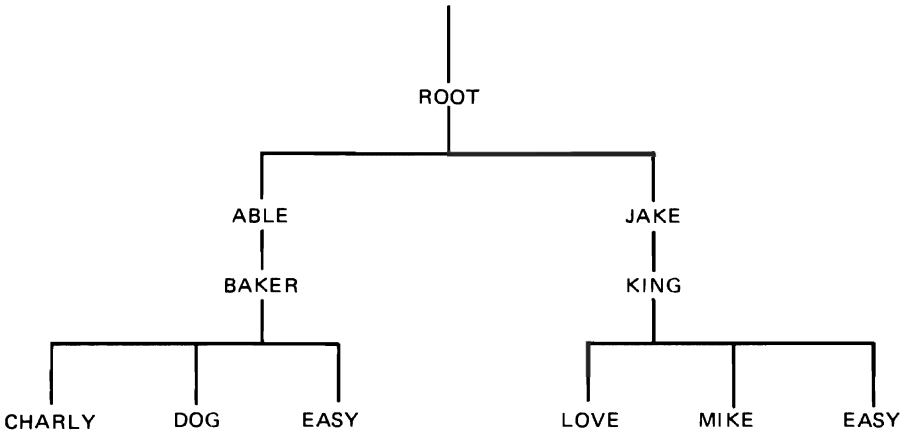
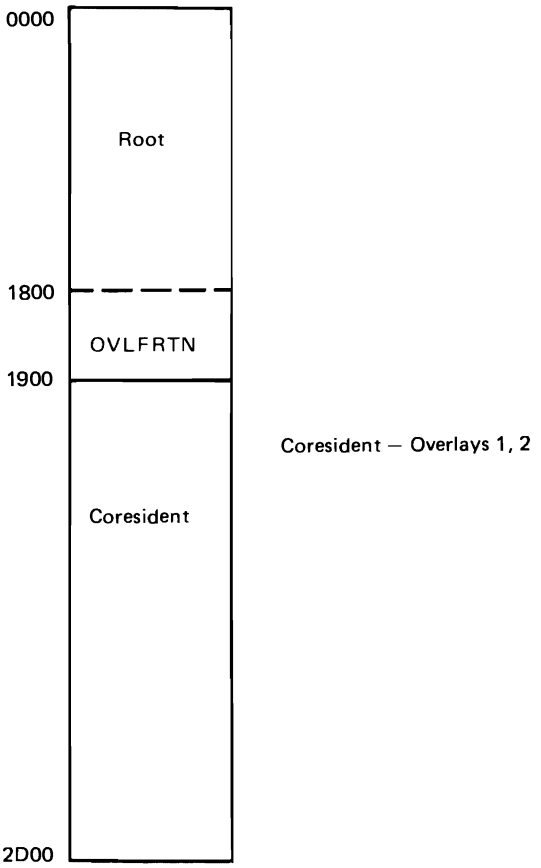


Figure 4-2. Calling Sequence of Modules Linked in Examples 8 through 11

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 assigned to both segments, but space was available in the root so EASY was included in the root segment.

```
// LOAD #OLINK
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-50
// FILE NAME-$WORK,RETAIN-S,BLOCKS-50
// RUN
// PHASE NAME-RIZZI
// OPTIONS MAP-XREF,STORE-T11K
// MODULE NAME-ROOT
// CATEGORY NAME-'ABLE,JAKE',VALUE-8
// END
```

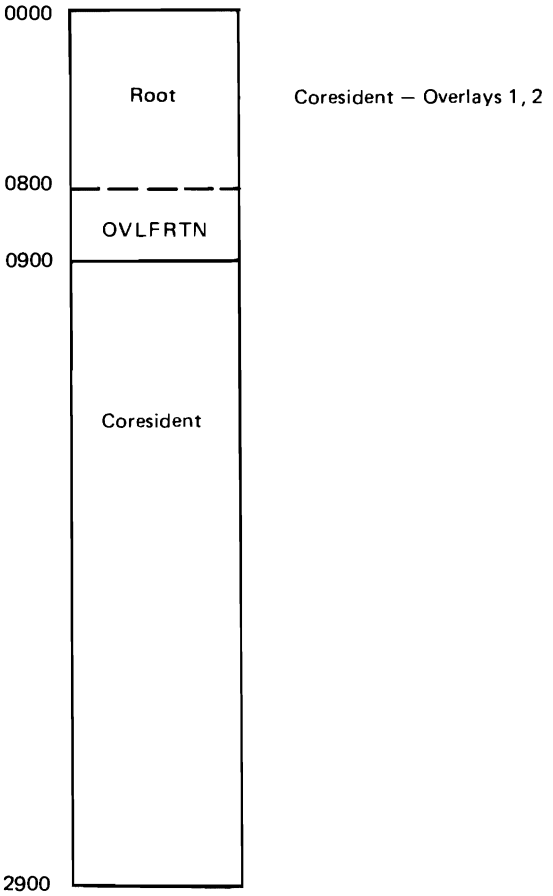



```
                OVERLAY LINKAGE EDITOR STORAGE USAGE MAP AND CROSS REFERENCE LIST

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

 0000                    0      ROOT          0800    2048
 0800                    0      BAKER         0400    1024    ABLE
 0C00                    0      KING          0400    1024    JAKE
 1000                   37      EASY          0800    2048    KING    BAKER
 1800                           OVLFRTN       00C9     201
 1900       1    C      0,8     ABLE          0328     808    ROOT
 1C28       1    C      37      CHARLY        0800    2048    BAKER
 2428       1    C      37      DOG           0800    2048    BAKER
 1900       2    C      0,8     JAKE          0400    1024    ROOT
 1D00       2    C      37      LOVE          0800    2048    KING
 2500       2    C      37      MIKE          0800    2048    KING

SYS-3130 I RIZZI  MODULE'S MAIN STORAGE SIZE IS
          11520 DECIMAL
SYS-3131 I 0004 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3132 I THE NONOVERLAY MAIN STORAGE SIZE IS
          16168 DECIMAL
SYS-3134 I RIZZI  MODULE IS CATALOGED AS A LOAD MEMBER
           #LIBRARY IS THE LIBRARY NAME
               67 TOTAL NUMBER OF LIBRARY SECTORS
```

Example 9

All modules have an overlay category (nonzero). 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.

```
// LOAD #OLINK
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-50
// FILE NAME-$WORK,RETAIN-S,BLOCKS-50
// RUN
// PHASE NAME-RIZ
// OPTIONS MAP-XREF,STORE-H10K
// MODULE NAME-ROOT
// CATEGORY NAME-'BAKER,KING',VALUE-8
// CATEGORY NAME-'ABLE,JAKE',VALUE-8
// END
```

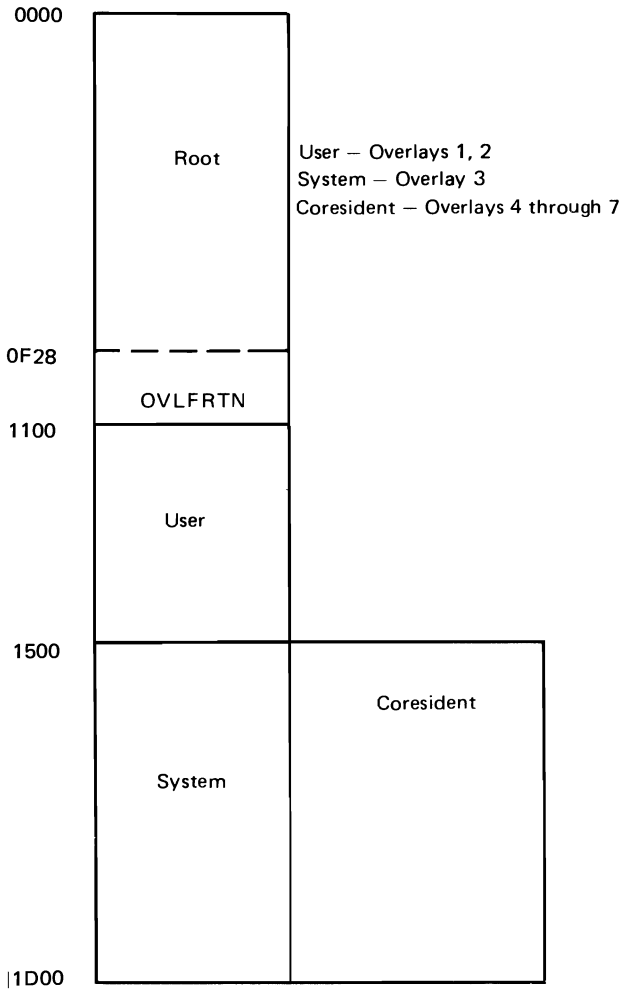



```
                OVERLAY LINKAGE EDITOR STORAGE USAGE MAP AND CROSS REFERENCE LIST

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

 0000                    0      ROOT          0800    2048
 0800                           OVLFRTN       00F5     245
 0900       1    C      0,8    ABLE          0328     808    ROOT
 0C28       1    C      0,8    BAKER         0400    1024    ABLE
 1028       1    C      37     CHARLY        0800    2048    BAKER
 1828       1    C      37     DOG           0800    2048    BAKER
 2028       1    C      37     EASY          0800    2048    KING    BAKER
 0900       2    C      0,8    JAKE          0400    1024    ROOT
 0D00       2    C      0,8    KING          0400    1024    JAKE
 1100       2    C      37     LOVE          0800    2048    KING
 1900       2    C      37     MIKE          0800    2048    KING
 2100       2    C      37     EASY          0800    2048    KING    BAKER

SYS-3130 I RIZ    MODULE'S MAIN STORAGE SIZE IS
          10496 DECIMAL
SYS-3131 I 0004 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3132 I THE NONOVERLAY MAIN STORAGE SIZE IS
          16168 DECIMAL
SYS-3134 I RIZ    MODULE IS CATALOGED AS A LOAD MEMBER
           #LIBRARY IS THE LIBRARY NAME
               75 TOTAL NUMBER OF LIBRARY SECTORS
```

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 dependant and are assigned to user overlay segments. The remaining four modules are I/O independant and are assigned to coresident overlay segments.

```
// LOAD #OLINK
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-50
// FILE NAME-$WORK,RETAIN-S,BLOCKS-50
// RUN
// PHASE NAME-EV2
// OPTIONS MAP-XREF,STORE-H7K
// MODULE NAME-ROOT
// CATEGORY NAME-'BAKER,KING',VALUE-8
// CATEGORY NAME-EASY,VALUE-2
// END
```



OVERLAY LINKAGE EDITOR STORAGE USAGE MAP AND CROSS REFERENCE LIST

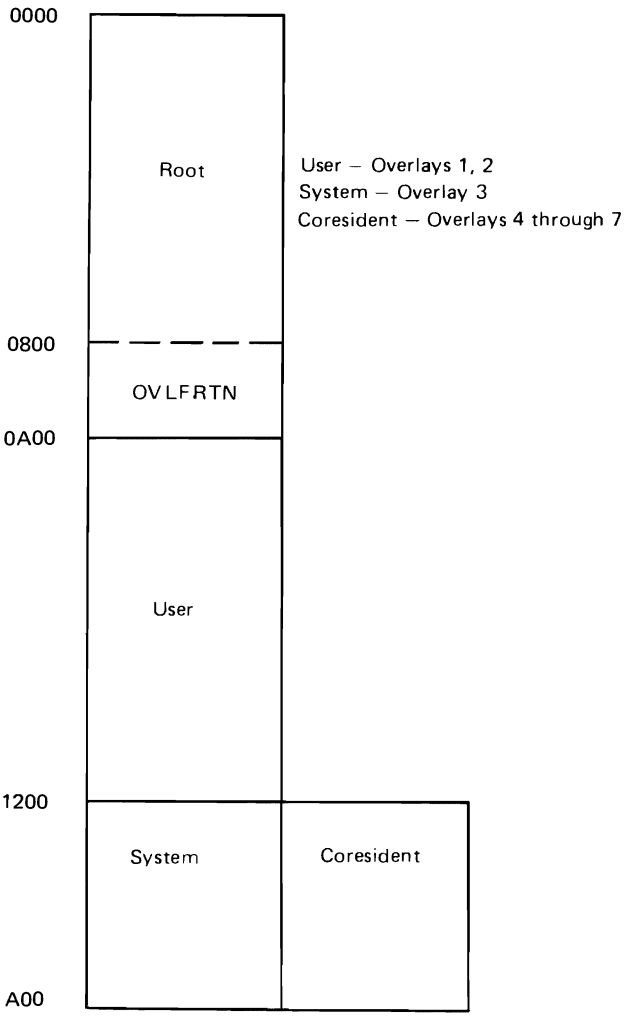
START ADDRESS	OVERLAY NUMBER	AREA	CATEGORY	NAME AND ENTRY	CODE LENGTH HEXADECIMAL	CODE LENGTH DECIMAL	REFERENCED BY	
0000			0	ROOT	0800	2048		
0800			0	ABLE	0328	808	ROOT	
0B28			0	JAKE	0400	1024	ROOT	
0F28				OVLFRTN	00F7	247		
1100	1	U	0,8	BAKER	0400	1024	ABLE	
1100	2	U	0,8	KING	0400	1024	JAKE	
1500	3	S	37,2	EASY	0800	2048	KING	BAKER
1500	4	C	37	CHARLY	0800	2048	BAKER	
1500	5	C	37	DOG	0800	2048	BAKER	
1500	6	C	37	LOVE	0800	2048	KING	
1500	7	C	37	MIKE	0800	2048	KING	

```
SYS-3130 I EV2    MODULE'S MAIN STORAGE SIZE IS
          7424 DECIMAL
SYS-3131 I 0004 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3132 I THE NONOVERLAY MAIN STORAGE SIZE IS
         16168 DECIMAL
SYS-3134 I EV2    MODULE IS CATALOGED AS A LOAD MEMBER
           #LIBRARY IS THE LIBRARY NAME
               72 TOTAL NUMBER OF LIBRARY SECTORS
```

Example 11

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

```
// LOAD #OLINK
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-50
// FILE NAME-$WORK,RETAIN-S,BLOCKS-50
// RUN
// PHASE NAME-EV28
// OPTIONS MAP-XREF,STORE-T6K
// MODULE NAME-ROOT
// CATEGORY NAME-'ABLE,BAKER,JAKE,KING',VALUE-8
// CATEGORY NAME-EASY,VALUE-2
// END
```




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

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

 0000                    0      ROOT          0800    2048
 0800                           OVLFRTN       010D     269
 0A00       1    U      0,8    ABLE          0328     808    ROOT
 0D28       1    U      0,8    BAKER         0400    1024    ABLE
 0A00       2    U      0,8    JAKE          0400    1024    ROOT
 0E00       2    U      0,8    KING          0400    1024    JAKE
 1200       3    S     37,2    EASY          0800    2048    KING    BAKER
 1200       4    C      37     CHARLY        0800    2048    BAKER
 1200       5    C      37     DOG           0800    2048    BAKER
 1200       6    C      37     LOVE          0800    2048    KING
 1200       7    C      37     MIKE          0800    2048    KING

SYS-3130 I EV28   MODULE'S MAIN STORAGE SIZE IS
           6656 DECIMAL
SYS-3131 I 0004 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3132 I THE NONOVERLAY MAIN STORAGE SIZE IS
          16168 DECIMAL
SYS-3134 I EV28   MODULE IS CATALOGED AS A LOAD MEMBER
           #LIBRARY IS THE LIBRARY NAME
               72 TOTAL NUMBER OF LIBRARY SECTORS
```

Appendix A. Messages

The error conditions diagnosed by the Overlay Linkage Editor are included in this section. Messages are printed on the system printer, and if requested by the compiler, the messages are also added to the Diagnosed Source File. (See the *IBM System/34 FORTRAN IV Reference Manual*, SC21-7706, for further information about this file.) If MAP–NO is specified on the OPTIONS statement, information messages are not printed.

Messages Requested by the Compiler

If a Diagnosed Source File has been requested and a subroutine member is to be cataloged, one of the following messages is placed in the fifth record of the Diagnosed Source File (when this record is displayed, it appears as the fourth record on the screen):

- MODULE WAS NOT CATALOGED AS A SUBROUTINE MEMBER
  (Issued because the linkage editor aborted)
- SYS–31nn WAS DISPLAYED, MODULE WAS NOT CATALOGED AS A SUBROUTINE MEMBER
- nnnn DECIMAL IS THE CODE LENGTH OF THE SUBROUTINE MEMBER
  (Appears when the subroutine member is cataloged in the library)

If a subroutine member was not requested to be cataloged, the record area is blank.

If a Diagnosed Source File has been requested and a load member is to be cataloged, one of the following messages is placed in the sixth record of the Diagnosed Source File (when this record is displayed, it appears as the fifth record on the screen):

- MODULE WAS NOT CATALOGED AS A LOAD MEMBER
  (Issued because the linkage editor aborted)
- SYS–31nn WAS DISPLAYED, MODULE WAS NOT CATALOGED AS A LOAD MEMBER
- nnnn DECIMAL IS THE MAIN STORAGE SIZE OF THE LOAD MEMBER
  (Appears when the load member is cataloged in the library)

If a load member was not requested to be cataloged, the record area is blank.

There are three classes of messages: informational, warning, and terminal. The informational messages are indicated by an I in print position 10. These messages are printed without halts. Warning messages are indicated by a W in print position 10. A halt with options 0 (continue processing) and 3 (immediate cancel) is issued with warning messages. Terminal messages have a T in position 10 and are issued with a halt with option 3 only.

SYS–3100 W ENTRY LABEL IN OPTIONS STATEMENT 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.

The name on the OPTIONS statement should match one of the names on the storage map.

SYS–3101 T OBJECT MODULE TEXT OUT OF SEQUENCE

nnnn	BEGIN ADDRESS
nnn	OVERLAY NUMBER

An ORG instruction has caused code to overlay other code.

SYS–3102 T

namexx	MODULE HAS AN INVALID EXTERNAL SYMBOL LIST FIELD IN S RECORD
aaaaaa	INVALID EXTERNAL SYMBOL LIST NAME

If using Basic Assembler, you may have specified an invalid EXTRN subtype.

SYS–3103 T

namexx	MODULE HAS AN INVALID RELOCATION DIRECTORY ENTRY IN T RECORD
nnnn	TEXT RECORD ADDRESS

The object module named has a bad text record. The erroneous record has the nnnn address in bytes 3 and 4.

SYS–3104 T ENTRY POINT IN PROGRAM NOT FOUND

nnnn	BEGIN ADDRESS
nnn	OVERLAY NUMBER

There is an unresolved EXTRN to an entry point. Probable user error.

SYS–3105 T namexx SUBROUTINE MEMBER NOT FOUND IN LIBRARY

The name printed in the message was not found by autolink. To correct the error, copy the module to the library or change the EXTRN to the correct module.

SYS–3107 W PROGRAM WILL NOT FIT IN THE MAIN STORAGE SIZE SPECIFIED

Even with overlays, the program will not fit in storage. The storage size is either the size specified on the OPTIONS statement (STORE–annK), the main storage size specified to the compiler, or the default of the current main storage size. If more main storage is not available for execution, change the module sizes or categories to allow a different overlay structure. If this message is issued and no overlays are indicated on the storage map, overlay segments were not generated for one or more of the following reasons:

1. Overlays would not have provided a storage advantage.
2. All of the object modules had category 0.
3. Only one overlay segment was available for an overlay area.

SYS–3108 W namexx MODULE WAS NOT REFERENCED BY AN EXTRN

A module was read from the system input device or was referenced on the MODULE statement but was not referenced by an EXTRN in any of the included object modules. An EXTRN must reference the module name for the Overlay Linkage Editor to determine the program structure.

SYS–3109 W MAINLINE MODULE NAME IS IN A GROUP OR CATEGORY STATEMENT

The mainline module name has no meaning in a GROUP or CATEGORY statement. The name should be removed from the statement.

SYS–3110 W A MODULE IN A GROUP STATEMENT HAS A CATEGORY VALUE LESS THAN 8

The module with the 0-7 category value is ignored when grouping modules. If the user wants the module in a specific overlay segment, he must supply a CATEGORY statement with a value of 8-126, in addition to the GROUP statement. If no module named in a GROUP statement has a category value of 0-7, this message may result from a module being forced to category 0 by the linkage editor. This could be the mainline module, a zero-length module, or a module called from a system module (message 3112). Categories 127, 128, and 255 are treated as category 0.

SYS–3111 W namexx MODULE IN A CATEGORY OR GROUP STATEMENT NOT REFERENCED BY OBJECT PROGRAM

The name in the CATEGORY or GROUP statement is not referenced by any of the included or auto-linked modules. The name should be removed or the correct name determined from the storage map. This message may also appear if a module is named twice in CATEGORY assignments.

SYS–3112 T SYSTEM AREA MODULE CALLS MODULE OF WRONG CATEGORY

namexx SYSTEM AREA MODULE
namexn CATEGORY NUMBER
namexx CALLED MODULE

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.

SYS–3113 W namexx MODULE NAME OR ENTRY POINT IS A DUPLICATE

This message can occur for either of two reasons:

1. If the name is on the storage 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 would 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 MODULE statement. The duplicate name subparameter can be removed.

SYS–3114 T namexx MODULE HAS AN INVALID EXTERNAL SYMBOL LIST NUMBER IN TEXT RECORD
nnnn LOAD ADDRESS

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.

SYS–3115 T ENTRY POINT IS NOT RELATIVE ZERO IN A MODULE WITH COMMON AREA

The entry point must be the first byte of the module because the start control address on the header record 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.

SYS–3130 I namexx MODULE'S MAIN STORAGE IS
nnnn DECIMAL

The module named requires the amount of main storage given by nnnn to execute.

SYS–3131 I xxxx IS THE START CONTROL - ADDRESS OF THIS MODULE

The entry point of the root segment is specified by xxxx.

SYS–3132 I THE NONOVERLAY MAIN STORAGE SIZE
nnnnn IS DECIMAL

The amount of main storage this program needs to execute without overlays is nnnnn.

SYS–3133 I namexx MODULE IS CATALOGED AS A SUBROUTINE MEMBER
nnnnnnnn IS THE LIBRARY NAME
nnnn TOTAL NUMBER OF LIBRARY SECTORS
nnn CATEGORY NUMBER

This message is issued when the compiler entry is used to catalog a subroutine member.

SYS–3134 I namexx MODULE IS CATALOGED AS A LOAD MEMBER
nnnnnnnn IS THE LIBRARY NAME
nnnn TOTAL NUMBER OF LIBRARY SECTORS

This describes the load module cataloged into the library.

SYS–3135 I namexx MODULE'S CODE LENGTH IS
nnnn DECIMAL

Describes the number of bytes in the object module cataloged to the library. This size does not include:

1. Bytes reserved for common
2. Bytes bypassed for boundary alignment
3. Bytes used by routines referenced by EXTRNs

Appendix B. Object Modules

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

Type	Meaning
S-type	External symbol list (ESL) records
T-type	Text-relocation directory (RLD) records
E-type	End record

External Symbol List (ESL) Records

External symbol list fields occur within S-type records to define areas within an object module and contain external references to other modules. The Overlay Linkage Editor accepts the following types of external symbol list fields:

- External reference
- Module name
- Entry point

An S-type record contains up to five ESL fields in any combination of the above three types.

External Reference

External reference (EXTRN) fields 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)

EXTRN Subtypes 0 and 128: This external symbol list 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 $WORK file (object modules are placed on $WORK by a language processor or by #OLI2) to find a module. If the module is not found in the $WORK file, autolink is performed. Autolink means that the Overlay Linkage Editor searches the object library members to resolve all unresolved external references to module names. If the external reference cannot be resolved, a message is issued. External references to *entry points* are not resolved by autolink.

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.

EXTRN Subtype 4: This external symbol list record specifies a space allocation for a GLOBAL COMMON area. This area is allocated at the start of the user's program area. 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.

EXTRN Subtype 5: The Overlay Linkage Editor allocates an area of main storage for the COMMON area either at the beginning of the user's program area 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.

EXTRN Subtype 6: The function of the conditional external reference is the same as for the external reference. That is, autolink is performed if necessary, except that if the Overlay Linkage Editor still cannot resolve (find) the referenced name, the conditional external reference is ignored and remains unresolved.

Module Name

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

Entry Point

This external symbol list field provides the entry point name in the module and the address of the entry point in hexadecimal.

Text Relocation Directory (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 must not overlap from one text record to the next.

Each record is 64 bytes long in the following format:

Byte	Contents
0	T (denotes text relocation directory 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 relocation directory entries are inserted beginning in byte 63 from right to left. Unused bytes (at least one) between the text and the relocation directory contain X'00'. The relocation directory entry points to the right end of the address (displaced from beginning of text).

One-Byte RLD

Each 1-byte relocation directory entry contains the following:

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

Three-Byte RLD

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

Byte	Bit	Meaning
1-2 (leftmost)	ALL	Relative EXTRN external symbol list count so name of the EXTRN can be found. Relative external symbol list count starts with 0001.
3 (rightmost)	0	1 = entry points to an EXTRN with a known displacement.
	1	1 = 3-byte relocation directory entry.
	2-7	Displacement from the leftmost text byte in the record. Displacement count starts with 00.

Three-byte relocation directory entries are processed like 1-byte entries except that the base address is the address defined in the external symbol list entry corresponding to the relative EXTRN count.

End Record

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

Appendix C. Performance Improvements

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

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

2. Have the linkage editor locate the needed object modules via autolink rather than by you supplying multiple MODULE statements.

Appendix D. How to Specify the OLINK Procedure

The OLINK procedure resides in the system library (#LIBRARY). It can be used to call the Overlay Linkage Editor to create a load module. Following is the format of the procedure and a chart showing what each parameter is used for in the procedure:

```
OLINK module name, [object library   ], [load module name ]
                   [#LIBRARY         ]  [module name      ]

                  , [load module library], [attribute1] , [attribute2]
                    [#LIBRARY           ]

                  , [mrtmax value], [user subrlib1, user subrlib2]
                    [0           ]  [#LIBRARY                   ]

                  , [YES]
                    [NO ]
```

Note: If the module name (a required entry) is not entered, a prompt screen will appear. Each parameter appears on the prompt screen, along with its default. Any or all parameters may be keyed.

Parameter	Required/ Optional	Type	Specify	Default
1	Required	Name of object module	Name of object module	None
2	Optional	Library with object module	Library name where search for module	#LIBRARY
3	Optional	Name of load module	Name to put on load module	Object name
4	Optional	Library to place load module	Where to place load module	#LIBRARY
5 and 6	Optional	Attribute	Enter one of the following per parameter (maximum of two): COM (Common) DED (Dedicated) NEP (Never-ending program) NEX (Not executable) NIQ (Noninquirable) NSW (Nonswappable) LSC (Load only from system console) SIS (Scientific mode) SRQ (Source required) USC (Utility control statements)	Null
7	Optional	MRTMAX	Number of terminals available to allocate (0 to 255)	0
8 and 9	Optional	User subroutine library	Where to find user subroutine members (maximum of 2 parameters)	#LIBRARY
10	Optional	Whether to place on input job queue	Whether to place on input job queue	No

Examples

1. An object module, PROGA, resides in the user library called USERLIB. The user wants an executable load module from this. The load module name is LOADA, and it will be placed in the USERLIB.

   ```
   OLINK PROGA,USERLIB,LOADA,USERLIB
   ```

2. An object module, SAMPL, resides in the system library #LIBRARY. The user wants an executable load module of the same name in the same library.

   Note: By specifying only the object name, the defaults for the other parameters determine that the system library is to be searched, the load module name will be the same as the object module name, and the load module will be placed in the system library.

   ```
   OLINK SAMPL
   ```

Glossary

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

autolink: A process whereby the Overlay Linkage Editor searches the object library for object modules to resolve all unresolved *external references* to module names.

COMMON: An area of main storage that contains data areas common to more than one routine within one program. This area is not saved when control is passed from one program to another.

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.

diagnosed source file: An optional disk file of source input, containing error and diagnostic messages. This file can be viewed or updated from a display station.

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 that contains data areas common to more than one program. This area is saved when control is passed from one program to another.

heading: A title printed at the top of a column or page.

INVOKE: A 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*.

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

overlay: (1) To repeatedly use the same blocks of main 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*.

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*.

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.

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

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

Index

IBM

Technical Newsletter

This Newsletter No. SN21-7963
Date 14 July 1978

Base Publication No. SC21-7707-0
File No. S34-36

Previous Newsletters None

IBM System/34
Overlay Linkage Editor
Reference Manual

This technical newsletter, a part of release 02 modification 00 of the IBM System/34 System Support Program (Program Product 5726-SS1), provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Title Page, Preface	A-1, A-2
iii, iv	A-2.1, A-2.2 (added)
1-1, 1-2	D-1, D-2
2-1 through 2-6	D-3, D-4 (added)
4-5 through 4-8	E-1, E-2 (added)
	X-1 through X-4

Changes to text and illustrations are indicated by a vertical line at the left of the change.

Summary of Amendments

- Addition of Diagnosed Source File messages
- Appendix added to explain OLINK procedure
- Miscellaneous technical changes

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

Note: All comments and suggestions become the property of IBM.

Name _____

Address _____

_____

● No postage necessary if mailed in the U.S.A.

IBM System/34
Overlay Linkage Editor
Reference Manual

SC21-7707-0

Cut Along Line

Fold

Fold

FIRST CLASS
PERMIT NO. 40
ARMONK, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

System/34 Overlay Linkage Editor Reference Manual (File No. S34-36) Printed in USA SC21-7707-0

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

System/34 Overlay Linkage Editor Reference Manual (File No. S34-36) Printed in USA SC21-7707-0