File No. S360-31
Form C28-6538-3

OS

IBM Systems Reference Library

IBM System/360 Operating System

Linkage Editor

This publication provides programmers and systems analysts with the information necessary to make effective use of the linkage editor of IBM System/360 Operating System. Included are descriptions of the functions performed automatically by the linkage editor as well as those performed in response to control statements prepared by the programmer.

The linkage editor combines and edits modules to produce a single module that can be loaded by the control program. The linkage editor operates as a processing program rather than as a part of the control program.

PREFACE

This publication provides programmers and systems analysts with information on the operation and use of the linkage editor of IBM System/360 Operating System. It is part of an integrated library of IBM System/360 Operating System publications. Other publications that are required for an understanding of linkage editor processing are:

> IBM System/360 Operating System: Introduction, Form C28-6534
>
> IBM System/360 Operating System: Concepts and Facilities, Form C28-6535

The processing information required to perform the linking functions of the linkage editor is contained in the sections of this publication titled "Object and Load Modules," "Input Sources, Intermediate and Output Storage," "Linking Input Modules," "Error Diagnostics and Processing Options," and "Diagnostic Output and Special Processing."

The program modification functions are discussed in the sections titled "Editing Object and Load Modules," "Designing An Overlay Program," and "Specifying Additional Processing."

Third Edition (October 1966)

This is a reprint of C28-6538-2 incorporating changes issued in Technical Newsletter N28-2130, dated June 1, 1966.

Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

IGURES

ABLE

The linkage editor is one of the processing programs of IBM System/360 Operating System. It is a service program used to prepare loadable programs from the output of language translators, such as FORTRAN, COBOL, Report Program Generator, Assembler Language, or Programming Language/I. Linkage editor processing is the necessary step that follows source program assembly or compilation.

Two levels of the linkage editor program are available: level E, which is designed to process programs on all systems, and level F, which is designed to operate in 44K or more of main storage. A compatibility option is provided to ensure that programs processed by the level F linkage editor can be reprocessed by the level E linkage editor.

FUNCTIONS OF THE LINKAGE EDITOR

The basic function of the linkage editor is the linking of separately assembled or compiled modules of a program into one load module. The load module is in a format suitable for loading and execution by the control program of the operating system.

Although this linking or combining of modules is its primary function, the linkage editor also:

1. Incorporates modules from data sets other than its primary input, either automatically or upon request.
2. Constructs an overlay program for loading by the control program.
3. Aids program modification by replacing, deleting, and rearranging control sections as directed by linkage editor control statements.
4. Reserves storage for the common control sections generated by the Assembler and FORTRAN Languages and static external areas generated by Programming Language/I.
5. Provides processing options and logs diagnostic error messages.

The linkage editor functions are briefly described in the following paragraphs.

MODULE LINKAGE

Processing by the linkage editor makes it possible for the programmer to divide his program into several modules, each containing one or more control sections. The modules can be separately assembled or compiled. The linkage editor combines these modules into one output module (Figure 1), called a load module, with contiguous storage addresses. The output module is placed in a library (partitioned data set).

The linkage editor can also process its input to form more than one load module within a single job step. Each load module is placed in the library under a unique name.

ADDITIONAL INPUT SOURCES

Standard subroutines from a library can be included in the output module, thus reducing the work in coding programs. The programmer can specify that a subroutine be included at a particular time during the processing of his program by using a linkage editor control statement. When the linkage editor processes a program containing this statement, the module containing the subroutine is retrieved from the indicated input source and made an integral part of the output load module (Figure 2).

Symbols that are undefined after all input modules have been processed cause the automatic library call mechanism to search for modules that will resolve the references. When a module of that name is found, the module that has that name is processed by the linkage editor and becomes part of the output module (Figure 2).

PROGRAMS IN AN OVERLAY STRUCTURE

To minimize main storage requirements, the programmer can organize his program into an overlay structure by dividing it into segments according to the functional relationships of control sections. Two or more segments that need not be in main storage at the same time can be assigned the same relative storage addresses, and can be loaded at different times.

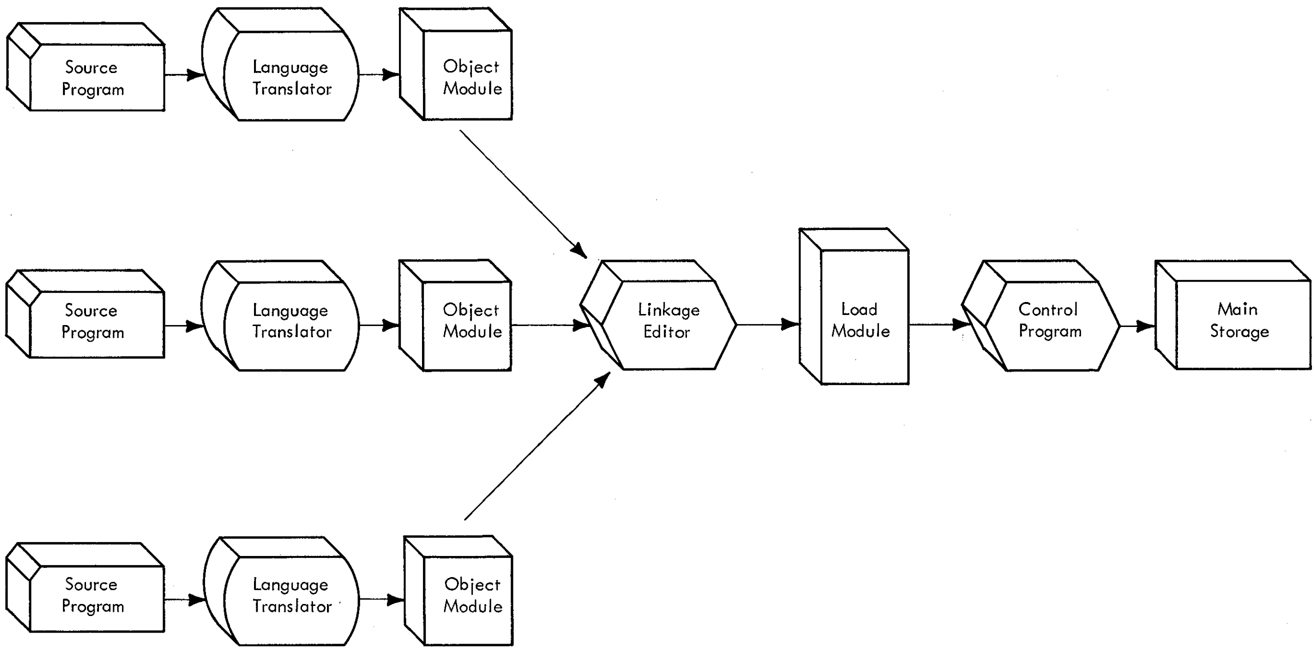
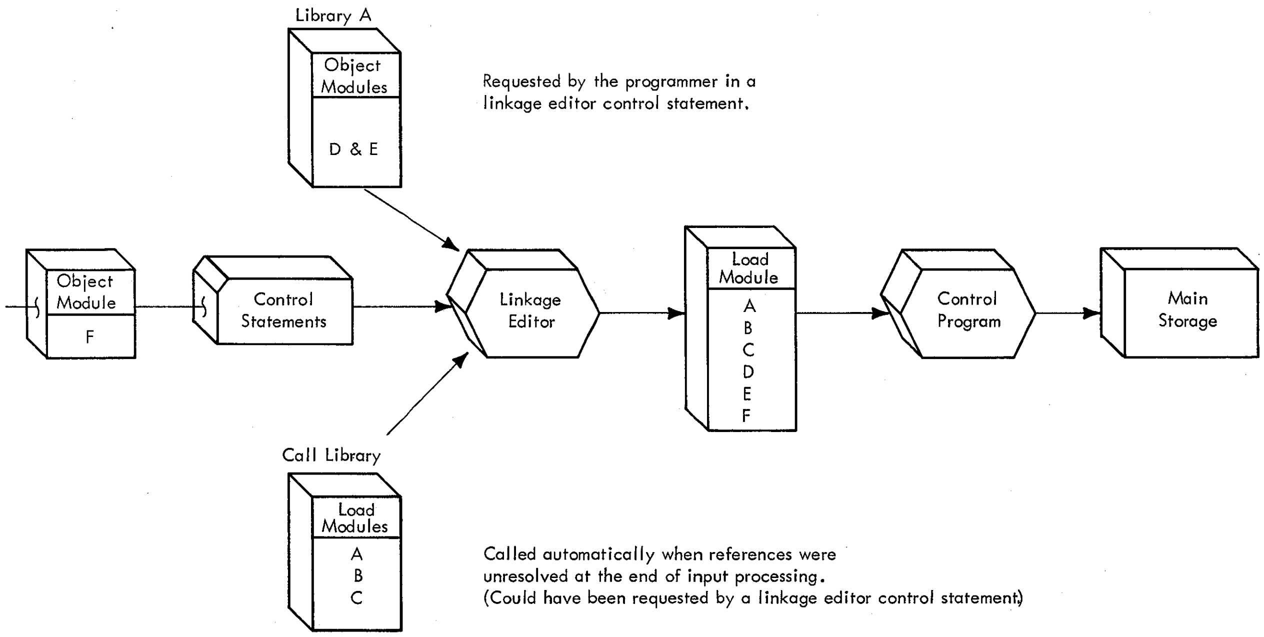


Figure 1. Linkage Editor Processing - Module Linkage



Library A
Object Modules
D & E
Requested by the programmer in a linkage editor control statement.
Object Module
F
Control Statements
Linkage Editor
Load Module
A B C D E F
Control Program
Main Storage
Call Library
Load Modules
A B C
Called automatically when references were unresolved at the end of input processing.
(Could have been requested by a linkage editor control statement)

Figure 2. Linkage Editor Processing - Additional Data Sources

The programmer uses linkage editor control statements to specify the relationship of segments within the overlay structure. The segments of the load module are placed in a library so that the control program can load them separately when the program is executed.

PROGRAM MODIFICATION

Program modification is facilitated by the editing functions of the linkage editor. When the functions of a program are to be changed, the programmer can modify and recompile affected control sections instead of recompiling the entire source program (Figure 3).

Control sections can be replaced, deleted, or moved as directed by linkage editor control statements, or can be automatically replaced by the linkage editor. External symbols can also be changed or deleted as directed by linkage editor control statements.

The linkage editor processes common control sections generated by the FORTRAN and Assembler Language translators. The static external storage areas generated by PL/I compiler are processed in the same way. The common areas are collected by the linkage editor and a reserved main storage area is provided within the output module.

OPTIONS AND ERROR MESSAGES

The linkage editor can produce a module map or cross-reference table that shows the arrangement of control sections in the output module and indicates how they communicate with one another. A list of the linkage editor control statements that were processed can also be furnished. In addition, special processing options that negate automatic library call or the effect of minor errors can be specified by the programmer.

Throughout processing by the linkage editor, errors and possible error conditions are logged. Serious errors cause the linkage editor to mark the output module "not executable."

Additional diagnostic data is automatically logged by the linkage editor. The data indicates the disposition of the load module in the output module library.

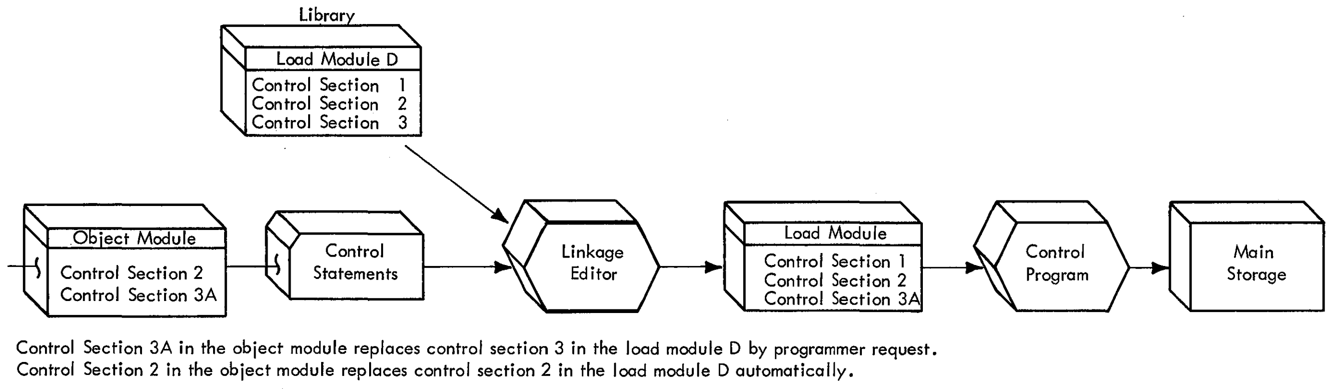


Figure 3. Linkage Editor Processing - Module Editing

PREPARING FOR LINKAGE EDITOR PROCESSING

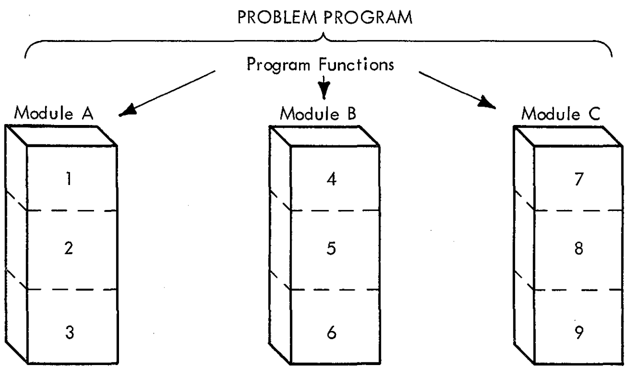
It is necessary for the programmer to be familiar with the data sources and destinations used by the linkage editor, no matter what category of processing he is using (module linking, overlay structuring, module editing, or any combination of these). Diagnostic and special processing options are also available to him.

Once the programmer thoroughly understands the linkage editor functions, the section of this publication titled "Specifying Linkage Editor Processing" may be used for information needed to perform these functions.

OBJECT AND LOAD MODULES

Every problem program is designed to fulfill a particular purpose. To achieve that purpose, the program can generally be divided into logical units that perform specific functions. A logical unit of coding that performs a function, or several related functions, is a module. Ordinarily, separate functions should be programmed into separate modules.

A module is composed of one or more control sections. A control section is a unit of coding (instructions and data) that is, in itself, an entity. All elements of a control section are loaded and executed in a constant relationship to one another. A control section is, therefore, the smallest separately relocatable unit of a program (Figure 4).



Note: 1 through 9 are Control Sections.

Figure 4. The Modules and Control Sections of a Program

A module can be separately assembled or compiled by a language translator. During processing by the language translator, references between control sections within the module are resolved.

OBJECT MODULES

An object module, the output of a language translator, consists of control dictionaries and text (instructions and data). The control dictionaries contain the information necessary to resolve cross-references between control sections and modules. Figure 5 illustrates the contents of an object module. Test symbol dictionary items (an Assembler Language function) to be used by the test translator must be the first items in the object module when testing is specified.

The programmer can specify in an END statement the symbolic address of the instruction with which execution of the module is to begin. As a result, the language translator produces an end statement that marks the end of the object module.

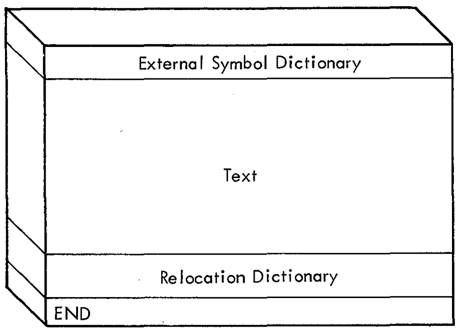


Figure 5. An Object Module

External Symbol Dictionary

The external symbol dictionary (ESD) contains the external symbols that are defined or referred to in the module. An external symbol dictionary entry identifies a symbol and its position within the module. Each entry in the external symbol dictionary is classified as one of the following:

1. external name - is a name that can be referred to by any control section or separately assembled or compiled module. It has a defined value within the module.

   a. control section name - is the symbolic name of a control section. The external symbol dictionary entry specifies the name, the assembled origin, and the length of a control section. The defined value of the symbol is the address of the first byte of the control section.
   b. entry name - is a name within a control section. The external symbol dictionary entry specifies the assembled address of the name and identifies the control section to which it belongs.
   c. blank or named common area - is a control section used to reserve a main storage area (containing no data or instructions) for control sections provided by other modules. The reserved storage areas can also be used as communication centers within a program. The external symbol dictionary entry specifies the name and length of the common area. If it is a blank common area, the name field contains blanks.
   d. private code - is an unnamed control section. The external symbol dictionary entry specifies the assembled address and assigned length of the control section. The name field contains blanks. Since it has no name, it cannot be referred to by other control sections.

2. external reference - is a symbol that is defined as an external name in another separately assembled or compiled module but is referred to in the module being processed. The external symbol dictionary entry specifies the name.

Note: In the Assembler Language, a control section name is defined by a CSECT or START statement; an entry name is specified by an ENTRY statement; a common area is specified by a COM statement; and an external reference is specified by an EXTRN statement or a V-type address constant.

Relocation Dictionary

The relocation dictionary (RLD) contains information about address constants in the module. Each relocation dictionary entry identifies an address constant by indicating its location within a module and the symbol in the external symbol dictionary whose value is used to determine the value of the address constant.

Text

The text (TXT) contains the instructions and data of the module. Instructions and data, as well as their address in the module, form a text item. The text item also indicates the external symbol dictionary entry that defines the control section containing the text.

LOAD MODULES

A load module, the output of the linkage editor, has the same logical structure as an object module. However, an object module is in relocatable format, while a load module is in relocatable and executable format.

Module Contents

A load module is composed of all the edited modules that were the input to the linkage editor. It contains a composite external symbol dictionary and a composite relocation dictionary in addition to the text items (Figure 6). If the load module is to be tested, it may also contain the testing symbol tables used by the test translator.

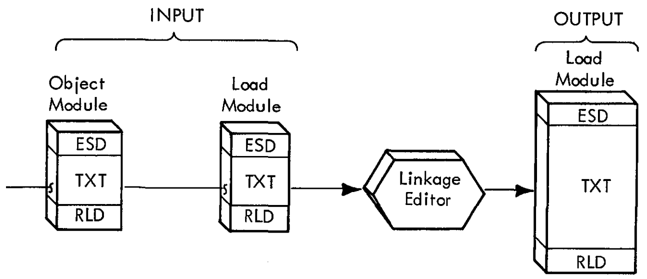


Figure 6. A Load Module Produced by the Linkage Editor

In processing object and load modules, the linkage editor resolves all references between control sections as if they had been assembled as one module. Object modules produced by several different language processors can be used to form one load module under the rules specified by each processor.

The output module produced by the linkage editor contains all the information necessary to load and relocate the module in main storage. It contains information necessary to compute the relocated value of location-dependent address constants. When the load module is placed in the output module library, the name of the module (its member name) and control information describing its attributes are placed in the library directory. The module attributes are used by the control program when the program is loaded for execution. Object modules are assumed to have no attributes.

Module Attributes

Some attributes of a load module can be specified by the programmer; others are specified by the linkage editor as a result of information gathered during processing. In the following list, those attributes marked with an asterisk cannot be specified directly by the programmer.

- Reenterable: A reenterable module can be used by more than one task at the same time; i.e., a task may begin executing a reenterable module before a previous task has finished executing it. A reenterable module is not modified during execution.
- Serially Reusable: A serially reusable module can be executed by only one task at a time. It is self-initializing, since all instructions and data altered during execution are restored before or during execution by another task.
- *Block Format: A module in block format is suitable only for block loading. The control program can load the module only into a contiguous main storage area large enough to contain the complete module.
- Scatter Format: An output module produced by the linkage editor in scatter format is suitable for either block or scatter loading. A module in scatter format can take better advantage of available main storage since the control sections within it can be loaded into noncontiguous areas.
- Not Editable: A module that is not editable has no external symbol dictionary and cannot be reprocessed by the linkage editor.
- Only Loadable: A module that is only loadable can be brought into main storage only by the LOAD macro-instruction.
- Downward Compatible: A module that is downward compatible can be reprocessed by either the level E or the level F linkage editor. The downward compatible option is assumed by the level E linkage editor.
- Overlay: A module that has the overlay attribute specified is placed in overlay structure as directed by linkage editor OVERLAY statements. It is suitable only for block loading and is neither reenterable nor serially reusable.
- Test: This attribute applies only to Assembler Language programs. When a program that is to be tested is being processed, the test symbol dictionary for the test translator is placed in the output module.
- *Not executable: The linkage editor indicates this attribute only if, during processing, errors were found that would prevent the output module from being executed successfully. The control program will not load a "not executable" module. (The module can, however, be modified during a later linkage editor execution.)

Note: The "not editable" and "only loadable" attributes are intended primarily for use by the control program. Use of these attributes by the problem program can degrade the usability of the module.

SEPARATING LOAD MODULES IN ONE JOB STEP

It is possible to create more than one load module in a single job step (a process called multiple load module processing). The linkage editor NAME statement can be used as a delimiter for the input to each load module. Each load module that is formed has a unique name and is placed in the library as a separate member.

INPUT SOURCES, INTERMEDIATE AND OUTPUT STORAGE

The linkage editor can receive its input from several sources that can be processed sequentially, as follows:

- The primary input, which must be contained in a sequential data set that can contain only object modules and linkage editor control statements.

- Additional input, which is contained in sequential or partitioned data sets. A partitioned data set (library) can contain either object modules and control statements, or load modules. (It cannot contain both.)

The level E linkage editor always buffers intermediate data on a direct-access device. The level F linkage editor places intermediate data on a direct-access device only when all the input data cannot be held in the available main storage.

Output of the linkage editor is of two types:

- A load module, which is produced by the linkage editor and placed in a partitioned data set as a named member. Up to five other names (aliases) can be attributed to the module.

- Diagnostic output, which is listed on a printer or placed in a sequential data set.

All data sets processed by the linkage editor must be defined in data definition statements as specified in the publication IBM System/360 Operating System: Job Control Language, Form C28-6539. The data definition name (ddname) for each linkage editor data set is indicated in the following paragraphs.

INPUT SOURCES

The modules that are to be processed by the linkage editor are contained in the following data sets:

1. Primary input data set (principal input).

2. Call library (for automatic library call).

3. Additional data sets (additions to either the primary input or the call library).

The primary input data set is required for all linkage editor job steps. The call library must be defined only if the automatic library call function is to be used. Additional sequential data sets or partitioned data sets are defined only as required by the programmer (Figure 7).

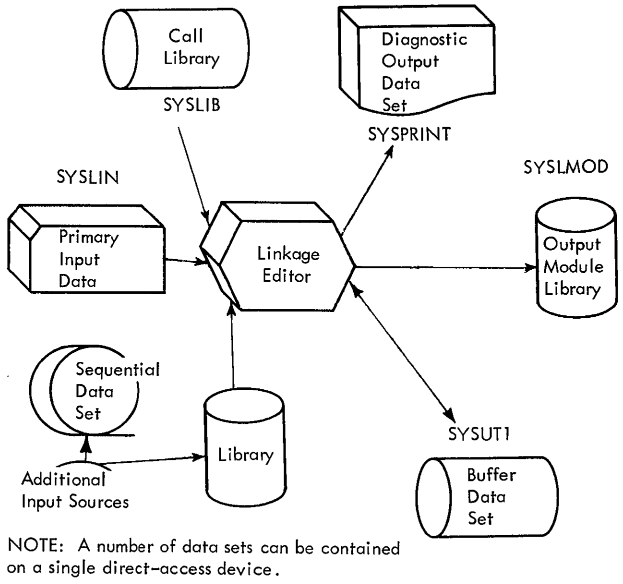


Figure 7. Linkage Editor Input Sources and Output Storage

For input sources the record format (RECFM), block size (BLKSIZE), and, if required, tape recording technique (TRTCH) and density (DEN) fields of the data control block must be made available to the linkage editor. If this information does not exist in the data set control block or tape header label, or if no labels are used, the programmer must specify it on the DD statement defining the sequential data set.

Note: RECFM must be specified as F, FS, FB, or FBS for object modules and U for load modules. The logical record length for object modules must be 80 bytes. The block size for data sets containing object module(s) must be as follows:

- For the level E linkage editor when RECFM is specified as F, FS, FB, or FBS, 80 bytes (one logical record per block).

- For the level F linkage editor when RECFM is specified as F or FB, a block size that does not exceed the capacity of the linkage editor being used; this capacity is the maximum blocking factor for input, as described in Appendix C.1. The size of blocks within the data set can vary, provided that the size does not exceed the linkage editor's capacity.

* For the level F linkage editor when RECFM is specified as FS or FBS, a block size that does not exceed the capacity of the linkage editor being used.

A RECFM specification of FS or FBS must ›e used with caution. All blocks in the lata set containing object modules must be :he same size. This size must be equal to :he specified block size. A truncated ›lock can occur only as the last block in :he data set.

If the DCBBLKSI field of the data con-:rol block contains zero when RECFM is ;pecified as FS or FBS, the level F linkage ›ditor terminates execution.

Primary Input Data Set

The primary input data set can contain only object modules and linkage editor control statements. The modules and control statements are processed sequentially and their order determines the basic order of linkage editor processing during a given execution. However, the order of the control sections after processing does not necessarily reflect the order in which they appeared in the input. Primary input processing stops when the end of the data set is reached. Input processing will continue, however, if the automatic library call mechanism is to process other data sets.

The primary input data set can be a sequential data set or a concatenation of sequential data sets. A library member can be specified on a DD statement so that it can be processed as a sequential data set. For details refer to the publication IBM System/360 Operating System: Job Control Language. The primary input data set must be specified by the ddname SYSLIN.

Additional Data Sets

Modules can be included in the input to the linkage editor from data sets called by the automatic library call mechanism or from other data sets specified by the programmer.

AUTOMATIC CALL LIBRARY: The call library is used by the automatic library call mechanism in the final step of input processing. If the automatic library call function is to be used, the call library must be contained on a direct-access device, and defined accordingly. It must be a partitioned data set and its ddname must be SYSLIB.

If the call library is an object module library, it can contain only object modules and control statements. If it is a load module library, it can contain only load modules.

Modules from libraries other than the call library can be processed by the automatic library call mechanism as directed by the LIBRARY statement.

INCLUDED DATA SETS: The primary input data set and the call library can contain control statements that request the linkage editor to use additional data sets as input. This input can be from sequential data sets, object module libraries, or load module libraries. Each additional data set must be defined in a data definition statement.

Sequential data sets or members of partitioned data sets are included by the linkage editor when an INCLUDE statement is processed.

CAUTION: When concatenated data sets are included, they must be on the same device type. Each data set must contain records of the same format, record size, block size, tape recording technique, and density.

INTERMEDIATE STORAGE

Intermediate data that cannot be contained in main storage during processing is buffered by the linkage editor to the buffer data set. The ddname of the buffer data set is SYSUT1 and the data set is on a direct-access device. This data set must always be defined with a DD statement, whether the linkage editor places data on it or not.

OUTPUT STORAGE

The output of the linkage editor is stored as follows:

- The principal output of the linkage editor is stored in the output module library.
- Error messages, module disposition data, and optional diagnostic output are stored in the diagnostic output data set.

Output Module Library

The load module produced by the linkage editor is always placed in a library as a named member. The output module library can contain many load modules.

The data set name of the library in which the output module is to be stored must be specified by the programmer in a data definition statement. The member name of the output module can be specified in the same data definition statement. If the member name is omitted from a data definition statement, it must be specified in a NAME statement.

The output module can be assigned aliases if the programmer wants the module identified by more than one name or entered at different points when the program is executed. Each name in a load module library must be unique.

The output modules produced in a multiple load module processing job step are stored in the same library.

The output module's library member name and aliases will appear as separate entries with the module attributes in the directory of the library. The ddname of the output module library is SYSLMOD.

Diagnostic Output Data Set

The diagnostic output data set is produced during linkage editor processing. It is a collection of diagnostic and error messages generated by the linkage editor, as well as any diagnostic options requested by the programmer. The diagnostic output data set, given the ddname SYSPRINT, is defined as any data set that can be processed sequentially.

DESIGNING AN OVERLAY PROGRAM

Overlay is a programming technique that minimizes the main storage requirements of a program. To use overlay, the programmer should be familiar with two related techniques:

1. Organization of the program as an overlay structure.
2. Communication with the control program during execution.

OVERLAY TREE STRUCTURE

In order to place a program in an overlay structure, the programmer should be familiar with the following terms:

- A <u>segment</u> is the smallest functional unit (one or more control sections) that can be loaded as one logical entity during execution of the program. The root segment (first segment) remains in main storage throughout execution.
- A <u>path</u> consists of a segment and all segments in the same region between it and the root segment. The root segment is a part of every path in every region. When a segment is in main storage, all segments in its path are also in main storage.
- A <u>region</u> is a contiguous area of main storage within which segments can be loaded independently of paths in other regions. An overlay program can be designed in single or multiple regions.
- A <u>tree</u> is the graphic representation that shows how segments can use main storage at different times. It does not imply the order of execution although the root segment is the first to receive control.

The design of an overlay program requires the organization of the control sections of the program in an overlay tree structure. The tree structure is developed considering:

1. The amount of available main storage.
2. The frequency of use of each control section.
3. The dependencies between control sections.
4. The manner in which control should pass within a path, from one path to another, and from one region to another.

When the programmer has determined the overlay tree structure for a program, he prepares OVERLAY statements that will segment the program in that manner. The use of these control statements is described in "Structuring an Overlay Module."

Single Region Design

To begin constructing an overlay tree, the programmer should select those modules that will receive control at the beginning of execution plus those that should always remain in main storage; these will form the root segment. The rest of the tree can be developed by determining the dependency of the remaining segments and how they can use the same main storage locations at different times during execution.

Module dependency is determined by the requirements of a control section or module for a given routine in another control section. A module is dependent upon a control section to which it branches or whose data it must process. That is, the required control section must be in main storage before execution can continue beyond a given point in the program. Figure 8 illustrates how modules depend on each other, and the paths that result from these dependencies.

The module containing control sections A and B can be used to form the root segment.

The module containing control sections C and D can use the same main storage as the module containing control sections J and K. Segments that use the same main storage area can overlay each other during execution.

The module containing control section E can use the same main storage as the module containing control sections F and G. The module containing control section H can use the same main storage as the module containing control section I. The module containing control section L can use the same main storage as the module containing control sections M and N.

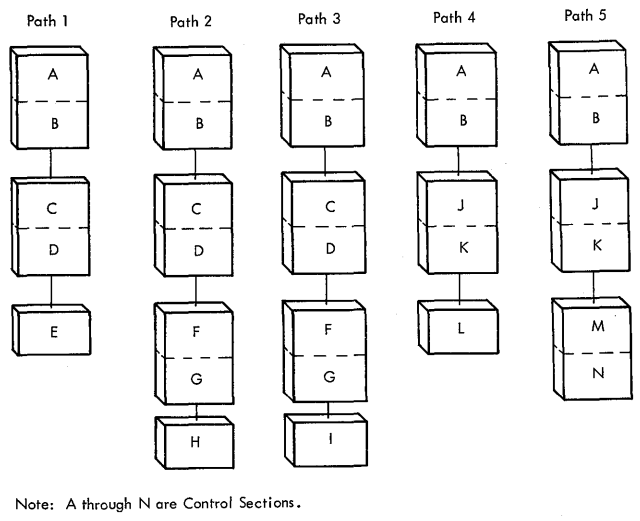


Figure 8. Module Dependency

The resulting overlay tree structure is shown in Figure 9. The longest path in this structure is formed by segments 1, 2, 4, and 5, since, when they are in main storage, the program requires 21,000 bytes. Thus, the minimum main storage requirement for the program is 21,000 bytes. If the program were not put in an overlay structure, it would require 46,000 bytes. The linkage editor will assign the relocatable origin of the root segment (the origin of the program) at 0. The relative origin of each segment will be determined by 0 plus the length of all segments in the path. For example, the origin of segments 3 and 4 is equal to 0 + 5,000 (the length of segment 2) + 6,000 (the length of the root segment).

When a segment is in main storage, all segments in its path are in main storage. segments in its path are in main storage. (Each time a segment is loaded, all segments in its path are also loaded if they are not already in main storage.) In Figure 9, when segment 4 is in main storage, segments 2 and 1 are also in main storage. That does not imply that segment 5 or 6 is in main storage since neither segment is in the path of segment 4.

The position of the segments in an overlay tree structure does not necessarily imply the order in which the segments are executed. A segment can be loaded and overlaid as many times as required by the logic of the program. If a segment is modified during execution, that modification remains only until the segment is overlaid. However, a segment cannot be overlaid by itself.

Segments that can be in main storage simultaneously are considered to be inclusive. Segments in the same region but not in the same path are considered to be exclusive segments; they cannot be in main storage simultaneously.

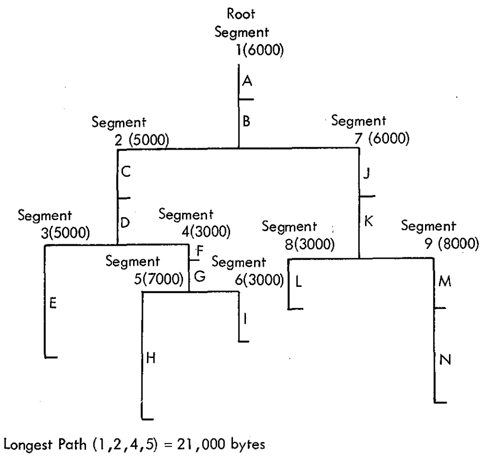


Figure 9. Single Region Overlay Tree Structure

Segments upon which two or more exclusive segments are dependent are called common segments. A segment common to two other segments is part of the path of each segment. In Figure 9, segment 7 is common to segments 8 and 9, but not to segment 2.

Multiple Region Design

In an overlay structure with more than one region, a segment has access to segments that are not in its path. Multiple-region structure can also be used to increase segment loading efficiency, because processing can continue in one region while the next path to be executed is being loaded into another region.

Figure 10 shows an example in which several control sections are used by most of the modules of the program. Control sections C, D, and G, which can overlay each other, depend on control section J as well as on control sections E and H. Placing the required control sections J, E, and H in the root segment would make the main storage requirement larger than necessary since control section J can overlay control sections E and H. If these control sections are placed in a different overlay region, they can be in main storage when needed, regardless of which path is being executed in the first region.

Figure 11 shows these control sections in a two-region structure. As shown, segment 2 can use segment 5 or segment 8 depending on the logic of the program. Those segments are also available to segment 4. Segment 8 is available for use by segment 3. Thus, either path in region 2 can be in main storage regardless of the path being executed in region 1. Segments in region 2 can cause segments in region 1 to be loaded without being overlaid themselves.

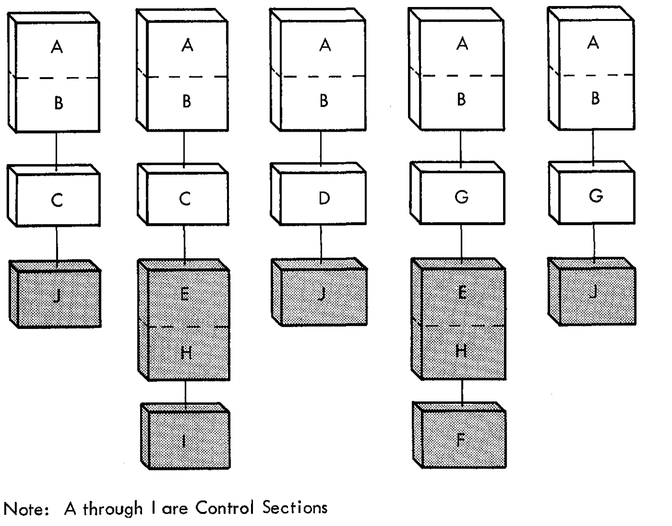


Figure 10. Modules Used by Several Paths

The relative origin of a second region is determined by the length of the longest path in the first region, which in Figure 11 is formed by segments 1 and 4. Segment 5, therefore, begins at 0 + 10,000 bytes. The relative origin of a third region would be determined by the length of the longest path in the first region plus the longest path in the second region.

The main storage required for the program is determined by adding the lengths of the longest path in each region. In Figure 11, the minimum main storage required for regions 1 and 2 is 15,000 bytes.

OVERLAY CHARACTERISTICS

During execution of an overlay program, the control program uses tables that were generated by the linkage editor and incorporated into the text (Figure 12). Since these tables are an integral part of the program, their size must be considered when planning the use of available main storage.

In addition to the storage area required by the program and the tables, the size of an area containing additional information used by the control program should also be considered. The formula is as follows:

Length in bytes = $4n+8$

where n is the number of segments in the overlay program.

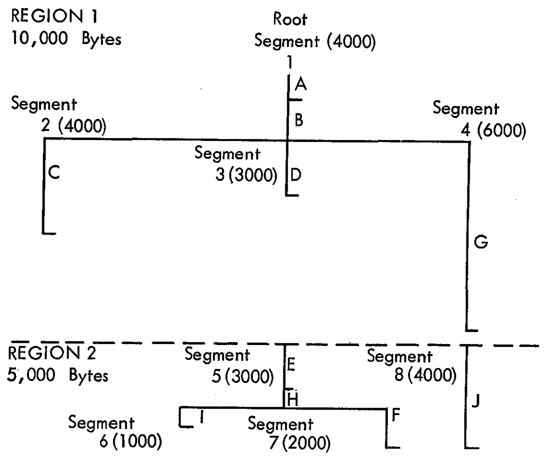


Figure 11. Multiple Region Overlay Tree Structure

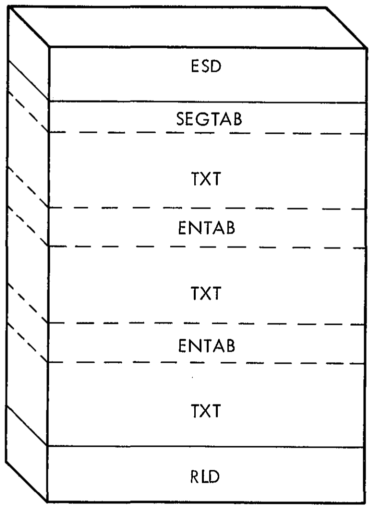


Figure 12. Load Module Containing Overlay Characteristics

Segment Table

There is only one segment table (SEGTAB) in an overlay program. The segment table is used to keep track of: (1) the relationship of the segments in the program; (2) which segments are in main storage or being loaded; and (3) other necessary control information.

The SEGTAB is the first control section in the root segment. For that reason, its size must be considered when the size of the root segment is being determined. The formula is as follows:

Length in bytes = 4n+24

where n is the number of segments in the overlay program.

Entry Table

There can be an entry table (ENTAB) in each segment of the program. The control program uses the entry table to determine the segment to be loaded when a branch instruction or macro-instruction refers to a segment not in the path.

An entry table may be produced as the last control section of a segment. An ENTAB entry is created for a symbol to which control is to be passed. The symbol is defined in a segment not in the path. In the Assembler Language, the symbol must be referred to by a 4-byte V-type address constant. An ENTAB entry is not produced for any symbol represented in an entry table closer to the root segment (higher in the path) or for a symbol defined in the path. Branches to a symbol in the path do not go through the control program since no overlay is necessary. An ENTAB entry is not created in the requesting segment for a symbol defined in an exclusive segment.

The size of the entry table must be taken into account when calculating the length of any segment that contains references as described above. The formula is as follows:

Length in bytes = 12(n+1)

where n is the number of ENTAB entries. If n=0, no ENTAB is created.

OVERLAY COMMUNICATION

The programmer must be aware of how his program can communicate with the control program during execution. There are four ways in which he can have his program request the use of the overlay facilities.

1. By a CALL macro-instruction, which causes a segment to be loaded and control to be passed to a symbol defined in that segment.
2. By a branch instruction, which causes a segment to be loaded and control to be passed to a symbol defined in that segment.
3. By a segment load (SEGLD) macro-instruction, which requests loading of a segment. Processing continues in the requesting segment while the requested segment is being loaded.
4. By a segment load and wait (SEGWT) macro-instruction, which requests loading of a segment. Processing continues in the requesting segment only after the requested segment is in main storage.

Communication Between Exclusive Segments

A reference between exclusive segments (exclusive reference) is made when the external symbol is:

1. Defined in the requested segment, and
2. Referred to by a 4-byte V-type address constant in the requesting segment.

The exclusive reference is valid only if the external symbol is referred to in an ENTAB entry created by a V-type address constant in a segment common to both segments.

If a common segment does not contain an ENTAB entry that refers to the external symbol, the exclusive reference is invalid.

When the program is processed by the linkage editor, an invalid exclusive reference is not resolved in an ENTAB entry but to the relative address of the symbol referred to. When the program is executed, the requested segment is not loaded and control is passed to an erroneous location. This error can be avoided by forcing an ENTAB entry in the common segment, i.e. placing in it a V-type address constant referring to the external symbol.

If the XCAL option is specified, the linkage editor does not consider a valid exclusive call as an error. Although an invalid exclusive call is always an error, the LET option will permit the module to be marked executable. (See "Special Processing Options.")

A valid exclusive reference is used to pass control to an exclusive segment by means of a branch instruction or a CALL macro-instruction. The reference will cause the requesting segment, and possibly other segments in its path, to be logically overlaid.

An exclusive reference should not be used in a SEGLD or SEGWT macro-instruction. Since both imply that processing is to continue in the requesting segment, an exclusive reference will generally lead to erroneous results when the program is executed.

CALL Macro-Instruction

The CALL macro-instruction refers to an external name in the segment to which control is to be passed. The requested segment and any segments in its path are loaded if they are not part of the path already in main storage. After the segment is loaded, control is passed to the requested segment at the location specified by the external name.

INCLUSIVE CALLS: A CALL between inclusive segments is always valid. A return can be made by means of the RETURN macro-instruction.

For a detailed discussion of the CALL and RETURN macro-instruction formats and operands refer to the publication IBM System/360 Operating System: Control Program Services, C28-6541.

EXCLUSIVE CALL: A call between exclusive segments is described in the section "Communication Between Exclusive Segments." Because the segment issuing an exclusive call is overlaid, a return from the requested segment can be made only by another exclusive call or branch.

CAUTION: The external name specified in the CALL macro-instruction must be referred to by a 4-byte V-type address constant. The high-order byte is reserved for use by the control program, and must not be altered during execution of the problem program.

Unless the LET or XCAL option is specified, a module that contains an exclusive call is marked "not executable," even though the call is valid.

If a call between exclusive segments does not conform to the necessary conditions for exclusive references, no ENTAB entry is created in the requesting segment. If the LET option has been specified, an invalid call or branch will cause unpredictable results when the program is executed. Since no ENTAB entry exists, control is passed directly to the relative address specified, even though the requested segment may not be in main storage.

Branch Instruction (Assembler Language Only)

Any of the branching conventions shown in Figure 13 can be used in place of the CALL macro-instruction to request loading and branching to a segment.

R15
: is the register into which is loaded a 4-byte V-type address constant that is an entry name or control section name defined in the requested segment. It must be the standard entry point register, register 15.

R_n
: is any other register (usually register 14).

As a result of using any of the branch instructions listed in Figure 13, the requested segment and any segments in its path are loaded if they are not part of the path already in main storage. Control is then passed to the requested segment at the location specified by the address constant V(name).

Note: In using the format ($D_2(X_2,B_2)$), either the base register or index register can be loaded with the address constant. The remaining two fields must be zero.

In using the format ($D_2(B_2)$) the base register must be loaded with the address constant, and the displacement must be zero.

Examples 5, 6, and 7 in Figure 13 are unconditional branches. Branches on other conditions are also allowed.

INCLUSIVE BRANCHES: A branch instruction between inclusive segments is always valid.

A return may be made by means of the address stored in R_n by the BAL or BALR instruction.

EXCLUSIVE BRANCH: A branch to an exclusive segment is described in the section "Communication Between Exclusive Segments." Because the segment issuing an exclusive branch is overlaid, a return can be made only by another exclusive branch.

CAUTION: The address constant placed in register 15 must be a 4-byte V-type address constant. The high-order byte is reserved for use by the control program, and must not be altered during execution of the program.

	Name	Operation	Operand
1	anyname	L BALR	R15,=V(name) R_n,R15
2	anyname ADCON	L BALR . . . DC	R15,ADCON R_n,R15 V(name)
3		L BAL	R15,=V(name) R_n,0(0,R15)
4	anyname	L BAL	R15,=V(name) R_n,0(R15)
5		L BCR	R15,=V(name) 15,R15
6	anyname	L BC	R15,=V(name) 15,0(0,R15)
7	anyname	L BC	R15,=V(name) 15,0(R15)

Figure 13. Branching Instructions

Segment Load (SEGLD) Macro-Instruction

The SEGLD macro-instruction is used to provide overlap between segment loading and processing within the requesting segment.

As a result of using the SEGLD macro-instructions listed in Figure 14, the loading of the requested segment and any segment in its path is initiated if they are not part of the path already in main storage. Processing resumes at the next sequential instruction while the segment or segments are being loaded.

In examples 1 and 2 in Figure 14, control is passed to the requested segment by the CALL macro-instruction or a branch instruction.

In example 3, the SEGWT macro-instruction ensures that the data in the control section specified by the external symbol is in main storage before processing of that data begins.

CAUTIONS: An exclusive reference should not be used in a SEGLD macro-instruction.

The external name specified in the SEGLD macro-instruction must be referred to by a four-byte V-byte address constant. The high-order byte is reserved for use by the control program, and must not be altered during execution of the problem program.

	Name	Operation	Operand
1	anyname	SEGLD ---- ---- ---- CALL	external name external name
2		SEGLD ---- ---- ---- branch	external name
3		SEGLD ---- ---- ---- SEGWT L	external name external name R_n,=A(name)

external name is the name of a control section or an entry name in the requested segment.

R_n is any register (usually register 14).

Figure 14. Processing After a SEGLD Macro-Instruction

Note: Some subsets of the control program do not have the capability of processing the SEGLD macro-instruction. In those subsets, the macro-instruction is treated as a NOP (no operation) and the segment is loaded when a SEGWT or branch is executed. If the rules of overlay are followed, correct execution occurs.

Segment Wait (SEGWT) Macro-Instruction

The SEGWT macro-instruction is used to stop processing in the requesting segment until the requested segment is in main storage.

As a result, the SEGWT macro-instruction in Figure 15 ensures that no further processing will take place until the requested segment and all segments in its path are loaded if not already in main storage. Control is returned to the next sequential instruction in the requesting segment.

In example 1, the SEGLD macro-instruction causes overlap between processing and segment loading.

In example 2, no overlap is provided. The SEGWT macro-instruction initiates loading. Processing is stopped until the requested segment is in main storage.

	Name	Operation	Operand
1	anyname	SEGLD	external name

		SEGWT	external name
		L	R_n,ADCON

		----	
		branch	
	ADCON	DC	A(name)
2		SEGWT	external name
		L	R_n,=A(name)
external name is the name of a control section or an entry name in the requested segment.			
R_n is any register (usually register 14).			

Figure 15. Processing After a SEGWT Macro-Instruction

Note: If the contents of a main storage location in the requested segment are to be processed, the entry name of the location must be referred to by an A-type address constant.

CAUTIONS: An exclusive reference must not be used in a SEGWT macro-instruction.

The external name specified in the SEGWT macro-instruction must be referred to by a four-byte V-type address constant. The high-order byte is reserved for use by the control program, and must not be altered during execution of the problem program.

EDITING OBJECT AND LOAD MODULES

The linkage editor performs editing functions either automatically, or as directed by linkage editor control statements. It also reserves storage for common control sections and static external storage areas.

MODULE EDITING

The editing functions of the linkage editor facilitate program modification by making changes on a control section basis. Thus, changes to a program do not require the recompilation of the entire source program. Changes can be made to external symbols within a module. External symbols and control sections can be deleted or replaced; control sections can be repositioned during the structuring of an overlay program.

The editing functions performed on object or load modules can be requested by the programmer by means of the CHANGE and REPLACE statements. Replacement of control sections can also be accomplished automatically.

When a load module in an overlay structure is being processed, the overlay characteristics of the module are automatically removed. The programmer can respecify the original overlay structure or place the program in another structure by means of the OVERLAY and INSERT control statements.

If the module is not to be placed in an overlay structure, performance of the load module can be improved by removing any SEGLD or SEGWT macro-instructions. If the module is to be placed in a new overlay structure, care should be taken to ensure that any new location of a macro-instruction will not cause an exclusive reference.

Changes to a module can be accomplished by compiling the affected control sections. The modified control sections can then be made a part of the module as specified in the section "Automatic Replacement of Control Sections."

RESERVING STORAGE

In FORTRAN, the Assembler Language, and PL/I the programmer can create control sections that reserve main storage areas containing no data or instructions. Referred to as "common" or "static external," these control sections are produced in the object modules by the language translators. These common areas are used either as communication regions for different parts of a program or to reserve main storage areas for control sections that may be provided by other modules. These common areas are either named or blank (unnamed).

During processing, the linkage editor collects common areas. Thus, if more than one blank common area is found in the input, the largest blank common area is contained in the output module. If two or more common areas have the same name, the largest common area having that name is used to form the output module. All references in the output module to a blank common area refer to the one area retained. All references to a named common area refer to the largest of those identically named, which was the only one retained.

If the linkage editor finds a control section with the same name as a previously defined common area (or the reverse), the control section and common area are collected. Thus, the data and instructions in the control section are placed in the main storage locations reserved by the named common area. If the control section is smaller than the named common area, the rest of the area remains reserved in the output module. If the control section is larger than the named common area, it is collected and regarded as the largest of the common areas processed. When a control section is placed in a named common area, that area logically becomes a named control section. All subsequent control sections with the same name are deleted.

When modules containing common areas are to be placed in an overlay structure, the common control sections are collected. However, the linkage editor "promotes" the common area automatically; that is, places it in the common segment of the paths containing references to it so that it is in main storage when needed. The position of the promoted common area in relation to other control sections within the common segment is generally unpredictable.

In Figure 16, the design of the overlay structure to be processed indicates that segments 3 and 4 contain blank common areas. Segments 6, 7, and 8 contain common areas with the same name.

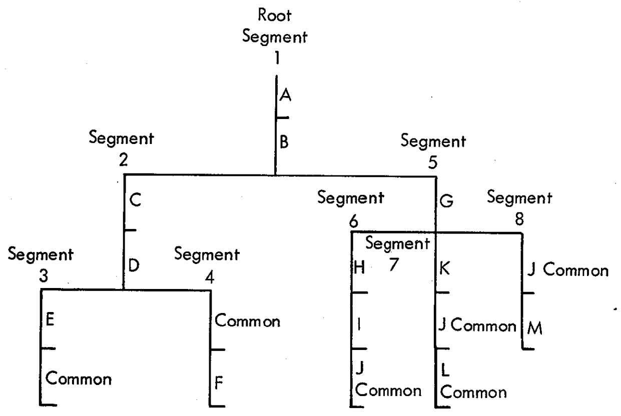


Figure 16. Common Areas Before Processing

During processing by the linkage editor, the blank common areas are collected and promoted to segment 2 (first common segment in the paths), as shown in Figure 17. The identically named common areas are collected and promoted to segment 5.

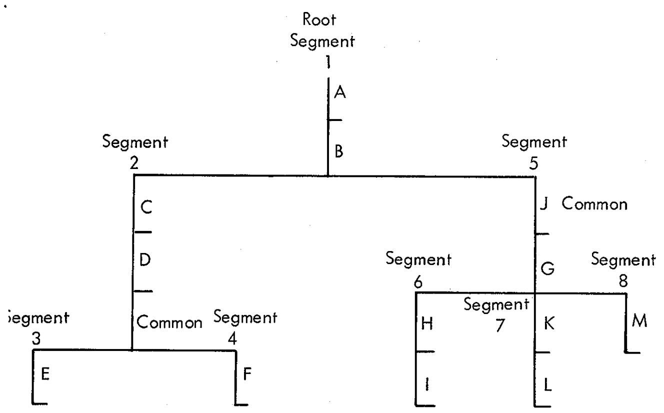


Figure 17. Automatic Promotion of Common Areas During Processing

ERROR DIAGNOSTICS AND PROCESSING OPTIONS

Error diagnostic options providing a number of aids are available to assist the programmer in testing the problem program.

Special processing options are available to inform the linkage editor that it is to give special consideration to error conditions or to negate the use of the automatic library call mechanism.

DIAGNOSTIC OUTPUT OPTIONS

The programmer can request that the linkage editor produce a list of all processed control statements and a module map or a cross-reference table to help him in testing his program. Additional information on the disposition of the output module is automatically provided.

LIST Option

When the LIST option is specified, all control statements processed by the linkage editor are listed in card-image format on the diagnostic output data set.

MAP and Cross-Reference Table (XREF) Options

When the MAP option is specified, the linkage editor produces a map of the output module. If the output module is not in an overlay structure, the module map lists the control sections in ascending order according to their assigned origins. Under each control section is a list of all entry names defined in the control section.

If the output module is in an overlay structure, the control sections are grouped by segment. Within each segment, the control sections are listed in ascending order according to their assigned origins. The number of the segment in which they appear is also listed.

When the XREF option is specified, the linkage editor produces a cross-reference table of the output module. The cross-reference table includes a module map and a list of all address constants that refer to other control sections.

Since the cross-reference table contains a module map, both XREF and MAP cannot be specified for one linkage editor run.

SPECIAL PROCESSING OPTIONS

The special processing options allow the programmer to indicate that the linkage editor is to give special consideration to certain error conditions or negate the automatic library call mechanism.

LET and Exclusive Call (XCAL) Options

When the LET option is specified, the linkage editor marks the output module as executable even though a severity 2 error condition was found during processing. See the section "Diagnostic Messages" for a definition of error types and severity codes.

When the XCAL option is specified, the linkage editor marks the output module as executable even though valid exclusive references between segments have been made. However, other errors may cause the module to be marked "not executable."

LET includes XCAL, so only one of the options needs to be specified to permit exclusive branches.

No Automatic Library Call (NCAL) Option

When the NCAL option is specified, the linkage editor automatic library call mechanism does not call library members to resolve external references within the linkage editor input. The output module is marked executable even though unresolved external references have been recognized.

The LIBRARY statement can be used to negate the automatic library call for selected external references when the NCAL option is not specified.

A "cataloged procedure" can be defined by the programmer after the system is generated. It will provide all the information necessary for him to load and execute his program. However, if a cataloged procedure is not provided, or if the programmer wishes to override any part of a cataloged procedure, he must provide job control statements that describe the jobs to be performed by the control program. Refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550 for details.

Linkage editor control statements can be used by the programmer to specify additional input sources, editing functions to be performed, or construction of an overlay program.

LINKING INPUT MODULES

An execution of the linkage editor requires that the appropriate job control statements and modules to be linked be placed in the primary input data set (SYSLIN).

JOB EXECUTION

Job control statements describe to the control program jobs to be performed by the system. A job can consist of the execution of one program or of a series of job steps (execution of a number of programs). The job and job steps are described in job control statements placed in the order in which they are to be performed.

Every job is indicated by a JOB statement preceding the definitions of the steps within the job. The execution of a job step is described by an execute (EXEC) statement and one or more data definition (DD) statements. The EXEC statement indicates what program is to be executed. The DD statements describe the data sources and destinations of the program. The sequence of statements for one job step is:

```
          JOB
          EXEC  program name
ddname    DD    dsname
ddname    DD    dsname
ddname    DD    dsname
          etc.
```

The sequence of job control statements for a three-step job may be:

```
          JOB
          EXEC compiler program name
ddname    DD   dsname
ddname    DD   dsname
ddname    DD   dsname
          EXEC linkage editor program name
SYSLIN    DD   dsname primary input
SYSLIB    DD   dsname call library
SYSUT1    DD   dsname buffer data set
SYSLMOD   DD   dsname output library
SYSPRINT  DD   dsname diagnostic output
ddname    DD   dsname additional library
ddname    DD   dsname additional library
          EXEC problem program
ddname    DD   dsname
ddname    DD   dsname
```

Member Name

The name of the output module must be unique in the output library. It can be specified in the SYSLMOD DD statement. If the member is to replace an identically named member in the library, the subparameter OLD can be specified in the disposition field (DISP) of the SYSLMOD DD statement.

The subparameters NEW or MOD indicate that no member of the same name exists in the library. If either NEW or MOD is specified, the member is added to the library. These subparameters need not be specified.

If the member name field of the SYSLMOD DD statement is omitted, the member name for the output module can be specified in a NAME statement. If the member is to replace an identically named member in the library, the replace function can be specified in the NAME statement.

Linkage Editor Completion Code

The linkage editor passes a completion code to the control program upon completion of the job step. The control program compares the completion code with the values specified in the COND field of (1) the JOB control statement that was specified for this job step, and (2) the EXEC statement specified in any succeeding job step. The results of the comparisons are used to determine subsequent action.

The completion codes, in decimal, are as follows:

00	Normal conclusion.
04	Warning messages have been listed, execution should be successful.
08	Error messages have been listed, execution may fail.
12	Severe errors have occurred, execution is impossible.
16	Terminal errors have occurred, the processor has terminated.

Refer to the publication IBM System/360 Operating System: Job Control Language for details on the COND field of the JOB and EXEC statements.

CATALOGED PROCEDURE

To facilitate the operation of the system, the control program enables the programmer to store job control statements under a unique name so that they can be recalled at any time to define a job. Thus, the series of job control statements shown could be retained by the control program. To request this procedure, the programmer places in the primary input data set an EXEC statement indicating the name of the series desired.

The formats and parameters of job control statements and the method of setting up or overriding a cataloged procedure in the system is presented in the publication IBM System/360 Operating System: System Programmer's Guide.

MODULE ATTRIBUTES

Unless specific module attributes are indicated by the programmer, the output module will neither be tested nor will it be in an overlay structure. It will be in block format, not reenterable, and not serially reusable. To specify particular attributes, the programmer must place the desired attribute symbols in the parameter field of the EXEC statement that defines linkage editor processing.

Although there are eight module attributes, several are mutually exclusive. The attributes, and those with which they are incompatible, are described below.

Note: The "not editable" and "only loadable" attributes are intended primarily for use by the control program. Use of these attributes by the problem programmer can diminish the usability of the module.

Scatter Load (SCTR) and Overlay (OVLY) Attributes

When the SCTR attribute is specified, the linkage editor produces a load module in a format suitable for scatter or block loading.

When the OVLY attribute is specified, the load module is in an overlay structure that is suitable only for block loading. If OVLY is specified and no OVERLAY statements are found in the linkage editor input, the overlay attribute is negated. The condition is considered a recoverable error; i.e., if the LET option is specified, the module will be marked executable.

Only one of these attributes can be specified for one linkage editor job. If neither is specified, the load module will be in block format.

Note: Where the scatter load feature is not available in the control program, programs with the SCTR attribute are block loaded.

Reenterable (RENT) and Serially Reusable (REUS) Attributes

When the RENT attribute is specified, the linkage editor marks the output module as reenterable. However, if any load modules that are not reenterable become a part of the input to the linkage editor, the RENT attribute is negated.

When the REUS attribute is specified, the linkage editor marks the output module as serially reusable. If any load modules that are neither reenterable nor serially reusable become part of the input to the linkage editor, the REUS attribute is negated.

Only one of these two attributes can be specified for a linkage editor job. If OVLY or TEST is specified, neither RENT nor REUS can be specified. If neither RENT nor REUS is specified, the output module will be not reenterable and not serially reusable.

Not Editable (NE) Attribute

When the NE attribute is specified, the resulting load module has no external symbol dictionary. The load module cannot be reprocessed by the linkage editor. The load module produced requires less direct-access storage.

If a map or cross-reference table is requested the "not editable" attribute is negated.

Only Loadable (OL) Attribute

When the OL attribute is specified, the module can be brought into main storage only by the LOAD macro-instruction. The "only loadable" module must be entered by means of a branch instruction or a CALL macro-instruction. If an attempt is made to enter the module via a LINK, XCTL, or ATTACH macro-instruction, the program making the attempt is terminated abnormally by the control program.

Note: Some subsets of the control program use a smaller control table when the load module is loaded for execution. This reduces the overall main storage requirements of the module.

Downward Compatible (DC) Attribute

The DC attribute ensures that the load module processed by the level F linkage editor can be reprocessed by either the level E or the level F linkage editor. If the E level linkage editor is requested to process a load module that does not have this attribute, the request will be treated as an error.

The E level linkage editor automatically assigns the DC attribute to all load modules it produces.

Note: The level F linkage editor program is designed to process in main storage environments where the space available to the linkage editor is 44K or more. The level E linkage editor is designed to process programs in all environments.

TEST Attribute (Assembler Language Only)

When the TEST attribute is specified, the linkage editor accepts the testing symbol tables for the test translator within the input modules. The tables are placed as part of the output module. The module is marked as being "under test."

When TEST is not specified, symbol tables are ignored and not placed in the output module.

When TEST is specified, neither RENT nor REUS can be specified.

Incompatible Attributes

When mutually exclusive attributes are specified for a load module, the linkage editor ignores the less significant attributes. Figure 18 illustrates the significance of the incompatible attributes.

Attributes Specified				
Accepted	Ignored (X)			
	SCTR	REUS	RENT	NE
OVLY	X	X	X	
TEST		X	X	X
RENT		X		

Figure 18. Incompatible Module Attributes

SPECIFYING ADDITIONAL PROCESSING

The programmer can specify linkage editor functions, in addition to those it performs automatically, by means of linkage editor control statements. These functions permit specification of:

- Multiple load modules processed in a single job step.
- The load module entry point.
- Aliases for the output module name.
- Additional input sources.
- An overlay program.
- Editing functions to be performed on input modules.
- Maintenance information for IBM-supplied load modules.

The following conventions are used in this publication to illustrate the format and coding of control statements:

- Upper-case letters (coded value), numbers, and punctuation marks must be coded by the programmer exactly as shown. Exceptions to this convention are brackets, [ ], and braces, { }. These are never coded.
- Lower-case letters and words represent variables for which the programmer must substitute specific information or specific values.
- Items or groups of items within brackets are optional. They may be omitted at the programmer's discretion.
- Braces group related items, such as several alternative items. One item within the braces must be selected.
- Stacked items, enclosed in either brackets or braces, represent alternative items. No more than one of the stacked items should be coded by the programmer.

LINKAGE EDITOR CONTROL STATEMENTS

General format and placement information for the linkage editor control statements is contained in the following paragraphs. For examples of the use of the control statements, refer to Appendix A; for a summary of their functions and formats, refer to Appendix B.

General Statement Format

All linkage editor control statements have the following format:

Operation	Operand
VERB	a,b(c),(d),(VALUE)

As used in this publication:

a is an unsubscripted symbol.
b is a subscripted symbol.
c is a subscript symbol.
d is a parenthesized symbol.
(VALUE) is a coded value.
a, b(c), (d), and (VALUE) are operands.

The operation field must contain the name of the operation to be performed. The operand field must contain one or more symbols, subscripted symbols, or parenthesized symbols. Operands in the operand field must be separated by commas. Two or more symbols within parentheses must be separated by commas. A coded value must be written exactly as shown.

If the operand field is blank, the linkage editor will not process the control statement.

No symbols are allowed preceding the operation field, which must begin to the right of column 1. The operation field must be separated from the operand field by at least one blank position.

The control statement can be continued on as many cards as necessary by placing a nonblank character in column 72 of the card. Continuation must begin in column 16 of the next card. A symbol cannot be split; that is, it cannot begin on one card and be continued on the next.

General Placement Information

Linkage editor control statements are placed before, between, or after modules. They can be grouped, but they cannot be placed within a module. However, specific placement restrictions may be imposed by the nature of the functions being requested by the control statement. Any placement restrictions are noted in the discussions of linkage editor functions.

MULTIPLE LOAD MODULE PROCESSING

The linkage editor can produce more than one load module in a single job step. A NAME statement in the input stream is used as a delimiter for input to a load module. If additional input modules follow the NAME statement in the input stream, they are used in the formation of the next load module.

The module name field of the SYSLMOD DD statement should be omitted when a NAME statement is used to specify the name of the first (or only) load module. However, if the SYSLMOD statement does specify a member name, the name must be identical to that specified in the first NAME statement or an ALIAS statement for the first output module produced. In either case, the NAME statement is regarded as the last item to be processed for the preceding load module.

When processing multiple load modules in a single job step, the options and attributes specified in the EXEC statement for that job step apply to each load module created.

If the linkage editor terminates during processing of any of the output modules, neither that module nor any of the modules yet to be processed in the job step is processed or placed in the library.

NAME Statement

The NAME statement specifies the name of the load module created from the preceding input modules.

It can also indicate that the load module replaces an identically named module in the library.

Operation	Operand
NAME	membername[(R)]

member name
: is the name to be assigned to the load module that is created from the preceding input.

(R)
: indicates that this load module replaces an identically named module in the library. If the module is not a replacement, the parenthesized coded value, (R), should be omitted.

PLACEMENT: The NAME statement is placed at the end of the last input module that is made a part of the output module. Any ALIAS statement used must precede the NAME statement.

CAUTION: A NAME statement found in a data set other than the primary input data set is invalid. It is ignored by the linkage editor.

THE LOAD MODULE ENTRY POINT

The linkage editor selects the entry point of a load module as follows:

- The programmer codes a language translator END statement as the last statement in the input to the assembler or compiler program. This END statement may or may not specify an entry point in its operand.

- From each input module, the assembler or compiler program produces one object module. At the end of this object module, the program places an END statement. This END statement indicates an entry point if the programmer's END statement specified an entry point.

- From one or more object modules, the linkage editor produces a load module. Besides the one or more assembler- or compiler-produced END statements in the object modules, the input to the linkage editor can contain a linkage editor ENTRY statement. (For assembler-produced modules, the ENTRY statement can specify only the name of a named control section or an entry point specified by an assembler ENTRY statement.) From this input, the linkage editor selects the entry point for the load module as follows:

  1. From the first linkage editor ENTRY statement in the input.

  2. If no linkage editor ENTRY statement is in the input, from the first assembler- or compiler-produced END statement that specifies an entry point.

  3. If no linkage editor ENTRY statement or no assembler- or compiler-produced END statement specifies an entry point, the first byte of the first control section of the load module is used as the entry point.

When a load module is reprocessed by the linkage editor, its assembler- or compiler-produced END statement is not present. Therefore, if the first byte of the first control section of the load module is not a suitable entry point, the module's entry point must be specified in one of two ways:

- Through a linkage editor ENTRY statement.

- Through the language translator END statement of another module, which is being processed for the first time. This object module must be the first module to be processed by the linkage editor.

For a load module that was originally written in assembler language, the entry point so specified can be the same as the one originally indicated by the module's END statement only if the operand of the assembler END statement was declared as the name of a named control section or as an entry point specified by an assembler ENTRY statement. When the entry point is being specified through the END statement of another module, it must also be declared in an assembler EXTRN statement in the assembly that produced the module.

In general, an entry point should be specified by a language translator END statement or a linkage editor ENTRY statement, because it is not always possible to predict which control section will be first in the output module.

In an overlay program, the first instruction to be executed must be in the root segment.

ENTRY Statement

The ENTRY statement specifies the first instruction to be executed.

Operation	Operand
ENTRY	externalname

external name
: is defined as a control section name or an entry name in a linkage editor input module. It must be the name of an instruction, not of data. In an overlay program, the external name must be defined as the name of an instruction in the root segment.

PLACEMENT: An ENTRY statement can be placed before, between, or after object modules or other control statements.

EXAMPLE: ENTRY GO

GO
: is defined as the external name of the first instruction to be executed when the module is loaded by the control program.

As a result, the address of the instruction indicated by the symbolic name GO is specified by the linkage editor as the starting point of the program when it is called by its module name for execution. The control program will pass control to the instruction specified.

OAD MODULE ALIAS NAMES

An output module can be referred to by p to five aliases specified by the ALIAS statement. The aliases exist in addition o the name of the output module specified n the SYSLMOD data definition statement or he NAME statement. A module referred to y an alias will begin execution at the xternal name specified by the alias. If he name specified by the ALIAS statement oes not exist within the module, the ddress of the main entry point will be ssigned to the alias.

LIAS Statement

The ALIAS statement specifies alternative names for the output library member, nd can also specify alternative points of ntry for execution.

Operation	Operand
ALIAS	{externalname / symbol}

xternal name
: is defined as a control section name or entry name in the output module. Up to five alias names can be specified on one or separate statements. The names exist in addition to the name of the module specified in the data definition statement for the library member. The alias, which is an external name in the output module, allows the load module to be called by other modules that refer to that name for execution or for linkage editor processing. When the module is called for execution, execution will begin at the external name referred to.

  Any additional external name specified in the ALIAS statement must be preceded by a comma.

  In an overlay program, the external name specified by the ALIAS statement must be in the root segment.

ymbol
: specifies a name that is not an external name within the output module. The entry point used when the module is called for execution is that of the main entry point.

  Any additional symbol specified in the ALIAS statement must be preceded by a comma.

PLACEMENT: An ALIAS statement can be placed before, between, or after object modules or other control statements. It must precede a NAME statement used to specify the member name.

ADDITIONAL DATA SOURCES

The linkage editor can accept input from sources other than the primary input source. Additional input sources can be specified by means of the INCLUDE statement or automatic library call. The automatic library call mechanism can be directed to data sets other than that specified in the SYSLIB DD statement by means of the LIBRARY statement, used to designate specific external references to be resolved. All ddnames specified in INCLUDE or LIBRARY statements must be defined in DD statements.

The record format (RECFM), block size (BLKSIZE), and, if required, tape recording technique (TRTCH) and density (DEN) fields of the data control block must be made available to the linkage editor. If this information does not exist in the data set control block or tape header label, or if no labels are used, the programmer must specify it on the DD statement defining the data set.

The INCLUDE statement causes the linkage editor to process the module or modules indicated. The next primary input item is then processed. If the included data set also uses an INCLUDE statement, that statement is processed as the last item in the included data set (Figure 19).

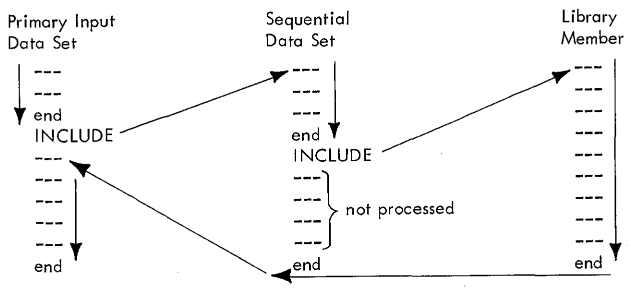


Figure 19. Processing of Additional Data Sources

The automatic library call process is used to resolve external references that were not resolved during primary input processing. An automatic library call can resolve an external reference when the following conditions exist. The external reference must be:

- A member name or an alias of a module in a partitioned data set, and
- Defined as an external name in the external symbol dictionary of the module with that name.

If an external reference is resolved by automatic library call, the entire member is processed as input to the linkage editor.

Unresolved external references found in modules from additional data sources are processed by the automatic library call mechanism.

Note: Modules contained in data sets called automatically because of unresolved external references in segments of an overlay program are placed in the root segment, not in the segment that called them. To place the control sections of a module in a different overlay segment, the programmer must use the INSERT statement.

The LIBRARY statement is not needed if all references can be resolved from the call library defined in the SYSLIB DD statement.

INCLUDE Statement

The INCLUDE statement indicates additional input sources.

Operation	Operand
INCLUDE	ddname[(membername)]

ddname
  is the name of a DD statement that defines a library containing either object modules and control statements or load modules; or a sequential data set containing object modules and control statements.

  Any additional ddname must be preceded by a comma.

member name
  is the name of a member of the library.

  Any additional member name must be preceded by a comma.

The operand field must contain one or more ddnames separated by commas. If the ddname specifies a library, it must be followed by one or more subscript symbols separated by commas. Each subscript symbol must be either a member name or an alias name in the specified library. If the ddname specifies a sequential data set, it must not be subscripted.

PLACEMENT: An INCLUDE statement can be placed before, between, or after object modules or other control statements.

LIBRARY Statement

The LIBRARY statement can be used to specify:

- Additional call libraries.

- Restricted no-call: External references not to be resolved by the automatic library call mechanism during the current linkage editor job step.

- Never-call: External references not to be resolved by the automatic library call mechanism during any linkage editor job step.

Operation	Operand
LIBRARY	ddname(membername) [*](externalreference)

ddname
  is the name of a DD statement that defines a library.

  Any additional ddname must be preceded by a comma.

member name
  is the name of a member of the library.

  Any additional member name within the subscript must be preceded by a comma.

*
  is a coded value used to indicate the never-call function.

external reference
  is an external reference that may be unresolved after primary input processing. The external reference is not to be resolved.

  Any additional external reference within the subscript must be preceded by a comma.

If additional libraries are to be used to resolve external references by automatic library call, the operand field must contain one or more subscripted symbols separated by commas. Each subscript may contain one or more symbols separated by commas. Each symbol must be a member name or an alias name in the data set specified by the ddname.

If the restricted no-call function is being specified, the operand field must contain only parenthesized symbols separated by commas.

If the never-call function is being specified, the subscript expression must be preceded by an asterisk.

Combinations of LIBRARY statement functions can be written in the same LIBRARY statement.

PLACEMENT: A LIBRARY statement can be placed before, between, or after object modules or other control statements.

CAUTION: If the unresolved external symbol is not a member name in the library specified, the external reference will remain unresolved unless defined in another input module.

If the unresolved external symbol is a member name or an alias in the library specified, but is not an external name in that member, the member is processed but the external reference will remain unresolved unless defined in another input module.

If the NCAL option is specified, the LIBRARY statement cannot be used to specify additional call libraries.

Including Library Modules

Object modules and control statements, or load modules, contained in libraries can be included in the output module by means of the INCLUDE statement or the automatic library call process. If the INCLUDE statement is used, they are included immediately; if the automatic library call process is used, they are included by the automatic library call mechanism at the end of primary input processing. The LIBRARY statement can be used to direct automatic library call to a library other than that specified in the SYSLIB DD statement for resolution of specific external references.

CAUTION: The downward compatible option must be specified when load modules produced by the level F linkage editor are to be processed later by the level E linkage editor. If load modules produced by the level F linkage editor (88K version) and placed in IBM 2301 Drum Storage are to be processed later by any other linkage editor, the downward compatible option must also be specified.

EXAMPLE: INCLUDE LIBA(ADD,SUB,MULT)

LIBA
    is the ddname of a DD statement that defines a load module library containing load modules named ADD, SUB, and MULT.

As a result, the three load modules are included in the processing of the load module that is the output of the linkage editor.

EXAMPLE: LIBRARY LIBA(ROUT1,ROUT2)

LIBA
    is the ddname of a DD statement that defines a library containing the object modules named ROUT1 and ROUT2.

As a result, any unresolved external reference to either ROUT1 or ROUT2 causes the automatic library call mechanism to search for the member by that name in the indicated library. If there is no unresolved external reference to the name specified, the member is not called at the end of primary input processing.

Including Sequential Data Sets

Sequential data sets containing object modules and control statements can be specified by the INCLUDE statement for inclusion immediately. The record format (RECFM), block size (BLKSIZE), and, if required, tape recording technique (TRTCH) fields of the data control block must be made available to the linkage editor. If this information does not exist in the data set control block or tape header label, or if no labels are used, the programmer must specify it on the DD statement defining the sequential data set.

EXAMPLE: INCLUDE MOD1,MOD2,MOD3

MOD1,MOD2,MOD3
    are the ddnames of DD statements that define sequential data sets containing object modules that are to be included in the linkage editor input.

As a result, all object modules and control statements in the specified data sets are processed by the linkage editor. They will become part of the load module placed in the output module library.

The Restricted No-Call Function

The programmer can use the LIBRARY statement to specify those external references in the output module for which there is to be no search during the current linkage editor job step.

EXAMPLE: LIBRARY (SINE,TAN,COTAN)

SINE,TAN,COTAN
: are external references in the output module.

As a result, if SINE, TAN, or COTAN is unresolved after primary input processing, no automatic library call is made since the ddname is omitted.

The Never-Call Function

The never-call function specifies those external references that are not to be resolved by automatic library call during this or any subsequent linkage editor run. The never-call function is specified by an asterisk in the ddname position of the LIBRARY statement.

The never-call function is negated when a module containing the external name referred to is part of the input to the linkage editor.

Example: LIBRARY *(SINE)

*
: specifies the never-call function.

SINE
: is an external reference in the output module.

As a result, if SINE is unresolved after input processing, no automatic library call is made. During later linkage editor runs, SINE will not be resolved by automatic library call.

STRUCTURING AN OVERLAY MODULE

Once the programmer has designed an overlay tree structure for a module, he must place the module in that structure by indicating to the linkage editor the relative positions of the segments and the regions in the tree structure, and the control sections within the segments. Positioning is accomplished as follows:

- Segments. Segments are positioned by OVERLAY statements. Since segments are not named, the programmer must identify a segment by giving its origin a symbolic name and specifying that name in the OVERLAY statement. Each OVERLAY statement signifies the start of a new segment. The first time a symbolic name is used, a node point is created at the end of the previous segment. That node point is logically assigned a position one greater than the last item in the preceding segment. Subsequent use of the same name indicates that the next segment is to have its origin at that node point.

- Regions. Regions are positioned by OVERLAY statements. The programmer indicates the origin of a region by specifying the origin of the first segment of the region and the coded value (REGION).

- Control sections. Control sections are positioned in the segment specified by the OVERLAY statement that they follow in the input sequence. Control sections that precede the first OVERLAY statement or that are called automatically are positioned in the root segment. They can be repositioned by means of the INSERT statement. Common control sections are automatically repositioned as described in "Reserving Storage."

The input sequence of control statements and modules should reflect the order of the segments in the overlay tree structure from top to bottom, left to right, and region by region. The same symbolic name cannot be used to begin a new segment at the same node point once processing at a point higher in the tree structure has resumed.

OVERLAY Statement

The OVERLAY statement indicates either:

1. The beginning of an overlay segment, or
2. The beginning of an overlay region.

Operation	Operand
OVERLAY	symbol[(REGION)]

symbol
: is the symbolic origin of a segment. The symbol is not related to external symbols in a module.

(REGION)
: specifies the origin of a new region.

The operand field contains only the symbol when the origin of a segment is being specified. The operand must contain the symbol followed by the coded value (REGION) when the origin of a new region is being specified.

PLACEMENT: The OVERLAY statement must precede the first module of the next segment, the INCLUDE statement specifying the first module of the segment, or the INSERT statement specifying the control sections to be positioned in that segment.

Note: An efficient method of specifying an overlay structure is to group all of the OVERLAY statements with the appropriate INCLUDE and INSERT statements and then place the complete package either before or after the modules that form the program.

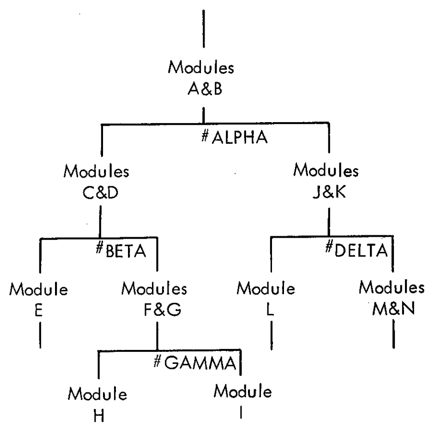
CAUTION: No OVERLAY statement should precede the root segment.

Segment Origin

The symbolic origin of every segment, other than the root segment, must be specified by the programmer in an OVERLAY statement, as described in the following example.

EXAMPLE: The input sequence listed in Figure 20 will produce the overlay structure shown. The modules are named for purposes of illustration.

```
Module A
Module B
OVERLAY ALPHA
Module C
Module D
OVERLAY BETA
Module E
OVERLAY BETA
Module F
Module G
OVERLAY GAMMA
Module H
OVERLAY GAMMA
INCLUDE MODI
OVERLAY ALPHA
Module J
Module K
OVERLAY DELTA
Module L
OVERLAY DELTA
Module M
Module N
```



# Node Point

Figure 20. Single Region Overlay Tree Structure

Region Origin

The symbolic origin of each region, other than region 1, must be specified by the programmer in an OVERLAY statement. The origin of the region will be the origin of any segment in the region specified by the same symbolic name.

EXAMPLE: The input sequence listed in Figure 21 will produce the overlay structure shown. The modules are named for purposes of illustration.

```
Module A
Module B
OVERLAY ALPHA
Module C
OVERLAY ALPHA
Module D
OVERLAY BETA
Module E
OVERLAY BETA
Module F
OVERLAY ALPHA
Module G
OVERLAY GAMMA (REGION)
INCLUDE MODH,MODI
OVERLAY GAMMA
Module J
OVERLAY DELTA
Module K
OVERLAY DELTA
Module L
Module M
OVERLAY GAMMA
Module N
```

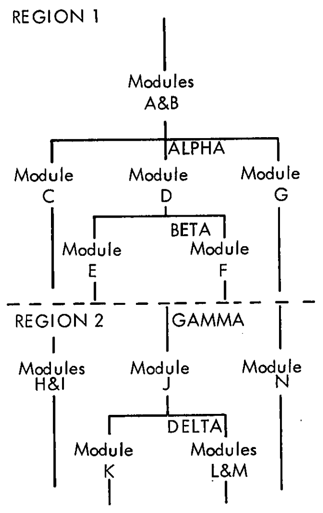


Figure 21. Multiple Region Overlay Tree Structure

Positioning Control Sections

A control section can be repositioned by moving it from its position in the input sequence to a specific segment by one of two methods:

1. The use of the INSERT statement, or
2. Automatic promotion of common areas.

A control section to be repositioned can appear in any module in the input sequence. The INSERT statement takes precedence. Care should be taken in applying the rules for exclusive references if any will result from the move.

INSERT Statement

The INSERT statement positions control sections in overlay segments.

Operation	Operand
INSERT	controlsectionname

control section name
    is the name of the control section to be repositioned.

Any additional control section name must be preceded by a comma.

The operand must contain one or more control section names separated by commas.

If the symbol specified in the operand field of the INSERT statement is not presently in the external symbol dictionary, it is entered as an external reference. If the reference has not been resolved at the end of primary input processing, the automatic library call mechanism attempts to resolve it.

PLACEMENT: The INSERT statement must be placed in the input sequence following the OVERLAY statement that specifies the origin of the segment in which the control section is to be positioned. If the control section is to be positioned in the root segment, the INSERT statement must be placed before the first OVERLAY statement.

CAUTION: A control section can appear only once within a load module.

Control sections that are positioned in a segment must contain all address constants used during execution unless:

1. The A-type address constants are located in a segment in the path.
2. The V-type address constants used to pass control to another segment are located in the path. If an exclusive reference is made, the V-type address constant must be in a common segment.
3. The V-type address constants used with the SEGLD and SEGWT macro-instructions are located in the segment.

EXAMPLE: The input sequences listed in Figure 22 will produce the overlay structure shown. The modules are named for purposes of illustration.

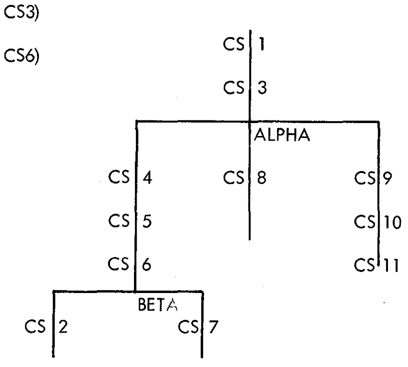
EDITING MODULES

The editing functions of the linkage editor facilitate program modification; they make it possible to modify a program by changing a control section within it, rather than by recompiling the entire source program. The following editing functions can be performed by the linkage editor:

- External symbols can be changed within a module by means of the CHANGE statement.

- Control sections within a module can be replaced either automatically or by means of the REPLACE statement.

```
Module A (CS1, CS2, CS3)
OVERLAY ALPHA
Module B (CS4, CS5, CS6)
OVERLAY BETA
INSERT CS2
OVERLAY BETA
INSERT CS7
OVERLAY ALPHA
Module C (CS7, CS8)
OVERLAY ALPHA
INCLUDE MODD
```



The same overlay structure could be achieved by either of the following input sequences:

Input Sequence	Input Sequence
Module A (CS1,CS2,CS3)	INSERT CS1,CS3
Module B (CS4,CS5,CS6)	OVERLAY ALPHA
Module C(CS7,CS8)	INSERT CS4,CS5,CS6
INCLUDE MODD	OVERLAY BETA
OVERLAY ALPHA	INSERT CS2
INSERT CS4,CS5,CS6	OVERLAY BETA
OVERLAY BETA	INSERT CS7
INSERT CS2	OVERLAY ALPHA
OVERLAY BETA	INSERT CS8
INSERT CS7	OVERLAY ALPHA
OVERLAY ALPHA	INCLUDE MODD
INSERT CS8	Module A (CS1, CS2, CS3)
OVERLAY ALPHA	Module B (CS4, CS5, CS6)
INSERT CS9,CS10,CS11	Module C (CS7, CS8)

Figure 22. Repositioned Control Sections in Overlay Structure

- Control sections or external symbols can be deleted from a module by means of the REPLACE control statement.

Note: Certain external symbols (entry names and external references) can be changed by the REPLACE statement; in such cases its operand field is the same as that of the CHANGE statement.

Editing Conventions

In requesting linkage editor editing functions, certain conventions should be followed to ensure that the specified change or deletion is processed correctly.

1. External references from other modules to a changed external name must be changed by a separate control statement. (External references and address constants within the same input module automatically refer to the new symbol.)

2. An external symbol will be deleted only if no address constant refers to it from within the same input module.

If an address constant does refer to it from within the same input module, the symbol will be changed to an external reference. If the external reference is unresolved at the end of primary input processing, the automatic library call mechanism will attempt to resolve it.

3. External references from other modules to a deleted external symbol will cause the output module to be marked "not executable" unless one of the following occurs:

   a. The LET or NCAL option is specified.
   b. The restricted no-call or never-call function is specified for the unresolved external reference.
   c. The external symbol is defined in another module processed in that linkage editor execution.

4. When an INCLUDE statement follows a CHANGE or REPLACE statement, it must specify only one module.

5. (Applicable only to Assembler Language programming.) When control sections that were or are part of a separately assembled module are to be replaced, A-type address constants that refer to a deleted symbol will be incorrectly resolved unless the entry name is in the same position relative to the origin of the replaced control section and the new control section. If all control sections of a separately assembled module are replaced, no restrictions apply.

6. Two identical external symbols should not appear as both subscript and subscripted symbols in one linkage editor run unless the statements that specify the symbols apply to different input modules.

7. Each time the linkage editor reprocesses a load module, the module's entry point should be specified in one of two ways:

   - Through a linkage editor ENTRY statement.

   - If the first object module is being processed for the first time, through its language translator END statement.

   The entry point of any module that may be reprocessed should be an external name within the module, so that the entry point can be specified in a linkage editor ENTRY statement or in the language translator END statement of another module. (For assembler language programming, if the entry point is specified in an assembler END statement and the entry point is not in the same object module, the entry point symbol must be declared in an assembler EXTRN statement in the assembly that produced the module.)

CHANGE Statement

The CHANGE statement changes a control section name, an entry name, or an external reference.

Operation	Operand
CHANGE	externalsymbol(newexternal symbol)

external symbol
    is a control section name, entry name, or external reference that is to be changed.

    Any additional subscripted external symbol must be preceded by a comma.

new external symbol
    is the name to which the subscripted symbol is to be changed.

PLACEMENT: The CHANGE control statement must be placed immediately before either the module containing the external symbol to be changed or the INCLUDE control statement specifying the module.

REPLACE Statement

The REPLACE statement performs one of the following:

1. Deletes a control section, an entry name, or external reference.

2. Deletes a control section to be replaced by another.

3. Changes an entry name or an external reference.

Operation	Operand
REPLACE	externalsymbol[(newexternal symbol)]

external symbol
is the name of a control section, an entry name, or an external reference to be replaced. If the external symbol is not followed by a subscript, the linkage editor deletes the control section, entry name, or external reference.

new external symbol
is the name that is to replace the subscripted symbol.

Any additional external symbol must be preceded by a comma.

PLACEMENT: The REPLACE statement must immediately precede either the module containing the control section or external symbol to be deleted or the INCLUDE statement specifying the module.

Changing an External Symbol

Names of external symbols in a module can be changed by means of the CHANGE control statement.

EXAMPLE: CHANGE ROUTINE1(CSECT1),BEGIN (REPEAT)

ROUTINE1
is a control section name that is to be changed.

BEGIN
is an external reference to a control section that has been replaced by another named REPEAT.

As a result, the name of the control section is changed to CSECT1. The external reference BEGIN is changed to REPEAT.

Replacing Control Sections

The REPLACE statement can be used to replace a control section with a control section of another module. In replacing a control section, the linkage editor first deletes the specified control section from the input module, and then prepares that module to receive the new control section. References to the old control section become unresolved external references unless the same entry names appear in the new control section or in some other control section in the linkage editor input.

EXAMPLE: REPLACE ROUTINE1(CSECT1), ROUTINE2(CSECT2)

ROUTINE1 and ROUTINE2
are names of control sections in the module following the control statement.

CSECT1 and CSECT2
are names of control sections in other modules that are input to the linkage editor.

As a result, the control sections ROUTINE1 and ROUTINE2 are removed from the module and replaced by CSECT1 and CSECT2, respectively. Any address constants within the module that refer to ROUTINE1 or ROUTINE2 now refer to CSECT1 or CSECT2.

Automatic Replacement of Control Sections

To replace a control section automatically, the programmer places the module containing the new control section, or an INCLUDE control statement specifying the module containing the new control section, in the primary input so that it is processed ahead of the control section to be replaced. Both control sections must be identically named.

The first of the identically named control sections processed by the linkage editor is made a part of the output module. All subsequent identically named control sections are deleted from their modules. (See example 7 in Appendix A.)

External references to identically named control sections are resolved with respect to the first such control section processed by the linkage editor.

Note: By concatenating the output of a language translator and the primary input data set, modified control sections automatically replace control sections in the original object module.

CAUTION: When identically named control sections appear in modules being placed in exclusive overlay segments, the second control section encountered is deleted from its module. Resolution of external references may cause invalid exclusive references. Invalid exclusive references cause the linkage editor to mark the output module "not executable" unless the LET option is specified.

Deleting a Control Section or External Symbol

The REPLACE statement can be used to delete a control section, entry name, or external reference in a module, as shown in the following example.

EXAMPLE: REPLACE ROUTINE1

ROUTINE1
is the name of a control section in the module following the control statement.

As a result, the control section ROUTINE1 is deleted from the input module. If no address constants refer to it from other control sections in the module, the control section name is also deleted. If address constants refer to the control section name, the name is retained as an external reference. Any external references to ROUTINE1 from other modules are unresolved.

CAUTION: Unresolved external references are not deleted from the output module even though a deleted control section contains the only reference to a symbol.

Multiple Editing Functions

A REPLACE statement can specify more than one function, as shown in the following example.

EXAMPLE:

REPLACE ROUTINE1(CSECT1),BEGIN,REPEAT

ROUTINE1
: is a control section in the module following this control statement, and is to be replaced by control section CSECT1.

BEGIN
: is an entry name referred to only in ROUTINE1.

REPEAT
: is an entry name in another control section in the module following this control statement. There are no address constants that refer to it within the module.

As a result, the control section ROUTINE1 is deleted from the module, which is then prepared to receive the new control section CSECT1 when it is processed in the input. BEGIN and REPEAT are deleted from the module.

PROVIDING SYSTEM STATUS INDEX INFORMATION

The following information is intended for systems personnel responsible for maintaining IBM-supplied load modules. It is not applicable to non-IBM load modules.

Four bytes in the library directory entry for IBM-supplied load modules are used to store system status index information. This information, which is used for maintenance of the modules, is placed in the directory entries by means of the SETSSI statement. For details on the use of the SETSSI statement, refer to the publication IBM System/360 Operating System: Maintenance, Form C27-6918.

SETSSI Statement

The SETSSI statement specifies hexadecimal information to be placed in the system status index of the output module library directory entry.

Operation	Operand
SETSSI	xxxxxxxx

xxxxxxxx
: represents eight characters of hexadecimal (0-9 and A-F) information that is placed in the 4-byte system status index of the output module library directory entry.

PLACEMENT: The SETSSI statement can be placed before, between, or after object modules or other control statements.

CAUTION: A SETSSI statement must be provided whenever an IBM-supplied load module is processed by the linkage editor. If the statement is omitted, no system status index information will be present.

DIAGNOSTIC OUTPUT AND SPECIAL PROCESSING

Diagnostic output and special processing options can be chosen to negate the effect of error conditions, or produce a module map, cross-reference table, or a listing of the control statements processed.

Two special processing options can be chosen in a single execution: LET or XCAL, and NCAL. Also, two diagnostic options can be chosen: MAP or XREF, and LIST. The option symbols are placed in the parameter (PARM) field of the EXEC statement that defines the linkage editor execution.

During a linkage editor execution, messages are generated to describe:

1. The options and attributes specified for the load module.
2. The disposition of the load module in the library.

EXAMPLE:

```
//STEPA EXEC PGM=IEWL,PARM='OVLY,TEST,
                XREF,XCAL'
```

As a result, the output module is in an overlay structure and valid exclusive branches are not regarded as errors. The testing symbol tables are placed in the output module for use by the test translator, and a cross-reference table is produced on the diagnostic output data set.

DIAGNOSTIC OUTPUT OPTIONS

In addition to diagnostic error and dispositional messages, the linkage editor will provide either a map or cross-reference table to show the structure of the load module produced.

Module Map

The module map shows all control sections in the output module and all entry names in each control section. The control sections are arranged in ascending order according to their assigned origins. All entry names are listed below the control section in which they are defined.

If the module is in an overlay structure, the control sections are arranged by segment. The segments are listed as they appear in the overlay structure, top to bottom, left to right, and region by region.

Named common areas are listed as control sections. The following are identified by a dollar sign:

- Blank common area.
- Private code (unnamed control section).
- Segment table entries.
- Entry table entries.

Each control section that is included from a library during automatic library call is indicated by an asterisk. At the end of the module map is the relative address of the instruction with which processing of the module begins. It is followed by the total length of the module in bytes. In the case of an overlay load module, the length is that of the longest path. The addresses shown in the module map are those assigned by the linkage editor prior to loading for execution.

EXAMPLE: Figure 23 shows a module map provided by the linkage editor. The output module is in overlay structure and contains eight segments. In the example, segments 3-7 are omitted. There are five control sections shown: $SEGTAB, BASIC, $ENTAB, AAAAAAAA, and GGGGGGGG.

The origin, length, and segment number of each control section are listed under the control section heading. Also provided is a listing of entry names within each control section. The entry point (the origin of BASIC,38) and total length of the output module (641 bytes) are listed at the end of the map.

Cross-Reference Table

The cross-reference table consists of a module map and a list of cross-references for each control section. Each address constant that refers to a symbol defined in another control section is listed with its assigned location, the symbol referred to, and the name of the control section in which the symbol is defined. For overlay programs, this information is provided for each segment. In addition, the number of the segment in which the symbol is defined is provided.

If a symbol is unresolved after processing by the linkage editor, it is identified by $UNRESOLVED in the list. However, if an unresolved symbol is marked by the never-call function, it is identified by $NEVER-CALL.

EXAMPLE: Figure 24 shows a cross-reference table provided by the linkage editor. Except for the cross-references, the infor-

```
 CONTROL SECTION                          ENTRY

   NAME    ORIGIN LENGTH SEG. NO.  NAME     LOCATION   NAME     LOCATION   NAME     LOCATION

$SEGTAB       00     38     1
 BASIC        38    281     1
                                  RETURN1       106  RETURN2       10E  RETURN3       116
                                  RETURN5       126  RETURN6       12E  BASEDUMP      136
$ENTAB       2C0     60     1

 AAAAAAAA    320    109     2
                                  ARETURN       376
-------------------------------------------------------------------------------------------
                                    Segments 3-7 not shown
-------------------------------------------------------------------------------------------
 GGGGGGGG    320    109     8
                                  GRETURN       376
ENTRY ADDRESS        38
TOTAL LENGTH        641
```

Figure 23. Module Map

```
 CONTROL SECTION                     ENTRY

   NAME    ORIGIN LENGTH SEG. NO.  NAME    LOCATION    NAME    LOCATION    NAME    LOCATION

$SEGTAB         00     38     1
 BASIC          38    281     1
                                 RETURN1       106  RETURN2       10E  RETURN3       116
                                 RETURN5       126  RETURN6       12E  BASEDUMP      136
$ENTAB         2C0     60     1


 LOCATION  REFERS TO SYMBOL   IN CONTROL SECTION  SEG. NO.
     184          AAAAAAAA         AAAAAAAA           2
     19C          GGGGGGGG         GGGGGGGG           8

 CONTROL SECTION                     ENTRY

   NAME    ORIGIN LENGTH SEG. NO.  NAME    LOCATION    NAME    LOCATION    NAME    LOCATION

 AAAAAAAA      320    109     2
                                 ARETURN       376
-------------------------------------------------------------------------------------------
                                   Segments 3-7 not shown
-------------------------------------------------------------------------------------------
 CONTROL SECTION                     ENTRY

   NAME    ORIGIN LENGTH SEG.NO.  NAME    LOCATION    NAME    LOCATION    NAME    LOCATION

 GGGGGGGG      320    109     8
                                 GRETURN       376


 LOCATION  REFERS TO SYMBOL   IN CONTROL SECTION  SEG. NO.

     38C          BASEDUMP         BASIC              1
ENTRY ADDRESS       38
TOTAL LENGTH       641
```

Figure 24. Cross-Reference Table

mation provided is identical to that provided by the module map (see Figure 23).

In Figure 24, location 19C refers to symbol GGGGGGGG in control section GGGGGGGG, located in segment eight. The remaining cross-references are identified in like manner.

Disposition Data

Information indicating the options and attributes specified is printed for each load module produced. Invalid options or attributes are replaced by "INVALID" in the printout. Messages are generated to inform the programmer that incompatible attributes have been specified.

There are, in addition, nine disposition messages used to indicate conditions of the load module in the output module library (see Figure 25). These messages are:

- (member name) NOW ADDED TO DATA SET.
- (member name) NOW REPLACED IN DATA SET.
- (member name) DOES NOT EXIST BUT HAS BEEN ADDED TO THE DATA SET -- the replacement function is specified; however, the member does not exist in the data set.
- (alias name) IS AN ALIAS FOR THIS MEMBER.
- MODULE HAS BECOME NOT EXECUTABLE.
- MODULE HAS BECOME REUSABLE.
- MODULE HAS BECOME NOT REUSABLE.
- MODULE HAS BECOME NOT EXECUTABLE AND REUSABLE.
- MODULE HAS BECOME NOT EXECUTABLE AND NOT REUSABLE.

DIAGNOSTIC MESSAGES

Certain conditions that are present when a module is being processed can cause an error or warning message to be printed. An error or warning message consists of message code and message text.

The message code is used to provide the programmer with the following information:

- The system component in which the error was noted (positions 1-3). IEW indicates a linkage editor message.
- The error message number (positions 4-6).
- The severity of the error (position 7).

Four types of severity codes (1-4) are generated, according to the magnitude of the error:

Type 1: Indicates a condition that may cause an error during execution of the output module. A module map or cross-reference table is produced if specified by the programmer. The output module is marked executable.

Type 2: Indicates an error that could make execution of the output module impossible. Processing continues. When possible, a module map or a cross-reference table is produced if specified by the programmer. The output module is marked "not executable" unless the LET option has been specified.

Type 3: Indicates an error that will make execution of the output module impossible. Processing continues. When possible, a module map or cross-reference table is produced if specified by the programmer. The output module is marked "not executable."

Type 4: Indicates an error condition from which no recovery is possible. Processing terminates. The only output is diagnostic messages.

Note: A severity code of zero (IEW0000) is generated for each control statement printed as a result of the LIST option. Severity zero does not indicate an error or warning condition.

The highest severity code encountered during processing is multiplied by 4 to create a return code that is placed into register 15 at the end of processing. The control program compares the return code with the values specified in the COND field of (1) the JOB control statement that was specified for this job step, and/or (2) the EXEC statement specified in any succeeding job step. The results of the comparisons are used to determine subsequent action. For details, refer to the publication _IBM System/360 Operating System: Job Control Language_.

The message text contains combinations of the following:

- The message classification (either error or warning).
- Cause of the error.
- Identification of the symbol, segment number (when in overlay), member, or input item to which the message applies.
- Instructions to the programmer.
- Actions taken by the linkage editor.

If an error is encountered during processing, the message code for that error is printed with the applicable symbol, symbols, or record in error. After processing has been completed, the diagnostic message associated with that message code is printed.

EXAMPLE: Figure 25 shows message codes printed during processing; and corresponding diagnostic messages printed after processing is complete.

The information printed during processing consists of options specified (top line), control statements used (IEW0000), message codes (IEW0201 and IEW0461), and disposition messages (****BBBBBBBB).

The information printed after processing is complete consists of the actual error or warning messages. If XREF or MAP is specified, the cross-reference table or module map follows the error messages.

```
   LINKAGE EDITOR OPTIONS SPECIFIED LET,NCAL,XREF,OVLY,LIST
IEW0000     NAME BBBBBBBB
IEW0201
IEW0461   CCCCCCCC
IEW0461   BASEDUMP
****BBBBBBBB NOW ADDED TO DATA SET
                                                   DIAGNOSTIC MESSAGE DIRECTORY


   IEW0201 WARNING - OVERLAY STRUCTURE CONTAINS ONLY ONE SEGMENT -- OVERLAY OPTION
           CANCELED.
   IEW0461 WARNING - SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE, NCAL WAS
           SPECIFIED.
```

Figure 25. Diagnostic Messages

In each of the examples in this appendix, the following assumptions are made:

1. Optional output (LIST, MAP/XREF) is placed in the diagnostic output data set if specified by the programmer.
2. Error messages also appear in the diagnostic output data set.
3. The buffer data set is used, as defined by the programmer, for any intermediate storage required.
4. Automatic library call is not needed unless explicitly covered in the example (Examples 6 and 7). This assumption merely simplifies the illustrations; automatic library call using the call library could be a part of any example in this appendix.

The actual sequence of control sections (i.e., the allocation of contiguous storage locations) may be quite different. This is of no consequence to the programmer since the control sections are logically connected as required during linkage editor processing, and the load module is a complete program.

In the case of programs in an overlay structure, as shown in Examples 8, 9, and 10, the control sections are not necessarily placed in the output module library in the same sequence as they appear in the input.

EXAMPLE 1: CARD SEQUENCES FOR LINKAGE EDITOR PROCESSING

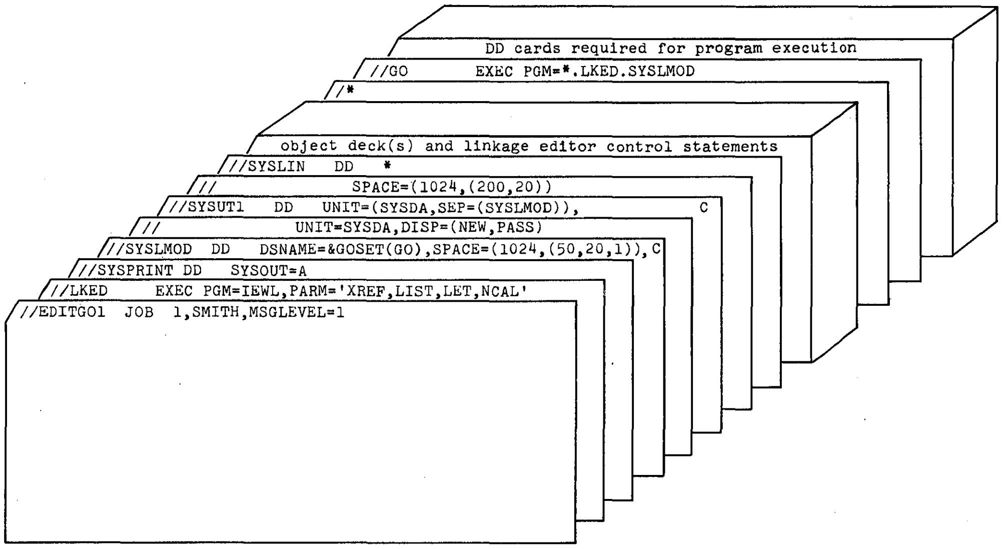
Figure 26 is a card sequence illustrating the job control statements necessary in the input stream to perform an edit and execute procedure.

Figure 27 illustrates a linkage editor step of a procedure for a compile, edit, and execute job. The linkage editor job step processes the input passed to it by an assembler or compiler step within the same job. In the linkage editor procedure, SYSLIN defines the input data set passed from an assembler or compiler procedure, SYSUT1 defines a buffer data set, SYSPRINT defines a data set in which messages are placed, and SYSLMOD defines an output module library. The partitioned data set named GOSET will be allocated a directory quantity of one 256-byte record.

For detailed information on the uses of the various subparameters, refer to the publication <u>IBM System/360 Operating System: Job Control Language</u>.

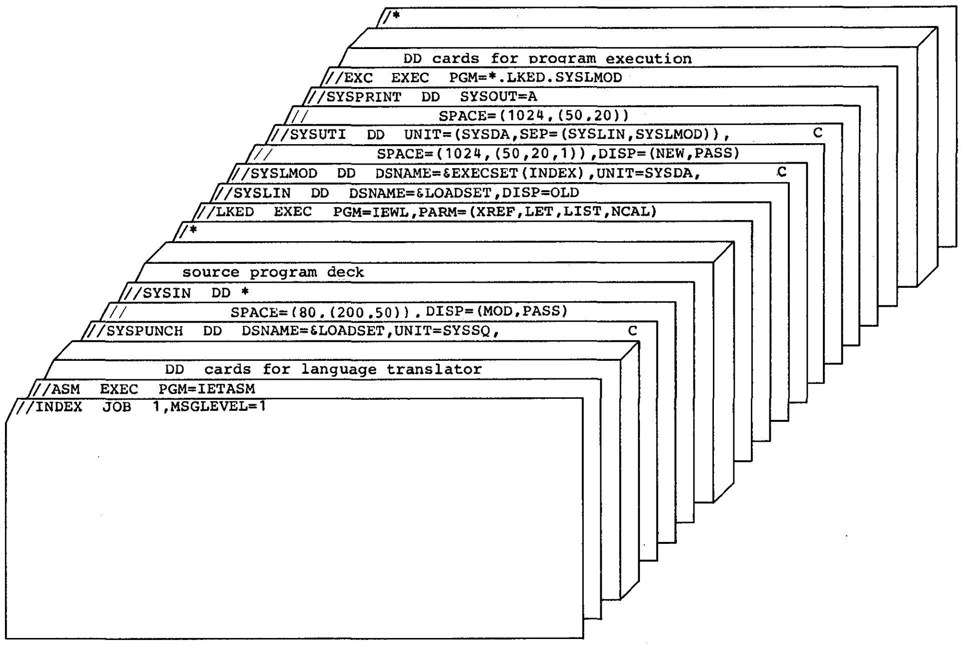
EXAMPLE 2: COMBINING TWO OBJECT MODULES

This example is illustrated in Figure 28. The primary input data set contains



```
//EDITGO1  JOB  1,SMITH,MSGLEVEL=1
//LKED      EXEC PGM=IEWL,PARM='XREF,LIST,LET,NCAL'
//SYSPRINT DD   SYSOUT=A
//SYSLMOD  DD   DSNAME=&GOSET(GO),SPACE=(1024,(50,20,1)),C
//                 UNIT=SYSDA,DISP=(NEW,PASS)
//SYSUT1   DD   UNIT=(SYSDA,SEP=(SYSLMOD)),               C
//                 SPACE=(1024,(200,20))
//SYSLIN   DD   *
 object deck(s) and linkage editor control statements
/*
//GO        EXEC PGM=*.LKED.SYSLMOD
         DD cards required for program execution
```

Figure 26. The Linkage Editor Step of an Edit and Execute Procedure



```
//INDEX  JOB  1,MSGLEVEL=1
//ASM  EXEC  PGM=IETASM
       DD  cards  for  language  translator
//SYSPUNCH  DD  DSNAME=&LOADSET,UNIT=SYSSQ,                         C
//             SPACE=(80,(200,50)),DISP=(MOD,PASS)
//SYSIN  DD  *
   source  program  deck
/*
//LKED  EXEC  PGM=IEWL,PARM=(XREF,LET,LIST,NCAL)
//SYSLIN  DD  DSNAME=&LOADSET,DISP=OLD
//SYSLMOD  DD  DSNAME=&EXECSET(INDEX),UNIT=SYSDA,                    C
//             SPACE=(1024,(50,20,1)),DISP=(NEW,PASS)
//SYSUTI  DD  UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                     C
//             SPACE=(1024,(50,20))
//SYSPRINT  DD  SYSOUT=A
//EXC  EXEC  PGM=*.LKED.SYSLMOD
       DD  cards  for  program  execution
/*
```

Figure 27. The Linkage Editor Step of a Compile, Edit, and Execute Procedure

two object modules. The overall operation is a simple object module combination; no control statements are required. Each module has four control sections. The end of each module is specified by an END statement, which can indicate where execution of the load module would begin.

The primary input data set is the only input source. It is processed in sequence from CS1 to the END Y statement. The linkage editor combines the two object modules into one load module and places that module in the output module library.

The load module in the output module library can be loaded and executed by the control program. To initiate execution of that load module, the programmer places job control statements in the primary input data set. When the load module receives control, the execution will begin at location X.

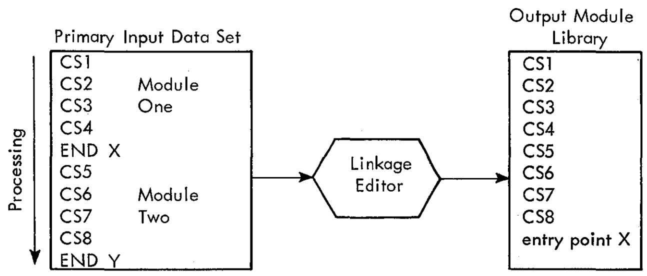


Figure 28. Module Linkage

EXAMPLE 3: COMBINING LOAD MODULES

In Figure 29, the INCLUDE statements in the primary input data set direct the linkage editor to select members of additional libraries that are defined for the job step. These members (all load modules) are the input to be combined.

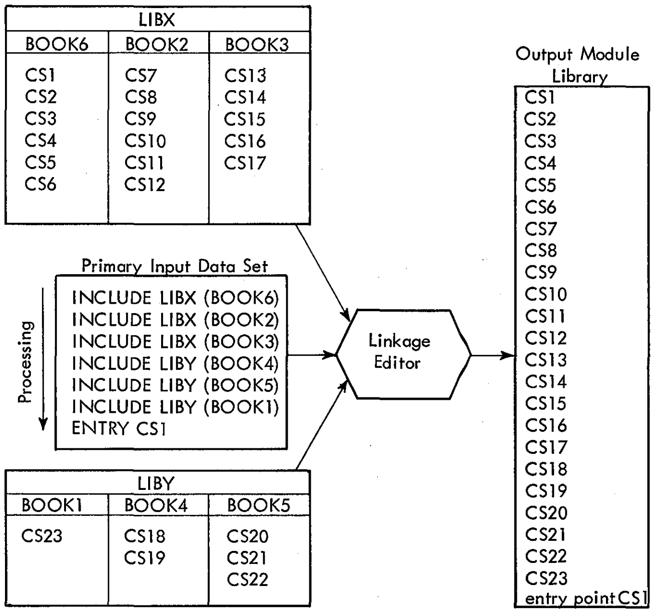


Figure 29. Combining Input From Libraries

The primary input data set is processed sequentially from INCLUDE LIBX(BOOK6) to

ENTRY CS1. The load modules are retrieved and processed in the order specified by the INCLUDE statements.

The control sections of the input modules are placed in the output module library, with main storage addresses assigned beginning at zero. CS1, the entry point of the output module, denotes the first instruction to be executed when the module is loaded and given control. CS1 is used because it was specified on an ENTRY statement in the primary data set.

EXAMPLE 4: COMBINING AND EDITING OBJECT MODULES

The primary input data set (Figure 30) contains three object modules. Each object module ends with an END statement. Control statements are placed between the object modules to specify how the modules are to be edited.

The primary input data set is processed sequentially from CS1 to the END Z statement. CS5 is replaced by CS3 as requested by the first REPLACE statement. The control section CS5 is deleted from the program. All references in module 2 to the symbol CS5 are resolved by the symbol CS3. In the example, all references to entry names in CS5 are resolved with entry names in CS3. The external symbol SYM1 is changed to SYM2 as requested by the CHANGE statement, if SYM1 appears in CS8, CS9, or CS10. Control section CS10 is deleted from the program as requested by the second REPLACE control statement. In the example, all external references to CS10 are assumed to be resolved.

The control sections of the input modules, except for CS5 and CS10, are placed in the output library with main storage addresses assigned beginning at zero. START, the entry point of the output module, denotes the first instruction to be executed when the module is loaded and given control.

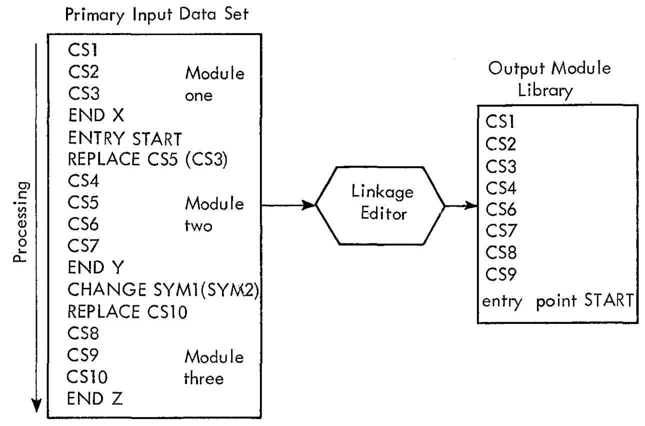


Figure 30. Editing Modules

EXAMPLE 5: INCLUDING MODULES FROM MANY ADDITIONAL INPUT SOURCES

The primary input data set for this example, shown in Figure 31, contains one object module and three control statements. It is processed sequentially from INCLUDE LIBA(BOOK2) to INCLUDE LIBA(BOOK1). LIBA is the ddname of an object module library. The first module processed is contained in BOOK2, and the linkage editor is directed to that module by the first INCLUDE statement in the primary input data set. When BOOK2 has been processed, the linkage editor returns to the primary input data set for the next input item. This is the

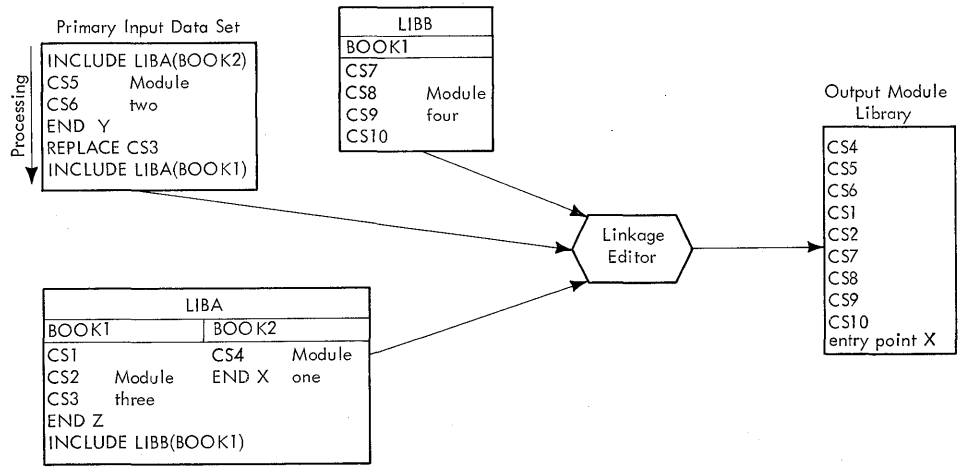


Figure 31. Collecting Library Modules

second module processed. It contains CS5, CS6, and the END Y statement. The REPLACE statement following the second module applies to the module indicated by the second INCLUDE statement. This module, BOOK1, is in LIBA. It is the third module processed. During its processing, control section CS3 is deleted. The INCLUDE statement in BOOK1 of LIBA is processed immediately. The statement directs the linkage editor to LIBB(BOOK1) for the fourth input module.

The control sections of the input modules, except for CS3, are placed in the output library with main storage addresses assigned beginning at zero. X, the entry point of the output module, denotes the first instruction to be executed when the module is loaded and given control. X is used because it was the symbol specified in the first END statement in the input, and no ENTRY statement appeared in the input.

EXAMPLE 6: COMBINING MODULES AND STANDARD ROUTINES

This example is illustrated in Figure 32. The primary input data set contains one object module and four control statements. It is processed sequentially from CS1 to ENTRY START. The first input processed is the object module in the primary input data set. The REPLACE statement, which is next, applies to the module in BOOK1 of LIBRYX, as indicated in the INCLUDE statement. CS2 replaces CS4. For the purpose of this example, it is assumed that all references to CS4 and entry names within CS4 are resolved, except for a reference to MATH. After processing module two, the linkage editor INCLUDE statement directs the linkage editor to BOOK2 in LIBRYX. Module three is processed from that member and the linkage editor returns to the primary input data set to process the ENTRY statement. Primary input processing is complete at this point, however, an unresolved external reference to MATH still remains. The automatic library call mechanism, therefore, searches the call library for a module that will resolve the reference. If MATH exists (as assumed here) as a member in the call library, the linkage editor processes MATH as the fourth input module.

All control sections of the input modules, except CS4, are placed in the output library with main storage addresses assigned beginning at zero. START, the entry point of the output module, denotes the first instruction to be executed when the module is loaded and control is passed to it. START is used because it was indicated in the ENTRY statement that overrides all END statements in the input. Entry points for input load modules are ignored.

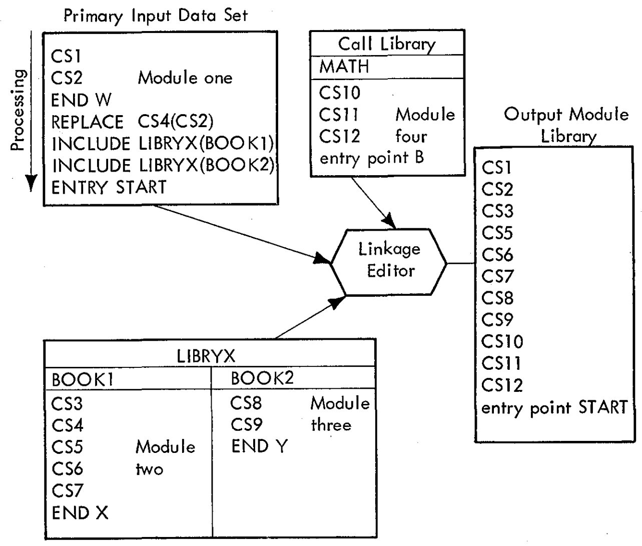


Figure 32. Automatic Library CALL

EXAMPLE 7: EDITING BY AUTOMATIC REPLACEMENT

The primary input data set for this example, shown in Figure 33, contains one control statement and a module consisting of only one control section. The LIBRARY statement indicates that a reference to CS1 can be resolved with the load module in CS1 of LIBA, and not with a library member in the call library. The control section CS5 contains at least one external reference to the symbol CS1.

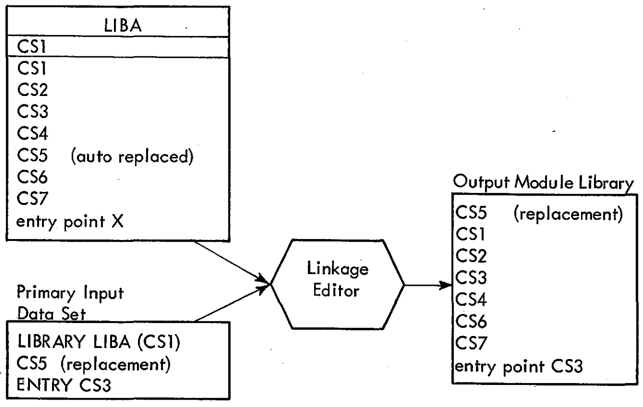


Figure 33. Editing by Automatic Replacement

Automatic library call retrieves the load module, and the linkage editor combines it with the control section in the primary input. Since there are two control sections named CS5 in the linkage editor input, automatic replacement accepts the first CS5 and deletes the second.

The control sections in the input, except the second control section named CS5, are placed in the output module library.

EXAMPLE 8: PROCESSING OVERLAY PROGRAMS (WITH ONLY PRIMARY INPUT)

As illustrated in Figure 34, the primary input data set for this example consists of six object modules separated by OVERLAY statements. The organization of the primary input data set reflects the overlay tree structure. The primary input data set is processed sequentially from the ENTRY statement to the END F statement.

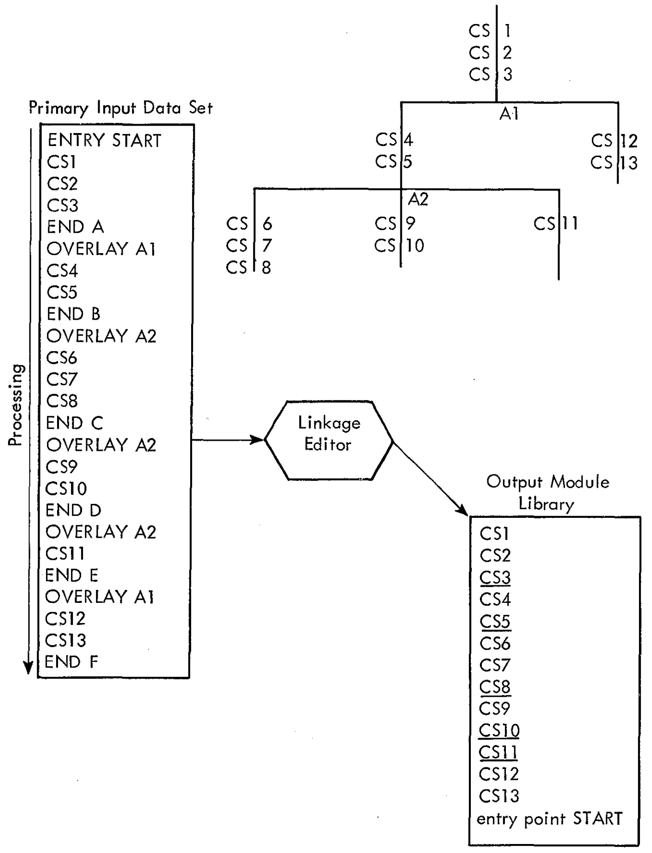


Figure 34. Processing an Overlay Program

The control sections of the input modules are placed in the output library. A1 is a symbolic origin whose numeric value is the length of module one. A2 is a symbolic origin whose numeric value is the sum of the lengths of modules one and two. START, the entry point of the output module, denotes the first instruction to be executed when the root segment is loaded for execution. When the program is given control, the root segment is the first segment loaded by the control program. Other segments are loaded by the control program when requested by the segment being executed.

EXAMPLE 9: PROCESSING AN OVERLAY PROGRAM FROM LIBRARIES

The primary input data set for this example, shown in Figure 35, contains a series of control statements and one module. The primary input data set is processed sequentially from ENTRY START to INCLUDE LIBA(BOOK2). The INCLUDE statements in the primary input data set direct the linkage editor to libraries that contain the majority of the control sections of the program. The control statements in the primary input data set are organized to reflect the overlay tree structure shown.

This example differs from the previous one in that the control sections of the program are not included in the primary input data set. They are represented in the primary input by INCLUDE statements. When the control sections are to be processed, these statements direct the linkage editor to the library members that contain the control sections.

EXAMPLE 10: PROCESSING AN OVERLAY PROGRAM USING THE INSERT CONTROL STATEMENT

This example is illustrated in Figure 36. The primary input data set consists of linkage editor to BOOK1 and BOOK2 in LIBA. The control statements in BOOK1 are organized to reflect the overlay tree structure shown. The order in which the library items are processed does not matter, nor does the INCLUDE statement indicate any order. No matter what the order of processing, the control sections in BOOK2 are arranged in the output module in the order indicated by the control statements in BOOK1. The output module is exactly the same as the output modules produced in Examples 8 and 9.

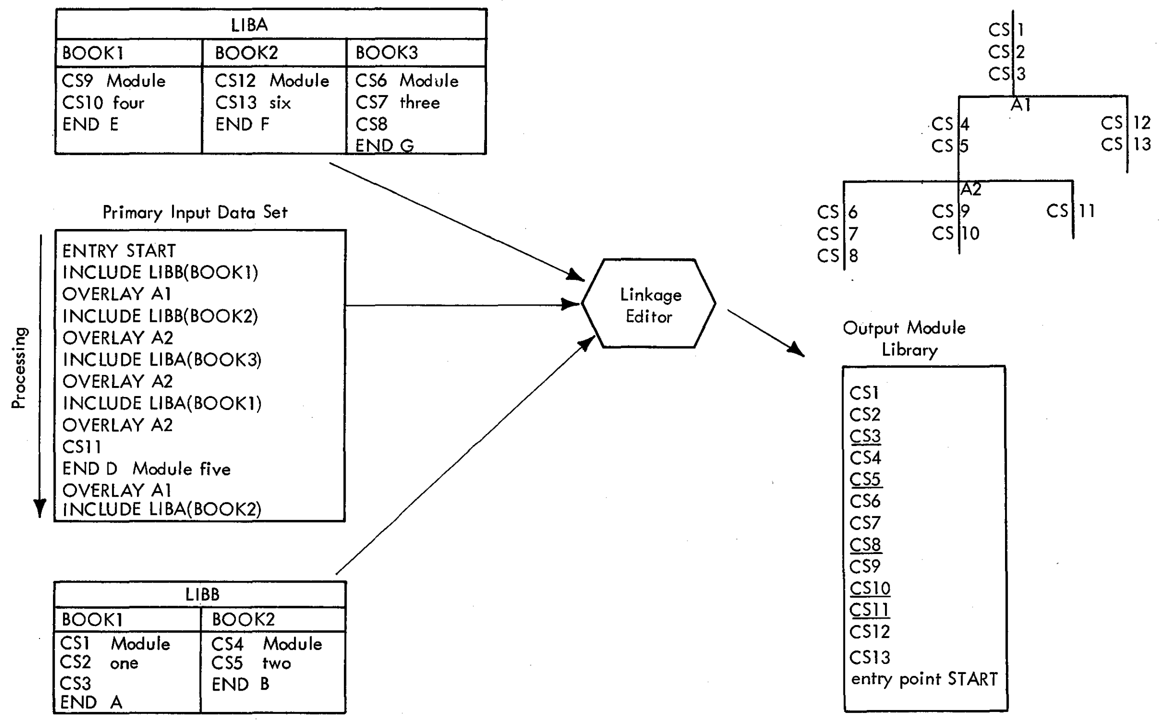


Figure 35. Processing an Overlay Program From Libraries

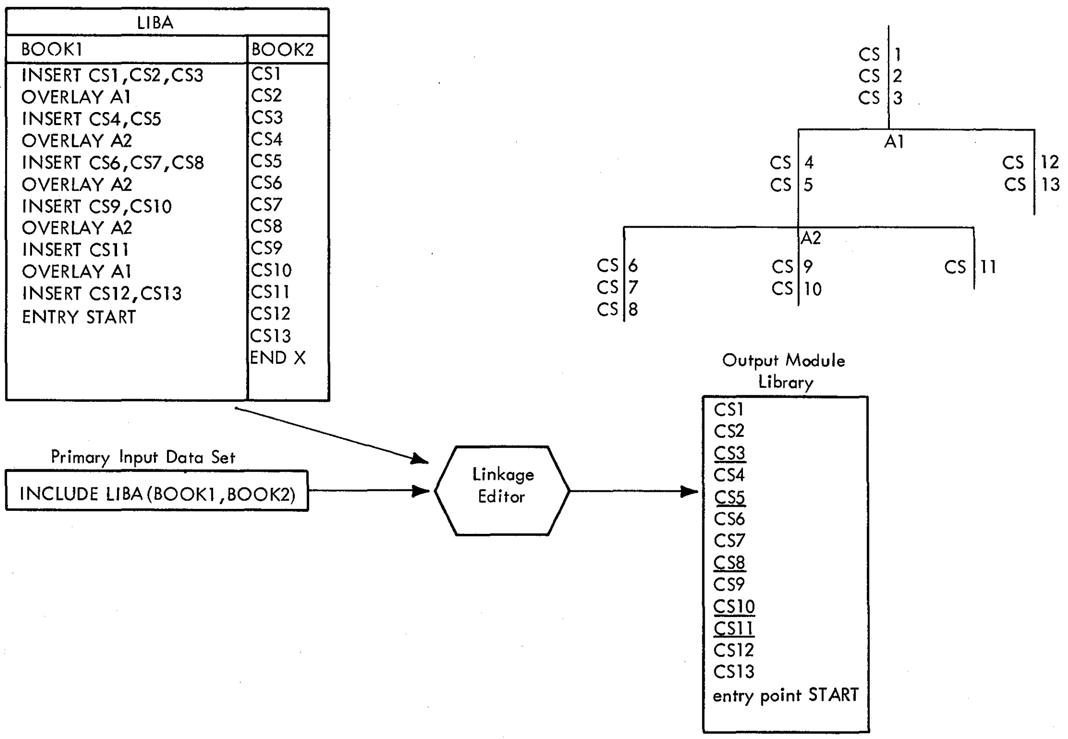


Figure 36. Processing an Overlay Program With Insert Statement

ALIAS	externalname

Function: To provide up to five entry points in addition to the one attributed to the module name. Execution of the module will begin at the alias entry point referred to.

Placement: Before, between, or after object modules or other control statements.

Caution: In an overlay program, the first instruction to be executed must be in the root segment.

ALIAS	symbol

Function: To specify a name that is not an external name within the output module. The relative address referred to, when the module is called for execution is that of the main entry point.

Placement: Before, between, or after object modules or other control statements.

CHANGE	oldcontrolsectionname (newcontrolsectionname)

Function: To change the name of a control section.

Placement: Immediately before the module or INCLUDE statement specifying the module that contains the control section.

Caution: External references from other modules to the old control section name will not be changed.

CHANGE	oldentryname(newentry name)

Function: To change an entry name.

Placement: Immediately before the module or INCLUDE statement that specifies the module containing the entry name.

Caution: External references from other modules to the old entry name are not changed.

CHANGE	oldexternalreference(new externalreference)

Function: To change an external reference to a name defined in another module.

Placement: Immediately before the module or the INCLUDE statement specifying the module that contains the external reference.

ENTRY	externalname

Function: To specify the name of the first instruction to be executed when the module is referred to by its module name.

Placement: Between linkage editor input object modules or control statements.

Caution: In an overlay program, the first instruction to be executed must be in the root segment.

INCLUDE	ddname

Function: To specify the data definition name of a sequential data set containing an object module or modules to be included in the output module.

Placement: Between linkage editor input object modules and control statements, at the point at which the module is to be included.

Caution: Data sets are not necessarily processed in the order in which they appear in the INCLUDE statement.

INCLUDE	ddname(membername)

Function: To specify a module that is a named member of a partitioned data set and is to be included as part of the output module.

Placement: Between linkage editor input object modules and control statements, at the point at which the module is to be included.

Caution: The modules are not necessarily processed in the order in which they appear in the INCLUDE statement.

INSERT	controlsectionname

Function: To reposition a control section from its position in the input sequence to a segment in an overlay program.

Placement: In the input sequence that will become the segment into which the control section is to be placed.

LIBRARY	ddname(membername)

Function: To specify to the automatic library call a library containing modules that will resolve external references found in the program.

Placement: Between linkage editor input object modules and control statements.

Caution: Members called by automatic library call will be placed in the root segment of an overlay program, unless they are placed elsewhere by the INSERT statement.

LIBRARY	(externalreference)

Function: To specify to the automatic library call those external references that are not to be resolved by the automatic library call mechanism during the current linkage editor run.

Placement: Between linkage editor input object modules and control statements.

LIBRARY	*(externalreference)

Function: To specify to automatic libary call those external references that are not to be resolved by the automatic libary call mechanism during any linkage editor run.

Placement: Between linkage editor input object modules and control statements.

NAME	symbol

Function: To specify the name of the load module that was just created, and to serve as a delimiter for input to the load module.

Placement: At the end of the input used to form a load module.

NAME	symbol(R)

Function: To specify the name of a load module that replaces an identically named load module previously placed in the library and to serve as a delimiter to the load module.

Placement: At the end of the input used to form a load module.

OVERLAY	symbol

Function: To specify the symbolic origin of an overlay segment.

Placement: Immediately before either the first module of a new segment or an INCLUDE statement specifying the first module, or before an INSERT statement that specifies control sections to be placed in the segment.

OVERLAY	symbol(REGION)

Function: To specify the symbolic origin of a new region.

Placement: The OVERLAY statement must precede the first module of the next segment, the INCLUDE statement specifying the first module of the segment, or the INSERT statement specifying the control sections to be positioned in that segment.

REPLACE	oldcontrolsectionname(new controlsectionname)

Function: To replace one control section with another.

Placement: Immediately before either the module that contains the control section to be replaced or the INCLUDE statement specifying the module.

Caution: When control sections that were or are part of a separately assembled module are to be replaced, A-type address constants that refer to a deleted symbol will be incorrectly resolved unless the entry name is in the same position relative to the origin of the replaced control section and the new control section.

REPLACE	controlsectionname

Function: To delete a control section from its module.

Placement: Immediately before either the module that contains the control section to be deleted or the INCLUDE statement specifying the module.

Caution: The control section is deleted, but the control section name will be changed to an external reference if there are any references to it from other control sections in the same input module. Automatic library call will attempt to resolve the external reference.

External references from other modules to a deleted control section cause automatic library call to attempt to resolve them if they have not been resolved by other input modules.

Unresolved external references are not deleted from the output module even though a deleted control section contains the only reference to a symbol.

REPLACE	entryname

Function: To delete an entry name from a module.

Placement: Immediately before either the module that contains the entry name or the INCLUDE statement specifying the module.

Caution: The entry name will be changed to an external reference if there are any references to it within the same input module.

External references from other modules to a deleted entry name will cause automatic library call to attempt to resolve them if they have not been resolved by other input modules.

REPLACE	externalreference

Function: To delete an external reference for which no RLD entry exists.

Placement: Immediately before either the module that contains the external reference to be deleted or the INCLUDE statement specifying the module.

SETSSI	xxxxxxxx

Function: To store system status index information (hexadecimal) in a library directory entry for an output module.

Placement: Before, between, or after object modules or other control statements.

APPENDIX C: INVOCATION OF THE LINKAGE EDITOR

The linkage editor can be invoked by a problem program at execution time through the use of the ATTACH or LINK macro-instruction.

The problem program must supply the following information to the linkage editor:

- The options and attributes for the load module.
- The ddnames of the data sets to be used during processing by the linkage editor.

Name	Operation	Operand

[symbol]	LINK
	ATTACH
		EP=linkeditname,
		PARAM=(optionlist
		[,ddnamelist]),VL=1

EP
: specifies the symbolic name of the linkage editor. The entry point at which execution is to begin is determined by the control program (from the library directory entry).

PARAM
: specifies, as a sublist, address parameters to be passed from the problem program to the linkage editor. The first full-word in the address parameter list contains the address of the option and attribute list for the load module. The second full-word contains the address of the ddname list. If standard ddnames are to be used, this list may be omitted.

option list
: specifies the address of a variable length list containing the options and attributes. This address must be written even though no list is provided.

The option list must begin on a half-word boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. If no options or attributes are specified, the count must be zero. The option list is free form with each field separated by a comma. No blanks or zeros should appear in the list.

ddname list
: specifies the address of a variable length list containing alternative ddnames for the data sets used during linkage editor processing. If standard ddnames are used, this operand may be omitted.

The ddname list must begin on a half-word boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. Each name of less than eight bytes must be left justified and padded with blanks. If an alternate ddname is omitted from the list, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end by merely shortening the list.

The sequence of the 8-byte entries in the ddname list is as follows:

<u>Entry</u>	<u>Alternate Name For:</u>
1	syslin
2	member name
3	syslmod
4	syslib
5	not applicable
6	sysprint
7	not applicable
8	sysut1

VL
: specifies that the sign bit is to be set to 1 in the last full-word of the address parameter list.

When the linkage editor completes processing, a condition code is returned in register 15. See "Linkage Editor Completion Code."

Two design levels of the linkage editor program are available: level E and level F. Both levels can operate as part of any System/360 Operating System; for a particular system, the linkage editor program is selected during system generation. The following discussion contrasts the two levels.

Level E: This level is intended for a 32K[1] computing system; however, it can be executed in a larger main storage.

- Program sizes. Two level E programs are available: 15K and 18K. These sizes represent the minimum amounts of main storage that must be available for each of the programs.

- Capacities. When the main storage available to either of these programs is increased, the program has increased capacities for external symbol dictionary entries, intermediate text records, relocation dictionary records, and segments per program. The overlay regions per program, blocking factor for input object modules, and intermediate and output text record lengths remain constant.

In comparison: for a given amount of available main storage, the 15K program has bigger capacities, but the 18K program is faster.

Level F: This level is intended for a 64K, 128K, or larger computing system.

- Program sizes. Two level F programs are available: 44K and 88K. These sizes represent the minimum amounts of main storage that must be available for each of the programs.

- Capacities. The 44K and 88K programs have fixed capacities for external symbol dictionary entries, intermediate text records, relocation dictionary records, segments per program, overlay regions per program, and blocking factor for input object modules. However, as the amount of available storage increases, the size of intermediate and output text records may increase up to 18,432 bytes maximum, depending on the direct-access devices being used.

In comparison: the 88K program has greater capacities than the 44K program. In comparison to the level E programs, the level F programs have more speed and generally greater capacities; therefore, when 44K or more of main storage is available to the linkage editor, a level F program should be used.

Capacities

The capacities of the four linkage editor programs are shown in Table 1. For the level E programs, the capacities are given first for the program in the minimum amount of available main storage and then for the program in a larger amount that reflects a typical machine size.

For the composite external symbol dictionary, the number of entries permitted for a particular program can be computed by subtracting, from the maximum number given in Table 1, one entry for each of the following:

- A data definition name (ddname) specified in LIBRARY statements.

- A data definition name (ddname) specified in INCLUDE statements.

- An ALIAS statement.

- A symbol in REPLACE or CHANGE statements that are in the largest group of such statements preceding a single object module in the input to the linkage editor.

- The segment table (SEGTAB) in an overlay program.

- An entry table (ENTAB) in an overlay program.

To compute the number of intermediate text records that will be produced during processing of a particular program, add one record for each group of x bytes within each control section, where x is the block size for the intermediate or buffer data set. For the level E programs, x is 1,024; for the level F programs, x is 2,048 minimum and 18,432 maximum, depending on the amount of main storage available to the linkage editor and the devices allocated

---
[1]1K=1,024 bytes

Table 1. Capacities of Linkage Editor Programs

Linkage Editor Program		15K E Level		18K E Level		44K F Level	88K F Level
Main Storage Allocated to Program (in Bytes)		15K	18K	18K	20K	44K	88K
Maximum number of entries in composite external symbol dictionary (ESD)		119	229	75	140	349	999
Maximum number of intermediate text records		67	147	35	83	213	501
Maximum number of relocation dictionary (RLD) records		63	143	31	79	147	357
Maximum number of segments per program		33	38	32	34	63	255
Maximum number of overlay regions per program		4	4	4	4	4	4
Maximum blocking factor for input object modules (i.e., number of 80-column card images per physical record)		1	1	1	1	5	40
Output Text Record Length (in Bytes)	On IBM 2311 Disk Storage Drive	1024	1024	1024	1024	3072[1]	3072[1]
	On IBM 2301/2303 Drum Storage	1024	1024	1024	1024	6144[1]	18432[1]

[1]When downward compatibility has been specified, this value is 1024.

for the intermediate and output data sets. In determining the number of records, add one record for a remainder of less than x bytes.

The number of text records that can be handled by a linkage editor program is less than the maximums given in Table 1 if the text of one or more control sections is not in order by address in the input to the linkage editor.

To compute the number of relocation dictionary records in a particular program, add one record for each group of y relocatable address constants within each control section, where y is 30 for the level E programs, 147 for the 44K level F program, and 357 for the 88K level F program. In determining the number of records, add one record for a remainder of less than y address constants.

Intermediate Data Set

The intermediate, or buffer, data set (SYSUT1) is used by the linkage editor to hold intermediate data records during processing of a problem program. The level E linkage editor programs always place intermediate data in this data set. The level F linkage editor programs place intermediate data in this data set only when all the input data cannot be held in available main storage. (However, whether this data set will or will not be used during a level F linkage editor execution, it must always be defined.)

The following direct-access devices, if supported by the system, can be used for this data set:

IBM 2311 Disk Storage Drive
IBM 2301 Drum Storage Drive
IBM 2303 Drum Storage Drive

This appendix, message appendixes in other publications, and the publication IBM System/360 Operating System: Control Program Messages and Completion Codes, Form C28-6608, are designed so that the user can select the messages applicable to his installation and incorporate them in a binder.

Severity codes used in linkage editor diagnostic messages appear as the final position of the message code and are defined as follows:

Severity Code	Meaning
1	Indicates a condition that may cause an error during execution of the output module. A module map or cross-reference table is produced if specified by the programmer. The output module is marked executable.
2	Indicates an error that could make execution of the output module impossible. Processing continues. When possible, a module map or a cross-reference table is produced if specified by the programmer. The output module is marked "not executable" unless the LET option has been specified.
3	Indicates an error that will make execution of the output module impossible. Processing continues. When possible, a module map or cross-reference table is produced if specified by the programmer. The output module is marked "not executable."
4	Indicates an error condition from which no recovery is possible. Processing terminates. The only output is diagnostic messages.
Note: A severity code of zero (IEW0000) is generated for each control statement printed as a result of the LIST option. Severity code zero does not indicate an error or warning condition.	

IEW0012 ERROR-INPUT CONTAINS INVALID TWO-BYTE RELOCATABLE ADDRESS CONSTANT, CONSTANT HAS NOT BEEN RELOCATED.

Explanation: A relocatable A-type or V-type address constant of less than three bytes has been found in the input.

System Action: The constant is not relocated.

User Response: Delete or correct the invalid address constant.

IEW0022 ERROR - INPUT CONTAINS INVALID V-TYPE ADDRESS CONSTANT, CONSTANT HAS NOT BEEN RELOCATED.

Explanation: A V-type address constant of less than four bytes has been found in the overlay structure.

System action: The constant is not relocated.

User Response: Delete or correct the invalid V-type address constant.

IEW0033 ERROR - INVALID ENTRY POINT FROM END CARD, NO ENTRY POINT ASSIGNED.

Explanation: The entry point for the program was specified as a relative address in an END card. The entry point that was specified appeared to be valid when the END card was processed, however, the entry point was found to be invalid when the entry point of the load module was being determined.

System Action: No entry point is assigned.

User Response: Remove invalid entry point specification from the input. Specify a valid entry point.

IEW0043 ERROR - INPUT CONTAINS INVALID EXTERNAL SYMBOL ID.

Explanation: An ESD card is probably mispunched.

System Action: The invalid item is ignored.

IEW0053 ERROR - ENTRY STATEMENT SYMBOL PRINTED IS INVALID (NOT AN EXTERNAL NAME), NO ENTRY POINT ASSIGNED.

Explanation: The symbolic entry point specified in an ENTRY statement is not a control section or entry name.

System Action: No entry point is assigned.

User Response: Remove the invalid entry point specification from the input. Specify a valid entry point.

IEW0063 ERROR - END CARD SYMBOL PRINTED IS INVALID (NOT AN EXTERNAL NAME), NO ENTRY POINT ASSIGNED.

Explanation: The symbolic entry point specified in an END statement is not a control section or entry name.

System Action: No entry point is assigned.

User Response: Remove the invalid entry point specification from the input. Specify a valid entry point.

IEW0073 ERROR - ENTRY STATEMENT SYMBOL PRINTED IS NOT IN ROOT SEGMENT OF OVERLAY STRUCTURE, NO ENTRY POINT ASSIGNED.

Explanation: The entry point specified by the programmer is in a segment other than the root segment. Either, (1) the module containing the entry point was placed in a segment other than the root segment by means of the INSERT statement, or (2) the entry point is incorrectly specified on the ENTRY statement.

System Action: No entry point is assigned.

User Response: Specify an entry point in the root segment.

IEW0083 ERROR - END CARD SYMBOL PRINTED IS NOT IN ROOT SEGMENT OF OVERLAY STRUCTURE, NO ENTRY POINT ASSIGNED.

Explanation: The entry point is in a segment other than the root segment Either, (1) the INSERT statement was used to place the control section containing the entry point in another segment, or (2) th symbol specified on the END statement is incorrect.

System Action: No entry point i assigned.

User Response: Specify an entr point in the root segment.

IEW0093 ERROR - END CARD ENTRY POIN ADDRESS PRINTED IS NOT IN ROO SEGMENT OF OVERLAY STRUCTURE, N ENTRY POINT ASSIGNED.

Explanation: The entry point is i a segment other than the root segment. Either, (1) the INSER statement was used to place th control section containing th entry point in another segment, o (2) the address specified on th END statement is incorrect.

System Action: No entry point i assigned.

User Response: Specify an entr point in the root segment.

IEW0102 ERROR - INVALID ENTRY POINT ON EN CARD, ENTRY POINT IGNORED.

Explanation: A possible entry point for the program was specified as a relative address in an END card. When the END card was processed, the control section identificatio of the specified entry point wa found to be invalid.

System Action: The entry point i ignored. The first valid entr point encountered is used; if ther is none, no entry point i assigned.

User Response: Remove invalid entr point specification from the input. Specify a valid entry point.

IEW0113 ERROR - OUTPUT MODULE CONTAINS N CONTROL SECTIONS IN ROOT SEGMENT O OVERLAY STRUCTURE, NO ENTRY POIN ASSIGNED.

Explanation: There are no control sections in the root segment. (1) All control sections originally i the root segment have been deleted, or (2) there were no control sections originally in the root segment, or (3) an OVERLAY statement preceded the input.

System Action: No entry point is assigned.

User Response: Place at least one control section in the root segment. Specify a valid entry point.

IEW0123 ERROR - NO ESD ENTRIES, EXECUTION IMPOSSIBLE.

Explanation: There are no external symbol dictionary entries. There are no control sections in the output.

IEW0132 ERROR - SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE.

Explanation: An external reference is unresolved at the end of input processing. None of the following is specified: restricted no-call, never-call, or NCAL.

System Action: The module is marked "not executable" unless LET is specified.

User Response: Either, (1) specify the proper library or module to resolve the external reference, or (2) specify NCAL, never-call or restricted no-call.

IEW0143 ERROR - NO TEXT.

Explanation: No text remains in the output module. Either, (1) all the control sections originally in the input are deleted, or (2) there are no control sections that originally contained text.

IEW0152 ERROR - INVALID OVERLAY STRUCTURE, NO CALLS OR BRANCHES MADE FROM ROOT SEGMENT.

Explanation: There are no calls or branches from the root segment to a segment lower in the tree structure. Other segments cannot be loaded.

System Action: The module is marked "not executable" unless LET is specified.

IEW0161 WARNING - EXCLUSIVE CALL FROM SEGMENT NUMBER PRINTED TO SYMBOL PRINTED -- XCAL WAS SPECIFIED.

Explanation: There is a valid exclusive reference; the XCAL option is specified for this job step.

IEW0172 ERROR - EXCLUSIVE CALL FROM SEGMENT NUMBER PRINTED TO SYMBOL PRINTED.

Explanation: A valid reference is made from a segment to an exclusive segment; XCAL is not specified.

System Action: The module is marked "not executable" unless the LET option is specified.

User Response: Either, (1) rearrange the overlay structure to place both segments in the same path, or (2) specify XCAL.

IEW0182 ERROR - INVALID EXCLUSIVE CALL FROM SEGMENT NUMBER PRINTED TO SYMBOL PRINTED.

Explanation: There is an invalid exclusive reference from a segment to a symbol in an exclusive segment.

System Action: The module is marked "not executable" unless the LET option is specified.

User Response: Either, (1) place the segments in the same path, or (2) place a V-type address constant in a common segment.

IEW0193 ERROR - MAIN STORAGE REQUIREMENTS FOR OUTPUT LOAD MODULE NOT FEASIBLE.

Explanation: Address assignment limits have been exceeded. The maximum number allowed by the address field of ESD items is $2^{24}-1$.

System Action: Processing continues if possible. The module is marked "not executable."

IEW0201 WARNING - OVERLAY STRUCTURE CONTAINS ONLY ONE SEGMENT -- OVERLAY OPTION CANCELLED.

Explanation: There are no OVERLAY statements in the input.

System Action: The overlay option is canceled.

User Response: Either, (1) place overlay statements in the input, or

(2) remove the overlay option specification.

IEW0212 ERROR - EXPECTED CONTINUATION CARD NOT FOUND.

Explanation: A linkage editor control statement specifying a continuation (nonblank in column 72) is not followed by a continuation card.

System Action: The card is not processed as a continuation, but as normal input.

IEW0222 ERROR - CARD PRINTED CONTAINS INVALID INPUT FROM OBJECT MODULE.

Explanation: A control statement may have been placed within an object module.

System Action: The questionable record is ignored and processing continues.

IEW0232 ERROR - INPUT FROM LOAD MODULE IS INVALID.

System Action: The questionable record is ignored and processing continues.

IEW0241 WARNING - EXTERNAL SYMBOL PRINTED IS DOUBLY DEFINED -- ESD TYPE DEFINITIONS CONFLICT.

Explanation: Two identical external names have been found in the input, at least one of which is not a control section name. References to the name are resolved with respect to the first occurrence of the name. If the second occurrence is a control section name, the control section is deleted.

IEW0254 ERROR - TABLE OVERFLOW -- TOO MANY EXTERNAL SYMBOLS IN ESD.

Explanation: There are too many external symbols or control statement operands in the problem program.

User Response: Either, (1) combine control sections, or (2) delete unneeded EXTRN and ENTRY statements. If this is not sufficient, process in a larger main storage environment.

IEW0264 ERROR - TABLE OVERFLOW -- INPU MODULE CONTAINS TOO MANY EXTERNA SYMBOLS IN ESD.

Explanation: Either, (1) an inpu module contains too many externa symbols in the ESD, or (2) an ES card is mispunched.

User Response: Process the proble program in a larger main storag environment.

IEW0272 ERROR - LOAD MODULE FROM LIBRAR SPECIFIED UNACCEPTABLE TO LEVEL E

Explanation: The downward compat ible attribute was not specifie when the load module was created b the level F linkage editor.

System Action: The load module i not accepted as input.

User Response: Reprocess using th level F linkage editor. Specif the downward compatible attribute

IEW0284 ERROR - DDNAME PRINTED CANNOT B OPENED.

Explanation: The specified data se cannot be opened. The DD statemen defining the data set is missing.

IEW0294 ERROR - DDNAME PRINTED HAD SYNCHRO NOUS ERROR.

Explanation: The error is mos likely a hardware error.

IEW0302 ERROR - INVALID STATEMENT -- SCA TERMINATED.

Explanation: There is an error on a linkage editor control statement.

System Action: The statement i accepted as input up to the poin of the error.

User Response: Correct the error, if necessary, and reprocess.

IEW0314 ERROR - MAXIMUM NUMBER OF REGION (four) EXCEEDED.

Explanation: There are five or mor regions specified in this overla structure.

User Response: Reduce the number o

of the data control block for the data set.

System Action: Processing was terminated. The data definition name in the name field of the DD statement for the input data set was printed after the message code.

User Response: For a format of FB, check the maximum blocking factor for input to the linkage editor programs in Appendix C.1. If a larger linkage editor program can handle the block size, use the larger program, if possible. Alternatively, recreate the data set, using a smaller block size; then execute the linkage editor again.

For a format of FBS, specify the correct block size.

If the proper block size was specified, have the computing system checked.

IEW0302 ERROR - INVALID STATEMENT -- SCAN TERMINATED.

Explanation: There is an error on a linkage editor control statement.

System Action: The statement is accepted as input up to the point of the error.

User Response: Correct the error, if necessary, and reprocess.

IEW0314 ERROR - MAXIMUM NUMBER OF REGIONS (four) EXCEEDED.

Explanation: There are five or more regions specified in this overlay structure.

User Response: Reduce the number of

[illegible]

regions in the overlay structure to four.

IEW0324 ERROR - MAXIMUM NUMBER OF SEGMENTS EXCEEDED.

User Response: Reduce the number of segments in the overlay structure to a number that is compatible with the system configuration. For details, refer to the publication IBM System/360 Operating System: Storage Estimates, Form C28-6551.

IEW0332 ERROR - MAXIMUM NUMBER OF ALIASES (five) EXCEEDED, EXCESS IGNORED.

User Response: Reprocess the module under a different name with additional aliases.

IEW0342 ERROR - LIBRARY SPECIFIED DOES NOT CONTAIN MODULE.

Explanation: Either, (1) the wrong library or module was specified by an INCLUDE statement or a SYSLIB DD statement, or (2) the automatic library call mechanism cannot find a library member of the same name as an unmarked external reference; i.e., an external reference that is not marked by the restricted no-call function or the never-call function.

User Response: Specify, (1) the proper library or module, (2) the restricted no-call or never-call function for the applicable symbol, (3) the NCAL option for the linkage editor job step, or (4) delete the external reference.

IEW0354 ERROR - TABLE OVERFLOW -- TOO MANY CALLS BETWEEN CONTROL SECTIONS.

Explanation: There are too many V-type address constants referring to external symbols in a program that is being structured in overlay. The table recording these V-type address constants has overflowed.

User Response: Either, (1) assemble the coding of two or more control sections into one control section, or (2) remove any unnecessary V-type address constants that refer to external symbols. If this is not sufficient, process in a larger main storage environment.

IEW0364 ERROR - TABLE OVERFLOW -- INPUT TEXT EXCEEDED MAXIMUM OR TOO MANY CHANGES OF ORIGIN IN INPUT.

User Response: (1) Avoid unnecessary re-origins, (2) combine small control sections, or (3) in the case of too much text, process in a larger main storage environment.

IEW0374 ERROR - TABLE OVERFLOW -- INPUT CONTAINS TOO MANY RELOCATABLE ADDRESS CONSTANTS OR TOO MANY CONTROL SECTIONS CONTAINING SUCH CONSTANTS.

Explanation: (1) There are too many control sections with relocation dictionaries, or (2) there are too many relocatable address constants.

User Response: Either, (1) assemble the coding of two or more control sections into one control section, or (2) remove any unnecessary relocatable address constants. If this is not sufficient, process in a larger main storage environment.

IEW0382 ERROR - TEXT RECORD ID IS INVALID, CARD IGNORED.

Explanation: The ID of the text record refers to an invalid external symbol dictionary entry; i.e., it does not refer to a section definition entry or a private code entry. The input deck may be out of order or incomplete.

IEW0394 ERROR - MEMBER NOT STORED IN LIBRARY -- PERMANENT DEVICE ERROR.

Explanation: This is either a hardware error or no space is allocated for the library directory.

IEW0404 ERROR - MEMBER NOT STORED IN LIBRARY -- NO SPACE LEFT IN DIRECTORY.

System Action: The member is not stored in the specified library.

User Response: (1) Reprocess, placing the output module in a new library. When the original library is used as input, concatenate the new one with it, or (2) use a utility program to copy the library, allowing for more directory entries. Edit the member into the new library.

IEW0412 ERROR - ALIAS NOT STORED IN LIBRARY -- NO SPACE LEFT IN DIRECTORY.

System Action: The ALIAS is not stored in the specified library; however, the member can be referred to by the member name.

User Response: (1) Reprocess, placing the output module in a new library. When the original library is used as input, concatenate the new one with it, or (2) use a utility program to copy the entire library (except the member whose alias was not stored), and allow for more directory entries. Edit the member into the new library.

IEW0421 WARNING - IDENTICAL NAME IN DIRECTORY, WILL TRY TO STORE UNDER 'TEMPNAME'.

Explanation: The output module name has been used previously in the library. The replace function is not specified.

User Response: Either, (1) reprocess, using a different name in the SYSLMOD DD statement or NAME statement, or (2) reprocess, and specify the replacement function for the name originally specified in the SYSLMOD DD statement or the NAME statement.

IEW0432 ERROR - LIBRARY NAME PRINTED CANNOT BE OPENED, DD CARD MAY BE MISSING.

Explanation: The DD statement that defines the library is probably missing. This message also results when a sequential data set (encountered in the processing of an INCLUDE statement) cannot be opened.

System Action: Processing continues without input from the specified library.

IEW0444 ERROR - TABLE OVERFLOW -- TOO MANY DOWNWARD CALLS.

Explanation: There are too many V-type address constants that refer to segments lower in the tree structure.

User Response: Either, (1) use an overlay structure with fewer segments, or (2) remove unnecessary references.

IEW0454 ERROR - TABLE OVERFLOW -- SEGMENT CONTAINS TOO MANY DOWNWARD CALLS.

Explanation: One segment in the overlay structure contains too many V-type address constants that refer to segments lower in the tree structure.

User Response: (1) Incorporate some of the called control sections in the requesting segment, (2) divide the requesting segment into two or more segments, or (3) remove all unnecessary references from the requesting segment.

IEW0461 WARNING - SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE, NCAL WAS SPECIFIED.

Explanation: The NCAL option, restricted no-call, or never-call function was specified for the external reference.

System Action: The automatic library call mechanism does not attempt to resolve the external reference.

IEW0472 ERROR - INVALID ALIAS ENTRY POINT IN OVERLAY STRUCTURE.

Explanation: The specified alias entry point is not in the root segment.

System Action: The entry point for the member name is used.

User Response: Respecify the alias, entry point, or overlay structure.

IEW0484 ERROR - TABLE OVERFLOW -- TOO MANY EXTERNAL SYMBOLS AFFECTED BY REPLACEMENT.

Explanation: There are too many deletions or replacements.

User Response: Replace or delete limited numbers of control sections in successive edits.

IEW0492 ERROR - NAME CARD FOUND IN LIBRARY, CARD IGNORED.

Explanation: A NAME card can be placed only in the primary input.

User Response: Remove the NAME statement from the library or sequential data set. Reprocess if the load module is incorrect.

IEW0502 ERROR - ALIAS NOT STORED IN LIBRARY -- PERMANENT DEVICE ERROR.

Explanation: The alias could not be stored in the library directory because of a hardware error.

System Action: The load module has already been stored.

User Response: Execution of the module is possible using the member name or aliases already stored.

IEW0512 ERROR - INCLUDE STATEMENT SYNTAX CONFLICTS WITH RECORD FORMAT OF SPECIFIED DATA SET -- DDNAME PRINTED.

Explanation: The INCLUDE statement syntax conflicts with the characteristics of the data set specified on the DD statement.

System Action: The specified module is ignored.

User Response: Use proper syntax on the INCLUDE statement.

IEW0522 ERROR - SPECIFIED DATA SET HAS UNACCEPTABLE RECORD FORMAT -- DDNAME PRINTED.

Explanation: The record format of the specified data set is not type V or F and cannot be processed by the linkage editor.

System Action: The data set is not processed.

IEW0532 ERROR - BLOCKSIZE OF LIBRARY DATA SET EXCEEDED MAXIMUM -- DDNAME PRINTED.

Explanation: The blocksize of the specified library data set cannot be handled by the linkage editor.

IEW0543 ERROR - IDENTICAL NAME IN DIRECTORY

Explanation: The member name or alias already exists in the directory. In the case of a member, an attempt was made to store under TEMPNAME; however, TEMPNAME was also found in the directory.

System Action: The output module is not stored under this member name or alias name.

IEW0551 WARNING - INVALID OPTION ENCOUNTERED, OPTION PRINTED -- OPTION IGNORED.

Explanation: The option specified could not be recognized by the linkage editor.

System Action: The option was ignored and processing was continued. The option was printed after the message code.

User Response: If the linkage editor's action was not satisfactory, correct the option in either the PARM parameter of the EXEC statement or the PARAM operand of the LINK or ATTACH macro-instruction that invoked the linkage editor. Then execute the linkage editor step again.

IEW0564 ERROR - UNRECOGNIZABLE DEVICE CODE FOR SPECIFIED DATA SET.

Explanation: The UNIT parameter of a DD statement specified a device that is not acceptable to the linkage editor. The name field of the DD statement contained SYSUT1 or SYSLMOD.

System Action: Processing was terminated.

User Response: Change the UNIT parameter to indicate a device type acceptable to the linkage editor for the SYSUT1 or SYSLMOD data set, and execute the linkage editor step again. Acceptable device types are indicated in Appendix C.1.

IEW0574 ERROR - INPUT DATA SET BLOCKSIZE NOT SPECIFIED, DDNAME PRINTED.

Explanation: The DCBBLKSI field in the data control block for the primary input data set (SYSLIN) specified a block size of zero. The DCBRECFM field indicated a format of FS or FBS.

System Action: Processing was terminated. The data definition name in the name field of the DD statement for the primary input data set was printed after the message code.

User Response: Provide the correct block size, and execute the linkage editor step again. Maximum blocking factors for input are given in Appendix C.1.

IEW0584 ERROR - INPUT DATA SET CONTAINS INVALID FORMAT CODE, DDNAME PRINTED.

Explanation: The DCBRECFM field in the data control block for the SYSLIN data set specified a record format other than fixed (F, FB, FS, or FBS).

System Action: Processing was terminated. The data definition name in the name field of the DD statement for the input data set was printed after the message code.

User Response: Correct the DCBRECFM field, and execute the linkage editor step again.

IEW0601 WARNING - INCOMPATIBLE OPTION ENCOUNTERED -- OPTION IGNORED.

Explanation: In the PARM parameter of the EXEC statement or PARAM operand of the LINK or ATTACH macro-instruction that invoked the linkage editor, two of the options conflicted.

System Action: The second of the conflicting options was ignored and was printed after the message code.

User Response: If the linkage editor's action is not satisfactory, eliminate one of the conflicting options and execute the linkage editor step again.

IEW0614 ERROR - MORE THAN ONE NO LENGTH CSECT WITH TEXT ENCOUNTERED.

Explanation: An object module contained more than one type PC or SD control section that had a length field containing zero in its external symbol dictionary (ESD) entry. However, text (TXT) items associated with the control section were present in the module. Only one type PC or SD control section with an ESD length field of zero can appear in an object module.

System Action: The module was not processed, and the linkage editor terminated processing.

User Response: Recompile the module, making sure the object module contains the compiler-produced END statement. Then execute the linkage editor step again.

IEW0622 ERROR - INPUT DATA SET BLOCKSIZE NOT SPECIFIED, DDNAME PRINTED.

Explanation: The DCBBLKSI field in the data control block for an input data set, other than the primary input data set (SYSLIN), specified a block size of zero. The DCBRECFM field indicated a format of FS or FBS.

System Action: The data set was not processed. The data definition name in the name field of the DD statement for the input data set was printed after the message code.

User Response: Provide the correct block size, and execute the linkage editor to process the input data set. Maximum blocking factors for input are given in Appendix C.1.

address constant: An expression representing a quantity that can be used as a storage address or in the calculation of a storage address. In the Assembler Language, an address constant can be V-type (used for branching) or A-type (used for branching within a module or for retrieving data).

common segment: A segment upon which two exclusive segments are dependent.

control program: A collective or general term referring to all control routines of the operating system.

control section: The smallest separately relocatable unit of a program: that group of coding specified by the programmer to be an entity, all elements of which are to be loaded into contiguous main storage addresses for execution.

control statement: A statement in the external language of a routine that communicates directly with the routine for the purpose of controlling its processing.

cataloged procedure: A series of job control statements that define a series of job steps; it is stored under a unique name, which can be referred to by the programmer.

data definition statement: A job control statement that is used in a job step definition to establish a data set; it specifies the type of device on which the data set resides (or will reside, if the data set is to result from execution of the job step), and provides a name by which the control program can refer to the data set.

data set: A named collection of data and program instructions contained on an external storage device.

entry name: A name within a control section; that which is defined in the Assembler Language by an ENTRY statement.

exclusive reference: A reference between exclusive segments of an overlay program.

exclusive segments: Segments in the same region of an overlay program neither of which is in the path of the other. They cannot be in main storage simultaneously.

external name: A name that can be referred to by any control section or separately assembled module; that which is defined in the Assembler Language by a CSECT, START, or ENTRY statement.

external reference: An external symbol that is defined in another module; that which is defined in the Assembler Language by an EXTRN statement or by a V-type address constant.

external symbol: A control section name, entry name, or external reference; a symbol contained in the external symbol dictionary.

inclusive segments: Overlay segments that can be in main storage simultaneously.

```
      |
     1|
      |
      |
      |            Inclusive Segments:
 ┌----------┐                1 and 2
 |          |             1, 3, and 4
 |          |             1, 3, and 5
 |          |
2|         3|      Exclusive Segments:
 |          |                2 and 3
     ┌----------┐            2 and 4
     |          |            2 and 5
     |          |            4 and 5
     |          |
    4|         5|
     |          |
```

job: One or more job steps; a process that requires the execution of a program or a series of programs; a series of related processes or tasks.

library: A partitioned data set (in this publication). Such a data set always contains named members.

load module: An executable module produced by the linkage editor; a module in a format suitable for loading into main storage by the control program for execution.

module: One or more control sections processed in one execution by a language translator or the linkage editor.

multiple load module processing: A method of processing whereby two or more load modules can be produced in a single linkage editor job step.

object module: A relocatable module produced by a language translator.

overlay tree: A graphic representation showing the relationships of segments of an overlay program and how the segments are arranged to use the same main storage area at different times.

path: A series of segments in an overlay tree that form the shortest distance in a region between a given segment and the root segment.

program: A procedure, plan, method, or process; a plan of future procedure; a logically self-contained sequence of operations or instructions that, when followed in some predetermined order, will produce a specified result; a sequence of instructions to be performed by an electronic computer; one or more modules, in source language or relocatable object code, or one module in executable code, that are a logically self-contained process.

reenterable: A reenterable module may be used by more than one task at the same time; i.e., a task may begin executing a reenterable module before a previous task has finished executing it. A reenterable module is not modified during execution.

region: A contiguous area of main storage within which segments can be loaded independently of paths in other regions. Only one path within a region can be in main storage at one time.

relocation: The modification of address constants required to compensate for a change of origin of a module, program or control section.

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.

segment: The smallest functional unit (one or more control sections) that can be loaded as one logical entity during execution of an overlay program.

serially reusable: A serially reusable module may be executed by only one task at a time. All instructions and data altered during one execution are restored before execution by another task.

symbol: Any collection of up to eight alphameric characters that begins with an alphabetic character.

C28-6538-3

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENTS

Title: IBM System/360 Operating System
Linkage Editor

Form C28-6538-3

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?
___ As an introduction to the subject ___ For additional knowledge
Other ______________________________

fold

Please check the items that describe your position:

___ Customer personnel	___ Operator	___ Sales Representative
___ IBM personnel	___ Programmer	___ Systems Engineer
___ Manager	___ Customer Engineer	___ Trainee
___ Systems Analyst	___ Instructor	Other______________

Please check specific criticism(s), give page number(s), and explain below:
___ Clarification on page(s)
___ Addition on page(s)
___ Deletion on page(s)
___ Error on page(s)

Explanation:

fold

Name ______________________________

Address ______________________________

FOLD ON TWO LINES, STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

orm C28-6538-3

staple sta

fold f

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]