64SP-3150E-1

FACOM OS IV/F4

LINKAGE EDITOR AND LOADER USER'S GUIDE

-7 DEC 1978

Manual Correction List

Manual Name	FACOM OS IV/F4 LINKAGE EDITOR/LOADER USER'S GUIDE
Code	64SP-3150E-1

Fujitsu Limited
Communications & Electronics
Tokyo, Japan

Abbreviations in the list

U : updating forms 1 to 6

1. Test replacement	4. Page insertion
2. Page replacement	5. Text deletion
3. Text insertion	6. Others ( )

R : reasons for update A to C

A	Correction (Misprinting & Explanation)
B	Alteration of Specifications (Adding,alterating & deleting functions)
C	Others ( )

No.	Page	Line		Old Contents	New Contents	U	R	Remarks
		L	R					
1	iii	30			APPENDIX 4 SIZE OF VIRTUAL STORAGE AREA FOR LINKAGE EDITOR AND THE QUANTITY THAT CAN BE PROCESSED APPENDIX 5 SIZE OF THE VIRTUAL STORAGE AREA FOR LOADER	3		
2	23	44			Common areas in reenterable programs cannot be shared between tasks. They are treated as prototype sections. (See Section 7.4)	3		
3	26	20			• Dynamic linkage table , Common sections are prefixed by a number symbol (#), blank common sections by #$.	3		
4	60		31		a. LINECOUNT = $\left\{\frac{60}{n}\right\}$ option This specifies the number of lines per page output to the SYSPRINT data set. Its omission is interpreted as a specification for 60 lines ($10 \leq n \leq 99$).	3		
5	83	11			LINECOUNT = $\left\{\frac{60}{n}\right\}$ This specifies the number of lines per page output to the SYSOUT data set. It is ignored, however, when outputting to a terminal. ($10 \leq n \leq 99$)	3		

No.	Page	Line		Old Contents	New Contents	U	R	Remarks
		L	R					
6	97		35	JQA0581-E ERROR: COMMON AND CONTROL SECTIONS HAVE IDENTICAL NAME PRINTED — RENT OPTION CANCELLED. A control section and a common section having the same name were entered while the RENT option was specified. S: The RENT attribute is ignored. P: When giving an initial value to a common section in creating a load module with the RENT attribute, use a prototype section.		5		
7	100	6		JQA0981-U ERROR: TOO LARGE SYSPRINT BLOCKSIZE — LINKAGE EDITOR PROCESSING TERMINATED. The block size specified for the SYSPRINT data set cannot be handled by the Linkage Editor.		5		

No.	Page	Line		Old Contents	New Contents	U	R	Remarks
		L	R					
8	100	29		S: The data set is not opened. Linkage Editor processing terminates. P: Probable user error. Either decrease the block size of the data set, or increase value 2 of the SIZE option to allow for larger buffers, and increase value accordingly, if necessary. Increase the region size, if necessary. Rerun the Linkage Editor step.	JQB100I 1 EXTERNAL REFERENCE DYNAMICALLY RESOLVED. (Explanation) Unresolved external reference by the V type address constant results in dynamic linking because DYNAMIC option has been specified. (Action) Processing continued. (Measure) Unnecessary.	3		

No.	Page	Line L	Line R	Old Contents	New Contents	U	R	Remarks
9	102		40		JQB124I L MISTAKE: INVALID TEXT LENGTH-RELOCATION IMPOSSIBLE. (Explanation) Object module including two sections without length has been input but the END card does not specify length. (Action) Processing suspended. (Measure) Check if input module is valid or not. JQB125I U MISTAKE: TOO MANY DALTAB ENTRIES. (Explanation) Too many unresolved external references requiring dynamic linking. (Action) Processing suspended. (Measure) Number of unresolved external references reduced to 255 or less by incorporating the necessary module. JQB126I U MISTAKE: TOO MANY ALIASES. (Explanation) Too many additional entry points. (Action) Processing suspended. (Measure) Number of entry points is to be reduced to 255 or less by reducing the number of input modules through dynamic linking.	3		

No.	Page	Line		Old Contents	New Contents	U	R	Remarks
		L	R					
					Otherwise, NOALIAS option may be specified if additional entry points are not required.			
10	105				See attached page 1 through 6	4		

APPENDIX 4 Size of virtual storage area for Linkage Editor and the quantity that can be processed.

The virtual storage area necessary for the execution of the Linkage Editor is decided by various factors like the number and the characteristics of specified options and input modules. The section here discusses various figures needed for the estimation of the area size.

Supplementary Table 1.1 lists the main items when the Linkage Editor uses the virtual storage (excluding the input output buffer) and the quantity that can be processed in each case under the conditions as given below. The Table also shows the increment when v_1 of SIZE option and REGION size of EXEC statement are increased by 10 KB.

(Conditions)
- REGION size of EXEC statement: 128 KB or more
- SIZE = (118K, 36K)
- Presence of specification for DYNAMIC, OVLY, and AM256 options. (Unless these options are specified, the quantity that can be processed increases).
- SYSPRINT block length is 605 bytes.

The principal items listed are explained below.

(1) Number of external symbol items

Included here are the following items apart from the input external symbols.

- dd name specified by the LIBRARY or INCLUDE statements.
- Symbols specified by ALIAS, REPLACE, CHANGE statements.
- SEGTAB, ENTAB, and DALTAB.

TABLE 1.1 Processable quantities in Linkage Editor

Item	Processable quantity	Increments of processable amount when v1 is increased by 10 KB
Number of external symbol items	789	209
Number of intermediate text records	548	140
Number of intermediate RLD records	292	75
Number of V-type address constant	682	186
IDR translation data table 1	1406 (Bytes)	524 (Bytes)
IDR translation data table 2	512 (Bytes)	140 (Bytes)
IDR user data table	3306 (Bytes)	1024 (Bytes)
Number of IDRZAP data items	125	39
Number of ORDER and PAGE statement operands	381	109
Number of aliases	256 (64 in the absence of AM 256 option specification)	
Number of segments	255	
Number of regions	4	
Number of ENTAB items	340 *	
Number of DALTAB items	255 *	

* ENTAB and DALTAB sizes must not exceed the maximum record length of the text during load module output.

(2) Number of intermediate text records

Number n of intermediate text records per section is given by:

$$n = 1 + \frac{\text{Section length}}{\text{Maximum text record length in load module output}}$$

The number of the intermediate text records per section increases unless the texts are input in the address sequence.

(3) Number of intermediate RLD records

Number n of the RLD records per section is given by

$$n = 1 + \frac{\text{Relocatable address constants within the section}}{30}$$

(4) Number of V-type address constants

The limitation here applies only in the case of DYNAMIC or OVLY specification. The processable quantity of V-type address constants indicated in Table 1.1 is the value in the worst situation. Normally the value is slightly higher.

(5) IDR translation data table 1

Size M of table 1 for IDR translation data is given by

$$M = \sum_{j=1}^{p} (2 \times S_j + 6 \times \sigma_j) \text{ bytes}$$

where j: input module number ($j = 1 \sim p$)

Sj: Number of sections in the jth module ($j = 1 \sim p$)

$$\sigma j: \begin{cases} 1 \text{ ..where a module with the same IDR has been input} \\ 0 \text{ ..where a module with the same IDR has not been input} \end{cases}$$

(Module with the same IDR means the module created by the same language translator on the same day).

(6) IDR translator data table 2

Size N of table 2 required for IDR translation data

$$N = \sum_{j=1}^{p} A_j \times \delta_j \quad \text{bytes}$$

where j: input module number $(j = 1 \sim p)$

A_j: 16 byte IDR data legth for the jth input module (31 bytes if processed by preprocessor to support IDR)

δ_j: $\begin{cases} 0 \text{ ..where a module with the same IDR has been input} \\ 1 \text{ ..where a module with the same IDR has not been input} \end{cases}$

(7) IDR user data table

This is the table for storing data specified by the IDENTIFY statement. Necessary size of the table:

$$N = \sum_{j=1}^{q} (\ell_j + 6) \quad \text{bytes}$$

where, j: specified data number $(j = 1 \sim q)$

ℓ_j: number of data bytes

(8) Number of IDR ZAP data items

This is the number of IDR data items set through the JQPSPZAP program.

APPENDIX 5 Size of the virtual storage area for loader

Supplementary Table 1.2 shows the items using virtual storage during execution of the loader and their estimated sizes.

Supplementary Table 1.2 Size of virtual storage area necessary for loader

* : These areas are released before execution of the programs

Broad classification	Intermediate classification	**Fine classification**		Size	**Remarks**
Region (size specified by the REGION parameter)	Fixed part (32 K Bytes)	Loader program	Control section	4 K bytes	
			Processing section*	16 K bytes	
		Control program		4 K bytes	
		Others for data management		8 K bytes	
	Variable part (Size specified by SIZE option. However, where region size is not sufficient, the region size is 32KB).	Loaded program		Size of loaded program	
		DALTAB		16 x n bytes	n : Number of unresolved external references (referenced by the V type address constant) Maximum value is 255. Table is created only during DYNAMIC option specification.
		Symbol table		12 x n bytes	n : (number of sections + number of entry points) Created: when ALIAS option is specified or when virtual storage area is sufficient in TSS.

Broad classification	Intermediate classification	Fine classification		Remarks
Region (size specified by the REGION parameter)	Work table area *	$2000 + 8 \times m + 20 \times n$ bytes		m : Number of RLD entries n : Number of ESD entries
	DCB, buffer, DECB areas *	$BUFNO_1 \times (BLKSIZE_1 + 24)$ + $BUFNO_2 \times (BLKSIZE_2 + 24)$ + 620 bytes		$BUFNO_1$, $BLKSIZE_1$: Number of input object module buffers and block size $BUFNO_2$, $BLKSIZE_2$: SYSLOUT buffers and block size: minimum number of buffers in either case is 2)
	Temporary object module storage area *	Object module size		Necessary only when object module input is translated for RENT option in FORTRAN or COBOL. This memory area is released for others after the concerned input module has been processed.

PREFACE

This manual describes in detail the functions and facilities of the OS IV/F4 Linkage Editor and Loader, which are two service programs required by all users of OS IV/F4. The manual consists of ten chapters divided into three major sections.

Chapters 1 to 9 describe the Linkage Editor, explaining its purpose, standard and optional inputs, standard and optional outputs, various editing and program-structuring facilities, relevant JOB Control Language facilities, and the Linkage Editor control statements.

Chapter 10 describes the Loader, stressing facilities different from those of the Linkage Editor, its standard and optional inputs and outputs, etc. You should fully understand the basic material of (Chapters 1 — 9) before attempting to read this section.

The last section comprises four appendices which will be useful to most programmers for regular reference and trouble-shooting explanations. Appendix 1 shows the format of the standard OS IV/F4 **load module** (executable program). Appendix 2 describes, in alphabetical order, all warning/informational/diagnostic messages issued by the Linkage Editor and Loader. Appendix 3 shows a sample input stream for a Loader job.

To fully understand the contents of this manual, you should be familiar with the following OS IV/F4 manuals:

- **FACOM OS IV/F4 Job Control Language Reference Manual**
- **FACOM OS IV/F4 Job Management Functions and Facilities**
- **FACOM OS IV/F4 Data Management Functions and Facilities**
- **FACOM OS IV/F4 Supervisor Functions and Facilities**
- **FACOM OS IV/F4 Service Aid User's Guide**

First Edition June, 1977

CONTENTS

ILLUSTRATIONS AND TABLES

CHAPTER 1 INTRODUCTION

The Linkage Editor and the Loader are Fujitsu-supplied programs which prepare object modules created by compilers and/or the Assembler for execution. The Linkage Editor prepares **load modules,** which you can execute later. The Loader prepares an executable program in your address space and passes control to it directly.

The Linkage Editor provides several editing options such as creating overlay programs. The Loader rapidly loads programs that do not need sophisticated editing or to be saved for subsequent production usage.

You should use the Linkage Editor rather than the Loader:

- if your program requires editing other than the MAP, LET, NCAL, and SIZE options.
- if your program uses control statements such as INCLUDE, NAME, OVERLAY, etc.
- if you wish to create a temporary or permanent executable program on direct access storage.

1.1 TYPES OF PROGRAM STRUCTURES

You should use the Loader if your program only requires some (or none) of the following editing and output options: MAP, LET, NCAL, and SIZE. The Loader halves editing and loading time compared to the Linkage Editor.

Linkage Editor processing is performed in a **link edit** step. The Linkage Editor can be used for compile-link-go, compile-link, link-edit, and link-go jobs. Loader processing is performed in a **load** step, which is equivalent to link-go steps. You also use the Loader for compile-load and load-go jobs.

Each load module is designed in one of four basic structures (or certain combinations thereof): simple, overlay, dynamic program, or dynamic link. A **simple structure** does not pass control to any other load modules during its execution; it is brought into virtual storage all at one time. An **overlay structure** may, if necessary, pass control to other load modules during its execution; it is not brought into virtual storage all at one time. Instead, segments of the load module reuse the same area of virtual storage. A **dynamic program structure** is brought into virtual storage at one time and passes control to other load modules during its execution. A **dynamic link structure** combines aspects of the overlay and dynamic program structures. Each of the load modules to which control is passed can be one of the four structure types. Characteristics of load-module structure types are summarized in Table 1.1.

Table 1.1 Characteristics of load modules

Structure type	Loaded all at one time	Passes control to other load modules
Simple	Yes	No
Overlay	No	Rarely
Dynamic Program	Yes	Yes
Dynamic Link	No	Yes

Simple structure

A simple structure comprises a single load module produced by the Linkage Editor, containing all required instructions. It is paged into real storage by OS IV/F4 as it is executed. The simple structure is the most efficient of the four structure types because the instructions (and macro instructions) it uses to pass control — CALL, Branch, etc. — do not require control program assistance.

Overlay structure

An overlay structure comprises a single load module divided into logically self-contained units called **segments**. The load module is retrieved into virtual storage one segment at a time starting with the **root segment,** which is loaded by the Supervisor. Segments are defined when the program is linkedited. CALL/SAVE/RETURN linkages within an overlay structure are the same as within a simple structure. However, Supervisor assistance is sometimes required for calls to subroutines, entry points, etc. in different segments. Overlay structures are discussed in detail in Chapter 6 below.

Dynamic program structure
A dynamic program structure requires more than one load module during execution. Each load module can be either a simple, overlay, or another dynamic structure. The advantages of a dynamic structure over a simple structure increase as the program becomes more complex, particularly when the logical path of the program depends on which records are being processed. The load modules required in a dynamic structure are paged into real storage when required; they can be deleted from virtual storage when their use is completed. Dynamic program structures are described in Section 1.2 below and in **FACOM OS IV/F4 Supervisor Functions and Facilities.**

Dynamic link structure
This structure contains attributes of both the overlay and the dynamic program structures. The dynamic link structure uses CALL macro instructions to invoke sub-programs, just like simple and overlay structures. However, if the requested entry point is in a different load module, OS IV/F4 will retrieve it and pass control to it with a CALL-type linkage. Hence, if the entry point is in the same load module, or if its load module has already been retrieved by the Supervisor, control-program linkage assistance is unnecessary. Dynamic link structures are described in Chapter 7 and **FACOM OS IV/F4 Supervisor Functions and Facilities.**

1.2 COMPARISON OF PROGRAM STRUCTURES

In this section, characteristics of the four types of program structures will be explained by examples, all based on a program comprising one main routine and five sub-programs:

Abbreviation	**Function**	**Size**
M	Main program	10K
S1	Sub-program 1	8K
S2	Sub-program 2	8K
S3	Sub-program 3	6K
S4	Sub-program 4	10K
S5	Sub-program 5	8K

Simple structure
If you design this program as a simple structure, it becomes a load module of 50K bytes, as shown in Fig. 1.1. All of its sub-programs are retrieved simultaneously. All transfers of control within the structure are direct, via Branch instructions. Therefore, this simple structure requires more virtual storage than any other type of structure but it takes less time for loading compared with other possible structures.

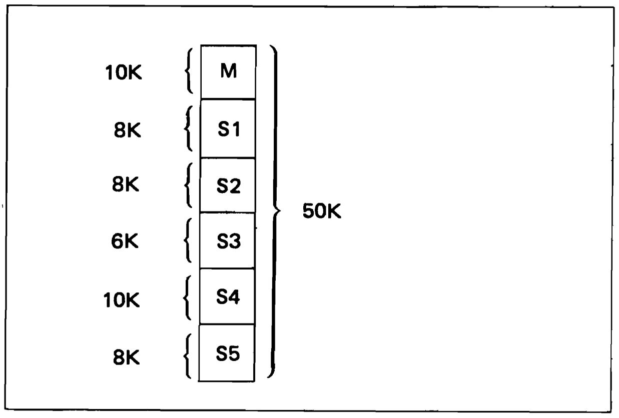


Fig. 1.1 Simple program structure (50K)

Overlay structures
Your program is now assumed not to need sub-programs S3, S4, and S5, when sub-programs S1 and S2 are active; similarly, S1 and S2 are not required while S3, S4, or S5 is active. It is also assumed that S5 is unnecessary while S4 is active. In this case, you can save virtual storage by dividing your sub-programs into segments, as shown in Fig. 1.2.

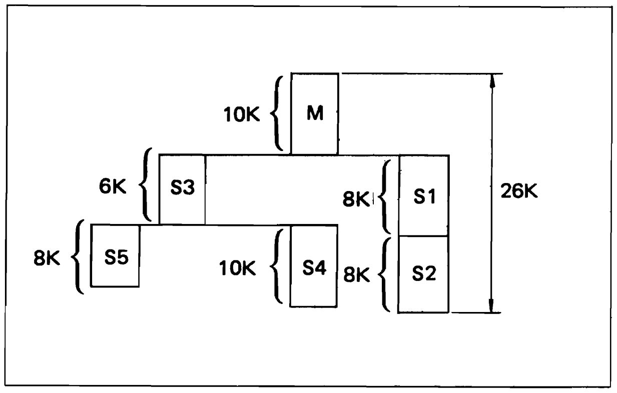


Fig. 1.2 First example of an overlay structure (26K)

You can load any segment whenever it is needed, since only certain sub-programs are necessary in virtual storage at any instant. If a segment is in virtual storage, it can receive control directly from another segment; this linkage uses a Branch instruction. Compared with the simple structure, your overlay structure can be executed in a smaller area, but takes somewhat more time for loading. In Fig. 1.2, the required memory area is 26K. If you allocate a larger memory area of 34K, loading time will be saved by the different (but logically equivalent) structure illustrated in Fig. 1.3.

Using a single load module, you can thus define overlay structures within memory sizes ranging from the minimum size of Fig. 1.2 to the maximum size of Fig. 1.1 (the simple structure). Of course, you must relink your entire program if you wish to change your main program or any sub-program.

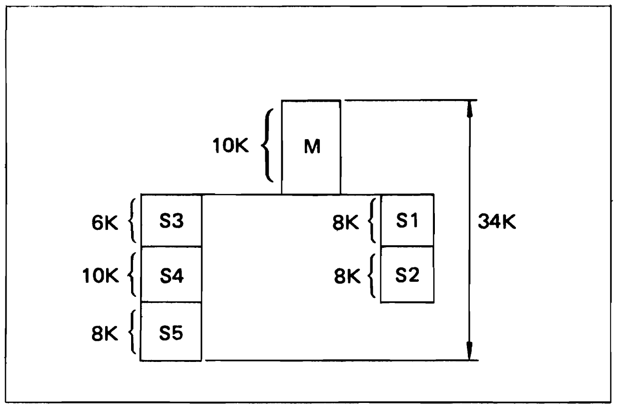


Fig. 1.3 Second example of an overlay structure (34K)

Since OS IV/F4 furnishes a private virtual storage of 16 million bytes, it imposes fewer memory limitations on your programs than any prior hardware/software system. However, if you execute a large program using a simple structure, it may take much more time to page in and page out leading to reduced CPU efficiency and long turnaround times. Loading a sub-program only when required and completing its execution without unnecessary paging improves system throughput and efficiency. Therefore, when evaluating whether or not to design overlay structures for OS IV/F4, you should give more consideration to loading and paging overheads than to program-design problems resulting from limited real memories. Paging overhead can be quite high if a mammoth program is loaded as a simple structure. Hence, you should consider average page-frame requirements for loading simple structures.

Dynamic program structure

As shown in Fig. 1.4, a dynamic program structure links some/all sub-programs into separate load modules, loading and transferring control via Supervisor macro instructions (LOAD, LINK, XCTL, etc.) as required. In this example, it is assumed that the relations between your sub-programs are identical to those in the previous subsection: M calls S1 and S3 successively which return to their caller; S1 calls S2; and S3 calls S4 and S5 in succession. The minimum virtual storage is 26K for executing this structure. If your sub-programs can be "re-used" (as described in Chapter 8), OS IV/F4 does not delete them from virtual storage. OS IV/F4 will automatically keep all sub-programs in virtual storage if they can be reused.

Thus, the dynamic program structure can automatically adapt to various environments, unlike an overlay structure which requires re-linking whenever its structure must be changed. However, a dynamic program structure leaves management of programs to the OS IV/F4 Supervisor; even if a necessary sub-program is already in memory, the Supervisor must assist in linking it to a calling program. Hence, a dynamic program structure typically imposes more

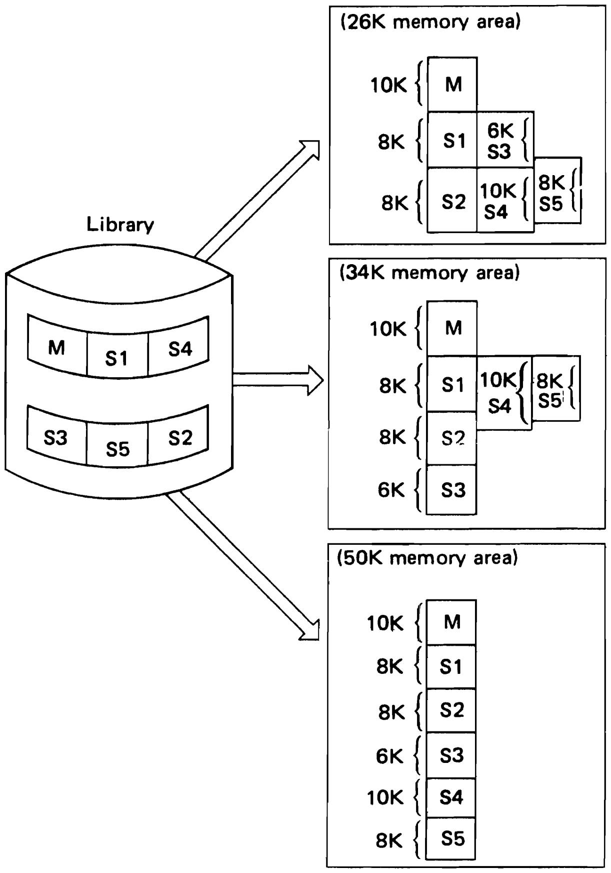


Fig. 1.4 Dynamic program structure

control-program overhead for linkage than other program structures.

Dynamic link structure

A dynamic link structure combines attributes of the other three program structures. You use it principally for testing programs which ultimately will become single, self-contained load modules.

During testing, a new program can be more conveniently handled as a collection of separate load modules—dynamic program structure—instead of being linked into a single, large load module. In other words, when an error is uncovered in a sub-program, a simple structure or an overlay structure requires linkage editing of the entire load module, including many error-free sub-programs. Linkage-editing effort and time can be reduced if each sub-program can be segregated from all others. During the test stage, often more time is required for link-editing of tasks than for tests. If you use a dynamic program structure, the Supervisor typically calls your major sub-programs repeatedly, thus imposing heavy supervisory overheads.

To resolve these strategic problems, OS IV/F4 furnishes a novel and efficient dynamic link facility, which uses CALL macro instruction (containing branch linkages) "most of the time" for program linkages. If a specified entry point is in the same load module as the caller, or if another load module with this entry point has already been loaded, the Branch instruction is **direct**, requiring no intervention by the OS IV/F4 Supervisor. If the requested sub-program is not currently in virtual storage, the dynamic link structure asks the Supervisor to load it, then branches to it immediately after loading. Thus, you need not know if the entry point of a called program is located in the same load module.

The dynamic link structure is fundamentally different from the dynamic program structure; the former is a group of subprograms, each of which has a simple structure. After initial loading, subprograms of a dynamic link structure are not deleted from virtual storage even if their processing has been completed. A dynamic link structure transfers control by a branch instruction no later than the second CALL to each entry point. Therefore, it lacks the virtual storage flexibility of the dynamic program structure, which adapts to the capacity of a given virtual storage region.

The dynamic link structure is designed primarily for program testing; its advantages are its simplicity and its reduced overhead for relinking programs while the latter are being frequently changed.

Table 1.2 summarizes the characteristics of each program structure.

Table 1.2 Characteristics of program structures

Program structure	Required memory area	Time required for program link-ages	Characteristics
simple	relatively large	minimal	● one load module ● required memory area equals total of sub-program lengths ● direct branches by CALL macro instructions
overlay	medium	medium	● one load module ● alterations require repeated link-editing of entire load module ● necessary segments can be loaded by CALL macro instructions
dynamic program	small	substantial	● more than one load module ● OS IV/F4 manages memory areas automatically ● inter-module linkages require LOAD, LINK and XCTL macro instructions ● alterations require minimal repeated link-editing
dynamic link	relatively large	medium	● more than one load module ● required memory area is nearly maximal (as for a simple structure) ● necessary load modules are loaded by CALL macro instructions ● direct branches are used thereafter ● alterations require minimal repeated link-editing

CHAPTER 2 LINKAGE EDITOR FUNCTIONS AND FACILITIES

OS IV/F4 compilers and the Assembler are collectively called **language translators.** Input to a language translator is called a **source module** (Fig. 2.1), output from a language translator an **object module.** Before an object module can be executed, it must be processed by the Linkage Editor. The primary output of the Linkage Editor is a **load module** (Fig. 2.2).

Each object module is composed of one or more control sections. A **control section** is a unit of coding (instructions and/or data) that is in itself an entity. All elements of a control section are loaded and executed with constant addressing relationships to one another. A control section is therefore the smallest separately relocatable unit of a program.

Each object module you provide to the Linkage Editor normally contains symbolic references to control sections in other modules; such references are called **external references.** These references utilize **address constants.** Each such symbol must be either the name of a control section or the name of an entry point in a control section. Control section names and entry names are called **external names.** By matching each external reference with an external name, the Linkage Editor resolves references between modules. External references and external names are collectively called **external symbols** (Fig. 2.3). An external symbol is one you define in one module and referenced in another module.

2.1 OBJECT AND LOAD MODULES

Object modules and load modules have the same basic contents:

- Control dictionaries, containing information neces-

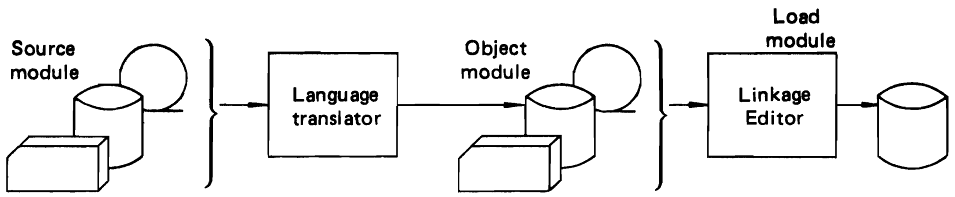


Fig. 2.1 Preparing a source module for execution

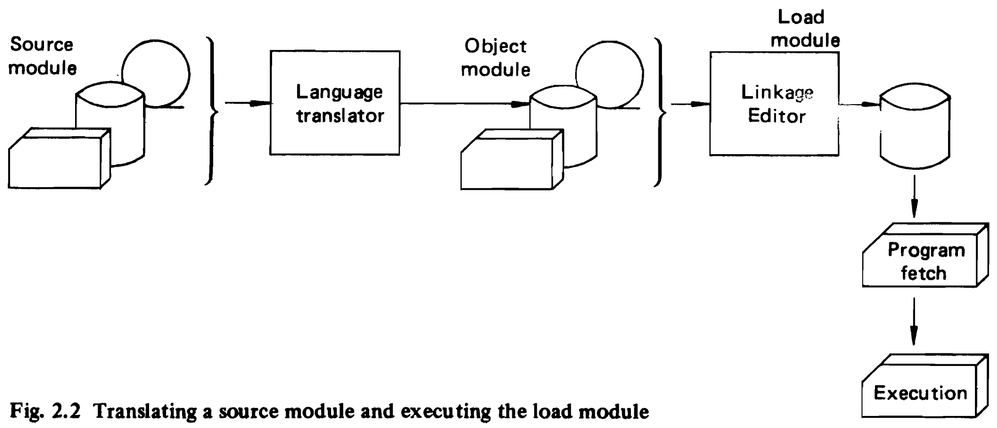


Fig. 2.2 Translating a source module and executing the load module

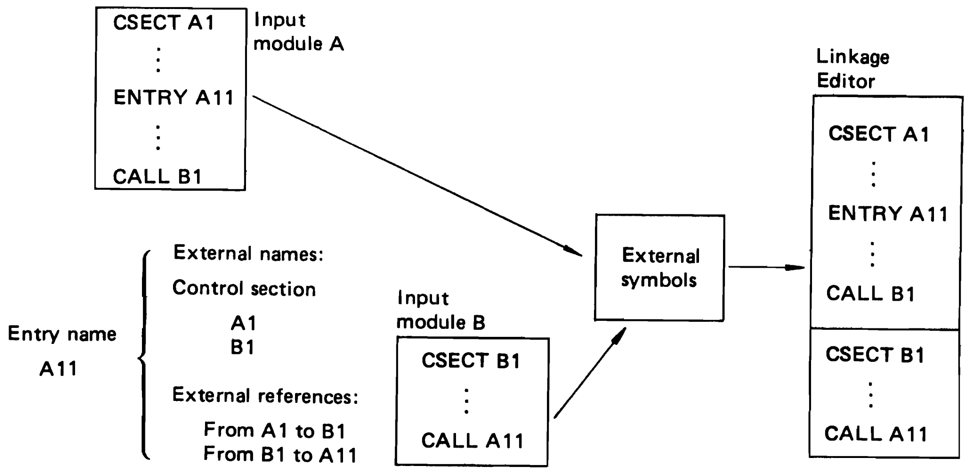


Fig. 2.3 External names and external references

sary to resolve symbolic cross-references between control sections of different modules, and to relocate address constants. Control dictionary entries are generated when external symbols, address constants, or control sections are processed by a language translator. Each language translator produces two kinds of control dictionaries: an external symbol dictionary (ESD) and a relocation dictionary (RLD).

- Text, containing the instructions and data of the program.
- An end-of-module indication: an END statement in an object module, an end-of-module indicator in a load module.

Each control dictionary, text, and end indication is described in greater detail in the following sections.

At your option, object modules and load modules retain data used by the Linkage Editor to create CSECT Identification (IDR) records. If the language translator creating an object module supports CSECT Identification, your input object module optionally furnishes translator data for Identification records on its END statement. Input load modules differ from object modules in the type of date they supply. Input load modules also provide JQPSPZAP data, Linkage Editor data, and user data to Identification records created by the Linkage Editor. During your link-edit step, you can optionally furnish an IDENTIFY control statement to supply your own data for the CSECT Identification records.

2.1.1 External Symbol Dictionary

The external symbol dictionary (ESD) contains one entry for each external symbol you define or reference within a module. The ESD contains an entry for each external reference, pseudo register (external dummy section), entry name, named or unnamed control section, and blank or named common area. You can reference an entry name, pseudo register, or named control section from any control section or separately processed module, but not an unnamed control section.

Each entry identifies a symbol or symbol reference and gives its location, if known, within the module. Each ESD entry is classified as one of the following:

- **External reference**
  a symbol defined elsewhere which you reference in the current module.

- **Weak external reference**
  a special type of external reference that is not to be resolved by the Automatic Library Call feature unless an ordinary external reference to the same symbol is found.

- **Entry name**
  an entry point you have identified in this module. The corresponding ESD entry specifies the symbol, its location, and the control section to which it belongs.

- **Control section name**
  the symbolic name of a control section. The corresponding ESD entry specifies the symbol, the length of the control section, and its location—that is, the address of its first byte.

- **Blank or named common area**
  a control section you use to reserve a virtual storage area that can be referred to by other modules. You can use a common area, for example, as a communications region within a program or to hold data supplied at execution time. The corresponding ESD entry specifies the name and length of the area. If there is no name, the name field contains blanks.

- **Private code**
  an unnamed control section. The corresponding ESD entry specifies the length of the control section

and the origin. The name field contains blanks.

- **Pseudo register**
  a special facility (corresponding to the external dummy section feature of the OS IV/F4 Assembler and PL/I Compiler) to help write re-enterable programs. A pseudo register is a dynamically-obtained location in virtual storage which your program uses to point to dynamically acquired storage. Space for such areas is not reserved in the load module, but is acquired during execution. The corresponding ESD entry contains the name, length, alignment, and displacement of the pseudo register.

When processing input modules, the Linkage Editor resolves references between modules by matching referenced symbols to defined symbols. To do this, it searches for the external symbol definition in the ESD of each input module. As shown in Fig. 2.4 the Linkage Editor matches an external reference to BB by locating its definition in the ESD of Module B. In the same way, it matches an external reference to A1 by locating its definition in the ESD of Module A.

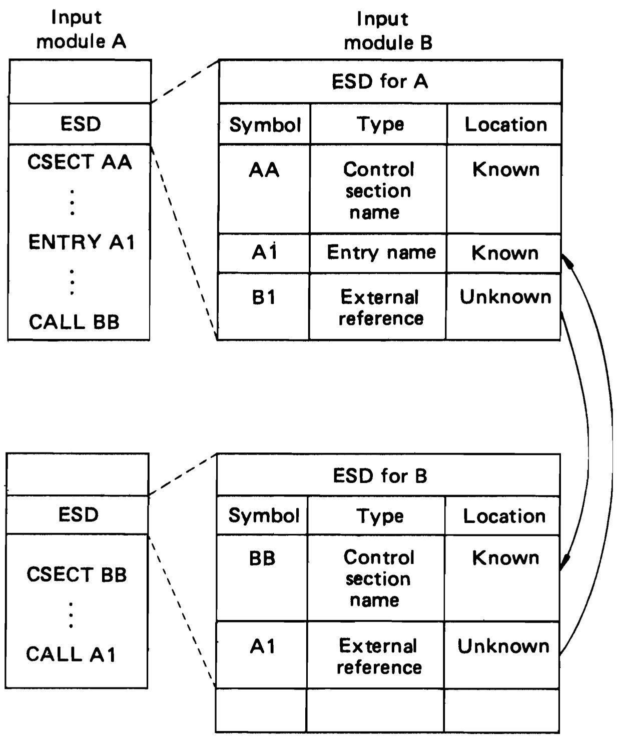


Fig. 2.4 Use of the external symbol dictionary

2.1.2 Text

The text contains the instructions and data of the module.

2.1.3 Relocation Dictionary

The relocation dictionary (RLD) contains one entry for each relocatable address constant that must be modified by the OS IV/F4 Supervisor as it loads this constant just prior to execution. An entry identifies an address constant by indicating both its location within a control section and the external symbol used to compute the value of the address constant.

The linkage Editor re-processes the relocation dictionary whenever it must adjust address constants for reference to other control sections and modules. The OS IV/F4 Supervisor again uses this dictionary to adjust these address constants whenever it loads the module into virtual storage for execution.

2.1.4 End Indication

Each load module ends with an **end-of-module indication** (EOM). Unlike an Assembler END instruction, a load-module EOM cannot specify an entry point. Therefore, whenever a load module is reprocessed by the Linkage Editor, you must specify its entry point on an ENTRY statement; if not specified, the Linkage Editor will assign the first byte of the first control section encountered as the **entry point**, i.e., first instruction to be executed after the Supervisor gives control to this processing program.

2.2 LINKAGE EDITOR PROCESSING

This section discusses input and output sources of the Linkage Editor and how it creates a load module.

2.2.1 Input and Output Sources

The Linkage Editor accepts input from several sources, as follows:

- Its primary input (SYSLIN) contains object modules and control statements for the Linkage Editor.
- Additional user-specified input, can comprise object modules, control statements, and/or load modules. You can explicitly furnish these inputs or incorporate them automatically from a call library.

During processing, the Linkage Editor generates **intermediate data** , which is written onto a direct-access storage device when virtual storage allocated for input data is exhausted.

Output from the Linkage Editor is of two types:

- A load module placed in a **library** (a partitioned data set on a direct access device) with a member name you specify.
- Diagnostic output written into a sequential data set you specify.

Fig. 2.5 shows input, intermediate, and output data sets for the Linkage Editor.

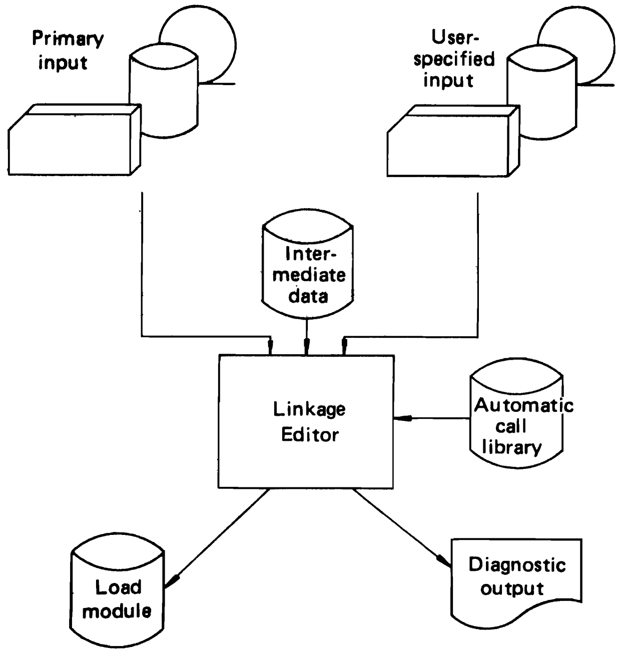


Fig. 2.5 **Input, intermediate, and output sources for the Linkage Editor**

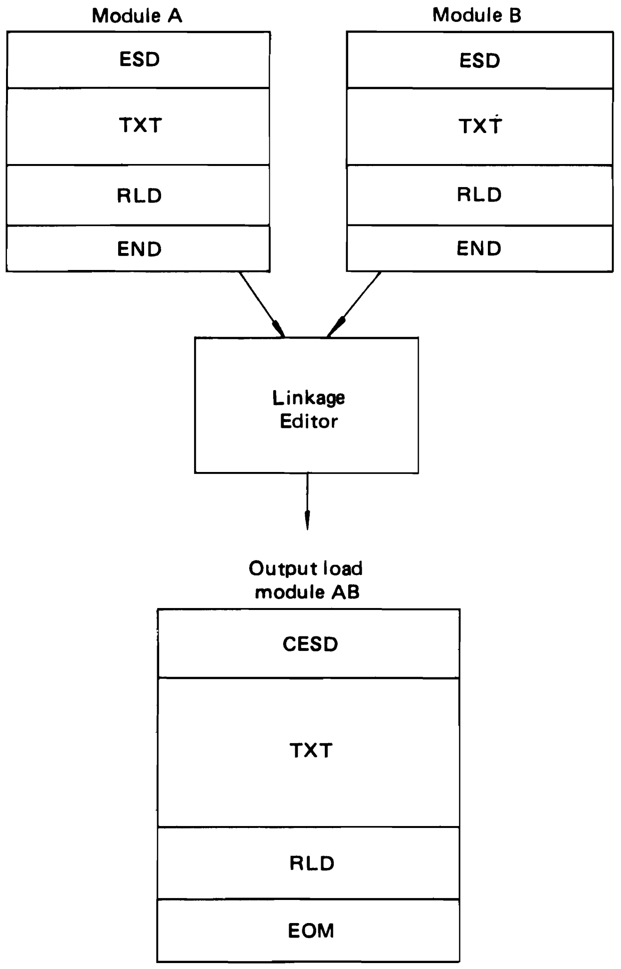


Fig. 2.6 **Load module produced by the Linkage Editor**

2.2.2 Load Module Creation

In processing object and load modules, the Linkage Editor assigns consecutive relative addresses to all control sections and resolves all references between control sections. You can even furnish object modules produced by different language translators to create a single load module, although this is relatively uncommon.

Your output load module is composed of all input object modules and input load modules you furnish to the Linkage Editor. Its control dictionaries are therefore the logical union of all control dictionaries in your input modules. The control dictionaries of a load module are called the **composite external symbol dictionary** (CESD) and the **relocation dictionary** (RLD). The load module also contains all text from each input module and one end-of-module indicator (Fig. 2.6).

Assigning addresses

Each module you furnish has an origin that was assigned during assembly, compilation, or a previous execution of the Linkage Editor. When several modules with independently assigned origins are processed by the Linkage Editor, their sequences of addresses may overlap; two input modules may even have the same origin.

Each input module contains one or more control sections. To produce an executable output load module, the Linkage Editor computes relative virtual storage addresses for each control section by assigning an origin to the first control section encountered and then assigning addresses, relative to that origin, to all other control sections of the output load module. Later, the OS IV/F4 Supervisor uses the value assigned as the origin of each control section to relocate its address-dependent items.

Although addresses in a load module are consecutive, they are relative to address 0 (zero). When you request execution of this module, the OS IV/F4 Supervisor loads it at a specific virtual storage location. The Supervisor increments addresses in the module by this base address, altering each address constant in virtual storage.

Resolving external references

The Linkage Editor also resolves external references in input modules. Your cross references between control sections in different modules were originally symbolic. They must be resolved relative to addresses assigned within the load module. The Linkage Editor calculates the new address of each relocatable expression in a control section and determines the assigned origin of the item to which it refers.

2.3 FUNCTIONS OF THE LINKAGE EDITOR

Linkage-editor input comprises object modules, load modules, and/or control statements. The primary function of the Linkage Editor is to combine these modules, in accordance with your control statements, into a single output load module. Although this linking (combining) of modules is its primary function, the Linkage Editor also:

- Edits modules by replacing, deleting, rearranging, and ordering control sections as directed by your control statements.
- Optionally aligns designated control sections and named common areas on page boundaries.
- Accepts additional input modules from data sets other than the primary input data set, either automatically or as you explicitly request.
- Reserves storage for common control sections generated by Assembler and FORTRAN language translators, likewise static external areas generated by PL/I.
- Computes total length and assigns displacements for all pseudo registers (external dummy sections).
- Creates overlay programs in structures defined by your control statements.
- Creates multiple output load modules, if so requested.
- Provides special processing and diagnostic output options.
- Assigns module attributes describing structure, content, and logical format of your output module.
- Modifies its default address-space storage allocations in response to your optional parameters for controlling Linkage Editor processing.
- Stores system status index (SSI) information into the directory of the output module library, for ease of record keeping and maintenance.
- Traces the processing history of a program.
- Allows you to lengthen a control section or named common section without changing source code, reassembling, or recompiling.
- Allows you to assign an authorization code to a load module that (a) makes it a restricted resource and (b) enables it to pass control to other restricted resources.

Each of these functions is described briefly in the following paragraphs.

Links modules
using the Linkage Editor, you can divide your program into several modules, each containing one or more control sections. Your modules can be separately assembled or compiled. The Linkage Editor combines these modules into one output load module (Fig. 2.7) with contiguous storage addresses. During its processing, the Linkage Editor resolves references between these modules and places the output module into a **library** (partitioned data set).

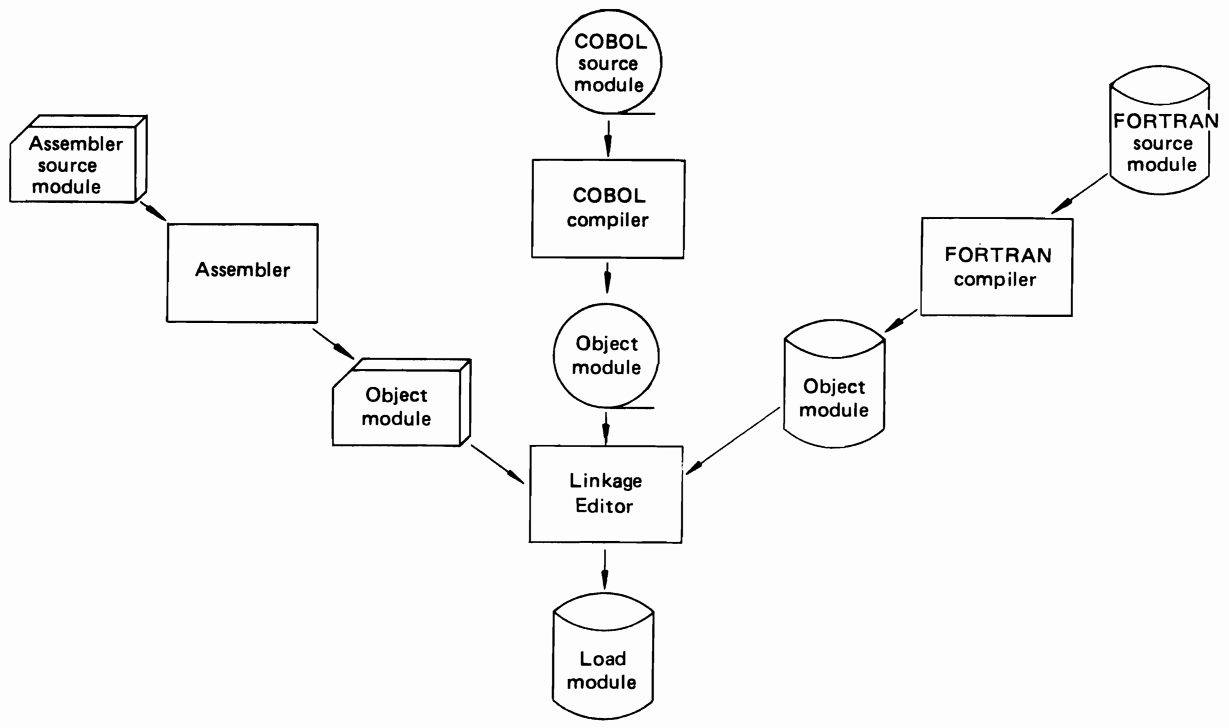


Fig. 2.7 Linkage Editor processing – module linkage

Edits modules
You can more easily modify your programs via editing functions of the Linkage Editor. If your program needs revision, you need modify, then re-compile and re-link only the affected control sections instead of the entire source module. (With the innovative OS IV/F4 **dynamic link** facility, you need not relink your entire program; you need link only those modules which you have changed.)

You can replace, rename, move or re-order control sections precisely as you wish. Control sections can also be automatically replaced by the Linkage Editor. You can change or delete external symbols by furnishing other control statements, as illustrated in Fig. 2.8

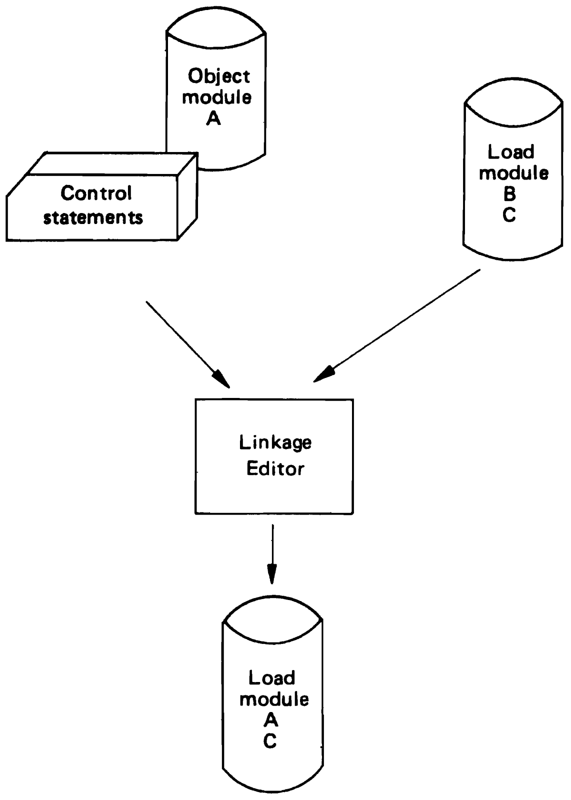


Fig. 2.8 Linkage Editor processing – module editing

Aligns control sections or common areas on page boundaries
Control sections or named common areas in the output load module can be aligned on either 2K or 4K **page boundaries** (addresses which are integral multiples of 2048 or 4096, respectively). Alignment on page boundaries enables you to use real storage more efficiently, and it can appreciably reduce the paging rates for your job.

Accepts additional input sources
Standard subroutines can be included in your output module, thus reducing your work in coding programs. You can specify that a subroutine be included at a particular time during the processing of your program by furnishing an appropriate control statement. When the Linkage Editor encounters this statement, it retrieves the module containing the subroutine from the indicated input source and includes it in the output module (Fig. 2.9).

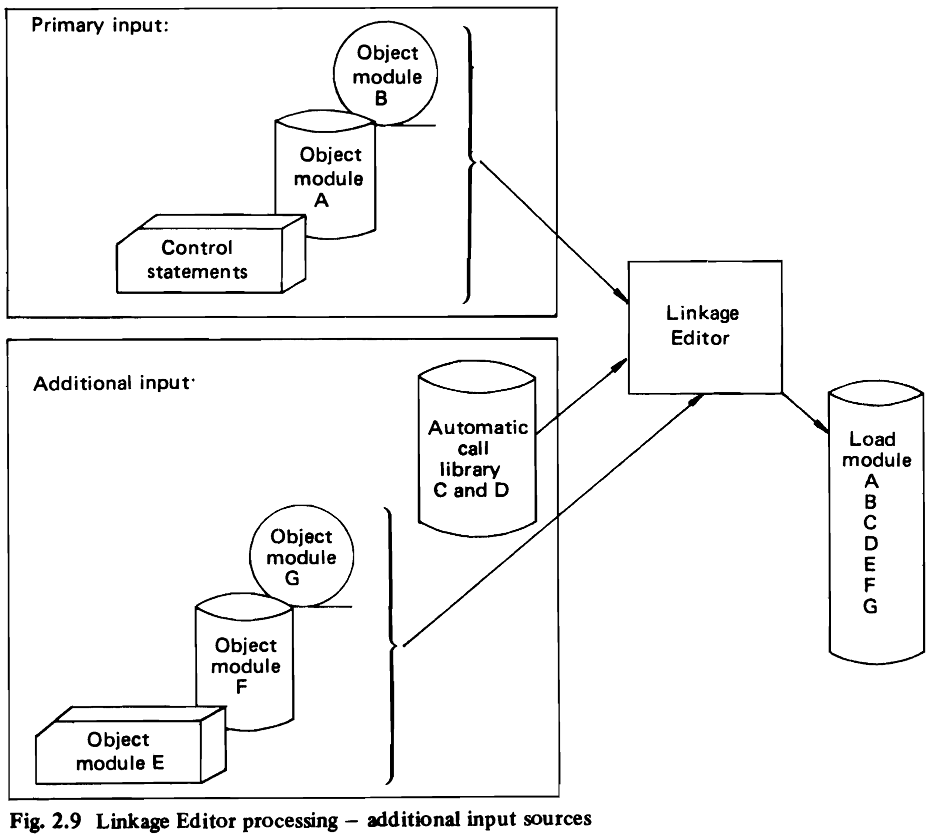


Fig. 2.9 Linkage Editor processing – additional input sources

Symbols still undefined after all input modules have been processed cause the Automatic Library Call facility of the Linkage Editor to search for modules that can resolve these references. When the Linkage Editor finds a module name matching an unresolved symbol, it processes the module and includes it into the output module (Fig. 2.9).

Note: The Linkage Editor also distinguishes a special type of external reference — the **weak external reference**. An unresolved weak external reference does not cause the Linkage Editor to use the Automatic Library Call mechanism. Instead, the reference is left unresolved, but the load module is marked "executable."

Reserves common storage areas
The Linkage Editor processes common control sections generated by FORTRAN compilers and/or the Assembler; static external storage areas generated by the PL/I compiler are processed in the same way. These common areas are collected by the Linkage Editor, which provides a single storage area within the output module.

Processes pseudo registers
Like the external dummy sections of the Assembler Language, the pseudo-register facility helps you to generate re-enterable code. The Linkage Editor processes pseudo registers by accumulating their total storage requirements and recording the displacement of each. During execution, the OS IV/F4 Supervisor helps your program to dynamically acquire necessary storage.

Processes prototype control sections
With the Assembler, COBOL, FORTRAN, or PL/I languages, you can request prototype control sections (PSECTs) for reentrant programs containing data to be changed, work data, and address constants. When you present a prototype control section for linking into a reentrant load module, the Linkage Editor links it with all other prototype control sections in the input stream. If the "reentrant" attribute is not specified, PSECTs are processed in the same way as ordinary control sections (CSECTs).

Each time a reentrant load module is attached or linked, the OS IV/F4 Supervisor copies its PSECT (if any) into a Supervisor-allocated work area within your address space.

Creates overlay programs
To minimize virtual storage requirement, you can organize your program into an overlay structure by dividing it into segments according to functional relationships of the control sections. You can assign the same virtual storage addresses to two or more segments that need not be in virtual storage at the same time.

During execution, you request the OS IV/F4 Supervisor to load these segments at different times. Your control statements specify the relationship of segments within your overlay structure. You place the segments of your load module into a library so that the OS IV/F4 Supervisor can load them separately when executing the load module.

Creates multiple load modules
The Linkage Editor can also create more than one load module within a single job step, as directed. Each load module is placed in the library under a unique member name, as specified by your control statements.

Provides special processing and diagnostic output options
You can specify special processing options that negate (a) Automatic Library Call or (b) effects of minor errors. In addition, the Linkage Editor can produce a module map or cross-reference table that shows the structure of control sections in your output module and indicates how they communicate with one another, plus a list of your control statements.

Throughout processing, errors and possible error conditions are logged. Serious errors cause the Linkage Editor to mark your 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.

Assigns load module attributes
When the Linkage Editor generates a load module, it places an entry for the module into the directory of the corresponding library. This entry contains attributes that describe the structure, content, and logical format of the load module. The OS IV/F4 Supervisor uses these attributes to determine how a module is to be loaded, what it contains, if it is executable, whether it can be multiply executed without reloading, and if it can be executed by concurrent tasks. You can explicitly specify some module attributes; others are determined by the Linkage Editor based on information it gathers during processing.

Allocates user-specified virtual storage areas
You can specify the total amount of virtual storage the OS IV/F4 Supervisor allocates for a particular execution of the Linkage Editor, also the amount to be used for the load module buffer.

Stores system status index information
The following information is intended for systems personnel responsible for maintaining Fujitsu—supplied load modules. The Linkage Editor uses four bytes in the library directory entry for each Fujitsu-supplied load module to store system status index information. This information can be subsequently used for maintenance of the modules. You insert this information

into the directory with a special control statement.

Traces processing history
Tracing the processing history of a program is simplified by the CSECT Identification (IDR) records created and maintained by the Linkage Editor. A CSECT Identification record contains data that describes:
- Language translator (including version level) and translation date for each control section.
- The most recent processing by the Linkage Editor.
- Any modification made to the executable code of any control section.

Optionally, you can enter your own data describing the executable code in one or more control sections.

Lengthens control sections or named common sections
You can lengthen control sections or named common sections of a program to add patch space without changing the source code, reassembling, or recompiling.

Added space is defined as binary zeros; you insert it at the end of a specified control section by using the EXPAND control statement. You cannot add space to a private code (PC) or blank common (CM) section.

Assigns an authorization code to output load modules
The **authorized program facility** (APF) limits the use of sensitive system and (optionally) user services and resources to authorized system and user programs. Program are authorized at the job-step level. For a job step to gain authorization initially, the first module you load at its start must be an authorized module, and you must load the module from an authorized library. Otherwise, the job step is not authorized initially and cannot subsequently gain authorization.

For a job step to maintain its authorization, all subsequent modules you invoke during the job step (via LINK, LOAD, ATTACH, and/or XCTL macro instructions) must be loaded from an authorized library. Otherwise, your job step loses its authorization.

A load module becomes **authorized** if you assign it an authorization code during linkage editing, via the PARM-field parameter AC or the control statement SETCODE, described in the following sections.

2.4 RELATIONSHIP TO REST OF OS IV/F4

The Linkage Editor has the same relationship to OS IV/F4 as any other processing program. You can execute it either as a job step, a subprogram or a subtask. You invoke the Linkage Editor in one of three ways:
- As a **job step.**
- As a **subprogram**, by issuing a CALL macro instruction (after issuing a LOAD macro instruction), a LINK macro instruction, or an XCTL macro instruction.
- As a **subtask** by issuing an ATTACH macro instruction.

You describe some options for the Linkage Editor and its data sets with **job control language statements.** You should not confuse these with **Linkage Editor control statements**. Job control statements are processed befor the Linkage Editor is executed; Linkage Editor control statements are processed during its execution.

Time sharing system (TSS)
When you request linkage editing under the OS IV/F4 TSS, you typically utilize the Linkage Editor Prompter, which acts as an interface between you, the operating system, and the Linkage Editor. Under TSS, you request execution of the Linkage Editor and definition of its data sets by a LINK command that causes the Prompter to be executed. The operands of your LINK command specify Linkage Editor options your job requires. Complete procedures for use of the LINK command are given in the **FACOM OS IV/F4 TSS General Reference Manual.**

2.5 LANGUAGE DEPENDENCIES

This section defines control section, entry name, external reference, common area and pseudo register (external dummy section) in terms of the source-language statements you use to create them. The languages described are Assembler, SL/100, COBOL, FORTRAN, and PL/I.

Assembler language and SL/100
In the Assembler or SL/100 languages, you define a control section by a CSECT or START statement. Either statement may specify a control section name. The control section delimiter is an END statement, or another CSECT or START statement.

You define an entry name with an ENTRY statement.

You define an external refernce to a data area with an EXTRN statement plus an A-type address constant. You specify an external reference to a control section or an entry name with a V-type address constant.

You specify a common area with a COM statement.

You define an external dummy section with a DXD instruction or a DSECT and a Q-type address constant; your subsequent CXD instruction defines a 4-byte field in which the Linkage Editor accumulates the length of all external dummy sections in your load module.

COBOL

The OS IV/F4 COBOL compiler creates at least two control sections for each compoliation. COBOL control sections are always named, since you must specify a name in the PROGRAM-ID paragraph of your IDENTIFICATION DIVISION.

You define an entry name with an ENTRY statement.

You create an external reference each time you issue a CALL statement.

The OS IV/F4 COBOL compiler automatically creates a prototype control section for all modifiable data items and work areas in each compilation.

FORTRAN

In FORTRAN, you define a control section with a SUBROUTINE, FUNCTION, or BLOCK DATA statement that specifies the control section name. If the first statement in your FORTRAN routine is not one of these, the compiler assumes it begins the main routine. Automatically, the statement defines a control section named MAIN (always assigned to the main routine of a FORTRAN program) unless you use a NAME option to assign a name. A control section is defined by a FORTRAN END statement.

You define an entry name with an ENTRY statement.

You create an external reference by each EXTERNAL statement or reference to a subroutine subprogram, function subprogram, or BLOCK DATA subprogram.

You specify a common area with a COMMON statement, which may be optionally named.

The OS IV/F4 FORTRAN GE and HE compilers automatically create prototype control sections for modifiable data items and work areas.

PL/I

In PL/I, you define a control section by the first statement label on each external PROCEDURE statement. If you specify the MAIN option the compiler creates the control section which obtains the address of the principal entry point. In both cases, a control section is compiled to provide appropriate linkage to the library storage management modules. Control section are also created for each STATIC EXTERNAL or EXTERNAL declaration with initial text and for each EXTERNAL file constant.

Note: If labels or variable names used for control section names exceed seven characters, the PL/I compiler generates a seven-character control section name by concatenating the first four and last three characters in the label or variable name.

A control section is also created for STATIC INTERNAL storage; it contains the items declared with their storage class attributes as well as work areas and control blocks added by the compiler. This control section takes its name from the name of the external procedure control section, followed by the letter A and padded to the left with asterisks to a length of eight characters.

You define an entry name with each ENTRY statement.

You define an external reference with an ENTRY declaration, explicitly or implicitly declared with the EXTERNAL attribute. Unresolved function references or procedure calls imply EXTERNAL scope and also cause external references to be generated.

You specify a named common area with a STATIC EXTERNAL or EXTERNAL declaration when the defined area not contain initial text. When the area is initialized, a control section is generated. PL/I does not use blank common areas.

You automatically create a pseudo register with each CONTROLLED variable, file, and PROCEDURE or PROCEDURE BEGIN block or ON unit in the program. The name of the pseudo register created for a CONTROLLED EXTERNAL variable is the name of the variable. In all other cases, the PL/I compiler generates the name of the pseudo register from the external procedure control section name followed by a letter (B, C, etc.), padded to the left with asteriks to a length of eight characters. The asterisks can be replaced, if necessary, to provide sufficient unique names.

CHAPTER 3
INPUTS TO THE LINKAGE EDITOR

The Linkage Editor accepts input from the **primary input data set** and **additional data sets** defined via Automatic Library Call or control statements you furnish. Primary and additional input data sets may contain any/all of the following types of data:

- One or more object modules
- One or more load modules
- Control statements

Object modules and control statements may be contained in either sequential or partitioned data sets.

This chapter describes "linking" functions of the Linkage Editor; "editing" functions are described in Chapter 5.

3.1 PRIMARY INPUT DATA SET

A primary input data set is required for every Linkage Editor job step, defined by a DD statement whose name is "SYSLIN." The primary input can be:

- A sequential data set
- A member of a partitioned data set
- A concatenation of sequential data sets and/or members of partitioned data sets

The primary input data set contains object modules and/or control statements. The modules and control statements are processed sequentially, and their order determines the processing sequence 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.

3.1.1 Object Module Input

The primary input to the Linkage Editor may consist solely of one or more object modules. The rest of this section discusses your options for furnishing object module inputs: from cards, as a member of a partitioned data set, passed from a previous job step, or created in a separate job.

From cards
Each card deck is treated as a sequential data set. You place your cards in the input stream after a DD* statement as follows:

```
//SYSLIN        DD   *
Object Deck A
Object Deck B
/*
```

If you furnish card decks in addition to modules input from other source, your JCL statements should be as follows:

```
//SYSLIN        DD   DSNAME=INPUT,...
//              DD   *
Object Deck A
Object Deck B
/*
```

By omitting the DD name from the second DD statement, card input is concatenated to the INPUT data set described on the SYSLIN DD statement.

As a member of a partitioned data set
You furnish an object module from a partitioned data set to the Linkage Editor by specifying its data set name and member name on your SYSLIN DD statement. In the following example, you designate the member named TAXCOMP in the object-module library LIBROUT as the primary input:

```
//SYSLIN      DD   DSNAME=LIBROUT(TAXCOMP),...
```

The library member is processed as a sequential data set.

Members of partitioned data sets can be concatenated with other input data sets, as follows:

```
//SYSLIN      DD   DSNAME=OBJLIB,
//                 DISP=(OLD, KEEP),...
//            DD   DSNAME=LIBROUT(TAXCOMP),...
```

Library member TAXCOMP is concatenated to data set OBJLIB; both must contain object modules, since they are the primary input.

Passed from a previous job step
You can pass an object module from a previous job step to a linkage edit step in the same job; output from the compiler becomes direct input to the Linkage Editor. In the following example, you pass an object module created in a previous job step (Step A) to the Linkage Editor (Step B):

```
Step A:
//SYSGO       DD   DSNAME=&&OBJECT,
              .    DISP=(NEW, PASS),...
              .
              .
Step B:
//SYSLIN      DD   DSNAME=&&OBJECT,
                   DISP=(OLD, DELETE)
```

Your data set name &&OBJECT identifies your object modules as the output of the language processor on the SYSGO DD statement, and as the primary input to the Linkage Editor on the SYSLIN DD statement.

Note: The double ampersand (&&) in the data set name defines a **temporary data set,** which exists for the duration of your job and is automatically deleted at the end of the job. If you wish to retain the data set longer than one job, the double ampersand must not be used.

You can also use the method of the preceding example to accumulate several modules from previous steps. If you use the same data set for the output of each language processor, your SYSLIN DD statement can retrieve all object modules as follows:

```
Step A:
//SYSGO       DD   DSNAME=&&OBJMOD,
//                 DISP=(NEW, PASS),..
              .
              .
              .
Step B:
//SYSPUNCH    DD   DSNAME=&&OBJMOD,
//                 DISP=(MOD, PASS)
              .
              .
              .
Step C:
//SYSLIN      DD   DSNAME=&&OBJMOD,
//                 DISP=(OLD, DELETE)
```

The two object modules from Step A and Step B are accumulated consecutively in the same sequential data set, &&OBJMOD. The SYSLIN DD statement in Step C causes both object modules to be used as the primary input to the Linkage Editor.

You can use another method to accomplish purpose: **concatenation** of these data sets. If the object modules were created in previous job steps with different member names, you can use the following JCL statements:

```
Step A:
//SYSGO       DD   DSNAME=&&OBJLIB(MODA),
//                 DISP=(NEW, PASS),...
              .
              .
              .
Step B:
//SYSPUNCH    DD   DSNAME=&&OBJLIB(MODB),
//                 DISP=(MOD, PASS),...
              .
              .
              .
Step C:
//SYSLIN      DD   DSNAME=&&OBJLIB(MODA),
//                 DISP=(OLD, DELETE)
//            DD   DSNAME=&&OBJLIB(MODB),
//                 DISP=(OLD, DELETE)
```

You store the object modules created in Step A and Step B into a partitioned data set with different member names. You then concatenate the two members in Step C as primary input.

Created in a separate job
If your only input to the Linkage Editor is an object module from a previous job, your SYSLIN DD statement contains all information necessary to locate the object module, as follows:

```
//SYSLIN      DD   DSNAME=OBJECT,
//                 DISP=(OLD, DELETE),
//                 UNIT=SYSDA,
//                 VOLUME=SER=LIB613
```

3.1.2 Control Statement Input

Your primary input data set may consist solely of control statements. In this case, you can specify your input modules by INCLUDE control statements. Your control statements may be either placed in the input stream or stored in a permanent data set.

In the following example, your primary input consists of control statements in the input stream:

```
//SYSLIN      DD   *
Linkage Editor control statements
/*
```

In the next example, your primary input consists of control statements stored in the member INCLUDES in the partitioned data set CTLSTMTS:

```
//SYSLIN      DD   DSNAME=CTLSTMTS(INCLUDES),
//                 DISP=(OLD, KEEP),...
```

In either case, your control statements can be any of those described in Chapter 9 as long as you follow the rules given there.

3.1.3 Input of Both Modules and Statements

Your primary input to the Linkage Editor can comprise an intermixture of object modules and control statements which may be in the same or different data sets. If your modules and statements are in the same data set, you describe this data set on your SYSLIN DD statement just like any other data set. If your modules and statements are in different data sets, you must concatenate these data sets.

Control statements in the input stream
you can place your control statements in the input stream and concatenate them to an object-module data set as follows:

```
//SYSLIN        DD   DSNAME=&&OBJECT,...
//              DD   *
Linkage Editor control statements
/*
```

Another method of handling control statements in the input stream is to use the DDNAME parameter, as follows:

```
//SYSLIN        DD   DSNAME=&&OBJECT,...
//              DD   DDNAME=SYSIN
                .
                .
                .
//SYSIN         DD   *
Linkage Editor control statements
/*
```

Note: Fujitsu-supplied cataloged procedures for linkage editing use DDNAME=SYSIN for the SYSLIN DD statement, to allow you to specify the required primary input data set.

Control statements in a separate data set
You can concatenate a separate data set containing control statements to a data set containing an object module. It is often good practice to store control statements for a frequently-used procedure in a permanent DASD data set. (For example, a complex overlay structure or a series of INCLUDE statements.) In the following example, CTLSTMTS is a PDS whose members contain Linkage Editor control statements. One of the members is concatenated to data set &&OBJECT:

```
//SYSLIN        DD   DSNAME=&&OBJECT,
//                   DISP=(OLD, DELETE),...
//              DD   DSNAME=CTLSTMTS(OVLY),
//                   DISP=(OLD, KEEP),...
```

3.2 AUTOMATIC LIBRARY CALL

The Autmatic Library Call feature resolves external references remaining unresolved after primary input processing. Unresolved external references found in modules from additional data sources are also processed by this mechanism.

Note: The following discussion of automatic library call does not apply to unresolved **weak external references** which are left unresolved.

Automatic Library Call comprises a search of the directory of the designated library for an entry that matches each unresolved external reference. If a match is found, the entire member is processed as input to the Linkage Editor.

Automatic Library Call can resolve an external reference when the following conditions exist; the external reference must be (1) a member name or alias of a module in the library and (2) defined as an external name in the ESD of the module with that name. If the unresolved external reference is a member name or an alias in the library, but is not an external name in that member, the member is processed but the external reference remains unresolved unless subsequently defined.

Automatic Library Call searches the library defined by your SYSLIB DD statement, which contains either (1) object modules and control statements or (2) load modules; it must not contain both.

You can request that modules from libraries other than the SYSLIB library be searched with Automatic Library Call by furnishing a LIBRARY control statement. The Linkage Editor searches each such library for member names matching specific external references unresolved at the end of input processing. If any unresolved references are found in the modules located by Automatic Library Call, they are resolved by another search of the library. Any external references not specified on a LIBRARY control statement are resolved from the library defined on the SYSLIB DD statement.

You can use the LIBRARY statement to negate Automatic Library Call for **selected** external reference unresolved after input processing. You can use the NCAL option on the EXEC statement to negate Automatic Library Call for **all** external references unresolved after input processing.

3.2.1 SYSLIB DD Statement

If you wish to utilize Automatic Library Call, you must name your library in a DD statement whose name field is "SYSLIB." Your library may be a **system call library** or a **private call library.** Call libraries may be concatenated.

System call library
Most major Fujitsu processing programs have their own automatic call libraries (Table. 3.1). You must explicitly or implicitly define each such library when you wish to link-edit an object module produced by that processor.

A system call library may contain I/O, data conversion, and/or other special routines needed to complete your module. The corresponding processor

Table 3.1 System automatic call libraries

Processing program	Library name
ASSEMBLER	(none)
COBOL	SYS1. COBLIB
FORTRAN GE	SYS1. FORTLIB
FORTRAN HE	SYS1. FORTLIB
PL/I	SYS1. PLIXLIB
SL/100	(none)
Sort/merge	SYS1. SORTLIB

creates external references for these special routines, and the Linkage Editor resolves reference from the appropriate call library.

In the following example, a FORTRAN object module created in Step A is to be link edited in Step B, and the FORTRAN automatic call library is used to resolve external references:

```
Step A:
//SYSOBJ     DD   DSNAME=&&OBJMOD,
//                DISP=(NEW, PASS),..
                .
                .
                .
Step B:
//SYSLIN     DD   DSNAME=&&OBJMOD,
//                DISP=(OLD, DELETE)
//SYSLIB     DD   DSNAME=SYS1. FORTLIB,
//                DISP=SHR
```

The disposition of "SHR" on your SYSLIB DD statement means that other tasks concurrently executing with Step B may also use SYS1. FORTLIB.

Private call libraries
The SYSLIB DD statement can also describe any private library of yours. In this case, Automatic Library Call searches your private library for unresolved external references. In the following example, you request that unresolved external references be resolved from your private library named PVTPROG:

```
//SYSLIB     DD   DSNAME=PVTPROG,
//                DISP=SHR, UNIT=SYSDA,
//                VOLUME=SER=PVT002
```

Concatenation of call libraries
System call libraries and private call libraries may be concatenated to one another, in which case they must all be either object module libraries or load module libraries but not a mixture of the two.

If you wish to link-edit object modules from different system processors to form one load module, you must define all relevant call libraries by concatenating them on your SYSLIB DD statement. In the following example, a FORTRAN object module and a COBOL object module are to be link edited; the two system call libraries are concatenated as follows:

```
//SYSLIB     DD   DSNAME=SYS1. FORTLIB,
//                DISP=SHR
//           DD   DSNAME=SYS1. COBLIB,
//                DISP=SHR
```

You can concatenate a system call library with a private call library in this way. For example, by adding the following statement to the two in the preceding example, your (uncataloged) private call library is concatenated to two system call libraries:

```
//           DD   DSNAME=PVTPROG,
//                DISP=SHR, UNIT=SYSDA,
//                VOLUME=SER=PVT002
```

Any external references not resolved from the two system libraries are resolved from your private library.

3.2.2 LIBRARY Control Statement

You can use a LIBRARY control statement to direct Automatic Library Call to a library other than that specified in the SYSLIB DD statement. Only external references listed on the LIBRARY statement are resolved in this way. All other unresolved external references are resolved from the library named on your SYSLIB DD statement.

You can use a LIBRARY statement to specify external references that are **not** to be resolved by Automatic Library Call. The LIBRARY statement specifies the duration of the nonresolution: either during the current Linkage Editor step, called **restricted no-call**; or during any subsequent Linkage Editor step, called **never-call.**

Examples of each use of the LIBRARY statement follow;a description of its format appears in Chapter 9.

Additional call libraries
If you wish to use one or more additional libraries to resolve specific references, your LIBRARY statement points to the DD statement that describes the library. Your LIBRARY statement also defines which external references are to be resolved from the library, i. e., names of members to be used. If an unresolved external reference is not named in the specified library, the reference remains unresolved unless subsequently defined.

For example, you may have rewritten two modules (DATE and TIME) from a system call library, which you wish to test with their calling modules before replacing the old modules. Because Automatic Library Call would otherwise sear[illegible] he system call library (which is needed for other modules), you could furnish a LIBRARY statement [illegible] follows:

```
//SYSLIB      DD   DSNAME=SYSI,
//                 COBLIB, DISP=SHR
//TESTLIB     DD   DSNAME=TEST,
//                 DISP=(OLD, KEEP),...
//SYSLIN      DD   DSNAME=ACCTROUT,...
//            DD   *
  LIBRARY          TESTLIB(DATE, TIME)
/*
```

Two external references, DATE and TIME, are resolved from the library described on the TESTLIB DD statement. All other unresolved external references are resolved from the library described on the SYSLIB DD statement.

Restricted No-Call function

You can use a LIBRARY statement to specify external references in your output module for which no libraries should be searched during the correct step, by specifying these external references in parentheses without specifying a DDname. These references remain unresolved, but the Linkage Editor marks the module "executable."

For example your program might contain references to two large modules that are automatically called from a library. One of the modules has been tested and corrected, the other is to be tested in this job step. Rather than execute the tested module again, you can use the restricted no-call function to prevent Automatic Library Call from processing the module as follows:

```
//           EXEC PGM=JQAL, PARM=LET
//SYSLIB     DD   DSNAME=PVTPROG,
//                DISP=SHR, UNIT=SYSDA
//                VOLUME=SER=PVT002
                .
                .
                .
//SYSLIN     DD   DSNAME=&&PAYROL,...
//           DD   *
  LIBRARY         (OVERTIME)
/*
```

As a result, the external reference to OVERTIME is not resolved by Automatic Library Call.

3.2.3 Never-Call (NCAL) Option

When you specify the NCAL option in your EXEC-statement PARM parameter, Automatic Library Call is entirely omitted. NCAL is similar to the restricted no-call feature of the LIBRARY statement, except that NCAL negates Automatic Library Call for all unresolved external references while restricted no-call negates Automatic Library Call for selected unresolved external references. With NCAL, all external references unresolved after input processing remain unresolved. However, the module is marked "executable."

3.3 INCLUDED DATA SETS

Your INCLUDE control statements request the Linkage Editor to use additional data sets as input. These can be (a) sequential data sets containing object modules and/or control statements, (b) members of partitioned data sets containing object modules and /or control statements, or (c) load modules.

Each INCLUDE statement points to a DD statement naming the data set furnishing additional input. If your DD statement describes a partitioned data set, your INCLUDE statement must also furnish the name of each member to be used. See Chapter 9 for a detailed description of the format of the INCLUDE statement.

When the Linkage Editor encounters an INCLUDE control statement it processes the module or modules indicated. In Fig. 3.1, the primary input is a sequential data set named PROG which contains an INCLUDE statement. After processing the included

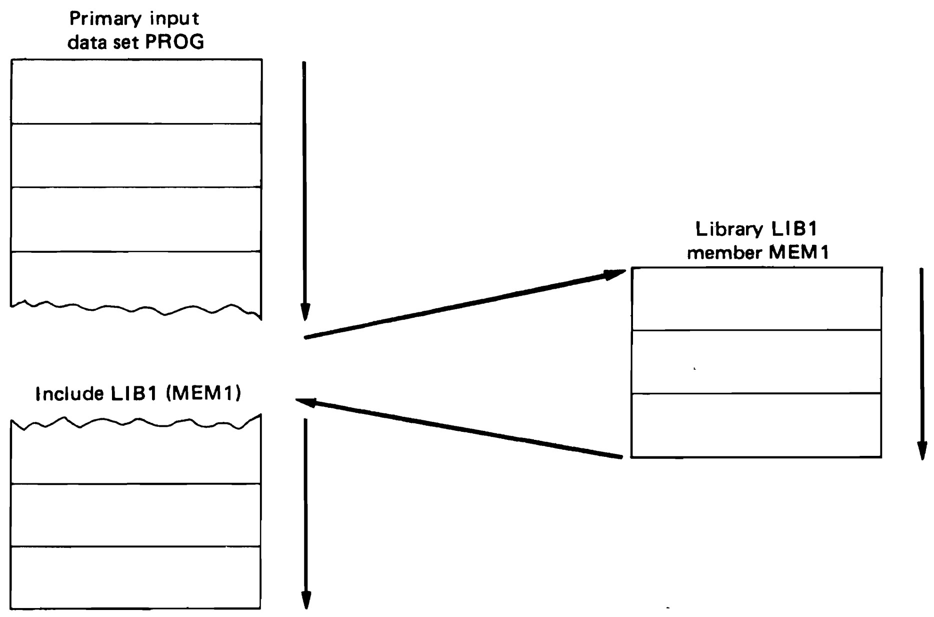


Fig. 3.1 Processing of one INCLUDE statement

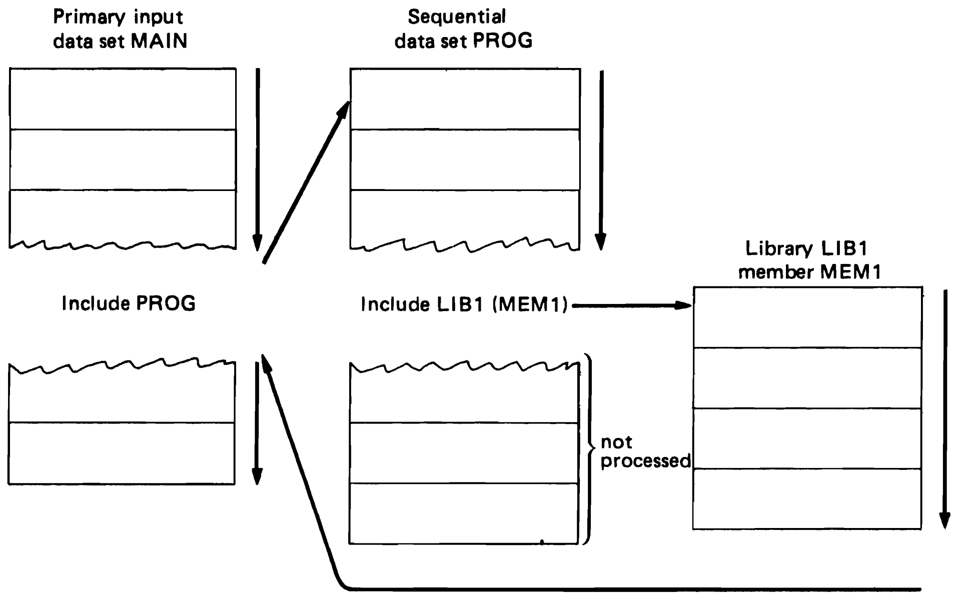


Fig. 3.2 Processing several INCLUDE statements

data set, the Linkage Editor processes the next primary input item. The arrows indicate the flow of processing.

If an included data set also contains an INCLUDE statement, this specified module is also processed. However, any data following the INCLUDE statement are not processed.

If the PROG data set shown in Fig. 3.1 is itself included, any data following the INCLUDE statement for LIB 1 are not processed. Fig. 3.2 shows the flow of processing for this example.

Including sequential data sets

You can specify sequential data sets containing object modules and/or control statements on an INCLUDE control statement. In the following example, an INCLUDE statement points to DD statements describing two sequential data sets used as additional input:

```
//ACCOUNTS   DD   DSNAME=ACCTROUT,
//                DISP=(OLD, KEEP)
//INVENTRY   DD   DSNAME=INVENTRY,
//                DISP=(OLD, KEEP),...
//SYSLIN     DD   DSNAME=QTREND,...
//           DD   *
  INCLUDE    ACCOUNTS, INVENTRY
/*
```

Each DD statement could have been named on a separate INCLUDE statement; with either method you must specify a DD statement for each data set to be included.

Another method of doing the preceding example is given in "Including Concatenated Data Sets" below.

Including library members

You can specify one or more members of a partitioned data set via an INCLUDE control statement. You must specify each member name on your INCLUDE statement, not on the DD statement itself.

In the following example, one member name is specified on the INCLUDE statement:

```
//PAYROLL    DD   DSNAME=&&PAYROUTS,
//                DISP=(OLD, KEEP),...
//SYSLIN     DD   DSNAME=&&CHECKS,
//                DISP=(OLD, DELETE)
//           DD   *
  INCLUDE    PAYROLL (FICA)
/*
```

If you wish to include more than one member of a partitioned data set, your INCLUDE statement specifies all members to be used from each library.

In the following example, an INCLUDE statement specifies two members from each of two libraries to be used as additional input:

```
//PAYROLL    DD   DSNAME=PAYROUTS,
//                DISP=(OLD, KEEP),...
//ATTEND     DD   DSNAME=ATTROUTS,
//                DISP=(OLD, KEEP),...
//SYSLIN     DD   *
  INCLUDE    PAYROLL(FICA, TAX),
             ATTEND(ABSENCE, OVERTIME)
/*
```

Including concatenated data sets

You can designate several data sets with a single INCLUDE statement pointing to one DD statement; additional data sets are then concatenated to the data set described on the specified DD statement. When data sets are concatenated for this purpose, their characteristics must be identical: format, record length, block size, etc.

In the following example, you concatenate two sequential data sets with one INCLUDE statement:

```
//CONCAT      DD   DSNAME=ACCTROUT,
//                 DISP=(OLD, KEEP),..
//            DD   DSNAME=INVENTRY,
//                 DISP=(OLD, KEEP),...
//SYSLIN      DD   DSNAME=SALES,
//                 DISP=OLD,..
//            DD   *
  INCLUDE     CONCAT
/*
```

When the Linkage Editor encounters the INCLUDE statement, it searches the two libraries PAYROUTS and ATTROUTS for the four members, which are then processed as input.

CHAPTER 4
OUTPUTS FROM THE LINKAGE EDITOR

The Linkage Editor produces two types of output: a load module and diagnostic information. The Linkage Editor always stores the load module into a partitioned data set. Error and/or warning messages, module disposition data, and optional diagnostic output are written to the diagnostic output data set.

4.1 OUTPUT LOAD MODULE

The Linkage Editor produces one or more load modules from its input. **Multiple load module** processing is a feature which you request when you create more than one load module in a single linkage-edit job step.

Whether the Linkage Editor produces one or more load modules, it performs the following steps:

- Stores the load module(s) into a partitioned data set called the **output module library**.
- Assigns an entry point to each load module if you have not assigned one.
- Assigns each output load module an authorization code.
- Reserves and collects common areas, as specified in the source-language program.
- Accumulates total length and individual displacements for each pseudo register (external dummy section).
- Deletes any zero-length private code (unnamed) control sections.

4.1.1 Output Module Library

This library is a partitioned data set that must be described by a DD statement named "SYSLMOD." The data set name of the library is also specified on this DD statement. The data set can be **temporary** (defined with a double ampersand) or **permanent** (defined without a double ampersand).

Note: If the data set name is either SYS1. LINKLIB or SYS1. SVCLIB you should request reloading of the OS IV/F4 Supervisor after Linkage Editor processing is complete, to insure that the corresponding Data Extent Block (DEB) is updated to reflect additional extents if secondary allocation of direct-access space was required.

Whether your data set is permanent or temporary, you should assign each load module a unique **member name.** You can assign **aliases** to an output module if you want it to be identified by more than one name or entered for execution at several different points. Each member name and alias in a load module library must be unique. The member name and aliases for each load module appear as separate entries in the library directory, along with the module attributes.

Member name

You can specify the member name for your output load module on your SYSLMOD DD statement, in a NAME statement, or both. If you fail to specify a member name, the default is "TEMPNAME."

If you name the member on your SYSLMOD DD statement, you should supply this name in parentheses immediately after the library name:

```
//SYSLMOD     DD    DSNAME=MATHLIB(SQDEV),
//                  DISP=(NEW, KEEP),
//                  UNIT=SYSDA, SPACE=(TRK,
//                  (100, 10, 1)),
//                  VOLUME=SER=LIB002
```

The member name SQDEV is assigned to the load module stored into the new library named MATHLIB.

If you do not name the member on your SYSLMOD DD statement, you should furnish a NAME control statement, for example:

```
//SYSLMOD     DD    DSNAME=MATHLIB,
//                  DISP=(NEW, KEEP)
//SYSLIN      DD    DSNAME=&&OBJECT,
//                  DISP=(OLD, DELETE)
//            DD    *
  NAME        SQDEV
/*
```

The member name SQDEV is assigned to the load module stored into the MATHLIB library.

If both your SYSLMOD DD statement and your NAME control statement specify a member name, the names should be identical; if different, the NAME control statement takes precedence.

Note: If you request a "link-edit and go" job where the program name in the EXEC statement of your "go" step contains a backward reference to the SYSLMOD DD statement of your link-edit step, you must ensure that the member name specified in the SYSLMOD DD statement is valid and is not overridden by a NAME control statement.

For example:

```
//LKED        EXEC PGM=JQAL
                .
                .
                .
//SYSLMOD     DD   DSNAME=&&LOADST(GO),
//                 DISP=(NEW, PASS),...
//SYSLIN      DD   DSNAME=&&OBJECT,
//                 DISP=(OLD, DELETE)
//            DD   *
  NAME        READ
/*
//GO          EXEC PGM=*. LKED. SYSLMOD
                .
                .
                .
```

The EXEC statement of the GO step specifies that the module to be executed is described in the LKED step in the SYSLMOD statement. The system tries to locate a member named "GO"; however, the output module was assigned the name "READ," and the "GO" step fails to execute correctly.

You can replace an output module with an identically-named member in either of two ways. Your SYSLMOD DD statement can name an existing data set as follows:

```
//SYSLMOD   DD   DSNAME=MATHLIB(SQDEV),
//               DISP=(OLD, KEEP),...
```

Or, your NAME control statement can specify the **replace** function as follows:

```
  NAME      SQDEV(R)
```

In either case, the member named SQDEV is replaced with a new module of the same name.

Alias names

You can assign an output module at most 64 aliases, using ALIAS control statements. (With the AM256 option on your EXEC statement, you can raise this limit to 256). When you refer to a module by an alias, the OS IV/F4 Supervisor begins execution at the external name specified by the alias. If the name specified by the ALIAS statement is not an external symbol within the module, the Supervisor begins execution at the main entry point.

For example, you may wish to assign two additional entry points, CODE1 and CODE2, to a module which has been written and tested using both ROUTONE and OUT1 to refer to its main entry point. Rather than correct calling modules, you can assign an alternate library member name (alias) as follows:

```
//SYSLMOD     DD    DSNAME=PVTLIB,
//                  DISP=OLD, UNIT=SYSDA,
//                  VOLUME=SER=LIB001
//SYSLIN      DD    DSNAME=&&OBJECT,
//                  DISP=(OLD, DELETE)
//            DD    *
  ALIAS       CODE1, CODE2, ROUTONE
  NAME        ROUT1
/*
```

The names CODE1, CODE2, and ROUTONE appear in the library directory along with ROUT1, the member name. Because CODE1 and CODE2 are defined as external symbols within the output module, you can request that execution begin at either of these secondary entry points. Control may be passed to the main entry point by using either the member name ROUT1 or the alias ROUTONE.

4.1.2 Entry Point

Every load module must have one main entry point, which you may specify in one of two ways:

- On a Linkage Editor ENTRY control statement.
- On an Assembler-language END statement, the last statement in each Assembler source program. The Assembler produces an EOM card for each object module. This object deck defines an entry point only if the source-language END statement contained one.

From its input, the Linkage Editor selects the entry point for the load module as follows:

1. From the first ENTRY statement in the input.
2. If there is no ENTRY statement in the input, from the first Assembler-produced END statement that specifies an entry point.
3. Otherwise the first byte of the first control section of the load module is used as the entry point.

In general, you should explicitly specify an entry point since you cannot always predict which control section will be first in the output module.

You may specify entry points other than the main entry point with an ALIAS control statement. The symbol specified on the ALIAS statement must be defined as an external symbol in the load module. Any reference to that symbol causes execution of the module to begin at that point instead of the main entry point.

In the following example, assume you have defined CDCHECK, CODE1, and CODE2 as external symbols in your output module:

```
//SYSLIN       DD   DSNAME=&&OBJECT,
//                  DISP=(OLD, DELETE)
//             DD   *
  ENTRY CDCHECK
  ALIAS CODE1, CODE2, ROUTONE
  NAME ROUT1
/*
```

As a result of the preceding control statements, CDCHECK is the main entry point; CODE1 and CODE2 are additional entry points. Any reference to ROUTONE or ROUT1 causes execution to begin at CDCHECK; any reference to CODE1 and CODE2 causes execution to begin at these points.

Authorization code

As you link edit each load module, the Linkage Editor assigns an authorization code that determines whether or not your module is allowed to use restricted system services and resources. A non-zero code allows the module to use restricted services and resources, a zero code disallows such usage. The authorization code becomes part of the directory entry for your module in the corresponding library.

4.1.3 Common Areas

In the FORTRAN, Assembler, SL/100, and PL/I languages, you can create control sections containing no data or instructions. These control sections are called **common** or **static external** areas; they typically are used as communication regions for different parts of a program or to reserve virtual storage areas for data supplied at execution time. These common areas are either **named** or **unnamed (blank common)**.

The Linkage Editor collects common areas; if two or more blank common areas are found in the input, the largest is reserved in the output module, and all references to "blank common" refer to the one retained. If you furnish two or more common areas with the same name, the largest of these identically-named areas is reserved in the output module; all references to common areas with this name use this area.

If a control section generated from a BLOCK DATA subprogram in FORTRAN and a named common area have the same name, the control section must be at least as long as the named common area. If the control section is smaller than the named common area, a diagnostic message is issued. The control section is regarded as the largest of the common areas processed with that name. All subsequent control sections and/or common areas with the same name are ignored.

4.1.4 Pseudo Registers

With PL/I you can use **pseudo registers** to define storage to be allocated dynamically during execution. External dummy sections generated by the OS IV/F4 Assembler and SL/100 compiler correspond to PL/I pseudo registers.

The Linkage Editor accumulates the total length of all pseudo registers and records their displacements. If two or more pseudo registers have the same name, the one with the longest length and most restrictive alignment will be retained. All other pseudo registers with the same name will be ignored; all references to identically-named pseudo registers will refer to the one retained.

4.1.5 Multiple Load Modules

The Linkage Editor can produce several load modules in a single job step. You furnish NAME control statements as delimiters for these load modules. Each load module has a unique name and is stored into the same library as a separate member. Options and attributes you specify in your EXEC statement for that job apply to all new load modules. If the Linkage Editor terminates abnormally, none of the modules yet to be processed in the job step is processed or placed in the library. Load modules processed before abnormal termination remain successfully stored in the library.

You should not specify a member name on your SYSLMOD DD statement when creating multiple load modules.

In the following example, you create two load modules in one Linkage Editor job step:

```
//LKED         EXEC PGM=JQAL,
//                  PARM='MAP, LIST'
                .
                .
                .
//SYSLMOD      DD   DSNAME=PAYROLL(OVERTIME),
//                  DISP=OLD, UNIT=SYSDA,
//                  VOLUME=SER=LIB002
                .
                .
                .
//MODTWO       DD   DSNAME=&&OBJECT,
//                  DISP=(OLD, DELETE)
//SYSLIN       DD   DSNAME=&&OBJECT(A),
//                  DISP=(OLD, DELETE)
//             DD   *
  ENTRY        INIT
  NAME         OVERTIME
  INCLUDE      MODTWO(B)
  ENTRY        HSKEEP
  NAME         VACATION
/*
```

The first load module is produced from the object module in the data set defined on the SYSLIN DD statement. The main entry point is INIT and the member name is OVERTIME.

The second load module is produced from the object module specified by the INCLUDE statement.

The main entry point is HSKEEP and the member name is VACATION.

Both load modules are stored in the PAYROLL library defined on the SYSLMOD statement. Note that the member name specified on the SYSLMOD statement is identical to the name given the first load module.

Parameters on your EXEC card specify that you wish a module map and control statement listing for each load module. These maps and listings are discussed in detail in Section 4.2.

4.2 DIAGNOSTIC OUTPUTS

The Linkage Editor creates information and diagnostic information which it directs to the data set defined by your SYSPRINT DD statement. In addition to routine messages generated by the Linkage Editor, you can request various optional outputs.

4.2.1 Diagnostic Messages

The Linkage Editor generates two types of messages module disposition messages, and error/warning messages. Descriptions of the latter can be found in Appendix B.

Module disposition messages

Several module disposition messages are printed for each load module. The first indicates the options and attributes you have specified for each module, indicated by A in Figs. 4.1 and 4.3. Invalid options or attributes are replaced by "INVALID" in the output, and the Linkage Editor also notifies you whenever it finds the attributes you specify are incompatible.

Disposition messages describe processing of each load module, indicated by G in Figs. 4.1 and 4.3. These messages are preceded by several asterisks:

```
****** 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.
****** alias name   IS AN ALIAS FOR THIS MEMBER.
****** MODULE HAS BEEN MARKED NOT
       EXECUTABLE.
```

In addition, the Linkage Editor issues module-disposition messages when you specify re-enterable (RENT), reusable (REUS), or refreshable (REFR) attribute. Each message indicates whether the load module has been marked "re-enterable" or "not reenterable", "reusable" or "not reusable," "refreshable" or "not refreshable," depending on option(s) used. See "Reusability Attributes" and "Refreshable Attribute" in Chapter 8 for more information on these options.

The message consists of several asterisks and "MODULE HAS BEEN MARKED" followed by attribute(s) you have assigned. You are responsible for verifying that your module actually is re-enterable, reusable, and/or refreshable. In particular, the OS IV/F4 Supervisor may mishandle a loaded copy of your module if the latter does not behave according to attributes you assigned while link-editing it.

The following messages are examples of some possible combinations:

```
****** MODULE HAS BEEN MARKED REFRESHABLE.
****** MODULE HAS BEEN MARKED NOT
       REFRESHABLE.
****** MODULE HAS BEEN MARKED REUSABLE AND
       NOT REFRESHABLE.
****** MODULE HAS BEEN MARKED REUSABLE AND
       REFRESHABLE.
```

When an error causes the Linkage Editor to mark a module "not executable," only the MODULE HAS BEEN MARKED NOT EXECUTABLE message appears; no attribute messages are generated.

Error/warning messages

If an error is encountered during linkage editing, the message code for that error is printed together with the erroneous symbol or record. (This is indicated by C in Figs. 4.1 and 4.3.) After processing is completed, the diagnostic message associated with that code is printed. (This is indicated by H in Figs. 4.1 and 4.3.) Error/warning messages have the following format:

JQA0mms message text

where **JQA0** indicates a Linkage Editor message; **mm** is the message number; **s** is the severity code, and may be one of the following values:

- Condition may cause one or more errors during execution of the output module.

  A module map or cross-reference table is produced if specified by the programmer.
- Execution of the output module may fail. Processing continues. When possible, a module map and/or cross-reference table is produced if you have requested them. The output module is marked "not executable" unless the LET option is specified on the EXEC statement.
- Execution of the output module is impossible. Processing continues. When possible, a module map and/or cross-reference table is produced if you have requested them. The output module is unconditionally marked "not executable."
- Error condition precluding recovery. Processing terminates at once. The only output are diagnostic messages.

  A special severity code of zero 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 standard severity-zero message code is "JQA0000".

  The Linkage Editor multiples its highest severity code by 4 to create a return code in register 15 at the end of processing. You can test this return code to determine whether or not processing should continue, as described in Section 8.2.6.

The **message text** contains combinations of the following:

- Message classification (error or warning)
- Cause of error
- Identification of the symbol, segment number (for an overlay structure), or input item to which the message applies
- Instruction for your corrective/interpretive actions
- Action taken by the Linkage Editor

Optionally, you can direct error/warning messages to a separate output data set by specifying "TERM" in the PARM field of your EXEC statement and including a SYSTERM DD statement. This separate SYSTERM data set contains only numbered error/warning messages. It supplements the SYSPRINT output data set, which can also include module disposition messages and optional diagnostic output. If you request SYSTERM output, the numbered error/warning messages appear in both data sets.

Appendix B contains a complete list of error/warning messages for the Linkage Editor and Loader, together with full explanations of probable causes of errors and how you should correct them.

Sample diagnostic output
Fig. 4.1 shows sample diagnostic outputs from the Linkage Editor. No optional outputs were requested other than the list of control statements. The leftmost letters indicate the disposition and error/warning messages as follows:

- **A** is a module disposition message that lists the options and attributes you have specified. Additional information is printed indicating all explicit and default options.
- **B** is a list of control statements you furnished, each preceded by "JQA0000".
- **C** is a list of error/warning messages, in this case citing problematical unresolved symbols (STRINGP and OUTPROC1) prefixed by "JQA0461".
- **G** is a module disposition message (****) that indicates that the new load module (ACCUM) has been added to the output data set, plus its authorization code (0).
- **H** is the diagnostic message directory that contains the text of the error codes listed in item C.

4.2.2 Optional Outputs

In addition to error/warning and disposition messages, you can ask the Linkage Editor to produce optional outputs such as a listing of your control statements, a module map, and a cross-reference table.

Control statement listing
If you specify the LIST option on your EXEC statement, the Linkage Editor creates a listing of your control statements. For each control statement, the listing contains a special message code (JQA0000) followed by the statement itself. Item B in Figs. 4.1 and 4.3 lists the NAME control statement you have furnished (only control statement in these examples).

Module map
If you specify the MAP option on your EXEC statement, the Linkage Editor prints a map of the output load module, showing all control sections in the output module and all entry names in each control section. Named common areas are listed as control sections.

For each control section, the module map indicates its origin (relative to zero) and length in bytes (in hexadecimal notation) as shown in Fig. 4.2, item D. For each entry name in each control section, the module map indicates the location at which the name

```
A     (V-02/L-02)  OS IV/F4 LINKAGE EDITOR OPTIONS SPECIFIED LIST,XREF,NCAL
                   DEFAULT OPTION(S) USED-SIZE=(120832,36864)

B     JQA0000      NAME ACCUM

C     JQA0461      STRNGP
      JQA0461      OUTPROC1

G     ****ACCUM         NOW ADDED TO DATA SET
      AUTHORIZATION CODE IS    0.
                        DIAGNOSTIC MESSAGE DIRECTORY

H     JQA0451-W    WARNING:  SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCENCAL
                   WAS SPECIFIED OR MARKED FOR RESTRICTED NO-CALL OR NEVERCALL
```

Fig. 4.1 Diagnostic messages issued by the Linkage Editor

is defined.

If your module is not in an overlay structure, the control sections are arranged in ascending order according to their origins. Each entry name is listed within the control section in which it is defined.

If your module is an overlay structure, its control sections are arranged by segment, listed as they appear in the overlay structure: top to bottom, left to right, and region by region. Within each segment, the control sections and their corresponding entry names are listed in ascending order according to their assigned origins. The number of the segment in which they appear is also listed.

In a module map, any of the following is prefixed by the currency symbols ($):

- Blank common area,
- Private code (unnamed control section).
- For overlay programs, the segment table and each entry table.

If your load module does not have an origin of zero, the Linkage Editor generates a one-byte private-code control section as its first text record. This private code is deleted in any subsequent reprocessing of the load module by the Linkage Editor.

Each control section obtained from a library via Automatic Library Call is identified by an asterisk after its control section name.

At the end of the module map is the entry address, the relative address of the main entry point, followed by the total length of the module in bytes, as shown by item F in Fig. 4.2 through 4.4. In the case of an overlay module, the length is that of the longest path. Any pseudo registers also appear at the end of the module maP; the name, length, and displacement of each pseudo register is displayed.

Cross reference table

If you specify the XREF option on your EXEC statement, the Linkage Editor prints a cross-reference table consisting 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 referenced symbol, and the name of the control section in which the symbol is defined. In cases where control sections are compiled together and simple address constants are used to refer from one control section to another (instead of using external symbols and entry names), the control section name is listed as the referenced symbol.

For overlay programs, this information is provided for each segment; in addition the Linkage Editor provides the number of the segment in which the symbol is defined.

If a symbol remains unresolved after processing by the Linkage Editor, it is identified as "$UNRESOLVED" in the list. However, if an unresolved symbol is marked by the never-call function (as specified on a LIBRARY control statement), it is identified "$NEVER—CALL". If an unresolved symbol is a weak external reference, it is identified "$UNRESOLVED(W)".

Fig. 4.2 contains a cross-reference table (item E for the program whose diagnostic messages appeared in Fig. 4.1. Fig. 4.3 shows the complete SYSPRINT listing in correct sequence. Fig. 4.4 shows a complete listing for a FORTRAN program including four unresolved weak external references.

```
                                          CROSS REFERENCE TABLE

     CONTROL SECTION                      ENTRY

    { NAME    ORIGIN  LENGTH                NAME     LOCATION   NAME    LOCATION   NAME   LOCATION   NAME   LOCATION
    | MAINP      00     1C80
    |                                     COMAREA       D74     COM1       115C     COM2      155C    PUTLP      185C
 (D){                                     LPDCB        195C     LPAREA     1A24     GETCD     1AA0    CDDCB      1B68
    |                                     CDAREA       1C30
    | ERPROC   1C80      504
    |                                     EROPT        1CA8
    ( INITPROC 2188     1080

    { LOCATION   REFERS TO SYMBOL  IN CONTROL SECTION          LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION
    |    200         INITPROC          INITPROC                    40C       STRNGP           $UNRESOLVED
 (E){    410         OUTPROC1         $UNRESOLVED                  1CD0      PUTLP              MAINP
    |   3128         COMAREA           MAINP                       312C      COM1               MAINP
    |   3130         LPDCB             MAINP                       3134      LPAREA             MAINP
    (   3138         PUTLP             MAINP                       3204      ERPROC             ERPROC

    { ENTRY ADDRESS    00
 (F){
    ( TOTAL LENGTH     3208
```

Fig. 4.2 Module map and cross reference table

```
(V-02/L-02) OS IV/F4 LINKAGE EDITOR OPTIONS SPECIFIED LIST, XREF, NCAL }
            DEFAULT OPTION(S) USED - SIZE=(120832,36864)               } Ⓐ
JQA0000          NAME   ACCUM                                          } Ⓑ
JQA0461     STRNGP                                                     } Ⓒ
JQA0461     OUTPROC1

                               CROSS REFERENCE TABLE

CONTROL SECTION                     ENTRY

  NAME     ORIGIN  LENGTH             NAME      LOCATION   NAME    LOCATION   NAME    LOCATION   NAME    LOCATION
MAINP         00     1C80
                                    COMAREA        D74     COM1       115C    COM2       155C    PUTLP       185C   } Ⓓ
                                    LPDCB         195C     LPAREA     1A24    GETCD      1AA0    CDDCB       1B68
                                    CDAREA        1C30
ERPROC      1C80      504
                                    EROPT         1CA8
INITPROC    2188     1080

LOCATION   REFERS TO SYMBOL   IN CONTROL SECTION          LOCATION   REFERS TO SYMBOL   IN CONTROL SECTION
    200         INITPROC          INITPROC                     40C         STRNGP           $UNRESOLVED
    410         OUTPROC1        $UNRESOLVED                   1CD0         PUTLP              MAINP          } Ⓔ
   3128         COMAREA           MAINP                       312C         COM1               MAINP
   3130         LPDCB             MAINP                       3134         LPAREA             MAINP
   3138         PUTLP             MAINP                       3204         ERPROC             ERPROC

ENTRY ADDRESS      00                    }
                                         } Ⓕ
TOTAL LENGTH     3208                    }
****ACCUM    NOW ADDED TO DATA SET       } Ⓖ
AUTHORIZATION CODE IS              0.    }

                             DIAGNOSTIC MESSAGE DIRECTORY

JQA0461-W  WARNING : SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE - NCAL WAS SPECIFIED OR MARKED FOR } Ⓗ
           RESTRICTED NO-CALL OR NEVERCALL.
```

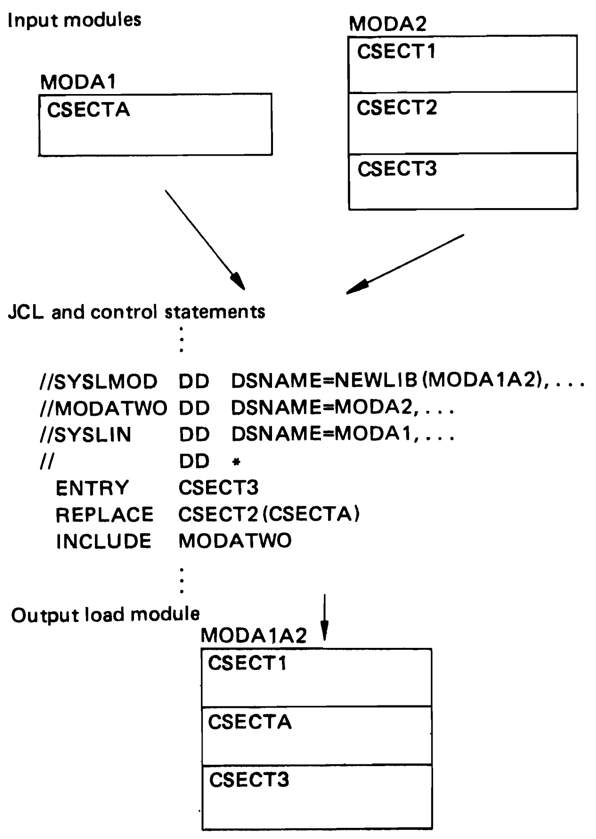
Fig. 4.3 Complete SYSPRINT listing

```
(V-02/L-02) OS IV/F4 LINKAGE EDITOR OPTIONS SPECIFIED LIST, XREF
            DEFAULT OPTION(S) USED – SIZE=(120832,36864)
****GO                DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
AUTHORIZATION   CODE IS           0.

                                                CROSS REFERENCE TABLE

CONTROL SECTION                        ENTRY

  NAME        ORIGIN    LENGTH          NAME     LOCATION    NAME     LOCATION    NAME     LOCATION    NAME     LOCATION
MAIN            00        186
JMFLBCME*      188        E86
                                       F#RCON        188   IBCOM#        188   MDAIOE#       244   FDIOCS#       244
                                       INTSWA        F48   INTSWTCH      F48
JMFLBCMA*     1010        740
                                       DAIRECT      1278
JMFIOCVT*     1750        A2D
                                       CVTAD#       1750   CVTAOUT      17FC   CVTLOUT      1884   CVTZOUT      19E0
                                       CVTIOUT      1DA0   CVTEOUT      1EAA   CVTCOUT      1EAA
JMFINTEM*     2180        690
                                       FPINT#       2180
JMFERTBL*     2810        4C8
JMFSQIOE*     2CD8        FF8
                                       SQIO#        2CD8   FIOCS#       2CD8   SQIOERR      2CDC   FIOCSBEP     2CDC
JMFSQIOA*     3CD0        598
JMFCKERR*     4270        DD3
                                       EROUT#       4270   ERPRM#       44CC
JMFERM*       5048        688
                                       ERRMON       5048   ERRSUM       5064
JMFICONF*     56D0        31D
                                       ICONEF#      56D0
JMFDSTBL*     59F0        638
JMFOCONF*     6028        4CA
                                       OCONEF#      6028
JMFICONE*     64F8         A0
                                       ICONVE#      64F8
JMFOCONE*     6598         9E
                                       OCONVE#      6598
JMFTRBEM*     6638        308
                                       JMFTRB       6638   ERRTRA       6640
JMFPTEN*      6940        198
                                       PTEN#        6940
JMFQPTEN*     6AD8        110
                                       QPTEN#       6AD8
JMFISNE*      6BE8         D0
                                       F#RERR       6BE8   IBERH#       6BE8

LOCATION    REFERS TO SYMBOL    IN CONTROL SECTION    LOCATION    REFERS TO SYMBOL    IN CONTROL SECTION
      D0        F#RCON              JMFLBCME              244         DAIRECT             JMFLBCMA
     29C        JMFLBCMA            JMFLBCMA              E24         SQIO#               JMFSQIOE
     E38        FPINT#              JMFINTEM              E34         CVTAD#              JMFIOCVT
     E28        JMFERTBL            JMFERTBL              E50         CVTZOUT             JMFIOCVT
     E48        CVTIOUT             JMFIOCVT              E44         CVTEOUT             JMFIOCVT
     E4C        CVTLOUT             JMFIOCVT              E40         CVTCOUT             JMFIOCVT
     E3C        CVTAOUT             JMFIOCVT              E2C         JMFASYCM            $UNRESOLVED(W)
     E30        DPROF#            $UNRESOLVED(W)          DA4         JMFERM              JMFERM
     E20        JMFLBCMA            JMFLBCHA              E54         ERRSUM              JMFERM
     E58        EROUT#              JMFCKERR             1134         JMFLBCME            JMFLBCME
    1F38        F#RCON              JMFLBCME             1F34         JMFERM              JMFERM
    1F3C        OCONEF#             JMFOCONF             1F40         ICONEF#             JMFICONF
    2800        F#RCON              JMFLBCME             2808         CVTAD#              JMFIOCVT
    2804        SQIO#               JMFSQIOE             280C         INTSWTCH            JMFLBCME
    269C        JMFERM              JMFERM               3B60         JMFASYCM            $UNRESOLVED(W)
    3B04        JMFDSTBL            JMFDSTBL             3B10         F#RCON              JMFLBCME
    3B5C        JMFSQIOA            JMFSQIOA             3B64         JMFERM              JMFERM
    3B68        EROUT#              JMFCKERR             3B6C         ERPRM#              JMFCKERR
    3B25        JMFSQIOA            JMFSQIOA             3B3C         JMFSQIOA            JMFSQIOA
    3CC9        JMFSQIOA            JMFSQIOA             44C0         F#RCON              JMFLBCME
    44C4        JMFSQIOE            JMFSQIOE             44C8         JMFERM              JMFERM
    56BC        F#RCON              JMFLBCME             56C0         JMFERTBL            JMFERTBL
    56C4        CVTAD#              JMFIOCVT             56C8         JMFTRBEM            JMFTRBEM
    56CC        SQIOERR             JMFSQIOE             5968         JMFICONE            JMFICONE
    5964        QPTEN#              JMFQPTEN             5960         PTEN#               JMFPTEN
    6434        JMFOCONE            JMFOCONE             6430         QPTEN#              JMFQPTEN
    642C        PTEN#               JMFPTEN              67F4         F#LDIO              $UNRESOLVED(W)
    67E4        SQIOERR             JMFSQIOE             67E8         JMFERTBL            JMFERTBL
    67EC        F#RCON              JMFLBCME             6938         F#RCON              JMFLBCME
    67F0        CVTAD#              JMFIOCVT             693C         CVTAD#              JMFIOCVT
    67F8        F#RERR              JMFISNE              6C48         JMFERM              JMFERM
    6C4C        F#RCON              JMFLBCME
ENTRY ADDRESS   00

TOTAL LENGTH    6CB8
```

Fig. 4.4 SYSPRINT listing of a representative FORTRAN program

CHAPTER 5 EDITING MODULES

The Linkage Editor performs editing on a control-section basis. That is, you can modify a control section within an object or load module without recompiling the entire source program. You can modify an entire control section or external symbols within a control section. Control sections can be deleted, replaced, or arranged in sequence; external symbols can be deleted or changed. **External symbols** comprise control section names, entry names, external references, named common areas, and pseudo registers. Changes, deletions, or replacements are made to input modules; requested alternations control Linkage Editor processing as shown in Fig. 5.1.



```
  :
//SYSLMOD  DD  DSNAME=NEWLIB(MODA1A2),...
//MODATWO  DD  DSNAME=MODA2,...
//SYSLIN   DD  DSNAME=MODA1,...
//         DD  *
  ENTRY    CSECT3
  REPLACE  CSECT2(CSECTA)
  INCLUDE  MODATWO
  :
```

Fig. 5.1 Editing a module

Editing conventions
You must follow certain conventions to ensure that your modifications are processed correctly. These conventions concern the following items:

- entry point for the new module,
- placement of control statements,
- identical old and new symbols.

Each time the Linkage Editor reprocesses your load module, you should specify the entry point for the output module in one of two ways:

- on an ENTRY control statement
- on the Assembler-produced END statement of an input object module, if one is present. If the entry point specified in the Assembler-produced END statement is not defined in the object module, you must define the entry name as an external reference.

The entry point you assign must be defined as an external name within the resulting load module.

Each control statement (such as CHANGE or REPLACE) specifying an editing function must precede either the module to be modified or the INCLUDE statement that specifies the module. If your INCLUDE statement specifies several modules, your CHANGE or REPLACE statement applies only to the first module included.

The same symbol should not appear as both an old external symbol and a new external symbol in one Linkage Editor run. If you wish to replace a control section with another identically-named control section, the Linkage Editor handles this automatically, as described in Section 5.2.

5.1 CHANGING EXTERNAL SYMBOLS

You can direct the Linkage Editor to change an external symbol to a new symbol while processing an input module. External references and address constants within the module automatically refer to the new symbol. You must use separate control statements to update external references from other modules to a changed external symbol.

You specify both the old and the new symbols with a CHANGE or REPLACE control statement. How the old symbol was used within the module determines whether the new symbol becomes a control section name, an entry name, or an external reference. The old symbol appears first, followed by the new symbol in parentheses.

The CHANGE control statement changes a control section name, an entry name, or an external reference. The REPLACE statement changes or deletes an entry name; if the symbols on a REPLACE statement are control section names, the entire control section is replaced or deleted, as described in Section 5.2.

Your CHANGE statement must immediately precede (a) the input module containing the external symbol to be changed or (b) the INCLUDE statement specifying the input module. The scope of the CHANGE statement spans the immediately-following object module or load module. The END record in the immediately-following object module or EOM indication in the load module terminates the scope of the CHANGE statement.

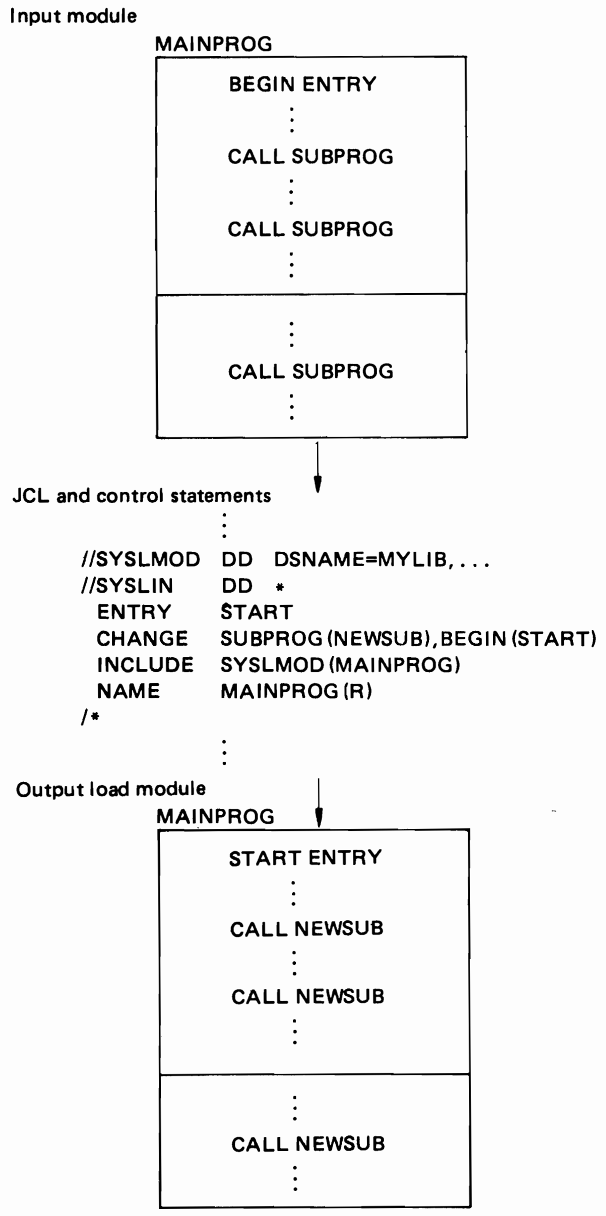
In the following example, assume that SUBPROG is defined as an external reference in the input load module. Your CHANGE statement changes the external reference to NEWSUB as shown in Fig. 5.2.

```
//SYSLMOD       DD      DSNAME=PVTLIB,
//                      DISP=OLD,
//                      UNIT=SYSDA,
//                      VOLUME=SER=PVT002
//SYSLIN        DD      *
  ENTRY         BEGIN
  CHANGE        SUBPROG (NEWSUB)
  INCLUDE       SYSLMOD (MAINPROG)
  NAME          MAINPROG (R)
/*
```

In the load module MAINPROG, every reference to SUBPROG is changed to NEWSUB. Note also that the INCLUDE statement specifies a DD name of SYSLMOD. This allows you to use a library both as input and to receive your output module.

You can specify more than one change on one control statement. If, in the same example, the entry point is also to be changed, the two changes can be specified at once, as shown in Fig. 5.2.

Your main entry point is now START instead of BEGIN. Your ENTRY control statement specifies the new entry point, which enters the library directory entry for the load module.



```
//SYSLMOD  DD  DSNAME=MYLIB,...
//SYSLIN   DD  *
  ENTRY    START
  CHANGE   SUBPROG(NEWSUB),BEGIN(START)
  INCLUDE  SYSLMOD(MAINPROG)
  NAME     MAINPROG(R)
/*
```

Fig. 5.2 Changing an external reference and an entry point

5.2 REPLACING CONTROL SECTIONS

You can replace an entire control section with a new control section, either automatically or using a REPLACE control statement. Automatic replacement can occur for any input module but a REPLACE statement acts only upon the module that immediately follows it.

Note 1: Any CSECT Identification (IDR) records associated with a particular control section are also replaced.

Note 2: For Assembler language programmers: when you wish to replace some but not all control sections of a separately assembled module, A-type address constants that refer to a deleted symbol will be incorrectly resolved unless the entry name is at the same displacement from the origin in both the old and new control sections. If you replace all control sections of a separately-assembled module, no restrictions apply.

5.2.1 Automatic Replacement

The Linkage Editor automatically replaces control sections if both the old and new control sections have the same name. The first of the identically-named control sections processed by the Linkage Editor enters the output module. All subsequent identically-named control sections are ignored; external references to identically-named control sections are resolved with respect to the first one processed. Therefore, you must give the new control section the same name as the old control section if you wish the latter to be automatically replaced.

Automatic replacement applies only to duplicate **control section names.** If duplicate **entry points** exist in control sections with different names, you must furnish a REPLACE statement specifying the entry point name. If a control section being automatically replaced contains unresolved external references and the control section replacing it does not, you must either (a) specify the NCAL parameter, (b) delete the unresolved external references explicitly with a REPLACE statement, or (c) mark them for restricted no-call or never-call using a LIBRARY statement. Otherwise, the unresolved external references are retained.

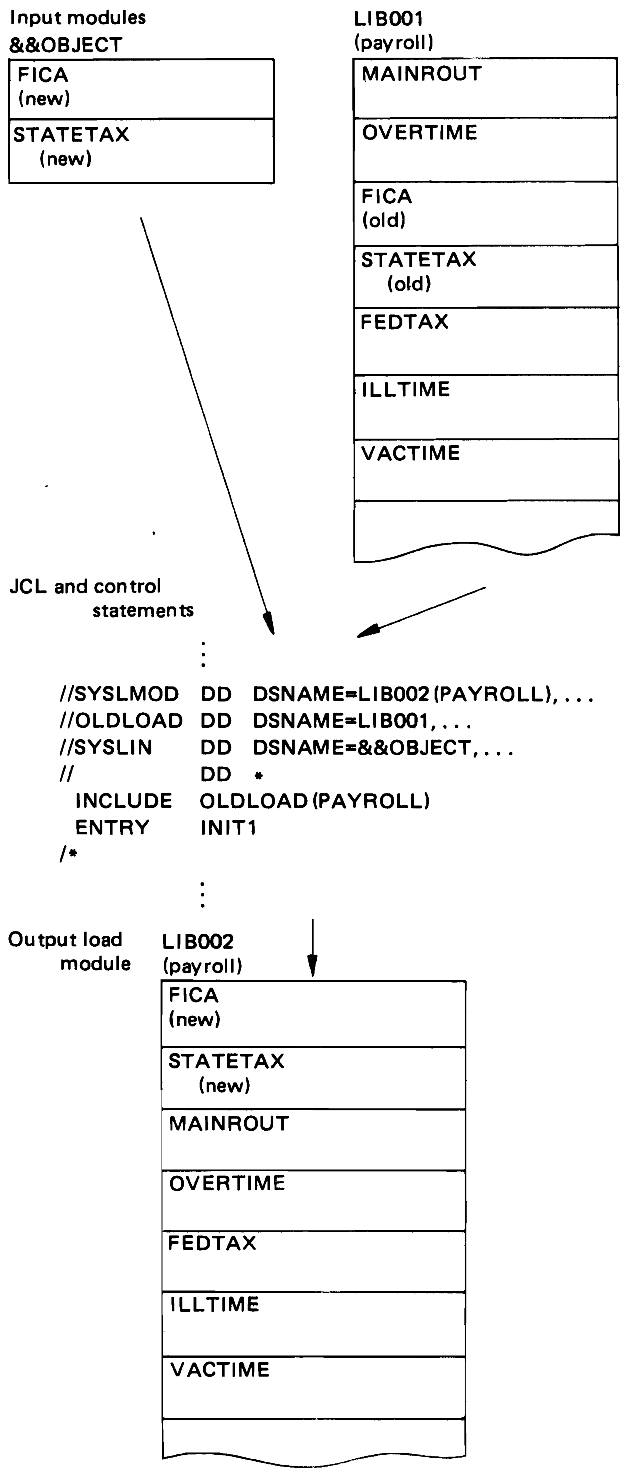
When identically-named control sections appear in modules being placed into an overlay structure, the second and subsequent control sections with that name are ignored, whether the modules are in segments in the same path or in exclusive segments. Resolution of external references may therefore cause invalid exclusive references. Invalid exclusive references cause the Linkage Editor to mark the output module "not executable" unless the XCAL option is specified on the EXEC statement.

Example 1
An object module deck contains two control sections, READ and WRITE; member INPUT of library PVTLIB also contains a control section WRITE.

```
//SYSLMOD        DD       DSNAME-PVTLIB,
//                        DISP=OLD,
//                        UNIT=SYSDA
//                        VOLUME=SER=PVT002
//SYSLIN         DD       *
Object Deck for READ
Object Deck for WRITE
  ENTRY          READIN
  INCLUDE        SYSLMOD(INOUT)
  NAME           INOUT(R)
/*
```

The output load module contains the new READ control section, the new WRITE control section (replacing the old WRITE control section in member INOUT), and all remaining control sections from INOUT.

Example 2
A large load module named PAYROLL, originally written in COBOL, contains many control sections. You recompile two control sections, FICA and STATETAX, and pass them to the Linkage Editor in the &&OBJECT data set. By including the PAYROLL load module, a member of the partitioned data set LIB001, as well as the recompiled object module, you cause modified control sections automatically to replace identically-named control sections, as shown in Fig. 5.3.



```
                     :
//SYSLMOD   DD   DSNAME=LIB002(PAYROLL),...
//OLDLOAD   DD   DSNAME=LIB001,...
//SYSLIN    DD   DSNAME=&&OBJECT,...
//          DD   *
  INCLUDE   OLDLOAD(PAYROLL)
  ENTRY     INIT1
/*
                     :
```

Fig. 5.3 Automatic replacement of control sections

```
//SYSLMOD     DD   DSNAME=LIB002(PAYROLL),
//                 DISP=OLD,
//                 UNIT=SYSDA,
//                 VOLUME=SER=LIB002
//SYSLIB      DD   DSNAME=SYS1.COBLIB,
//                 DISP=SHR
//OLDLOAD     DD   DSNAME=LIB001,
//                 DISP=(OLD,DELETE),
//                 UNIT=SYSDA,
//                 VOLUME=SER=LIB001
//SYSLIN      DD   DSNAME=&&OBJECT,
//                 DISP=(OLD,DELETE)
//            DD   *
  INCLUDE     OLDLOAD (PAYROLL)
  ENTRY       INIT1
/*
```

Your output module contains the modified FICA and STATETAX control sections and the rest of the control sections from the old PAYROLL module. Your main entry point is INIT1, and your output module is stored into the library named LIB002. The COBLIB system library resolves any external references remaining unresolved after the SYSLIN data sets are processed.

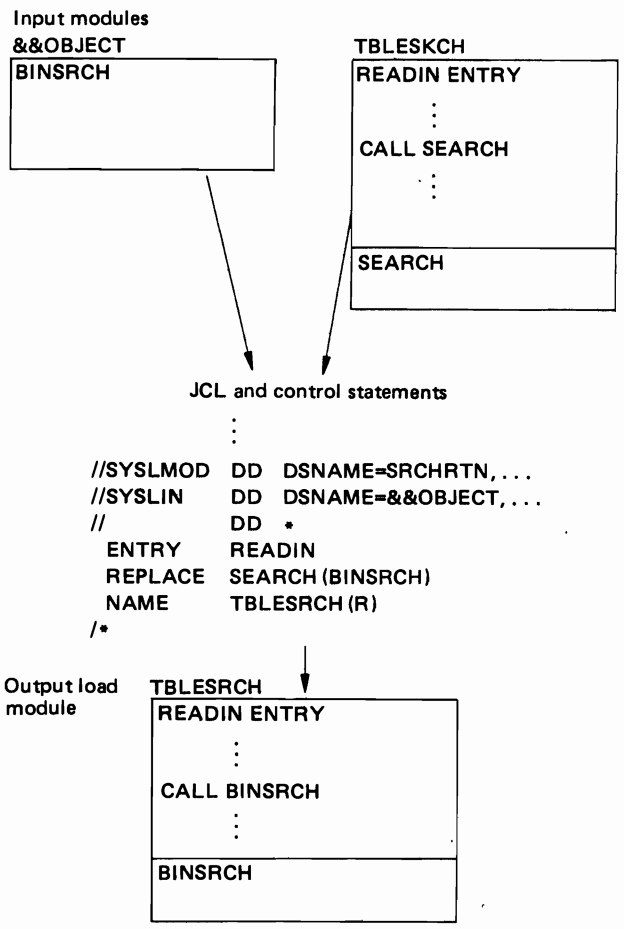
5.2.2 REPLACE Statement

You use a REPLACE statement to replace control sections when the old and new control sections have different names. The name of the old control section appears first, followed by the name of the new control section in parentheses. The REPLACE statement must immediately precede either the input module that contains the control section to be replaced or the INCLUDE statement that specifies the input module. The scope of the REPLACE statement spans the immediately-following object module or load module. The END record in the immediately-following object module or the EOM indication in the load module terminates the action of the REPLACE statement.

An external reference to the old control section from within the same input module is resolved to the new control section. An external reference to the old control section from any other module becomes an unresolved external reference unless one of the following occurs:

- You change the external reference to the old control section to the new control section with a separate CHANGE control statement.
- The same entry name appears in the new control section or in some other control section in the input stream.

In the following example, you use a REPLACE statement to replace one control section with another of a different name. Assume that the old control section SEARCH is in library member TBLESRCH, and that the new control BINSRCH is in the data set &&OBJECT, which you passed from a previous step (Fig. 5.4).



```
//SYSLMOD  DD  DSNAME=SRCHRTN,...
//SYSLIN   DD  DSNAME=&&OBJECT,...
//         DD  *
  ENTRY    READIN
  REPLACE  SEARCH (BINSRCH)
  NAME     TBLESRCH (R)
/*
```

Fig. 5.4 Replacing a control section with the REPLACE control statement

```
//SYSLMOD     DD   DSNAME=SRCHRTN,
//                 DISP=OLD,
//                 UNIT=SYSDA,
//                 VOLUME=SER=SRCHLIB
//SYSLIN      DD   DSNAME=&&OBJECT,
//                 DISP=(OLD,DELETE)
//            DD   *
  ENTRY       READIN
  REPLACE     SEARCH (BINSRCH)
  INCLUDE     SYSLMOD (TBLESRCH)
  NAME        TBLESRCH (R)
/*
```

Your output module contains BINSRCH instead of SEARCH; any references to SEARCH within the module refer to BINSRCH. Any external references to SEARCH from other modules will not be resolved to BINSRCH.

5.3 DELETING CONTROL SECTIONS AND ENTRY NAMES

You can use a REPLACE statement to delete a control section or an entry name. Your REPLACE statement must immediately precede either the module that contains the control section (or entry name) to be deleted or the INCLUDE statement that specifies the module. Only one symbol appears on each REPLACE statement; the appropriate deletion is made depending on how the symbol is defined in the module.

If the symbol is a control section name, the entire control section is deleted. The control section name is deleted from the ESD only if no address constants refer to it from within the same input module. If an address constant does refer to it, the control section name is changed to an external record.

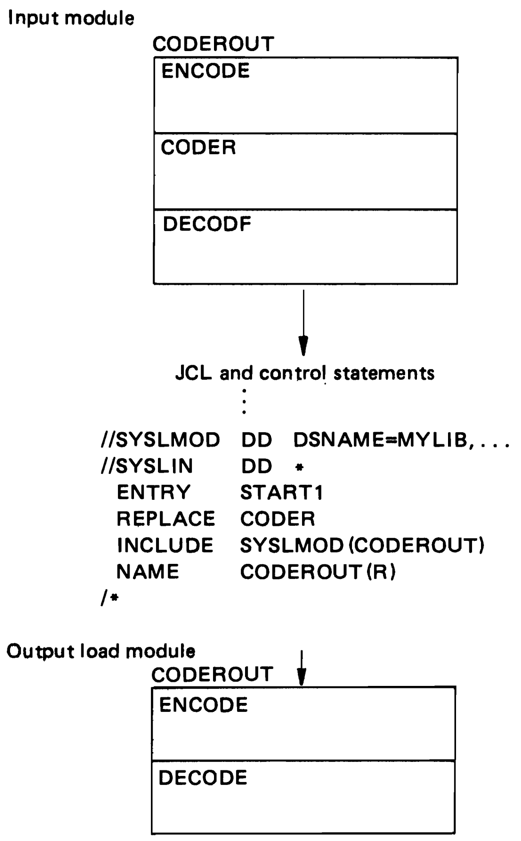
The preceding is also true of an entry name to be deleted. Any references to it from within the input module cause the entry name to be changed to an external reference.

Unless resolved with other input modules, these editor-supplied external references cause Automatic Library Call to attempt to resolve them. Also, deleting a control section or any entry name may cause external references from other input modules to become unresolved. Either condition can cause the output load module to be marked "not executable."

If a deleted control section contains an unresolved external reference, the reference remains.

Note: When you delete a control section, any CSECT Identification data associated with that control section is also deleted.

In the following example, you wish to delete control section CODER (Fig. 5.5):



```
//SYSLMOD  DD  DSNAME=MYLIB,...
//SYSLIN   DD  *
  ENTRY    START1
  REPLACE  CODER
  INCLUDE  SYSLMOD(CODEROUT)
  NAME     CODEROUT(R)
/*
```

Fig. 5.5 Deleting a control section

```
//SYSLMOD       DD   DSNAME=PVTLIB,
//                   DISP=OLD,
//                   UNIT=SYSDA
//                   VOLUME=SER=PVT002
//SYSLIN        DD   *
  ENTRY         START1
  REPLACE       CODER
  INCLUDE       SYSLMOD(CODEROUT)
  NAME          CODEROUT(R)
/*
```

CODER is deleted, and if no address constants refer to it from other control sections in the module, its control section name is also deleted. If address constants refer to CODER, the name is retained as an external reference.

5.4 ORDERING CONTROL SECTIONS AND NAMED COMMON AREAS

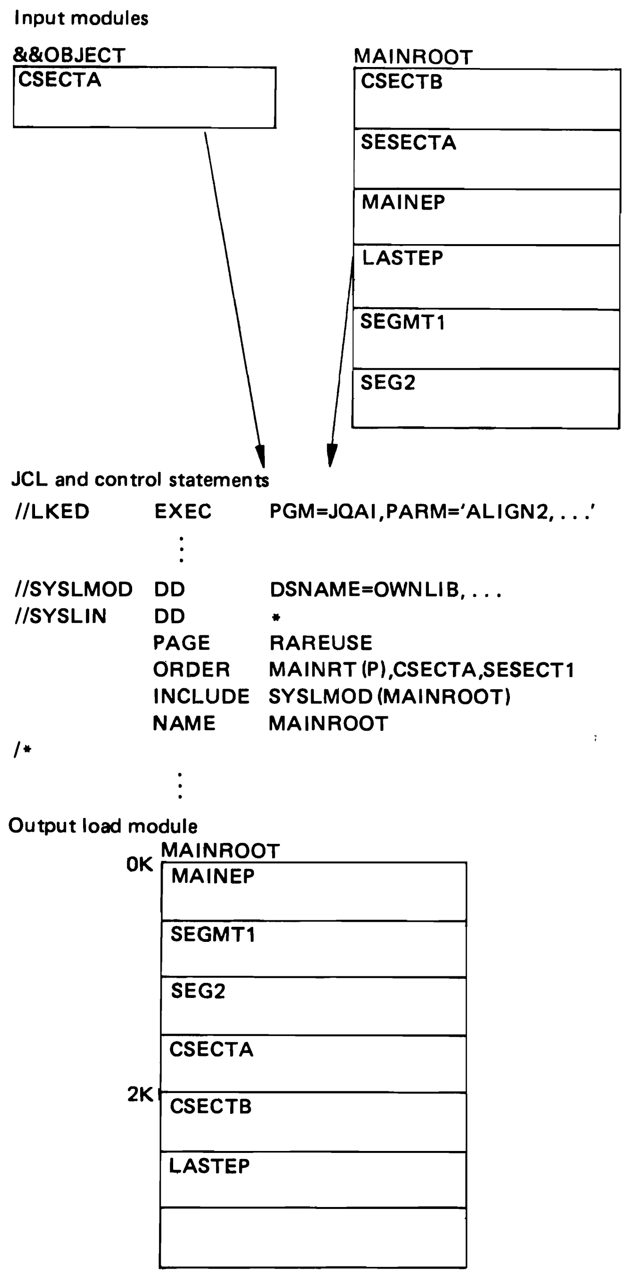
You can sequence control sections and named common areas within an output load module with ORDER control statements. Individual control sections or named common areas are arranged in the output load module according to the sequence in which you defined them on ORDER statements. If you furnish several ORDER statements in a job step, their sequence determines the sequence of control sections or named common areas in the load module.

Any control sections or named common areas you do not specify on ORDER statements appear last in the output load module. If a control section or named common area is changed by a CHANGE or REPLACE control statement, you must use the new name on your ORDER statement.

In the following example, you provide ORDER statements to sequence five of the six control sections in an output load module. Your REPLACE statement replaces the old control section SESECTA with CSECTA from the data set &&OBJECT, which was passed from a previous step. Assume that the control sections to be ordered are found in library member MAINROOT (Fig. 5.6):

```
//SYSLMOD       DD   DSNAME=PVTLIB,
//                   DISP=OLD,
//                   UNIT=SYSDA,
//                   VOLUME=SER=PVT002
//SYSLIN        DD   DSNAME=&&OBJECT,
//                   DISP=(OLD,DELETE)
//              DD   *
  ORDER         MAINEP(P),SEGMT1,SEG2
  REPLACE       SESECTA(CSECTA)
  ORDER         CSECTA,CSECTB(P)
  INCLUDE       SYSLMOD(MAINROOT)
  NAME          MAINROOT
/*
```

In the load module MAINROOT, the Linkage Editor rearranges control sections MAINEP, SEGMT1, SEG2, CSECTA, CSECTB in the output load module



```
//LKED      EXEC     PGM=JQAI,PARM='ALIGN2,...'
               :
//SYSLMOD   DD       DSNAME=OWNLIB,...
//SYSLIN    DD       *
            PAGE     RAREUSE
            ORDER    MAINRT (P),CSECTA,SESECT1
            INCLUDE  SYSLMOD (MAINROOT)
            NAME     MAINROOT
/*
               :
```

Fig. 5.6 Ordering control sections

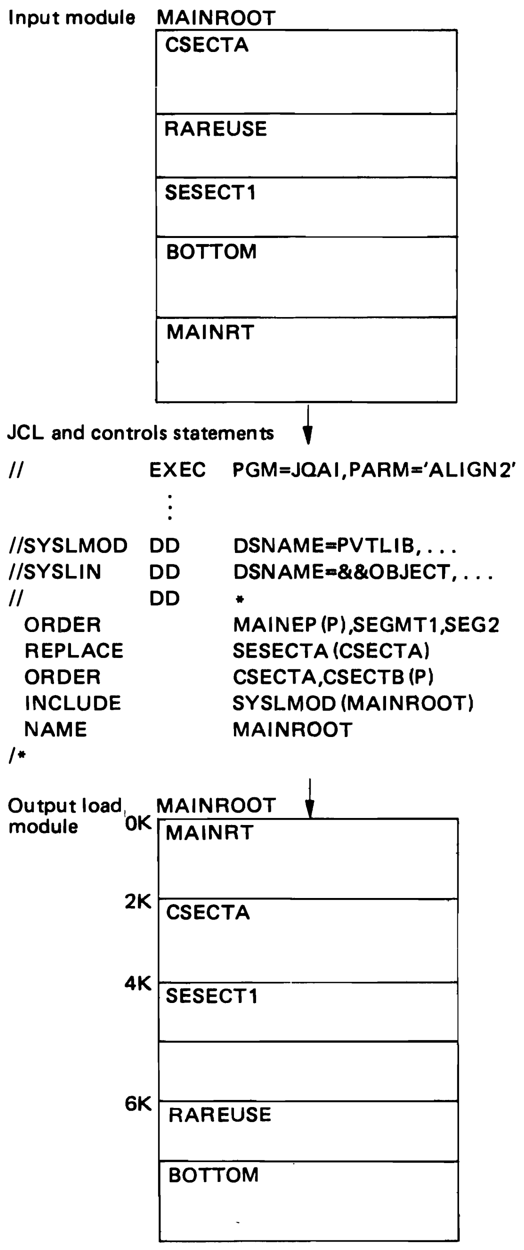
according to the sequence specified by your ORDER statements. Your REPLACE statement replaces SESECTA with CSECTA from the data set &&OBJECT, which was passed from a previous step. The ORDER statement refers to the new control section CSECTA. LASTEP appears after other control sections in the output load module because you did not name it on your ORDER statements.

5.5 ALIGNING CONTROL SECTIONS AND NAMED COMMON AREAS ON PAGE BOUNDARIES

You can align a control section or named common area on a page boundary (2K or 4K bytes) by using either an ORDER statement (with the "P" operand) or a PAGE statement. Alignment on a page boundary tends to lower the paging rate and thus make more efficient use of real storage.

You name each control section or common area to be aligned on either a PAGE statement or an ORDER statement with the P operand. Either the PAGE or ORDER statement causes the Linkage Editor to locate the starting address of the control section or common area on a page boundary within the load module. The default value for a page boundary is 4K.

In the following example, the control sections RAREUSE and MAINRT are aligned on 2K page boundaries by your PAGE and ORDER control statements in conjunction with the ALIGN2 attribute. Control sections CSECTA and SESECT1 are sequenced by your ORDER control statement. Assume that each control section is 2K in length except for SESECT1 and RAREUSE (Fig. 5.7).



```
//              EXEC   PGM=JQAI,PARM='ALIGN2'
                  :
//SYSLMOD       DD     DSNAME=PVTLIB,...
//SYSLIN        DD     DSNAME=&&OBJECT,...
//              DD     *
  ORDER                MAINEP (P),SEGMT1,SEG2
  REPLACE              SESECTA (CSECTA)
  ORDER                CSECTA,CSECTB (P)
  INCLUDE              SYSLMOD (MAINROOT)
  NAME                 MAINROOT
/*
```

Fig. 5.7 Aligning control sections on page boundaries

```
//LKED          EXEC PGM=JQAL,
//                   PARM='ALIGN2,...'


//SYSLMOD       DD   DSNAME=OWNLIB,
//                   DISP=OLD,
//                   UNIT=SYSDA,
//                   VOLUME=SER=OWN002
//SYSLIN        DD   *
  PAGE          RAREUSE
  ORDER         MAINRT(P),CSECTA,SESECT1
  INCLUDE       SYSLMOD(MAINROOT)
  NAME          MAINROOT
/*
```

The Linkage Editor places the control sections MAINRT and RAREUSE on 2K page boundaries because you specified ALIGN2 on your EXEC statement. Control sections MAINRT, CSECTA, and SESECT1 are sequenced as specified in the ORDER statement. While placed on a 2K page boundary, RAREUSE appears after control sections specified in the ORDER statement because it was not cited in this statement. BOTTOM comes after RAREUSE because it appeared after RAREUSE in the input module.

CHAPTER 6 OVERLAY STRUCTURES

Ordinarily when you create a load module, you intend that all of its control sections remain in virtual storage throughout execution. The length of the load module is therefore the sum of the lengths of its control sections. If storage space is not at a premium—typically the case on OS IV/F4—this is the most efficient way to execute a program. However, if your program is very large and forces a high paging overhead when loaded, you should consider using the overlay facilities of the Linkage Editor.

To convert an ordinary program to overlay structure, you need only add special control statements to the module in most cases. You identify the overlayable portions of your program, and OS IV/F4 automatically loads required portions during its execution.

When you request an overlay structure, your load module is segmented so that, at execution time, certain control sections are loaded only when referenced. When a reference is made from one executing control section to another, the OS IV/F4 Supervisor determines whether or not the code required is already in virtual storage. If it is not, the code is loaded dynamically and may overlay an unneeded part of the module already in storage.

Passing control in an overlay structure is basically identical to the CALL linkage of a simple structure. This chapter describes overlay structures, how to load overlay segments, how to pass control to segments, and how to use the SEGLD and SEGWT macro instructions.

6.1 DESIGN

The way in which you should define the structure of an overlay module depends on the relationships among its control sections. Two control sections that need not be in storage at the same time can overlay each other. Such control sections are **independent:** not referencing each other directly or indirectly. Independent control sections can be assigned the same load addresses and loaded only when needed. For example, control sections that handle error conditions or unusual data are used infrequently and need not occupy storage unless in use.

Control sections are grouped into segments. A **segment** is the smallest functional unit (one or more control sections) that can be loaded as one logical entity during execution. Control sections required continuously are grouped into a special segment called the **root segment**. This segment remains in storage throughout execution of an overlay program.

When a particular segment is to be executed, any segment between it and the root segment must also be in storage. This is a **path**. A reference from one segment to another segment lower in a path is a **downward reference**—the segment refers to another segment farther from the root. Conversely, a reference from one segment to another segment higher in a path (closer to the root segment) is an **upward reference**.

A downward reference causes overlay if the necessary segment is not yet in virtual storage. An upward reference can not cause overlay, since all segments between the executing segment and the root must already be in storage. These concepts are illustrated in Figs. 6.1 and 6.2.

Sometimes several paths need the same control sections. This problem may be solved by placing these control sections into another region. In an overlay structure, a **region** is defined as a contiguous area of virtual storage within which segments can be loaded independently of paths in other regions. An overlay program can be designed for single or multiple regions. Note that "region" in this sense is unrelated to an "OS IV/F4 region", the latter often used synonymous with "address space."

6.1.1 Single-Region Structures

For your overlay structure, you should include in the root segment those control sections that will receive control at the beginning of execution, plus those that should always remain in storage. You develop the rest of the structure by determining dependencies among the remaining control sections and how they can use the same virtual-storage locations at different times during execution.

Besides control-section dependency, other topics discussed in this section are inter-segment dependency, length of the overlay program, segment origin, communication between segments, and overlay processing.

Control section dependency

Control section dependency is determined by requirements of one control section for a given routine in another control section. A control section is **dependent** upon any control section from which it receives control or which processes its data. For example, if control section C receives control from control section B, then C is dependent upon B—both control sections must be in storage before execution can continue beyond a given point in the program.

As an example, assume your program contains seven control sections, CSA through CSG, and exceeds available/efficient storage for a simple structure. Rather than rewrite your program, you might evaluate whether or not it could be placed into an overlay structure. Fig. 6.1 shows the groups of dependent control sections in your program, arrows indicating dependencies.

Each dependent group is also a path. For example, if control section CSG is to be executed, CSB and CSA must also be in storage. Because CSA and CSB are in each path, they must be in the root segment. Control section CSC is in two groups and therefore is a **common segment** in two different paths.

A better way to show the relationship between segments is with a tree structure. A **tree** is a graphical representation of how segments can use virtual storage at different times. It does not imply the order of execution, although the root segment is always the first to receive control. Fig. 6.2 shows the tree structure for the dependent groups shown in Fig. 6.1. The structure is contained in one overlay region and has five segments.

Segment dependency

When a segment is in virtual storage, all segments in its path are also necessarily in virtual storage. Each time a segment is loaded, all segments in its path are loaded if not already in virtual storage. In Fig. 6.2, when segment 3 is in virtual storage, segments 1 and 2 are also in virtual storge. However, if segment 2 is in storage, this does not imply that segment 3 or 4 is in virtual storage, since neither segment is in the path of segment 2.

The positions of segments in an overlay tree structure does not imply the sequence in which they execute. You can load, overlay, and reload a segment as many times as required by the logic of your program. However, a segment cannot overlay itself. If you modify a segment during execution, that modification survives only until you overlay the segment.

Length

For purposes of illustration, assume that the control sections in your sample program have the following lengths:

Control Section	Length (in bytes)
CSA	3,000
CSB	2,000
CSC	6,000
CSD	4,000
CSE	3,000
CSF	6,000
CSG	8,000

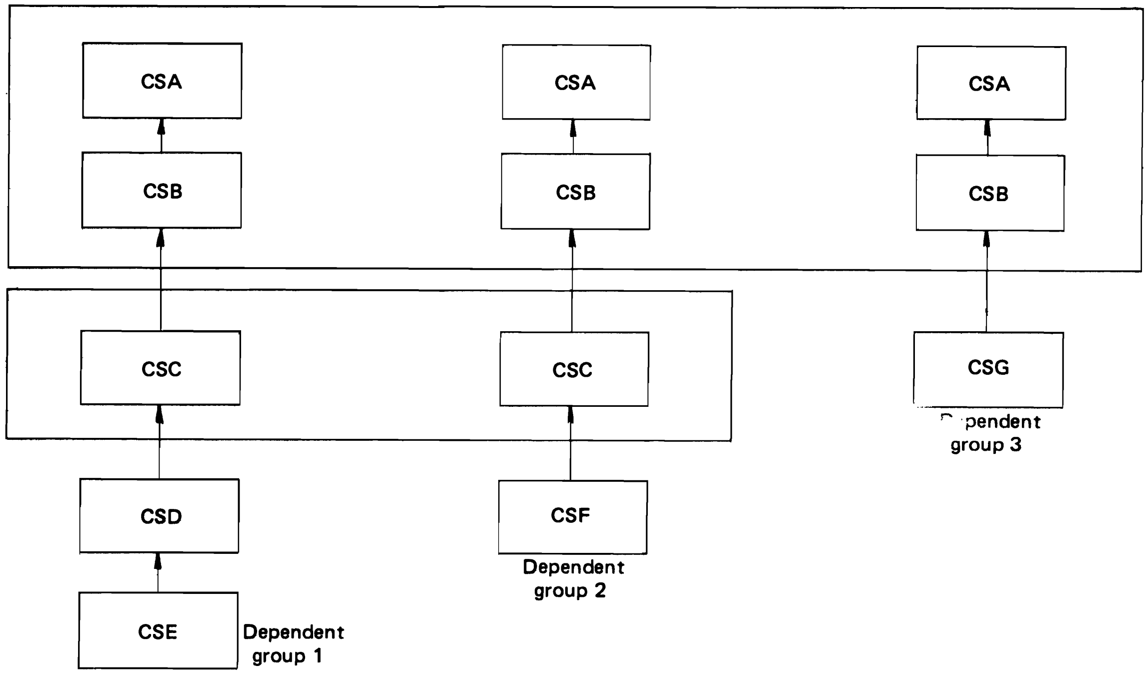


Fig. 6.1 Control section dependencies

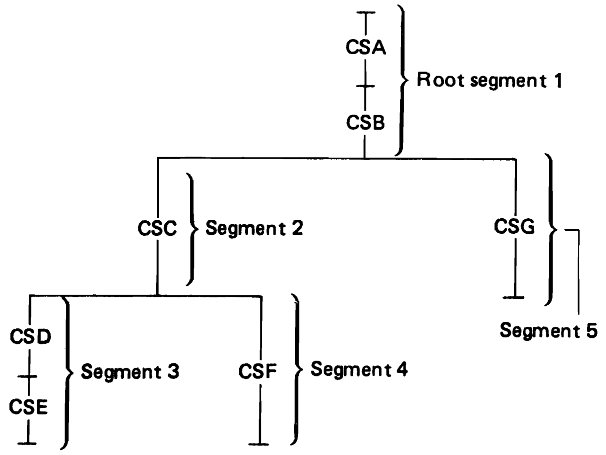


Fig. 6.2 Single-region overlay tree structure

If your program were not overlay-structured, it would require 32,000 bytes of virtual storage. In overlay, however, the program requires the amount of storage needed for the longest path. In this structure, the longest path is defined by segments 1, 2, and 3; when they are all in storage, they require 18,000 bytes, as shown in Fig. 6.3.

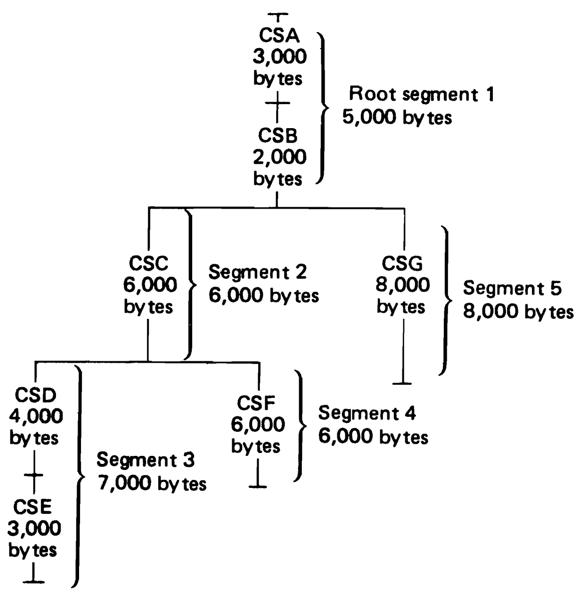


Fig. 6.3 Length of an overlay module

Segment origin

The Linkage Editor assigned the relative origin of the root segment (the origin of the program) at 0. The relative origin of each segment equals the length of all segments in its path. For example, the origin of segments 3 and 4 is 11,000: 6,000 (the length of segment 2) plus 5,000 (the length of the root segment). The origins of all the segments are as follows:

Segment	Origin
1	0
2	5,000
3	11,000
4	11,000
5	5,000

The segment origin is also called the **load point**, because it is the relative location at which the segment is loaded.

Fig. 6.4 shows the segment origin for each segment and the way storage is used by the sample program. In the illustration, the vertical bars indicate segment origins; all segments with the same origin may use the same storage area. Fig. 6.3 also shows that the longest path is defined by segments 1, 2, and 3.

Communication between segments

Segments that can be in virtual storage simultaneously are considered to be **inclusive**. Segments in the same region but not in the same path are considered to be **exclusive**; they cannot be in virtual storage simultaneously. Fig. 6.5 shows inclusive and exclusive segments in the sample program.

Segments upon which two or more exclusive segments depend are called **common segments**. A segment common to two other segments is part of the path of each segment. In Fig. 6.5, segment 2 is common to segments 3 and 4 but not to segment 5.

An **inclusive reference** is one between inclusive segments—from a segment in storage to an external symbol in a segment that does not overlay the calling segment. An **exclusive reference** is one between exclusive segments—from a segment in storage to an external symbol in a segment that overlays the calling segment.

Fig. 6.6 shows the difference between an inclusive reference and an exclusive reference; the arrows indicate references between segments.

Wherever possible, you should make inclusive rather than exclusive references. Inclusive references between segments are always valid and do not require special options. When you make inclusive references, you are less likely to commit errors in structuring your overlay program.

When your external reference within a requesting segment is to a symbol defined in a segment outside the current path, you have issued an exclusive reference, which is **valid** only if you cite the requested control section in a segment common to both the segment to be loaded and the segment to be overlaid. You must use the same symbol in the common segment and the exclusive reference. In Fig. 6.6, your reference from segment B to segment A is valid because there is an inclusive reference from the common segment to segment A.

In the same illustration, any reference from segment A to segment B is **invalid** because there is no reference from the common segment to segment B. You can **validate** a reference from segment A to segment B by including, in the common segment, an external reference to the symbol used in the exclusive reference to segment B.

Another way to eliminate exclusive references is to arrange your program so that all references causing overlays are made in higher segments. For example, you could eliminate the exclusive reference shown in

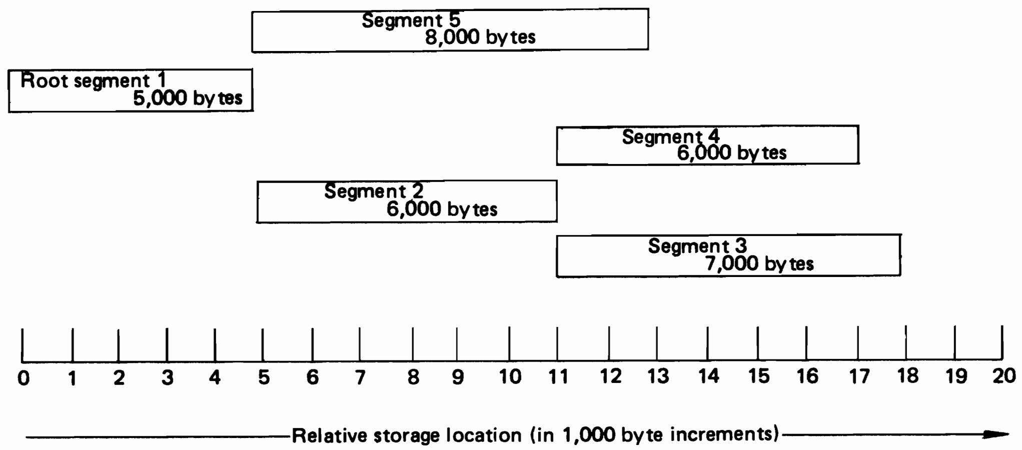


Fig. 6.4 Segment origin and use of storage

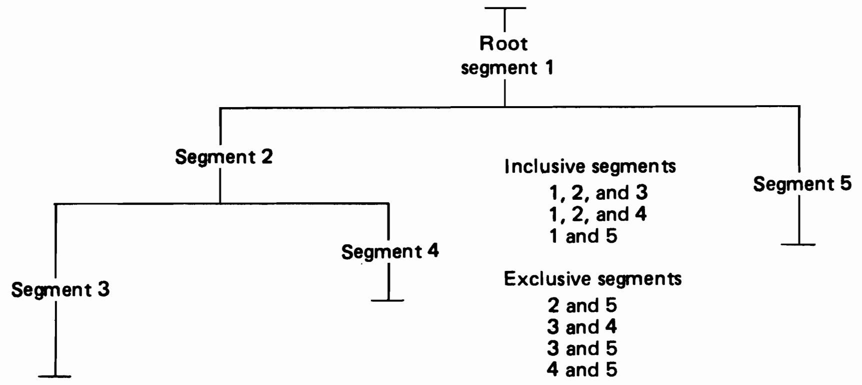


Fig. 6.5 Inclusive and exclusive segments

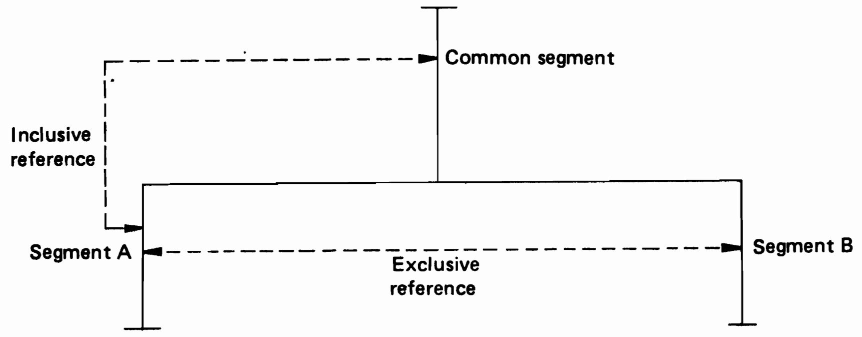


Fig. 6.6 Inclusive and exclusive references

Fig. 6.6 by creating a new module in the common segment; the new module's only function would be to reference segment B. You would then change the code in segment A to refer to the new module instead of to segment B. Control then would pass from segment A to the common segment, where the overlay of segment A by segment B would be initiated.

If any exclusive references appear in your program—valid or invalid—the Linkage Editor considers them errors unless you request one of the special options described later in this section.

Notes:

- During execution of a program written in a higher level language such as FORTRAN, COBOL, or PL/I, an exclusive call results in abnormal termination of the program if the requested segment attempts to return control directly to the invoking segment that has been overlaid.
- If a COBOL program includes a segment that contains a reference to a COBOL class test or TRANSFORM table, the segment containing the table must be either (1) in the root segment or (2) a segment that is higher in the same path than the segment containing the reference to the table.

Loading segments

The OS IV/F4 Supervisor can load segments without assistance from your program and in spite of the fact that linkage procedures in an overlay structure are identical to those in a simple structure.

V-type address constants are the key to loading an overlay structure. When the Linkage Editor encounters one or more V-type address constants undefined in the same segment, it creates a special table (ENTAB) at the end of the segment, as shown in Fig. 6.7. The Linkage Editor stores the address of the corresponding ENTAB entry into the field defined by the V-type address constant in the program. At the beginning of a root segment, the Linkage Editor creates a SEGTAB control block, used by the Supervisor to manage segment loading.

Fig. 6.7 shows a load module with its SEGTAB and an ENTAB entry for a downward reference. For each upward reference, the referenced entry point is guaranteed to be in virtual storage. Hence, the V-type address constant is resolved directly for upward references.

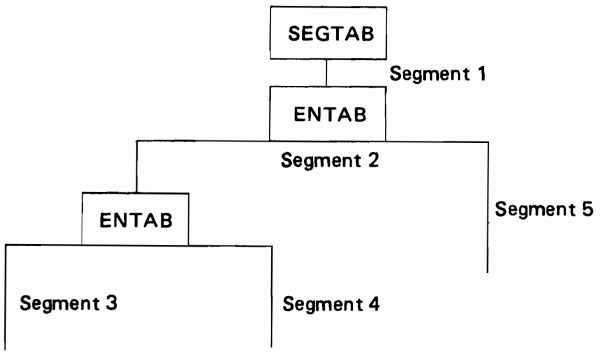


Fig. 6.7 Load module in an overlay structure

Fig. 6.8 illustrates the structure of an ENTAB and associated control flow in steps (a)–(g):

(a) As previously described, the Linkage Editor has stored the address of the ENTAB entry into the V-type address constant pointing to the external symbol of the other segment. Hence, control passes to ENTAB by branching to this address.

(b) If the requested segment has not been loaded, control passes to an Overlay-Supervisor call (SVC 45), since the current displacement points to this Overlay-Supervisor instruction in the same ENTAB.

(c) The Supervisor tests whether this segment is in memory. If not, the Supervisor loads this segment and increases the displacement in the corresponding ENTAB entry by two bytes. The displacement is increased only after the segment has been successfully loaded.

(d) On completion of step (c), the Supervisor passes control to the instruction following the SVC 45 instruction. This sequence loads the address of entry point KOBE into register 15 and branches to it. Since the overlay structure is within one load module, the address of this entry point was resolved by the Linkage Editor; it is not dynamically created during step (c).

(e) If segment N attempts to pass control to entry point KOBE of segment M, after loading segment M and until such time as a segment exclusive of M is subsequently loaded, control passes directly to KOBE without Supervisor intervention, since the corresponding displacement in ENTAB has been already changed.

(f) If segment N passes control to an entry point in segment L—which is exclusive of segment M—steps (a)–(d) are repeated to load segment L. At this time, if the displacement for KOBE remains unchanged, segment M cannot be loaded if control again passes to KOBE. Therefore, the ENTAB entry of the preceding segment is restored during loading of an exclusive segment.

(g) For an overlay structure, no ENTAB entry is required for upward references; direct branches always succeed. Therefore, in Fig. 6.7, segment 1 loads segments 5 and 3 at the same time. In an overlay structure, segments on the path of a given segment remain loaded including their entry points.

In the example of Fig. 6.8, you used a Branch and Link instruction to pass control. This method can be used only for inclusive references. For an exclusive reference, the calling segment has already been overlaid when it returns, so that control should be passed without return.

6.1.2 Multiple-Region Structures

If you use a control section in several segments, it is usually desirable to place that control section into the root segment. However, the root segment can get so large that the benefits of overlay are lost. If some control sections in your root segment could overlay each other (except for the requirement that all segments in a path must be in storage at the same time), your job may be appropriate for multiple-region structure. Multiple-region structures can also increase segment loading efficiency; processing can continue in one region while the next path to be executed is being concurrently loaded into another region.

With multiple regions, a segment has access to segments that are not in its path. Within each region, the rules for single-region overlay programs apply, but the regions are independent of each other. A maximum of four regions can be defined for one load module.

Fig. 6.9 shows the relationship between control sections in the sample program and two new control sections, CSH and CSI, each of which is used by two other control sections in different paths. Placing CSH

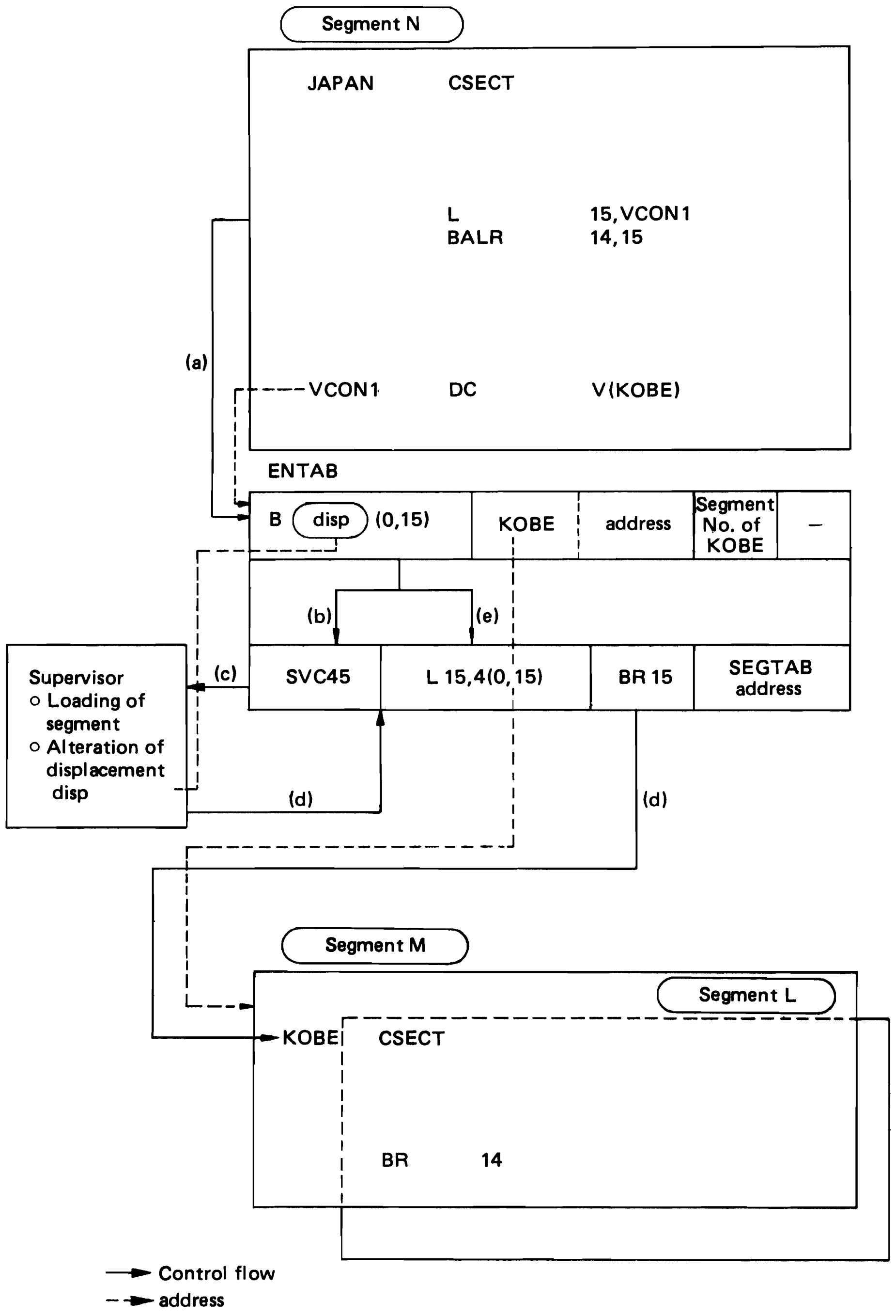


Fig. 6.8 Typical flow in an overlay structure

and CSI in the root segment makes it larger than necessary because CSH and CSI can overlay each other. These control sections should not be duplicated in both paths because the Linkage Editor automatically deletes the second pair, resulting in an invalid exclusive reference.

If you place the two control sections into another region, they can be in virtual storage when needed regardless of which path is executed in the first region. Fig. 6.10 shows all control sections in a two-region structure. Either path in region 2 can be in virtual 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.

The relative origin of a second region is determined by the length of the longest path in the first region (18,000 bytes). Region 2 begins at relative address 18,000. 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.

You determine the virtual storage required for your program by comparing the lengths of the longest path in each region. In Fig. 6.10, if CSH is 4,000 bytes and CSI is 3,000 bytes, the storage required is 22,000 bytes, plus the storage required by the SEGTAB and ENTAB tables described in the preceding section. You should be careful in planning multiple regions, since supervisory overhead may arise due to the Overlay Supervisor being unable to optimize segment loading.

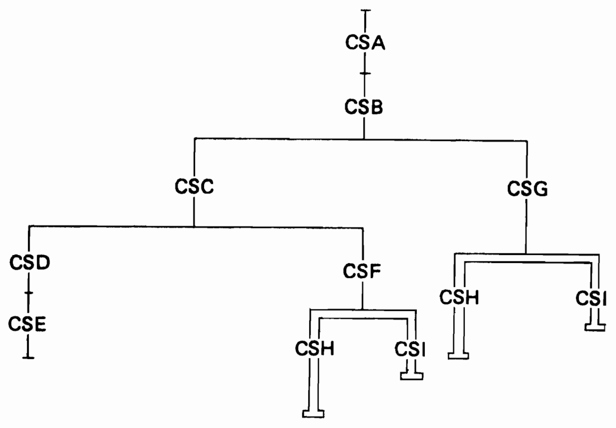


Fig. 6.9 Control sections used by several paths

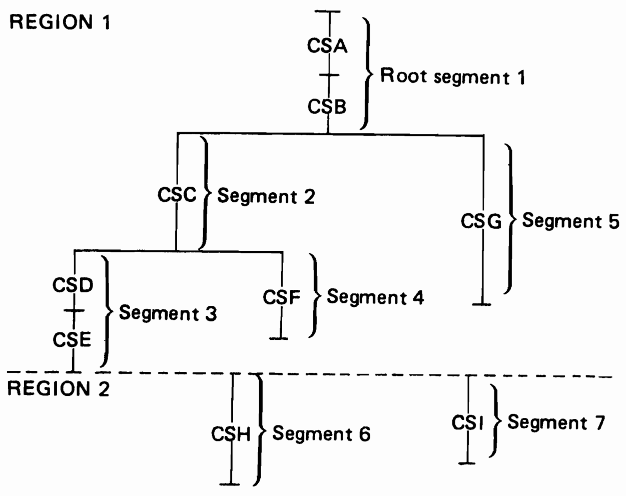


Fig. 6.10 Overlay tree for multiple-region program

6.2 SPECIFICATION

Once you have designed an overlay structure, you must indicate to the Linkage Editor the relative positions of segments and regions, and the control sections in each segment:

- **Segments** are positioned by OVERLAY statements. Since segments are not named, you identify a segment by giving its origin (or load point) a symbolic name and then using that name in an OVERLAY statement to specify a symbolic origin. Each OVERLAY statement begins a new segment.
- **Regions** are also positioned by OVERLAY statements. You specify the origin of the first segment of the region, followed by "(REGION)".
- **Control sections** are positioned in the segment specified by the OVERLAY statement with which they are associated in the input sequence. However, the sequece of control sectins within a segment is not necessarily the order in which the control sections are specified.

Your input sequence of control statements and control sections should reflect the sequence of segments in the overlay structure: top to bottom, left to right, region by region. This sequence is illustrated in later examples.

Note: If you wish to reprocess your overlay structure with the Linkage Editor, you must re-specify your OERLAY statements and EXEC-statement parameters such as OVLY. If you fail to provide these statements and options, your resulting load module will revert to a simple structure rather than remaining as an overlay structure.

6.2.1 Segment Origin

You must specify the symbolic origin of every segment other than the root with an OVERLAY statement. The first time a symbolic origin is specified, a **load point** is created at the end of the previous segment. That load point is logically assigned a relative address at the double-word boundary that follows the last byte in the preceding segment. Subsequent use of the same symbolic origin indicates that the next segment is to have its origin at the same load point.

In the sample single-region program, your OVERLAY statements assign symbolic origin names ONE and TWO to the two necessary load points, as shown in Fig. 6.10. Segments 2 and 5 are at load point ONE, segments 3 and 4 are at load point TWO.

The following sequence of OVERLAY statements will create the structure in Fig. 6.11 (the control sections in each segment are indicated by name):

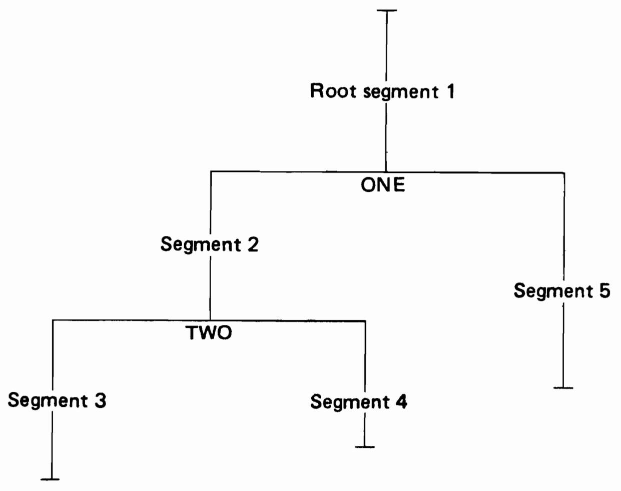


Fig. 6.11 Segment origins in a single-region structure

```
Control section CSA
Control section CSB
OVERLAY ONE
Control section CSC
OVERAY TWO
```

```
Control section CSD
Control section CSE
OVERLAY TWO
Control section CSF
OVERLAY ONE
Control section CSG
```

Note that the sequence of OVERLAY statements reflects the order of segments in the structure: top to bottom, and left to right.

6.2.2 Region Origin

You must specify the symbolic origin of every region other than the first with an OVERLAY statement. Once you have started a new region you may not reference a segment origin from a previous region.

In the sample multiple-region program, symbolic origin THREE is assigned to region 2, as shown in Fig. 6.12. Segments 6 and 7 are at load point THREE.

If you add the following OVERLAY statements to the sequence for the single-region program, you will create a multiple-region structure:

```
.
.
.
OVERLAY THREE(REGION)
Control section CSH
OVERLAY THREE
Control section CSI
```

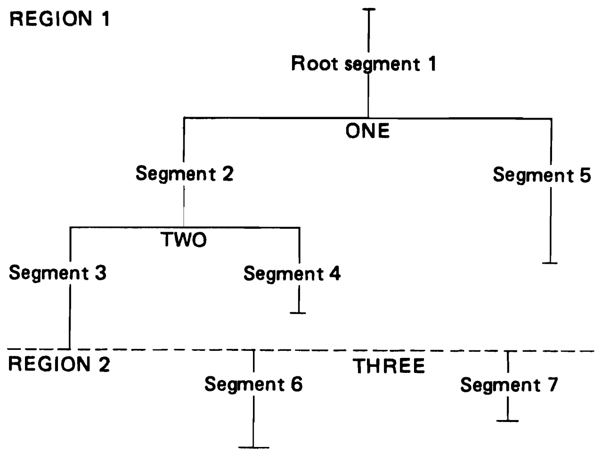


Fig. 6.12 Segment and region origins in a multiple-region structure

6.2.3 Positioning Control Sections

After each OVERLAY statement, you must specify the control sections for that segment in one of three ways:

- By placing object decks for each segment after the appropriate OVERLAY statement.
- By using INCLUD statements for modules containing the control sections for the segment.
- By using INSERT statements to reposition a control section from its position in the input stream to a particular segment.

The Linkage Editor places any control sections preceding the first OVERLAY statment into the root segment; they can be repositioned with an INSERT statement. Control sections from the automatic call library are also placed into the root segment. You can use INSERT statements to place these control sections into another specific segment. Common areas in an overlay program are described in Section 6.3.

Examples of the three methods for positioning control sections follow. Each example results in the structure for the single-region sample program. An example of repositioning control sections from the automatic call library follows the first three examples:

Using object decks
The primary input for this example contains an ENTRY statement and seven object decks, separated by OVERLAY statements:

```
//LKED          EXEC PGM=JQAL,
//                   PARM='OVLY'
                  .
                  .
                  .
//SYSLIN        DD  *
  ENTRY BEGIN
  Object deck for CSA
  Object deck for CSB
  OVERLAY ONE
  Object deck for CSC
  OVERLAY TWO
  Object deck for CSD
  Object deck for CSE
  OVERLAY TWO
  Object deck for CSF
  OVERLAY ONE
  Object deck for CSG
/*
```

You must specify the OVLY parameter on the EXEC statement for every overlay structure to be created by the Linkage Editor.

Using INCLUDE statements
The primary input for this example comprises a series of control statements. INCLUDE statements in this data set direct the Linkage Editor to library members that contain control sections for the program.

```
//LKED          EXEC PGM=JQAL,
//                   PARM='OVLY'
                  .
                  .
                  .
//MODLIB        DD   DSNAME=OBJLIB,
//                   DISP=(OLD,KEEP),...
//SYSLIN        DD   *
  ENTRY BEGIN
  INCLUDE MODLIB(CSA,CSB)
  OVERLAY ONE
  INCLUDE MODLIB(CSC)
  OVERLAY TWO
  INCLUDE MODLIB(CSD,CSE)
  OVERLAY TWO
  INCLUDE MODLIB(CSF)
  OVERLAY ONE
  INCLUDE MODLIB(CSG)
/*
```

This example differs from the previous one in that control sections of the program are not directly in the primary input. Instead, they are named in the primary input by INCLUDE statements. When the Linkage Editor processes an INCLUDE statement, it retrieves the appropriate control section(s) from the library and processes them.

Using INSERT statements

When you use INSERT statements, they and any OVERLAY statements may either follow or precede all input modules. However, the order of the control sections in a segment is not necessarily the same as the order of the INSERT statements for each segment. An example of each is given, as well as an example of repositioning automatically-called control sections.

Following All Input: You can furnish control statements following all the input modules, as shown in the following example:

```
//LKED          EXEC PGM=JQAL,
//                   PARM='OVLY'
                 .
                 .
                 .
//SYSLIN        DD   DSNAME=OBJECT,
//                   DISP=(OLD,KEEP),...
//              DD   *
  ENTRY BEGIN
  INSERT CSA,CSB
  OVERLAY ONE
  INSERT CSC
  OVERLAY TWO
  INSERT CSD,CSE
  OVERLAY TWO
  INSERT CSF
  OVERLAY ONE
  INSERT CSG
/*
```

The primary input data set comprises object modules containing the desired control sections concatenated to the input stream.

Preceding All Input: You can insert your control statements ahead of all input modules, as shown in the following example:

```
//LKED          EXEC PGM=JQAL,
//                   PARM='OVLY'
//MODULES       DD   DSNAME=OBJSEQ,
//                   DISP=(OLD,KEEP),...
                 .
                 .
                 .
//SYSLIN        DD   *
  ENTRY BEGIN
  INSERT CSA,CSB
  OVERLAY ONE
  INSERT CSC
  OVERLAY TWO
  INSERT CSD,CSE
  OVERLAY TWO
  INSERT CSF
  OVERLAY ONE
  INSERT CSG
  INCLUDE MODULES
/*
```

The primary input data set contains all control statements for the overlay structure and an INCLUDE statement. The sequential data set specified by the INCLUDE statement contains all object modules for the structure.

Repositioning Automatically-Called Control Sections: You can use INSERT statements to move automatically-called control sections from the root to the desired segment. This is helpful when control sections from the automatic call library are used in only one segment. By moving such control sections, you can ensure that the root will contain only control sections used by more than one segment.

When you write a program in a higher-level language, the Linkage Editor retrieves special control sections from the appropriate automatic call library. Assume your sample program is written in COBOL and that two control sections (ILBOVTR0 and ILBOSCH0) are called automatically from SYS1.COBLIB. Ordinarily, these control sections are placed into the root segment. However, you can furnish INSERT statements, as in the following example, to place these control sections outside the root segment.

```
//LKED          EXEC PGM=JQAL,
//                   PARM='OVLY'
//MODLIB        DD   DSNAME=OBJLIB,
//                   DISP=(OLD,KEEP),...
//SYSLIB        DD   DSNAME=SYS1.COBLIB,
//                   DISP=SHR
                 .
                 .
                 .
//SYSLIN        DD   *
  ENTRY BEGIN
  INCLUDE MODLIB(CSA, CSB)
  OVERLAY ONE
  INCLUDE MODLIB(CSC)
  OVERLAY TWO
  INCLUDE MODLIB(CSD, CSE)
  INSERT ILBOVTR0
  OVERLAY TWO
  INCLUDE MODLIB(CSF)
  INSERT ILBOSCH0
  OVERLAY ONE
  INCLUDE MODLIB(CSG)
/*
```

As a result, segments 3 and 4 will also contain ILBOVTR0 and ILBOSCH0, respectively.

This example shows two of the ways for specifying the control sections for a segment.

6.2.4 Special Options

The Linkage Editor provides you three special job-step options—OVLY, LET, and XCAL—to help specify overlay structures on your linkage-edit EXEC statement. You must specify these options each time you re-link your load module into an overlay structure.

OVLY option
You must specify the OVLY option for every overlay program. If you inadvertently omit "OVLY," all of your OVERLAY and INSERT statements are considered invalid. Your output module is marked "not executable" unless you also specify the LET option. Your output module is linked into a simple structure unless you specify the OVLY option.

LET option
If you specify the LET option, your output module is marked "executable" even though the Linkage Editor may detect certain error conditions. When you specify LET, any exclusive reference (valid or invalid) is accepted. At execution time, a valid exclusive reference is executed correctly; an invalid exclusive reference usually causes your program to terminate abnormally.

With the LET option, unresolved external references do not prevent your module from being marked "executable." This could be helpful when part of a large program is ready for testing; the segments to be tested may contain references to segments not yet coded. If you specify "LET", you can test those parts of your program that are finished, as long as the references to the absent segments are not executed. If you omit "LET", the Linkage Editor will detect these unresolved references and mark your module "not executable".

XCAL option
If you specify the XCAL option, the Linkage Editor does not consider valid exclusive calls as errors, and it marks your load module "executable". However, other errors could cause the module to be marked "not executable" unless you specify the LET option. In this case, "XCAL" need not be specified also.

6.3 OTHER OVERLAY CONSIDERATIONS

This section discusses several special considerations for overlay programs: handling of common areas, special storage requirements, and overlay communication.

6.3.1 Common Areas

When the Linkage Editor encounters common areas (blank or named) in an overlay program, it collects them as described previously (i.e., the largest blank or identically-named common area is used). The final location of each common area in your output module depends on whether you furnished INSERT statements to structure your program.

If you furnish INSERT statements, each named common area should either be (a) part of the input stream in the appropriate segment or (b) placed there with an INSERT statement.

You cannot use INSERT statements to reference blank common areas; a blank common area should always be part of the input stream in the appropriate segment.

If you do not use INSERT statements and if you place the control sections for each segment between OVERLAY statements, the Linkage Editor promotes the common area automatically—places it into the common segment of paths referencing it, so that the common area is in storage when needed. The position of the promoted area in relation to other control sections within the common segment is unpredictable.

If the Linkage Editor encounters a common area in a module from the automatic call library, it automatically promotes it to the root segment. In the case of a named common area, you can override this promotion by furnishing an INSERT statement.

Assume that your sample program is written in FORTRAN and that common areas are present as shown in Fig. 6.13. Further assume that you have structured your overlay program with INCLUDE statements between the OVERLAY statements, so that automatic promotion occurs.

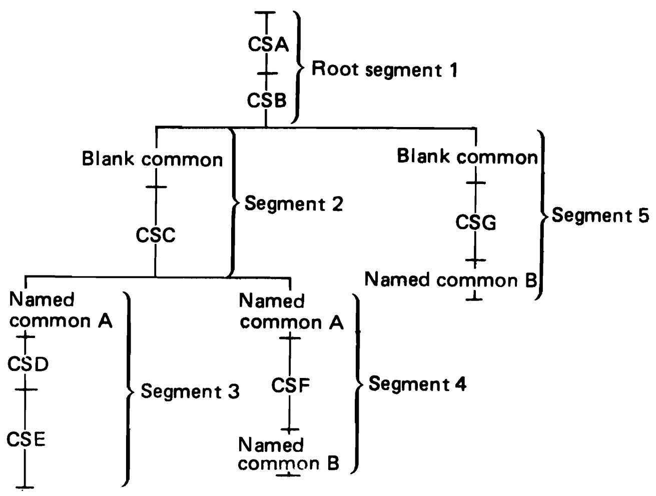


Fig. 6.13 Common areas before processing

Segments 2 and 5 contain blank common areas, segments 3 and 4 contain named common area "A", and segments 4 and 5 contain named common area "B". The Linkage Editor collects blank common areas and promotes the largest area to the root segment (the first common segment in the two paths).

The Linkage Editor collects the "A" common areas and promotes the largest to segment 2; the "B" common areas are collected and promoted to the root segment. Fig. 6.14 shows the location of the common areas after processing by the Linkage Editor.

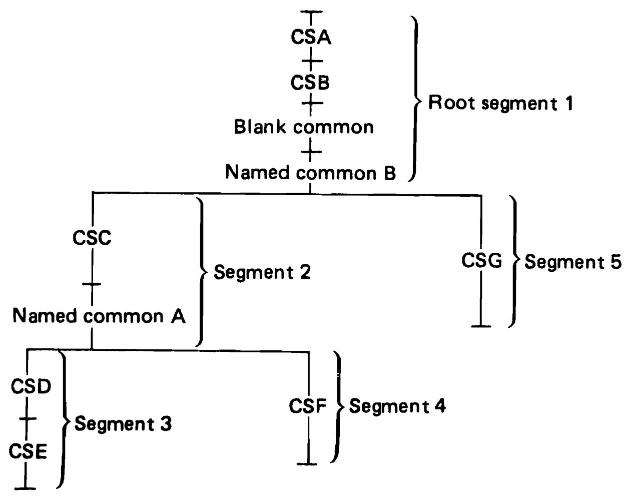


Fig. 6.14 Common areas after processing

6.3.2 Storage Requirements

Virtual storage requirements for your overlay program include tables added to your module by the Linkage Editor and the OS IV-F4 Overlay Supervisor necessary for execution. The former comprise the segment table (SEGTAB), entry tables (ENTABs), and other control information. Their size must be included in the minimum requirements for an overlay program, along with the storage required by the longest path and any control sections from the automatic call library.

Every overlay program contains one segment table in its root segment, whose storage requirements are:

SEGTAB=4n+24

where:

n=the number of segments in the program.

Some segments will contain entry tables. The requirements of entry tables for segments in the longest path must be added to the storage requirements for the program. The requirements for an entry table are:

ENTAB=12(x+1)

where:

x=the number of entries in the table.

Finally, the OS IV/F4 Supervisor creates a NOTE list prior to executing each overlay program, whose storage requirements are:

NOTELST=4n+8

where:

n=the number of segments in the program

To the minimum requirements of your load module, you must add the requirements of the Overlay Supervisor; during execution of your module, the OS IV/F4 Supervisor may request its Overlay Supervisor routine to initiate an overlay.

6.3.3 Inter-SegmentCommunications

You can use several techniques for communicating between segments of your overlay program. A higher-level or Assembler-language program ordinarily uses a CALL statement or macro instruction, respectively, to pass control to a symbol defined in another segment. The CALL causes the segment to be loaded if it is not already present in storage. An Assembler-language program may also use three additional ways to communicate between segments:

- By a Branch instruction, which causes a segment to be loaded and control to be passed to a symbol defined in that segment.
- 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.
- 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 loaded.

These two macro instructions are fully described in the **FACOM OS IV/F4 Supervisor Macro Instructions Reference Manual.**

You can only issue CALL or Branch instructions for exclusive references. You should **not** use either the SEGLD or SEGWT macro instruction to make an exclusive reference, since both imply that processing is to continue in the requesting segment, and an exclusive reference leads to erroneous results when the program is executed.

CALL statement or macro instruction

Your CALL statement or macro instruction refers to an external name in the segment to which you pass control. You must define this external name as an external reference in the requested segment. With the Assembler language, you must define the name as a four-byte V-type address constant, reserving the high-order byte for exclusive usage by OS IV/F4 Supervisor.

When you issue a CALL, the OS IV/F4 Supervisor loads the requested segment and any segments in its path if they are already in virtual storage. After the Supervisor loads this segment, it passes control to it at the location specified by the external name.

A CALL between inclusive segments is always valid. You can return to the requesting segment by another source language statement such as RETURN. A CALL between exclusive segments is valid if conditions for a valid exclusive reference are met; you can return from the requested segment only via another exclusive reference, because your requesting segment has been overlaid.

Branch instruction

You can use any of the branching conventions shown in Table 6.1 to request loading and branching to a segment. The OS IV/F4 Supervisor loads the requested segment and any segments in its path if they are not already in virtual storage. The Supervisor then passes control to the requested segment at the location specified by the address constant in general register 15.

Table 6.1 Branch sequences for overlay programs

Variation	Coding		
	Name[1]	Operation	Operand[2,3]
1		L BALR	R15,=V(name) Rn,R15
2	 ADCON	L BALR : DC	R15,ADCON Rn,R15 V(name)
3		L BAL	R15,=V(name) Rn,0(0,R15)[4]
4		L BAL	R15,=V(name) Rn,0(R15)[5]
5[6]		L BCR	R15,=V(name) 15,R15
6[6]		L BC	R15,=V(name) 15,0(0,R15)[4]
7[6]		L BC	R15,=V(name) 15,0(R15)[5]

[1] When the name field is blank, specification of a name is optional.
[2] R15 is the register into which is loaded a 4-byte address constant that is an entry name or a control section name in the requested segment. The address constant must be loaded into register 15.
[3] Rn is any other register and is used to hold the return address. This register is conventionally register 14.
[4] This may also be written so that the index register contains the address; the other fields must be zero.
[5] The base register must contain the address; the displacement must be zero.
[6] Conditional branches are also allowed.

Your address constant must be a 4-byte V-type address constant. You must reserve its high-order byte for use by the OS IV/F4 Supervisor.

Branching between inclusive segments is always valid; you can return via the address in Rn. Branching between exclusive segments is valid if the conditions for a valid exclusive reference are met; you can return only via another exclusive reference.

Segment load (SEGLD) macro instruction
You issue a SEGLD macro instruction to provide overlap between segment loading and processing within your requesting segment. With any of the coding sequences in Table 6.2, you initiate loading of the requested segment (and any segments in its path) if they are not already in virtual storage. Processing then resumes at the next sequential instruction in your requesting segment while the new segment(s) are being loaded. You can then pass control to the requested segment with a CALL or Branch instruction, as shown in variations 1 and 2, respectively. You can issue a SEGWT macro instruction to ensure that the specified control section is in virtual storage before processing begins, as shown in Variation 3.

Table 6.2 Use of the SEGLD macro instruction

Variation	Coding		
	Name[1]	Operation	Operand[2,3]
1		SEGLD CALL	external name external name
2		SEGLD branch	external name
3		SEGLD SEGWT L	external name external name Rn,=A(name)

[1] When the name field is blank, specification of a name is optional.
[2] External name is an entry name or a control section name in the requested segment.
[3] Rn is any other register and is used to hold the return address. This register is conventionally register 14.

You must define any external name in your SEGLD macro instruction with a 4-byte V-type address constant. You must reserve its high-order byte for use by the OS IV/F4 Supervisor.

Segment wait (SEGWT) macro instruction
You issue a SEGWT macro instruction to stop processing in the requesting segment until your requested segment has been loaded. For any of the variations in Table 6.3, no further processing takes place until the requested segment (and all segments in its path) are loaded. Processing resumes at the next sequential instruction in the requesting segment after your segment has been loaded.

Table 6.3 Use of the SEGWT macro instruction

Variation	Coding		
	Name[1]	Operation	Operand[2,3]
1	 ADCON	SEGLD SEGWT L branch DC	external name external name Rn,ADCON A(name)
2		SEGWT L	external name Rn,=A(name)

[1] When the name field is blank, specification of a name is optional.
[2] External name is an entry name or a control section name in the requested segment.
[3] Rn is any other register and is used to hold the return address. This register is usually register 14.

If you use SEGWT and SEGLE macro instructions together, OS IV/F4 overlaps processing and segment loading; your SEGWT macro instruction checks whether or not necessary information is in storage when needed (see Example 1 in Fig. 6.17). In Example 2 of Fig. 6.17, no overlap occurs; your SEGWT macro instruction initiates loading, but processing halts in the requesting segment until the requested segment is in virtual storage.

You must define the external name in your SEGWT macro instruction with a 4-byte V-type address constant. You must reserve its high-order byte for use by the OS IV/F4 Supervisor. If you wish to reference a virtual storage location in the requested segment you must define an A-type address constant for the entry name of this location.

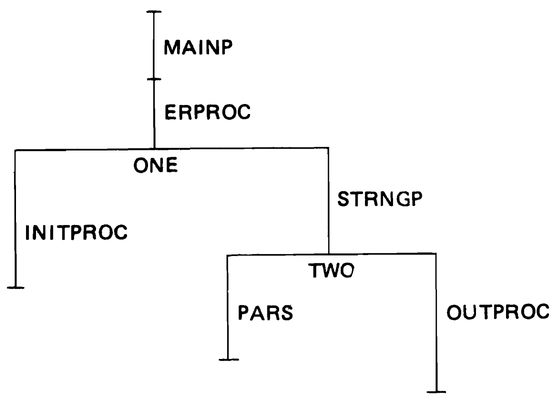


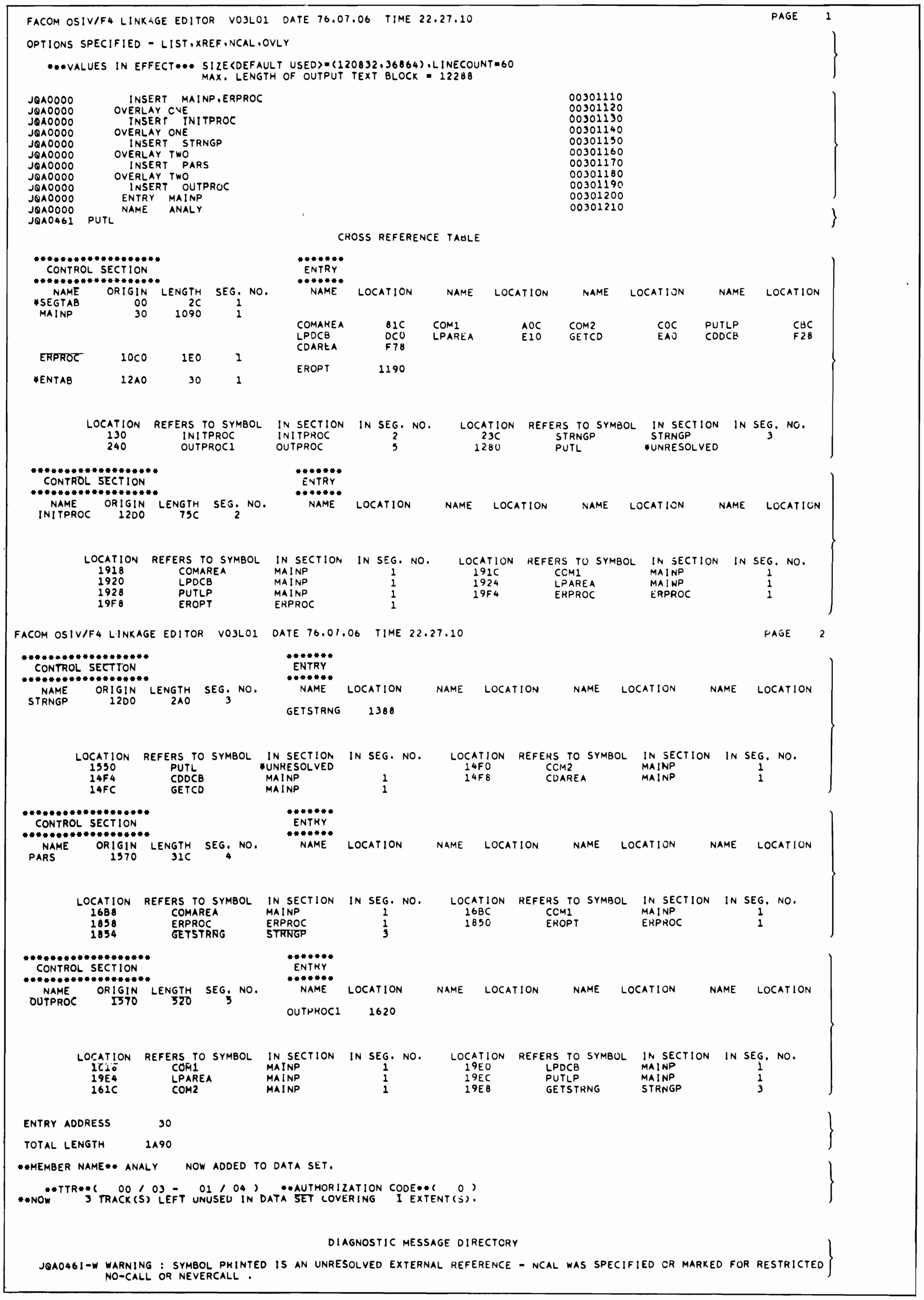
Fig. 6.15 Structure of sample overlay program

6.4 A SAMPLE OVERLAY PROGRAM

The following Assembler-language program is to be linked into an overlay structure comprising 5 segments constructed from 6 control sections: MAINP, ERPROC, INITPROC, STRNGP, PARS, and OUTPROC. The overlay structure is diagrammed in Fig. 6.15, and corresponding JCL and Linkage Editor control statements appear in Fig. 6.16. Fig. 6.17 shows the corresponding SYSPRINT output.

```
//LWK            EXEC    PGM=JQAL,
//                       PARM='LIST,XREF,
                         NCAL,OVLY'
//SYSPRINT       DD      SYSOUT=A
//SYSLMOD        DD      DSN=USER,LOADMOD,
//                       SPACE=(TRK,(100,
                         10,1)),UNIT=SYSDA,
//                       DISP=(MOD,PASS),
//                       VOL=SER=USER12
//SYSUT1         DD      SPACE=(TRK,(100,
                         10)),UNIT=SYSDA
//SYSLIN         DD      DSN=&&OBJ,
//                       DISP=(OLD,DELETE)
//               DD      *
               INSERT       MAIN,ERPROC
             OVERLAY      ONE
               INSERT       INITPROC
             OVERLAY      ONE
               INSERT       STRNGP
             OVERLAY      TWO
               INSERT       PARS
             OVERLAY      TWO
               INSERT       OUTPROC
             ENTRY        MAINP
             NAME         ANALY
/*
```

Fig. 6.16 JCL and Linkage Editor control statements for sample program



```
FACOM OSIV/F4 LINKAGE EDITOR  V03L01  DATE 76.07.06  TIME 22.27.10                                        PAGE    1

OPTIONS SPECIFIED - LIST,XREF,NCAL,OVLY

   ***VALUES IN EFFECT*** SIZE<DEFAULT USED>=(120832,36864),LINECOUNT=60
                          MAX. LENGTH OF OUTPUT TEXT BLOCK = 12288

JQA0000        INSERT  MAINP,ERPROC                                                 00301110
JQA0000      OVERLAY CNE                                                            00301120
JQA0000        INSERT  INITPROC                                                     00301130
JQA0000      OVERLAY ONE                                                            00301140
JQA0000        INSERT  STRNGP                                                       00301150
JQA0000      OVERLAY TWO                                                            00301160
JQA0000        INSERT  PARS                                                         00301170
JQA0000      OVERLAY TWO                                                            00301180
JQA0000        INSERT  OUTPROC                                                      00301190
JQA0000       ENTRY  MAINP                                                          00301200
JQA0000       NAME   ANALY                                                          00301210
JQA0461  PUTL
                                               CROSS REFERENCE TABLE

 *******************                      *******
  CONTROL SECTION                          ENTRY
 *******************                      *******
   NAME     ORIGIN  LENGTH  SEG. NO.        NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
 *SEGTAB       00      2C      1
  MAINP        30    1090      1
                                          COMAREA       81C    COM1         A0C    COM2         C0C    PUTLP         CBC
                                          LPDCB         DC0    LPAREA       E10    GETCD        EA0    CDDCB         F28
                                          CDAREA        F78
  ERPROC     10C0     1E0      1
                                          EROPT        1190
 *ENTAB      12A0      30      1


        LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.    LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.
          130         INITPROC      INITPROC        2             23C        STRNGP        STRNGP           3
          240         OUTPROC1      OUTPROC         5            1280        PUTL         *UNRESOLVED

 *******************                      *******
  CONTROL SECTION                          ENTRY
 *******************                      *******
   NAME     ORIGIN  LENGTH  SEG. NO.        NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
  INITPROC   12D0     75C      2


        LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.    LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.
          1918        COMAREA       MAINP           1            191C        CCM1          MAINP            1
          1920        LPDCB         MAINP           1            1924        LPAREA        MAINP            1
          1928        PUTLP         MAINP           1            19F4        ERPROC        ERPROC           1
          19F8        EROPT         ERPROC          1
FACOM OSIV/F4 LINKAGE EDITOR  V03L01  DATE 76.07.06  TIME 22.27.10                                        PAGE    2

 *******************                      *******
  CONTROL SECTION                          ENTRY
 *******************                      *******
   NAME     ORIGIN  LENGTH  SEG. NO.        NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
  STRNGP     12D0     2A0      3
                                          GETSTRNG     1388


        LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.    LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.
          1550        PUTL         *UNRESOLVED                   14F0        CCM2          MAINP            1
          14F4        CDDCB         MAINP           1            14F8        CDAREA        MAINP            1
          14FC        GETCD         MAINP           1

 *******************                      *******
  CONTROL SECTION                          ENTRY
 *******************                      *******
   NAME     ORIGIN  LENGTH  SEG. NO.        NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
  PARS       1570     31C      4


        LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.    LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.
          16B8        COMAREA       MAINP           1            16BC        CCM1          MAINP            1
          1858        ERPROC        ERPROC          1            1850        EROPT         ERPROC           1
          1854        GETSTRNG      STRNGP          3

 *******************                      *******
  CONTROL SECTION                          ENTRY
 *******************                      *******
   NAME     ORIGIN  LENGTH  SEG. NO.        NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
  OUTPROC    1570     520      5
                                          OUTPROC1     1620


        LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.    LOCATION  REFERS TO SYMBOL  IN SECTION  IN SEG. NO.
          1C16        COM1          MAINP           1            19E0        LPDCB         MAINP            1
          19E4        LPAREA        MAINP           1            19EC        PUTLP         MAINP            1
          161C        COM2          MAINP           1            19E8        GETSTRNG      STRNGP           3

 ENTRY ADDRESS        30

 TOTAL LENGTH       1A90

**MEMBER NAME** ANALY     NOW ADDED TO DATA SET.

    **TTR**(    00 / 03 -    01 / 04 )    **AUTHORIZATION CODE**(    0 )
**NOW      3 TRACK(S) LEFT UNUSED IN DATA SET COVERING    1 EXTENT(S).


                                          DIAGNOSTIC MESSAGE DIRECTORY

   JQA0461-W WARNING : SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE - NCAL WAS SPECIFIED OR MARKED FOR RESTRICTED
             NO-CALL OR NEVERCALL .
```

Fig. 6.17 SYSPRINT data set for sample program

CHAPTER 7 DYNAMIC LINK STRUCTURES AND PROTOTYPE CONTROL SECTIONS

7.1 OVERVIEW OF DYNAMIC LINKING

A dynamic link structure somewhat resembles both overlay and dynamic program structures, the latter utilizing OS IV/F4 Supervisor series to link (LINK), load (LOAD), or transfer control (XCTL) among various subprograms. The principal objective of the novel and powerful OS IV/F4 dynamic linking facility is to help you test large programs (a) consisting of many subprograms and (b) ultimately to become large simple-structured or overlay-structured production programs.

During testing of a new program, you can more conveniently manage it as a collection of separately-linked load modules rather than linking it into a single large module. Whenever you uncover an error in a large simple (or overlay) structure, you must not only correct and re-compile the erroneous object module but also link-edit the entire load module, including many error free subprograms. You can reduce your link-editing effort, time, and consequent exposure to mechanical error if you segregate each major sub-program—or substantial collection of smaller subprograms—into a separate load module. Without the dynamic linking facility, more test time has often been required for link-editing of programs than for the tests themselves. However, in a dynamic program structure, the Supervisor calls most major sub-programs, imposing heavy supervisory overheads.

To resolve these strategic problems, OS IV/F4 furnishes the dynamic link structure, which uses CALL macro instructions (containing Branch linkages) most of the time to link subprograms. If a specified entry point is in the same load module as its callers, or if another load module with this entry point has already been loaded, the Branch instruction is direct, i.e., without assistance from the OS IV/F4 Supervisor. If the requested sub-program is not currently in virtual storage, the dynamic link structure yields control to the Supervisor; after the latter has loaded your program automatically, it branches to your program. You need never know whether the entry point of a called program is located in the same load module as its caller, or not.

The dynamic link structure is fundamentally different from the dynamic program structure. A dynamic link structure comprises a group of sub-programs, each of which has a simple structure; after initial loading, these sub-programs are not deleted from virtual storage even if their processing has been completed. A dynamic link structure transfers control by a Branch instruction no later than the second CALL to each entry point. Therefore, it lacks the virtual storage flexibility of the dynamic program structure, which adapts to the capacity of a given virtual storage region.

The dynamic link structure is designed primarily for program testing; its advantages are its simplicity and reduced overhead for relinking programs while the latter are being frequently changed.

7.2 PASSING CONTROL IN A DYNAMIC LINK STRUCTURE

Procedures for passing control in a dynamic link structure are identical to those for a simple structure: CALL, SAVE and RETURN macro instructions, or equivalent machine instructions. In this section, the discussion concentrates on how you load modules in a dynamic link structure, plus some restrictions on using the structure.

To create a dynamic link structure, you must specify external subroutines by V-type address constants. Such a constant is automatically prepared by the OS IV/F4 Assembler and the major compilers for every CALL you issue. Alternatively, you can write your own subroutine linkages and associated V-type constants. You must also link-edit your structure specifying PARM=DYNA on your Linkage-Editor EXEC statement.

If one or more V-type address constants cannot be resolved within your load module, the Linkage Editor creates a Dynamic Address Linking Table (DALTAB) at its end as shown in Fig. 7.1. This example will be used throughout the explanation of loading and passing control in a dynamic link structure.

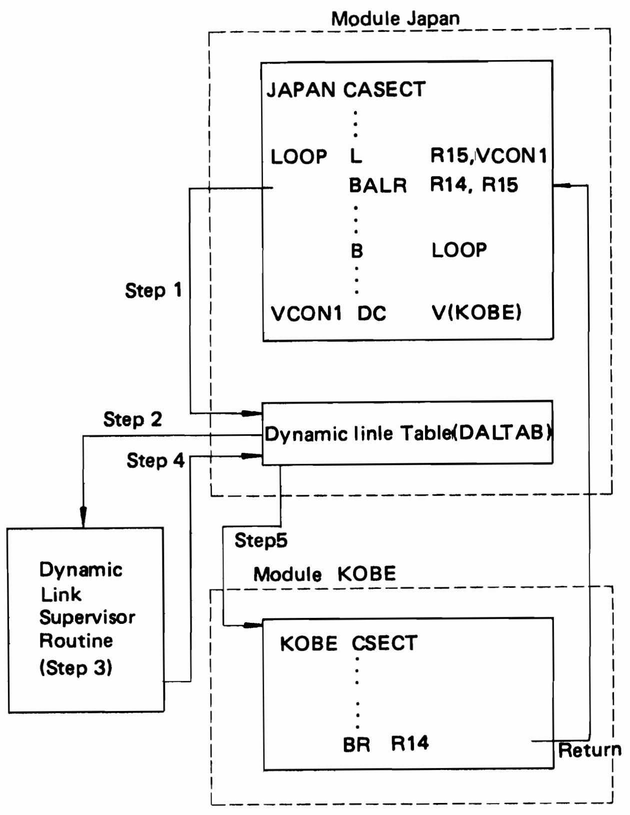


Figure 7-1 Example of dynamic linkage

Step 1 Within the JAPAN load module, a call is made to the independently-compiled KOBE routine. The CALL macro instruction generates two machine instructions:

L 15, VCON1
BALR 14, 15

and the VCON1 address constant naming "KOBE". This load module is link-edited without the KOBE routine; instead, the DYNA parameter instructs the Linkage Editor to insert the address of DALTAB into VCON1 (and any other unresolved V-type address constants).

Step 2 The first time KOBE is called, its DALTAB entry points to an "SVC45" instruction. This instruction passes control to the dynamic-link supervisory routine.

Step 3 The Supervisor tests whether the KOBE load module was requested from the name entry. It also tests whether KOBE has already been dynamically linked by another CALL macro instruction. If KOBE has not yet been loaded, the Supervisor (a) retrieves it from the appropriate step, job, or link library, (b) stores its loaded address into the second word of its name entry in DALTAB, and (c) adds "2" to the displacement of the first word in its entry. Step (c) causes the Branch instruction at the beginning of the name entry to point to the following two-instruction sequence:

L 15, 4 (0, 15)
BR 15

Step 4 After completing step 3, the Supervisor returns control to this two-instruction sequence in DALTAB, after restoring all registers (including register 15) to their values at the time the programmers called KOBE. This return passes control to KOBE just as if you had link-edited it into the JAPAN load module.

Step 5 Thereafter, each linkage to KOBE from JAPAN will be direct, just as if you had link-edited them together. In some respects, a dynamic link structure resembles a dynamic program structure; in other respects, it resembles an overlay structure.

7.3 RESTRICTIONS ON DYNAMIC LINKING

In either a dynamic link structure or dynamic program structure, subprograms are separately link-edited and loaded on demand. However, the subprograms of a dynamic link structure must themselves be simple, overlay, or dynamic program structures; they cannot be dynamic link structures. Fig. 7.2 shows the complete flow of two relatively complex dynamic link structures.

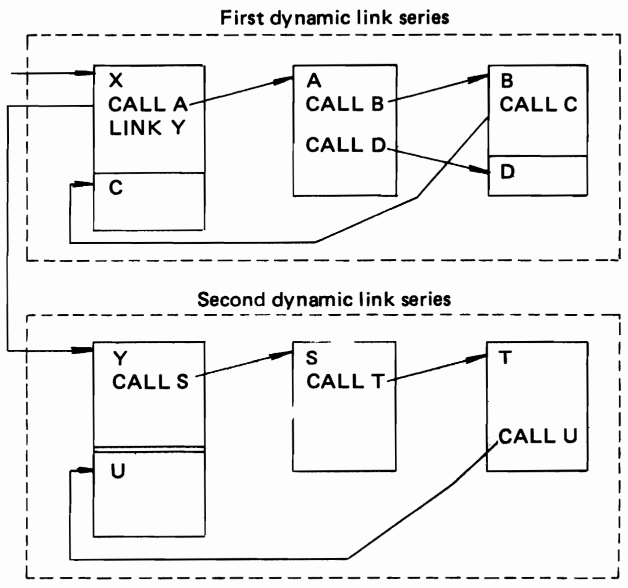


Fig. 7.2 Dynamically linking several subprograms

7.3.1 Dynamic Link Series

A dynamic link series is a collection of load modules among which control flows in a dynamic link structure. Fig. 7.2 contains two series, which you should handle as a simple structure with respect to reenterability and serial reusability.

7.3.2 Usability Attributes of Dynamic Link Structures

Whatever usability attributes its component subprograms may have, a dynamic link structure must manage these subprograms directly and completely. You manage the first series in Fig. 7.2 as a simple structure: subprograms are called directly by one another, and thus load modules A and B are nonreusable.

Regardless of how many different calls—and their frequencies—are made to entry points B and D, module B is loaded only once for the first dynamic link series. Hence, attributes of its modules can be summarized as followed:

- If nonreusable: each module of a dynamic link series is either never loaded (if never requested) or loaded once.
- If reusable: irrespective of whether you invoke a reusable load module within a dynamic link structure or not, one copy is loaded into the job pack area. If you make several calls concurrently to a serially-reusable module from a dynamic link series, the task is likely to fail.

7.3.3 Deleting Load Modules

No load module in a dynamic link series is deleted until the entire series is deleted. All load modules in the second series of Fig. 7.2 are deleted when the original load module ("Y") returns control to its caller ("X") via the Supervisor. Likewise, the first series is deleted when it returns control to the Supervisor. However, reusable load modules are not deleted from your address space until/unless the latter becomes exhausted; in this respect, dynamically linked modules are managed differently than those accessed via LINK or XCTL macro instructions.

In Fig. 7.2, you could call Y from X by issuing a LOAD macro instruction followed by a CALL, rather than issuing a LINK macro instruction. In this case, the second dynamic link series would not be distinct from the first series; load modules S and T would be incorporated into the first series from an OS IV/F4 standpoint. When Y returned control to X, only load-module Y would be deleted, and S and T would remain loaded. These latter two modules would be deleted from the dynamic link series only when X returns control to the Supervisor.

7.3.4 Program Libraries

A dynamic link structure cannot utilize a private library; every load module it retrieves must be in the step, job, or link libraries.

7.4 CREATING REENTERABLE PROGRAMS WITH PSECTS

As will be discussed in Chapter 8, a reenterable program does not modify its instructions or data areas during execution. To create reenterable programs, you must acquire data areas and work areas dynamically during execution by issuing GETMAIN macro instructions. When finished with these areas, you must accurately release them — whatever storage areas you acquire via GETMAIN macro instructions must be identified precisely in your corresponding FREEMAIN macro instructions with respect to locations and sizes.

If your reenterable program needs to initialize certain data areas and parameters, you must issue machine instructions—explicitly or implicitly—to set these initial values. To facilitiate initializing reenterable programs, OS IV/F4 provides the prototype control section (PSECT) facility, which permits you to define a pre-initialized work area when assembling or compiling the program.

You must notify the Linkage Editor that you are creating a reenterable program by specifying PARM=RENT on your EXEC statement.

7.4.1 Overview of PSECTs

PSECTs were developed primarily to support COBOL, FORTRAN, and PL/I programs, but you may also use them in Assembler language programs. The following discussion uses Assembler language terminology and examples.

Definition
Each reenterable program comprises two or more control sections. You should define all unmodified instructions and constants (such as Assembler literals) in one or more ordinary control sections (CSECTs); modifiable instructions and work areas should be defined in one or more PSECTS. All necessary facilities of the OS IV/F4 Assembler Language can be defined in PSECTs.

Loading and deleting
In any OS IV/F4 load module, the Linkage Editor segregates any PSECTs from all CSECTs. During execution of your program, each time you ask OS IV/F4 to load a module (by a LINK, LOAD, etc. macro instruction), the Supervisor loads a fresh copy of its PSECTs. Therefore, your job pack area may

contain one or more reenterant CSECTs and as many PSECTs as the current number of subtasks using this load module. Your link pack area will contain only the reentrant CSECTs; PSECTs are automatically allocated from your region as you request LPA load modules. Each PSECT can address the corresponding CSECT via V-type and EXTRN address constants, but it is clearly impossible for a reenterable CSECT to address locations in a PSECT (since several PSECTs may be simultaneously active with this CSECT).

PSECTs can only be used once. Hence, whenever a DELETE macro instruction is issued to a load module containing one or more PSECTs, OS IV/F4 deletes the PSECTs and frees their virtual-storage areas.

```
         CSECT
STRTPROC DS       OH
         USING    DATA,7        BASE REG FOR DATA
         USING    STRTPROC,12   BASE REG FOR PROG
                  :
         L        3,VALUE       EXAMPLE OF ACCESS
                  :
         PSECT
         ENTRY    EPA1
EPA1     DS       OH
         USING    EPA1,15
         SVG      14,12,SAVEPT  SAVE CALLER'S
                                REGS
                  :
         L        12,VPROC      OBTAIN PROCEDURE
                                ADDRESS
         LA       7,DATA        OBTAIN DATA
                                ADDRESS
         BR       12
VPROC    DC       V(STRTPROC)
DATA     DS       OF
SAVEPT   DC       A(REGSAVE)
REGSAVE  DC       18F'0'
VALUE    DC       CL8'OS4/F4'
```

Fig. 7.3 Example of A PSECT

Entry points
For a CSECT to address data in its PSECT, your CSECT routine must furnish the address of its PSECT in a special base register. Hence, each PSECT must furnish an entry point (ENTRY attribute for a linkage instruction) and establish its own addressability before branching to the corresponding CSECT, as shown in Fig. 7.3.

7.4.2 PSECTs in a Dynamic Link Structure

It is often advantageous to define one or more PSECTs in a dynamic link structure, since PSECTs permit efficient testing of reentrant programs, as described bbelow.

Loading a PSECT
In a dynamic program structure, the OS IV/F4 Supervisor loads a PSECT each time you invoke the corresponding load module. Since a dynamic link structure is treated as one load module by OS IV/F4, any PSECT it contains is loaded only once.

Calling a sub-program
When a dynamic link structure contains a PSECT, the latter must contain a V-type address constant pointing to the entry point of the corresponding CSECT, as shown in Fig. 7.4. With this V-type constant, the Linkage Editor creates a DALTAB in this PSECT, as defined in Section 7.2. During your second and subsequent calls to this PSECT, control flows directly via a Branch instruction. In the dynamic link structure defined in Fig. 7.4, this Branch instruction operates correctly so long as the calling procedure contains a V-type constant pointing to an entry point in the PSECT.

Specifying appropriate linkage editor options
When you wish to link-edit a reentrant program with dynamic link structure, you must specify "PARM=(RENT, DYNA)" on your EXEC statement.

```
            Calling program           |             Called program
--------------------------------------+---------------------------------------
          CSECT                       |             CSECT
MPROC     DS        OH                |  SUBP       DS        OH
          USING     MPROC,12          |             USING     *,8
          USING     DATA,7            |             USING     SUBD,10
          :                           |             :
          L         15,VSUBE          |
          BALR      14,15             |
          :                           |             RETURN
                    PSECT             |             PSECT
                    ENTRY             |  MEPA       ENTRY     SUBRTINE
MEPA      DS        OH                |  SUBRTNE    DS        OH
          USING     MEPA,15           |             USING     *,15
          SVG       14,12,SVPOINT     |             SVG       14,12,SUBSVP

          L         12,VPROC          |             L         8,VSUBP
          LA        7,DATA            |             LA        10,SUBD
          BR        12                |             BR        8
          :                           |             :
VPROC     DC        V(MPROC)          |  VSUBP      DC        V(SUBP)
VSUBE     DC        V(SUBRTNE)        |  SUBD       DS        OF
DATA      DS        OF                |  SUBSVP     DC        A(SUBSAVE)
SVPOINT   DC        A(SVAREA)         |  SUBSAVE    DC        18F'0'
SVAREA    DC        18F'0'            |
          :         Data              |             :         Data
                    area              |                       area
```

Fig. 7.4 Example of dynamically linking PSECTs

CHAPTER 8
JOB CONTROL LANGUAGE FOR THE LINKAGE EDITOR

This chapter summarizes aspects of the OS IV/F4 Job Control Language (JCL) that pertain directly to the Linkage Editor: the EXEC statement, DD statements, and cataloged procedures for the Linkage Editor. In addition, Section 8.4 describes how you can invoke the Linkage Editor from another Assembler-language program. You should already be familiar with JCL, as described in the **FACOM OS IV/F4 Job Control Language Reference Manual** or **FACOM OS IV/F4 Job Control Language User's Guide**.

8.1 INTRODUCTION TO THE EXEC STATEMENT

The EXEC statement is the first statement of every job step. For linkage editing, the following topics are pertinent:

- Program name of the Linkage Editor: JQAL (alias LINKEDIT)
- Linkage Editor options
- Region size for the Linkage Editor
- Linkage Editor return codes

Either of the following EXEC statements cause the Linkage Editor to be invoked:

```
//LKED        EXEC    PGM=JQAL
//LKED        EXEC    PGM=LINKEDIT
```

8.2 JOB STEP OPTIONS

Your EXEC statement also contains a list of options or parameters you can pass to the Linkage Editor:

- Module attributes, which describe the characteristics of the output load module.
- Special processing options, to direct Linkage Editor processing.
- Space allocation options, which affect the amount of storage used by the Linkage Editor for processing and output-module buffers.
- Output options, which specify the kind of printed outputs the Linkage Editor is to produce.

The rest of this section describes the options in each category. All options for a particular execution are listed in the PARM parameter on the EXEC statement, in any sequence so long as the corresponding coding rules are followed.

8.2.1 Module Attributes: AC, ALIGN2, DYNA, NE, OL, OVLY, REFR, RENT, REUS, TEST

Your module attributes describe various characteristics applicable to all load modules you edit in one job step. The Linkage Editor stores these attributes in the directory entry for each module, along with its member name and any aliases.

Module attributes are as follows:

AC	Authorization code for access to security-restricted system programs and data sets
ALIGN2	Page-boundary alignment of specified control sections
DYNA	Dynamic link structure for the load module(s)
NE	Not editable by subsequent Linkage Editor jobs
OL	Only loadable by the OS IV/F4 Supervisor; the module(s) cannot be attached, linked, etc.
OVLY	Overlay structure for the load module(s)
REFR	Refreshable by the OS IV/F4 Recovery Management Supervisor, should they be damaged during execution.
RENT	Reenterable by OS IV/F4 Job Management, should two or more subtasks simultaneously require the same module.
REUS	Serially reusable by two or more subtasks
TEST	Testable format for usage from a TSS terminal

Authorization code (AC)
You can assign to a load module an **authorization code**, which determines whether or not it can use restricted system services and resources.

To assign an authorization code through the PARM field, code the AC parameter as follows:

```
//LKED          EXEC    PGM=JQAL,
//                      PARM='AC=n,...'
```

The authorization code **n** must be a decimal integer less than 256.

"AC=", "AC=,..." and "AC=" are equivalent to "AC=0". The authorization code you assign in the PARM field is overridden by any authorization code you assign with a SETCODE control statement.

Page boundary attribute (ALIGN2)
Control sections within a load module with the page boundary attribute are aligned in storage on page boundaries (i.e., address 0 of each such control section is assigned an address which is an integral multiple of 2048 or 4096). Used with the PAGE control statement or the ORDER statement with the P operand, this attribute requests the Linkage Editor to align specified control sections on 2K boundaries.

To assign the 2K page boundary attribute, code ALIGN2 in the PARM field, as follows:

```
//LKED          EXEC    PGM=JQAL,
//                      PARM='ALIGN2,...'
```

Note: If you omit the ALIGN2 attribute but furnish one or more PAGE statements or ORDER statements with P operands, the Linkage Editor assigns 4K boundaries to the specified control sections.

Dynamic link (DYNA) structure
You create a dynamic link structure by specifying the following EXEC statement:

```
//LKED          EXEC    PGM=JQAL,
//                      PARM='DYNA,...'
```

Any external references remaining unresolved after your (optional) usage of Automatic Library Call and explicit inclusion of modules will be entered into the DALTAB table for the corresponding load module, as described in Chapter 7. When you execute this module subsequently, the OS IV/F4 Supervisor resolves these references dynamically from the step, job, and/or link libraries.

Non-editable (NE) attribute
The Linkage Editor cannot re-process any load module which you have marked NE (not editable). If you request a module map or a cross reference table, the "not editable" attribute is ignored.

To assign the "not editable" attribute, code NE in the PARM field, as follows:

```
//LKED          EXEC    PGM=JQAL,
//                      PARM='NE,...'
```

Only-loadable (OL) attribute
You can bring a module with the "only loadable" attribute into virtual storage only with a LOAD macro instruction. To execute a module with the "only loadable" attribute, you issue a Branch instruction or a CALL macro instruction. If you attempt to enter the module with a LINK, XCTL, or ATTACH macro instruction, your program is terminated abnormally by the OS IV/F4 Supervisor.

To assign the "only loadable" attribute, code OL in the PARM field as follows:

```
//LKED          EXEC    PGM=JQAL,
//                      PARM='OL,...'
```

Note: The "only loadable" attribute is intended primarily for use by the OS IV/F4 Supervisor. You may impair usability of your module by specifying OL.

Overlay (OVLY) attribute
You edit your program into an overlay structure as directed by Linkage Editor OVERLAY control statements if you assign it the "overlay" attribute. Your module cannot be also designated as refreshable, re-enterable, or serially reusable.

If you specify the "overlay" attribute but furnish no OVERLAY control statements, the attribute is negated.

If you omit specifying the "overlay" attribute any OVERLAY or INSERT statements are considered invalid, and your module is not linked into an overlay structure. This condition is also recoverable; if you specify the LET option, your module is marked executable.

To assign the overlay attribute, code OVLY in the PARM field as follows:

```
//LKED          EXEC    PGM=HEWL,
//                      PARM='OVLY,...'
```

See Chapter 6 for information on the design and specification of an overlay structure.

Refreshable (REFR) attribute
A refreshable module can be replaced by a new copy during execution by a Recovery Management routine (part of the OS IV/F4 Supervisor) without changing either the sequence or results of processing. This type of module cannot be modified by itself or by any other module during execution. The Linkage Editor only stores the attribute in the directory entry; it does not check whether the module is truly refreshable. Refreshability and reenterability are essentially identical attributes. Most of your programs need not be designated "refreshable", although it may be useful for programs processing real-time data or other applications requiring ultra-reliable operation.

All control sections of a refreshable load module must also be refreshable. If any load modules that are not refreshable enter the input to the Linkage Editor, the resulting load module is not refreshable.

To assign the refreshable attribute, code REFR in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='REFR,...'
```

Reusability attributes (RENT and REUS)
Reusability means that the same copy of a load module can be used by more than one task either concurrently or serially. The reusability attributes are **reenterable** and **serially reusable**. If you specify neither attribute, your module is not reusable. In this case, the OS IV/F4 Supervisor must bring a fresh copy into virtual storage for each task wishing to use the module.

The Linkage Editor stores the attribute you specify into the directory entry; it does not check whether the module is truly re-enterable or serially reusable. A re-enterable module is automatically reusable, but not conversely.

A **reenterable** module can be executed by more than one task at a time; that is, you may begin using a reenterable module before a previous task has finished executing it. You cannot modify a reenterable module, by definition—nor can any other user.

All control sections within a reenterable module are obviously themselves reenterable. If any non-reenterable modules are merged with reenterable modules by the Linkage Editor, the resulting load module is non-reenterable.

To assign the reenterable attribute, code RENT in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='RENT,...'
```

A **serially reusable** module can be used by only one task at a time; that is, you may not begin executing a serially reusable module before a previous task has finished executing it. This type of module must initialize itself and/or restore any instructions or data in the module altered during execution.

All control sections of a serially reusable module must be either serially reusable or reenterable. If any load modules that are neither serially reusable nor reenterable enter the input to the Linkage Editor, the resulting load module is not serially reusable.

To assign the serially reusable attribute, code REUS in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='REUS,...'
```

Test attribute
A module with the test attribute contains symbol tables appropriate for program testing by TSS TEST commands. The Linkage Editor accepts these symbol tables as input and places them in the output module. The module is marked as being under test. If the TEST attribute is not specified, any symbol tables are ignored by the Linkage Editor, and they are not placed into the output module. If you specify the TEST attribute but furnish no symbol tables, the output load module will not contain symbol tables needed for TSS testing.

To assign the test attribute, code TEST in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='TEST,...'
```

Default attributes
Unless you specify corresponding module attributes your output module is not linked into an overlay (OVLY) structure or a testable (TEST) format. Likewise, your module is neither refreshable nor reenterable nor serially reusable. The Linkage Editor aligns its control sections on 4K page boundaries if—and only if—you request page boundary alignment.

One other attribute is automatically specified by the Linkage Editor after it completes processing each load module. If it detected errors preventing the output module from being executed successfully (Severity 2), the Linkage Editor assigns the "not executable" attribute. The OS IV/F4 Supervisor will refuse to load a module with this attribute.

If you specify the LET option, the Linkage Editor marks the output module "executable" even if severity-2 errors occur. The LET option is discussed later in this section.

If you omit the AC parameter or code it incorrectly, the Linkage Editor assigns a default authorization code of zero (0) to the output module.

Incompatible attributes
Of the module attributes which you can specify, several are mutually exclusive. If you specify two or more mutually exclusive attributes for a load module, the Linkage Editor ignores the less significant attributes. For example, if you specify both OVLY and RENT, your module will be linked into an overlay structure but it will not be reenterable.

Certain attributes are also incompatible with other job-step options. For convenience, all job step options are shown in Fig. 8.3 at the end of this chapter along with those options that are incompatible.

8.2.2 Other Processing Options: LET, NCAL, XCAL

You can specify several other processing options which determine whether you can later execute your output module or whether you wish to invoke the Automatic Library Call mechanism. These options are "exclusive call", "let execute" and "no automatic call".

Let execute (LET) option
If you specify the LET option, the Linkage Editor markes your output module "executable" even if it detects one or more severity-2 error conditions during processing. (A severity-2 error condition could make execution of the output load module impossible.) Some examples of severity-2 errors are as follows:
- Unresolved external references.
- Valid or invalid exclusive calls in an overlay program.
- Error on a Linkage Editor control statement.
- Missing library module (or misspelled reference)
- No available space in the directory of the output module library.

To specify the "let execute" option, code LET in the PARM field as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='LET,...'
```

Note: If you specify LET, you need not also specify XCAL.

No automatic library call (NCAL) option
If you specify the NCAL option, the Linkage Editor automatically retrieves library members to resolve external references. The output module is marked "executable" even if it contains unresolved external references. If you specify this option, you need not furnish LIBRARY statements to negate Automatic Library Call for selected external references. With this option, you need not furnish a SYSLIB DD statement.

To specify the NCAL option, code NCAL in the PARM field as follows:

```
//LKED          EXEC   PARM=JQAL,
//                     PARM='NCAL,...'
```

Note: Other errors may cause the module to be marked "not executable" unless you also specify the LET option.

Exclusive call (XCAL) option
if you request the exclusive call option, the Linkage Editor marks your output module "executable" even if your program (in overlay format) issues valid exclusive references among its segments. However, the Linkage Editor prints a warning message for each valid exclusive reference.

To specify the exclusive call option, code XCAL in the PARM field as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='XCAL,OVLY,...'
```

Note: Other errors may cause the module to be marked "not executable" unless the "let execute" option is specified.

8.2.3 Storage Allocation Options: AM256, DCBS, SIZE

These options allow you to influence how the Linkage Editor should allocate virtual storage for itself, and also to specify the blocksize for the output module and to increase the maximum number of aliases permitted per module.

Aliases maximum of 256 (AM256)
This parameter notifies the Linkage Editor that your module(s) may have more than the default maximum of 64 aliases per member. This option is typically required only for large dynamic link structures.

DCBS option
The DCBS option allows you to specify the block size for your SYSLMOD data set in the DCB parameter of your SYSLMOD DD statement.

If you specify the DCBS option, the block-size value in the SYSLMOD DSCB may be overridden. If you omit the DCBS option, this block-size value may not be overridden.

If you specify the DCBS option but no block-size value in the DCB parameter of your SYSLMOD DD statement, the Linkage Editor uses the maximum track size for the device. If you omit the DCBS option but you provide a block-size value in your DCB parameter the Linkage Editor ignores this block-size value.

Even if you specify a DCBS option, the Linkage Editor will not allow you to set the block size for the SYSLMOD data set to a subminimal value, i.e., less than the block size of a DSCB for an existing data set.

Any block size you specify will be used unless (1) it is larger than the maximum record size for the device, in which case the maximum record size is used, or (2) it is less than the minimum block size, in which case the minimum block size is used.

The following example shows the use of the DCBS option for a F478 disk drive:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='XREF,DCBS'
                 .
                 .
                 .
//SYSLMOD       DD     DSNAME=LOADMOD(TEST),
//                     DISP=(NEW,KEEP),
//                     DCB=(BLKSIZE=3072),...
```

As a result, the Linkage Editor uses a 3K blocksize for your output module library.

Note: When you furnish a DCBS option, you must specify a blocksize subparameter in the DCB parameter of your SYSLMOD DD statement.

SIZE option
You can specify the amount of virtual storage to be used by the Linkage Editor and, separately, how much

of this storage should be used for the load module buffer.

Each installation selects default values for the SIZE option during system generation. These default values are used if you omit one or both SIZE parameters or if you fail to specify them correctly. Most OS IV/F4 installations will choose sufficiently generous SIZE default values so that its users need never specify SIZE parameters.

Format: Permissible formats for the SIZE option are as follows:

```
SIZE=(value1, value2)
SIZE=(value1)
SIZE=(value1,)
SIZE=(,value2)
SIZE=(,)
```

When coded in the PARM field, the expression is enclosed in single quotes, as follows:

```
//LKED        EXEC   PGM=JQAL,
//                   PARM='SIZE=(value1,
//                   value2),...'
```

"Value1" and "value2" may be expressed as (a) integers specifying the number of bytes of virtual storage or (b) nK where **n** represents the number of **kilobytes** (1024 bytes) of virtual storage.

When determining the values for the SIZE option, you should calculate "value2" first, then "value1".

"**Value2**" specifies the number of bytes of storage to be allocated for the module buffer. The allocation specified by "value2" is a part of the virtual storage specified by "value1".

The minimum for "value2" is 6144 (6K) or the length of the largest input load module text block, whichever is larger. If you specify a value less than 6144 (6K), the Linkage Editor was its default value for "value2".

Space allocated by "value2" is used for the following buffers:

- buffer into which input load-module text is read,
- buffer from which load-module text is written to the intermediate data set,
- buffer into which load-module text is read from the intermediate data set, and
- buffers from which load-module text is written to the output data set.

Therefore, the determination of "value2" requires that you consider the maximum block sizes of data sets from which load-module test is read (SYSLIB, or any data set referenced by an INCLUDE statement, or any library data set), block size for the intermediate data set (SYSUT1) and block size for the output load module data set (SYSLMOD).

Table 8.1 lists FACOM direct access devices that you can use for input load modules, the intermediate data set, and output load modules. This table lists the maximum block size used for each device by the Linkage Editor. You can always specify these maximum block sizes when calculating "value2".

Table 8.1 SYSUT1 and SYSLMOD device types and corresponding maximum block sizes

Device	Maximum block size
F6625	13312 or 13K
F478	12288 or 12K
F479	12288 or 12K

You must specify "value2" so that the Linkage Editor has sufficient storage to allocate buffers compatible with block sizes for the intermediate and output data sets.

The Linkage Editor optimizes the record size for the output data set unless one of the following conditions exists.

1. You specify PARM='...DCBS...' and your SYSLMOD DD statement contains a BLKSIZE subparameter in the DCB parameter, forcing the Linkage Editor to write records having a maximum length equal to the BLKSIZE specification.
2. Your (existing) data set has its block size less than the optimum record size, forcing the Linkage Editor to write records no longer than that block size.
3. You specify "value2" smaller than twice the maximum record size for the output data set, forcing the Linkage Editor to write records having a maximum size of 50%* "value2".
4. Your intermediate and output data sets have dissimilar record sizes, forcing the Linkage Editor to write records having a maximum size compatible with both data sets.

The Linkage Editor optimizes the record size of the output data set for its device type, but it selects a record size compatible with the intermediate data set (as described above). Therefore, you optimize processing of load-module buffers if you allocate the intermediate and output data sets to the same type of device. The performance of the Linkage Editor is also improved if you allocate these data sets to different units of the same type.

"**Value1**" specifies how much virtual storage you wish to allocate to the Linkage Editor regardless of region size. The storage specified by "value1" includes the allocation specified by "value2".

The minimum for "value1" is 64K. If you specify less than this minimum the Linkage Editor uses installation default options for both "value1" and "value2".

The Linkage Editor supports blocking factors of 5, 10, and 40 for SYSLIN, object module libraries, and SYSPRINT data sets. The requirement for additional space depends upon the requested blocking factor.

The Table 8.2 shows the additional "value1" space—in excess of 64K—required to support each blocking factor.

Table 8.2 Space increment to support blocking

Blocking factor	Required increment to "value1"
5 to 1	0
10 to 1	18432 or 18K
40 to 1	28672 or 28K

Blocking factors of 1 through 4, 6 through 9, and 11 through 39 are treated as blocking factors of 5, 10, and 40, respectively. Blocking factors greater than 40 are invalid.

If you specify "value1" greater than the region size, the Linkage Editor may use some storage required for data management and other system functions; this lack of storage will result in its abnormal termination.

"Value1" should be as large as possible. The performance of the Linkage Editor slightly improves if you allocate it additional storage.

Examples of value1 determination

1. Assume you have already determined an optimum "value2" of 36K for a paricular linkage-edit job step. An appropriate "value1" is 94K, since 30K bytes above the minimum of 64K is needed for the "value2" allocation and no additional storage is required to raise the blocking factor for SYSLIN, SYSPRINT, and object-module libraries.
2. Assume you assign a minimum "value2" (6K) and that all object module libraries are blocked 5 to 1, except one that is blocked 10 to 1. Assume your SYSLIN and SYSPRINT data sets require blocking factors of 5. An appropriate "value1" for this link-edit is 82K: 18K above the minimum needed to support the blocking factor of 10 to 1 on the object module library.
3. Assume the same situation exists as in example 2, except that the minimum region size is 100K. A more appropriate "value1" under these circumstances is 90K. Since extra space is available, you can optimize the allocated region and increase "value2" to 18K.

8.2.4 Output options: LIST, MAP, XREF, TERM

These options control the optional diagnostic output produced by the Linkage Editor. You can request that the Linkage Editor produce a list of all control statements, a module map, and/or a cross-reference table to help you test your program. The format of each is described in Chapter 4.

In addition, you can request that any numbered error/warning messages generated by the Linkage Editor should appear on the SYSTERM data set as well as the SYSPRINT data set.

Control statement listing option

To request a control statement listing, code LIST in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='LIST,...'
```

When you specify the LIST option, your control statements are listed in card-image format on the diagnostic output data set.

Module map option

To request a module map, code MAP in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='MAP,...'
```

When you specify the MAP option, the Linkage Editor produces a storage map of the output module on the diagnostic output data set.

Cross reference table option

To request a cross-reference table, code XREF in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='XREF,...'
```

When you specify the XREF option, the Linkage Editor produces a cross-reference table of the output module on the diagnostic output data set. Since the cross reference table includes a module map, you need not specify both XREF and MAP.

Alternate output (SYSTERM) option

To request that any numbered error/warning messages be displayed on the data set defined by your SYSTERM DD statement, code TERM in the PARM field, as follows:

```
//LKED          EXEC   PGM=JQAL,
//                     PARM='TERM,...'
```

If you specify the TERM option you must furnish a SYSTERM DD statement; otherwise "TERM" is ignored.

8.2.5 Incompatible Options

If you specify mutually exclusive job-step options, the Linkage Editor ignores the less significant options. Table 8.3 illustrates the relative significance of these options. When an "X" appears at an intersection, the options are incompatible, and the option that appears higher in the list is selected.

For example, to check the compatibility of XREF and NE, follow the XREF column down and the NE row across until they intersect. Since an X appears where they intersect, they are incompatible; XREF is selected, NE is negated.

If you furnish incorrect values for the SIZE parameter, the Linkage Editor uses your installation default values.

If the Linkage Editor detects incompatible options, it prints the following message:
OPTIONS INCOMPATIBLE
after the standard module disposition message.

Table 8.3 Incompatible job step options for the Linkage Editor

	OVLY	RENT	REUS	REFR	TEST	XREF	MAP	NE	ALIAS	ALIGN2	DCBS	DYNAMIC	LET	LIST	NCAL	OL	SIZE	XCAL	AC
RENT	x																		
REUS	x	x																	
REFR	x																		
TEST																			
XREF																			
MAP						x													
NE					x	x	x												
ALIAS																			
ALIGN2																			
DCBS																			
DYNAMIC																			
LET																			
LIST																			
NCAL																			
OL																			
SIZE																			
XCAL																			
AC																			
AM256																			

Table 8.4 Linkage Editor return codes

Return code	Severity code	Description
00	0	Normal conclusion.
04	1	Warning messages have been listed, but execution should be successful. For example, if you requested an overlay structure of only one segment, a return code of 04 ia issued.
08	2	Error messages have been listed, and execution may fail. The Linkage Editor marks your module "not executable" unless you have specified the LET option. For example, if the blocksize of a specified library data set cannot be handled by the Linkage Editor, a return code of 08 is issued.
12	3	Severe errors have occurred and execution is impossible. For example, if you specify an invalid entry point, the Linkage Editor issues a return code of 12.
16	4	Terminal errors have occurred, and processing has terminated. For example, if the Linkage Editor cannot handle the blocking factor you requested for SYSPRINT, it issues a return code of 16.

8.2.6 Return Codes

The Linkage Editor passes a return code to OS IV/F4 Job Management upon completion of the job step. The return code reflects the highest severity code recorded in any iteration of the Linkage Editor within that job step. The highest severity code encountered during processing is multiplied by 4 to create the return code; the Linkage Editor loads this code into register 15 at the end of its processing. Table 8.4 contains return codes, corresponding severity codes, and their descriptions.

You can use this return code to determine whether or not to execute the load module by using a condition-code test (COND) on the EXEC statement for the load module. OS IV/F4 Job Management compares the return code with values you specify in the COND parameter; results from these comparisons determine subsequent action.

8.2.7 DD Statements

Every data set used by the Linkage Editor must be described in a DD statement. Each DD statement must have a name, except for the second and subsequent data sets of a concatenation. DD statements required by the Linkage Editor have pre-assigned names; those for additional input data sets have names you assign; those for concatenated data sets (after the first) have no names.

In addition to its name, each DD statement provides OS IV/F4 with information about the input/output device on which the data set resides, and a description of the data set itself. All job-control facilities for describing devices are available to users of the Linkage Editor.

Besides information about the device, your DD statement also contains a data set description, which includes the data set name and its disposition. You may also supply information for its data control block (DCB).

Data Set Name

The Linkage Editor uses either sequential or partitioned data sets. For sequential data sets, only their names need be specified; for partitioned data sets, member names must also be specified either on corresponding DD statements or with control statements.

When you pass input data sets from a previous job step, or when you are testing a new load module, you should typically use **temporary data set** names (i.e., &&dsname) to ensure that no duplicate data sets remain. Any data set with a temporary name is automatically deleted at the end of your job. When you wish to store a module permanently, you must use a data set name without ampersands.

DCB Information

As you create each new data set, information about it must be placed in the associated data control block (DCB). If this information does not exist in a DCB or header label, you must specify it in the DCB parameter on your DD statement.

Record format (RECFM), logical record size (LRECL), and blocksize (BLKSIZE) subparameters of the DCB parameter are discussed below as they apply to the Linkage Editor. Specific information on each appears in its description, which follows later in this section. Other DCB parameters (tape recording technique, density, and so forth) are described in the **FACOM OS IV/F4 Job Control Language Reference Manual** and the **FACOM OS IV/F4 Job Control Language User's Guide**.

Record Format

The following record formats are used with the Linkage Editor:

RECFM=	**Records are:**
F	fixed length
FB	fixed length and blocked.
FBA	fixed length, blocked, and contain ANSI control characters
FBS	fixed length, blocked, and written in standard blocks
FA	fixed length and contain ANSI control characters
FS	fixed length and written in standard blocks
U	undefined length
UA	undefined length and contain ANSI control characters

You should use a record format of FS or FBS with caution; all blocks in the data set must be the same size; this size must be equal to the specified blocksize; and a truncated block can occur only as the last block in the data set.

Note: Track overflow is never used by the Linkage Editor. When moving or copying load modules, you should not use the track overflow feature for target data sets, since errors may subsequently occur when you attempt to retrieve the modules for execution.

Logical Record and Block Sizes

Blocking is allowed for input object module data sets and the diagnostic output data set. The blocking factors used to determine buffer allocations are 10 and 40. BLKSIZE must therefore be a multiple of LRECL. See the description of blocking factors in Section 8.2.3 under "SIZE Option."

Also, you should specify a blocksize for your output load module library whenever you specify the DCBS option, as described under "SYSLMOD DD Statement" later in this section.

8.2.8 Standard Linkage Editor DD Statements

The Linkage Editor uses up to six data sets of which four are required. DD statements for these data sets must use the preassigned DD names given in Table 8.5. The descriptions that follow give pertinent device and dataset information for each data set.

Table 8.5 Linkage Editor DD names

Data set	DD name	Required
Primary input data set	SYSLIN	Yes
Automatic call library	SYSLIB	Only if the Automatic Library Call mechanism is used
Intermediate data set	SYSUT1	Yes
Diagnostic output data set	SYSPRINT	Yes
Output module library	SYSLMOD	Yes
Alternate output data set	SYSTERM	Only if the TERM option is specified

SYSLIN

The SYSLIN DD statement is required; it describes the primary input data set, which you can assign to a direct-access device, magnetic tape unit, card reader, or the JES input stream (DD*). The data set may be either sequential or partitioned; in the latter case you must specify a member name.

If you assign SYSLIN to a card reader or the JES input stream, input records must be unblocked and 80 bytes long. This data set must contain object modules and/or control statements. Furnishing load modules as primary input is considered a severity-4 error.

Table 8.6 DCB requirements for object module and control statement input

LRECL	BLKSIZE	RECFM
80	80	F,FS
80	800,3200*	FB,FBS

* These are the maximum blocksizes allowed. Which maximum is applicable depends on any values you assign to "value 1" and "value 2" of the SIZE option.

SYSLIB

A SYSLIB DD statement is required whenever you wish to use the Automatic Library Call mechanism. This DD statement describes the automatic call library, which must be assigned to a direct-access device. The data set must be partitioned, but you should not specify any member names.

If you furnish concatenated call libraries you must not intermix object and load module libraries.If you only furnish object modules in SYSLIB, your call library may also contain control statements.

The DCB requirements for object-module call libraries are given in Table 8.6. The DCB requirement for load-module call libraries is a record format of U; the blocksize used for storage allocation is equal to the maximum for the device used, not the record read.

SYSUT1

The SYSUT1 DD statement is required; it describes the intermediate sequential data set, which must be assigned to a direct-access device. Space must be allocated for this data set but its DCB requirements are supplied by the Linkage Editor.

SYSPRINT

The SYSPRINT DD statement is required; it describes the diagnostic output written by the Linkage Editor to a sequential data set assigned to a printer, intermediate storage device, or the JES output stream (SYSOUT=A). If an intermediate storage device is used, the data records contain a carriage control character as the first byte. The usual specification for this data set is SYSOUT=A. You may assign a blocksize, but the record format assigned by the Linkage Editor depends on whether blocking is used or not.

Table 8.7 shows the DCB requirements for SYSPRINT. The only parameter you can furnish is the blocksize.

Table 8.7 DCB requirements for SYSPRINT

LRECL	BLKSIZE	RECFM
121	121	FA
121	n x 121 where n is less than or equal to 40	FBA

Note: The value specified for BLKSIZE, either on the DCB parameter of the SYSPRINT DD statement or in the DSCB (data set control block) of an existing data set, must be a multiple of 121; if it is not, the Linkage Editor issues a message to the operator's console and terminates processing.

SYSLMOD

The SYSLMOD DD statement is required; it describes the output module library, which must be a partitioned data set assigned to a direct-access device.

You may optionally specify a member name on your SYSLMOD DD statement, used only if you omitted specifying its name on a NAME control statement. Your member name implies replacement of an identically-named member in the load-module library, if one exists.

If your new member is to replace an identically-named member, you should code DISP=OLD on your SYSLMOD statement. If your member is to be added to an existing library, its disposition should be MOD, OLD, or SHR. If no library exists and your member is the first to be created, its disposition should be NEW or MOD. If your member is to be added to an existing library used concurrently in another region or partition, the disposition should be SHR.

The record format U is assigned by the Linkage Editor. See Appendix 1 for details on load-module formats.

The Linkage Editor assigns a blocksize by:

1. Finding the smallest of the following values:
   - The maximum track size for the device
   - The value of the BLKSIZE subparameter in the DCB parameter on the SYSLMOD DD statement, if you specified the DCBS option
   - The actual output buffer length (half the number specified for "value2" of the SIZE option)
2. Comparing the smallest value above to the value currently in the DSCB. The greater value is assigned as the block size.

SYSTERM

The SYSTERM DD statement is optional; it describes a data set used only for numbered error/warning messages. Although primarily for keyboard terminals operating under the OS IV/F4 Time Sharing System you can use the SYSTERM DD statement in any environment to define a data set consisting only of numbered error/warning messages.

You define SYSTERM output by (a) including a SYSTERM DD statement in your JCL and (b) specifying TERM in the PARM field of your EXEC statement. If you request SYSTERM output the Linkage Editor writes numbered messages to both the SYSTERM and SYSPRINT data sets.

DCB requirements for SYSTERM (LRECL=121,BLKSIZE=121, and RECFM=FBA) are supplied by the Linkage Editor. If necessary, the Linkage Editor will modify the DSCB (data set control block) of an existing data set to reflect these values.

8.2.9 Additional DD Statements

Each DD name you specify on an INCLUDE or LIBRARY control statement must correspond to a DD statement you must furnish. These DD statements describe sequential or partitioned data sets assigned to magnetic tape units or direct-access devices. DCB requirements for these data sets are shown in Table 8.8.

Table 8.8 DCB requirements for additonal input data sets

Data set contents	RECFM	LRECL	BLKSIZE
Object modules and/ or control statements	F,FS	80	80
Load modules	U	1K	1K
Object modules and/ or control statements	F,FS FB,FBS	80	80 400,800,3200*
Load modules	U	maximum for device or one-half of "value2", whichever is smaller	equal to LRECL

* These are the maximum block sizes allowed. Which maximum is applicable depends on the values given to "value 1" and "value 2" of the SIZE option.

If you concatenate two or more data sets, their records must have the same formats, record sizes, and block sizes. If these data sets reside on magnetic tape, their tape recording techniques and densities must also be identical.

8.3 CATALOGED PROCEDURES

OS IV/F4 allows each installation to store a frequently-useful collection of EXEC and DD statements under a unique member name in a procedure library. Such a series of job control language (JCL) statements is called a **cataloged procedure.** You can recall these statements at any time to specify your job requirements. To request a particular procedure, place you an EXEC statement in the input stream specifying its member name (**procedure name**).

You can override parameters of a cataloged procedure temporarily (i.e., for the duration of a single job) and also add DD statements to selected job steps. Information you alter in this way is in effect only for the duration of one job step; the cataloged procedures themselves are not altered permanently by invocation as described below. Any DD statements you add must follow those that override statements predefined within a particular procedure.

8.3.1 Standard Linkage Editor Procedures

OS IV/F4 provides two standardized cataloged procedures for using the Linkage Editor: a single-step procedure that link-edits the input and produces a load module (procedure LKED), and a two-step procedure that link-edits the input, produces a load module, and executes that module (procedure LKEDG). Many cataloged procedures OS IV/F4 defines for the language translators also contain Linkage Editor steps, as shown in Table 8.9.

Table 8.9 Standardized OSIV/F4 cataloged procedures using the Linkage Editor

Language	Compile and link edit procedure	Compile, link, and go procedure
Assembler	ASMFCL	ASMFCLG
Cobol	COBUCL	COBUCLG
Fortran GE	FORTACL	FORTACLG
Fortran HE	FORTBCL	FORTBCLG
PL/I	PL1XFCL	PL1XFCLG
SL/100	SL1CL	SL1CLG
(none)	LKED	LKEDG

LKED procedure
The standard OS IV/F4 cataloged procedure named LKED is a single-step procedure that link edits input, produces a load module, and passes the load module to another step in the same job. The statements in this procedure are shown in Fig. 8.1; the following is a description of these statements.

- **Statement Numbers**
  You can use the 8-digit numbers on the right-hand side of each statement to identify it. They could be used, for example, when permanently modifying the cataloged procedure with the JSEUPDTE utility program. For a description of this utility program, see the **FACOM SO IV/F4 Data Set Utility User's Guide.**

- **EXEC Statement**
  The PARM field specifies the XREF, LIST, LET, and NCAL options. If you wish to use the Automatic Library Call mechanism, you must override the NCAL option and add a SYSLIB DD statement. Overriding and adding DD statements are discussed later in this section.

- **SYSPRINT Statement**
  The SYSPRINT DD statement specifies SYSOUT class A, which is either a printer, an intermediate storage device, or the JES output stream. If you use JES or an intermediate storage device, you should furnish carriage control characters preceding the data.

- **SYSIN Statement**
  By specifying DDNAME=SYSIN, you can subsequently define any data set as long as it fulfills the requirements for Linkage Editor input. You define your input data set with a DD statement whose DD name is SYSIN. This data set may be either in the JES input stream or residing on a separate volume.

  If your data set is in the JES input stream, you must furnish the following SYSIN statement:

```
//LKED.SYSIN        *
```

```
//LKED        EXEC   PGM=JQAL,PARM='XREF,LIST,LET,NCAL',REGION=110K    00020000
//SYSPRINT    DD     SYSOUT=A                                           00040000
//SYSLIN      DD     DDNAME=SYSIN                                       00060000
//SYSLMOD     DD     DSNAME=&&GOSET(GO),SPACE=(1024,(50,20,1)),        C00080000
//                   UNIT=SYSDA,DISP=(MOD,PASS)                         00100000
//SYSUT1      DD     UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),                C00120000
//                   SPACE=(1024,(200,20))                              00140000
```

Fig. 8.1 Statements in the LKED cataloged procedure

It may be anywhere among your DD statements for this job step as long as it follows all overriding DD statements. Any object module decks and/or control statements should immediately follow your SYSIN statement, with a delimiter statement (/*) at the end of the input.

If the data set resides on a separate volume, the following SYSIN statement is used:

```
//LKED.SYSIN   DD      parameters describing an input
                       data set
```

If this SYSIN statement is used, it may be anywhere among your DD statements as long as it follows all overriding DD statements for this step. You can concatenate several data sets, described in Chapter 3.

- **SYSLMOD Statement**
  The SYSLMOD DD statement specifies a temporary data set and a general space allocation. The disposition allows the next job step to execute the load module. If the load module is to reside permanently in a library, various parameters of these general specifications must be explicitly overridden.

- **SYSUT1 Statement**
  The SYSUT1 DD statement specifies that the intermediate data set is to reside on a direct-access device, but not the same device as either the SYSLMOD or the SYSIN data sets. Again, a general space allocation is given.

- **SYSLIB Statement**
  Note that there is no SYSLIB DD statement. If you wish to use the Automatic Library Call mechanism with this cataloged procedure, you must add a SYSLIB DD statement and also negate the NCAL option in the PARM field of your EXEC statement.

- **Invoking the LKED Procedure**
  To invoke the LKED procedure, code the following EXEC statement:

```
//stepname      EXEC    LKED
```

where **stepname** is optional and is the name of the job step.

The following example shows the use of the SYSIN DD* statement:

```
Step A:
//LESTEP        EXEC    LKED
[Overriding and additional DD statements for the LKED
step, each beginning "//LKED.DDname..."]
//LKED.SYSIN    DD      *
[Object module decks and/or control statement]
Step B:
//EXSTEP        EXEC    PGM=*.LESTEP.LKED.
                            SYSLMOD
[DD statements and data for load module execution]
```

If you supply data for your execution step, your data must be followed by a delimiter (/*) statement.

Step A invokes the LKED procedure and Step B executes the load module produced in Step A. The job control language statements for these two steps are combined in LKEDG cataloged procedure.

LKEDG procedure
The cataloged procedure named LKEDG is a two-step procedure that link-edits input, produces a load module, and executes that load module. The statements in this procedure are shown in Fig. 8.2. The steps are named LKED and GO. The specifications in the statements in the LKED step are identical to the specifications in the LKED in the LKED procedure.

- **GO Step**
  The EXEC statement specifies that the program to be executed is the load module produced in the LKED step of this job. This module was stored in the data set described on the SYSLMOD DD statement in that step. If a NAME statement was used to specify a member name other than that used on the SYSLMOD statement, you should use the LKED procedure rather than LKEDG.

  The condition (COND) parameter on the GO EXEC statement specifies that the execution step will be bypassed if the return code issued by the LKED step is greater than 4.

- **Invoking the LKEDG Procedure**
  To invoke the LKEDG procedure, code the following EXEC statement:

```
//stepname      EXEC    LKEDG
```

```
//LKED        EXEC   PGM=JQAL,PARM='XREF,LIST,NCAL',REGION=110K        00020000
//SYSPRINT    DD     SYSOUT=A                                          00040000
//SYSLIN      DD     DDNAME=SYSIN                                      00060000
//SYSLMOD     DD     DSNAME=&&GOSET(GO),SPACE=(1024,(50,20,1)),       C00080000
//                   UNIT=(SYSDA,DISP=(MOD,PASS)                       00100000
//SYSUT1      DD     UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),               C00120000
//                   SPACE=(1024,(200,20))                             00140000
//GO          EXEC   PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)               00160000
```

Fig. 8.2 Statements in the LKEDG cataloged procedure

where **stepname** optionally names the job step. The following example shows the use of the SYSIN DD* statement with the LKED procedure:

```
//TWOSTEP      EXEC     LKEDG
[Overriding and additional DD statements for the LKED
step, each beginning //LKED.DDname...]
//LKED.SYSIN   DD       *
[Object module decks and/or control statements]
/*
[DD statements for the GO step, each beginning
//GO.DDname...]
//GO.SYSIN     DD       *
[Data for the GO step]
/*
```

8.3.2 Overriding Procedure Statements

The programmer may override any EXEC or DD statement specifications in a cataloged procedure. These temporary specifications remain in effect only for the duration of his job step. For a detailed description of overriding cataloged procedures, see the **FACOM OS IV/F4 Job Control Language Reference Manual** or **FACOM OS IV/F4 Job Control Language User's Guide.**

Overriding an EXEC Statement
You can override one or more EXEC statements in a cataloged procedure by specifying changes and additions on the EXEC statement that invokes the cataloged procedure. You must ordinarily specify the stepname when overriding EXEC-statement parameters. For example, you can increase the REGION parameter as follows:

```
//LESTEP    EXEC    LKED,REGION.LKED=136K
```

The rest of the specifications on the EXEC statement of the LKED procedure remain in effect.

If you need to override the PARM field, all PARM options specified in the cataloged procedure are negated. For example, if you wish to specify the XREF, LIST, or NCAL options when overriding the PARM field, you must re-specify each non-default value. In the following example, the OVLY option is added and the NCAL option is negated:

```
//LESTEP        EXEC     LKED,PARM.LKED='OVLY,
//                       XREF,LIST'
```

As a result, you have retained the XREF and LIST options but have dropped the NCAL option. Remember, if you negate NCAL in this way, you must add a SYSLIB DD statement.

If you use the LKEDG procedure to execute a load module just built, an efficient way is to specify the LET parameter in your LKED step, i.e., invoke the LKEDG procedure with the following EXEC statement:

```
//stepname      EXEC     LKEDG,PARM.LKED='XREF,
//                       LIST,NCAL,LET'
//                       COND.GO=(8,LT,LKED)
```

Overriding DD statements
You can override any DD statements in a cataloged procedure **as long as the overriding statements are in the same order as they appear in the procedure.** DD statements not overridden follow the specifications in the cataloged procedure.

Only those parameters you explicitly override are affected; other parameters remain as specified in the procedure. In the following example, the output load module is to be placed in a permanent library:

```
//LIBUPDTE        EXEC LKED
//LKED.SYSLMOD DD      DSNAME=LOADLIB(PAYROLL),
//                     DISP=OLD
//LKED.SYSIN      DD   DSNAME=OBJMOD,
//                     DISP=(OLD,DELETE)
```

As a result of the statements in the example, the LKED procedure processes the object module in the OBJMOD data set, storing the output load module in the LOADLIB data set with member name PAYROLL. The SPACE parameter on the SYSLMOD DD statement and the other specifications in the procedure remain in effect.

Note: A common source of JCL errors is incorrectly sequencing overriding statements. In the preceding example, if the SYSIN statement had preceded the SYSLMOD statement, the latter would be erroneously placed **vis a vis** the LKED procedure. Since the overriding SYSLMOD statement would then be ignored by OS IV/F4 Job Management, the entire job would probably fail.

8.3.3 Adding DD Statements

You can supply DD statements for additional data sets when using cataloged procedures. These additional DD statements must follow any overriding DD statements; the former can be in any sequence.

In the following example, Automatic Library Call is used along with the LKEDG procedure:

```
//CPSTEP          EXEC LKEDG,PARM.LKED='XREF,
//                     LIST'
//LKED.SYSLMOD DD      DSNAME=LOADLIB(TESTER),
//                     DISP=OLD,...
//LKED.SYSLIB     DD   DSNAME=SYS1.PL1LIB,
//                     DISP=SHR
//LKED.SYSIN      DD   *
[Object module decks and/or control statements]
/*
//GO.SYSIN        DD   *
[Data for execution step]
/*
```

You have negated the NCAL option and added a

SYSLIB DD statement between the overriding SYSLMOD DD statement and the SYSIN DD statement.

8.4 DYNAMIC INVOCATION OF THE LINKAGE EDITOR

You can invoke the Linkage Editor during execution of another program by issuing one of the following macro instructions:

```
symbol   LINK      EP=JQAL,
                   PARAM=(optionlist,DDname-list),
                   VL=1
symbol   ATTACH    EP=JQAL,
                   PARAM=(optionlist,DDname-list),
                   VL=1
symbol   LOAD      EP=JQAL
symbol   XCTL      EP=JQAL
```

EP=JQAL
Specifies the symbolic name of the Linkage Editor. You can use "EP=LINKEDIT" for compatibility with other operating systems, since "LINKEDIT" is a standard OS IV/F4 alias for "JQAL."

PARAM=(optionlist, DDname-list)
Specifies, as a sublist, address parameters to be passed from the problem to the Linkage Editor. The first fullword in the address parameter list contains the address of the option and attribute list for the load module. The second fullword contains the address of a list of DD names. If you use standard DD names, you can omit this list.

optionlist
Specifies the address of a variable-length list containing any options and module attributes. You must furnish this address even if you need no list.

The option list must begin on a halfword boundary. The two high-order bytes count the number of bytes in the remainder of the list. If you specify no options or attributes, your count should be zero. Your option list is **free-form**; fields separated by commas, with no embedded blanks or zeros.

DDnamelist
Specifies the address of a variable-length list containing alternative DDnames for data sets used during linkage editing. If you use all standard DDnames you may omit this operand.

The list must begin on a halfword boundary. Its two high-order bytes count the number of bytes in the remainder of the list. Each name less than 8 bytes long must be left-justified and padded with blanks. If you omit an alternate DDname for a particular entry on the list, the Linkage Editor will use its standard DD name; you must store binary zeros into its 8-byte entry. Names can be omitted from the end by merely shortening the list, i.e., reducing the count field (head of the list) appropriately.

The sequence of the 8-byte entries in the DD-namelist is as follows:

Entry	**Alternate Name For:**
1	SYSLIN
2	member name (name under which the output load module is stored in the SYSLMOD data set; the Linkage Editor uses this entry if you omit the name from your SYSLMOD DD statement or furnish no NAME control statement)
3	SYSLMOD
4	SYSLIB
5	(not applicable)
6	SYSPRINT
7	(not applicable)
8	SYSUT1
9–11	(not applicable)
12	SYSTERM

VL
Specifies that the sign bit is to be set to 1 in the last fullword of the address parameter list.

Note: Among major program packages supplied by Fujitsu, SORT and several others invoke the Linkage Editor dynamically during their initialization phases in order to create tailored, high-efficiency programs incorporating optional user-supplied exit routines.

When the Linkage Editor completes processing, it returns a condition code to the OS IV/F4 Supervisor in register 15, as described in Section 8.2.6.

```
MAIN      CSECT
          :
          LA    13,SAVEAREA   SET SAVE AREA
          :                   POINTER
          LINK  EP=JQAL,PARAM=(PLIST,DLIST),VL=1
          :
          DS    0H
PLIST     DC    AL2(L1)       LENGTH OF OPTION
                              LIST
OPT       DC    C'LIST,XREF'  OPTIONS
L1        EQU   *-OPT
          DS    0H
DLIST     DC    AL2(L2)       LENGTH OF DD NAME
DDN       DC    XL8'0'        LIST (NO SYSLIN
          DC    XL8'0'        ALTERNATE)
          DC    C'LOADLIBU'   (NO ALTERNATE MEM-
L2        EQU   *-DDN         BER,NAME)
SAVEAREA  DS    18F           ALTERNATE DD NAME
          :
          END
```

Fig. 8.3 Example of invoking the Linkage Editor

Example:
The Fig. 8.3 shows how you can LINK to the Linkage Editor. After executing—whether successful or unsuccessful—the Linkage Editor returns control to the OS IV/F4 Supervisor, which returns control to the Assembly program of this example just after the LINK macro instruction.

Options requested in this example are LIST and XREF, to which PLIST points. An alternate DD name is used for SYSLMOD; hence, this job invoking the Linkage Editor should furnish a LOADLIB DD statement, whose data set will receive the new load module.

Register 13 points to the save area required by the OS IV/F4 Supervisor whenever you issue a LINK macro instruction.

CHAPTER 9
SUMMARY OF LINKAGE EDITOR CONTROL STATEMENTS

This chapter summarizes the OS IV/F4 Linkage Editor control statements:

- What each statement does
- Format of the statement
- Placement of the statement in the input stream
- Notes on its use
- One or more examples, together with appropriate job control language statements.

The control statements are described in alphabetical order. Before using this chapter, you should be familiar with following information on general format, format conventions, and placement. This chapter describes control statements in terms of punched cards, but the discussion applies equally to inputs from keyboard terminals, magnetic tape, etc.

General Format
Each control statement specifies an **operation** and one or more **operands.** Nothing must be written preceding the operation, which begins at or after column 2. The operation must be separated from the operand by one or more blanks.

A control statement can be continued on as many cards as necessary by terminating the operand at a comma, and 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 continue on the next card.

Format Conventions
The following conventions are used to describe the coding of Linkage Editor control statements:

- Capital letters indicate exact characters you must enter.
- Lower-case letters must be supplied by the user.
- Other punctuation (parentheses, commas, spaces, etc.) must be entered as shown.
- Braces { } indicate a choice of entry; unless a default is indicated, you must choose one of the entries.
- Brackets [ ] indicates an optional field or parameter.
- An ellipsis (...) indicates that multiple entries of the type immediately preceding the ellipsis are allowed.
- Items separated by a vertical bar (|) represent alternative items. No more than one of the items may be selected.

Placement Information
You may place Linkage Editor control statements 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.

9.1 ALIAS Statement

The ALIAS statement specifies additional names for the output library member; it also can specify names of alternative entry points. You can specify up to 64 names on one or more ALIAS statements for each library member. (The AM256 parameter raises this aliases limit to 256/member.) The Linkage Editor enters the names into the directory of the partitioned data set, in addition to the member name.

Format
The format of the ALIAS statement is:

$$\text{ALIAS} \qquad \left\{\begin{matrix}\text{symbol}\\ \text{externalname}\end{matrix}\right\} \left[\begin{matrix}\text{,symbol}\\ \text{,externalname}\end{matrix}\right] \ldots$$

symbol
specifies an alternate name for the load module. When the module is executed, the main entry point is used as the starting point for execution.

externalname
specifies a name that is defined as a control section name or entry name in the output module. When the module is called for execution, execution begins at the external name referred to.

Placement
You may place an ALIAS statement before, between, or after object modules or other control statements. It must precede a NAME statement used to specify the member name, if one is present.

Notes
- In an overlay program, an external name specified by the ALIAS statement must be in the root segment.
- You can assign no more than 64 alias names to one output module unless you raise this limit to 256 alias names with the AM256 parameter.
- Each alias for a load module is retained in the directory entry for the module; the Linkage Editor does not delete old aliases. Therefore, each alias must be unique; attempting to assign the same alias to more than one load module can cause incorrect module reference.
- You should delete obsolete alias names from the PDS directory with a system utility such as JSEPROGM to avoid future name conflicts.
- If the Replace (R) option is in effect for an output load module (that is, the load module replaces an identically-named module in the library), Replace applies to each ALIAS name for the load module as well as its primary name.

Example
You wish to assign an output module ROUT1 two alternate entry points, CODE1 and CODE2. In addition, you have written calling modules using both ROUT1 and ROUTONE to refer to the output module. Rather than correct the calling modules, you can assign an alternative library member name as follows:

```
ALIAS        CODE1,CODE2,ROUTONE
NAME         ROUT1
```

Since CODE1 and CODE2 are entry names in the output module, when you call the module by these names, execution begins at the referenced points. Modules that call the output module as "ROUTONE" now correctly refer to "ROUT1" at its main entry point. The names CODE1, CODE2, and ROUTONE appear in the library directory along with ROUT1.

9.2 CHANGE STATEMENT

You use a CHANGE statement to replace a specific external symbol by an immediately-following parenthesized symbol. The external symbol to be changed can be a control section name, an entry name, or an external reference. You can specify several changes with a single CHANGE statement.

Format
The format of the CHANGE statement is:

```
CHANGE     externalsymbol(newsymbol)
           [,externalsymbol(newsymbol)]...
```

externalsymbol
is the control section name, entry name, or external reference that is to be changed.

newsymbol
is the name to which the external symbol is to be changed.

Placement
You must place a CHANGE statement immediately before the module containing the external symbol to be changed or an INCLUDE statement specifying the module. The scope of the CHANGE statement spans the immediately-following object module or load module. The END record in the immediately-following object module, or End-of-Module indication in the immediately-following load module, delimits the scope of the CHANGE statement.

Notes
- External references from other modules to a changed control section name or entry name remain unresolved unless you take further action.
- If you misspell a symbol on a CHANGE statement, the Linkage Editor will not change it. You can review outputs such as the cross-reference listing or module map to verify each change.

Example 1
Two control sections in different modules have the name TAXROUT. If you wish to link edit the modules together, you must change one of the control section names. If the module to be changed is defined by a DD statement named OBJMOD, you could change the control section name as follows:

```
//OBJMOD      DD      DSNAME=TAXES,
//                    DISP=(OLD,KEEP),...
//SYSLIN      DD      *
  CHANGE TAXROUT(STATETAX)
  INCLUDE OBJMOD
/*
```

As a result, you change the name of TAXROUT control section in the TAXES module to STATETAX. Any references to TAXROUT from other modules are not affected.

Example 2
A load module contains references to TAXROUT that you wish to change to STATETAX. If you define this module with DD statement named LOADMOD, you could change the external references at the same time you change the control section name as follows:

```
//OBJMOD      DD      DSNAME=TAXES,
//                    DISP=(OLD,DELETE),...
//LOADMOD     DD      DSNAME=LOADLIB,
//                    DISP=OLD,...
//SYSLIN      DD      *
  CHANGE TAXROUT(STATETAX)
  INCLUDE OBJMOD
  CHANGE TAXROUT(STATETAX)
  INCLUDE LOADMOD(INVENTRY)
/*
```

As a result you change both the control section name TAXROUT in the TAXES module and external reference TAXROUT in the INVENTRY module to STATETAX. Any references to TAXROUT from other modules are not affected.

9.3 ENTRY STATEMENT

The ENTRY statement specifies the symbolic name of the first instruction to be executed when you call the program by its module name for execution. **You must furnish an ENTRY statement whenever you re-link a load module.** If the Linkage Editor encounters more than one ENTRY statement for a single load module, it accepts the first statement as the main entry point and ignores all other ENTRY statements.

Format
The format of the ENTRY statement is:

```
ENTRY       externalname
```

externalname
is defined as either a control section name or an entry name in a linkage editor input module.

Placement
You can place an ENTRY statement before, between, or after object modules or other control statements. It must precede any NAME statement for the module.

Notes
- In an overlay program, the first instruction to be executed must be in the root segment.
- The external name specified must be the name of an instruction, not a data name, if the module is to be executed (as contrasted with being used as data).

Example
In the following example, the main entry point is INIT1:

```
//LOADLIB     DD      DSNAME=LOADLIB,
//                    DISP=OLD,...
//SYSLIN      DD      *
  ENTRY INIT1
  INCLUDE LOADLIB(READ, WRITE)
  .
  .
  .
  ENTRY READIN
/*
```

INIT1 must be either a control section name or an entry name in the linkage editor input. The Linkage Editor ignores the redundant READIN entry point.

9.4 EXPAND STATEMENT

The EXPAND statement lengthens control sections or named common sections by a specified number of bytes.

Format
The format of an EXPAND statement is:

```
EXPAND      name(xxxx) [,name(xxxx)]...
```

name
is the symbolic name of a common section or control section whose length is to be increased.

xxxx
is the decimal number of bytes you wish to add to the length of a common section. The Linkage Editor fills each expansion area with binary zeros, up to a maximum of 4096 bytes.

Placement
You can place an EXPAND statement before, between, or after other control statements or object modules. However, you must place it following the module containing the control or named common section to which it refers. If you have entered this control section or named common section with an INCLUDE statement, the EXPAND statement must follow the INCLUDE statement.

Notes
You should use expand with caution, so as not to increase a program beyond its design limitations. For example, if you add space to a control section beyond the range of its base register, that space is unusable.

Example
In the following example, EXPAND statements add a 250-byte patch area (initialized to zeros) at the end of control section CSECT1 and increase the length of named common section COM1 by 400 bytes.

```
//LKED        EXEC    PGM=JQAL
//SYSPRINT    DD      SYSOUT=A
//SYSUT1      DD      UNIT=SYSDA,
//                    SPACE=(TRK,(10,4))
//SYSLMOD     DD      DSNAME=PDSX,DISP=OLD
//SYSLIN      DD      DSNAME=&&LOADSET,
//                      DISP=(OLD,PASS),
//                      UNIT=SYSDA
//           DD         *
  EXPAND                CSECT1(250)
  EXPAND                COM1(400)
  NAME                  MOD1(R)
/*
```

9.5 IDENTIFY STATEMENT

The IDENTIFY statement writes descriptive data you supply into the CSECT Identification (IDR) records for a particular control section. You can also use IDENTIFY to associate system-supplied data with executable code.

Format

The format of the IDENTIFY statement is:

```
IDENTIFY    csectname('data')
            [,csectname('data')]...
```

csectname
is the symbolic name of the control section to be identified.

data
specifies up to 40 EBCDIC characters of identifying information. You may supply any information desired for identification purposes.

The rules of syntax for the operand field are:

1. No blanks or characters may appear between the left parenthesis and the leading quote, nor between the trailing quote and the right parenthesis.
2. The data field consists of 1 to 40 characters; therefore, a null entry must be represented, minimally, by a single blank.
3. Blanks may appear between the leading quote and the trailing quote. Each blank counts as 1 character toward the 40 character limit.
4. A single quote between the leading quote and the trailing quote is represented by 2 consecutive quotes. The pair of quotes counts as 1 character toward the 40-character limit.
5. Any other EBCDIC character may appear between the leading quote and the trailing quote.
6. An IDENTIFY statement may be continued onto additional cards; however, a whole operand must appear on a single card image and at least 1 whole operand must appear on each card image of the continued statement.
7. If the Linkage Editor finds a leading quote it processes all characters until it finds a trailing quote or reaches the 40-character limit.
8. Blanks may not appear between the CSECT name and the left parenthesis.
9. A blank following a left parenthesis terminates the operand field; a blank following a comma that terminates an operand terminates the operand field of that card image.

Placement
You can place an IDENTIFY statement before, between, or after other control statements or object modules. The IDENTIFY statement must follow the module containing the control section to be identified or the INCLUDE statement specifying the module.

Example
In the following example, IDENTIFY statements identify the source levels of a control section, a PTF application to a control section, and the functions of several control sections.

```
//LKED        EXEC    PGM=JQAL
//SYSPRINT    DD      SYSOUT=A
//SYSUT1      DD      UNIT=SYSDA,
//                    SPACE=(TRK,(10,5))
//SYSLMOD     DD      DSNAME=LOADSET,
//                    DISP=OLD
//OLDMOD      DD      DSNAME=OLD.LOADSET,
//                    DISP=OLD
//PTFMOD      DD      DSNAME=PTF.OBJECT,
//                    DISP=OLD
//SYSLIN      DD      *
  [input object deck for a control section named FORT]
  IDENTIFY    FORT('LEVEL 03')
  INCLUDE     PTFMOD(CSECT4)
  IDENTIFY    CSECT4('PTF99999')
  INCLUDE     OLDMOD(PROG1)
  IDENTIFY    CSECT1('I/O ROUTINE'),
              CSECT2('SORT ROUTINE'),
              CSECT3('SCAN ROUTINE')
/*
```

From these control statements, the Linkage Editor creates IDR records containing the following identification data:

- The name of the Linkage Editor that produced the load module, the Linkage Editor version and modification level, and the date of the current Linkage Editor processing of the module. This information is provided automatically.
- Your data describing the functions of several control sections in the module, as indicated on the third IDENTIFY statement.
- If your language translator supports IDR, the Identification records produced by the Linkage Editor also contain the name of the translator that produced the object module, its version and modification level, and data of compilation/assembly.

You can reference IDR records created by the Linkage Editor with the LISTIDR function of the JQNLIST service aid program, which is described in the **FACOM OS IV/F4 System Utility User's Guide**.

9.6 INCLUDE STATEMENT

The INCLUDE statement specifies sequential data sets and/or libraries that furnish additional input for the Linkage Editor, which processes them in the order in which they appear. However, the sequence of data sets and modules within the output load module does not necessarily follow the order of the INCLUDE statements.

Format
The format of the INCLUDE statement is:

```
INCLUDE    ddname [(membername [,...])]
           [,ddname[(membername [,...])]]...
```

ddname
is the name of a DD statement that describes a sequential or partitioned data set used as additional input to the Linkage Editor. For a sequential data set, you need specify only the DD name. For a partitioned data set, you must also specify at least one member name.

membername
is the name of or an alias for a member of the library defined in the specified DD statement. The membername must not be specified again within the DSNAME parameter of the DD statement.

Placement
You can place an INCLUDE statement before, between, or after object modules or other control statements.

Note
A NAME statement within a data set specified in an INCLUDE statement is invalid; the NAME statement is ignored. All other control statements are processed.

Example 1
In the following example, an INCLUDE statement specifies two data sets to be the input to the Linkage Editor:

```
//OBJMOD      DD      DSNAME=&&OBJECT,
//                    DISP=(OLD,DELETE)
//LOADMOD     DD      DSNAME=LOADLIB,
//                    DISP=SHR,...
              .
              .
              .
//SYSLIN      DD      *
  INCLUDE OBJMOD,LOADMOD(TESTMOD,READMOD)
/*
```

Note that you must supply a DD statement for every DD name in your INCLUDE statement.

Example 2
You could have used two separate INCLUDE statements in the preceding example, as follows:

```
INCLUDE      OBJMOD
INCLUDE      LOADMOD(TESTMOD,READMOD)
```

9.7 INSERT STATEMENT

The INSERT statement repositions a control section from its location within the input stream to the current segment in an overlay structure. However, the sequence of control sections within a segment is not necessarily determined by your INSERT statements; you use ORDER statements for sequencing within a segment.

If an operand of an INSERT statement is not already present in the external symbol dictionary, the Linkage Editor defines it as an external reference. If the reference has not been resolved at the end of primary input processing, Automatic Library Call attempts to resolve it.

Format
The format of the INSERT statement is:

```
INSERT      csectname,...
```

csectname
is the name of the control section to be repositioned. A particular control section can appear only once within a load module.

Placement
You must place each INSERT statement in the input sequence following the OVERLAY statement defining the origin of its segment. If you wish to insert the control section into the root segment, you must place the INSERT statement before the first OVERLAY statement.

Notes
Control sections positioned in a segment must contain all address constants to be used during execution unless:

- A-type address constants are located in a segment in the path.
- V-type address segment are located in the path. If an exclusive reference is made, the V-type address constant must be in a common segment.
- 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.

Example
The following INSERT (and OVERLAY) statements specify the overlay structure shown in Fig. 9.1:

```
//              EXEC     PGM=JQAL,PARM='OVLY,
//                          XREF,LIST'
                     .
                     .
                     .
//SYSLIN        DD          *
  INSERT CSA
  INSERT CSB
  OVERLAY ALPHA
  INSERT CSC,CSD
  OVERLAY ALPHA
  INSERT CSE
```

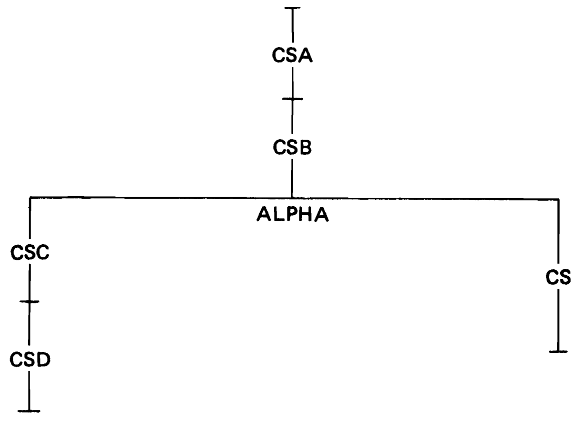


Fig. 9.1 Overlay structure for INSERT statement example

9.8 LIBRARY STATEMENT

You use a LIBRARY statement to specify:
- Additional automatic call libraries, which contain modules used to resolve external references found in the program.
- Restricted no-call function: external references you do not wish resolved by Automatic Library Call during this linkage edit.
- Never-call function: external references you do not wish resolved by Automatic Library Call mechanism during this or any subsequent linkage edit of this module.

You can furnish combinations of these functions in a single LIBRARY statement.

Format
The format of the LIBRARY statement is:

```
          ⎧ ddname(membername[,membername,...])            ⎫
LIBRARY   ⎨ (externalreference[,externalreference,...])    ⎬ ,...
          ⎩ *(externalreference[,externalreference,...])   ⎭
```

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

membername
is the name (or an alias) for a member of the specified library. You specify which members are used to resolve references.

externalreference
is an external reference that may remain unresolved after primary input processing and should not then be resolved by Automatic Library Call.

*****
indicates that the external reference is **never** to be resolved; if you omit the asterisk, the Linkage Editor leaves the corresponding reference unresolved only during its current run.

Placement
You can place a LIBRARY statement before, between, or after object modules or other control statements.

Notes
- If an unresolved external symbol is not a member name in the library specified, the external reference remains unresolved unless defined in another input module.
- If you specify NCAL you cannot also use the LIBRARY statement to specify additional call libraries.
- Members retrieved by Automatic Library Call are placed into the root segment of an overlay program, unless you reposition them with INSERT statements.
- Specifying an external reference for restricted no-call or never-call by means of the LIBRARY statement prevents the external reference from being resolved by automatic inclusion of the necessary module from an automatic call library. However, it does not prevent the external reference from being resolved if the module necessary to resolve the reference is specifically included or is included as part of an input module.

Example
The following example shows all three uses of the LIBRARY statement:

```
//             EXEC    PGM=JQAL,PARM='LET,
//                     XREF,LIST'
//TESTLIB      DD      DSNAME=TEST,
//                     DISP=SHR,...
                .
                .
                .
//SYSLIN       DD      *
  LIBRARY      TESTLIB(DATA,TIME),(FICACOMP),
               *(STATETAX)
/*
```

As a result, you utilize the DATA and TIME members from the TEST library to resolve external references. FICACOMP and STATETAX are not resolved; therefore, you must specify the LET option on your EXEC statement if the module is to be marked "executable." In addition, STATETAX will not be resolved in any subsequent reprocessing by the Linkage Editor.

9.9 NAME STATEMENT

Each NAME statement specifies the name of the load module created from preceding inputs and serves as a delimiter for input to this load module. As a delimiter the NAME statement allows you to create multiple load modules in one Linkage Editor job step. You can also use the NAME statement to indicate that the new load module replaces an identically-named module in the library.

Format
The format of the NAME statement is:

```
NAME        membername [(R)]
```

membername
is the name you assign to the load module.

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

Placement
You place each NAME statement after the last input module or control statement used for the corresponding output module.

Notes
- You must furnish any corresponding ALIAS statements before the NAME statement.
- A NAME statement found in a data set other than the primary input data set is invalid, and it is ignored.

Example
In the following example, the Linkage Editor creates two load modules, RDMOD and WRTMOD, in one job step:

```
//SYSLMOD     DD     DSNAME=AUXMODS,
//                   DISP=MOD,...
//NEWMOD      DD     DSNAME=&&WRTMOD,
//                   DISP=OLD
//SYSLIN      DD     DSNAME=&&RDMOD,
//                   DISP=OLD
//            DD     *
  NAME RDMOD(R)
  INCLUDE NEWMOD
  NAME WRTMOD
/*
```

As a result, your first module, RDMOD, replaces an identically-named module in the AUXMODS library, the second module, WRTMOD is added to the library.

9.10 ORDER STATEMENT

You use ORDER statements to select the sequence in which control sections or named common areas should appear in an output load module.

Format
The format of the ORDER statement is:

$$\text{ORDER} \quad \left\{\begin{array}{l}\text{common-area-name[(P)]}\\ \text{csectname[(P)]}\end{array}\right\},\ldots$$

common area name
is the name of the common area to be sequenced.

csectname
is the name of the control section to be sequenced.

(P)
indicates that the starting address of the control section or named common area is to be on a page boundary within the load module. The control sections or common areas are aligned on 4K page boundaries unless you specify the ALIGN2 attribute on your EXEC statement.

Placement
You can place an ORDER statement before, between, or after object modules or other control statements.

Notes
- You may name a control section or common area on only one ORDER statement. If you supply the same name more than once, it is ignored along with the balance of the control statement on which it appears.
- Control sections and common areas can appear in either the primary input, the Automatic Call Library, or both.
- If you change a control section or named common area by a CHANGE or REPLACE control statement and also wish to sequence it, you should specify the new name on the ORDER statement.

Example
In this example, you wish to sequence the control sections in the LDMOD module according to the sequence specified on your ORDER statements. Page-boundary alignments and control section sequences resulting from these statements are shown in Fig. 9.2, assuming each control section is 1K in length.

Notes
The control section name PART1 is changed by a CHANGE statement to FSTPART. The ORDER statement refers to the control section by its new name.

JCL and control statements

```
                    :
//SYSLMOD      DD        DSNAME=PVTLIB,DISP=OLD,...
//SYSLIN       DD        *
  ORDER        ROOTSEG(P),MAINSEG,SEG1,SEG2
  ORDER        SEG3(P),ENTRY1
  CHANGE       PART1(FSTPART)
  ORDER        FSTPART,SESECTA,SESECTB(P)
  INCLUDE      SYSLMOD(LDMOD)
```

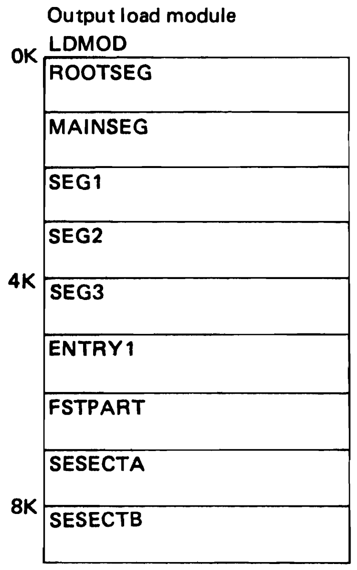


Fig. 9.2 Output load module for ORDER statement example

9.11 OVERLAY STATEMENT

You use an OVERLAY statement to designate the beginning of an overlay segment or overlay region. Since a segment or a region is not externally named, you identify it by giving its origin (or load point) a symbolic name, used only for Linkage-editing purposes, on a OVERLAY statement to signify the start of a new segment or region.

Format
The format of the OVERLAY statement is:

```
OVERLAY      symbol [(REGION)]
```

symbol
is the symbolic name you assign to the origin of a segment. This symbol is not related external symbols in a module.

(REGION)
specifies the origin of a new region.

Placement
The OVERLAY statement must precede (a) the first module of the next segment, (b) your INCLUDE statement specifying the first module of the segment, or (c) your INSERT statement specifying the control sections to be positioned in the segment.

Notes

- You must specify the OVLY option on your EXEC statement when you wish to use OVERLAY statements.
- Your sequence of OVERLAY statements should reflect the order of the segments in your overlay structure: top to bottom, left to right, and region by region.
- No OVERLAY statement should precede the root segment.

Example
The following OVERLAY and INSERT statements specify the overlay structure in Fig. 9.3.

```
//                 EXEC     PGM=JQAL,PARM='OVLY,
//                          XREF,LIST'
                    .
                    .
                    .
//SYSLIN           DD       DSNAME=&&OBJ,...
//                 DD       *
  INSERT SECT1
  OVERLAY ONE
  INSERT SECT2
  OVERLAY TWO
  INSERT SECT3
  OVERLAY TWO
  INSERT SECT4
  OVERLAY ONE
  INSERT SECT5, SECT6
  OVERLAY THREE(REGION)
  INSERT SECT7
  OVERLAY THREE
  INSERT SECT8
/*
```

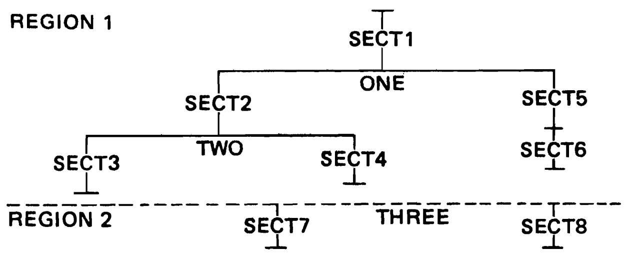


Fig. 9.3 Overlay structure for OVERLAY statement example

9.12 PAGE STATEMENT

A PAGE statement aligns a control section or named common area on a 4K page boundary in the load module. If you specify the ALIGN2 attribute on the EXEC statement for your linkage-edit job step and if you furnish one or more PAGE statements, the latter serve to align specified control sections or common areas on 2K page boundaries within your load module. Whether individual control sections are aligned on 2K or 4K boundaries, the OS IV/F4 Supervisor will load the associated module on a 4K page boundary; that is, address 0 of the module will be loaded at a virtual-storage address which is a multiple of 4096.

Format
The format of the PAGE statement is:

$$\text{PAGE}\qquad \begin{Bmatrix}\text{common-area-name}\\ \text{csectname}\end{Bmatrix},\ldots$$

common area name
is the name of a common area to be aligned on a page boundary.

csectname
is the name of a control section to be aligned on a page boundary.

Placement
The PAGE statement can be placed before, between, or after object modules or other control statements.

Notes
- If you change a control section or named common area by a CHANGE or REPLACE statement and also wish to insure page alignment you should specify the new name in the PAGE statement.
- The control sections and common areas named as operands can appear in the primary input and/or the automatic call library.

JCL and control statements

```
//LKED        EXEC      PGM=JQAL,
                        PARM='ALIGN2,...'
                 :
//SYSLMOD     DD        DSNAME=PVTLIB,
                        DISP=OLD,...
//SYSLIN      DD        *
  PAGE        ALIGN,BNDRY4K, EIGHTK
  INCLUDE     SYSLMOD(LDMOD)
/*
```

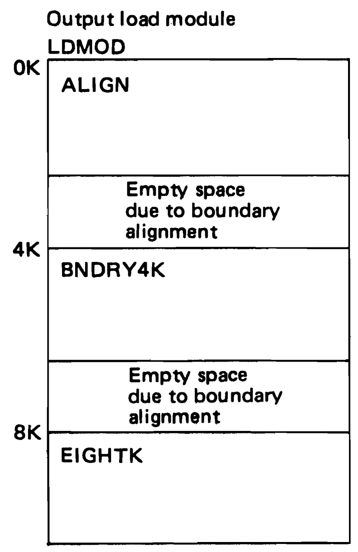

Output load module
LDMOD
0K
ALIGN
Empty space due to boundary alignment
4K
BNDRY4K
Empty space due to boundary alignment
8K
EIGHTK

Fig. 9.4 Output load module for PAGE statement example

Example
In this example, you wish to align the control sections in the LDMOD module on page boundaries, as specified in the following PAGE statement:

```
PAGE          ALIGN,BNDRY4K,EIGHTK
```

Appropriate JCL and Linkage Editor control statements, as well as the output load module, are shown in Fig. 9.4, assuming each control section is 3K bytes in length.

9.13 REPLACE STATEMENT

The REPLACE statement specifies one of the following:
- Replacement of one control section with another.
- Deletion of a control section.
- Deletion of an entry name.

A REPLACE statement can specify more than one function.

When you replace a control section, the Linkage Editor changes all references within the input module to the old control section to point to the new control section. Any external references to the old control section from other modules are unresolved unless changed.

When you delete a control section, the Linkage Editor deletes the control section name from the external symbol dictionary unless references are made to the control section within the input module. If there are any such references, the Linkage Editor changes the control section name to an external reference. External references from other modules to a deleted control section also remain unresolved.

When deleting an entry name, the Linkage Editor changes it to an external reference if there are any references to it within the same input module.

Format
The format of the REPLACE statement is:

$$\text{REPLACE}\qquad \begin{Bmatrix}\text{csectname-1[(csectname-2)]}\\ \text{entryname}\end{Bmatrix},\ldots$$

csectname
is the name of a control section. If you furnish only **csectname-1**, the Linkage Editor deletes the control section; if you also furnish **csectname-2**, the Linkage Editor replaces the first control section with the second.

entryname
is the entry name to be deleted.

Placement
Your REPLACE statement must immediately precede either (1) the module containing the control section or entry name to be replaced or deleted, or (2) your INCLUDE statement specifying the module. The scope of the REPLACE statement spans the immediately-following object module or load module. The END record in the immediately-following object module, or the end-of-module indication in the load module, terminates the action of the REPLACE statement.

Notes
- The Linkage Editor does not delete unresolved external references from the output module, even if a deleted control section contains the only reference to a symbol.
- When you wish to replace some—but not all—control sections of a separately-assembled module, A-type address constants that refer to a deleted symbol will be incorrectly resolved unless the entry name is at the same displacement from the origin in both the old and the new control sections.
- If you misspell a control section name on your REPLACE statement, the control section will not be replaced or deleted. You can use Linkage Editor output such as the cross-reference listing and module map to verify each change.

Example
In the following example, assume that the INT7 control section is in the LOANCOMP member and that the INT8 control section, which is to replace INT7, is in the &&NEWINT data set. Also assume that you wish to delete the PRIME control section in the LOANCOMP member.

```
//NEWMOD        DD      DSNAME=&&NEWINT,
//                      DISP=(OLD,DELETE)
//OLDMOD        DD      DSNAME=PVTLIB,
//                      DISP=OLD,...
//SYSLIN        DD      *
   ENTRY MAINENT
   INCLUDE NEWMOD
   REPLACE INT7(INT8),PRIME
   INCLUDE OLDMOD(LOANCOMP)
/*
```

As a result, the Linkage Editor deletes INT7 from the input module described by the OLDMOD DD statement, and replaces INT7 with INT8. All references to INT7 in the input module now refer to INT8. Any reference to INT7 from other modules remain unresolved. Control section PRIME is deleted; the control section name is also deleted from the external symbol dictionary if there are no references to PRIME in LOANCOMP.

9.14 SETCODE STATEMENT

A SETCODE statement assigns an authorization code to an output load module, which is placed in the directory entry for the module.

Format
The format of the SETCODE statement is as follows:

```
   SETCODE        AC(authorization-code)
```

authorization-code
is 1 to 8 decimal digits specifying a value from 0 to 255.

Placement
You can place a SETCODE statement before, between, or after object modules or other control statements. It must precede the NAME statement for the module if one is present.

Notes
If you assign an authorization code with a SETCODE statement, it overrides any authorization code assigned by an AC parameter in the PARM field of your EXEC statement.

If the Linkage Editor encounters more than one SETCODE statement while editing a load module, it uses the last valid authorization code it encounters.

The operand "AC( )" results in an authorization code of zero (0).

Example
In the following example, the Linkage Editor assigns an authorization code of 1 to the MOD1 load module.

```
//LKED          EXEC    PGM=JQAL
//SYSPRINT      DD      SYSOUT=A
//SYSUT1        DD      UNIT=SYSDA,
//                      SPACE=(TRK,(10,5))
//SYSLMOD       DD      DSNAME=SYS1.LINKLIB,
//                      DISP=OLD
//SYSLIN        DD      DSNAME=&&LOADSET,
//                      DISP=(OLD,PASS)
//                      UNIT=SYSDA
//              DD      *
   SETCODE              AC(1)
   NAME                 MOD1(R)
/*
```

9.15 SETSSI STATEMENT

You use a SETSSI statement to place System Status Index information into the directory entry for the output module.

Format

The format for the SETSSI statement is:

```
SETSSI          xxxxxxxx
```

xxxxxxxx

represents eight hexadecimal characters (0 – 9, A – F) to be placed in the 4-byte System Status Index of the directory entry.

Placement

You can place a SETSSI statement before, between, or after object modules or other control statements. It must precede the NAME statement for the module, if one is present.

Notes

You must provide a SETSSI statement whenever you re-process a Fujitsu module with the Linkage Editor. If you omit the statement, no System Status Index information is retained.

CHAPTER 10
LOADER FUNCTIONS AND FACILITIES

The Loader combines basic editing and loading functions of the Linkage Editor and OS IV/F4 Supervisor, respectively, in one job step. Therefore, the **load** function is equivalent to the **link edit-go** function. You can use the Loader for compile-load-go and load-go jobs.

The Loader will accept object modules produced by any language processor, together with load modules produced by the Linkage Editor. Optionally, it will search a call library (SYSLIB) and/or resident **link-pack area** (LPA) to resolve external references. The Loader does not produce load modules for program libraries.

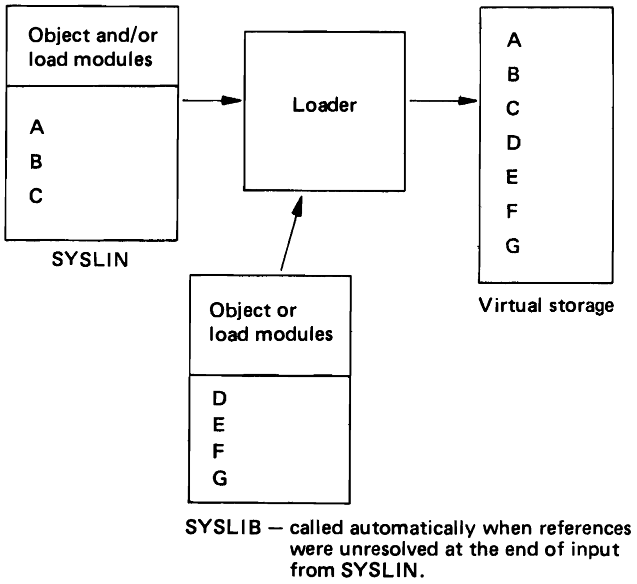
The functional characteristics, compatibility and restrictions, performance considerations, and storage considerations of the Loader are described in the following sections. For additional details on program loading, you should consult **FACOM OS IV/F4 Supervisor Functions and Facilities.**

10.1 FUNCTIONAL CHARACTERISTICS

The Loader combines the following basic functions of the Linkage Editor and the **Program Fetch** function of the OS IV/F4 Supervisor:

1. Resolution of external references between program modules.
2. Optional inclusion of modules from a call library (SYSLIB), a link pack area, (LPA) or both (Figs. 10.1 and 10.2). The Loader includes modules from a call library or the LPA upon your request, if any external references remain unresolved after analyzing the primary input. If you request both call-library and LPA resolution, the Loader will search the latter first.
3. Automatic deletion of duplicate copies of program modules (Fig. 10.3). The first copy is loaded, and all succeeding requests use that copy.
4. Relocation of all address constants so that the Loader can pass control directly to the selected entry point in virtual storage.

Diagnostic messages produced by the Loader are similar to those of the Linkage Editor.



Object and/or load modules
A
B
C
SYSLIN
Loader
A
B
C
D
E
F
G
Virtual storage
Object or load modules
D
E
F
G
SYSLIB – called automatically when references were unresolved at the end of input from SYSLIN.

Fig. 10.1 Loader processing – SYSLIB resolution

10.2 COMPATIBILITY AND RESTRICTIONS

The Loader accepts the same basic input as the Linkage Editor:

1. You can submit any object modules that can be processed by the Linkage Editor to the Loader.
2. You can submit any load modules produced by the Linkage Editor to the Loader, except those edited with the "not executable" attribute.

The Loader supports the following Linkage Editor options: MAP, LET, NCAL, DYNA, SIZE, and TERM. No other Linkage Editor options nor attributes are supported; if you furnish them, the Loader will ignore them, i.e., they will not be considered as errors. The Loader will print appropriate messages on the SYSLOUT data set indicating that these options and/or attributes are not supported. You can specify supported options in the PARM field of your EXEC

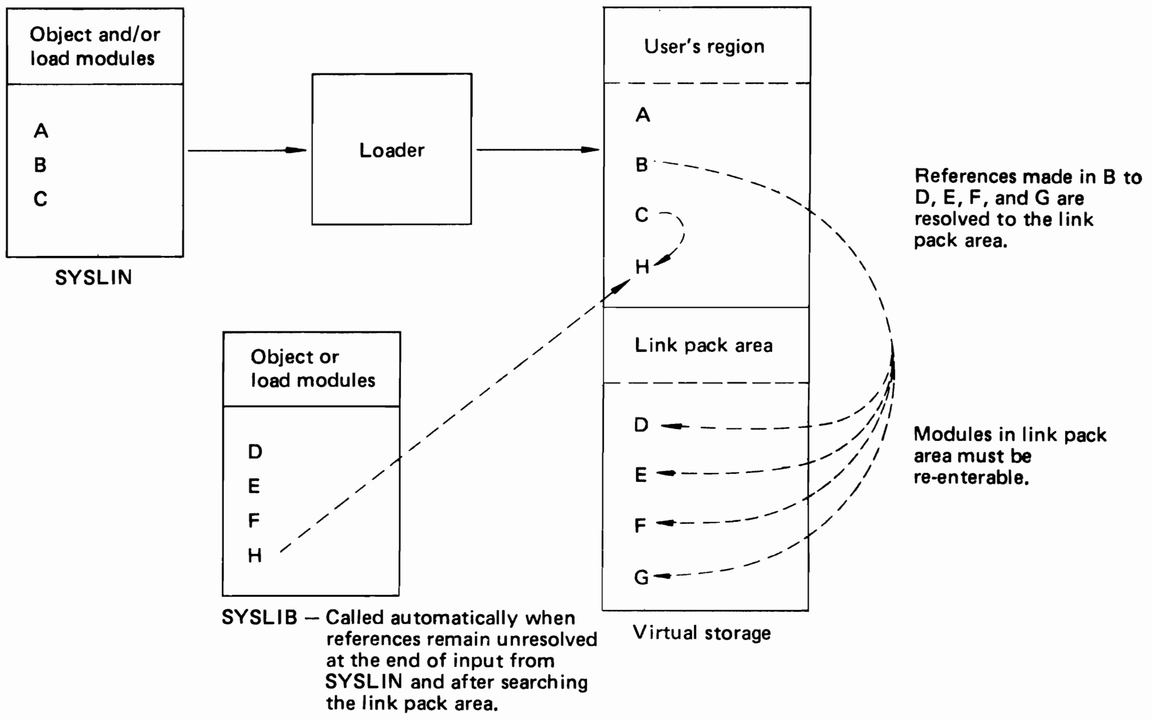

Object and/or load modules
A
B
C
SYSLIN
Loader
User's region
A
B
C
H
Link pack area
D
E
F
G
Virtual storage
References made in B to D, E, F, and G are resolved to the link pack area.
Modules in link pack area must be re-enterable.
Object or load modules
D
E
F
H
SYSLIB – Called automatically when references remain unresolved at the end of input from SYSLIN and after searching the link pack area.

Fig. 10.2 Loader processing – link pack area and SYSLIB resolution

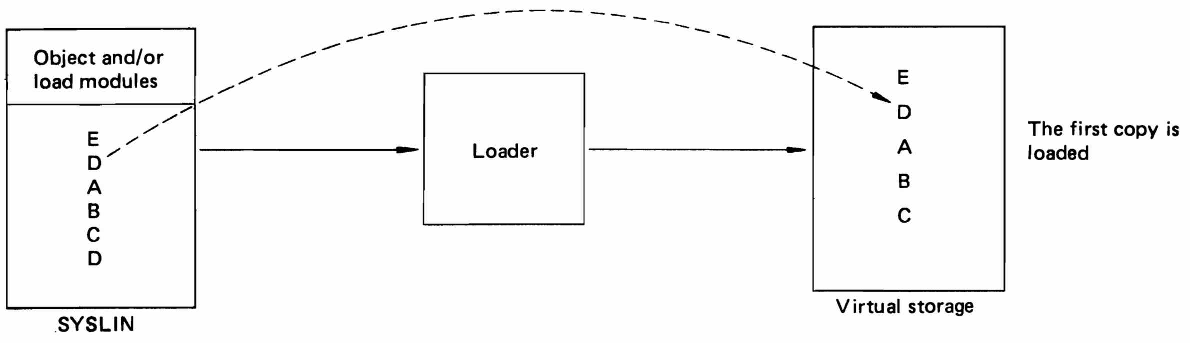


Fig. 10.3 Loader processing – automatic editing

statement or with LINK, ATTACH, LOAD, or XCTL macro instruction. In addition to supported Linkage Editor options, the Loader provides several other options. Loader options are described in Section 10.3.1.

The Loader does not process Linkage Editor control statements (INCLUDE NAME, OVERLAY, etc.). If you furnish them, they will not be treated as errors; instead, the Loader will furnish messages on the SYSLOUT data set indicating that the control statements are not supported.

The Loader and Linkage Editor follow the same input conventions. In contrast to the Linkage Editor, the Loader can accept load modules from the SYSLIN dats set; it also accepts already-loaded object modules in virtual storage.

The Loader needs no auxiliary storage space comparable to the SYSUT1 data set of the Linkage Editor.

Time sharing system (TSS)
When you request the Loader while using the OS IV/F4 TSS, you actually invoke the **Loader Prompter,** a program that interfaces your terminal to OS IV/F4 and its Loader. Under TSS, you define Loader options and data sets via the LOADGO command. Complete procedures for using the LOADGO command to load and execute an object module may be found in the **FACOM OS IV/F4 TSS Terminal User's Guide.**

10.3 INPUT FOR THE LOADER

The input deck for the Loader must contain job control language statements for the Loader and—as necessary—for your application program if it is to be executed after loading (Fig. 10.4).

```
//name      JOB    parameters               (optional)
//name      EXEC   PGM=LOADER,PARM=(parameters)
//SYSLIN    DD     parameters
//SYSLIB    DD     parameters               (optional)
//SYSLOUT   DD     parameters               (optional)
//SYSTERM   DD     parameters               (optional)

//        (optional DD statements and data
//        required for loaded program)
```

Fig. 10.4 Input deck for the loader – basic format

Only the EXEC and SYSLIN statements are required for a Loader step. The JOB statement is required if the Loader is the first step in the job. Aliases for "LOADER" in the EXEC statement are "JQBMLGO" and "JQBLDRGO".

10.3.1 EXEC Statement

You use an EXEC statement to invoke the Loader and specify its options. You specify Loader and loader-program options in the PARM field of this EXEC statement. The PARM field must have the following format:

,PARM='[loader-option[,...]]
[/loaded-program option[,...]]'

Note that you must separate any loaded-program options from your Loader options by a slash (/). If you furnish no Loader options, your loaded-program options must begin with a slash. You may omit the entire PARM field if you need no Loader nor loaded-program options.

The Loader options are:

- MAP
  The Loader produces a map of the loaded program that lists external names and their absolute storage addresses on the SYSLOUT data set. If you omit a SYSLOUT DD statement, OS IV/F4 ignores any MAP option you request.
- NOMAP
  A map is not produced.
- RES
  The Loader performs an automatic search of the **link pack area** (LPA) list after processing the primary input (SYSLIN) and before searching the SYSLIB data set. If you specify this option the Loader automatically selects the CALL option.
- NORES
  The Loader omits any automatic search of the LPA list.
- CALL
  The Loader performs an automatic search of the SYSLIB data set. If you omit a SYSLIB DD statement, this option is ignored.
- NOCALL or NCAL
  The Loader omits any automatic search of the SYSLIB data set. If you select this option, the order automatically selects the NORES option.
- LET
  The Loader will try to execute the object program even if it detects one or more severity-2 error conditions. (A **severity-2 error condition** is one that often makes execution of a loaded program impossible.)
- NOLET
  The Loader will not try to execute the loaded program if it detects at least one severity-2 error condition.
- SIZE=size
  Specifies the size, in bytes, of the virtual-storage region that you furnish to the Loader explicitly or (normally) implicitly.
- EP=name
  Specifies the entry point of the loaded program. You must specify an external name for this parameter if the entry point of the loaded program is in an input load module. For FORTRAN and PL/I, these entry points must be MAIN and JXXNTRY, respectively, unless you change the compiler options.
- NAME=name
  Specifies the name you use to identify the loaded program to the system. If omitted, the default name is "**GO".
- PRINT
  The Loader displays informational and diagnostic messages on the SYSLOUT data set.
- NOPRINT
  The Loader does not open the SYSLOUT data set, and it omits all informational and diagnostic messages.
- TERM
  The Loader directs any numbered diagnostic messages to the SYSTERM data set. Although intended to be used when operating under the Time Sharing System (TSS), the SYSTERM data set can replace or supplement the SYSLOUT data set at any time. If you omit the SYSTERM DD statement, the Loader ignores this option.
- NOTERM
  The Loader does not display numbered diagnostic messages on the SYSTERM data set.
- ALIAS
  The name you furnish with the EP parameter is an alias for the desired load module.
- DYNA
  The modules are to be loaded as a dynamic link structure, as described in Chapter 7.

The default options are: NOMAP, RES, CALL, NOLET, SIZE=100K, PRINT, NAME=**GO and NOTERM.

In the following examples of the EXEC statement, X and Y are parameters required by the loaded program.

```
//          EXEC PGM=JQBMLGO
//ABC       EXEC PGM=LDRGO,
//                   PARM='MAP,
//                   EP=FIRST/X,Y'
//LOAD1     EXEC PGM=LOADER,
//                   PARM='/X, Y'
//LOAD2     EXEC PGM=LOADER,
//                   PARM=NOPRINT
//LOAD3     EXEC PGM=LOADER,
//                   PARM=(MAP, LET)
//LOAD4     EXEC PGM=LOADER,
//                   PARM='NAME=NEWPROG,
//                   TERM, NOPRINT'
```

For further details on how to code the EXEC statement, you should refer to the **FACOM OS IV/F4 Job Control Language** or the **FACOM OS IV/F4 Job Control Language User's Guide.**

10.3.2 DD Statements

The Loader uses up to four DD statements named SYSLIN, SYSLIB, SYSLOUT, and SYSTERM. You must furnish a SYSLIN statement for every Loader job; the other three DD statements are optional.

The following considerations apply to the DCB parameters for SYSLIN, SYSLIB, and SYSLOUT:

- For better performance, you should explicitly specify the BLKSIZE and BUFNO subparameters.
- If you omit BUFNO, the Loader assumes BUFNO=2.
- If you specify RECFM=U, the Loader assumes BUFNO=2 and ignores any BLKSIZE and LRECL subparameters.
- RECFM=V is not accepted.
- RECFM=FBSA is always assumed for SYSLOUT.
- If you omit RECFM, the Loader assumes RECFM=F for SYSLIN and SYSLIB.
- If you omit BLKSIZE, the Loader assigns the LRECL value for the block size.
- The Loader assumes LRECL=121 for SYSLOUT unless it is operating under the Time Sharing System (TSS), in which case it assumes LRECL=81.
- If you omit LRECL, the Loader assumes LRECL=80 for SYSLIN and SYSLIB.
- If you use OPTCD=C to specify chained scheduling, you must allocate an additional 2K (2048 bytes) of virtual storage in your region if necessary Data Management routines are not resident.

Note: The SYSTERM data set always comprises unblocked 81-character records with BUFNO=2 and RECFM=FSA. Because these values are fixed, you need not furnish a DCB parameter.

In addition to the DD statements required for the Loader, you must include any DD statements and data required by the loaded program in your input deck.

SYSLIN

The SYSLIN DD statement defines the input data for the Loader, which can be object modules produced by a language translator, load modules produced by the Linkage Editor, or both. The data set defined by the SYSLIN statement can be sequential data sets, members of a partitioned data set, or both. The DSNAME parameter for a partitioned data set must also furnish the member name:

DSNAME=dsname(membername).

You can concatenate more than one module in your SYSLIN stream.

The following are examples of the SYSLIN statement. The first example defines a member of a previously-cataloged partitioned data set:

```
//SYSLIN    DD    DSNAME=OUTPUT.FORT(MOD12),
//                DISP=OLD, DCB=BLKSIZE=3200
```

The second example defines a sequential data set on magnetic tape:

```
//SYSLIN    DD    DSNAME=PROG15, UNIT=2400,
//                DISP=(OLD, KEEP),
//                VOLUME=(PRIVATE, RETAIN,
//                SER=MCS167)
```

The third example defines a data set which was the output of a previous step in the same job:

```
//SYSLIN    DD    DSNAME=*.COBOL.SYSLIN,
//                DISP=(OLD, DELETE)
```

The fourth example shows the concatenation of three data sets. The first two data sets are members of different partitioned data sets; the first is an object module and the second is a load module. The third data set is in the input stream following a SYSLIN DD statement, as described in Section 10.3.3.

```
//SYSLIN    DD    DSNAME=PGMLIB.SET1(RFS1),
//                DISP=OLD,
//                DCB=(BLKSIZE=3200, RECFM=FB)
//          DD    DSNAME=PGMLIB.SET2(ABC5),
//                DISP=OLD, DCB=RECFM=U
//          DD    DDNAME=SYSIN
```

SYSLIB

The SYSLIB data set contains Fujitsu-supplied or installation library routines you wish to include in your loaded program. The Loader searches this data set when it detects unresolved references after processing SYSLIN and optionally searching the link pack area.

The SYSLIB data set resolves an external reference whenever it is as follows: (1) a member name or an alias of a module in the data set, and (2) defined as an external name in the external symbol dictionary of the module with that name. If the unresolved external reference is a member name or an alias in the library, but is not an external name in that member, the Loader processes the member but leaves the external reference unresolved unless subsequently defined.

The data set defined by your SYSLIB DD statement must be a partitioned data set that contains object modules or load modules, but not both. You can use concatenation to include additional partitioned data sets in SYSLIB. All concatenated data sets must contain the same type of modules (object or load).

The following are examples of the SYSLIB DD statement. The first example defines a cataloged partitioned data set that can be shared by other steps:

```
//SYSLIB     DD     DSNAME=SYS1.COBLIB, DISP=SHR
```

The second example shows the concatenation of two data sets:

```
//SYSLIB     DD     DSNAME=SYS1.PL1BASE,DISP=SHR
//           DD     DSNAME=LIBMOD. MATH,
//                  DISP=OLD
```

SYSLOUT

The Loader writes into your SYSLOUT data set any error or warning messages; it also displays there the optional map of external references if you request it as described in Section 10.5. The data set defined by this DD statement must be sequential. You can furnish a DCB parameter to specify its blocking factor (BLKSIZE) or number of buffers (BUFNO).

The following are examples of the SYSLOUT DD statement. The first example specifies the system output unit:

```
//SYSLOUT   DD  SYSOUT=A
```

The second example defines a sequential data set on a F651 printer:

```
//SYSLOUT   DD  UNIT=F651,
//              DCB=(BLKSIZE=121,BUFNO=4)
```

SYSTERM

The SYSTERM DD statement defines a data set used only for numbered diagnostic messages. When you use the Loader under the OS IV/F4 Time Sharing System (TSS), your SYSTERM DD statement defines your terminal output data set. You can also use SYSTERM at any time to replace or supplement the SYSLOUT data set. Because the Loader does not open the SYSTERM data set unless it must issue a diagnostic message, using SYSTERM instead of SYSLOUT can reduce Loader processing time.

If you use SYSTERM rather than SYSLOUT, the numbered messages in the SYSTERM data set are your only diagnostic output. If you use SYSTERM to supplement SYSLOUT, the numbered messages appear in both data sets.

The DCB parameters for SYSTERM are fixed and need not be specified. The SYSTERM data set always consists of unblocked 81-character records with BUFNO=2 and RECFM=FSA.

The following example shows a SYSTERM DD statement specifying the standard JES output stream:

```
//SYSTERM   DD   SYSOUT=A
```

10.3.3 Submitting Data to a Loaded Program

Data for your loaded program can follow your Loader data in the JES input stream. You define such loaded-program data by a DD statement following the Loader data.

Fig. 10.5 shows how you can load a previously-compiled FORTRAN problem program. The program to be loaded (**loader data**) follows the SYSLIN DD statement. The loaded-program data follows the FT05F001 DD statement.

```
//LOAD        JOB     MSGLEVEL=1
//LDR         EXEC    PGM=LOADER,PARM=MAP
//SYSLIB      DD      DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLOUT     DD      SYSOUT=A
//FT06F001    DD      SYSOUT=A
//SYSLIN      DD      *
          [Loader data]
/*
//FT05F001    DD      *
          [Loaded program data]
/*
```

Fig. 10.5 Loader and loaded-program data in the JES input stream

10.4 DYNAMICALLY INVOKING THE LOADER

You can invoke the Loader by either its program name, JQBMLGO, or either of its two standard aliases, JQBLDRGO or LOADER. You can request dynamic loading via one of the following macro instructions:

```
                 ⎧ LINK   ⎫  EP=loadername,
[symbol]         ⎨        ⎬  PARAM=(optionlist
                 ⎩ ATTACH ⎭  [,ddname list]),
                             VL=1

[symbol]         ⎧ LOAD ⎫    EP=loadername
                 ⎩ XCTL ⎭
```

EP=LOADER

specifies the symbolic name of a standard alias for the Loader.

PARAM=(optionlist [ddname list])
specifies, as a sublist, address parameters to be passed to the Loader. The first full-word in the address parameter list contains the address of the option list for the Loader and/or loaded program. The second full-word contains the address of the list of alternative DD names. If you use all standard DD names (the normal situation), you can omit this list.

Option list
points to a variable-length list containing options for both the Loader and your loaded program. You must furnish this address even if you furnish no options.

Your option list must begin on a halfword boundary. The two high-order bytes count the number of bytes in the remainder of the list. If you furnish no options, your count should be zero.

The option list is free-form, with options for the Loader and your loaded program separated by a slash (/) and with each option 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 DD names for Loader data sets. If you use all standard DD names you can omit this operand.

The format of this list is identical to the format of the comparable list for invoking the Linkage Editor, described in Section 8.4:

Entry	Alternate Name For:
1	SYSLIN
2	(not applicable)
3	(not applicable)
4	SYSLIB
5	(not applicable)
6	SYSLOUT
7–11	(not applicable)
12	SYSTERM

VL
specifies that the sign bit is to be set to 1 in the last fullword of the address parameter list.

Fig. 10.6 shows an Assembler-language program using the LINK macro instruction to invoke the Loader.

If desired, you can use the Loader to process a program but not execute it. To invoke just the portion of the Loader that processes input modules, specify either the JQBLOAD or JQBLOADR in your LOAD macro instruction, as shown in Fig. 10.7.

The first CALL macro instruction passes control to the Loader, and the second CALL macro instruction requests execution of the loaded program.

JQBLAD both loads and identifies a program, returning the address of an 8-character name in register 1. You can furnish this name with an ATTACH, LINK, LOAD, or XCTL macro instruction to invoke the loaded program. If another of your programs attaches a loaded program, you should avoid specifying SZERO=NO in your ATTACH macro instruction. If you must specify SZERO=NO, your program should issue a LOAD macro instruction for the loaded program before issuing an ATTACH macro instruction. Likewise, your program should issue a DELETE macro instruction for the loaded program after it returns from operating as an attached subtask.

JQBLOADR loads your program but does not identify it. JQBLOADR returns the entry point of your loaded program in register 0. Register 1 points to two full words: the first points to the beginning of the loaded program; the second contains the size of the loaded program. You can use these location and size values in a FREEMAIN macro instruction to free storage occupied by the loaded program when you no longer need it.

Fig. 10.7 shows an Assembler-language program that uses the LOAD and CALL macro instructions to

```
          SVG      (14,12),=A(SAVEAREA)     initialize save
           :                                registers and point
           :                                to new save area
           :
          LINK     EP=LOADER,PARAM=(PARM),VL=1
           :
          L        13,4(13)
          RETURN   (14,12),T
           :
          DS       0H
PARM      DC       AL2(LENGTH)              length of options
OPTIONS   DC       C'NOPRINT,CALL/X,Y,Z'    loader and loaded
LENGTH    EQU      *-OPTIONS                program options
SAVEAREA  DS       18F                      save area
           :
          END
```

Fig. 10.6 Using the LINK macro instruction to invoke the loader

```
                SVG        14,12,=A(SAVEAREA)     initialize save registers and
                 :                                point to new save area
                LOAD       EP=JQBLOADR            load the Loader
                LR         15,0                   get its entry point address
                CALL       (15),(PARM1),VL        invoke the Loader
                 :
                LR         7,15                   save return code
                LR         5,0                    save entry to loaded program
                LR         6,1                    save pointer to list containing
*                                                 start address and length
                DELETE     EP=JQBLOADER           delete Loader
                CH         7,=H'4'                verify successful loading
                BH         FREE                   negative branch
                LR         15,5                   loading successful-get entry
*                                                 point address for CALL
                CALL       (15),(PARM2),VL        invoke loaded program
                 :
FREE            L          0,4(6)                 get length into register 0
                L          1,0(6)                 get start address
                FREEMAIN   R,LV=(0),A=(1)         delete loaded program
                 :
                L          13,4(13)
                RETURN     (14,12),T
                DS         0H
PARM1           DC         AL2(LENGTH1)           length of Loader options
OPTIONS1        DC         C'NOPRINT,CALL'        Loader options
LENGTH1         EQU        *-OPTIONS1
                DS         0H
PARM2           DC         AL2(LENGTH2)           Length of loaded-program options
OPTIONS2        DC         C'X,Y,Z'               loaded-program options
LENGTH2         EQU        *-OPTIONS2
SAVEAREA        DS         18F                    save area
                 :
                END
```

Fig. 10.7 Using LOAD and CALL macro instructions to invoke JQBLOADR (Loading without identification)

```
                SVG        14,12,=A(SAVEAREA)     initialine save registers and
                 :                                point to new save area
                LOAD       EP=JQBLOAD             load the Loader
                LR         15,0                   get its entry point address
                CALL       (15),(PARM1),VL        invoke the Loader
                LR         7,15                   save the return code
                MVC        PGMNAM(8),0(1)         save program name
                DELETE     EP=HEWLOAD             delete the Loader
                CH         7,=H'4'                verify successful loading
                BH         ERROR                  negative branch
                LINK       EPLOC=PGMNAM,
                 :         PARM=(PARM2),VL=1      loading successful, invoke program
                L          13,4(13)
                RETURN     (14,12),T
                DS         0H
PARM1           DC         AL2(LENGTH1)           length of Loader options
OPTIONS1        DC         C'MAP'                 Loader options
LENGTH1         EQU        *-OPTIONS1
                DS         0H
PARM2           DC         AL2(LENGTH2)           length of loaded-program options
OPTIONS2        DC         C'X,Y,Z'               loaded-program options
LENGTH2         EQU        *-OPTIONS2
SAVEAREA        DS         18F                    save area
PGMNAM          DS         2F                     program name
                 :
                END
```

Fig. 10.8 Using LOAD and CALL macro instructions to invoke JQBLOAD (Loading with identification)

invoke JQBLOADR. Fig. 10.8 shows an Assembler-language program that uses the LOAD and CALL macro instructions to invoke JQBLOAD.

For further information on the use of these macro instructions, refer to the **FACOM OS IV/F4 Supervisor Macro Instructions Reference Manual.**

10.5 PRINTED OUTPUTS

Loader printed outputs comprise a collection of diagnostic and error messages plus an optional storage map of the loaded program. This output is created in the data set defined by your SYSLOUT and SYSTERM DD statements. If you omit these, the Loader prints no outputs.

SYSLOUT output includes a loader heading and a list of options and defaults requested in the PARM field of your EXEC statement. SIZE is the size region obtained, not necessarily the size you requested in the PARM field. The Loader prints various messages when it detects errors. After it finishes processing, it also prints an explanation of each error. Loader error messages are similar to those of the Linkage Editor and are listed in Appendix 2.

SYSTERM output includes only numbered warning and error messages. After the Loader completes its processing, it writes an explanation of each error message onto SYSTERM.

The storage map includes the name and absolute address of each control section and entry point in the loaded program. Each map entry marked with an asterisk (*) comes from the data set specified on the SYSLIB DD statement. Two asterisks (**) indicate the entry was found in the link pack area. The Loader writes its storage map as it processes input modules; hence, all map entries appear in the sequence ESD entries are encountered. It also displays the total size and highest address (in virtual storage) of the loaded program.

Return code	Loader return code[1]	Loaded program return code	Conclusion or meaning
0	0	0	Program loaded successfully, and execution of the loaded program was successful.
	4 8(LET)	0 0	The Loader found a condition that may cause and error during execution, but no error occurred during execution of the loaded program.
4	0	4	Program loaded successfully, and an error occurred during execution of the loaded program.
	4 8(LET)	4 4	The Loader found a condition that may cause an error during execution, and an error did occur during execution of the loaded program.
8	0	8	Program loaded successfully, and an error occurred during execution of the loaded program.
	4 8(LET)	8 8	The Loader found a condition that may cause an error during execution, and an error did occur during execution of the loaded program.
	8		The Loader found a condition that could make execution impossible. The loaded program was not executed.
12	0	12	Program loaded successfully, and an error occurred during execution of the loaded program.
	4 8(LET)	12 12	The Loader found a condition that may cause an error during execution, and an error did occur during execution of the loaded program.
	12		The Loader could not load the program successfully, execution impossible.
16	0	16	Program loaded successfully, but the loaded program found a terminating error.
	4 8(LET)	16 16	The Loader found a condition that may cause an error during execution, and a terminating error was found during execution of the loaded program.
	16		The Loader could not load the program at all, and execution is clearly impossible.

1 Error diagnostics (SYSI OUT and/or SYSTEM data set) show the severity of errors found by the Loader.

Fig. 10.9 Return codes

10.6 RETURN CODES

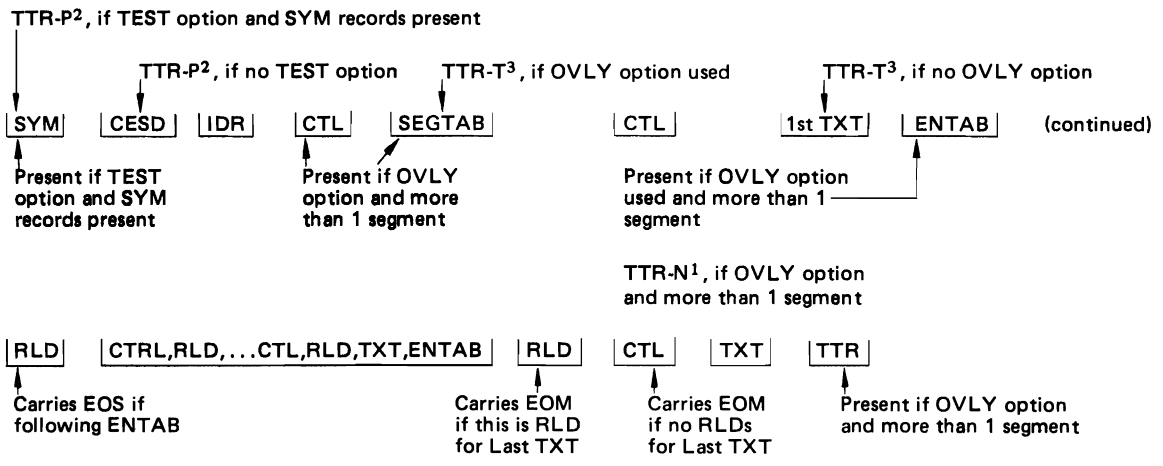
The return code of a Loader step summarizes the return codes from loading **and** from the loaded program after it has executed.

The return code indicates whether errors occurred during loading or execution. You can test the return code with a COND parameter on your JOB statement and/or with COND parameter on your EXEC parameters on your EXEC statements for succeeding job steps. For details, see the publication **FACOM OS IV/F4 Job Control Language Reference Manual.** Fig. 10.9 shows the return codes from the Loader step, followed by corresponding return codes internal to the Loader and those of the loaded program.

APPENDIX 1: LOAD MODULE FORMAT

The format of a load module built by the Linkage Editor is shown in Fig. A1.1.

When writing the output load module to the SYSLMOD data set, the Linkage Editor never uses the Track Overflow feature. When moving or copying load modules, you should not specify the Track Overflow feature when creating the target data set, as the OS IV/F4 Supervisor may encounter an erroneous format when fetching the load modules for execution.


TTR-P2, if TEST option and SYM records present
TTR-P2, if no TEST option
TTR-T3, if OVLY option used
TTR-T3, if no OVLY option
SYM
CESD
IDR
CTL
SEGTAB
CTL
1st TXT
ENTAB
(continued)
Present if TEST option and SYM records present
Present if OVLY option and more than 1 segment
Present if OVLY option used and more than 1 segment
TTR-N1, if OVLY option and more than 1 segment
RLD
CTRL,RLD,...CTL,RLD,TXT,ENTAB
RLD
CTL
TXT
TTR
Carries EOS if following ENTAB
Carries EOM if this is RLD for Last TXT
Carries EOM if no RLDs for Last TXT
Present if OVLY option and more than 1 segment

TTR-N[1] : TTR of the nore list used for overlay-structured modules.
TTR-N[2]: TTR of the first block of the named member (load module).
TTR-N[3]: TTR of the first block of text.

Fig. A1.1 Load module format

APPENDIX 2: ERROR DIAGNOSTIC AND WARNING MESSAGES

Each message contains a severity code in the final position of the message code which indicates the nature of the message. If an error is encountered during processing, the message code is printed with the applicable symbol or record in error. After processing is completed, the diagnostic message associated with that code is printed.
Message Format:
(1) Error/warning messages written when the errors are detected.

$\left\{\begin{matrix}\text{JQA0}\\ \text{JOB1}\end{matrix}\right\}$ mm& (applicable symbol or record in error)

(2) Error/warning messages written after processing is completed.

$\left\{\begin{matrix}\text{JQA0}\\ \text{JQB1}\end{matrix}\right\}$ mmI-x (message text)

where:

Table A2.1 Table of Linkage Editor and loader severity codes

Severity code		Return code	Meaning
&	x		
0	I	0	Informational message: appears when control statement is printed as result of LIST option. Condition will not cause an error during execution.
1	W	4	Warning message; condition may cause error during execution of module.
2	E	8	Error message; condition may make execution of module impossible; module is marked "hot executable", unless LET option was specified; module processing is continued.
3	S	12	Error message; condition will make execution of module impossible; module is marked "hot executable", module processing is continued.
4	U	16	Error message; no recovery from error condition is possible; module is not produced; module processing is terminated; only output is diagnostic messages.

- JQA0 identifies a Linkage Editor message
- JQB1 identifies a Loader message
- mm is the message number
- &, x are the severity codes of the message.

The severity codes used in Linkage Editor and Loader messages are defined in Table A2.1.

The return code reflects the highest severity encountered during processing. The highest severity code recorded is multiplied by 4 to create the return code. This code is placed in register 15 at the end of processing.

Message-severity code
Text
Explanation
S: System action
P: Recommended programmer response

JQA020I-W
(control statement)
The control statement is printed as a result of the LIST option.

JQA001I-E
ERROR: INVALID TWO-BYTE RELOCATABLE ADDRESS CONSTANT HAS BEEN FOUND IN INPUT—ADDRESS CONSTANT HAS NOT BEEN RELOCATED.
A relocatable A-type or V-type address constant of less than three bytes has been found in the input.
S: The constant is not relocated.
P: Probable user error. Check assembler language input for V-type address constants, which cannot be relocated. Delete or correct the invalid address constant.

JQA002I-E
ERROR: INVALID V-TYPE ADDRESS CONSTANT HAS BEEN FOUND IN INPUT—ADDRESS CONSTANT HAS NOT BEEN RELOCATED.

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

S: The constant is not relocated.

P: Probable user error. Specify a length of four bytes for all V-type address constants.

JQA003I-S

ERROR: ENTRY POINT FROM END CARD IS INVALID—ENTRY POINT IS NOT ASSIGNED.

The entry point for the program as specified as a relative address in an END card. The entry point that as 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.

S: No entry point is assigned.

P: Check object module input for completeness. Then either specify an entry point name on the ENTRY control statement, or, if entry points were specified at compilation or assembly, make sure the object module containing the desired entry point precedes all other object modules with assembled or complied entry points.

JQA005I-S

ERROR: SYMBOL PRINTED FROM ENTRY STATEMENT IS NOT AN EXTERNAL NAME. ENTRY POINT IS NOT ASSIGNED.

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

S: No entry point is assigned.

P: Probable user error. Correct the ENTRY control statement, or make sure that the control section containing the entry point is inculuded in the input and has not been accidentally deleted or redefined by a REPLACE or CHANGE control statement.

JQA006I-S

ERROR: SYMBOL PRINTED FROM END CARD IS NOT AN EXTERNAL NAME—ENTRY POINT IS NOT ASSIGNED.

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

S: No entry point is assigned.

P: Check that the entry point section or entry name has not been accidentally deleted or redefined by a REPLACE or CHANGE control statement. Check the module containing the entry point for completeness.

JQA007I-S

ERROR: ENTRY POINT FROM ENTRY STATEMENT IS IN A SEGMENT OTHER THAN THE ROOT SEGMENT OF OVERLAY STRUCTURE—ENTRY POINT IS NOT ASSIGNED.

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

S: No entry point is assigned.

P: Probable user error. Either correct the ENTRY control statement, or move the module containing the entry point to the root segment.

JQA008I-S

ERROR: ENTRY POINT FROM END CARD IS IN IN A SEGMENT OTHER THAN THE ROOT SEGMENT OF THE OVERLAY STRUCTURE—ENTRY POINT IS NOT ASSIGNED.

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

S: No entry point is assigned.

P: Probable user error. Move the object module containing the entry point to the root segment, or specify an entry point in the root segment using the ENTRY control statement.

JQA009I-S

ERROR: ENTRY POINT ADDRESS FROM END CARD IS IN A SEGMENT OTHER THAN THE ROOT SEGMENT OF OVERLAY STRUCTURE—ENTRY POINT IS NOT ASSIGNED.

The entry point is in a segment other than the root segment. Either the INSERT statement was used to place the control section containing the entry point in another segment, or the address specified on the END statement is incorrect.

S: No entry point is assigned.

P: Probable user error. Move the object module containing the entry point to the root segment, or specify an entry point in the root segment using the ENTRY control statement.

JQA010I-E

ERROR: ENTRY POINT ON END CARD IS INVALID—ENTRY POINT IS IGNORED.

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 identification of the specified entry point was found to be invalid.

S: The entry point is ignored. The first valid entry point encountered is used; if there is none, no entry point is assigned.

P: Probable user error. Check the input object modules for completeness and proper sequence. If necessary, either recreate any module which has been in card from, or isolate the incorrect module by executing the linkage editor with the NCAL option specified, using the NAME control statement for each input object module. Diagnostic

Message-severity code
Text
Explanation
S: System action
P· Recommended programmer response

JQA010 should recur and isolate the incorrect module. Recreate the module and rerun the step.

JQA011I-S
ERROR: NO SECTION HAS BEEN FOUND IN ROOT SEGMENT OF OVERLAY STRUCTURE—ENTRY POINT IS NOT ASSIGNED.

There are no control sections in the root segment. Either (1) all control sections originally in 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.

S: No entry point assigned.
P: Probable user error. Place at least one control section in the root segment.

JQA012I-S
ERROR: NO CESD ENTRIES—EXECUTION IMPOSSIBLE.

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

S: Processing is terminated.
P: Probable user error. Check other messages issued for cause of error (i.e., invalid input from object module). Insure that at least one control section appears in the input and is not deleted by the REPLACE control statement.

JQA013I-E
ERROR: SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE.

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

S: The module is marked not executable unless LET is specified.
P: Probable user error. Check that the reference is valid and not the result of a keypunch or programming error. If the reference is valid, add the needed module or alias to one of the input data sets. Make sure the SYSLIB data set DD statement has been specified, if needed. If resolution is not desired, specify NCAL, never-call, or restricted no-call. If the reference was found in a control section replaced by another control section not containing that same reference, delete the reference, or specify NCAL, never-call, or restricted no-call.

JQA014I-S
ERROR: NO TEXT REMAINS IN OUTPUT MODULE.

No text remains in the output module. Either all the control sections originally in the input are deleted, or there are no control sections that originally contained text.

S: Processing is terminated.
P: Probable user error. Check other messages issued for cause of error (i.e., invalid input from object module). Insure that at least one control section contains text and is not deleted by the REPLACE control statement or by automatic replacement.

JQA015I-E
ERROR: NO CALLS OR BRANCHES THAT REFER FROM ROOT SEGMENT TO LOWER SEGMENTS.

There are calls or branches from the root segment to a segment lower in the tree structure. Other segments cannot be loaded.

S: The module is marked not executable unless LET is specified.
P: Probable user error. Make sure the root segment contains a control section that refers to at least one other segment in the overlay structure by means of a V-type address constant.

JQA016-W
WARNING: EXCLUSIVE CALL THAT REFERS FROM SEGMENT NUMBER PRINTED TO SYMBOL PRINTED—XCAL WAS SPECIFIED.

There is a valid exclusive branch-type reference; the XCAL option is specified for this job step.

S: Processing continues.
P: No response is necessary normally. You can check that the printed branch-type references between exclusive segments are correct according to your overlay structure.

JQA017I-E
ERROR: EXCLUSIVE CALL THAT REFERS FROM SEGMENT NUMBER PRINTED TO SYMBOL PRINTED.

A valid branch-type reference is made from a segment to an excusive segment; the XCAL option is not specified.

S: The module is marked not executable unless the LET option is specified.
P: Probable user error. Either rearrange the overlay structure to place both segments in the same path, or specify the XCAL option.

JQA018I-E
ERROR: INVALID EXCLUSIVE CALL THAT REFERS FROM SEGMENT NUMBER PRINTED TO SYMBOL PRINTED.

There is an invalid exclusive branch-type reference from a segment to a symbol in an exclusive segment.

S: The module is marked not executable unless the LET option is specified.

P: Probable user error. Either place the segments in the same path, or place a V-type address constant in a common segment.

JQA020I-W

WARNING: OUTPUT MODULE WITH OVERLAY STRUCTURE CONTAINS ONLY ONE SEGMENT—OVLY OPTION CANCELLED.

There are no OVERLAY statements in the input.

S: The overlay option is cancelled.

P: Probable user error. Either place OVERLAY statements in the input, or remove the OVLY option from the EXEC statement.

JQA021I-E

ERROR: EXPECTED CONTINUATION CARD HAS NOT FOLLOWED.

A Linkage Editor control statement specifying a continuation (nonblack in column 72) is not followed by a continuation card.

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

P: Probable user error. Either remove the nonblank character in column 72 or insert the necessary continuation record.

JQA022I-E

ERROR: INVALID CARD PRINTED HAS BEEN FOUND IN INPUT OBJECT MODULE.

One of the following occurred during the processing of an object module: a record of invalid type was encountered, a text (TXT) record was encountered in which the data length (columns 11-12) is invalid or mispunched; an invalid, probably mispunched, RLD record was encountered in an object module.

S: The record in error is ignored and processing continues.

P: (1) Remove all extraneous records from the input to the Linkage Editor, (2) insure that Linkage Editor control statements are placed either or after object modules, (3) insure that all records in the object module have a 12-2-9 punch in column 1, (4) insure that all records in the object module contain one of the following in columns 2-4; ESD, SYM, TXT, RLD, or END, (5) locate the TXT or RLD record having the invalid or mispunched data, regenerate the object module or investigate the punching device or generating processor for malfunctions.

JQA023I-E

ERROR: INVALID RECORD HAS BEEN FOUND IN INPUT LOAD MODULE.

The member being read does not contain a valid load module.

S: The erroneous record is ignored and processing continues. The output load module is marked not executable.

P: Check that all input data sets are specified correctly on the DD statements. Isolate the incorrect load module by executing the Linkage Editor with INCLUDE and NAME statements for each suspected load module. When the incorrect load module is isolated, recreate it and rerun the job step.

JQA024I-W

WARNING: DOUBLY DEFINED EXTERNAL NAME HAS BEEN FOUND IN INPUT—ESD TYPE DEFINITIONS CONFLICT.

Two identical external names have been found in the input. (1) The invalid match involves a label reference (LR) or labe definition (LD) matching an existing section definition (SD), common (CM), or label reference (LR). The section definition for the input LR or LD must be marked delete in order for this not to be an error. (2) It is always invalid for a CM to match an existing LR.

S: References to the name are resolved with respect to the first occurrence of the name.

P: Probable user error. Correct the existing symbol conflict.

JQA025I-U

ERROR: TABLE OVERFLOW—TOO MANY EXTERNAL SYMBOLS IN ESD.

There are too many external symbol or control statement operands in the problem program.

S: Processing is terminated.

P: Probable user error: Check that no unnecessary modules or control statements are included in the input. Then, either increase the Linkage Editor's table space by increasing valuel (or decreasing value2) of the SIZE parameter, making sure the region or partition size is also increased, if necessary; or reduce the number of external symbols in the input (control sections, entry points, and named common areas).

JQA026I-U

ERROR: TABLE OVERFLOW—TOO MANY EXTERNAL SYMBOLS FROM INPUT MODULE.

Either (1) an input module contains too many external symbols in the ESD, or (2) an ESD card is mispunched.

S: Processing is terminated.

P: Probable user error. Check that input object modules are complete and not mispunched. Then, either break down any large input module into a number of smaller modules, or increase the linkage editor's table space by increasing valuel (or decreasing value2) of the SIZE parameter making sure the region or partition size is also increased, if necessary.

Message-severity code
Text
Explanation
S: System action
P: Recommended programmer response

JQA027I-E
ERROR: LOAD MODULE FROM LIBRARY SPECIFIED WAS NOT EDITABLE.
When the load module was created, it was marked "not editable".
S: The load module was not accepted as input.
P: Probable user error. If the module is unacceptable because it is marked "not editable," it must be recreated before it can be input to the Linkage Editor.

JQA028I-U
ERROR: INDISPENSABLE DDNAME PRINTED CANNOT BE OPENED—DD STATEMENT MISSING.
The specified data set cannot be opened. The DD statement defining the data set is missing.
S: Processing is terminated.
P: Probable user error. Either supply the missing DD statement, or correct erroneous information on the DD statement. If the Linkage Editor was invoked by a macro instruction such as LINK rather than through the EXEC statement, make sure any passed list of DDnames is correct.

JQA029I-U
ERROR: SYNCHRONOUS ERROR OCCURRED DURING ACCESS TO DDNAME PRINTED.
Either (1) a physical uncorrectable I/O error occurred, or (2) an object module is missing an END card as the last card, or (3) if the data definition name that was printed is for a DD statement that defines a blocked input data set of fixed format, an input record larger than the specified block size or logical record length was found.
S: Processing is terminated. The data definition name in the name field of the DD statement for the input data set was printed after the message code. If an input/output error occurred, the information provided by the SYNADA macro instruction was printed after the message code in the following format: SYNAD EXIT, jobname, stepname, unit address, device type, DDname, operation attempted, error description, block count or BBCCHHR, access method.
P: For any fixed format, specify the correct block size. If the block size was correct and the data set was an input data set, recreate or restore the data set.

JQA030I-E
ERROR: INVALID CONTROL STATEMENT IN INPUT—SCAN FOR CARD PRINTED TERMINATED.
Either there is an error on a Linkage Editor control statement, or an OVERLAY control statement was encountered and the OVLY attribute was not specified on the EXEC statement.
S: A statement in error is accepted as input up to the point of the error; the OVERLAY statements are ignored and the module is not in overlay format.
P: Probable user error. Either correct the error if necessary, or specify OVLY on the EXEC statement.

JQA031I-U
ERROR: REGIONS SPECIFIED BEYOND MAXIMUM NUMBER 4.
There are five or more regions specified in this overlay structure.
S: Processing is terminated.
P: Probable user error. Reduce the number of regions in the overlay structure to four or fewer.

JQA032I-U
ERROR: SEGMENTS SPECIFIED BEYOND MAXIMUM NUMBER.
The number of segments exceeded 255.
S: Processing is terminated.
P: Reduce the number of segments in the overlay structure to 255 or less.

JQA033I-E
ERROR: ALIASES SPECIFIED BEYOND MAXIMUM NUMBER—EXCESS IGNORED.
More than 64 (or 256 if AM256 is specified) aliases were specified for the output load module.
S: The excess aliases are ignored.
P: Probable user error. Either (1) specify AM256, (2) reduce the number of aliases, or (3) create a second copy of the module under a different name with the additional aliases specified.

JQA034I-E
ERROR: MODULE WAS NOT FOUND IN LIBRARY SPECIFIED.
The module or alias name specified on an INCLUDE or LIBRARY control statement was not found in the specified library.
S: Any references to the module are not resolved. The output load module is marked "not executable" unless the LET option has been specified.
P: Probable user error. Correct the library or module name on the DD, INCLUDE or LIBRARY control statement.

JQA035I-U

ERROR: TABLE OVERFLOW—TOO MANY EXTERNAL REFERENCES BETWEEN SECTIONS.

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.

S: Processing is terminated.

P: Probable user error. Either (1) increase the Linkage Editor's table space by increasing value1 (or decreasing value2) of the SIZE parameter, making sure the region size is also increased, if necessary; or (2) reduce the number of V-type address constants by combining control sections; or (3) change V-type address constants that do not refer across segments to A-type address constants with EXTRN statement.

JQA036I-U

ERROR: TABLE OVERFLOW—TOO MANY TEXTS OR CHANGES OF ORIGIN IN INPUT.

The internal tables used by the Linkage Editor to account for the text of a load module have overflowed due to discontinuities in the input text or the division of the text into pieces equal to the block size for the SYSMOD data set.

S: Processing is terminated.

P: Probable user error. (1) Increase the Linkage Editor's table space by increasing value1 (or decreasing value2) of the SIZE parameter, making sure the region or partition size is also increased if necessary; or (2) increase the Linkage Editor's buffer space by increasing both value1 and value2 of the SIZE parameter, making sure the region or partition size is increased proportionally; or (3) reduce the number of ORG statements specified in assembler language routines; or (4) break down the step into a number of link-edits, performing only part of the necessary linkage function in each successive step.

JQA037I-U

ERROR: TABLE OVERLOW—TOO MANY RELOCATABLE ADDRESS CONSTANTS OR TOO MANY SECTIONS CONTAINING SUCH CONSTANTS.

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

S: Processing is terminated.

P: Probable user error. Either increase the Linkage Editor's table space by increasing value1 (or decreasing value2) of the SIZE parameter, making sure the region size is also increased, if necessary; or reduce the number of relocatable address constants in the input. (One method is to assemble the coding of two or more control sections into one control section.)

JQA038I-E

ERROR: INVALID TEXT RECORD ID-CARD IGNORED.

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 sequence or incomplete.

S: The record is ignored. Processing continues.

P: Probable user error. Check the input object modules for completeness and proper sequence. If necessary, recreate any module which has been in card form.

JQA039I-U

ERROR: NO SPACE FOR LIBRARY DIRECTORY OR PERMANENT DEVICE ERROR—MEMBER NOT STORED.

This is either an input/output error or no space was allocated for the library directory.

S: Processing terminates.

P: Check the SYSLMOD data set to make sure it is a partitioned data set with space allocated for a directory. If necessary, restore library to a different volume, and rerun the job.

JQA040I-U

ERROR: NO SPACE LEFT IN DIRECTORY—MEMBER NOT STORED.

All the directory blocks allocated when the output data set was created have been used.

S: Processing terminates.

P: Probable user error. Either reprocess, placing the output module in a new library; when the original library is used as input, concatenate the new one with it; or use a utility program to copy the library, allowing for more directory entries. Edit the member into the new library.

JQA041I-E

ERROR: NO SPACE LEFT IN DIRECTORY—ALIAS NOT STORED.

All directory blocks allocated when the output data set was created have been used.

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

P: Probable user error. Either reprocess, placing the output module in a new library; when the original library is used as input, concatenate the new one with it, or 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.

Message-severity code
Text
Explanation
S: System action
P: Recommended programmer response

JQA042I-W
WARNING: IDENTICAL NAME IN DIRECTORY—ATTEMPT TAKEN TO STORE UNDER 'TEMPNAME' INSTEAD.

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

S: An attempt is made to store the output module into the library under the name TEMPNAME.

P: Probable user error. Either reprocess, using a different name in the SYSMOD DD statement or NAME statement, or reprocess, and specify the REPLACE function for the name originally specified in the SYSLMOD DD statement or the NAME statement.

JQA043I-E
ERROR: LIBRARY SPECIFIED CANNOT BE OPENED—DD STATEMENT MISSING.

The DD statement that defines the library is probably missing.

S: Processing continues without input from the specified library.

P: Probable user error. Either supply the missing DD statement, or correct erroneous information on the DD statement.

JQA044I-U
ERROR: TABLE OVERFLOW—TOO MANY REFERENCES TO LOWER SEGMENTS.

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

S: Processing is terminated.

P: Probable user error. Either increase the Linkage Editor's table space by increasing value1 (or decreasing value2) of the SIZE parameter, making sure the region size is also increased if necessary; or use an overlay structure with fewer segments.

JQA045I-U
ERROR: TABLE OVERFLOW—TOO MANY REFERENCES TO LOWER SEGMENTS IN ONE SEGMENT.

One segment in the overlay structure contains too many V-type address constants that refer to segments lower in the tree structure. The maximum is determined by the size of output load module record.

S: Processing is terminated.

P: Probable user error. Either (1) increase the size of an output load module record by specifying SYSMOD as a library with a larger block size, or (2) incorporate some of the called control sections in the requesting segment, or (3) divide the requesting segment into two or more segments.

JQA046I-W
WARNING: SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE—NCAL WAS SPECIFIED OR MARKED FOR RESTRICTED NO CALL OR NEVERCALL.

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

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

P: No response is necessary normally. Check that the reference is valid and not the result of a keypunch or programming error.

JQA047I-E
ERROR: ALIAS ENTRY POINT IS INVALID—OUT OF ROOT SEGMENT.

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

S: The entry point for the member name is used.

P: Probable user error. Respecify the alias, entry Point, or overlay structure.

JQA048I-U
ERROR: TABLE OVERFLOW—TOO MANY EXTERNAL SYMBOLS UNDERGOING RELOCATION.

There are too many symbols being relocated.

S: Processing is terminated.

P: Probable user error. Increase the Linkage Editor's table space by increasing value1 (or decreasing value2) of the SIZE parameter making sure the region or partition size is also increased if necessary.

JQA049I-E
ERROR: NAME STATEMENT FOUND IN OTHER THAN PRIMARY INPUT—STATEMENT IS IGNORED.

A NAME statement has been encountered in an included data set or an automatic library. NAME statements may be placed only in the primary input.

S: The record is ignored. Processing continues.

P: Remove the NAME statement fom the library or sequential data set. Reprocess if the load module is incorrect.

JQA050I-E
ERROR: PERMANENT DEVICE ERROR—ALIAS NOT STORED.

The alias could not be stored in the library directory because of an input/output error.

S: The load module has already been stored.

P: Execution of the module is possible using the member name or aliases already stored. The module can be link edited again with the new alias specified.

JQA051I-E
ERROR: SYNTAX INCOMPATIBLE WITH DATA SET RECORD FORMAT—INCLUDE STATEMENT FOR DDNAME PRINTED IGNORED.
The INCLUDE statement syntax conflicts with the characteristics of the data set specified on the DD statement.
S: The specified module is ignored.
P: Probable user error. Either specify a member name on the INCLUDE or DD statement if the data set is partitioned; or remove all member names from the INCLUDE statement if the data set is not partitioned.

JQA052I-E
ERROR: RECORD FORMAT OF DATA SET SPECIFIED IS UNACCEPTABLE—DDNAME PRINTED.
The record format of the specified data set is not type U or F and cannot be processed by the Linkage Editor.
S: The data set is not processed.
P: Probable user error. Correct the data set specification.

JQA053I-E
ERROR: BLOCK SIZE OF LIBRARY SPECIFIED CANNOT BE HANDLED AS EXCEEDING MAXIMUM—DDNAME PRINTED.
The block size of the specified library data set cannot be handled by the Linkage Editor.
S: The data set is not processed.
P: Probable user error. Either decrease the block size of the data set, or increase value2 of the SIZE parameter to allow for larger buffers, and increase value1 accordingly, if necessary.

JQA054I-S
ERROR: IDENTICAL NAME IN DIRECTORY—UNABLE TO STORE EVEN UNDER 'TEMPNAME'.
The member name 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.
S: The output module is not stored under this member name.
P: Probable user error. Either specify a unique member name for the module on the NAME control statement or the SYSLMOD DD statement, or specify the REPLACE function on the NAME statement.

JQA055I-E
ERROR: COMMON PRINTED EXCEEDED CONTROL OR PROTOTYPE SECTION SIZE WITH IDENTICAL NAME.
A named common area has been encountered which is larger than a control section with the same name.
S: The Linkage Editor uses the length specified for the control section. Processing continues.
P: Ensure that no named common area is larger than the control section initializing it.

JQA056I-E
ERROR: EXPAND CONTROL STATEMENT MISPLACED.
The section specified by an EXPAND statement has not yet been input to the Linkage Editor.
S: The EXPAND statement is ignored and processing continues.
P: The EXPAND statement must specify a section already in the input module.

JQA057I-E
ERROR: COMMON PRINTED AND SUBROUTINE HAVE MATCHING NAME.
This message appears only when the Linkage Editor is processing an object program originally written in FORTRAN. It is issued when a common area defined in the program has the same name as a subrogram.
S: Processing continues. The output module is marked "not executable" unless the LET option is specified.
P: Change the name of either the common area or the subprogram so that the names are no longer the same.

JQA058I-E
ERROR: COMMON AND CONTROL SECTIONS HAVE IDENTICAL NAME PRINTED—RENT OPTION CANCELLED.
A control section and a common section having the same name were entered while the RENT option was specified.
S: The RENT attribute is ignored.
P: When giving an initial value to a common section in creating a load module with the RENT attribute, use a prototype section.

JQA059I-U
ERROR: BLOCK SIZE OF INPUT DATA SET IS INVALID.
The block size for the primary input data (SYSLIN) is not an even multiple of the logical record length, or exceeds the allowable maximum.
S: Linkage Editor processing terminates.
P: Probable user error. The region for the job step must be large enough to allow the size values specified, as described in "EXEC Statement—REGION Parameter," in the

Message-severity code
Text
Explanation
S: System action
P: Recommended programmer response

Linkage Editor manual. If the region is not large enough, increase the REGION parameter before executing the Linkage Editor step again. If the blocking factor is greater than 40 to 1 or is not a multiple of the logical record length, correct the BLKSIZE field, or recreate the data set, or both. Execute the Linkage Editor step again.

JQA060I-E
ERROR: END CARD IS NOT CONTAINED IN INPUT OBJECT MODULE.

The END card of an object module being processed by the Linkage Editor is missing.

S: Linkage Editor processing continues. The load module produced is marked "not executable" unless the LET option has been specified.

P: Verify that the last card is an END card. Rerun the Linkage Editor step using the object deck.

JQA061I-U
ERROR: LENGTH FOR EXTERNAL SYMBOL PRINTED IS NOT SPECIFIED.

An object module contained a control section that had a length field containing zero in its external symbol dictionary (ESD) entry, and either the control section was not last in the object module or the length was not specified on the END card.

S: The module was not processed, and the Linkage Editor terminated processing.

P: Probable user error. Check the input object modules for completeness and proper sequence.

JQA063I-I
SYNCHRONOUS ERROR OCCURRED DURING ACCESS TO DDNAME PRINTED—XREF FAILED.

A permanent input/output error occurred while attempting to produce a cross-reference table. The output module was success fully edited.

S: The information provided by the SYNADAF macro instruction was printed after the message code in the following format; SYNAD EXIT, jobname, stepname, unit address, device type, DDname, operation attempted, error description, block count or BBCCHHR, access method.

P: Rerun the Linkage Editor step.

JQA064I-E
ERROR: SYMBOL PRINTED ON CONTROL STATEMENT WAS NOT MATCHED.

A control section name or common name appearing on an ORDER or PAGE control statement was not found in the primary or additional input sources.

S: The name is ignored. Processing continues.

P: Probable user error. Include the specified control section or common area in the input or delete the name from the control statement.

JQA065I-E
ERROR: ORDER SPECIFIED FOR SYMBOL PRINTED IS INVALID.

A control section or common area was named more than once in a series of ORDER statements. After a name appears once, any subsequent use of the name is invalid unless the name appears as the last operand on one ORDER statement and the first operand on the next.

S: The first use of the name determines the order of the control section or common area in the output load module. Any subsequent use of the name is ignored. Linkage Editor processing continues.

P: Probable user error. Correct the ORDER statement so the name appears only once or appears as the last operand on one statement and the first operand on the next.

JQA067I-I
THE IDENTIFY DATA HAS BEEN ADDED TO IDR RECORD FOR SECTION NAME PRINTED.

The Linkage Editor has added the data specified on the IDENTIFY control statement to the IDR record for the control section indicated.

S: Processing continues.

P: None. this message is for information.

JQA068I-E
ERROR: NO SECTION FOR THE IDENTIFY DATA OR IDR CONTROL STATEMENT MISPLACED—DATA IGNORED.

The control section named on the IDENTIFY control statement either does not exist in the load module or has not been read in by the Linkage Editor by the time it encountered the IDENTIFY statement.

S: The data specified on the IDENTIFY statement is ignored. Linkage Editor processing continues.

P: Probable user error. Check the IDENTIFY statement to verify that the control section name has been specified correctly and that the IDENTIFY statement has been placed correctly in the input. Verify that the required control section has been included in the input to the Linkage Editor step. Correct the input and rerun the Linkage Editor step.

JQA069I-U
ERROR: TABLE OVERFLOW-SIZE VALUE NOT LARGE ENOUGH FOR SECTION IDR INPUT—LINKAGE EDITOR PROCESSING TERMINATED.

The space available for CSECT identification records was insufficient for the actual input.

S: Linkage Editor processing terminates.

P: Rerun the link edit, increasing the space available to the Linkage Editor by increasing value1(or decreasing value2, or both) of the SIZE option. Be sure that the region size is also increased correspondingly. If this fails, divide the link edit into two or more smaller link edits.

JQA070I-U
ERROR: INVALID CODE DETECTED IN SECTION IDR INPUT LINKAGE EDITOR PROCESSING TERMINATED.

An unrecoverable error was detected while processing an input module containing CSECT Identification (IDR) records.

S: Linkage Editor processing terminates.

P: Probable user error. Examine all data sets containing input load modules. Check all secondary input sources.

JQA071I-U
ERROR: NO SPACE AVAILABLE FOR STOW ACTION—MEMBER NOT STORED.

The conditional GETMAIN macro instruction issued by the STOW routine to obtain work space in virtual storage was unsuccessful.

S: The member is not stored in the specified library; Linkage Editor processing is terminated.

P: Rerun the Linkage Editor job step. The error may be a temporary one caused by fragmentation of virtual storage.

JQA072I-E
ERROR: ALIAS NAME PRINTED IS INVALID.

An ALIAS name has been specified that either does not begin with an alphabetic character, $, #, @, or 12-0 punch, or contains a character that is not alphanumeric, $, #, @, or 12-0 punch.

S: The ALIAS name is ignored.

P: Correct the invalid character(s) in the ALIAS name according to the rules above and rerun the link-edit job step.

JQA073I-W
WARNING: ALIAS MATCHED MEMBER NAME—ALIAS IGNORED.

An ALIAS name has been specified that duplicates the member name of the output load module.

S: The ALIAS name is ignored.

P: Either (1) delete the ALIAS name, or (2) make the ALIAS name unique.

JQA074I-I
THE SPECIFIED ACTION TAKEN FOR AN EXPAND REQUEST.

The Linkage Editor has increased the size of a control section or named common section by the number of bytes specified in an EXPAND control statement. Details of the expansion are provided in the message text that appears immediately following the EXPAND control statement.

S: Processing continues.

P: None. This message is for information only; no error has occurredand no response is required.

JQA075I-W
WARNING: NO UNRESOLVED EXTERNAL REFERENCE REQUIRES DYNAMIC LINK—DYNAMIC OPTION CANCELLED.

While the DYNAMIC option is specified, there is no unresolved V-type address constant.

S: This is a warning. The DYNAMIC option is ignored.

P: Eliminate the DYNAMIC option. Execution is not hampered.

JQA076I-U
ERROR: TABLE OVERFLOW—TOO MANY UNRESOLVED EXTERNAL REFERENCES REQUIRE DYNAMIC LINK.

There are too many unresolved external references which require dynamic linkage.

P: Processing is discontinued.

P: Expand the table area by increasing the value1 (or decreasing the value2) of the SIZE option. If necessary, also enlarge REGION size of the EXEC statement. Or, decrease the number of unresolved external references by inputting needed modules.

JQA077I-U
ERROR: TABLE OVERFLOW—MODULE CONTAINS TOO MANY UNRESOLVED EXTERNAL REFERENCES REQUIRING DYNAMIC LINK.

There are too many unresolved external references which require dynamic linkage.

S: Processing is discontinued.

P: Decrease the number of modules requiring dynamic linkage by inputting needed modules.

JQA078I-E
ERROR: INVALID REENTERABLE PROGRAM. INPROPER REFERENCE IN ADDRESS PRINTED—RENT OPTION CANCELLED.

An address printed in a reenterable program makes a reference from the control section group to the prototype section group.

S: The RENT option is ignored.

P: Correct the erroneous reference made by the address constant.

Message-severity code
Text
Explanation
S: System action
P: Recommended programmer response

JQA098I-U
ERROR: TOO LARGE SYSPRINT BLOCKSIZE—LINKAGE EDITOR PROCESSING TERMINATED.
The block size specified for the SYSPRINT data set cannot be handled by the Linkage Editor.
S: The dataset is not opened. Linkage Editor processing terminates.
P: Probable user error. Either decrease the block size of the data set, or increase value2 of the SIZE option to allow for larger buffers, and increase valiel accordingly, if necessary. Increase the region size, if necessary. Rerun the Linkage Editor step.

JQA099I-U
ERROR: SYSPRINT DD STATEMENT MISSING—LINKAGE EDITOR PROCESSING TERMINATED.
The SYSPRINT data set cannot be opened.
S: Linkage Editor processing terminates.
P: Probable user error. The SYSPRINT DD statement is probably missing. Supply the missing SYSPRINT DD statement, and execute the job step again.

JQB101I-W
WARNING: NO ENTRY POINT
No entry point was specified in the parameter field or END card. The END card entry point specification could be incorrect (i.e., invalid ID, bad column alignment, etc.) The parameter field specification could also be incorrect.
S: The first assigned address is used as the entry point.
P: Probable user error. Specify the entry point name in the loader parameter list, EP=. If the entry point occurs in load module input, this parameter must be specified.

JQB102I-W
WARNING: ILLEGAL RECORD—IGNORED.
The card read has a blank in column one.
S: The card is ignored.
P: Probable user error. Check input for a blank card or linkage ditor control card. If other errors occur, recreate all object modules which have been in card form. Rerun the step using the Linkage Editor instead of the loader, and save the resulting output.

JQB103I-W
WARNING: EXTERNAL REFERENCE—UNRESOLVED (NOCALL SPECIFIED).
The NCAL, NOCALL, or NORES option or never-call function was specifed for the external reference.
S: The SYSLIB data set is not searched if the NCAL or NOCALL option has been specified. The Link Pack Area queue is not searched if the NORES option has been specified. Neither the SYSLIB data set nor the Link Pack Area queue are searched if the ER is marked 'never-call' from a previous Linkage Editor run.
P: No response is necessary normally. If you wish the reference resolved, either (1) add the needed module to the SYSIN input data set; (2) remove the NOCALL, NCAL, or NORES option, if specified; or (3) if an input load module contained a never-call reference, recreate the load module without specifying never-call. Run the failing step using the Linkage Editor instead of Loader and save the resulting output.

JQB104I-E
MISTAKE: EXTERNAL REFERENCE—UNRESOLVED.
The external reference was not found on the defined SYSLIB data set or in the Link Pack Area.
S: No attempt is made to execute the module unless the LET option is specified.
P: Probable user error. Make sure that the reference is valid and not the result of a keypunch or programming error. If the reference is valid, add the needed module or alias to either (1) the SYSLIB data set, (2) the link pack area, or (3) the SYSLIN input data set.

JQB105I-E
MISTAKE: INVALID ID.
Input contains an invalid external symbol ID. This error is the result of the following conditions: (1) the SD for an ID does not appear in the input module, (2) text is received before the ESD defining it is received, (3) an RLD is received before the ESDs to which it pertains, (4) the ID defining the entry point on the END card is not a defined SD, PC, or LR ESD type.
S: The invalid item is ignored.
P: Check that input object modules are complete and that assembly or compilation errors did not occur when object modules were generated. Rerun the step with the NOCALL option specified.

JQB106I-E
MISTAKE: DOUBLY DEFINED ESD ENTRY.
Two identical external names have been found in the input. The invalid match involves a label refer-

ence (LR) or label definition (LD) matching an existing (SD, PS, CM), or label reference (LR). The section definition for the input LR or LD must be deleted in order for this not to be an error. It is always invalid for a CM to match an existing LR.

S: References to the name are resolved with respect to the first occurrence of the name.

P: Probable user error. Correct the existing symbol conflict.

JQB107I-E

MISTAKE: COMMON EXCEEDS SIZE OF CSECT OR PSECT WITH IDENTICAL NAME.

A named common area has been encounted which is larger than the control section with same name.

S: The loader uses the length of the control section. Processing continues.

P: Ensure that no named common area is larger than the control section initializing it.

JQB108I-E

MISTAKE: INVALID TWO-BYTE ADCON.

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

S: The constant is not relocated.

P: Probable user error. Check assembler language input for Y-type address constants, which can't be relocated. Delete or correct the invalid address constant. Rerun the step using the Linkage Editor instead of the Loader, and save the resulting output.

JQB109I-E

MISTAKE: NO END RECORD.

An END card is missing for an input object module.

S: Processing continues.

P: Probable user error. Check input object modules. The last record of each should have a 12-2-9 punch in column 1 and the END identifier in columns 2–4. If an END record is missing, recreate the module and rerun.

JQB110I-E

MISTAKE: INVALID RECORD IN OBJECT MODULE.

An unrecognizable record type was received while reading an object module.

S: The card is ignored.

P: Probable user error. Check object module input for invalid records.

JQB-111I-E

MISTAKE: INVALID TEXT LENGTH.

The length of a control section in an object module was not specified in either its ESD entry or on the END record, and text was received for the control section.

S: The total length of the text received was used.

P: Check if an END record in any object module is missing or has been replaced. If so, recreate the object module and rerun. Execute the failing step using the Linkage Editor instead of the Loader and save the resulting output.

JQB112I-E

MISTAKE: INVALID BLKSIZE.

In the specified data set, the BLKSIZE was not an integral multiple of LRECL.

S: BLKSIZE was rounded up to the next higher multiple of LRECL and processing continued.

P: Probable user error. Change BLKSIZE to be an integral multiple of LRECL.

JQB113I-S

MISTAKE: ENTRY POINT NAME NOT MATCHED.

The entry point name specified in the parameter field or on an END card was not matched to an incoming LR, SD, or PC.

S: The first assigned address is used as the entry point address.

P: Probable user error. (1) Check to see if the EP= parameter was specified correctly. (2) Check to see if the module containing the entry point is included in either the SYSLIN or SYSLIB input. (3) Check other messages issued for the cause of error (i.e., invalid record).

JQB114I-S

MISTAKE: NO TEXT.

No valid text has been received for the loaded module.

S: The loader returns to the caller with a condition code of 12.

P: Probable user error. Make sure that the SYSLIN data was specified correctly. Check other error messages issued for cause of error (e.g., invalid record).

JQB115I-S

MISTAKE: INVALID LOAD MODULE FORMAT.

An unrecognizable record was found while reading a load module.

S: The record was ignored and processing continued.

P: Check that all input data sets are specified correctly on DD statements. When the incorrect load module is isolated, recreate it and rerun the job step.

JQB116I-S

MISTAKE: I/O ERROR WHILE SEARCHING SYSLIB.

A permanent I/O error occurred while attempting a BLDL.

S: Automatic library call processing is terminated.

P: Ensure that the SYSLIB defined data set is

Message-severity code
Text
Explanation
S: System action
P: Recommended programmer response

partitioned. If it is, recreate or restore the data set and rerun the job step. Execute the failing step using the Linkage Editor instead of the Loader and save the resulting output.

JQB117I-U
MISTAKE: TOO MANY ESD ENTRIES IN INPUT MODULE.
The external symbol ID is too large to fit in the translation table.
S: Processing is terminated.
P: If the program is large and/or complex, either run the step using the Linkage Editor, or break down the large program module into a number of smaller routines. If the program is not particularly large or complex, check other messages issued for the cause or error. Object module input may be incomplete or mispunched.

JQB118I-U
MISTAKE: INVALID RECFM.
Only object module (FIXED record format) and load module (UNDEFINED record format) data sets are accepted by the loader.
S: Processing was terminated.
P: Probable user error. Make sure that the record format specification is correct.

JQB119I-U
MISTAKE: PRIMARY INPUT CANNOT BE OPENED.
The SYSLIN data set cannot be opened. The DD statement defining the data set is missing or incorrect.
S: Processing terminates.
P: Probable user error. Either supply a missing SYSLIN DD statement or correct erroneous information on the SYSLIN DD statement.

JQB120I-U
MISTAKE: WORKING AREA OVERFLOWED.
The amount of virtual storage available to the Loader is insufficient to allow construction of the required tables and loaded program.
S: Processing was terminated.
P: Probable user error. (1) Increase the SIZE parameter, or (2) make sure the REGION specification is sufficient, or (3) make sure that sufficient virtual storage is available to satisfy the SIZE specification.

JQB121I-U
MISTAKE: SYNCHRONOUS I/O ERROR.
A physical uncorrectable input/output error occurred. If it occurred on a blocked data set, the block size may have been specified incorrectly.
S: The message supplied by the SYNADAF macro was printed. Processing was terminated.
P: For any fixed format, specify the correct block size. If the block size was correct and the data set was an input data set, recreate or restore the data set. Execute the failing step using the Linkage Editor instead of the Loader and save the resulting output.

JQB122I-U
MISTAKE: IDENTIFICATION FAILED.
The IDENTIFY routine located an error in the parameter list passed to it by the Loader.
S: Processing is terminated.
P: Verify that the appropriate IDENTIFY macro instruction support is included in the system. The release level of the IDENTIFY macro instruction should be the same as the release level of the Loader.

JQB123I-U
MISTAKE: IDENTIFICATION FAILED (EXISTING PROGRAM NAME).
When trying to identify the loaded program to the system, the IDENTIFY routine found a duplicate program name in the user's region or partition or in the Link Pack Area.
S: Processing is terminated.
P: Probable user error. Specify a unique program name using the NAME option or let the Loader default the name to **GO. Rerun the job.

JQB199I-L
MISTAKE: USER PROGRAM ABNORMALLY TERMINATED.
This message is issued by the loader when it determines that the loaded program has terminated abnormally.
S: Loaded program execution is terminated abnormally, and control is returned to the Loader.
P: Eliminate the cause of the abnormal termination and rerun the job.

APPENDIX 3: SAMPLE INPUT FOR THE LOADER

Fig. A3.1 shows an input deck for a loading job. You wish to load a previously-assembled program, MASTER, without using any SYSLOUT, SYSLIB, or SYSTERM DD statements.

```
//LOAD       JOB    MSGLEVEL=1
//           EXEC   PGM=LOADER
//SYSLIN     DD     DSNAME=MASTER,DISP=OLD
  (DD statements and data required for execution of MASTER)
/*
```

Fig. A3.1 Input deck for a loading job

Fig. A3.2 shows an input deck for a compile-load job using the JIS COBOL compiler for the compile step. The loaded program requires the SYSOUT, TAXRATE, and SYSIN DD statements.

Fig. A3.3 shows the compilation and loading of three modules. In the first three steps, you use the FORTRAN HE compiler (JMFAA00) for one main program, MAIN, and two subprograms, SUB1 and SUB2. You place each of the object modules into a separate sequential data set and pass them to the Loader job step. In addition to the FORTRAN library, you use a private library, MYLIB, to resolve external references. In the Loader step, you concatenate MYLIB with the SYSLIB DD statement. Include SUB1 and SUB2 by concatenating them with the SYSLIN DD statement. Your SYSTERM statement defined the diagnostic output data set. The loaded program requires FT01F001 and FT10F001 DD statements for execution, but it does not require data from the JES input stream.

```
//JOB          JOB    22,MCS,MSGLEVEL=1
//COBOL        EXEC   PGM=JMXCBL00,PARM=MAP,REGION=86K,RD=R
//SYSPRINT     DD     SYSOUT=A
//SYSPUNCH     DD     UNIT=SYSCP
//SYSUT1       DD     UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT2       DD     UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT3       DD     UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT4       DD     UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSLIN       DD     DSNAME=&&LOADSET,DISP=(MOD,PASS),
//                    UNIT=SYSSQ,SPACE=(TRK,(30,10))
//SYSIN        DD     *
  (Source program)
//LOAD         EXEC   PGM=LOADER,PARM='MAP,LET',COND=
//                    (5,LT,COBOL)
//SYSLIN       DD     DSNAME=*.COBOL SYSLIN,DISP=
//                    (OLD,DELETE)
//SYSLOUT      DD     SYSOUT=A
//SYSLIB       DD     DSNAME=SYS1.COBLIB,DISP=SHR
//SYSOUT       DD     SYSOUT=A
//TAXRATE      DD     DSNAME=TAXRATE,DISP=OLD
//SYSIN        DD     *
  (Data for loaded program)
/*
```

Fig. A3.2 Input deck for a compile-load job

```
//JOBX          JOB
//STEP1         EXEC     PGM=JMFAA00,PARM='NAME=MAIN,LOAD'
                  :
//SYSLIN        DD       DSNAME=&&GOFILE,DISP=(,PASS),UNIT=SYSSQ
//SYSIN         DD       *
  (Source module for MAIN)
/*
//STEP2         EXEC     PGM=JMFAA00,PARM='NAME=SUB1,LOAD'
                  :
//SYSLIN        DD       DSNAME=&&SUBPROG1,DISP=(,PASS),UNIT=SYSSQ
//SYSIN         DD       *
  (Source module for SUB1)
/*
//STEP3         EXEC     PGM=JMFAA00,PARM='NAME=SUB2,LOAD'
                  :
//SYSLIN        DD       DSNAME=&&SUBPROG2,DISP=(,PASS),UNIT=SYSSQ
//SYSIN         DD       *
  (Source module for SUB2)
/*
//STEP4         EXEC     PGM=LOADER
//SYSTERM       DD       SYSOUT=A
//SYSLIB        DD       DSNAME=SYS1.FORTLIB,DISP=OLD
//              DD       DSNAME=MYLIB,DISP=OLD
//SYSLIN        DD       DSNAME=*.STEP1.SYSLIN,DISP=OLD
//              DD       DSNAME=*.STEP2.SYSLIN,DISP=OLD
//              DD       DSNAME=*.STEP3.SYSLIN,DISP=OLD
//FT01F001      DD       DSNAME=PARAMS,DISP=OLD
//FT10F001      DD       SYSOUT=A
//
```

Fig. A3.3 Input deck for compilation and loading of the three modules

INDEX

64SP-3150E-1

FACOM OS IV/F4
Linkage Editor and Loader
User's Guide

COMMENT FORM

Please use the form below to write whatever comments and suggestions you may have regarding this publication. The completed form should be given to the FACOM representative in your area.

Your opinions please.
Please mark each item below with the appropriate letter representing your frank views on this publication, i.e. E (excellent), G (good), F (fair), P (poor).

☐ Text usefulness	☐ Illustrations/tables	☐ General appearance
☐ Text clarity	☐ Index coverage	☐ Paper quality
☐ Text accuracy	☐ Cross referencing	☐ Printing
☐ Text organization	☐ ________________	☐ Binding

Detailed comments:

Name: ______________________ Position: ______________________

Company or organization: ______________________

Address: ______________________ Reply requested: No

______________________ Yes

FOR OFFICE USE ONLY. Do not fill in here.

Local representative: Date received:

Documentation section Date received:

Action:

Seen and checked by______________________