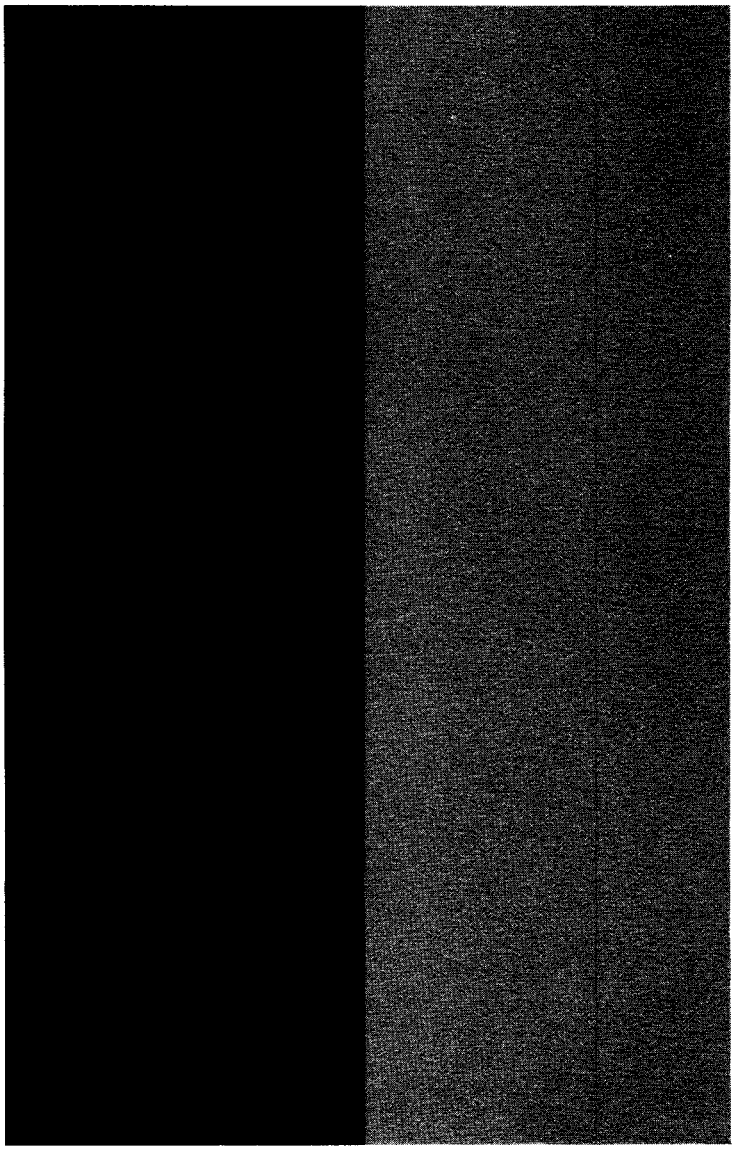
SPERRY RAND

UNIVAC 9400 SYSTEM

LINKAGE EDITOR



PROGRAMMER REFERENCE

UP-7703

This document contains the latest information available at the time of publication. However, the Univac Division reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Univac Representative.

UNIVAC is a registered trademark of the Sperry Rand Corporation.

Other trademarks in this publication are:

UNISERVO

PRINTED IN U.S.A.

UNIVAC 9400 System
Linkage Editor Programmers Reference
UP-7703

May 21, 1971

UPDATE PACKAGE "A"

UNIVAC 9400 System P.I.E. Bulletin 24, UP-7593.24, announces the release and availability of Updating Package "A" to "UNIVAC 9400 System Linkage Editor Programmers Reference," UP-7703, cover and 18 pages plus page i and 1 Updating Summary Sheet.

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Front Cover	†	†
Contents	1 and 2	1 Rev. 1 and 2 Rev. 1
Section 2	3 and 4	3 Rev. 1 and 4*
	5 and 6	5* and 6 Rev. 1
	7 and 8	7 Rev. 1 and 8 Rev. 1
Section 3	1 thru 6	1 Rev. 1 thru 6 Rev. 1
	N. A.	7**
Appendix B	N. A.	1** thru 3**

†Destroy old cover and file new cover.
*These pages are backups of revised pages and remain unchanged.
**These are new pages.

CONTENTS

APPENDIXES

FIGURES

1. INTRODUCTION

1.1. GENERAL

The primary function of the Linkage Editor is to link the output of various runs of language processors into one program that can be loaded and executed by the Supervisor Control program. This manual describes the requirements for linking several program units (object modules) to produce a single, executable program unit (load module).

Use of this manual requires a knowledge of *UNIVAC 9400 Assembler/Central Processor Unit Programmers Reference, UP-7600* (current version), *UNIVAC 9400 Supervisor for Disc Systems Programmers Reference, UP-7587* (current version), and *UNIVAC 9400 Job Control for Disc Systems Programmers Reference, UP-7585* (current version).

The ability of the Linkage Editor to construct a single load module from several object modules has the following advantages:

- If a change is required in one of the object modules, only that object module which is changed must be reassembled.
- The various object modules may be written in the appropriate language, such as, Assembler language, COBOL, FORTRAN, or RPG.
- Routines which are common to two or more object modules may be assembled or compiled only once and the resulting object code linked as needed, thereby reducing the total time required to assemble.

In addition to the linking function, the Linkage Editor performs the following:

- Searches the appropriate library and incorporates object modules other than those in its primary input, either upon request or automatically.
- Performs program modification by deleting and rearranging control sections of an object module as directed.
- Produces an overlay structure to be used by the Supervisor Control program during loading.
- Reserves storage automatically for common storage requests generated by the language processors.

The output of a language processor is a single unit, referred to as an *object module*. A single execution of the Linkage Editor combines one or more object modules into a single executable unit, referred to as a *load module*. Some of the object modules being linked may be standard subroutines from the system library. The Linkage Editor resolves all cross-references among the object modules being linked. It recognizes requests for labeled or unlabeled common storage and allocates the necessary space. It also analyzes external references to enable it to obtain object modules automatically from the library to be included in the load module it is creating.

The Linkage Editor can prepare programs with an overlay structure; that is, a load module may consist of more than one phase. A phase is a portion of a load module which can be loaded as an overlay by a single execution of a Supervisor LOAD or FETCH macro instruction. The first phase is called the root phase. It is loaded by the Job Control program; all other phases are loaded by a previously loaded phase. The name, starting address, and relative position of the phase within the load module are specified by Linkage Editor control statements.

Figure 1–1 illustrates the Linkage Editor input/output flow of a problem program in a Tape Operating System (TOS) or a Disc Operating System (DOS).

1.2. LINKAGE EDITOR INPUT

An object module produced by a language processor consists of one or more control sections. It contains the following types of records in the quantity indicated in parentheses:

- Object Module Header (1)

  The object module header must be the first record in the object module. It identifies the module and is used to locate the module.

- Control Section Records (1 or more)

  These records define the name, starting address, and length of each control section.

- External Symbol Definition Records – ESD (0, 1, or more)

  These records contain the information used to define cross-references when object modules are linked.

- Text and Relocation Records (0, 1, or more)

  Text and Relocation records contain the data (and instructions) comprising the object module, as well as the information to make the necessary modifications when the problem program is relocated.

- Transfer Record (1)

  This record indicates the end of the object module. It may specify the address at which the execution of the program is to start. A COBOL-produced object module may contain more than one Transfer record.

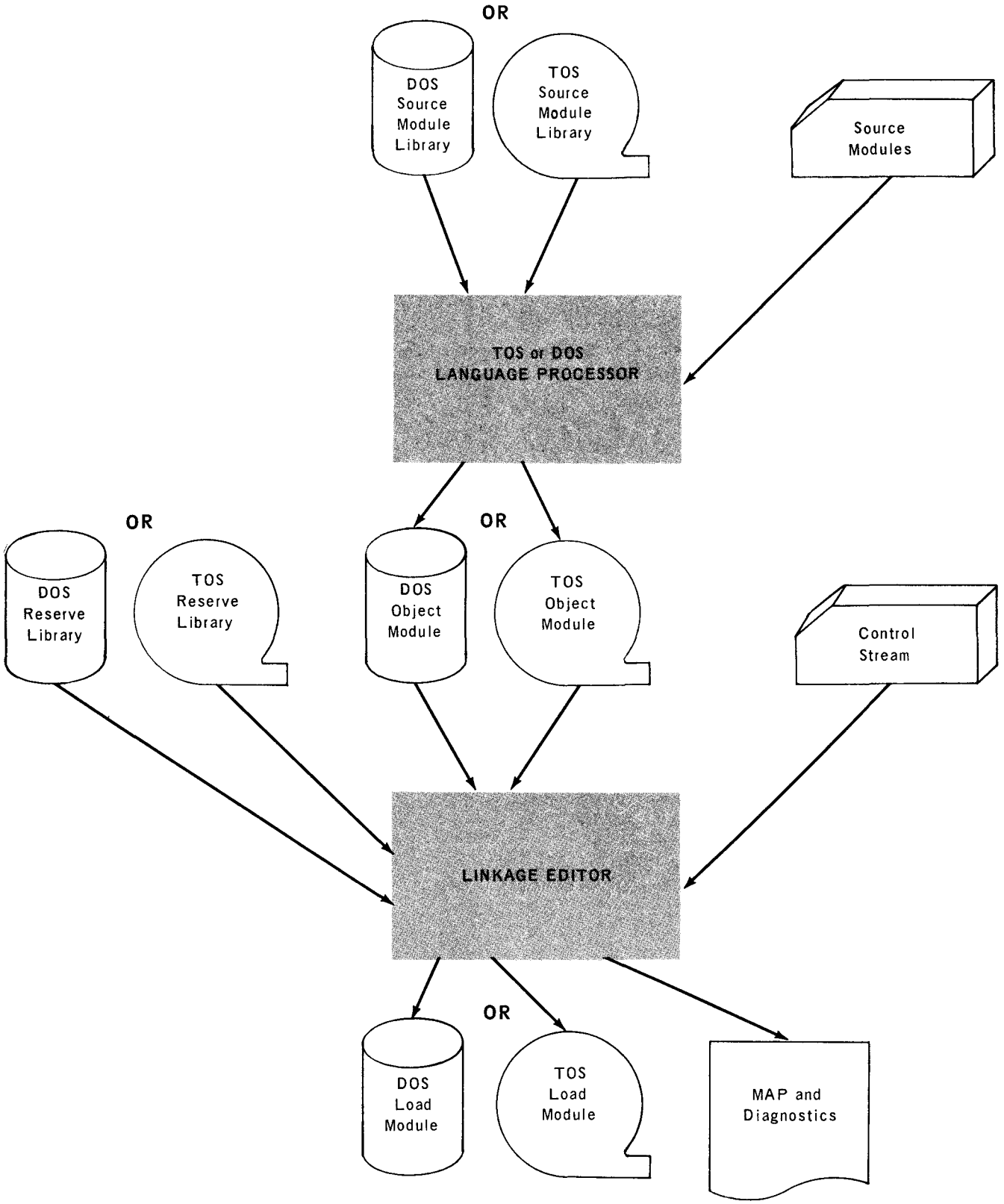


Figure 1–1. Linkage Editor Input/Output Flow

1.3. LINKAGE EDITOR OUTPUT

A load module produced by the Linkage Editor is relocatable and consists of one or more phases. Each phase contains the following types of records in the quantity indicated in parentheses:

- Phase Header (1)

  The phase header is the first record in the phase. It defines the name, starting address, and length of the phase, and is used to locate the phase.

- Tape Loader Record (1)

  The tape loader record is the second record in the phase. It contains the coding required to load a phase into storage. This record is only required for programs produced on tape.

- External Symbol Definition Records – ESD (0, 1, or more)

  Only those ESD records which define DTF areas are preserved in the output of the Linkage Editor.

- Text and Relocation Records (0, 1, or more)

  These records contain the information selected from the Text and Relocation records in the object modules included in the phase. The Text and Relocation records in the load module are in the same format as the corresponding records in an object module. However, the information has been modified by the Linkage Editor.

- Transfer Record (1)

  This record is in the same format as the Transfer record in an object module. It contains the address at which execution of the phase is to start.

2. LOAD MODULE STRUCTURE

2.1. ORDER OF PROGRAM UNITS IN STORAGE

The Linkage Editor not only allocates storage to object modules included specifically by means of Linkage Editor control statements, but also automatically allocates storage to certain program units based on requirements indicated in the included object modules.

The order of storage allocated to a load module is shown in Figure 2–1. The units that are included automatically are incorporated into the first phase. This phase is called the *root phase* and must not be overlaid. Multiphase load modules are discussed in 2.3.

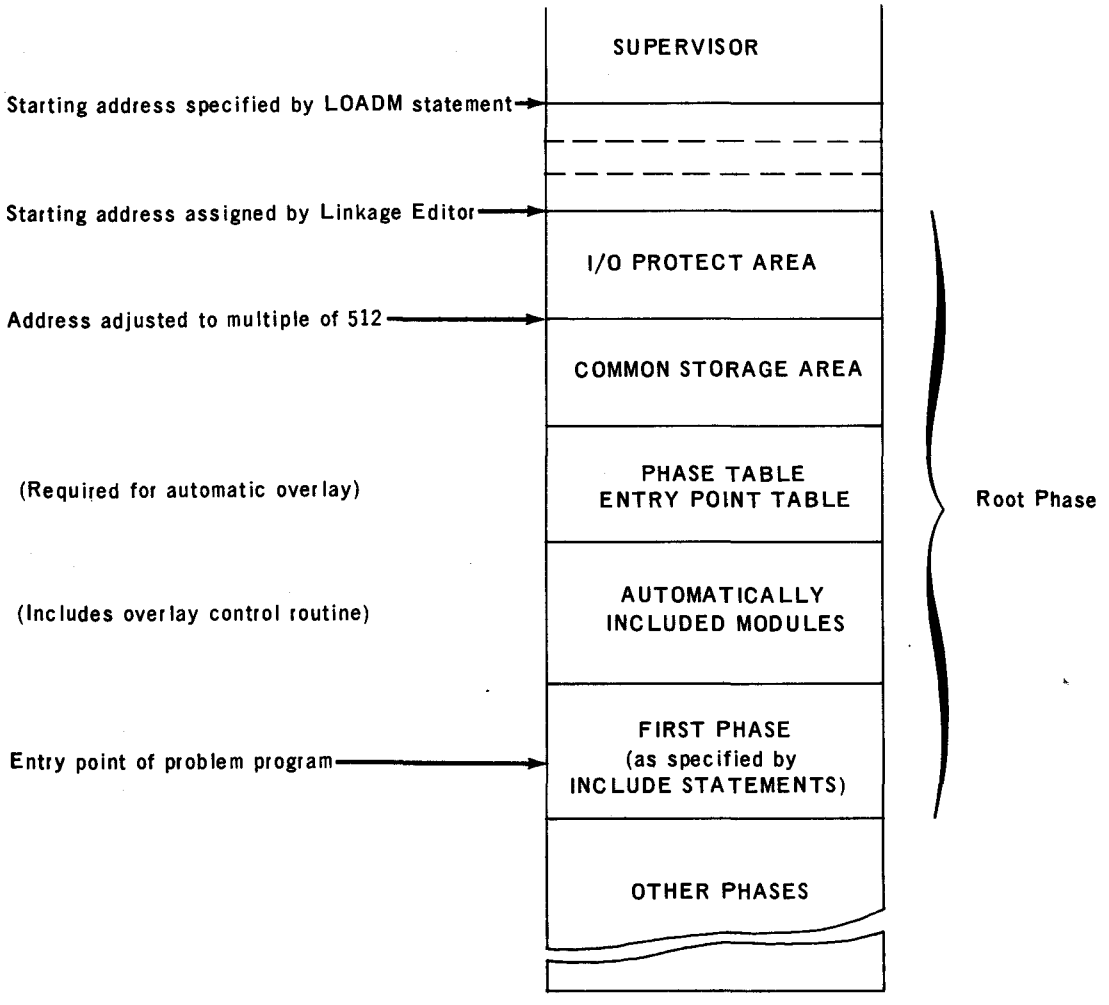


Figure 2–1. Structure of a Multiphase Load Module in Main Storage

2.2. AUTOMATICALLY ALLOCATED PROGRAM UNITS

Certain program units may require storage allocation based on indications in the object modules. Such indications may be for I/O protection, common storage, or the inclusion of specified routines. The Linkage Editor automatically allocates storage for these program units.

2.2.1. I/O Protect Area

An optional hardware feature provides storage protection. When this feature is installed and is selected when generating the Supervisor, the operating system loads programs on a boundary that is a multiple of 512 and locks out that portion of storage not allocated to the program being executed. This prevents the program from altering portions of storage allocated to other programs or to the Supervisor. This hardware protection does not include protecting any given portion of storage against alteration by an input operation initiated by another program.

One way of providing I/O protection is by separating certain critical portions of each DTF macro instruction from the rest of the program and placing these portions in a protected area of storage. When the Linkage Editor is requested by the P option of the LINKOP control statement to produce a load module which has I/O protection, it segregates the relevant portion of each DTF macro instruction into an I/O protect area. The Linkage Editor forces this area to end at an address that is a multiple of 512 so that storage protection (the hardware option) can lock out this particular area.

When a Data Management routine forms an address to be used in an input operation, it checks that the input operation affects only those storage locations allocated to the program requiring the input operation and does not violate the I/O protect area.

Each portion of the I/O protect area is loaded when the phase containing the related DTF macro instruction is loaded.

2.2.2. Common Storage

The Linkage Editor reserves common storage areas as requested and described by FORTRAN and Assembler statements. A common storage area may be labeled (named) or unlabeled (blank). An object module must contain one common storage definition record for each common storage area required. The common storage definition record requests a common storage area and gives the name of the area and the amount of storage required.

If more than one object module refers to a common storage area with the same name, the references are to the same storage area. Only one common storage area is allocated within a load module to satisfy all requests for common storage with the same name. The size of a common storage area in a load module is determined by the maximum size requested by any object module for common storage with that name. Blank common is allocated in the same way.

In a load module with several phases, common storage areas are not overlaid.

The following rules apply to the use of common storage:

- An entry point may not have the same name as a labeled common storage area included in the load module.
- The first time the Linkage Editor includes a control section with the same name as a labeled common storage area, that section is treated as a Block Data subprogram (see *UNIVAC Systems Fundamentals of FORTRAN, UP-7536* (current version)), and is loaded into all or a portion of the common storage area. A Block Data subprogram is loaded when the phase in which it was included is loaded. Blank common cannot be initialized during loading.
- If an object module has requested common storage, the partial inclusion of a single control section from that object module results in the allocation of the common storage area in the load module whether or not the included control section refers to that common storage.

2.2.3. Automatic Inclusion

The Linkage Editor constructs a load module from the object modules and control sections specified by INCLUDE control statements contained in the control stream. If the control statements defining the load module contain a LINKOP statement with the NOAUT option (see 3.3), the Linkage Editor uses only those object modules and control sections specified by INCLUDE statements. Any references to labels which are undefined are flagged as errors.

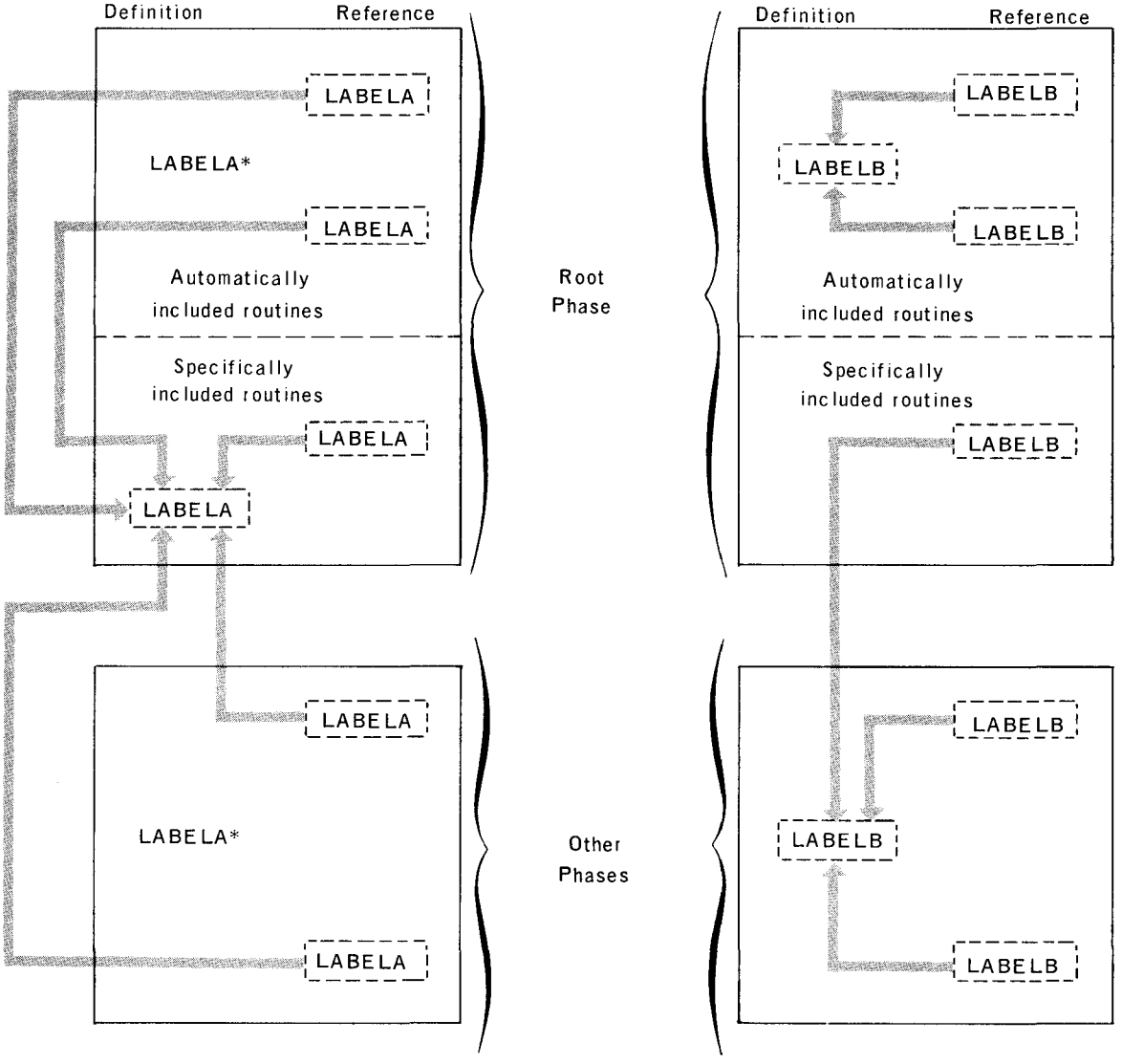
In the absence of the NOAUT option, the Linkage Editor performs automatic inclusion. When all control statements defining a load module have been read and the object modules have been obtained, the Linkage Editor determines whether each label declared as an external reference (EXTRN) by one object module has been defined as an entry point (ENTRY) in a previously included object module. If the label has not been defined, the Linkage Editor automatically searches the reserve library portion of the system library, or a library specified by a PARAM Job Control statement, for the label definition and includes the entire object module in which the label is defined. In a disc system, the MCL (Module Complex Library) is not searched during the automatic include search.

If the object module contains a control section whose name is defined in the first phase or in an object module previously included automatically, that named control section is not included. An entry point defined within that named control section does not cause the remainder of the object module to be included.

The search for label definitions continues until all labels are defined or until the specified library file is scanned once without creating new references which remain undefined. In this latter case, an error indication is given and processing then proceeds.

If a label is defined only in an object module that is included automatically, that definition is accepted and is the value used to satisfy any reference in the load module to the label. If a label is defined in the first phase of a load module and a second definition of the same label is provided by an automatically included object module, the second definition is ignored by the Linkage Editor. If a label is defined in a phase other than the first phase and is also defined in an automatically included object module, the latter definition is used to satisfy only those references within automatically included object modules.

Figure 2–2 illustrates the manner in which labels may be referenced when the labels are defined in the first phase, a phase other than the first, and in automatically included object modules.



*These definitions are deleted by the Linkage Editor.

NOTE: The arrows indicate which definition is used to satisfy each reference.

Figure 2–2. Referencing Label Definitions in a Load Module

If the control stream contains no Linkage Editor control statements defining the load module to be constructed, the Linkage Editor constructs a single load module from the first object module in OBJFIL (Tape Object File) or in MCL (Module Complex Library). OBJFIL and MCL are the files in which the language processors write their output in the tape and disc operating systems, respectively.

2.3. MULTIPHASE LOAD MODULES

The Linkage Editor can produce a load module that consists of more than one phase. The name and relative position of the phase within the load module are determined by the placement of the OVERLAY statement that creates the phase. The Linkage Editor can also produce a multiphase load module with the necessary information and subroutines to load the phases automatically.

2.3.1. Phase Name

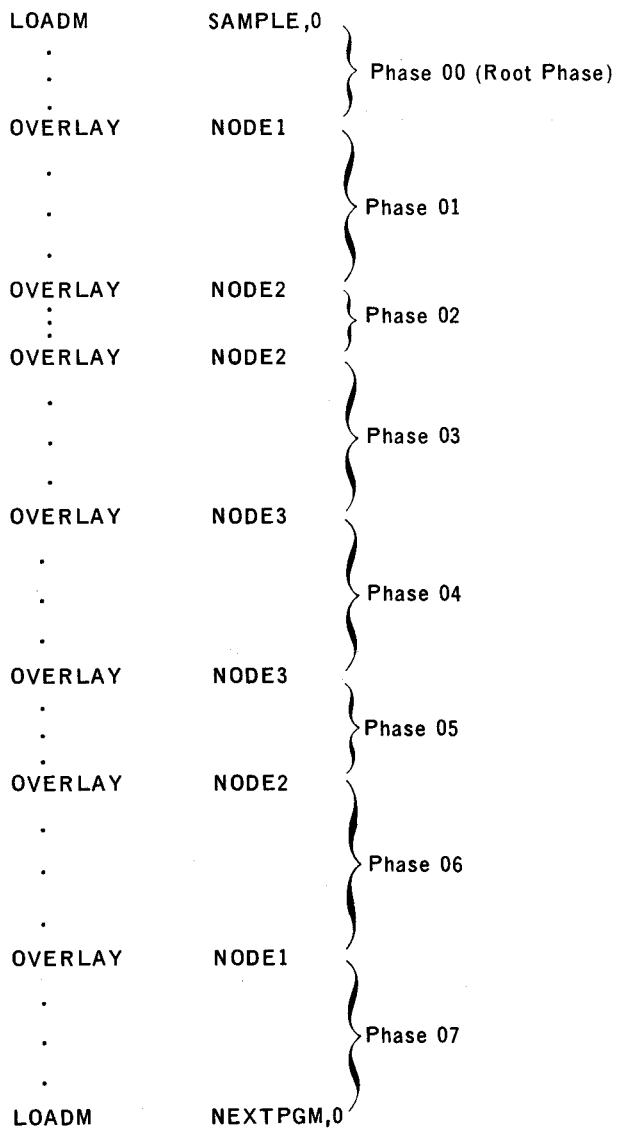
The user does not explicitly supply the phase name. The phase name is constructed by the Linkage Editor from the load module name and consists of eight alphanumeric characters. The first six characters are the same as the load module name as specified by the LOADM control statement (see 3.2). The last two characters of the phase name are the phase number. The Linkage Editor numbers the phases consecutively from 00 to 99 (decimal) in the order in which the phases are defined by Linkage Editor control statements.

2.3.2. Phase Structure Definition

In a load module the starting address of each phase other than the root phase is defined by the symbolic label in the operand field of an OVERLAY control statement (see 3.6). This starting address is the terminal address of a previous phase (adjusted to the next multiple of 8) and is called a node point; this may also be the starting address of other phases in the load module structure. OVERLAY statements with the same symbolic label in the operand field define additional phases with the same node point.

Until a subsequent OVERLAY statement is encountered, the object modules and control sections specified by INCLUDE statements define a single phase.

Figure 2–3 illustrates the phase structure definition of the load module SAMPLE which contains seven OVERLAY statements. These OVERLAY statements define the node points for the phases within the load module. Figure 2–4 shows the storage allocated to this same load module at execution time.



NOTE: The ellipses represent INCLUDE statements for object modules to be included in a particular phase. These statements must follow the proper LOADM or OVERLAY statement.

Figure 2–3. Phase Structure Definition

When an OVERLAY statement is encountered by the Linkage Editor that refers to a previously defined node point, all intervening node points are eliminated. In the load module SAMPLE illustrated in Figures 2–3 and 2–4, once phase 6 is defined, no additional phases may be defined starting at NODE3. Once phase 7 is defined, no additional phases may be defined starting at NODE2 or NODE3. If phase 7 is followed by an OVERLAY NODE2 statement, this statement defines a completely new node point and a subsequent phase would immediately follow phase 7 in storage. References to a given label in the load module may be traced along a "path" from one phase through another back to the root phase. Phase x is said to be on the path of phase y if they are the same phase or if they are on the same path but phase x is closer to the root phase than phase y. The root phase is on the path of every other phase. In Figure 2–4 phases 0, 1, 3, and 4 are on the path of phase 4 as indicated by the shading. However, phase 4 is not on the path of phases 0, 1, or 3. Phase 1 is not on the path of phase 7; phase 7 is not on the path of phase 1. The maximum number of phases allowed on any path is 10.

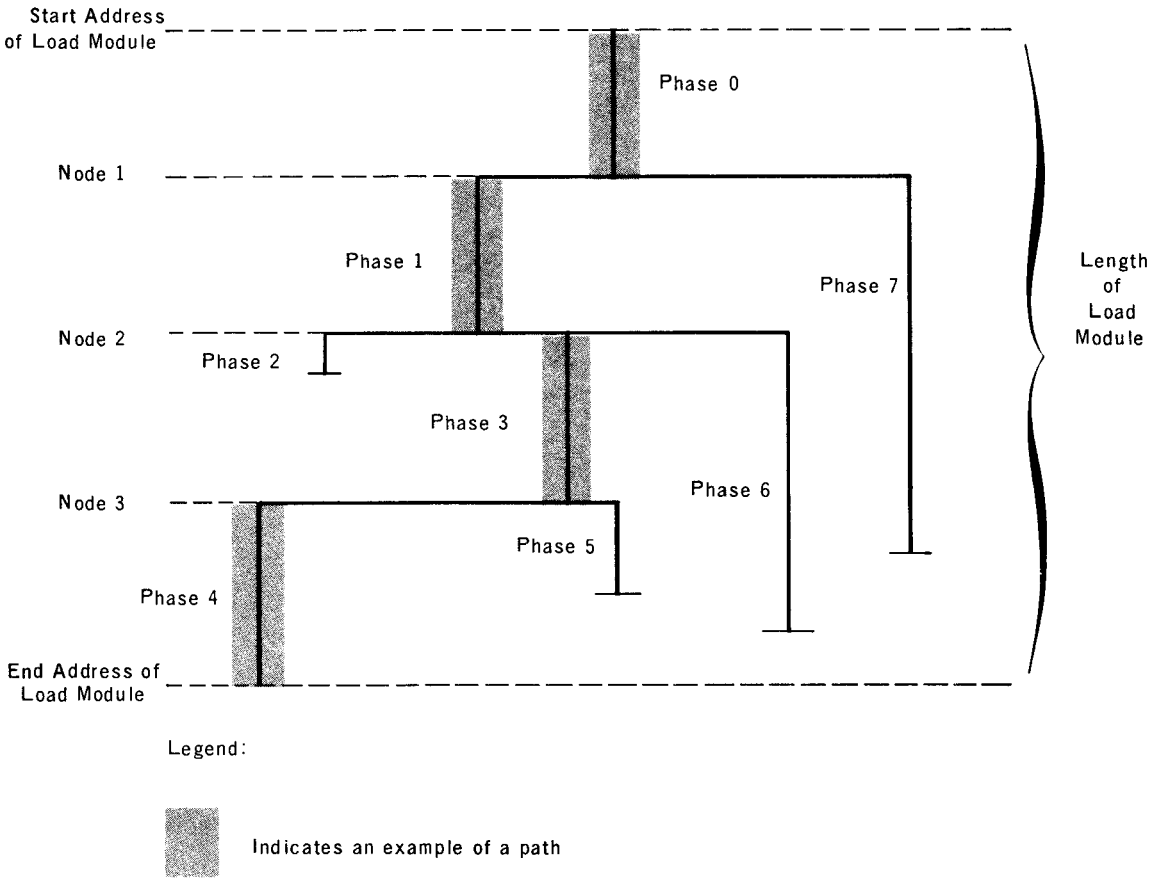


Figure 2–4. Storage Allocation for Execution of Load Module SAMPLE

The end address of the load module is the last addressable location of the longest path in the load module. This address is defined as KE$ALP and may be declared as an external reference (EXTRN) in any object module included in the program.

2.3.3. Automatic Loading of Phases

Phases of a load module may be automatically loaded in an overlay structure by specifying the V option of the LINKOP control statement (see 3.3). With this option, branching to a routine which may be in another phase is accomplished by loading into register 15 an address constant of type V which references the label of the required routine and by executing a branch, under control of register 15, to that routine. This transfers control to an overlay control routine which performs the following:

- It checks whether or not the required phase is already in main storage. If it is, the overlay control routine branches directly to the proper address.
- If the required phase is not in storage, it loads that phase and any phase on its path that is not in storage.
- It records the phases currently in main storage and then branches to the proper address.

When the V option is selected, the Linkage Editor processes address constants of type V in a special manner. If there is at least one such constant addressing a label which is not on the path of the reference, automatic loading of overlays is possible. The Linkage Editor constructs a phase table and an entry point table and includes them in the root phase. It also creates references to the entry point of the overlay control routine so that this routine may be included by the automatic inclusion mechanism. The phase table and entry point table are used by the overlay control routine to determine which phase is required by the current reference and whether that phase presently resides in main storage. The starting address of the phase table is defined as KL$PTB; the starting address of the entry point path is defined as KL$NTB; and the starting address of the entry point path is defined as KL$OCP1. Linkage Editor automatically generates these names as entry points, which permits the program being linked to reference them. These names must be unique and not duplicated within the program.

A problem program using automatic loading of phases must not issue any LOAD or FETCH macro instructions. The overlay control routine executes LOAD macro instructions as needed to obtain the required phases.

The address constant of type V loaded into register 15 for the automatic loading of overlays has the following rules concerning its use.

- It always occupies four bytes. The most significant byte is reserved for system use.
- It is used only for branching and may not be used for addressing data. Data references do not cause the referenced phase to be loaded automatically.
- When an address constant of type V addresses a phase on the same path, the constant is treated as though it were a constant of type A. If such a constant is loaded into register 15, the problem program branches directly to the required location rather than transferring control to the overlay control routine.

Addressing a label which is multiply defined does not cause a phase to be loaded automatically. If a multiply defined label is referenced by an address constant of type V, it must be defined on the path of each such reference.

2.4. AUTOMATIC DELETION

If an INCLUDE statement specifies an object module which contains a control section previously defined in a phase on the path of the current phase, the control section is not included in the load module. The control section is said to be automatically deleted. Unlabeled control sections are not deleted automatically.

If more than one control section has the same name as a labeled common storage area, the first control section with that name is treated as a Block Data subprogram to initialize the common storage, and all other control sections with the same name are deleted.

If an entry point definition, other than a control section record, is encountered for a name which has already been defined on the path of the current phase, the new definition is flagged as an error and ignored.

Therefore, with automatic deletion for a program consisting of a single phase, only one definition is accepted for each label.

3. CONTROL STATEMENTS

3.1. GENERAL

Input to the Linkage Editor consists of control statements and one or more object modules. The control statements direct the construction of a load module from the object modules. The Linkage Editor control statements that define the load module are contained in the control stream, beginning with a LOADM control statement. These statements are terminated by another LOADM statement or by the end of data. Object modules to be included in the load module may be interspersed among these control statements are terminated by the end of data. Object modules to be included in the load module may be interspersed among these control statements.

3.1.1. Placement of Control Statements

Control statements may be placed before, between, or after the control sections in an object module, but may not be placed within a control section. Some specific restrictions on placement are imposed by the nature of the functions requested by the control statement. Any such restrictions are noted in the discussion of each control statement.

3.1.2. Control Statement Conventions

The general format of a Linkage Editor control statement is the same as that of an Assembler statement. The label field is blank for all Linkage Editor control statements except the EQU statement (see 3.5). The field begins in column 1, is terminated by a blank column, and may contain up to eight alphanumeric characters. The operation field must be preceded and followed by at least one blank column. If there is no label, the operation field may start in column 2. It contains the name of the function to be performed. The operand field contains one or more parameters, depending on the control statement used, and begins with the first nonblank following the operation field. The field cannot extend beyond column 71 and is terminated by a blank.

The conventions used to illustrate control statements in this manual are as follows:

- Capital letters and punctuation marks (except brackets and ellipses) indicate information that must be coded exactly as shown.
- Lowercase letters and terms represent terms that must be supplied by the user.
- Information contained within brackets represent optional entries that are included or omitted (depending on program requirements).
- An ellipsis (a series of three periods) indicates the presence of a variable number of entries.

3.2. LOADM – LOAD MODULE

The LOADM control statement specifies the name and the starting address of the load module. The LOADM statement is the first control statement for each load module and indicates that the Linkage Editor can begin constructing the specified load module.

The format of the LOADM statement is:

LABEL	OPERATION	OPERAND
	LOADM	name[,addr]

where:

name specifies the name of the load module. The name may be from one to six alphanumeric characters, the first of which must be alphabetic. If the name is less than six characters, it is padded on the right with EBCDIC zeros. If there are no Linkage Editor control statements in the control stream, the name of the load module is taken as the first six characters of the name of the first object module included from OBJFIL for the tape system or from MCL for the disc system.

addr is the starting address of the load module and may be a hexadecimal number of the form X'nnnnnn' or a decimal number. If this parameter is omitted, the address of the load module is set to zero. If I/O protection is requested, the starting address may be adjusted by the Linkage Editor so that the I/O protect area ends at an address that is a multiple of 512.

3.3. LINKOP – OPTIONS

The LINKOP control statement specifies the options to be used in creating the load module. The options are written in the operand field. This control statement must immediately follow the LOADM control statement. The format of the LINKOP statement is:

LABEL	OPERATION	OPERAND
	LINKOP	$p_1, \ldots, p_n$

where:

$p_1, \ldots, p_n$ specifies the options to be used. They may be listed in any order and must be separated by commas, with no intervening spaces. The options are explained in the following paragraphs.

- NOAUT – This option suppresses the automatic inclusion function. Normally, the Linkage Editor searches the system library to resolve any remaining undefined references. (See 2.2.3 for further description of automatic inclusion.)

- NOV – A load module is generated in a form that is not suitable for the automatic loading of overlays. Phase and entry point tables are not produced. All constants of type V are produced as though they were type A address constants and no reference to the overlay control routine is generated. (See 2.3.2 and 2.3.3 for further information of the use of overlay.)

- V – A load module is generated in a form suitable for the automatic loading of overlays. There must be at least one constant of type V which represents a reference that could cause an overlay to be loaded. Then, phase and entry point tables are produced for the load module, and all constants of type V receive special processing. If the NOAUT parameter is also specified, the overlay control routine must be included in the root phase. If neither the V nor the NOV option is specified, the V option (automatic loading of phases) is assumed.

- NOP – A load module is generated in a form that does not provide for I/O protection. (See 2.2.1 for further information on I/O protection.)

- P – A load module is generated in a form that is suitable for I/O protection. Whether or not the protection is provided is determined when the load module is executed. If neither the P nor the NOP option is specified, the NOP option is assumed.

3.4. INCLUDE – INCLUDE OBJECT MODULE

The INCLUDE control statement requests that a specified object module or selected control sections of a specified object module be included in the phase of the load module being constructed. The INCLUDE statement may follow any Linkage Editor control statement except ENTER. It may also follow the header record of an object module in the library. The format of the INCLUDE statement is:

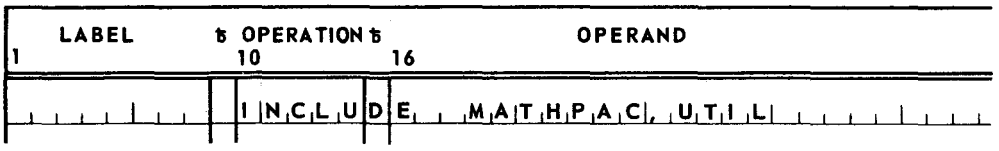
LABEL	♭ OPERATION ♭	OPERAND
	INCLUDE	[modulename][(s_1,...,s_n)][,filename]

where: modulename is the name of the object module to be included. If this parameter is omitted, it is assumed that the object module immediately follows this statement in the control stream.

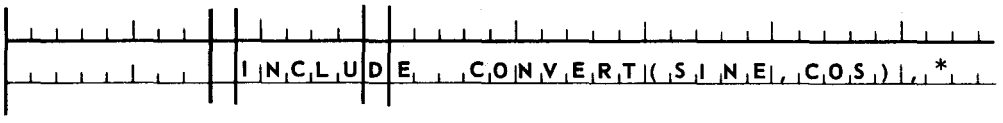
(s_1,...,s_n) specifies the control sections to be included from the specified object module. Up to nine control sections may be specified and only these control sections are included in the phase being constructed. The order of the control section records in the object module determines the order of the control sections in the phase being constructed. If this parameter is omitted, the entire object module is included, except for those control sections which may be deleted automatically (see 2.4).

filename — is the symbolic name of the file in which the object module is located and must be the same as that specified on the LFD Job Control statement for this file. An asterisk specified for the filename indicates that the object module is in OBJFIL for a tape system or in MCL for a disc system. If this parameter is omitted, the object module is assumed to be in the system library.

An object module that is specified by an INCLUDE statement in the control stream may contain an INCLUDE statement specifying an object module. However, the object module specified by this last INCLUDE statement must not contain an INCLUDE statement. For example, if the following control statement is located in the control stream for a phase being constructed,



the object module MATHPAC located in the user file name UTIL is included in the phase. MATHPAC might specify the following:



This causes the control sections named SINE and COS from the object module CONVERT located in OBJFIL (or in MCL) to be included in the phase.

3.5. EQU – LABEL DEFINITION

The EQU control statement provides the Linkage Editor with the value of a label that would not otherwise be defined. The format of the EQU control statement is:

LABEL	OPERATION	OPERAND
symbol	EQU	expression

where: symbol — is the label to be defined. Definition of a symbol by an EQU statement is subject to the same rules for automatic deletion as entry points (see 2.4).

expression is the expression that defines the value to be assigned to the label. The expression must have one of the following forms:

- a decimal number of one to eight digits
- a hexadecimal number of one to six digits in the form X'nnnnnn'
- a previously defined label (the name of a control section or an entry point in an object module that was previously included, or by a previous EQU statement)
- a previously defined label plus or minus a decimal number or a hexadecimal number in the above form.

3.6. OVERLAY – BEGIN OVERLAY PHASE

The OVERLAY control statement indicates the beginning of an overlay phase (a phase other than the root phase) and defines the relative position of the phase within the load module structure (see 2.3.2). The object modules included thereafter constitute a single, separate phase until the next OVERLAY statement, LOADM statement, or the end of data. The format of the OVERLAY control statement is:

LABEL	b OPERATION b	OPERAND
	OVERLAY	symbol

where: symbol is the name of the node point that defines the starting address of the phase. The symbol consists of one to eight alphanumeric characters.

The starting address of a phase is not necessarily the entry point to that phase. Therefore, the use of the symbol in the operand field of an OVERLAY statement does not define the symbol as an entry point. However, the same symbol may also be used to define the entry point to this phase, or some other phase, without creating a duplicate definition or a conflict.

3.7. MOD – MODIFY LOCATION COUNTER

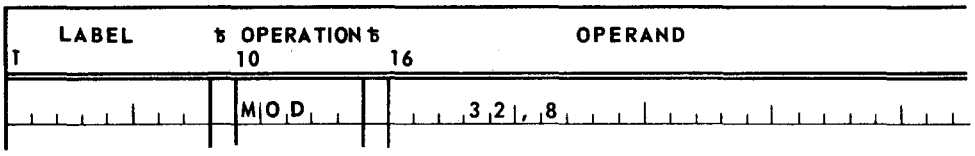
The MOD control statement instructs the Linkage Editor to adjust its location counter to the next value which is greater than, or equal to, its present value and which has a specified remainder when divided by a given power of 2. Initially, the location counter is set by the LOADM statement; it is incremented by the length of each control section included in a phase, and it is also set to the value associated with the related node point when an OVERLAY statement is encountered in the control stream. The format of the MOD statement is:

LABEL	b OPERATION b	OPERAND
	MOD	power[,remainder]

where: power is the power of 2 relative to which the location counter is to be adjusted. The acceptable powers of 2 are: 16, 32, 64, 128, 256, and 512.

remainder is the desired remainder of the new value of the location counter relative to the acceptable power of 2. The remainder must be a multiple of 8. If it is not a multiple of 8, the remainder is rounded up to the next higher multiple of 8. It is then truncated to a value less than the power of 2 used. If this specification is omitted, the remainder is considered to be zero.

Example:



This statement instructs the Linkage Editor to adjust the location counter to a value greater than, or equal to, the present value that is 8 more than a multiple of 32. If the current value of the location counter is 16384, which is a multiple of 32 (namely, 32 times 512), the new value would be 8 greater, or 16392. If the current value is 16392, the new value would be the same; but if the current value is 16400 (which is 16 greater than a multiple of 32; namely, 32 times 512 plus 16), the new value would be 16424.

3.8. ENTER – ENTRY POINT STATEMENT

The ENTER control statement provides the address of the entry point of the current phase. This is the address to which control is normally transferred if the phase is loaded by a Supervisor FETCH macro instruction. The ENTER statement is the last control statement for a phase. It may be followed only by the OVERLAY statement for the next phase or the LOADM statement for the next load module.

If no ENTER statement is provided for a phase, the transfer address is obtained from the first Transfer record encountered in the phase, which has a valid address. If no object module supplies a valid transfer address, the address used is the lowest address assigned to the phase.

The format of the ENTER control statement is:

LABEL	OPERATION	OPERAND
	ENTER	expression

where: expression is the transfer address for the phase. The expression may have one of the following forms:

- a decimal number of one to eight digits
- a hexadecimal number of one to six digits in the form X'nnnnnn'
- a previously defined label (the name of a control section or an entry point in an object module that was previously included, or by a previous EQU statement)
- a previously defined label plus or minus a decimal number or a hexadecimal number in the above form.

3.9. RES – RESERVE STORAGE

The RES control statement instructs the Linkage Editor to reserve space in main storage following KE$ALP which defines the end of the longest path.

The format of the RES statement is:

LABEL	↹ OPERATION ↹	OPERAND
	RES	value

where: value is the number bytes in main storage to be reserved. Value may have one of the following forms:

- a decimal number of one to eight digits
- a hexadecimal number of one to six digits in the form X'nnnnnn'

APPENDIX A. SAMPLE LOAD MODULE SPECIFICATION

The control stream for a sample Linkage Editor run is shown in Figure A–1. It includes the Linkage Editor control statements and the immediately preceding and following Job Control statements. The Job Control statements JOB, DVC, LFD, and VOL are not shown.

```
                LABEL          OPERATION          OPERAND
                1              10                 16
(1)  //EXEC    LINK
(2)  //PARAM   LIN=SY         SRES/MYLIB
(3)  /$
(4)           LOADM            TSTMAT
(5)           INCLUDE          FLTGPT
             INCLUDE          TSTPGM1(TSTPGM1),MYLIB
(6)           ENTER            TSTSTART
(7)           OVERLAY          A
(8)           INCLUDE          TSTPGM1(PHASE1),MYLIB
(9)           OVERLAY          B
(10)          INCLUDE          TSTPHS2,MYLIB
(11)          OVERLAY          B
(12)          INCLUDE          TSTPH3,MYLIB
(13)          OVERLAY          A
(14)          MOD              256,0
              INCLUDE          TSTPH4,*
              MOD              256,0
              INCLUDE          SINCOS,*
(15) /*
```

Figure A–1. Sample of a Control Stream for a Linkage Editor Run

NOTES:

(1) Load and execute the Linkage Editor program.

(2) The PARAM statement informs the Linkage Editor that MYLIB is a library to be used in the linking process. The system library is also to be used for the automatic inclusion process, since it is the first name in the LIN specification.

(3) Start-of-data command.

(4) The name of the load module being constructed is TSTMAT. The starting address is assumed to be zero.

(5) The first phase (phase 0) contains the object module named FLTGPT and the control section TSTPGM1 from the object module named TSTPGM1. The phase name is TSTMAT00.

(6) The transfer address for phase 0 is TSTSTART.

(7) The OVERLAY statement defines node point A and defines the start of phase 1 which has the phase name of TSTMAT01.

(8) Phase 1 of the load module is constructed solely from the control section PHASE1 contained within the object module TSTPGM1.

(9) The OVERLAY statement defines node point B and defines the start of phase 2 which has the name of TSTMAT02.

(10) Phase 2 of the load module is constructed solely from the object module TSTPHS2.

(11) The OVERLAY statement defines the start of phase 3 at node point B. The phase name is TSTMAT03.

(12) Phase 3 contains the object module named TSTPH3.

(13) The OVERLAY statement defines the start of phase 4 at node point A. The phase name TSTMAT04. Note that node point B is no longer defined to the Linkage Editor.

(14) The MOD statements in phase 4 ensure that TSTPH3 and SINCOS both start at a multiple of 256. Since they were assembled relative to 0, this will make it easier to read the storage dumps obtained during debugging.

(15) End-of-data command.

FLTGPT is obtained from the system library; TSTPGM1, TSTPHS2, and TSTPH3 are in MYLIB. TSTPH4 and SINCOS are obtained from OBJFIL.

Figure A–2 illustrates the storage allocated to each object module and control section included in the load module TSTMAT coded in the example in Figure A–1.

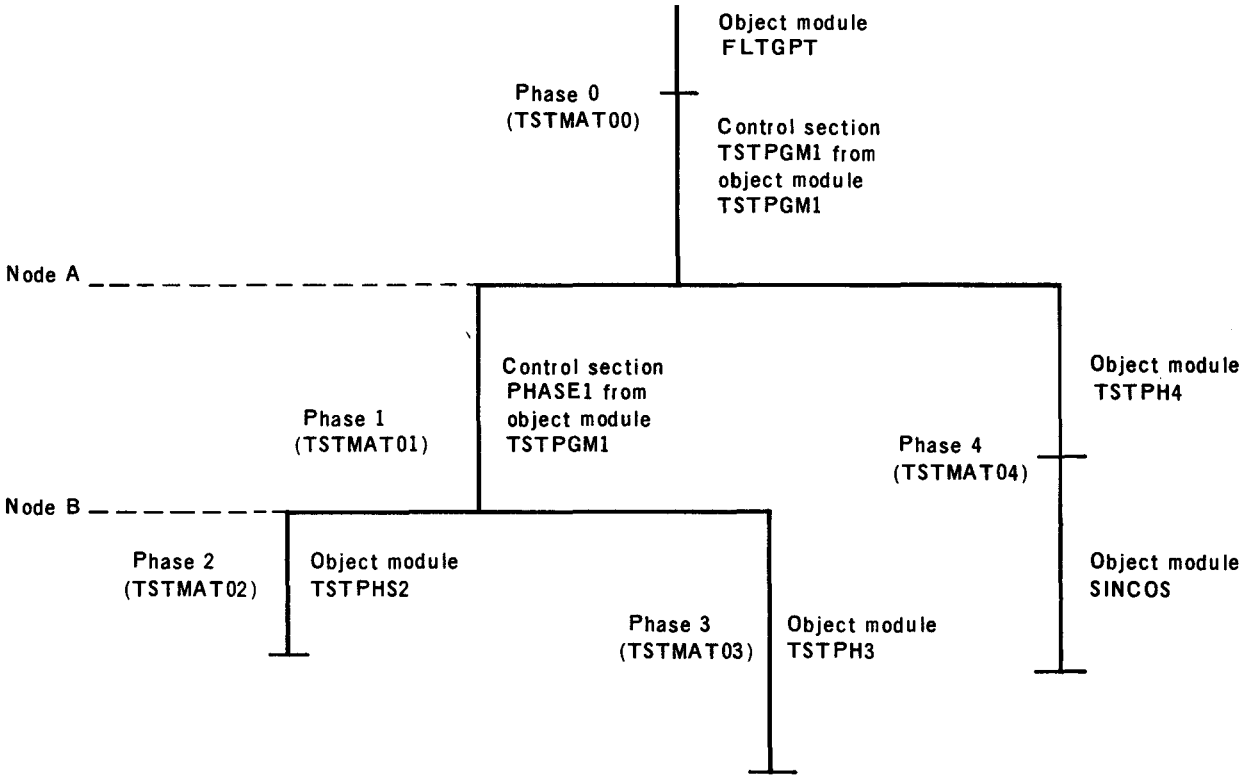


Figure A–2. Storage Allocation of the Load Module TSTMAT

APPENDIX B. USE OF THE PARAM STATEMENT

B.1. GENERAL

A general explanation of the use of the PARAM statement is contained in the *UNIVAC 9400 System Job Control Programmers Reference Manual, UP-7793* (current version). This appendix describes options that may be specified in the operand field of a PARAM statement for use by the Linkage Editor.

An error detected in the operand field of a PARAM statement results in the automatic abort of the job stop after any remaining operands in the statement are evaluated and any additional PARAM statements are evaluated.

Error messages are written on the line printer and a message indicating a program abort is sent to the console typewriter.

Termination of a job step occurs when any of the following conditions are detected:

(a) an end-of-data response is given,

(b) a start-of-data image is detected (/$), or

(c) a slash is not detected in column 1.

The remaining statements in that job are bypassed until the first statement of the next job step is encountered; statement processing then continues.

B.1.1. CNL Option

This option allows cancellation of a job step when an error in a PARAM statement occurs.

LABEL	OPERATION	OPERAND
	// PARAM	CNL=$\left\{\begin{matrix}(N)\\(E)\end{matrix}\right\}$

where: N indicates that the cancel macro instruction will be executed when an error occurs that prevents a load module from being generated.

E indicates that the cancel macro instruction will be executed when an error occurs which prevents a load module from being generated, or a load module contains errors that render it unexecutable.

B.1.2. LIN Option

This option is used to identify the library file from which object modules are to be included.

LABEL	♭ OPERATION ♭	OPERAND
	// PARAM	LIN=file-name-1/file-name-2

where: file-name-1 identifies the library file containing the object modules. The name can consist of up to eight characters.

file-name-2 identifies a second library file containing object modules.

The file names appearing in the LIN option must be as specified in an LFD control statement.

The file identified by file-name-1 is used for automatic include processing. If the object modules are to be included from the reserve library on the SYSRES tape, an LFD statement must specify the name SYSRES in the first LIN option.

If the LIN option is not used, the SYSRES library tape is used.

B.1.3. LST Option

This option allows the programmer to indicate the type of listing desired.

LABEL	♭ OPERATION ♭	OPERAND
	// PARAM	LST=$\left\{\begin{matrix}(N)\\(W)\end{matrix}\right\}$

where: N indicates that all listable outputs are to be inhibited.

W indicates that listings of warning diagnostics are to be inhibited.

The use of parentheses is optional. If the LST option is omitted, all listable outputs and warning diagnostics are listed.

B.1.4. OUT Option

This option allows selection of the type of Linkage Editor output.

LABEL	♭ OPERATION ♭	OPERAND
	// PARAM	OUT=$\left\{\begin{matrix}N\\T\end{matrix}\right\}$

where: N indicates that Linkage Editor output is to be inhibited. For disc Linkage Editor operation, output to the MCL is inhibited.

T indicates that an LDMFIL or an OBJFIL tape is to be produced in addition to the MCL output.

If the OUT option is omitted, a load module is produced. When the T specification is used with this option, either an LFD LDMFIL or an LFD OBJFIL statement must appear in the control stream. If neither control statement is included in the control stream, the disc Linkage Editor aborts immediately. Standard system conventions for stacking load modules using the tape Linkage Editor also apply when using the disc Linkage Editor with OUT=T.

B.1.5. VER Option

This option allows a change to be made to the version number of an object module.

LABEL	♭ OPERATION ♭	OPERAND
	// PARAM	VER=level-number/update-number

where: level-number is one or two decimal numbers (0 to 99) representing a level number.

update-number is one or two decimal numbers (0 to 99) representing an update number.

If the VER option is not used, the version number of the first object module being included is used as the version number of the load module.