File No. S360-31
Form Y28-6667-0

Program Logic

IBM System/360 Operating System

Linkage Editor (F)

Program Number S360-ED-521

This publication describes the internal logic of the IBM System/360 Operating System Linkage Editor (F), Version 2, with design points of 44K, 88K, and 128K. It identifies areas of the program that perform specific functions and relates those areas to the program listing.

The linkage editor, a processing program, combines and edits modules to produce a load module that can be loaded into main storage by the control program. The linkage editor:

- Allocates storage, analyzes attributes and options, and initializes tables and buffers. (Initialization)
- Transforms input into an internal format for subsequent processing. (Input Processing)
- Assigns relative storage addresses to external symbols, writes records on the output data set, and produces an optional module map and/or cross-reference table. (Intermediate Processing)
- Relocates address constants found in the input text, and writes the remaining records on the output data set. (Second Pass Processing)
- Completes the partitioned data set directory for the output data set, produces an error diagnostic directory, and releases storage allocated to the linkage editor. (Final Processing)

This program logic manual is directed to the IBM customer engineer who is responsible for program maintenance. Because program logic information is not necessary for program operation and use, distribution of this manual is restricted to persons with program maintenance responsibilities.

Restricted Distribution

First Edition (January 1968)

This publication corresponds to Release 15.

Specifications contained herein are subject to change from time to time. Any such changes will be reported in subsequent revisions or Technical Newsletters.

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

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

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

PREFACE

This publication provides customer engineers and other technical personnel with information describing the internal organization and logic of the level F linkage editor, version 2. It is part of an integrated library of IBM System/360 Operating System Program Logic Manuals. Other publications that are required for an understanding of the linkage editor are:

IBM System/360 Operating System:

- Introduction to Control Program Logic, Program Logic Manual, Form Y28-6605
- Concepts and Facilities, Form C28-6535
- Linkage Editor, Form C28-6538
- Assembler Language, Form C28-6514

The reader should also refer to the corequisite publications:

IBM System/360 Operating System:

- Storage Estimates, Form C28-6551
- System Control Blocks, Form C28-6628

This manual consists of seven parts:

1. An Introduction, which describes the linkage editor as a whole, including its relationship to the operating system. The major divisions of the program and the relationships among them are also described in this section.

2. A Method of Operation section which provides: (a) an overview of, and an introduction to the logic of the linkage editor, and (b) detailed descriptions of specific operations. Operation diagrams, included at the end of this section, are designed to be used with the text, and illustrate the flow of data through tables and buffers used during linkage editor processing.

3. A section describing the organization of Linkage Editor F. Program components (modules, control sections, and routines) are described both in terms of their operation and their relation to other components. Flowcharts are included at the end of this section.

4. A directory which helps the reader find named areas of code in the program listing, which is contained on microfiche cards.

5. A section illustrating the layouts of tables used by linkage editor F. Table layouts may not be essential for an understanding of the basic logic of the program, but are essential for analysis of storage dumps.

6. Diagnostic aids, including general register contents at entry to modules, and an error message -- module cross reference table.

7. An appendix, which includes input conventions and record formats.

If more detailed information is required, the reader should refer to the comments and coding in the linkage editor program listings.

CONTENTS

FIGURES

TABLES

CHARTS

OPERATION DIAGRAMS

This section provides general information describing the purpose, organization, and internal operation of the linkage editor, and its relationship to the operating system.

The level F linkage editor, version 2, (hereinafter referred to as the linkage editor) is available in 44K, 88K, and 128K design points; they differ in speed and table size. The 44K and 88K design points use different overlay structures, and the 128K design point is not in overlay. All versions of the linkage editor operate in essentially the same manner.

PURPOSE OF LINKAGE EDITOR

The linkage editor is one of the processing programs of IBM System/360 Operating System. It is a service program used in association with the language translators to prepare machine-language programs from symbolic-language programs written in FORTRAN, COBOL, report program generator, the assembler language, or PL/I. Linkage editor processing is a necessary step that follows source program assembly or compilation.

Linkage editor processing allows the programmer to divide his program into several parts, each containing one or more control sections. Each part may then be coded in the programming language best suited to it and may then be separately assembled or compiled by a language translator (under the rules applicable to each language translator).

The primary purpose of the linkage editor is to combine and link object modules (the output of the language translators) into a load module in which all cross references between control sections are resolved as if they had been assembled or compiled as one module. The load module produced by the linkage editor consists of executable machine-language code in a format that can be loaded into main storage and relocated by program fetch.

In addition to combining and linking object modules, the linkage editor performs the following functions:

- Library Calls. Modules (such as standard subroutines) stored in a library can be placed in the input to linkage editor, either automatically or upon request. If unresolved external references remain after all input to the linkage editor is processed, an automatic library call routine retrieves the modules required to resolve the references.
- Program Modification. Control sections can be replaced, deleted, or rearranged (in overlay programs) during linkage editor processing, as directed by linkage editor control statements. Common control sections generated by the FORTRAN, PL/I, and assembler language translators are provided locations within the output load module.
- Overlay Module Processing. Linkage editor prepares modules for overlay by assigning relative locations within the module to the overlay segments and by inserting tables to be used by the overlay supervisor during execution.
- Options and Error Messages. The linkage editor can:
  1. Process special options that override automatic library calls or the effect of minor errors.
  2. Produce a list of linkage editor control statements that were processed.
  3. Produce coded diagnostic messages and a directory describing those diagnostic messages that were printed out during linkage editor processing.
  4. Produce a module map or cross-reference table of control sections in the output load module.

RELATIONSHIP TO THE OPERATING SYSTEM

The linkage editor has the same relationship to the operating system as any other processing program. Control is passed to the linkage editor in one of three ways:

1. As a job step, when the linkage editor is specified on an EXEC job control statement in the input stream.
2. As a subprogram, via the execution of a CALL macro instruction (after execution of a LOAD macro instruction), a LINK macro instruction, or an XCTL macro instruction.

3. As a subtask, in multitasking systems, via execution of the ATTACH macro instruction.

GENERAL DESCRIPTION

Linkage editor input may consist of a combination of object modules, load modules, and linkage editor control statements. The prime function of the linkage editor is to combine these modules, in accordance with requirements stated on control statements, into a single output load module that can be relocated and loaded into main storage by program fetch for execution. Output load modules are placed in partitioned data sets (libraries).

Each module to be processed by linkage editor has an origin that was assigned during assembly, during compilation, or during a previous execution of the linkage editor. Each module in the input to linkage editor may contain symbolic references to control sections in other modules; such references are called external references.

To produce an executable output load module, the linkage editor:

1. Assigns relative main storage addresses to the control sections to be included in the output module. Since each input module has an origin that was assigned independently by a language translator, the order of the addresses in the input is unpredictable. (Two input modules, for example, may have the same origin.) Linkage editor assigns an origin to the first control section and then assigns addresses, relative to this origin, to all other control sections in the output.[1] Each item in a control section is relocated the same number of bytes as the control section origin.

2. Resolves external references in the input modules. Cross references between control sections in different modules are symbolic, and must be resolved (translated into relocatable machine addresses), relative to the contiguous main storage addresses assigned to the output load module.

   These symbolic cross-references are made by means of address constants. The linkage editor calculates the new address of each relocatable expression in a control section and determines the assigned origin (value) of the item to which it refers.

Linkage editor processing is affected by specified options, operations requested on control statements, module attributes contained in partitioned data set directories, and control information contained within the modules themselves. The following paragraphs describe the relationship of module structure, selected options, and module attributes to linkage editor processing.

MODULE STRUCTURE

Object modules and load modules have the same basic logical structure (see Figure 1). Each consists of:

- Control dictionaries, containing the information necessary to resolve symbolic cross references between control sections of different modules, and to relocate address constants.

- Text, containing the instructions and data of the program.

- An end of module (EOM) indicator (END statement in object modules; EOM indication in load modules).

Each language translator usually produces two kinds of control dictionaries: an external symbol dictionary (ESD) and a relocation dictionary (RLD). An object module always contains an ESD; a load module contains an ESD, unless it is marked with the "not editable" attribute. Object and load modules usually contain an RLD (unless there are no relocatable address constants in the module). Control dictionary entries are generated when external symbols, address constants, or control sections are processed by a language translator.

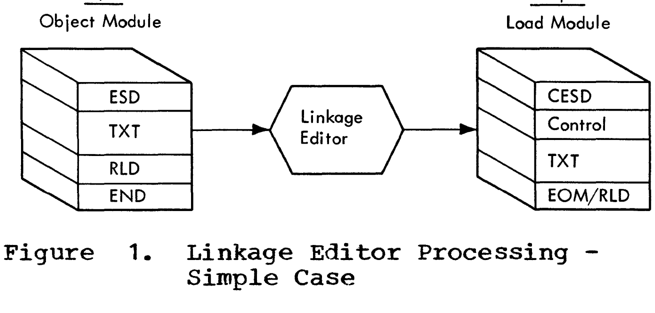


Figure 1. Linkage Editor Processing - Simple Case

[1] If the program is in overlay, an origin is assigned to the first control section in each segment. Within each segment, contiguous addresses are assigned relative to the segment origin.

External Symbol Dictionary

The external symbol dictionary contains entries for all external symbols defined or referred to within a module. (An external symbol is one that is defined in one module and can be referred to in another.) Each entry identifies a symbol, or a symbol reference, and gives its location, if any, within the module. When combining input modules, linkage editor resolves references between different input modules by matching the referenced symbols to defined symbols; it does this by searching for the external symbol definitions in each input module's ESD. There is an ESD entry for each named control section and each named common area. The ESD also contains entries that identify unnamed control sections and unnamed common areas.

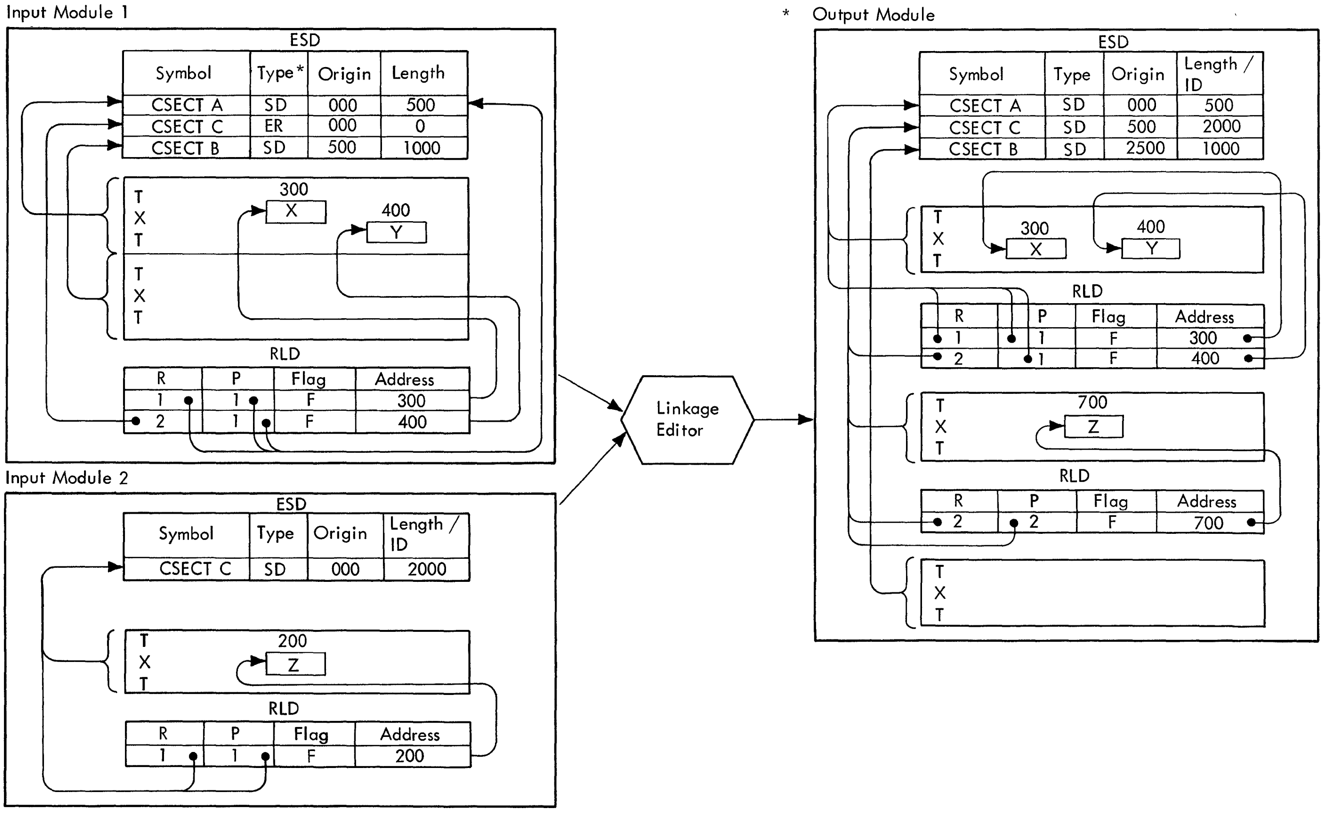
Relocation Dictionary

The relocation dictionary (RLD) lists all relocatable address constants that must be modified when the linkage editor produces an output load module. The linkage editor uses the RLD whenever it processes a module. The RLD is also used to adjust the value of address constants after program fetch reads an output load module from a library and loads it into main storage for execution. The RLD contains at least one entry for every relocatable address constant in a module. An RLD entry identifies an address constant by indicating both its location within a control section and the external symbol (in the ESD) whose value must be used to compute the value of the address constant.

Composite Dictionaries

An output load module is composed of all input object modules and input load modules processed by the linkage editor (except those that are replaced or deleted). The control dictionaries of an output module are therefore a composite of all the control dictionaries in the linkage editor input. The control dictionaries of a load module are called the composite ESD (CESD) and the RLD.

Figure 2 shows how the control dictionaries of two input modules are combined into composite dictionaries by the linkage



*See "ESD Record Types"

Figure 2. Combining Control Dictionaries

editor. The control dictionaries and their associated text are interrelated through a system of line numbers and pointers. Within an input module, each ESD item on which an address constant may depend has a line number (ESD identifier, or ESD ID); the line number indicates the position of the item, relative to the other ESD items associated with the text.[1] Every item of text in an object or load module has associated control information that describes it. This control information includes the ESD ID of the ESD item for the control section that contains the text. (In Figure 2, the ESD ID of the text item that contains X and Y points to line 1 of the ESD for input module 1. The ESD ID of the text item containing Z points to line 1 of the ESD for input module 2.)

Each RLD item must point to two ESD items:

1. The ESD item for the symbol on which the address constant depends. This is referred to by the RLD relocation pointer (R pointer).

2. The ESD item for the control section that contains the address constant. This is referred to by the RLD position pointer (P pointer).

In input module 1, X and Y are address constants in the same control section (CSECT A). X refers to a symbol in CSECT A; therefore, both pointers of its associated RLD item refer to the ESD entry for CSECT A (line 1). The value field of Y refers to a symbol in a different control section (CSECT C); therefore, the R pointer of its associated RLD points to the ESD entry for the external reference (line 2), whereas the P pointer refers to the ESD entry for its control section (line 1).

When the linkage editor combines the input modules, it must maintain this system of pointers by renumbering the ESD items to reflect their relative positions in the CESD of the output module. It must also update the RLD pointers and control information for the text so that they refer to the renumbered CESD items; the resulting CESD and RLD items are shown in Figure 2.

SELECTED OPTIONS

Linkage editor processing also depends on selected options. Figure 1 shows a simple case in which a single object module, containing only one control section, is processed by the linkage editor for block loading.

Figure 3 shows the processing of an object module and a load module, each containing several control sections. In this example, test translator macro instructions were included in an assembler language source program and test symbol (SYM) records were produced by the assembler language translator. The TEST and overlay options were specified on the execute (EXEC) statement and overlay control statements were included in the input to linkage editor. With these options, the output load module produced by the linkage editor contains:

- SYM records to be used by the test translator. (If the TFST option is not specified on the EXEC statement, SYM records in input are not included in the output load module.) These records contain blocked SYM and ESD statements created during a previous execution of linkage editor. SYM records in load modules are passed through the linkage editor unmodified to the output device.

- A composite ESD. CESD records contain the ESD items for the module. There is a maximum of 15 ESD items per record on the output device. The first eight bytes of the CESD record contain control information pertaining to the ESD items in the record. This information consists of the ESD ID of the first ESD item and the number of bytes of ESD items in the record.

- A control record, or a composite control/RLD record, preceding each text record. The RLD portion, if present, contains the RLD items used to relocate the previous text.[2] The control portion may contain:

  1. An end of segment (EOS) indication, if the following text record

[1] In an object module, one type of ESD item (LD) may not have associated text or address constants that depend on it. (Refer to "ESD Processing.") Such ESD items are excluded from the numbering system.

[2] If there is a large number of RLD items for the previous text, there may be several RLD records preceding the next text record. The last of these is a control/ RLD record.

is the last text record of an overlay segment.[3]

2. An end of module (EOM) indication, if the following text record is the last text record of the module.[3]

3. The number of bytes of RLD information that follow, if it is a composite control/RLD record.

4. The number of bytes of control information.

The control portion also contains the IDs and lengths (in bytes) of all the control sections in the following text, to a maximum of 60, and a channel command word (CCW). The channel command word contains the address assigned by the linkage editor to the first byte of that record, plus the total length of the record. This information is used by program fetch to read the following text.

Note: The control portion contains as many IDs and lengths as there are control sections in the following text record.

- Text for each control section. Text records contain the instructions and data for the module. In overlay, the linkage editor produces two special types of text records, the segment table (SEGTAB) and entry table (ENTAB). The SEGTAB, located in the root segment, is used by the overlay supervisor to keep track of the relationship of segments during execution. The ENTAB is a separate control section that may be created by the linkage editor for each overlay segment. An ENTAB is used by the overlay supervisor to determine the segment to be loaded when a segment not in the path is referred to.

- A note list. The note list gives the location of each overlay segment in the output module library.

Figure 4 shows the module structure when the scatter loading and test options are requested. With these options, the output load module contains:

- SYM records.

- A composite ESD.

- A scatter/translation record used by program fetch to compute the relocated addresses required for scatter loading the module into the main storage. The record contains a scatter table and a translation table. The scatter table is a list of control section addresses; the translation table correlates the CESD entry for each control section with the address indicated in the scatter table. (When a load module in scatter format is processed again by the linkage editor, this information is ignored.)

- Text for each control section, preceded by a control/RLD record describing it. (Any RLDs pertaining to a text record are contained in the control/RLD record that follows it.)

- An EOM indication that marks the end of the module.

The Appendix (Section 7) contains the format of each record type.

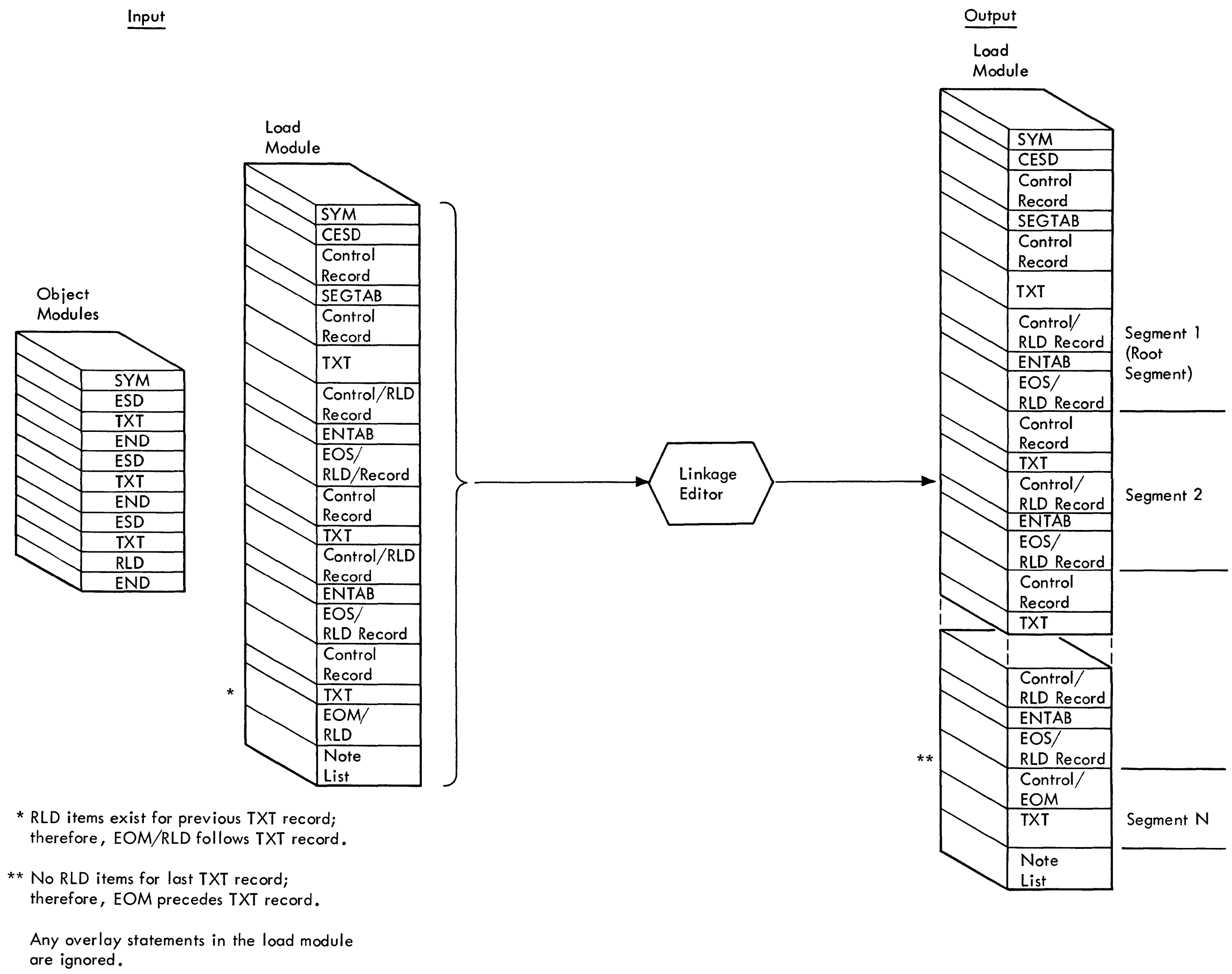
MODULE ATTRIBUTES

When the linkage editor generates a load module in a library (partitioned data set) it places an entry for the module in the PDS directory. This entry contains "attributes" describing the structure, content, and logical format of the load module. The control program uses these attributes to determine how a module is to be loaded, what it contains, if it is executable, whether it is executable more than once without reloading, and if it can be executed by concurrent tasks.

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

- Reenterable. A reenterable module can be executed by more than one task at a time and cannot be modified by itself or by any other module during execution; i.e., a task may begin executing a reenterable module before a previous task has finished executing it.

[3] If there are no RLD items for the last text record, the control record that precedes the text contains the EOS or EOM indication. If there are RLD items, the EOS or EOM follows the text record. (See Figure 3.)



* RLD items exist for previous TXT record; therefore, EOM/RLD follows TXT record.

** No RLD items for last TXT record; therefore, EOM precedes TXT record.

Any overlay statements in the load module are ignored.

Figure 3. Linkage Editor Processing - Using Overlay and Test Options

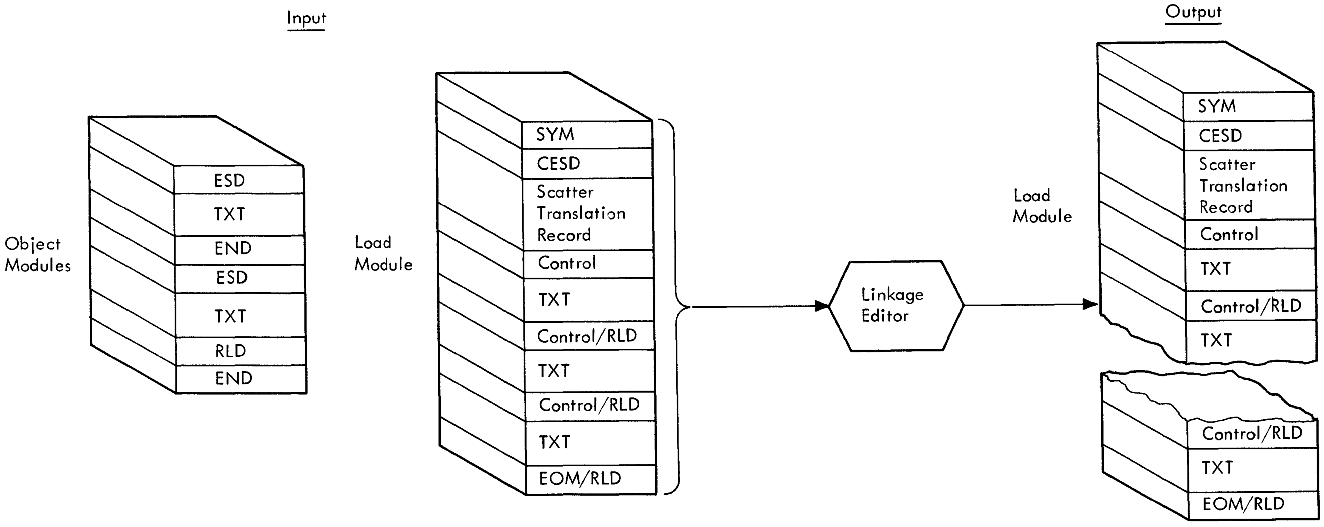


Figure 4. Linkage Editor Processing - Using Scatter Load and Test Options

- Refreshable. A refreshable module cannot be modified by itself or by any other module during execution; i.e., a refreshable module can be replaced by a new copy during execution by a recovery management routine without changing either the sequence or results of processing. (For details on recovery management, refer to the publication IBM System/360 Operating System, Concepts and Facilities, Form C28-6535.

- Serially reusable. A serially reusable module will be executed by only one task at a time, and it will either initialize itself and/or it will restore any instructions or any data in the module that it alters during its execution.

- Overlay format. A load module structured for overlay includes a segment table (SEGTAB) to enable the overlay supervisor to load the proper segments, and at least one ENTAB to assist in passing control from one segment to another. If a load module has the overlay format attribute, the reenterable, reusable, and scatter attributes cannot be present.

- Test. If this module is an assembler language program and testing by the test translator is desired, this attribute can be specified. Test will cause SYM records to be written. If the TEST attribute is specified, the module cannot be reenterable or serially reusable.

- Only loadable. This attribute indicates that the control program may load this module only via the LOAD macro instruction.

- Scatter format. A load module in scatter format is suitable for block or scatter loading. The scatter-translation table and the relocation dictionary maintain logical linkage between scattered control sections when program fetch loads them into main storage.

- *Block format. If neither the overlay nor scatter attributes are specified, it is implied that the module can only be block loaded. The control program will load the module only if enough contiguous main storage space is available for the entire module.

- *Executable. This attribute indicates that linkage editor did not find any errors that would prevent successful execution. If this attribute is not present the control program will not load the module.

- *Module contains one text record and no relocation dictionary records. This attribute indicates that the control program does not have to allocate main storage for relocation dictionary items when loading the module. It also indicates that the first text record is the last one; there is no control record following it. The entire module can be read by program fetch in a single read operation.

- Downward compatible. Indicates that the module can be processed by either the level E or F linkage editor. The downward compatible option is assumed by the level E linkage editor. Modules processed by the level F linkage editor that are not marked "downward compatible" cannot be processed by the level E linkage editor.

- *Linkage editor assigned origin of first text record is zero. If this attribute is present, the first byte of instruction or data in the first text record is assigned to location zero.

- *Entry point assigned by linkage editor is zero. Indicates that the entry point is at the first byte of the module.

- *No relocation dictionary items present. Indicates to the control program that no allocation of main storage is necessary to receive relocation dictionary items when program fetch loads them into main storage.

- Not editable. Indicates that the load module cannot be accepted by the linkage editor for subsequent processing. (For example, the programmer may drop the CESD from an output load module in order to conserve space on the library; such a load module cannot be reprocessed by linkage editor.)

- Symbol statements present. If a module produced by the assembler language translator is to be tested by the test translator, it may contain a testing symbol dictionary. In a load module, this dictionary contains the information from the symbol statement images that were input to linkage editor.

INPUT/OUTPUT FLOW

Four data sets must be specified for linkage editor processing; their ddnames and functions are:

- SYSLIN. This is the "primary input data set," containing object modules and control statements. All input from

SYSLIN must be in 80-column card image format.[1] The SYSLIN source may be a card reader, magnetic tape, a direct access device, or a concatenation of data sets from different types of input devices.[2]

- SYSPRINT. This is the "diagnostic output data set." Diagnostic messages, the module map, and the cross-reference table are written on SYSPRINT. (In the Sequential Scheduling System, the SYSPRINT device is normally a printer or magnetic tape.)

- SYSUT1. This is the "intermediate data set." Linkage editor uses this data set for temporary storage of text and RLD items being processed. SYSUT1 must be on a direct access volume.

  Note: SYSUT1 is only opened when two-pass processing is in effect.

- SYSLMOD. this is the "output module data set." It is a partitioned data set on a direct access volume. SYSLMOD contains load modules; their attributes are described in the user's portion of the directory entry for the member.

An additional data set, SYSLIB, is used by linkage editor if there are any automatic library calls to be processed. SYSLIB can be defined only as a partitioned data set. The members of SYSLIB can be either load modules or object modules (but object and load modules cannot be contained in the same PDS). When SYSLIB is opened, the linkage editor determines whether the PDS contains object or load modules by checking the format in the data control block (DCB). If the PDS contains object modules, the record format (RECFM) field of the DCB indicates "fixed (F) format;" if it contains load modules, the DCB indicates "unknown (U) format." (Load module records are of variable length.) If SYSLIB contains object modules, the linkage editor ignores the user's portions of the PDS directory entries for the object modules.

Other data sets may be read by linkage editor when it processes INCLUDE or LIBRARY statements specifying ddnames. Data sets read into main storage with INCLUDE statements may be either sequential or partitioned. SYSLIB and data sets specified in LIBRARY statements for use by automatic library call must be partitioned.

The attributes for the "execute linkage editor" job step are the attributes specified on the EXEC statement. These attributes may be modified if a load module having different attributes is processed.

Figure 5 shows the input/output flow. During the initial processing, SYSLIN, SYSPRINT, SYSUT1, and SYSLMOD are opened. During input processing, the primary input is read from SYSLIN. If an INCLUDE statement is read in the primary input, the data set whose ddname is specified on the statement is opened, and is processed. At the end of all SYSLIN input, SYSLIB and any other data sets whose ddnames are specified on LIBRARY statements are processed through automatic library calls.

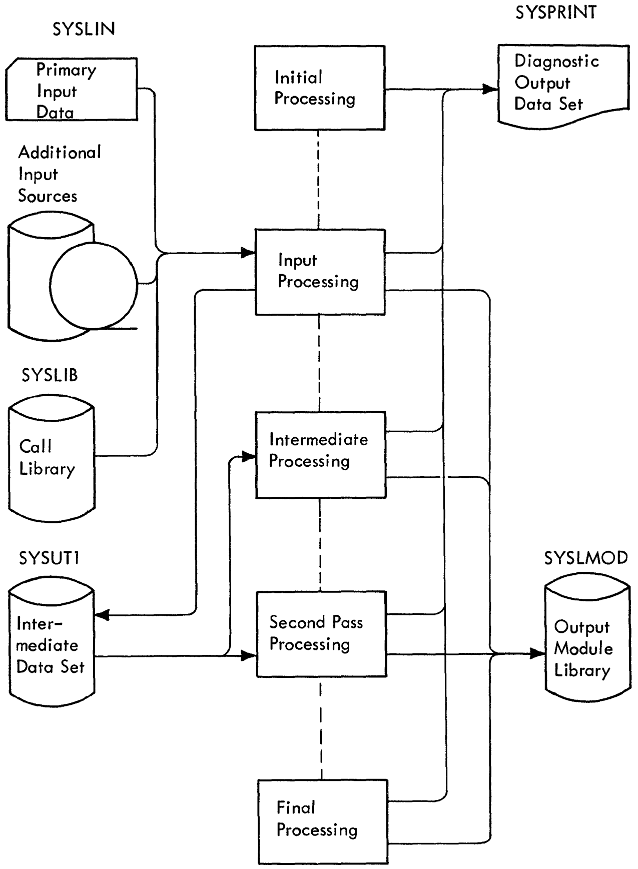


Figure 5. Input/Output Flow

If the TEST option has been selected, SYM records are written during input processing; text and RLD items are written sequentially on SYSUT1, except during single pass processing. The location of each text record on SYSUT1 is entered in a text note list. The location of each RLD record on SYSUT1 is entered in an RLD note list. If either note list overflows, it is written out on SYSUT1; either note list may overflow three times.

---

[1] The card images may be blocked.

[2] A concatenation of data sets cannot contain both object and load modules.

In intermediate processing, the CESD is written on SYSLMOD (unless the not editable attribute is indicated). If a scatter table, translation table, or SEGTAB is required, it is also written on SYSLMOD. The note list for the text and RLD items on SYSUT1 are read into main storage. If a module map was required, the CESD is used in producing the map. If a cross-reference table was requested and all RLDs are in storage, the table is produced during intermediate processing.

During second pass processing, text and RLD records are read into main storage from SYSUT1 in the order of assigned addresses within each segment (using the note lists to find the records) and are written out on SYSLMOD.

In final processing, the member name and any alias names are entered into the PDS directory entry of the output load module, via the STOW macro instruction. If any coded diagnostic messages were written on SYSPRINT during linkage editor processing, a diagnostic message directory containing error message text is written out on SYSPRINT. If a cross-reference table was requested and was not produced during intermediate processing, SYSLMOD is opened for input, RLDs are read, and the cross-reference table is produced. At the end of final processing, SYSLMOD is closed (if it was opened for input). All other data sets are then closed and control is returned to the calling program, unless the SYSLIN input during input processing was terminated by a NAME statement. If a NAME statement terminated the primary input, and it is not followed by end-of-file, control is returned to initial processing and SYSLMOD is opened for output, if it had been closed during final processing.

When a NAME statement is used to produce multiple load modules in a single execution of linkage editor, SYSLIN, SYSPRINT, and SYSUT1 remain open for the entire execution. (A pointer in the DCB for SYSUT1 is repositioned to the beginning of extent of SYSUT1 after each load module is produced.) If neither a module map nor a cross-reference table is requested, or if a cross-reference table is requested and all RLDs are in core, SYSLMOD remains open for output for the entire linkage editor execution.

This section contains an introduction to the logic of the linkage editor, which emphasizes the flow of primary data and control information through tables and buffers, and detailed functional descriptions of its phases.

LOGIC OF THE LINKAGE EDITOR

The linkage editor can be functionally divided into five phases:

- Initialization
- Input processing
- Intermediate processing, including address assignment and intermediate output
- Second pass processing
- Final processing

Operation diagrams (see Figures 6-10, 16-19, 22, and 26) at the end of this section illustrate the functional operation of the linkage editor. The shaded areas of the diagrams correspond to operations described in the text.

Initialization

When the linkage editor receives control from the job scheduler or a calling program, it performs initialization functions in preparation for all subsequent processing. (See Diagram A1). The operations included in initial processing (area A) are:

- Initialize DCBs and open data sets to be used during linkage editor processing.
- Allocate storage for all tables, buffers, and work areas to be used by linkage editor processing.
- Build the all purpose table (APT) and enter addresses and descriptions of all other tables and buffers into it.
- Analyze the attributes and options passed by the calling program (specified by the programmer) and save them in the all purpose table.

When all initialization functions are completed, the linkage editor is ready to accept input.

Input Processing

All linkage editor input is processed initially during the first pass. (See Diagram A1.) Object modules from SYSLIN (primary input data set) are read into the SYSLIN buffer (area B). Object modules from SYSLIB or a specified user's library (secondary input data sets) are read into the object module buffer (area C). Text records in load modules from SYSLIB or a user's library are read into the input text buffer (area F); all other load module records are read into the first pass RLD buffer (area D). The various records which constitute these modules are processed as follows.

Control Statements: These records, which may precede or follow object modules, contain information which is later used in symbol resolution and which specifies libraries containing secondary input. Depending on the type of control statement, entries are made in either the all purpose table (APT) or the composite external symbol dictionary (CESD).

ESD Records: These records from object modules, and CESD records from load modules, describe symbols that have been defined for external use. Entries for the symbols are made in the CESD (area E). Entries are made in the renumbering table to allow the translation of the input ESD indentifiers (IDs) into new CESD IDs. Entries are made in the delink table for symbols that are to be deleted or replaced.

TXT Records: These records, containing the instructions and data of the program, are moved from the SYSLIN buffer and object module buffer to the input text buffer (text records from load modules are read directly into the input text buffer) (area F). They are arranged in the proper sequence and recorded in the text I/O table and the text note list. When the input text buffer is filled, its contents are written onto SYSUT1; if it does not become filled, text records are retained in the buffer, and "single-pass" processing is in effect. Text note list entries contain the location of text records (SYSUT1 address or buffer address) and other descriptive information. Text I/O table entries contain information identifying text records by ESD ID.

RLD Records: These records, to be used later in relocating address constants, are moved from the SYSLIN buffer and object module buffer to the RLD buffer (area G). The relocation and position pointers (R and P pointers) are updated, using control information from the renumbering table and the delink table. RLD items are examined and marked for future processing. If V-type (branch-type) address constants are found in overlay programs, entries are made in the calls list for use during intermediate processing. When the RLD buffer is full, RLD records are written onto SYSUT1, and control information identifying RLD records by size (byte count), P pointer, and location on SYSUT1 is entered into the RLD note list. If the RLD buffer does not become filled, RLD records are retained in the buffer and "single-pass" processing is in effect.

SYM Records: These records, which are not involved in linkage editor processing, are gathered in the RLD buffer and are written directly onto SYSLMOD if the TEST option has been specified. If TEST has not been specified, SYM records are ignored.

When all input records have been processed (all external symbols have been entered into the CESD) control is passed to intermediate processing.

Intermediate Processing

The operations included in intermediate processing (see Diagram A2) have two primary objectives: to assign relative storage addresses to symbols in the CESD, and to write some of the records to be included in the output load module onto the SYSLMOD data set. MAP and XREF options may also be produced during intermediate processing.

Address Assignment: Entries which require no further processing are deleted from the CESD; all other CESD symbols are assigned temporary linked addresses. Relocation constants are determined for all control sections, and the relocation constant table (RCT) is built (area A).

For all programs in overlay, additional processing is required. The calls list is used to determine ENTAB entries to be placed in the CESD, and the downward calls list is built (area F). The segment length table (SEGLGTH) is built (area B), and segment relocation constants are computed. Temporary linked addresses in the CESD and entries in the relocation constant table are adjusted for overlay by adding to them the segment relocation constants (area B).

Temporary linked addresses and relocation constants are combined to determine final linked addresses for symbols, and the results are placed in the CESD. The alias table is built from alias symbols in the CESD. At this point CESD processing is complete.

MAP/XREF Processing: If the MAP option has been specified, a module map, containing sorted CESD items, is built and written on SYSPRINT. If the XREF option has been specified and all RLDs are in storage, a cross-reference table is built from RLDs (in the RLD buffer) and written on SYSPRINT. If all RLDs are not in storage, the cross-reference table is built during final processing.

Intermediate Output: The principal function of this section of intermediate processing is to write the CESD onto the output load module data set (SYSLMOD). The half ESD (HESD), containing control information from CESD entries, is built (area C) and held in main storage for use during second pass processing. The text I/O table (area E) is scanned to determine the ID of the last control section containing text in the program (or in each segment of an overlay program); this information is placed in the high ID table (HIID) (area E), and noted in the HESD for use during second pass processing.

For a program in overlay, the segment table (SEGTAB), which defines the relationships among segments, is built and written (with a control record) onto SYSLMOD (area D).

For a program that is to be scatter loaded, a scatter table and a translation table are built from information in the CESD, and scatter/translation records are written onto SYSLMOD (area G).

Module IEWLMOUT is the Intermediate Output Processor.

Second Pass Processing

The objectives of second pass processing (see Diagram A3) are relocating address constants in the text and writing onto the SYSLMOD data set the remaining records that constitute the output load module.

Text records are read from SYSUT1 (intermediate data set) into the second pass text buffer (area A), using the text I/O table and the text note list to locate the records on SYSUT1. The text I/O table is also used to determine the order in which text records are to be processed. RLD records associated with the text being processed are read into the second pass RLD input buffer, using the RLD notelist to locate the required records (area B).

Single-Pass Processing: If the linkage editor did not write text or RLD records onto SYSUT1, single-pass processing is in effect for these records. The records are accessed directly in the input text buffer and the RLD buffer, which are physically the same storage areas as the second pass text buffer and the second pass RLD input buffer. If text records or RLD records were written onto SYSUT1, they are read back into the same locations.

Relocation: Address constants described by RLD items are moved from the second pass text buffer to a work area, where relocation is performed (area C). The manner in which each address constant is relocated depends on whether it is a V-type (branch type) or an A-type (non-branch type) address constant, or a pseudo register (type 1 or type 2).

A V-type address constant can refer to a named location in some other control section (branch type address constant). The value field of such a V-type address constant always contains a zero because the address was not known at compilation time. During second pass processing, the linkage editor address (absolute relocation factor) that was assigned to the symbol and saved in the HESD is inserted into the value field. This is called *absolute relocation*. If the V-type address constant is in an overlay program, the address of an ENTAB entry for the symbol and the segment number of the current text is inserted in the value field. (ENTABs are created in the second pass RLD buffer from information in the HESD and the *entry list*, which contains an entry for each V-type address constant in the path of a referred-to symbol (area E).)

The value field of an A-type address constant that refers to a named location in the *same* input module (non-branch type address constant) contains an address assigned by the language translator. During second pass processing, this address is modified by adding or subtracting the relative relocation factor that was determined for the symbol referred to by the address constant. Relative relocation factors are saved in the relocation constant table. This process is called *relative relocation*.

When each address constant is relocated, it is placed back in the text, and the address field of the associated RLD item is updated (area D). The RLD item is then moved to the *second pass RLD output buffer*. When all address constants in the text buffer are relocated, the text is written onto SYSLMOD, followed by the associated RLD items (area F). A control record pertaining to the next text record is written onto SYSLMOD following the RLD records. If the output load module is structured for overlays a *TTR list*, containing the address of the first control record of each segment (for the first segment the list contains the address of the first *text* record) is also created and retained in main storage.

Second pass processing continues until all segments in the output module are processed. The last control record contains end of module indicators. Control is then passed to final processing.

Final Processing

The objectives of final processing (see Diagram A4) include writing remaining output to SYSLMOD, producing certain optional output, and "cleanup" functions.

The partitioned data set directory for SYSLMOD is completed, including modifications for ALIAS symbols (found in the *ALIAS table*), and a STOW macro is issued (area B). The TTR list, containing the address of the first text record in each segment, is written onto SYSLMOD for overlay programs (area A).

The *error logging map*, produced as errors are encountered throughout linkage editor processing, is scanned and an *error diagnostic directory* is built and written on SYSPRINT, (area C). Main storage allocated to linkage editor is released.

If the XREF option is specified, and was not processed during intermediate processing, RLD records are read from SYSLMOD, and a *cross-reference table* is built and written on SYSPRINT, (area D).

At the completion of linkage editor processing, control is returned to the calling program.

INITIALIZATION (IEWLMINT)

When the linkage editor receives control from the job scheduler, or from another program via a CALL (after execution of LOAD, LINK, XCTL, or ATTACH macro instruction), control information may be passed to it.[1] This information includes the attributes and options that control linkage editor processing. When control is passed to the linkage editor from the job scheduler, the passed control information is the information contained in the operand field of the EXEC statement. The control information is interpreted, checked for validity, and saved for later use in linkage editor processing.

[1] The method of passing information to the linkage editor is described in the System Reference Library publication *IBM System/360 Operating System: Linkage Editor*.

A program that passes control to the linkage editor may provide a substitute list of ddnames to be used in place of the standard names, and a name that is to be assigned to the output load module in the PDS directory.

Initialization functions performed by the linkage editor include:

- Building an all purpose table, which contains descriptions of other tables used by the linkage editor, and contains decision indicators that control linkage editor operation. The APT remains in main storage throughout the linkage editing process and is the major communication area among internal functions.

- Opening all data sets used by the linkage editor, except SYSLIB and SYSUT1, after the standard ddnames (or passed ddnames) have been entered into the data control blocks of the data sets. (The SYSLIB DCB is used for automatic library calls or INCLUDE statements; it is opened during input processing only if there are any automatic calls or INCLUDE statements specifying it. The SYSUT1 DCB is opened only when needed.)

- Setting an "unlike attributes" indicator in the SYSLIN DCB. This indicates to the open routine that SYSLIN may be a concatenation of data sets stored on different devices.

- Scanning and analyzing the control information that was previously passed in a list to linkage editor. The processing options requested by the user and the attributes to be assigned to the output load module are compared against an option table and noted in the all purpose table. When mutually exclusive attributes are specified for a load module, the linkage editor ignores the incompatible attribute (refer to Table 1). If the SIZE option is specified, the associated value is placed in the all purpose table. If the SIZE option is not specified, the default values chosen at system generation time are used.

- Requesting main storage space for internal tables, buffers, and work areas. The allocation processor issues a request for a minimum requirement of main storage space. The minimum value depends on whether or not the module being processed is structured for overlay; it includes an amount to be used by data management functions. If sufficient main storage space is available, the supervisor returns control to the allocation processor and the space exceeding the minimum requirement is divided among the tables and buffers. If sufficient main storage space is not available, the control program will not return control to the linkage editor; instead, a system ABEND will occur.

Table 1. Incompatible Module Attributes

	OVLY	TEST	XREF	MAP	LET	RENT	REUS	SCTR	NE	XCAL	LIST	NCAL	OL	DC	SIZE
TEST															
XREF															
MAP			X												
LET															
RENT	X	X													
REUS	X	X				X									
SCTR	X														
NE		X	X	X											
XCAL					X										
LIST															
NCAL															
OL															
DC															
SIZE															
REFR	X	X													

Note: An X indicates incompatible attributes; the attribute that appears lower in the list is ignored. 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 attributes. NE is ignored.

Main Storage Allocation

To obtain the required main storage space, the allocation processor:

1. Issues the GETMAIN macro instruction, and if sufficient main storage space is available, assigns storage for the maximum buffer lengths to each of the object module buffers, SYSLIN buffers, and SYSPRINT buffers. If sufficient space for maximum buffer lengths is not available, intermediate buffer lengths are assigned. If sufficient space for intermediate lengths is not available, the minimum buffer lengths are assigned.

2. Assigns main storage to the RLD buffer and the text buffer. The text buffer area is referred to as the input text buffer during input and intermediate processing, and as the second pass

text buffer during second pass processing. The text buffer will be assigned the minimum length (6K bytes) unless additional space was requested via the SIZE parameter, in which case the text buffer will be expanded, as specified, up to a maximum of 100K bytes.

<u>Note:</u> All space allocated for buffers is released only at the completion of linkage editor processing.

3. Determines the excess of main storage space allocated by the supervisor.

4. Divides the total excess by the total <u>weight factor</u>. A weight factor is a ratio based on the individual main storage requirements of linkage editor tables that are not fixed in size. (Fixed tables have weight factors of zero.) The total weight factor depends on whether or not the module is structured for overlay.

5. Multiplies the quotient obtained in step 6 (rounded to the nearest lower integer) by the weight factor for each table and adds the result to the minimum requirement for the table. This is done for all tables and buffers.

6. Divides the total byte count for each table by the number of bytes per entry, and saves the result in the all purpose table.

7. Computes the addresses for the tables and places them in the all purpose table.

8. Releases excess main storage space, saving the last address used.

When the required main storage space has been allocated, tables are initialized to zero, and the linkage editor is ready to accept input.

<u>INPUT PROCESSING (IEWLMINP)</u>

The operations performed during input processing depend on the nature of the input; special processing is required for each input record type. Each input record is read, using one of two read blocks. The first read control block contains the address of the SYSLIN buffer, the address of the SYSLIN DCB, and the block size and logical record length. The second read control block contains the address of the buffer for library records (object module buffer or load module buffer), the address of the library DCB, and the block size and logical record length. A pointer is used to indicate which read control block is to be used for the input record. Initially, the pointer is set to the SYSLIN read control block.

The type of input processing required is determined by the following conditions:

- For all object module records whose first column character is a blank, control statement scanning is required, provided that the record is not encountered "in module". (Control statements encountered within a module cause an error indication.)

- Either object module processing or load module processing is required, depending on the type of input module. Only object modules are read from SYSLIN. Input modules from libraries are identified by record format. F format indicates object modules; U format indicates load modules.

- At end-of-input (from SYSLIN or SYSLIB), include processing is required if more modules must be included before rerunning normal processing.

- At end-of-input from SYSLIN, automatic library call processing is required if the NCAL option (no automatic library calls) was not selected. If the NCAL option was selected, input processing is complete.

- If a NAME statement, which may indicate a multiple execution of linkage editor, is detected during control statement scanning, processing proceeds as if an end-of-input has occurred on SYSLIN (automatic library call processing is performed). The next record is read to determine if end-of-input has occurred; if not, input processing will be repeated at the end of final processing.

- If an end-of-input occurs on SYSLIN, but no valid input was received, linkage editor processing is terminated.

<u>Reading Blocked Input</u>

The linkage editor can accept blocked card image input from the SYSLIN data set and blocked object module records from the SYSLIB data set (or from a user's library). Maximum block sizes allowed by the linkage editor are shown in Table 2. Generally, the record format, block size, and logical record length are established either when the data set is created, or when they are specified on the DD statement for the data set in an execution of the linkage editor. If the BLKSIZE field is not specified, the linkage editor assumes a block size of 80.

The logical record length (LRECL) is fixed at 80.

Table 2. Block size Determination

Maximum Block size	Main Storage Available
5	44K (+xK) - 52K (+xK)
10	52K (+xK) - 88K (+xK)
40	88K (+xK) - 9999K
xK is the (optional) additional storage allocated to the load module buffer (i.e., storage in excess of 3K).	

If the block size specified on primary input exceeds the allowable maximum (see Table 2), or is not a multiple of the logical record length, an error message (IEW0594) is issued and linkage editor processing is terminated; if the invalid block size is specified on input from a library, the data set is ignored, but processing is not terminated. The block size specified by the user is used as the read count; if a short block is read, the linkage editor determines (via an exit at SYNAD) if the length of the short block is valid (a multiple of the logical record length), and the number of the logical records it contains.

If SYSLIN is a concatenation of data sets, the input processor reexamines the block size fields whenever a data set boundary is crossed to determine if their values have changed.

Blocked Output on SYSPRINT

The logical record length for output to SYSPRINT is fixed at 121. If the BLKSIZE is not specified by the user, it is set equal to the logical record length. If the specified block size exceeds the allowable maximum (see Table 2), or is not an integral multiple of the logical record length, linkage editor processing is terminated and a condition code of 16 is returned.

Control Statements

When an input record is found to be a control statement (blank in column 1), it is scanned to detect format errors and continuation of comments or operands. A vector table is scanned to determine the appropriate processor; separate processing is required for each type of control statement (INCLUDE, REPLACE, LIBRARY, CHANGE, INSERT, OVERLAY, ENTRY, ALIAS, NAME, or SETSSI). Diagram B1 illustrates general processing of each control statement type.

The general format for linkage editor control statements is shown in Figure 11. The control statement scanner interprets symbols enclosed in parentheses as "level 1" symbols; symbols not enclosed within parentheses are "level 0." ENTRY, ALIAS, INSERT, and SETSSI control statement operands contain only level 0 symbols. CHANGE statement operands always contain both a level 0 symbol and a level 1 symbol.

The operands of REPLACE, INCLUDE, OVERLAY, and NAME control statements contain level 0 symbols, or both level 0 and level 1 symbols. LIBRARY statement operands may contain level 1, or both level 0 and level 1 symbols. The operation to be performed depends on the operand format.

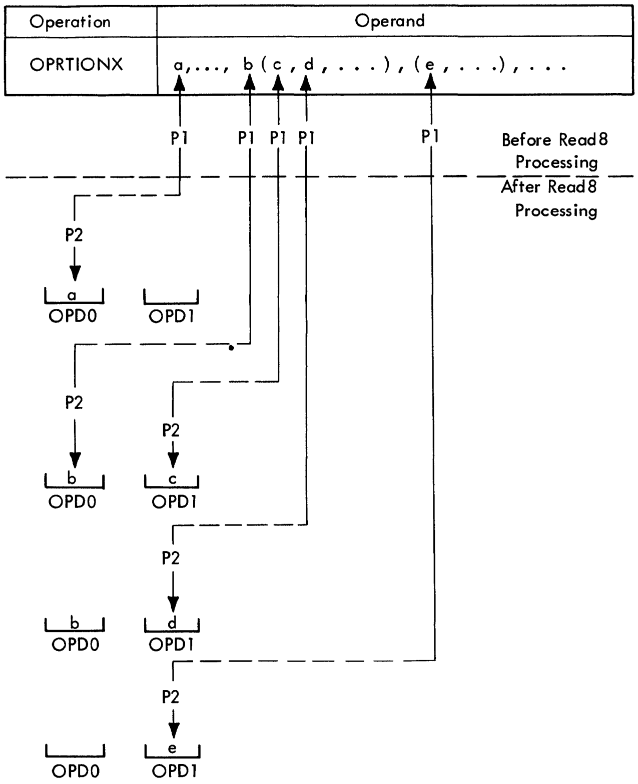


Figure 11. Control Statement Scanner Operation

The control statement scanner searches a vector table for the operation symbol to determine the associated control statement processor. It then analyzes the operands using two work areas, "OPD1" and "OPD0," and two pointers, "P1" and "P2." OPD1 is used for level 1 operand symbols; OPD0 is for level 0 operand symbols. P1 points to the operand symbol being analyzed; P2 points to either OPD0 or OPD1, depending on the level of the operand symbol referred to by P1.

An operand symbol referred to by P1 is placed by the READ8 routine into the work area referred to by P2. Parentheses and commas control the switching of pointer P2 between the work areas. For example, when a left parenthesis is encountered, P2 moves to OPD1 because a level 1 operand symbol will follow. When a comma, blank, or right parenthesis is detected, the PROCENTY routine passes control to the control statement processor that was previously found during the search of the vector table.

Control Statement Processors

When the operand symbols have been read into work areas OPD0 and OPD1, control is passed to the control statement processor at the saved entry point. Scanning of the control statement resumes when the control statement processor returns control. The individual control statement processors are described in the following paragraphs.

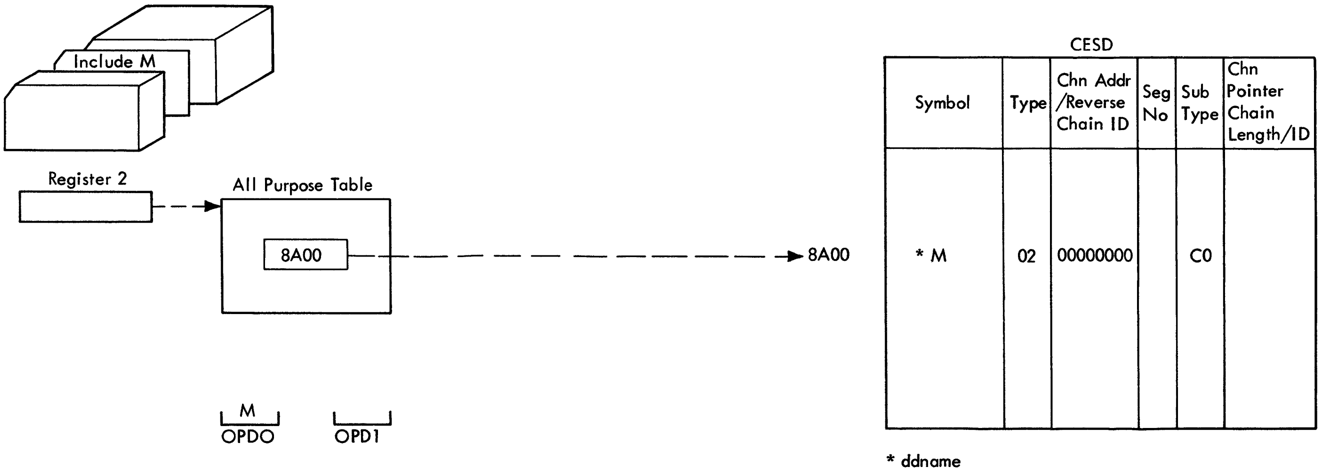
INCLUDE STATEMENT PROCESSOR: The include statement processor builds a chain in the CESD of items to be included. Each item in the chain contains the address of the next item in the chain (in the chain/address field - bytes 9, 10, and 11). The last item in the chain contains zeros in this field.

Chained include items have two kinds of subtypes: "include with pointer" and "include without pointer." In Figure 12, the statement INCLUDE M defines M as a sequential data set. The include statement processor creates an entry for the ddname M in the CESD with the subtype "include without pointer."

In the statement INCLUDE LIBX(A), A is defined as a member of a PDS. The include statement processor creates an entry for A in the CESD with the subtype "include with pointer." The pointer is in the chain pointer/chain ID field (bytes 14 and 15); it contains the CESD line number of the ddname LIBX. A single ddname, such as LIBX, may be referred to by several pointers.

In Figure 13, the statement INCLUDE TEMP(A,B,C) indicates that A, B, and C are members to be included from library TEMP. Member B contains the nested statement INCLUDE LIBX(U,V,W); this is the last statement processed in member B. The CESD is shown at the time when the control statement scanner has read operand V, but not W. The include statement processor has created a CESD line for operand V in the LIBX include chain. C is currently the last item in the TEMP include chain. When the control statement scanner reads operand W, the include statement processor enters a CESD line for W between V and C; this process is distinct from the one that actually searches the members U, V, and C on the library. (Refer to the paragraph "Include Processor.") At the time chosen for this example, the data set member B is being read; data set member A has been read and therefore is no longer in the CESD as a member name, but data set members U, V, and C have not yet been read.

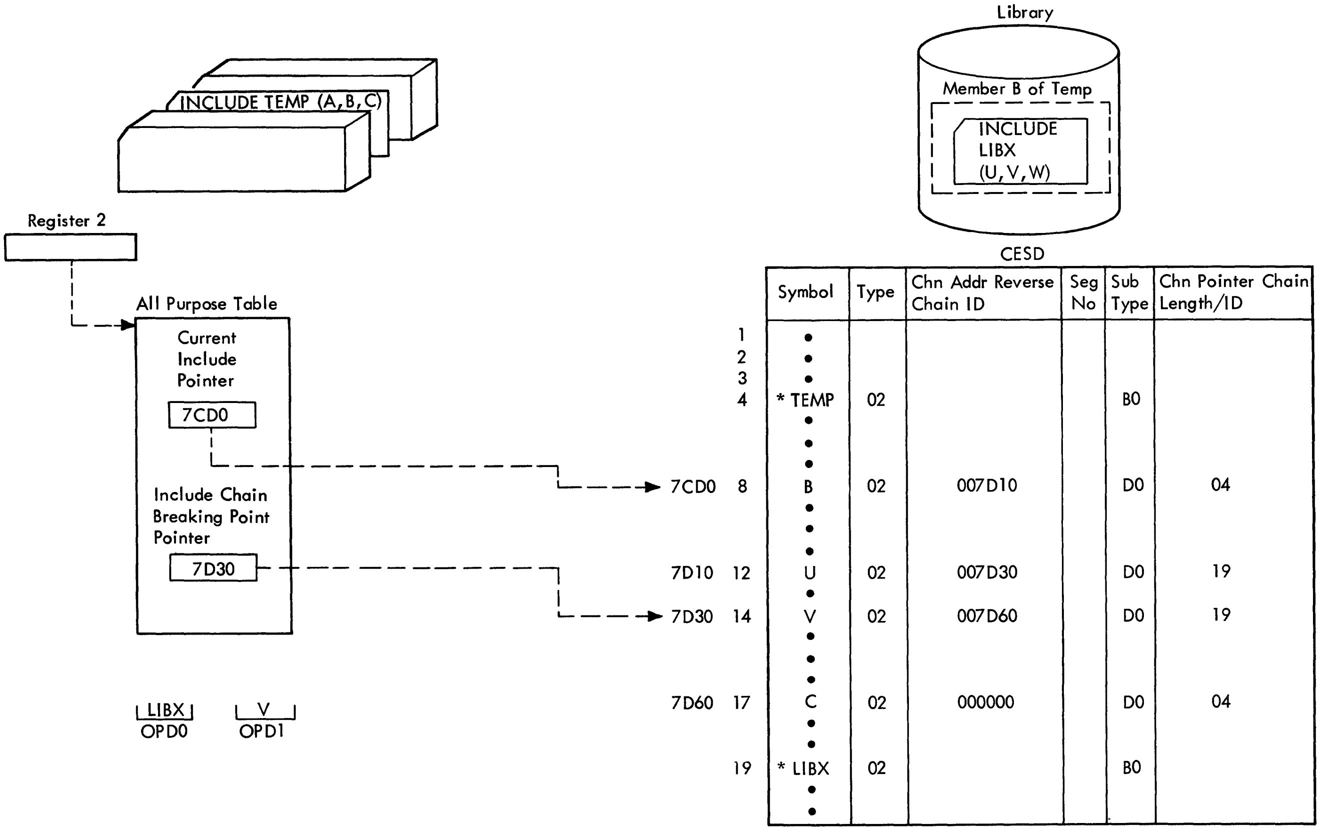
The chained CESD entries created by the include statement processor are later processed by the include processor (Chart JR).



Symbol	Type	Chn Addr /Reverse Chain ID	Seg No	Sub Type	Chn Pointer Chain Length/ID
* M	02	00000000		C0	

* ddname

Figure 12. Include Statement Processing for a Sequential Data set



		Symbol	Type	Chn Addr Reverse Chain ID	Seg No	Sub Type	Chn Pointer Chain Length/ID
	1	•					
	2	•					
	3	•					
	4	* TEMP	02			B0	
		•					
		•					
		•					
7CD0	8	B	02	007D10		D0	04
		•					
		•					
		•					
7D10	12	U	02	007D30		D0	19
		•					
7D30	14	V	02	007D60		D0	19
		•					
		•					
		•					
7D60	17	C	02	000000		D0	04
		•					
		•					
	19	* LIBX	02			B0	
		•					
		•					

* ddname

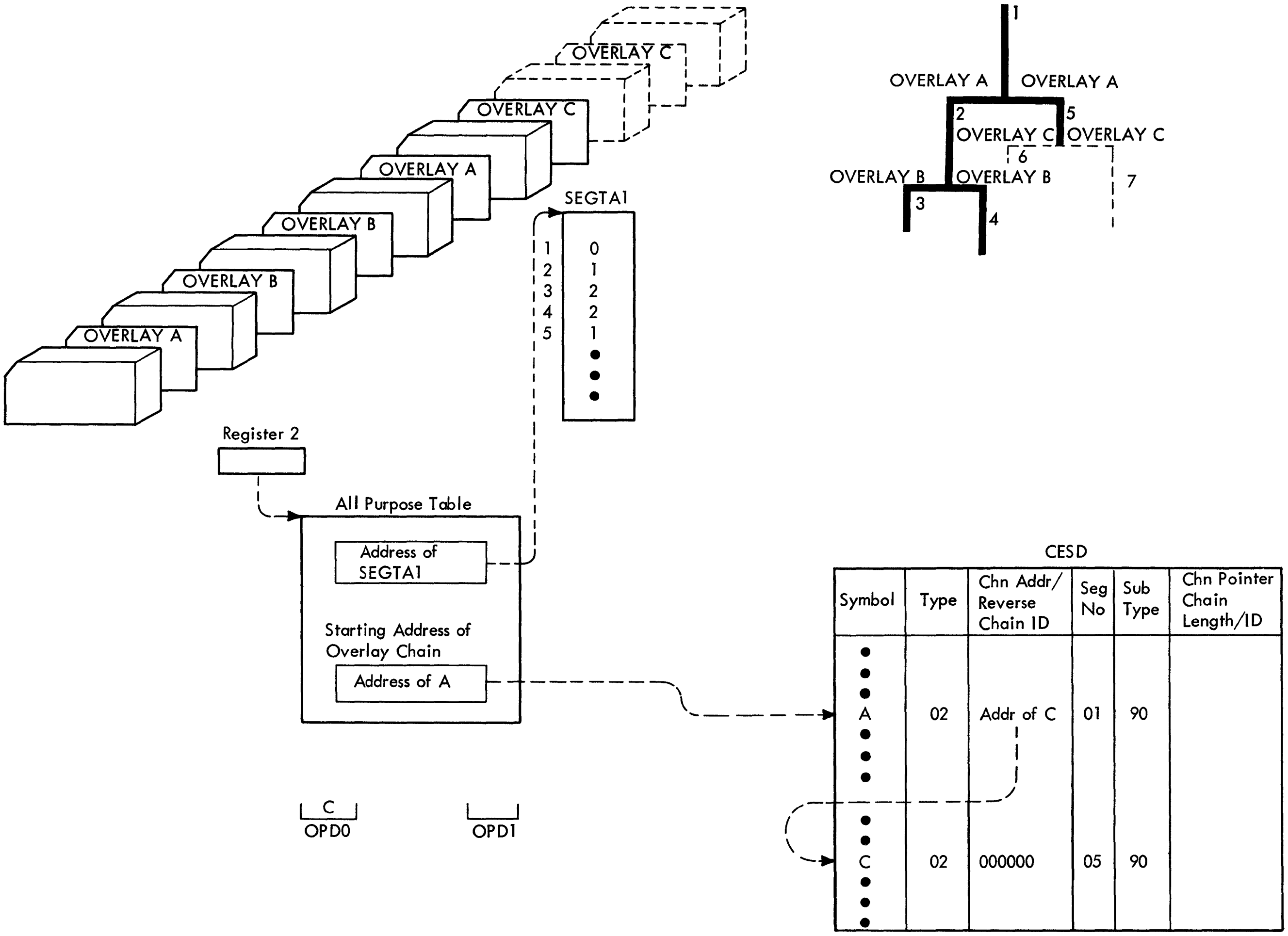
Figure 13. Include Statement Processing With Nested Members

OVERLAY STATEMENT PROCESSOR: The overlay statement processor maintains a record of the current segment number and updates it by one each time a new OVERLAY statement is encountered. The relationship of segments in an overlay tree structure is kept in SEGTA1 (see Figure 14). Entry n in SEGTA1 contains the number of the segment that precedes the nth segment of the overlay tree structure (the next higher segment in its path). The overlay statement processor creates a chain of overlay items in the CESD and updates SEGTA1. If the level 1 operand (REGION) is detected, the current region number is incremented by one, and a zero is entered as the previous segment number in SEGTA1.

If an OVERLAY statement is encountered that refers to a node point higher in the overlay tree structure, all symbols identifying node points higher in the path are removed from the chain; their CESD lines are marked "null." For example, in Figure 14, when the statement OVERLAY A is encountered after segment 4, the CESD entry for symbol B is marked null and is no longer in the chain. If an OVERLAY B statement was encountered at the end of segment 5, a new node point would be established for B, and symbol B would again be entered in the CESD.

INSERT STATEMENT PROCESSOR: The insert statement processor scans the CESD for the symbol indicated in the INSERT statement. If the symbol is found, the segment number field is changed to the number of the segment that contains the INSERT statement. If the symbol is not found in the CESD, a new ER-type CESD entry is created. In either case, the new CESD entry is marked "insert" in the subtype field, and the segment number of the INSERT statement is placed in the segment number field.

REPLACE AND CHANGE STATEMENT PROCESSORS: The replace and change statement processors build a chain of CESD entries. Each entry to be replaced, changed, or deleted is so marked in the subtype field. The ESD processor examines the replace/change chain before processing any ESD item. Since a REPLACE or CHANGE statement applies only to the module that immediately follows it in the input, the replace-change chain is removed from the CESD at the end of the module.



Symbol	Type	Chn Addr/ Reverse Chain ID	Seg No	Sub Type	Chn Pointer Chain Length/ID
● ● ●					
A	02	Addr of C	01	90	
● ● ● ●					
C	02	000000	05	90	
● ● ●					

Note: In this example, card OVERLAY C has just been read. Name B is no longer in the chain.

Figure 14. Overlay Statement Processing

When a REPLACE statement or a CHANGE statement operand contains two symbols, such as CHANGE A (B), A and B are entered in consecutive lines of the CESD. Only the first line of the pair (the line for A) contains the address (in the chain address field) of the next item in the replace/change chain.

NAME STATEMENT PROCESSOR: The name statement processor places an entry in the all purpose table containing the name under which the following input module is to be STOWed in the PDS directory. If the operand contains the level 1 symbol (R), a bit is set to indicate that the module is to be STOWed as a replacement for a module of the same name. Another bit is set to indicate that a NAME statement was encountered; the input processor tests this indicator and terminates input operations for this load module if it is set. If a NAME statement is received from any input source other than SYSLIN, the error routine is entered; NAME statements are accepted only if they are in the primary input.

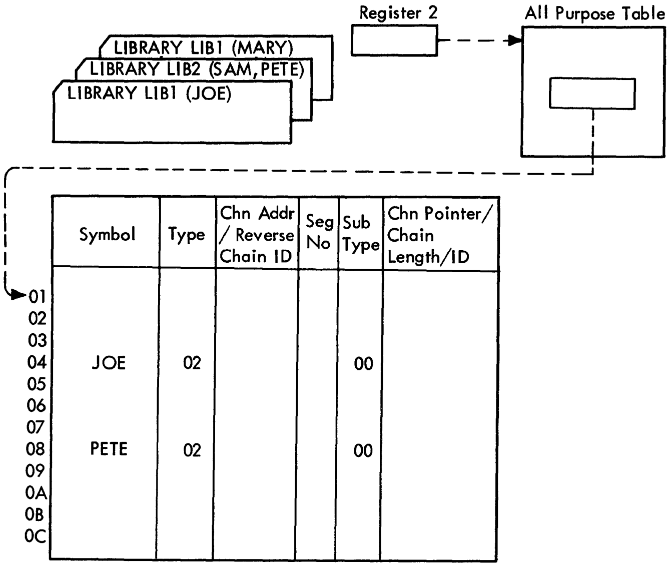
SETSSI STATEMENT PROCESSOR: The SETSSI statement processor converts the eight bytes of hexadecimal information specified on a SETSSI statement to a 4-byte field, and enters it into the APT. During final processing, this information is entered into the system status index, a 4-byte extension of the user data area in the PDS directory. The index contains information describing the status of members in the library and is used for maintenance purposes.

ENTRY STATEMENT PROCESSOR: The entry statement processor places the symbol specified in an ENTRY statement in the all purpose table. The symbol will override any symbol specified in an END statement as the entry point for the module.

ALIAS STATEMENT PROCESSOR: The alias statement processor creates chained CESD entries for a maximum of five alias names specified in ALIAS statements. During address assignment, these entries are used to build the alias table.

LIBRARY STATEMENT PROCESSOR: The library statement processor creates chained CESD entries for the operands specified in LIBRARY statements; a chain is created for each distinct library. Each chain begins with a library ddname and contains all member names specified for the library (see Figure 15).

A member name specified in a LIBRARY statement can result in two kinds of ER subtypes: "matched library member" or "unmatched library member." If a CESD entry is created for a member name specified in an input ER and also specified in a LIBRARY statement, it is called a "matched library member." However, if the member name was specified only in a LIBRARY statement, the entry subtype is "unmatched library member."



	Symbol	Type	Chn Addr / Reverse Chain ID	Seg No	Sub Type	Chn Pointer/ Chain Length/ID
01						
02						
03						
04	JOE	02			00	
05						
06						
07						
08	PETE	02			00	
09						
0A						
0B						
0C						

Diagram A

	Symbol	Type	Chn Addr/ Reverse Chain ID	Seg No	Sub Type	Chn Pointer Chain Length/ID
01						
02						
03						
04	JOE	02	0C		03	0A
05						
06	LIB2	02	00		B0	07
07	SAM	02	06		02	08
08	PETE	02	07		03	00
09						
0A	MARY	02	04		02	00
0B						
0C	LIB1	02	00		B0	04

Diagram B

	Symbol	Type	Chn Addr /Reverse Chain ID	Seg No	Sub Type	Chn Pointer Chain Length/ID
01						
02						
03						
04	JOE	00				
05						
06	LIB2	02	00		B0	07
07	SAM	02	06		02	08
08	PETE	02	07		03	00
09						
0A	MARY	02	0C		03	00
0B						
0C	LIB1	02	00		B0	0A

Diagram C

Notes:

- The CESD shown in diagram B results from the CESD shown in diagram A after reading in three library cards. A chain with direct and reverse pointers is created for LIB1 and also for LIB2.
- JOE and PETE were ERs (subtype 00) and became "matched library member" (subtype 03).
- SAM and MARY were not previously in the CESD. They are created as "unmatched library member" (subtype 02).
- The CESD shown in diagram C results from the CESD shown in diagram B after reading in an input module containing the ER MARY and the SD JOE. (Only the library chains are shown).
- JOE is removed from the chain in diagram C, and the chain pointers are modified.
- MARY becomes a "matched" subtype and will be called by the automatic library call processor (unless resolved by other input).
- SAM remains "unmatched" and will be ignored by the automatic library call processor (unless matched in other input).

Figure 15. Library Statement Processing

Object Module Processing

If input to be read by linkage editor consists of object modules (F record format indicates object modules from a library) the following operations are performed:

- Determine record type
- Set up general registers
- Special event processing

The record type is determined by examining columns 2 through 4 of each logical input record. For each record type (SYM, ESD, TXT, RLD, END), special processing is required.

The general registers are loaded with input record information to be used in the required processing, as described in Table 3.

Following is a description of special event processing:

- When end-of-input is detected, any data still contained in the input RLD buffer or the input text buffer is written out on SYSUT1, if necessary.

- If the TEST option is selected, the SYM records from the object module are gathered in the input RLD buffer. When the first TXT statement in a module is encountered (or if no text statement has been encountered when the END statement is detected), the contents of the input RLD buffer are written out on SYSLMOD.

- When ESD processing is completed, indicators in the all purpose table are examined to determine if:

  1. A control section (SD, PC, or common) was indicated on the ESD statement.

  2. The TEST option was specified.

  If both conditions are met, the ESD record is blocked with any other ESD records in the input RLD buffer.

- If a control statement continuation is expected and an object module record is read, an error condition occurs, and a coded diagnostic message is produced. Normal object module processing is then performed on the record.

- If, during object module processing, a statement is encountered which is not one of the five acceptable types (SYM, ESD, TXT, RLD, or END), an error condition occurs and a diagnostic message is produced. The input record is then ignored.

Table 3. General Register Information - Object Module processing

Input Record Type (See the Appendix For record formats)	General Register			
	3	4	5	6
SYM		SYM statement byte count		Address of SYM statement in buffer
ESD		Number of bytes of ESD information	ESDID of first ESD item on statement	Address of first byte of ESD in buffer
TXT	Assigned address of first byte of text	Number of bytes of text information	ESDID of CSECT to which text belongs	Address of first byte of text in buffer
RLD		Number of bytes of RLD information		Address of first byte of RLD in buffer
END	Absolute address of entry point on END statement	Length of CSECT for which no length was given in ESD item	ESDID of CSECT containing entry point	

Load Module Processing

Load modules included in the input to linkage editor are processed in the following manner:

- The input record type is determined by an identification field (byte 1 of the record), as shown in Table 4. Special processing is performed for each record type.

- The parameter registers are loaded with input record information to be used in the required processing, as described in Table 5.

- If the record is not identified as a TXT, CESD, Scatter/Translation, SYM, or CCW/RLD record, an error condition occurs, and a diagnostic message is printed out. The input record is otherwise ignored.

- If the TEST option was not specified on the EXEC statement, all SYM records are ignored.

- If an end-of-module indication is found in a CCW or RLD record, cleanup functions are performed.

- When a CCW record is detected, the following TXT record is immediately read into the input text buffer if it is not to be deleted.

- If the TEST option was specified on the EXEC statement and a SYM record is received, the record is written out as test translation data from the RLD input buffer.

The following text describes the special processing performed, during object and load module processing, for the ESD, TXT, RLD, and END records.

Table 4. Record Types

Record Type	Identifier
TXT	*
CESD	hex'20'
Scatter/Translation	hex'10'
SYM	hex'40'
CCW	hex'01'
CCW/RLD	hex'03'
RLD	hex'02'
If End of Module indication is on:	
CCW	hex'0D'
CCW/RLD	hex'0F'
RLD	hex'0E'
*Identified by preceding control record	

ESD Record Types

Every object module in the input to linkage editor must contain at least one ESD item. An ESD item is created by a language translator whenever it finds a symbol that is defined for external use. In the assembler language, for example, ESD items are created whenever an ENTRY, EXTRN, COM, START, or CSECT statement, or a V-type address constant is found. An ESD item is created to define the beginning of each control section, and to define a common area. Each ESD item has a type assigned to it that indicates its function. The ESD types are:

- Section Definition (SD). Defines the beginning of a named control section.

- Private Code (PC). Defines the beginning of an unnamed control section.

- Label Definition (LD). Defines a label (symbol) whose location is defined

Table 5. General Register Information - Load Module Processing

Load module Record Type	General Register			
	3	4	5	6
SYM		Zero		
CESD		Byte count of ESD items in record	ESDID of first CESD item in record	Address of first CESD item in buffer
CCW (TXT)	Assigned address of first byte of	Number of entries in ID-Length list	ESDID of CSECT to which text belongs	
RLD		Byte count of RLD items in record		Address of first RLD item in buffer

relative to the location of the control section in which it is contained. An LD type ESD item contains the ESD ID of the control section that contains the label.

- Common (CM). Defines a common area for which a main storage address is assigned during linkage editor processing. The area may be named or unnamed; an unnamed area is referred to as a "blank common" area.

- Pseudo Register (PR). Defines an area external to the output module, but referred to by it, for which main storage space is allocated at execution time. The linkage editor treats PR symbols as a block that is external to the program. The value assigned to each symbol is a displacement within this block.

- External Reference (ER). Refers to a symbol that is referred to but not defined within an input module.

CESD Record Types and Subtypes

A load module in the input to linkage editor contains at least one CESD record (240 bytes, maximum). The CESD record types are the same as for ESD records, with the following additions:

- Null type. This indicates that the item is to be ignored in any reprocessing of the module by linkage editor.

- Label Reference (LR). This defines a label (symbol) within a control section. An LR type CESD entry is numbered; it contains the ESD ID of the control section entry in the ID/length field. An LR may be referenced directly by an RLD item in the same module, whereas an LD may not. All LD items are changed to LR items during linkage editor processing (LDs are contained only in object modules, never in load modules).

- Private Code (PC) Marked Delete. This is a CESD item created only for ENTABs and SEGTABs. PC-delete entries are placed in the renumbering table, indicating that associated TXT and RLD information is to be deleted.

CESD items may also contain a "subtype." The subtypes are listed in the internal CESD format in the Appendix (Section 7).

ESD Processing

The main function of ESD processing is symbol resolution. Individual ESDs in the input to linkage editor are combined into a composite ESD, which contains all symbols in the input which were not changed, deleted, or replaced. A chained REPLACE/ CHANGE list (produced by the control card scanner) specifies which ESD items are to be changed, deleted, or replaced. A renumbering table (RNT) is also produced during ESD processing; it is used during TXT, RLD, and END processing to translate the ESD ID of the input ESD items to CESD IDs. Diagram B2 provides a general illustration of several types of ESD processing.

At the beginning of ESD processing, control information from the ESD record is saved: the ESD ID of the ESD record, the number of bytes of ESD information, and the type field of the first ESD item. The current segment number is placed in the ESD, unless it is a PR-type (PRs have an alignment value in the segment number field). If the automatic library call indicator is on, the segment number is set to 1 so that called modules will be placed in the root segment. The ESD item is then processed according to its type, in the following manner:

- If the ESD item is an ER, bytes 10, 11, and 12 are set to zero in the input buffer (either the object module buffer, the SYSLIN buffer, or the first pass RLD input buffer). Byte 10 must be cleared because automatic library call processing uses it to indicate if automatic library calls have been processed. Bytes 11 and 12 must be cleared because any nonzero data (including blanks) will be entered in the delink table if delinking is required for the symbol. If the input item is an ER item from an object module, the CESD subtype field is also reset to zero to indicate that there are no modifiers in the subtype field.

- If a REPLACE/CHANGE function has been requested for the input module, the REPLACE/CHANGE chain that was built in the CESD by the control statement scanner is examined and the appropriate modifications are made. For example, if the scanner received the statement CHANGE A (B), the CESD contains a line for A, marked as a change statement item in the subtype field; the next line contains the symbol B. The input ESD item symbol is changed from A to B during ESD processing.

- If the ESD item is a PC, the CESD is not searched because each PC entry is treated as a unique entry. The PC is placed in the next available CESD line and is processed in the same manner as an SD.

- If the ESD item is NULL, the renumbering routine is entered. (This routine is described in "Non-Resolution Processing.")
- If the ESD item is an LD, it is changed to an LR. The item is then processed as an LR. (There are some minor differences in processing LDs that have been changed to LRs; for this reason, an internal indicator is set when the type is changed to LR.)

After the ESD type is determined, the CESD is scanned for a matching symbol. If no match is found, non-resolution processing is performed. If the input ESD symbol matches a symbol in the CESD, resolution processing is performed. Resolution processing results in only one CESD entry for each unique input ESD symbol; multiple occurrences of the same input ESD symbol are listed in the renumbering table (RNT) with pointers to the single CESD entry.

NON-RESOLUTION PROCESSING: If no matching symbol is found in the CESD, the input ESD item is processed as described in the following paragraphs.

SD Items: If the input ESD item is an SD (see Diagram B2, Area A):

- The Freeline routine selects an empty line in the CESD. The line following the current line is chosen unless a previous CESD line is marked null. (Null lines are used whenever possible to save space.)
- If automatic library calls are being processed, an indicator is set in the type field of the selected CESD line. (If a module map was requested, this indicator is checked during module map processing. If the indicator is set, the control section is marked with an asterisk in the module map or cross reference table to indicate that it was obtained from a library during automatic library call processing.)
- A "write" indicator is set in the all-purpose table to note that SDs, PCs, or CMs were encountered in the input record. When ESD processing is completed, the write indicator is tested. If it is on and the TEST option was specified, ESD records containing SDs, PCs, or CMs are saved, blocked into 244-byte records (including four bytes of control information), and written out on SYSLMOD.
- In any input object module the CESD line number of the first SD entry whose length is zero is saved. END processing uses this CESD line to enter the length specified on the END card.
- The enter routine creates a CESD entry for the input ESD item; it moves the symbol, length, segment number, ID, and type into the selected CESD line.
- The renumber routine places the line number of the new CESD entry into the renumbering table to provide a means of translating the input IDs to the new CESD IDs. For example, if the input ESD item has a line number (ESDID) of 3 but the item is placed into the CESD at line 5, 5 is placed in the third line of the renumbering table. (For each input ESD line, except LD lines, there is a corresponding RNT line. The RNT contains information for the current module; it is set to zero at the end of each input module.)

ER Items: If the input ESD item is an ER, it is entered in the CESD and renumbered as described above; no special processing is required.

CM Items: If the input ESD item is CM (see Diagram B2, Area E), a "common" indicator is set and the item is treated as a delete item. If the address that was assigned to the CM item by the language translator is not zero, it is saved in the delink table for later use. (Two CM items with the same identifying symbol may have different assigned addresses; therefore, the assigned address in the input must be subtracted from all address constants that refer to the CM items so that they are returned to their displacement value before relocation.) The CM item is then renumbered and entered into the CESD.

LR (or LD) Items: If the input ESD item is an LR or LD (see Diagram B2, Area C):

- When processing an LR, the Label routine determines if the SD for the control section has been processed. If the SD has not been received, any LRs that refer to that SD are chained together in the CESD until the SD is received. (The SD might be marked replace; therefore, the LR cannot be processed until the SD is received.) When the SD is received all dependent LRs are processed. Each LR ID field is renumbered using the renumbering table so that it refers to the CESD ID of the SD.
- LDs are not renumbered because they are not referred to by RLDs and are not numbered in language translator output.

The enter routine places them directly in the CESD. If an LD is received before the SD to which it belongs, it is handled as an LR.

PR Items: If the input ESD item is a pseudo register, the current segment number is not entered in column 12 of the ESD item (Chart JE). Column 12 of a PR item may contain an alignment value which indicates that the PR must be aligned to a halfword, fullword, or doubleword boundary. The PR is then processed by the freeline, enter, and renumber routines, as described previously.

RESOLUTION PROCESSING: If a matching symbol is found in the CESD, the type fields of the input item and the matching CESD item are compared and resolution processing is then performed. The following conventions are observed during resolution processing:

1. Input PR items may match only PR-type entries in the CESD. If a PR-type input item matches a non-PR item in the CESD, it is not treated as a match; the CESD search for a matching PR item continues.

2. If the matching CESD item is marked "chained," resolution is performed on the item to which it is chained.

3. If the CESD line is marked null, the match is ignored and the search continues.

4. If the CESD item is an ER produced from a REPLACE, CHANGE, OVERLAY, or ALIAS statement, or from the ddname field of an INCLUDE or LIBRARY statement, the match is ignored and the search continues.

Matching items are processed in the following manner:

- If the input ESD item is CM, SD, or LR, and it matches an ER in the CESD, the input type replaces the type indicated in the CESD item (see Diagram B2, Area B). Non-resolution processing is then performed on the input item.

- If the input ESD item is an LR and it matches a CM, SD, or LR in the CESD, a "match" bit is set, indicating that a double symbol definition is possible. If the SD for the control section has been entered in the CESD and is marked for deletion, the label routine deletes the label; if it is not marked for deletion a "double symbol definition" message is produced. If the SD for the control section is not in the CESD, the LR is chained to the matching LR; when the SD is received, the LR is deleted or a double symbol definition is produced, depending on whether or not the SD is being deleted.

- If an input PR matches a PR in the CESD (Diagram B2, Area D), the greater length and the most "constrictive" boundary alignment are placed in the CESD entry. (A doubleword alignment is more constrictive than fullword alignment; fullword is more constrictive than halfword; etc.) The input PR entry is then renumbered to the updated PR entry in the CESD.

- If an input SD item matches an SD entry in the CESD, automatic replacement of the control section occurs. The input SD item is entered into the CESD as a delete-type and is chained to the matching SD entry. (During second pass processing, the assigned address of the control section being replaced will be subtracted ("delinked") from the addresses of any non-branch type address constants that refer to the ER-delete entry.) The SD-delete item remains chained only while the module is being processed; END processing will change the chained items to null-type entries. (Refer to "Delinking Non-Branch Type Address Constants.")

- If an input SD item matches a CM entry in the CESD, the greater length is entered in the length field of the SD entry. If the program is in overlay, the common path routine scans SEGTA1 to find the segment in the overlay structure that is common to both items and places the segment number in the SD entry. The SD item is then written over the CM line and renumbered. (This is referred to as "automatic promotion of common.")

- If an input SD or CM item matches an LR in the CESD, a "double symbol definition" message is produced and the SD or CM item is entered in the CESD as a delete-type item and is chained to the matching LR entry, causing the SD or CM to be replaced.

- If the input item is CM, it may be "blank common." Blank common may match a PC-type CESD item because both contain blanks in the symbol field. In such a case, the match is ignored and the search continues.

- If an input CM item matches an SD or CM item in the CESD (Diagram B2, Area F), the greater of the two lengths is entered in the CESD item. (The CESD type is not changed.) If the module is being processed for overlay, the segment number of the segment common to both the input item and the CESD item is also entered in the CESD item (automatic promotion of common).

- Whenever an input ER item matches an ER in the CESD, both the type and subtype fields are examined; the ER items are then resolved in the following manner:

  1. If the subtype fields of both ER items are not marked, the input item is not entered into the CESD; the matching ER remains in the CESD and a pointer to it is placed in the renumbering table entry for the input item.

  2. If both items are marked "delete," the new ER is entered into the CESD and the old item remains there so that they can be delinked individually (in this case, the CESD may contain two ER items for the same symbol). Delinking is described in "Second Pass Processing."

  3. If the input ER item is marked for deletion, but the ER item in the CESD is not marked delete, the input ER is chained to the matching ER in the CESD. The chained ER item remains in the CESD until the end of module is detected so that the delink value can be saved.

  4. If the input ER item is not marked for deletion and the ER item in the CESD is marked "delete" or "replace," the delete bit in the subtype field is cleared (delete is changed to replace) and the item is renumbered. If the matching ER item in the CESD is marked "no call" or "library member" it is marked "matched" before renumbering.

  5. If the input ER item is marked in the subtype field, but is not "delete" or "replace," it is assumed to be "never call"; if the matching ER item in the CESD is "library member," the CESD item is removed from the chain of library members and the input ER item is entered into the CESD and renumbered.

TXT Processing

The manner in which TXT records are processed depends on whether they are part of a load module or an object module. A load module contains records in a specified order. However, in an object module the records may not be in the proper sequence because the language translator may have created them out of order. (The restrictions on linkage editor input are described in the Appendix under "Input Conventions.") Diagrams B3 and B4 illustrate processing of TXT records from object and load modules, respectively.

Before any address constants can be relocated within a control section of an object module, all TXT records must be placed in the proper order. This is done in the input text buffer (TXTBFBEG), which is variable in length, allowing grouping of data within the buffer.

Each "multiplicity" of text is assigned a number as it is moved (or read) into TXTBFBEG. A multiplicity is a portion of text equal in length to the maximum size of a SYSLMOD output record. Within each control section, multiplicity numbers are assigned consecutively, starting at 0.

Text records from object modules contain both text data and the control information needed for processing. Text records from load modules contain only text, so the associated control record must also be examined to obtain the required control information. During object module processing, control information is placed in registers; this information allows the object module text to be moved from the object module buffer into TXTBFBEG. For load module text, the assigned address of the first byte of text and a pointer to the ID-length list (in the control record) is determined during load module processing. This information allows the text record to be read directly into TXTBFBEG.

Processing Object Module Text

When text is received from an object module, the text record ID is renumbered, using the renumbering table, so that it refers to the CESD entry for the control section which contains the text. The size of the control section is obtained from the CESD, and a test is made to determine if the whole control section or a multiplicity (whichever is smaller) will fit into the space available in TXTBFBEG. (If the control section length was not specified in the CESD entry, only text for the current ID is accepted; refer to the paragraph headed "No-Length Control Sections.")

If there is sufficient space in TXTBFBEG to accommodate the control section or multiplicity, the text is moved into the buffer, and an entry (containing the ID and multiplicity number of the text) is made in the text I/O table. A corresponding entry, containing the location of the multiplicity and the length of the text, is made in the text note list. The text note list entry also contains a displacement field. When text is in order, or on the first occurrence of text for a multiplicity, the displacement field is set to 0; for out-of-order text the displacement field contains the displacement from the beginning of the multiplicity of the first byte of contiguous text.

If the SYSUT1 record size is smaller than the multiplicity size, each multiplicity is divided into pieces, each piece having a length equal to the SYSUT1 record size. New text I/O table and text note list entries are made for each piece; the displacement field will contain the displacement of each piece from the beginning of the multiplicity.

NO-LENGTH CONTROL SECTION: When text is received for a no-length control section (a control section for which no length is specified in its CESD item), space for one multiplicity is allocated in TXTBFBEG. Entries are made in the text I/O table and the text note list for the multiplicity, and the text is moved into TXTBFBEG. This procedure is repeated for each subsequent multiplicity of text for the no-length control section. If TXTBFBEG becomes full, its contents are written onto SYSUT1 as described below in the section headed "Writing Text on SYSUT1". When the length is received, it is entered in the text note list.

PROCESSING OUT-OF-ORDER TEXT: A load module contains records in a definite order. However, records in an object module may not be in the proper sequence because the language translator may have created them out of order.[1] Such records may contain discontinuities in addresses (due to a reorigin or a disjointed control section), or they may not be contiguous (i.e., text of a given ID and multiplicity may be interspersed with text of other IDs or multiplicities). Records of contiguous text must be built on SYSUT1 so that during second pass processing the text can be placed into its proper position, within its ID and multiplicity, in the second pass text buffer.

The first occurrence of a given ID and multiplicity is read into the input text buffer as it is received. Discontinuities and non-contiguous text are of no consequence at the first occurrence of an ID and multiplicity. However, once text of a given ID and multiplicity has been written out on SYSUT1, any subsequent text of that ID and multiplicity must be contiguous to be written out on SYSUT1 within each text record.

Text of a previously-written ID and multiplicity is read into the input text buffer until a discontinuity, or text of a different ID or multiplicity, is encountered. The contiguous text in the buffer is then written out on SYSUT1. The discontinuous (or non-contiguous) text is then placed in the buffer. If this text represents the first occurrence of an ID and multiplicity, the buffer is loaded without regard for discontinuities or non-contiguous text. If the text belongs to a previously-written ID and multiplicity, the text processor will again place only continuous text of that ID and multiplicity in the buffer.

A record that contains non-contiguous text is called a "loose" record; a record that contains contiguous text is called "dense." The text note list entry for a dense record usually has a nonzero value in the displacement field. When the text is read back from SYSUT1 into the second pass text buffer, during second pass processing, this displacement is used to place the text in its proper position within its ID and multiplicity.

Processing Load Module Text

Since text records from load modules are ordered and well-defined, they require little further processing by the text processor. The information in the ID-length list (in the control record) is scanned, and each ID is renumbered and checked to determine if it is to be deleted. If all IDs are to be deleted, the record is ignored, and control is returned to the input processor.

When an ID that is to be processed is found, the text record containing the ID must be read into TXTBFBEG. The text record length is obtained from the associated control record and compared against the free space available in TXTBFBEG. If sufficient space is available, the text record is read into the buffer; otherwise, the contents of the buffer is written onto SYSUT1 to ensure sufficient space, and the record is read.

Text is processed in the buffer in the order specified by the ID-length list (in the control record). IDs that are to be deleted are overlaid by IDs that are to be processed. The text is divided into multiplicities and entries are made in the text I/O table and the text note list. When all text identified by the ID-length list is processed, text processing is completed.

----

[1] The restrictions on linkage editor input are described in Appendix A under "Input Conventions."

Writing Text on SYSUT1

When no more control sections can be accommodated in TXTBFBEG, the contents of the buffer must be written onto the intermediate data set (SYSUT1). The text I/O table is scanned to determine the order in which control sections are to be written. The length of the first control section (i.e., corresponding to the first text I/O table entry) is obtained from its corresponding ESD ID; if the length is less than the size of the SYSUT1 record, the text I/O table entry for the control section is marked "written." Each subsequent control section is similarly processed, and its length is added to the sum of the lengths of previously processed control sections.

When the sum of control section lengths reaches the limit of a SYSUT1 record, the entire group of control sections is written onto SYSUT1. The relative track address (TTR) is placed in the text note list entry corresponding to the last text I/O table entry that was processed.

When a single control section is larger than a SYSUT1 record, the multiplicities of the control section are grouped, up to the limit of the SYSUT1 record size, and written.[1] When control sections or multiplicities are grouped on SYSUT1, the multiplicities must be in ascending consecutive order. If the overlay option has been specified, no grouped control sections are permitted on SYSUT1.

Note: Each time an entry is made in the text note list during text processing, a check is made to determine if the list is full. If it is full, the contents of TXTBFBEG are grouped (if possible) and written onto SYSUT1, and the TTRs are placed in the text note list. The list is then written onto SYSUT1, and its address is noted in the I/O control table. The text note list may be written a maximum of three times.

If neither TXTBFBEG nor the text note list becomes full during text proccessing, no text is written onto SYSUT1. The text is retained in the buffer, and single-pass processing is in effect for text records.

RLD Processing

RLD processing basically consists of:

1. Updating each set of relocation and position pointers (R and P pointers).
2. Processing each flag and address (FA) in the input item until the end of the record or the next item with an R and P pointer is detected.

RLD records from object modules and load modules are processed in the same manner. During object or load module processing, a pointer to the first RLD record encountered in a load module or object module record is placed in register 6.

RLD information is grouped in the RLD buffer by P pointer. Each P pointer of an input RLD record refers to the ESD entry in the input module for the control section that contains the address constant. Each time a new P pointer (one referring to a different ESD ID) is detected, an entry is made in the RLD note list for the RLD set (a set being an unbroken sequence of RLD items having the same P pointer). The RLD note list entry contains the following information for each set:

1. The renumbered P pointer to which these RLDs refer.
2. The lowest multiplicity of text to which these RLDs refer.
3. The number of bytes of RLDs.
4. The storage address of the first byte of RLD data if all RLDs remain in core; if RLDs are written onto SYSUT1, this field contains the accumulated byte count for intermediate chains, or the TTR of the record on SYSUT1.

All adjacent RLD items containing the same P pointer are referred to by only one RLD note list entry. Adjacent RLD items containing the same R and P pointers are chained, with the R and P pointers appearing only once, at the beginning of the chain. The remaining RLDs in the chain are compressed by setting the flag indicating continuation and discarding the four bytes containing the R and P pointers.

Each R pointer of an input RLD record refers to the ESD entry in the input module on whose value the address constant depends. The R and P pointers are updated, using the renumbering table. Before renumbering, the R and P pointers refer to ESD entries of the input module that contains the RLD items. The pointers are renumbered so that they point to the proper entries in the CESD being created for the output load module. If the R pointer refers to a deleted ESD entry, delinking may be performed. If the assigned address

[1] If the SYSUT1 record size is smaller than the SYSLMOD record size, no grouping is permitted.

of the symbol referred to by the address constant is zero, the address constant is not delinked. (Normal relocation is performed.) When delinking is necessary, an entry is placed in the delink table (a function of ESD processing). The delink table entry contains the address (delink value) of the symbol being deleted and the CESD entry number of the identically named symbol that is to replace the deleted symbol.

The ID of the delink table entry for the deleted symbol is saved in the renumbering table, and a "delink value saved" indicator is set. The ID of the indentically-named symbol and the ID of the new delink table entry are saved because they are later used to complete the delinking operation. The R pointer of the RLD item must be modified to refer to the delink table entry for the deleted symbol, but the original R pointer is needed to process any V-type address constants referred to in the RLD item. Therefore, the R pointer is not modified until the string of flag-address (FA) fields following the R and P pointers has been processed as described below. At that time, if the module is to be structured for overlay and it contains V-type address constants[1] that refer to the symbol, the ID of the identically-named symbol is inserted into the calls list.

Each FA field of the RLD record is processed as follows:

- The high-order bit of the flag field is set to zero.
- If the address constant is an A-type, the renumbering table entry referred to by the R pointer is checked to determine if it is marked as a PR type. If it is a PR, the RLD flag field is also marked PR (because second pass processing must handle PRs in a special manner). If the renumbering table entry is not an ER or marked delete, the RLD flag field is marked for relative relocation. This indicates to second pass processing that the difference between the origin of the control section in the input and the origin assigned by the linkage editor is to be used as a relocation factor for the value of the address constant. If the RNT entry is an ER or marked delete, the RLD flag field is not marked. This indicates to second pass processing that the address constant is to be relocated by absolute relocation; second pass processing uses the linkage editor assigned address of the symbol in the output module as a relocation factor for the value of the address constant. (This procedure is described in the paragraph "Second Pass Processing.")
- If the address constant is a 4-byte V-type ("branch-type"), and the program is in overlay, an entry is placed in the calls list, provided that the address constant refers across control sections (R not equal P). The calls list is used during address assignment processing to determine which segments require ENTABs, and the number of entries each ENTAB must contain.
- For both A-type and V-type address constants, the multiplicity of the address field is determined and is saved in the RLD note list if it is lower than any previous multiplicity in the RLD record. If two-pass processing is in effect, the RLD note list is used during second pass processing to read back RLD data from SYSUT1 (each RLD note list entry contains the relative track location (TTR) of an RLD record on SYSUT1). The second pass processor uses the multiplicity field of the RLD note list entry to determine if the associated RLD record should be read back from SYSUT1 for a given multiplicity of text.

  When the last FA field in the string has been processed, all items in the string have been checked to determine if they require delinking. If any A-type address constants in the string required delinking, the R pointer for the string is modified to refer to the associated delink table entry.

Table 6 shows the actions performed during RLD processing for each input flag format, and the format of the flags after RLD processing. (The "output" column shows the flag formats that are passed as input to the relocation routine of second pass processing; refer to Table 7.) After all FA fields have been processed, the next RLD record is processed.

If the RLD buffer becomes full, its contents must be written onto the intermediate data set (SYSUT1). The RLD buffer is allocated with a maximum length less than or

[1] V-type address constants do not require delinking, but may be in a FA string with A-type address constants that do require delinking (or other control sections in the same input module may contain A-type address constants that refer to the deleted control section).

equal to the size of a SYSUT1 record, so the entire buffer may always be written. As many consecutive RLD sets as possible are grouped in a SYSUT1 record. The RLD note list entry for each RLD set in the group contains a "grouped" indicator; the note list entry for the last RLD set in the group also contains the relative track address (TTR) of the group.

RLD sets whose length exceeds that of a SYSUT1 record (requiring more than one output record) are not grouped. RLD note list entries for RLD sets that are not grouped contain the relative track address (TTR) of the SYSUT1 record and a "non-grouped" indicator.

Each time an entry is made in the RLD note list, a check is made to determine if the list is full. If it is full, the RLD sets in the RLD buffer are grouped and written onto SYSUT1, and the TTR is placed in the appropriate RLD note list entry. The RLD note list is then written onto SYSUT1, and its address is noted in the I/O control table. The RLD note list may be written a maximum of three times.

Note: If neither the RLD buffer nor the RLD note list becomes full during RLD processing, no RLDs are written onto SYSUT1. The RLD information is retained in the RLD buffer, and single-pass processing is in effect for RLDs.

Table 6. Flag Field Processing

Input			Output	
≠ Flag	Type	Action Performed	Flag	Type
0000LLST	Not PR, ER, CM, or delete	Marked for relative relocation	1000LLST	Relative
0000LLST	ER ('02' in renumbering table)	Marked for absolute relocation	0000LLST	Absolute
0000LLST	Delete or CM ('05')	Marked for absolute relocation is assigned address of input item is zero	0000LLST	Absolute
0000LLST	PR ('06')	Marked as PR (displacement value)	0010LLST	Pseudo Register Type 1
0000LLST	Delete or CM	Marked "delink value saved" if assigned address of input item is not zero	High-order bit of P pointer	Delink
0001LLST	Type is not checked	RLD is marked branch-type	0001LLST	Branch
0001LLST or *1001LLST	Delete	Marked "delink value saved and other FA items in string exist that are non-branch type" and are being delinked	High-order bit of P pointer.	Delink
0010LLST	Pseudo Register Type 1	None - Remains as a PR (displacement value)	0010LLST	Pseudo Register Type 1
0011LLST	Type is not checked	Marked as PR (cumulative length)	0011LLST	Pseudo Register Type 2

*Internal types processed during second pass.
≠Refer to "RLD Input Record (card image)" and "RLD data" (load module) in Section 7: Appendix.

END Processing

When an END statement or the end of an input load module is detected, END processing is required. The functions of END processing include:

- Reset tables (such as the renumbering table) that were involved in the processing of the input module.
- Process entry point information.
- Delete any CESD lines marked CHAIN or DELETE, and keep track of deleted lines.
- Enters in the CESD the length of a control section for which no length was specified in the ESD item (if the length is contained on the end statement).

Include Processing

Include processing is required when:

1. The control statement scanner has detected an INCLUDE statement and the include statement processor has built an include chain.
2. End-of-input has been detected, and the "more includes" indicator in the all purpose table is on.

Include processing consists of preparatory functions (OPEN, BLDL, FIND) required before the module to be included can be read.

- An input pointer to the library read block is set.
- The SYSLIB DCB is closed (unless it is open for a partitioned data set currently being used).
- Each entry in the include chain is examined sequentially.

SEQUENTIAL DATA SETS: If an include chain entry specifies a sequential data set, the data set organization field of the DCB is changed from partitioned to physical sequential, and the ddname field is updated. The DCB is then opened, and the module is read in.

PARTITIONED DATA SETS: If an include chain entry specifies a member of a partitioned data set, the member name is entered into the BLDL list, and the next entry is examined. If the next entry specifies a different data set name, the partitioned data set is opened, and a BLDL macro instruction is executed for the single member name.

If the next entry specifies another member of the same partitioned data set, the member name is added to the BLDL list, and the next entry in the include chain is examined. Member names are added to the BLDL list until a different data set name is encountered, the BLDL list becomes full, or the end of the include chain is reached. Since the BLDL list must be in collating sequence, each member name is inserted into its proper position, moving other entries as necessary. Since included modules must be read in the order in which they appear in the INCLUDE statement (without regard for collating sequence), a separate table, indicating the order of processing BLDL list entries, is maintained.

When the BLDL list is completed, the partitioned data set is opened and the record format field (RECFM) in the DCB is tested to determine if the included modules are load modules (U-format) or object modules (F-format). If they are load modules, the "load module" indicator is set in the APT. This indicator is tested when each module is read in. A BLDL macro instruction is then executed for the member names in the list. The list is then examined in the order specified in the INCLUDE statement to obtain the attributes of each included module (if it is a load module); the attributes of the output load module may be "downgraded" accordingly in the APT.

If the BLDL macro instruction was successful for a particular member, the member is read in. The FIND macro instruction and the directory entry obtained from BLDL are used to set a pointer in the DCB to the first record of the member. If the BLDL was not successful for a particular member, a diagnostic message is printed.

Note: If a nested INCLUDE statement is encountered, it is processed immediately, without attempting to construct a multiple BLDL list.

An example of include processing is given in Figure 20. The input pointer is set to the address of the library read block. The address of the current include item is contained in the all purpose table.

Assuming that no includes have yet been processed, A will be the first item examined. The subtype 'D0' indicates that A is a member of a partitioned data set, so A will be entered into the BLDL list. The pointer 000D refers to the data set DATASETX. The next item in the include chain, B, is also a member of DATASETX, so it is added to the BLDL list. The next item in the chain, M, is a sequential data set (subtype C0), so the BLDL list is completed with two entries (A and B). Assuming that DATASETX is not currently open and the SYSLIB DCB is not opened for another data set, the SYSLIB DCB is opened for DATASETX. (The RECFM field of the data set DSCB is merged into the DCB.) Assuming that the RECFM field indicates U-format, a load module indicator is set in the all purpose table, and a pointer to the load module buffer is placed in the library read block. The attributes of A and B are obtained, using BLDL, and the attributes specified on the EXEC statement are updated accordingly. (The attributes of the output load module may be downgraded as a result.) A pointer in the DCB is then set to the first record of member A, using the FIND macro instruction, and the "include initiated" indicator is set in the all purpose table.

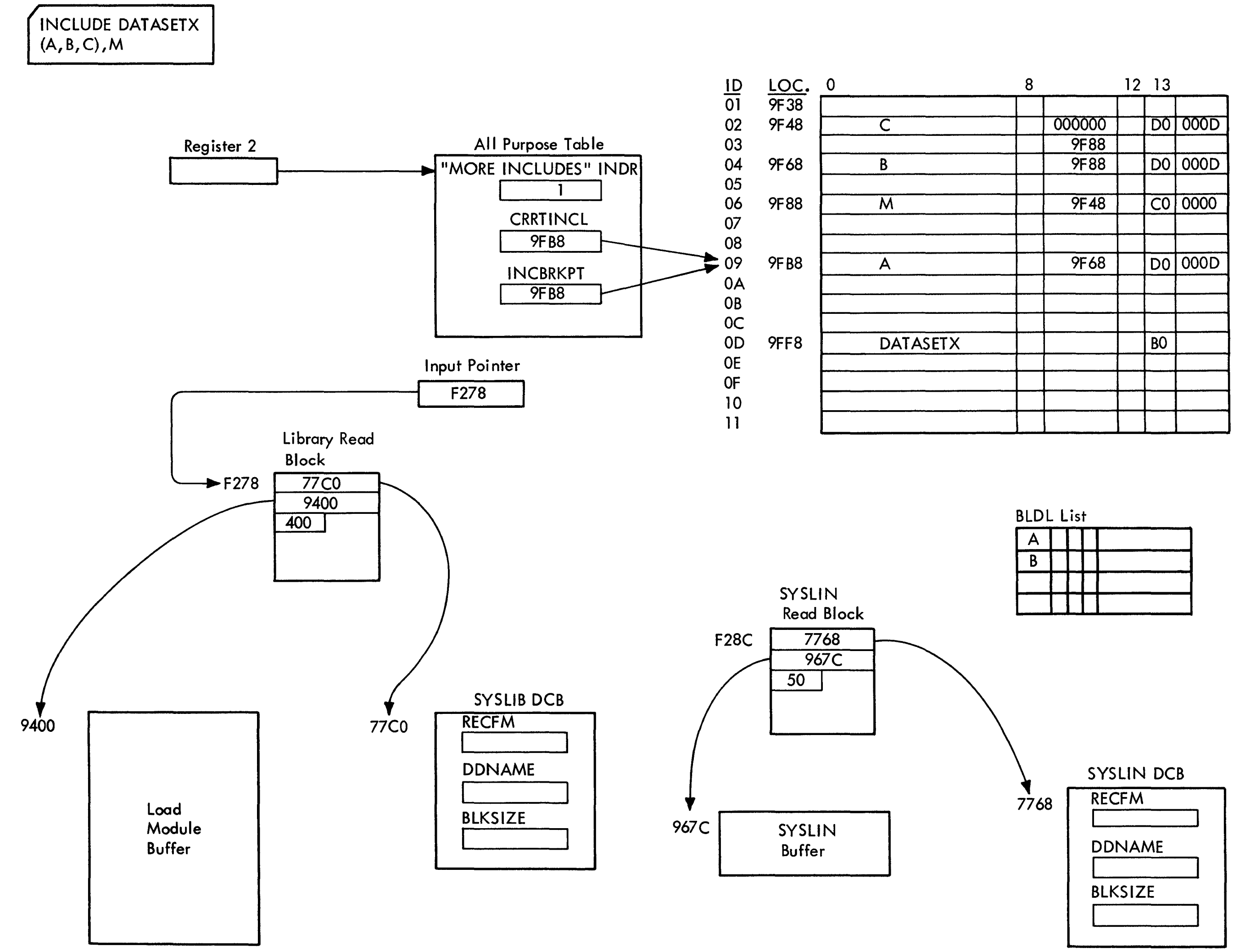
Member A is read using the input pointer and library read block. Module A is then processed. When the end of module A is reached, item A is deleted from the chain and the CESD line is marked "null." Member B is then read and processed.

When the end of module B is reached, item B is deleted from the chain, the CESD line is marked "null," and the remainder of the chain is processed.

Automatic Library Call Processing

Automatic library call processing is required:

- At the end of SYSLIN input when unresolved ERs still exist, and the NCAL option was not specified.
- When a NAME statement has been detected (provided that the NCAL option was not specified and no more includes are to be processed).



ID	LOC.	0	8		12	13	
01	9F38						
02	9F48	C		000000		D0	000D
03				9F88			
04	9F68	B		9F88		D0	000D
05							
06	9F88	M		9F48		C0	0000
07							
08							
09	9FB8	A		9F68		D0	000D
0A							
0B							
0C							
0D	9FF8	DATASETX				B0	
0E							
0F							
10							
11							

Figure 20. Include Processing

Automatic library call processing consists of two series of CESD scans. The first series of scans operates on unresolved ERs specified on LIBRARY statements. It finds the first ddname that contains a pointer in the chain pointer field (bytes 14 and 15). Such an entry is the first item in a chain of members associated with this ddname; there is a distinct chain for each ddname that was specified on a LIBRARY statement. Chained member names for a particular ddname are entered into a BLDL list which is processed as previously described under the heading "Include Processing."

The scan of the CESD continues until all ddname chains have been processed. A second scan of the CESD then searches for external references not specified on LIBRARY statements and attempts to resolve them by calling members of the same name from SYSLIB.[1]

An example of automatic library call processing is given in Figure 21. Diagram A shows two library chains that were built in the CESD by the library statement processor. In diagram B, an SD item for JOE has been entered into the CESD, resolving the reference to JOE. (JOE was removed from the chain by ESD processing, and the LIB1 chain ID now points to the line containing TOM.) Automatic library call processing operates on the library chains, as modified by ESD processing (diagram B).

In the first series of scans, the CESD is searched for a ddname (type 02, subtype B0) with a chain pointer. The ddname item LIB1 is found; its chain ID points to TOM. Because TOM is unmatched (subtype 02) it is not called and since TOM is the last item in the chain (0 in the chain ID field), the scan is resumed for another ddname with a chain pointer. LIB2 is found; its chain ID points to SAM. No call is issued for SAM, since it is unmatched. The chain ID of SAM points to PETE, which is matched (indicating that PETE is an external reference, and not just an operand of a LIBRARY statement). PETE is entered into the BLDL list; since PETE is the last item in the chain, the list is completed with one entry.

LIB2 is opened and the BLDL macro instruction is used to obtain the attributes of PETE (the attributes of PETE are not obtained if the format is F). A "BLDL attempted" indicator is set for the CESD entry for PETE so that no other search for PETE will be made in the event of an unsuccessful BLDL or non-resolution of the ER for PETE by the member PETE. The FIND macro instruction is used to set a pointer in the SYSLIB DCB to the member PETE; PETE is then read in.

When processing for PETE is completed, the scan for ddnames resumes at the beginning of the CESD, rather than at the CESD line where the scan was interrupted, because additional ddname items may have been entered at any available line in the CESD. (Object modules with additional LIBRARY statements may have been read in.) When the last line of the CESD is reached the second series of scans is begun.

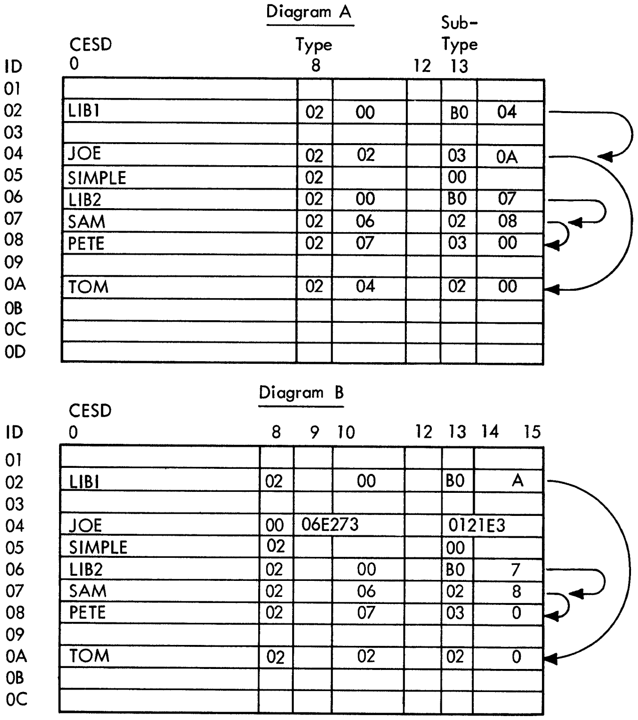


Diagram A

ID	CESD 0	Type 8		12	Sub- Type 13	
01						
02	LIB1	02	00		B0	04
03						
04	JOE	02	02		03	0A
05	SIMPLE	02			00	
06	LIB2	02	00		B0	07
07	SAM	02	06		02	08
08	PETE	02	07		03	00
09						
0A	TOM	02	04		02	00
0B						
0C						
0D						

Diagram B

ID	CESD 0	8	9	10	12	13	14 15
01							
02	LIBI	02		00		B0	A
03							
04	JOE	00	06E273			0121E3	
05	SIMPLE	02				00	
06	LIB2	02		00		B0	7
07	SAM	02		06		02	8
08	PETE	02		07		03	0
09							
0A	TOM	02		02		02	0
0B							
0C							

Figure 21. Automatic Library Call Processing

During the second series of scans, the CESD is searched for "unmarked" external references (type '02', subtype '00'). These are ER items not specified on LIBRARY statements. In diagram B, the scan finds SIMPLE. Assuming that SYSLIB is the ddname for the standard library, SIMPLE is called from SYSLIB in the same way that PETE was called from LIB2. Every time automatic library call processing is resumed after a module is read, the second series of scans resumes at the beginning of the CESD (because ER items from a library member may have been entered in any available CESD line).

When the second series of scans is finished, input processing is complete.

[1] SYSLIB is the standard library whenever the linkage editor is executed as a job step. If another program LINKs to the linkage editor, the ddname of the standard library is passed in a parameter list.

INTERMEDIATE PROCESSING

When all input processing is completed, the second phase of Linkage Editor F (intermediate processing) begins operation. The two major functions of the second phase are address assignment and intermediate output.

ADDRESS ASSIGNMENT (IEWLMADA)

At the conclusion of input processing, address assignment processing is required. (See Diagram C1.) Address assignment includes the following operations:

- CESD entries are deleted for ER items marked included, called, ddname, or overlay in the subtype field. These lines are marked "null" and are deleted if the module is processed again in a subsequent execution of the linkage editor.

- Compute, for programs in overlay, the size of SEGTAB[1] enter the size in the all purpose table, and place a private code delete entry for the SEGTAB in the CESD. The PC-delete type entry is deleted from the module if it is processed again by linkage editor. (Diagram C1, Area A)

- Enter segment numbers for label references in the CESD. If the program is in overlay, the calls list (built during RLD processing) is also scanned, and pointers from one chain of calls to the next chain are entered; (Area B) the number of ENTAB bytes[2] for each segment is determined; and a PC-delete type entry is placed in the CESD for each ENTAB. (Refer to "ENTAB Size Determination.")

- Assign temporary linked addresses to SD-, PC-, and CM-type entries in the CESD (Area C). CSECTs are processed according to the order of input determined by scanning entries in the text I/O table. Since an ID can appear more than once in the text I/O table, a "processed" bit (bit 4 of the "type" byte) is set in the CESD entry to indicate that a temporary linked address has been assigned to the associated CSECT. The "processed" bit must be reset to 0 before address assignment processing is terminated. CSECTs that do not contain text have no entries in the text I/O table. After processing all CSECTs with text, addresses are assigned to CSECTs without text by referring to the CESD.

- Each segment is considered to be at a zero origin. The temporary starting address of each control section is computed with respect to its location in the segment, relative to the zero origin (plus any adjustments for boundary alignment). These addresses are temporary because the starting addresses of the segments must later be relocated with respect to their positions in the overlay tree. If the program is not in overlay (consists of a single segment) the addresses are final, because no further relocation by address assignment is necessary.

- Compute the temporary relocation constant for each control section (the difference between the temporary linked address and the assigned address in the input) and place it in the relocation constant table (RCT) (Area D). If the program is not in overlay, these are the final relocation constants (relative relocation factors).

- Accumulate the length of each segment in the leftmost three bytes of an entry in the segment length table (SEGLGTH). The boundary alignment factor of the first control section in the segment is placed in the fourth byte of the entry.

- Determine the address of each PR-type entry in the CESD, using the total length of all PRs previously encountered, plus the boundary alignment factor. This address is placed in the CESD entry for the PR. The length of this PR is then added to the cumulative PR length.

- Process the SEGLGTH table (if the program is in overlay) to determine the starting address of each segment, relative to the beginning of the program. (Area E) SEGTA1 is checked to find the proper location of each segment in the tree. SEGLGTH at this time contains the length of each segment. To determine the starting address of a segment, the length of all previous segments in the same path are added, together with any adjustments for boundary alignment.

[1] SEGTAB size = 24 + (4 x number of segments).
[2] ENTAB size = 12 + (12 x number of unique downward calls per segment).

(Boundary alignment adjustment is determined by the last three bits of the address of the first control section in a segment.) This sum, minus the boundary alignment factor for the segment, is the segment relocation constant (SRC). The SRC is then placed in the rightmost three bytes of the SEGLGTH table. The sum of the SRC, the boundary alignment factor, and the segment length is placed in the leftmost three bytes of the SEGLGTH table entry for the segment. It is the length of the path of the segment (including the segment itself). At the completion of this process, the entry in SEGLGTH for each segment contains the cumulative length of its path; the longest of these lengths is the program length.

- Perform a second scan of the CESD if the program is in overlay. The segment relocation constant in the SEGLGTH table is added to the temporary linked address in the CESD entry for the control section; this sum is the final linked address. The SRC is also added to the temporary relocation constant in the relocation constant table; this sum is the final relocation constant for the control section.

- Make a final scan of the CESD to assign a final linked address to each label reference.

  The CESD entry for each LR contains a reference to the control section in which it resides. The relocation constant for that control section is located in the RCT and is added to the temporary linked address in the CESD entry for the LR. This sum, the final linked address for the LR, is placed in the CESD.

- Mark the program as not executable if there are still unresolved external references and if neither the no call (NCAL) option nor the LET option has been specified.

- Build the alias table and compute an entry point for the program. (Refer to "Entry Processing.")

ENTAB Size Determination

ENTAB size determination consists of computing the size of ENTABs so that the size of each segment in an overlay program can be determined and relative relocation factors can be computed for use by second pass processing. The size is determined by the number of downward calls, or calls across regions, to symbols that are not referred to by segments higher in the path of the calling segments.

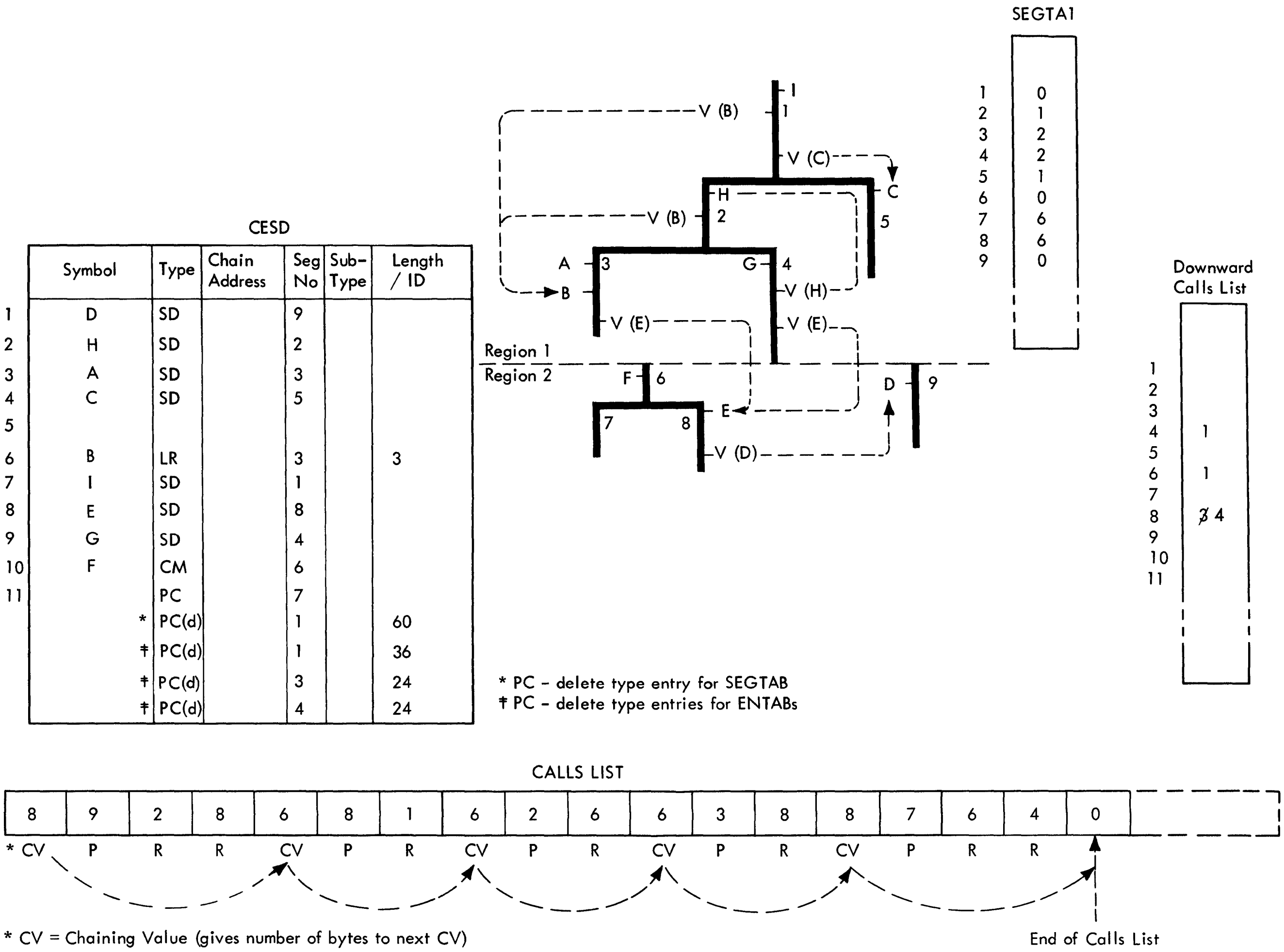
An example of ENTAB size determination is given in Figure 23. The overlay tree structure shown in the illustration consists of nine segments residing in two regions; all references between segments are made using V-type address constants. Functions of ENTAB size determination are:

- Scanning the CESD for LR-type entries and entering their segment numbers. In Figure 23, item 6 is an LR item; its ID/length field points to the CESD entry for the control section in which it resides (line 3). The segment number contained in line 3 (segment number 3) is entered in the segment number field of the LR item.

- Scanning the calls list, inserting chaining values that point from one group of R and P pointers to the next.

- Scanning the calls list, for each segment (starting with segment 1), find symbols referred to by that segment. For each reference found, the type of call (upward, downward, or exclusive) is determined. If an ENTAB is required for the segment, its size is determined and a PC-delete type entry for the ENTAB is made in the CESD. Referring to Figure 23, the segments are processed in the following manner:

  1. The calls list is scanned for P pointers that refer to control sections in segment 1. If one is found, the associated R pointers (which refer to referenced symbols) are examined to determine the segment in which each referenced symbol resides. In Figure 23, the fifth P pointer refers to line 7 of the CESD, which contains an SD-type entry for a control section in segment 1. The associated R pointers refer to line 6 (symbol B in segment 3) and line 4 (symbol C in segment 5). For each reference, the type of call (upward, downward, or exclusive) is determined, using SEGTA1 and the segment numbers of the calling and called segments. In Figure 23, SEGTA1 indicates that segment 1 is in the path of segments 3 and 5; therefore, the calls from segment 1 to B and C are downward calls. This is noted in the downward calls list by entering segment number 1 in the lines referred to by the R pointer (lines 6 and 4). Since segment 1 is the root segment, it must have an ENTAB; the size of the ENTAB is determined and a PC-delete type entry for the ENTAB is created in the CESD.



CESD

	Symbol	Type	Chain Address	Seg No	Sub-Type	Length / ID
1	D	SD		9		
2	H	SD		2		
3	A	SD		3		
4	C	SD		5		
5						
6	B	LR		3		3
7	I	SD		1		
8	E	SD		8		
9	G	SD		4		
10	F	CM		6		
11		PC		7		
	*	PC(d)		1		60
	†	PC(d)		1		36
	†	PC(d)		3		24
	†	PC(d)		4		24

SEGTA1

1	0
2	1
3	2
4	2
5	1
6	0
7	6
8	6
9	0

Downward Calls List

1	
2	
3	
4	1
5	
6	1
7	
8	~~3~~ 4
9	
10	
11	

* PC - delete type entry for SEGTAB
† PC - delete type entries for ENTABs

CALLS LIST

8	9	2	8	6	8	1	6	2	6	6	3	8	8	7	6	4	0
* CV	P	R	R	CV	P	R	CV	P	R	CV	P	R	CV	P	R	R	

* CV = Chaining Value (gives number of bytes to next CV)

End of Calls List

Figure 23. ENTAB Size Determination

2. When the scan for segment 1 is completed, the calls list is scanned for P pointers that refer to segment 2. In Figure 23, the third P pointer in the calls list refers to CESD line 6, which contains segment number 3. This indicates (via SEGTA1) a downward call from segment 2 to symbol B in segment 3. In this case, however, no entry is made in the downward calls list because it indicates a call to B in segment 3 from segment 1, which is higher in the path of the calling segment (segment 2). No ENTAB is required for segment 2 because the reference to symbol B in segment 2 can be resolved through the ENTAB entry in segment 1.

3. The calls list is scanned for P pointers that refer to segment 3. In Figure 23, the fourth P pointer in the calls list refers to CESD line 3 (segment 3). The R pointer refers to CESD line 8 (segment 8). SEGTA1 indicates that the call from 3 to 8 is downward, across regions, and the call is noted in the downward calls list. Segment 3 requires an ENTAB because it contains a downward call to a symbol not referred to by a segment in the path of the calling segment; the ENTAB size is determined, and a PC-delete type entry for the ENTAB is created in the CESD.

4. The calls list is scanned for P pointers that refer to segment 4. In Figure 23, the first P pointer in the calls list refers to CESD line 9 (segment 4). The R pointers refer to line 2 (segment 2) and line 8 (segment 8). SEGTA1 indicates that the call from 4 to 2 is upward, while the call from 4 to 8 is downward across regions.

The upward call is ignored because the address constant can be resolved directly to the referenced symbol. The downward call from 4 to 8 is noted in the downward calls list, replacing the previous entry for segment 3 (because no segment with a segment number greater than 4 can have segment 3 in its path). Since an ENTAB is required, the size is determined and a PC-delete type entry is created in the CESD.

This process continues until all segments have been processed. The required ENTABs are built during second pass processing (Refer to "ENTAB Creation" and "Relocation of V-Type Address Constants in Overlay.")

Entry Processing

Entry processing includes the following operations:

- Enters into the alias table any alias symbols that were chained together and saved in the CESD by the alias statement processor. Each entry in this table consists of an 8-byte symbol field and a 2-byte ESDID field. For each saved alias symbol, the entry processsor scans the CESD for a matching SD-type or LR-type entry. If no match is found, a zero is placed in the ESDID field of the alias table entry for the symbol. If a matching SD or LR entry is found, the ESDID of the alias entry in the chain is placed in the ESDID field of the alias table entry for the symbol. (See Figure 24.) The address assigned by linkage editor to the matching SD or LR and the ESDID of its control section are placed in the CESD entry for the chained symbol, and the type of the chained symbol is changed to null.

- Determines whether the entry point was specified as an address on an END statement, or as a symbol on an ENTRY statement or END statement:

  1. If the entry point was specified as an address on an END statement, the assigned address is determined by either absolute or relative relocation. If the ID on the END statement referred to an ER which was resolved with an SD or LR, the address assigned by the linkage editor to the SD or LR is added to the address from the END statement (absolute relocation). If the ID on the END statement referred directly to an SD or PC, the relocation constant for the SD or PC is added to the address from the END statement (relative relocation).

  2. If a symbolic entry point was specified on an ENTRY statement or END statement, the CESD is scanned for a matching SD- or LR-type symbol. The address of the matching symbol is used as the entry point.

  3. If no entry point was specified, the starting address of the SD- or PC-type control section (not marked delete) with the lowest assigned address is chosen as the entry point. The entry point associated with the main name (not an alias) and all alias entry points must be in segment number one if the program is in overlay.

INTERMEDIATE OUTPUT (IEWLMOUT)

Intermediate output processing includes the following operations:

- Writes out the CESD on SYSLMOD in groups of 15 entries per record.[1] (The last record may consist of less than 15 entries.)

- Builds a half ESD (HESD), consisting of the last eight bytes of each CESD entry. (The symbol is deleted from each CESD entry to conserve main storage space during second pass processing.) The HESD is not complete at this time. (The ID of each label reference is used in building the scatter and translation tables.)

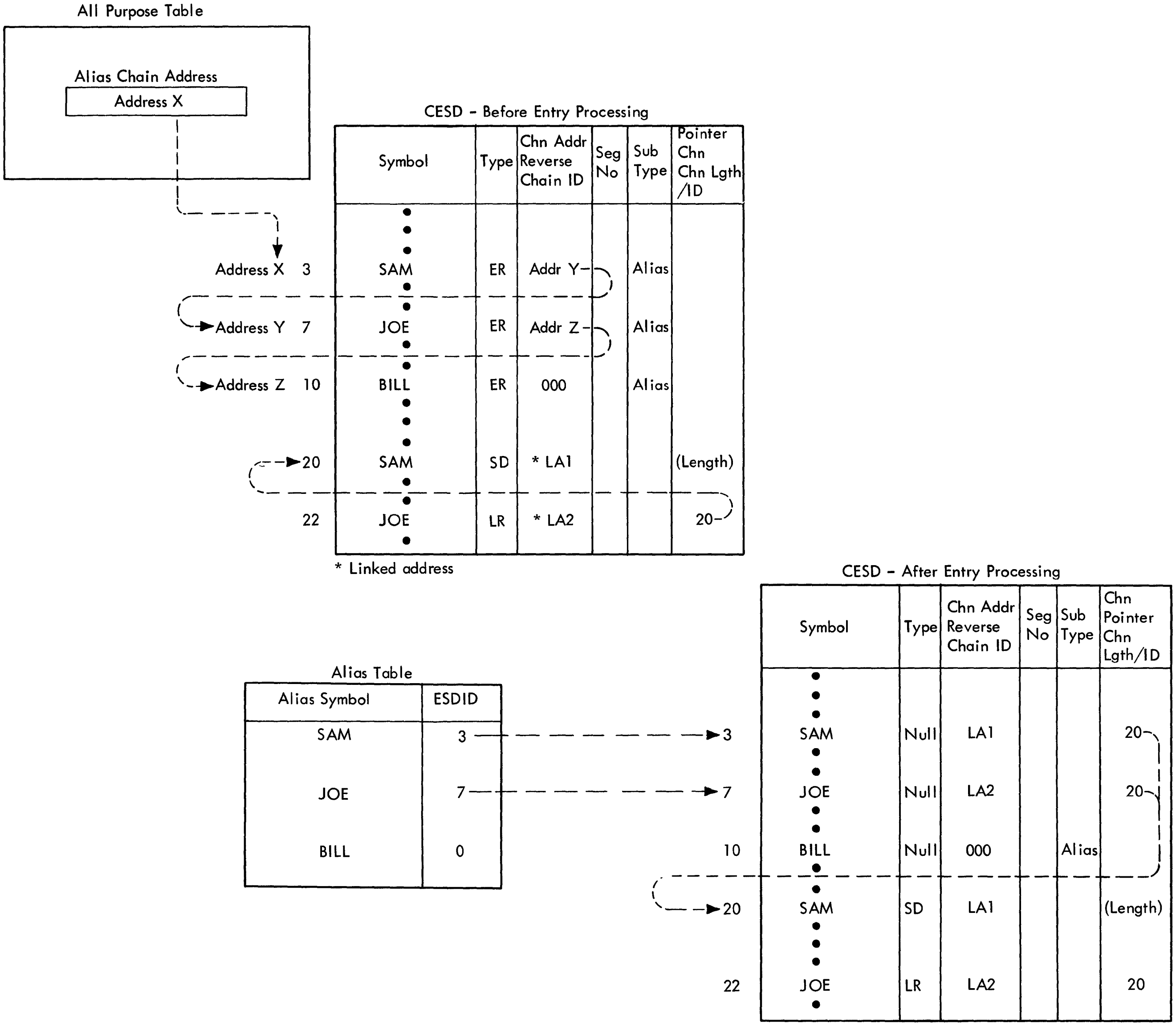
- Builds and writes out the segment table (SEGTAB), preceded by a control record describing it, if the program is in overlay.[2] SEGTAB contains information required by the overlay supervisor.

- Builds a scatter table and a translation table for a program that is to be scatter loaded and writes out scatter/translation records in a form acceptable to program fetch at execution time. The scatter/translation information is written out on SYSLMOD in 1024-byte records. The first four bytes of each record are used to identify the

----

The CESD and control record are not written out on SYSLMOD if the "not editable" attribute is specified.

[2] If it is negative, an indicator is set in the HESD to note that it is in complement form.



CESD - Before Entry Processing

	Symbol	Type	Chn Addr Reverse Chain ID	Seg No	Sub Type	Pointer Chn Chn Lgth /ID
	• • •					
Address X 3	SAM	ER	Addr Y		Alias	
	• •					
Address Y 7	JOE	ER	Addr Z		Alias	
	• •					
Address Z 10	BILL	ER	000		Alias	
	• • •					
20	SAM	SD	* LA1			(Length)
	• •					
22	JOE	LR	* LA2			20
	•					

* Linked address

Alias Table

Alias Symbol	ESDID
SAM	3
JOE	7
BILL	0

CESD - After Entry Processing

	Symbol	Type	Chn Addr Reverse Chain ID	Seg No	Sub Type	Chn Pointer Chn Lgth/ID
	• • •					
3	SAM	Null	LA1			20
	• •					
7	JOE	Null	LA2			20
	• •					
10	BILL	Null	000		Alias	
	• •					
20	SAM	SD	LA1			(Length)
	• • •					
22	JOE	LR	LA2			20
	•					

Figure 24. Processing of Alias Symbols by the Entry Processor

records as scatter/translation information. If the length of scatter/ translation information is greater than 1020 bytes, the last 1020 bytes (plus four bytes of header information) are written out as the first scatter/ translation record. The data in the last record may be 1020 bytes, or less. (See Figure 25.)

- Reads the TXT and RLD note lists into main storage if they were placed on SYSUT1 during TXT and RLD processing. (Each note list may have been written a maximum of three times on SYSUT1 for a large program. In this case, TTRs pointing to the locations of note list information are contained in the I/O control table.)

- Determines the control section containing the last text in the program (or in each segment, if the program is structured for overlay), and the highest segment number of the segments that contain text. (This information is necessary so that second pass processing can determine when to set the end-of-segment or end-of-module indicator.) The highest ESDID is determined by scanning the text I/O table for the ESDIDs of control sections that contain text. This ESDID is entered into the <u>high ID</u> (HIID) table along with its associated segment number.

- Determines, via bits in the <u>all purpose table</u> (APT), if the MAP option has been specified, or if the XREF option has

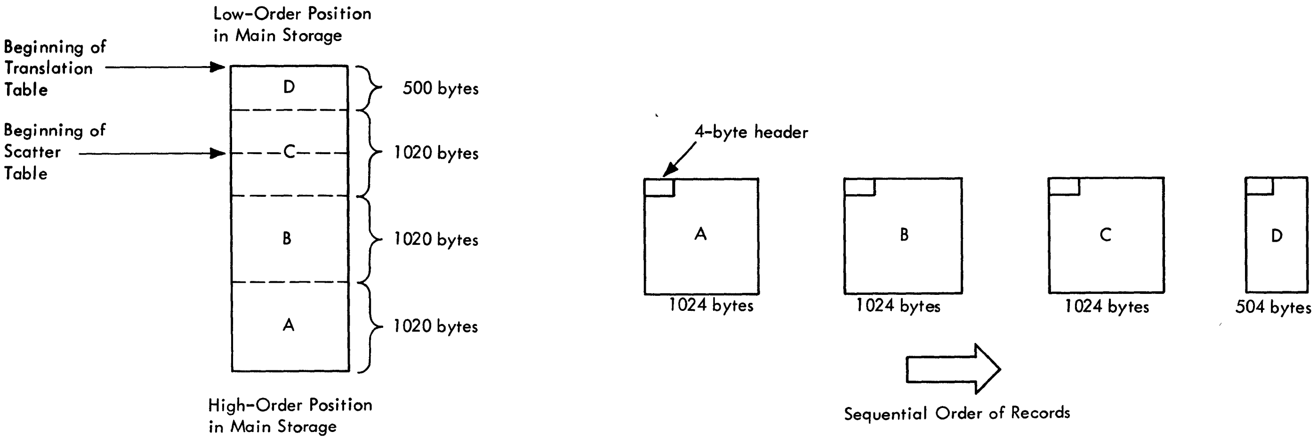

Low-Order Position in Main Storage
Beginning of Translation Table
Beginning of Scatter Table
D
C
B
A
500 bytes
1020 bytes
1020 bytes
1020 bytes
High-Order Position in Main Storage
4-byte header
A
B
C
D
1024 bytes
1024 bytes
1024 bytes
504 bytes
Sequential Order of Records

Figure 25. Writing Scatter/Translation Records

been specified and all RLDs are in storage. If either of these conditions exists, the module map and/or the cross-reference table are produced. If the XREF option is specified and all RLDs are not in storage, XREF processing will be done as part of final processing.

MAP/XREF Processing

When MAP/XREF processing is required as part of intermediate output processing, a table address is obtained from the APT, and a table of two-byte entries pointing directly to the CESD is constructed. The CESD records for the current segment are gathered and sorted by address. The module map is then printed out; the map lists, in ascending order according to their assigned origins, all control sections contained in the output module and the entry points within the control sections. Control sections in an overlay output module are grouped by segment.

If XREF processing is done during intermediate output processing, RLD items are incompletely relocated; their addresses are relative to the origins of their respective CSECTs rather than the origin of the load module, and the address of each RLD must be added to the linkage editor assigned address of its corresponding CSECT before the cross-reference table is produced. The cross-reference table includes a module map and a list of all references within a given segment that refer across control section boundaries. Each entry in the list contains the address of the reference, the symbol to which it refers, and the name of the control section in which the symbol is defined. For overlay programs, each item in the list also contains the number of the segment in which the symbol is defined.

If the MAP and XREF options are processed during intermediate output processing, ALIAS and NAME messages and the diagnostic message directory are printed after the module map and cross-reference table. If the cross-reference table is produced during final processing, the ALIAS and NAME messages are printed before the map and table, and the diagnostic message directory is printed after the map and table.

SECOND PASS PROCESSING (IEWLMSCD)

After intermediate processing is completed, the third phase of Linkage Editor F (second pass processing) begins. (See Diagram D1.) The major functions of second pass processing include:

- Relocate address constants contained in the text.

- Create control/RLD records.

- Write TXT and control/RLD records onto SYSLMOD in a format that can be loaded by program fetch.

- Create ENTABs and associated RLD items for overlay modules.

Operation Diagram D1 illustrates the functions of second pass processing.

SINGLE-PASS PROCESSING: "In-core" indicators in the text I/O table and the RLD note list are checked to determine if text and RLD records have been written onto SYSUT1 or have been retained in the text buffer and the RLD buffer. If either text or RLD records have been retained in storage, single-pass processing is in effect for that record type. If two-pass processing is in effect, the records are read into the buffers from SYSUT1.

ORDERING OF TEXT: In two-pass processing, the ID sequence in the text I/O table is used to determine the order in which CSECTs are to be read into the second pass text buffer (which is physically the same storage area as the input text buffer). The text I/O table entry for each ID and the corresponding text note list entry are used to locate text on SYSUT1. (See Diagram D1, Area A.) Text is read into the buffer a multiplicity at a time, using the displacement field in the text note list to determine where within the buffer the text must be placed. Information about the text is entered into the second pass text control table, which is used to control subsequent processing of the text (area B).

SECOND PASS RLD BUFFERS: When the required text is in the text buffer, the corresponding RLDs are read into the RLD input buffer, using the RLD note list to locate the RLD records (area C). The RLD input buffer can contain two RLD records from SYSUT1; for each RLD input buffer area, an RLD input control block is maintained (area D). The RLD output buffer is 768 bytes long and is divided into three buffer areas (the maximum RLD output record is 256 bytes long); for each RLD output buffer area, an RLD output control block is maintained (area F). While text is being relocated, the control record for that portion of text occupies one of the output buffers; the other two output buffers contain the relocated RLDs for the text being processed (area E). If the relocated RLDs exceed two buffers, the control record is written onto SYSLMOD; relocated RLDs may then be moved into the third output buffer.

When all three RLD output buffers and the RLD input buffers are filled and additional RLDs are required to relocate the text currently being processed, the contents of the output buffer must be written out. However, to maintain the required TXT/RLD sequence in the output module (area G), the associated text must precede the RLD record. Space for the text is reserved in the output module by writing the incompletely relocated text; the contents of the RLD output buffer may then be written, and processing can continue. When the text is completely relocated, it is written over the space reserved for it, using XDAP ("execute direct-access program").

GROUPING SYSLMOD OUTPUT: As many CSECTs as will completely fit in one SYSLMOD record (up to a maximum of 60) are grouped and written as one record. RLDs are grouped to correspond to the grouping of their associated text. If the overlay option is specified, only CSECTs belonging to the same segment will be grouped.

If a CSECT is larger than the SYSLMOD record size, the CSECT is divided in multiplicities, each multiplicity being equal to the SYSLMOD record size. (The length of the last multiplicity may be less than the SYSLMOD record size.) Each multiplicity is written as a record, followed by RLDs associated with only that multiplicity.

Note: If the downward compatible option (DC) or the scatter format option (SCTR) is specified, CSECTs will not be grouped.

END OF MODULE: When control sections for all segments of the output module have been processed (determined via the "high ID" indicator in the HESD type field and the "last segment with text" field in the all purpose table), indicators are set in the last control/RLD record to mark it as the end of the module. The control/RLD record is written out on SYSLMOD, and second pass processing is completed.

Note: If the output load module is to be structured for overlay, a list of relative track addresses (TTR list) is created to be used by program fetch when it loads the segments into main storage for execution. The TTR list contains one entry for each segment in the overlay load module. Each entry contains the relative track address of the first record (control record) of a segment, except for the first segment, which contains the relative track address of the first text record. A PC-type control section, which contains ENTAB entries in each segment where the text requires them, and the RLD records required by program fetch to relocate address constants contained in the ENTABs, are also created.

RELOCATION OF ADDRESS CONSTANTS

There are two types of relocatable address constants:

1. Branch type, such as DC V(X).

2. Non-branch type, such as DC A(X).

The value of a branch type or non-branch type address constant depends on a symbol in the CESD. To adjust an address constant to its proper value in the output load module, the linkage editor uses an absolute or relative relocation factor. The absolute relocation factor is the address assigned by linkage editor to the symbol on which the value of the address constant depends. The relative relocation factor is the difference between the address assigned to the symbol by linkage editor and the address of the symbol in the input module.

The relative relocation factor may be positive or negative. The absolute and relative relocation factor of each symbol in the CESD is computed during address assignment and is saved in the half ESD (HESD).

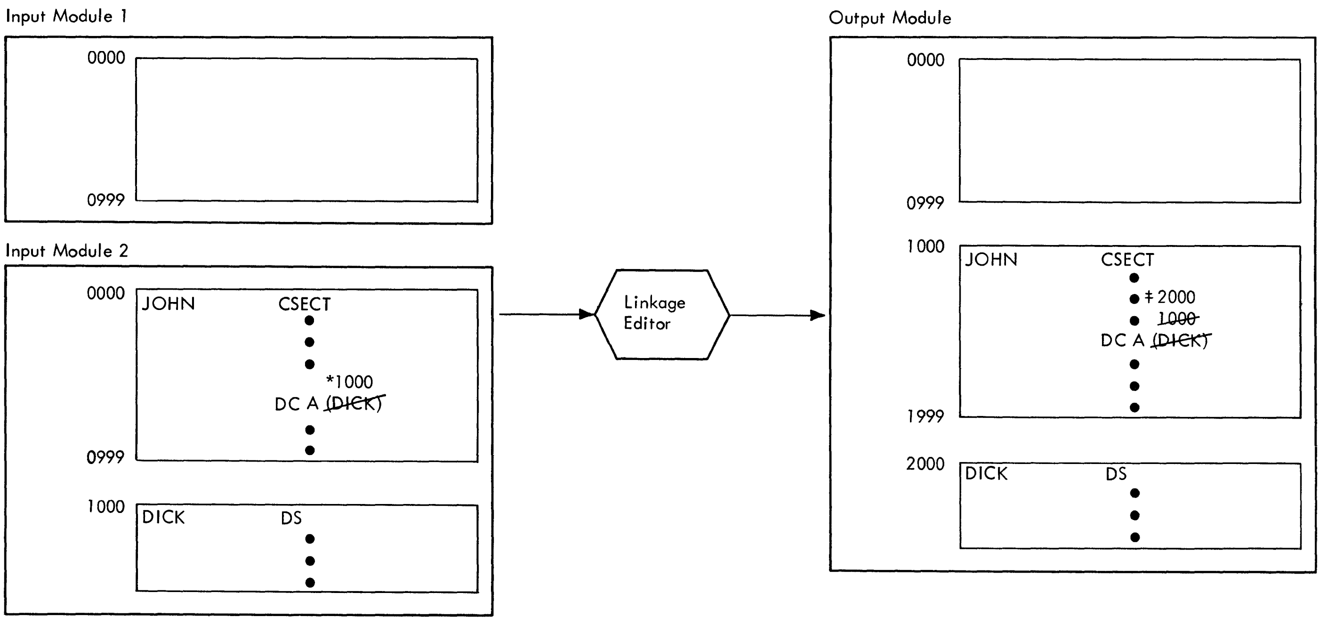
Relocation of Non-Branch Type (A-Type) Address Constants

A relative relocation factor is used for a non-branch type address constant if the symbol on which its value depends is in the same input module as the control section that contains the address constant. (The address constant and the symbol it refers to were assembled or compiled together, or were previously processed together by linkage editor.) An example of relative relocation of non-branch type address constants is shown in Figure 27. Since the address of DICK is known, the language translator places it in the value of the address constant. DICK is a known value prior to linkage editor processing (not an external reference in the input); therefore, a relative relocation factor (+1000) is used to relocate DICK during linkage editor processing.

An absolute relocation factor is used for a non-branch type address constant if the symbol referred to by the address constant does not have a defined value within the same input module. (The R pointer of the RLD item refers to an external reference.) An example of absolute relocation of a non-branch type address constant is shown in Figure 28. In this example, the value of SAM is unknown when input module 1 is processed by the language translator; therefore, zeros are placed in the value of the address constant. During second pass processing, the absolute relocation factor (the linkage-editor-assigned address) is used to relocate the address constant.

Figure 29 shows the use of both a relative relocation factor and an absolute relocation factor in relocating a symbol. Two input modules are to be processed by linkage editor. Input module 1 contains a non-branch type address constant whose value depends on the symbol PETE; PETE is an external reference in the same module. The language translator has assigned a value of +10 to the address constant. The R pointer of the RLD item refers to the ER entry for PETE in the ESD; this entry contains zeros in the origin and length fields. The P pointer refers to the SD entry for the control section that contains the address constant.

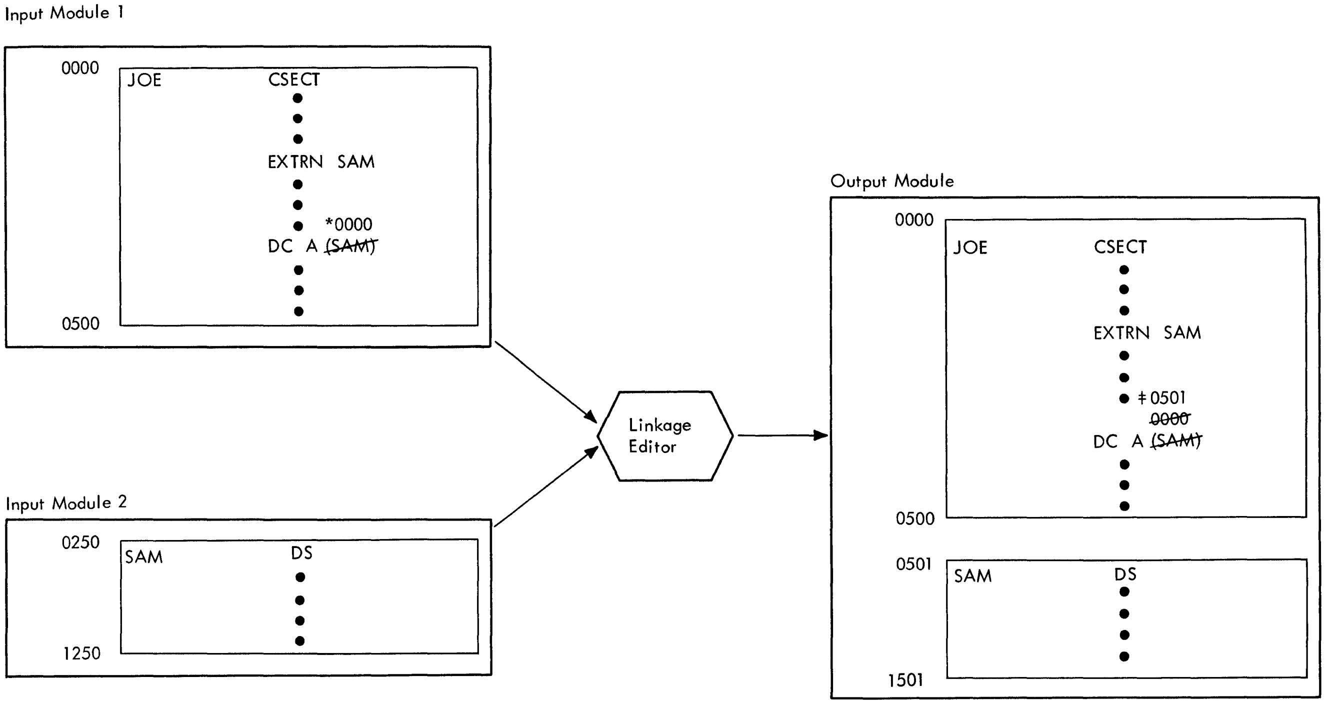
Input module 2 contains two control sections, BOB and PETE. BOB contains a non-branch type address constant whose value depends on PETE; since PETE has a defined value (300) in the same module, the language translator has used that value to



* Known value of DICK is inserted by language translator.

‡ Relative relocation factor is +1000.

Figure 27. Non-Branch Type Address Constants - Relative Relocation



* Language translator inserts zeros because value of SAM is unknown.

‡ Actual address of SAM in the output module (0501) is added to value of address constant. (Note that the relative relocation factor of SAM is +251.)

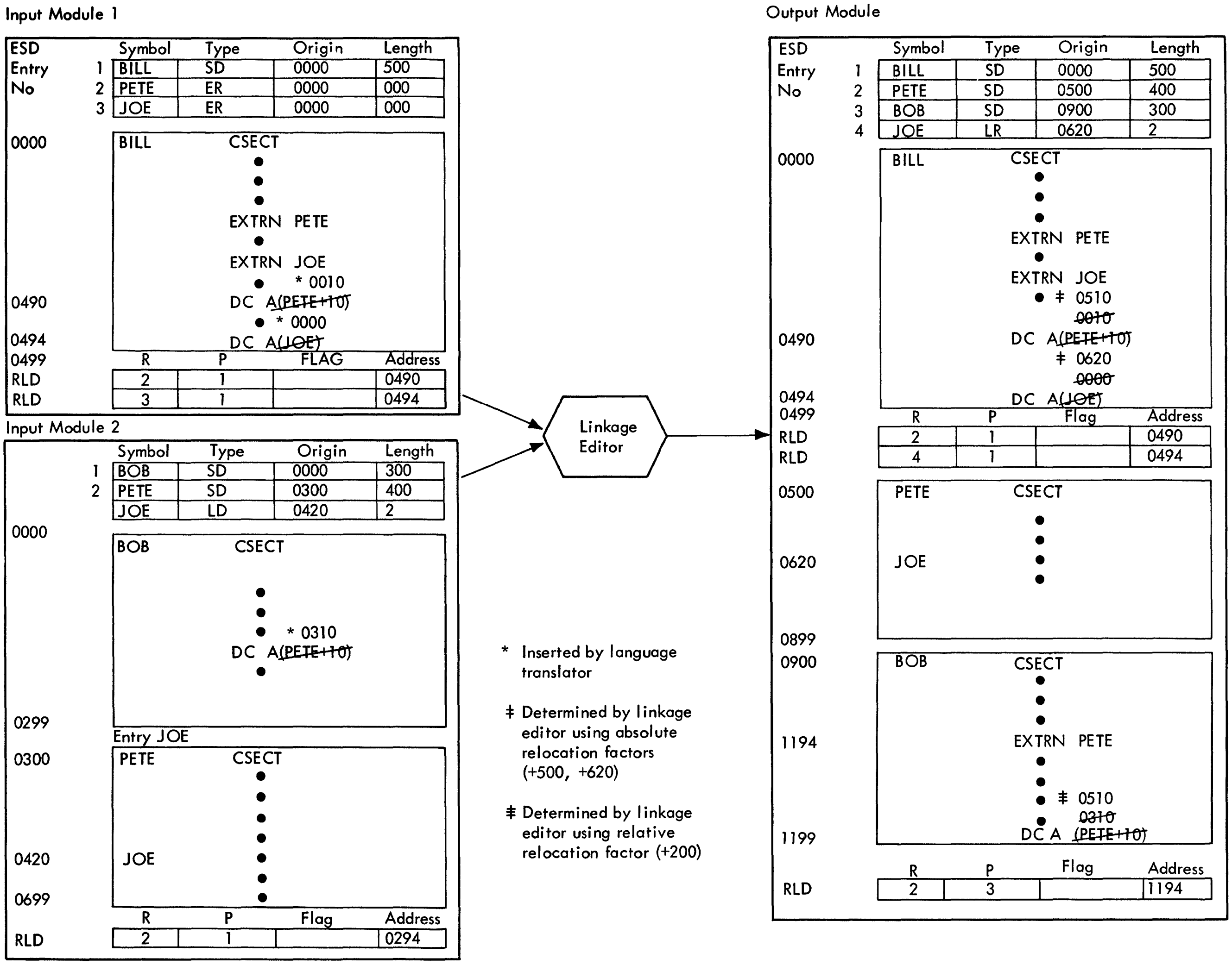
Figure 28. Non-Branch Type Address Constants - Absolute Relocation

compute the value of the address constant (PETE+10=310). The R pointer of the RLD item refers to the SD entry for PETE in the ESD; the P pointer refers to the SD entry for BOB (the control section that contains the address constant).

During linkage editor processing, the ER and SD entries for PETE are merged into one CESD entry; the R pointers of both RLD items in the output module will refer to that entry. The RLD P pointer for the address constant in control section BILL will refer to the SD entry for BILL; the P pointer for the other address constant will refer to the SD entry for BOB. In the output module, both address constants will contain the same value. Since the R pointer of the RLD item in input module 1 refers to an ER-type ESD entry in that module, it is marked for absolute relocation; the absolute relocation factor for PETE (+500) is added to the value (+10) assigned by the language translator. Since the R pointer of the RLD item in input module 2 refers to an SD-type ESD entry in module 2, it is marked for relative relocation; therefore, during relocation the relative relocation factor for PETE (+200) is added to the value (+310) assigned by the language translator. The relocated value for both address constants is 510.

Relocation of all non-branch type address constants requires an addition or subtraction of the relocation factor to or from the value of the address constant in the text of the input module. (Addition or subtraction is specified in the flag field of the RLD item for the address constant.)

DELINKING NON-BRANCH TYPE ADDRESS CONSTANTS: A relative relocation factor cannot be used to relocate an A-type address constant that refers to a symbol in a control section being replaced. Since the address constant has been previously relocated (by a language translator or by linkage editor), it contains the value of a symbol being replaced; therefore, the value of that symbol must be subtracted from the value of the address constant. This process is called delinking. In delinking, an address constant is reduced to the value it would have contained if it referred to an external reference in the input module. After delinking, the address constant contains the value required for proper relocation, should the replaced symbol appear later in the input, in another control section. Delinked address constants are treated like address constants whose values depend on external references. (Absolute relocation factors are used in relocating them.)



* Inserted by language translator

‡ Determined by linkage editor using absolute relocation factors (+500, +620)

‡ Determined by linkage editor using relative relocation factor (+200)

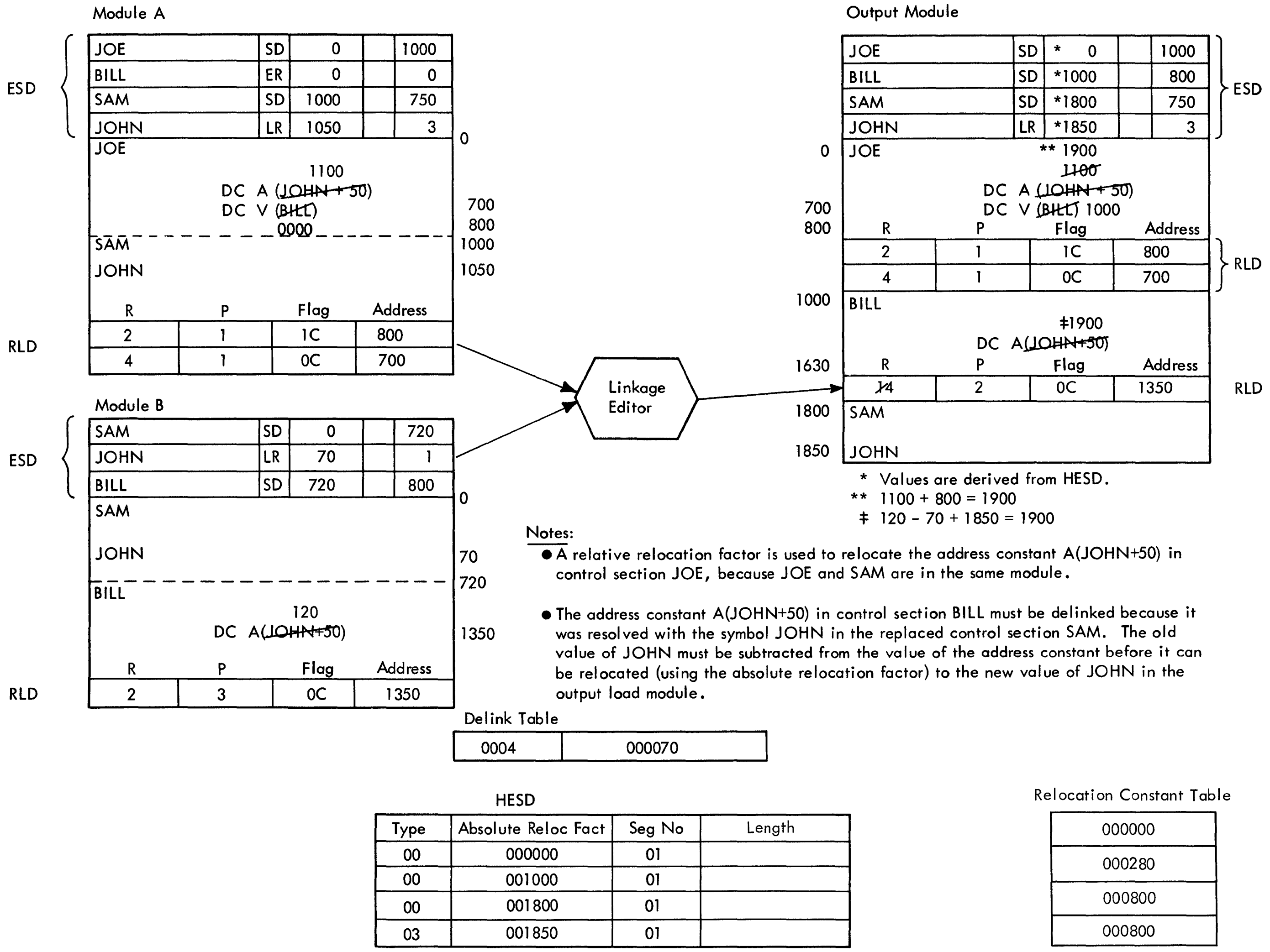
Figure 29. Non-Branch Type Address Constants - Absolute and Relative Relocation

Delinking of an A-type address constant is shown in Figure 30. Input load modules A and B both contain control section SAM. During linkage editor processing, the first occurrence of control section SAM is accepted, while the second occurrence is deleted through automatic control section replacement.

Control section BILL in module B contains a reference to symbol JOHN in control section SAM. Since SAM in module B will be deleted, the address constant A(JOHN+50) in module B must be delinked so that it may be properly resolved with the symbol JOHN in module A. In delinking, the old value of JOHN is subtracted from the value of the address constant in BILL (120-70=50). The absolute relocation factor for JOHN (1850) is then added to the delinked value of JOHN (50+1850=1900).

DELINKING COMMON CONTROL SECTIONS: Common control sections (either blank common or named common) must be "delinked" by linkage editor. All references to common control sections are made by means of non-branch type address constants.

If the assigned address of a common control section in the input to linkage editor is not zero, all such references must be delinked. Delinking is necessary because during linkage editor processing all blank common control sections are collected into a single control section. All identically named common control sections are gathered into individual control sections; references to them from different input modules must be delinked so that they can be properly relocated with respect to the locations of the common control sections in the output module.



Notes:

- A relative relocation factor is used to relocate the address constant A(JOHN+50) in control section JOE, because JOE and SAM are in the same module.
- The address constant A(JOHN+50) in control section BILL must be delinked because it was resolved with the symbol JOHN in the replaced control section SAM. The old value of JOHN must be subtracted from the value of the address constant before it can be relocated (using the absolute relocation factor) to the new value of JOHN in the output load module.

Delink Table

0004	000070

HESD

Type	Absolute Reloc Fact	Seg No	Length
00	000000	01	
00	001000	01	
00	001800	01	
03	001850	01	

Relocation Constant Table

000000
000280
000800
000800

Figure 30. Example of Delinking

Delinking adjusts the value of each address constant in a common control section so that it contains its correct displacement from the control section origin. The values of such address constants are then relocated so that they refer to linkage editor assigned addresses, using absolute relocation factors.

Relocation of Branch Type (V-Type) Address Constants

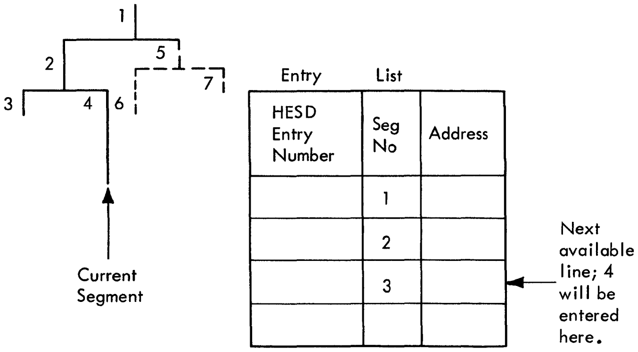
Only absolute relocation factors are used to relocate branch type address constants. Since a displacement is not allowed in the value of a V-type address constant, the absolute relocation factor is inserted in the value field during relocation. (It is not added to or subtracted from the value assigned by the language translator, as described for A-type address constants.) Because the value of a V-type address constant is inserted, delinking is never necessary for such address constants. Relocation of V-type address constants in an overlay structure is discussed in the following paragraph.

RELOCATION OF V-TYPE ADDRESS CONSTANTS IN OVERLAY: If the output of linkage editor is to be an overlay load module, a 4-byte[1] branch type address constant in the path of the symbol it refers to (but in a different segment), or in a different region, will be relocated in a special manner. The value

[1] Any address constant must be four bytes because the high-order byte is used by the overlay supervisor during execution. The number of the segment containing the address constant will be placed in the high-order byte of any V-type address constant resolved to an ENTAB entry. (The high-order byte must be zero if it is not resolved to ENTAB entry.)

field of the address constant will contain the address of an ENTAB entry. The ENTAB entry will contain the address assigned by linkage editor to the symbol referred to by the value of the address constant. An ENTAB entry is created for each V-type address constant that is in the path of the symbol it refers to (but is not in the same segment), or located in a different region, provided that the symbol is not referred to in a segment higher in the path of the calling segment. (Such address constants are resolved so that they refer to the ENTAB entry previously created for the symbol in the higher segment.) ENTAB entries are not created for address constants that refer to symbols higher in the path. Whenever an ENTAB entry is created, it is noted in an entry list; each item in the entry list contains the entry number of the referenced symbol in the HESD, the segment number of the calling segment, and the address assigned to the ENTAB entry by linkage editor. The ENTAB creation routine uses the entry list to build ENTAB entries. (Refer to "ENTAB Creation.")

When second pass processing begins to process a segment, the entry list is modified so that it contains only entries for segments higher in the path of the current segment. (In Figure 31 segment 4 is being processed; the entry for segment 3 is removed since it is not higher in the path of 4.)



HESD Entry Number	Seg No	Address
	1	
	2	
	3	

Next available line; 4 will be entered here.

Figure 31. Entry List Processing

During relocation, each V-type address constant is examined to determine if an ENTAB entry must be created for it. The R pointer of the RLD item for the address constant is used to find the associated HESD entry; this entry contains the segment number of the symbol referred to by the address constant. The relationship of this segment to the current segment is then determined, using SEGTA1. Depending on the relationship in SEGTA1, the address constant is relocated in one of three ways:

1. If the segment that contains the symbol is higher in the path of the current segment, the call is upward and the address constant is resolved directly. (The absolute relocation factor of the symbol is inserted in the value of the address constant.)

2. If the current segment is higher in the path of the segment that contains the symbol, the call is downward. The entry list is checked to determine if an ENTAB entry was previously created for the symbol in this segment, or in a segment higher in the path of this segment. If an ENTAB entry for the symbol exists, its address (contained in the entry list) is placed in the value field of the address constant. If no ENTAB entry exists for the symbol, a new entry is placed in the entry list, and an FNTAB entry will be created by the ENTAB creation routine. (Refer to "ENTAB Creation.") The ENTAB entry will contain the address assigned to the symbol by linkage editor, and the address of the ENTAB entry will be placed in the value of the address constant and in the entry list item.

3. If neither of the two segments is higher in the path of the other, the call is either exclusive or across regions. If the two segments are in different regions, and no ENTAB entry already exists for the symbol in the entry list, an ENTAB entry will be created and an entry is made in the entry list; the value field of the address constant is relocated to the address of the ENTAB entry, which in turn contains the relocated address of the symbol. If the two segments are in the same region, the call is exclusive. If there is an entry in the entry list for the symbol, the address constant is resolved through its ENTAB entry; if there is no entry for the symbol in the entry list, the call is an invalid exclusive call and the address constant is resolved directly to the symbol. (This usually leads to incorrect results during execution of the module.)

ENTAB Creation

The ENTAB creation routine uses the size field in the HESD to determine the number of ENTAB entries to be created for a given segment. The entry list is scanned for all entries that were created for the current segment; each of these entries contains the HESD entry number for the corresponding symbol. The value and segment number of the symbol are obtained from the HESD and are entered into the ENTAB entry, along with standard information shown in the Appendix.

ENTAB creation is shown in Figure 32. The V-type address constants referring to SAM and BILL in segment 1 meet the requirements for building ENTAB entries. The ESD and RLD input to the second pass processor, and the overlay tree structure are shown in diagram A. During relocation, entries are created for SAM and BILL in the entry list (see diagram B); each entry contains the address of the ENTAB entry created for the address constant.

In segment 1, location 136 of control section JOE contained a call to control section SAM before relocation. After relocation, location 136 contains the address of the ENTAB entry for SAM, and the high-order byte of the address constant contains the segment number of the calling segment. An ENTAB entry is created, in like manner, for BILL in segment 1.

In segment 2, the address constant referring to BILL does not meet the requirements for building an ENTAB entry. (It is not in the path of the segment containing the symbol.) Therefore, no ENTAB is created in segment 2. The call for segment 2 to BILL in segment 3 is an exclusive call. Since a call to the same symbol appears in a higher segment common to 2 and 3 (segment 1) the address constant may refer to the ENTAB entry for BILL in segment 1. (This is determined by scanning the entry list for the HESD entry corresponding to the symbol BILL.) If a call to BILL was not contained in a common segment, the address constant DC V(BILL) in segment 2 would be resolved using the value assigned by linkage editor to the symbol BILL, which results in an error.

In segment 3, the address constant is an upward call and is resolved directly.

Relocation Routine

The relocation of address constants is performed by the relocation routine; the routine operates on the following input data:

- The address of the RLD input buffers which contain RLD records.
- The address of the RLD notelist entry for the RLDs being processed.
- The address of the next available entry in the RLD output buffer.
- The buffer relocation constant (BRC) where:

  BRC = starting buffer address of current text + relative relocation constant of current control section - address assigned to current control section by linkage editor - multiplicity size X current multiplicity number

The relocation routine operates in the following manner:

1. The size of the RLD set[1] and the displacement from the beginning of the buffer is determined from the RLD note list.
2. Each RLD item in the current RLD set is scanned to determine if:
   a. It describes an address constant for the current text being processed (BRC + address contained in RLD address field falls within the text buffer boundaries of the current text.)
   b. The address constant is either a valid 2-, 3-, or 4-byte address constant. (The only valid 2-byte address constants are pseudo register type.)
3. Each address constant whose RLD meets the above requirements is moved from the text into a computation area. The address constant associated with the RLD item is then relocated according to the information in the flag field of the RLD item (refer to Table 7). The relocated address constant is then placed back into the text.
4. The RLD address field is updated using the relative relocation factor for the control section being processed. (The control section referred to by the P pointer of the RLD item).
5. The RLD is moved into the RLD output buffer if space is available. If space is not available, the contents of the RLD output buffer are written out on SYSLMOD.[2]
6. Steps 2 through 5 are repeated until all RLD items have been scanned in the RLD set being processed. The multiplicity number in the RLD notelist is updated if unprocessed RLDs remain in the set.
7. If there are more RLD sets in the input buffer to be processed, the address of the next record is determined and steps 1 through 6 are performed.

[1] An RLD set is a group of RLDs referred to by a particular RLD notelist entry.

[2] If the XDAP indicator is off, a dummy text record is written out before the contents of the RLD output buffer are placed on SYSLMOD. If the XDAP indicator is on, a dummy write of the text record is not required, because text is already written.

Diagram A.

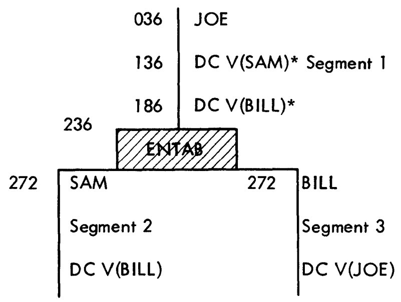
HESD

	Type	L.E. Assigned Address	Seg	Length
JOE	SD	36	1	
SAM	SD	272	2	
BILL	SD	272	3	
SEGTAB	PC	0	1	
ENTAB	PC	236	1	

Relocation Constant Table
200
500
500
36
36

	R	P	Flag	Address
RLD	2	1	1C	100
	3	1	1C	150

Input RLDs - Segment 1



Structure with V-type address Constants.

* Zero value assigned by the assembler.

Diagram B.

Output RLD Buffer

2	1	1C	136
3	1	1C	186

Entry List

2	1	236
3	1	248

Entab RLD Items

0	1	1D	240
		1D	252

RLDs and Entry List after relocation for control section JOE.

Diagram C.

Segment 1 after processing by Second Pass Processor.

	JOE				
	01000236				
136	DC V(~~SAM~~)				
	01000248				
186	DC V(~~BILL~~)				
236	47FF 0024	00000 272	02	000000	ENTAB
248	47FF 0012	00000 272	03	000000	
260	Standard Last ENTAB Entry				

Diagram D.

Segment 2 after processing by Second Pass Processor.

272	SAM
	02000248
752	DC V(~~BILL~~)

Input RLD Buffer

3	2	1C	680

Output RLD Buffer

3	2	1C	752

ENTAB RLD Items

None

Entry List

*

* Same as after processing segment 1.

Diagram E.

Segment 3 after Second Pass Processing

BILL

00000036
DC V(~~JOE~~)

Input RLD Buffer

1	3	1C	690

Output RLD Buffer

1	3	1C	762

ENTAB RLD Items

None

Entry List

*

* Same as after processing segment 1

Figure 32. ENTAB Creation

Table 7. Relationship of RLD Flag Field to Relocation

Input		Action Performed	Output	
Flag	Type		Flag	Type
0000LLST	Absolute	Absolute relocation factor is added to value of address constant	0000LLST	A-type
0001LLST	Branch	Absolute relocation factor is inserted into value of address constant	0001LLST	V-type
0010LLST	PR-displacement value (PR type 1)	Absolute relocation factor is inserted into value of address constant	0010LLST	PR-displacement value
0011LLST	PR-cumulative displacement value (PR type 2)	PR length from All Purpose Table is inserted into value of address constant	0011LLST	PR-cumulative displacement value
1000LLST	Relative	Relative relocation factor is added to value of address constant	0000LLST	A-type

Notes:

- If S (sign) in LLST is 1, subtraction is performed, rather than addition.
- In delink type, the delink value is added or subtracted according to the opposite of the sign; the absolute relocation factor is added to or subtracted from the address constant according to the indicated sign.
- If an RLD item refers to an undefined symbol, the associated address constant is not relocated. (It may have been delinked.) The high-order bit of the RLD item flag field is set to one (1000LLST for an A-type constant, 1001LLST for a V-type constant) and no relocation will be performed when the module is loaded into main storage for execution.
- Delinking is noted in the high-order bit of the P pointer.

Note: In order to minimize the number of times that RLD records are read from SYSUT1, RLD records for a control section are held in the input RLD buffer, when possible, until all RLD records in the buffer have been processed (because each RLD record may pertain to many multiplicities of text). After each set of RLDs is scanned, the multiplicity number in the RLD note list is updated to reflect the multiplicity of the remaining unprocessed RLD records in the set. An RLD record is removed from the buffer when:

1. All RLD items in the record have been processed. (Their associated address constants have been relocated.)

2. Another RLD record must be read into the buffer and space is not available.

When all records in the input RLD buffer have been scanned, the relocation routine determines if more RLD records for the current multiplicity of text are to be read in. (The read RLD routine sets an indicator when it encounters such a record but cannot read it into the buffer because the buffer is full.) When both buffers are full, the second buffer is freed, and the corresponding RLD note list entries are marked "out-of-core." The records to be read in are then placed in the second RLD buffer; these records are processed in the same manner as those already residing in the first buffer. This process is repeated until all records that contain RLD items pertaining to the current multiplicity of text have been scanned and processed.

When all RLDs in a buffer are processed, the buffer is marked "free" in the RLD control block. When a new multiplicity of text is to be relocated, the RLD note list is scanned sequentially (on ID and multiplicity number) from the first entry. If an entry indicates that the record is "in core" and the record contains RLD items pertaining to the new multiplicity of text, it is processed.

FINAL PROCESSING (IEWLMFNL)

The fourth phase of Linkage Editor F (final processing) performs "cleanup" functions, and is the last operation of linkage editor processing. Functions of final processing include:

- Write the TTR note list, created during second pass processing, on SYSLMOD if the output load module is to be used in overlay. The TTR list contains the relative track address of the first record of each segment of the overlay load module. It is used by program fetch to find the segments when it loads them into main storage for execution.

- Place each entry in the proper format for the partitioned data set directory, modify it if there are alias symbols, and issue a STOW macro instruction[1] for the member name and each alias.

- Check attributes (reusable, reentrant, and refreshable). If the attributes have become more restrictive, a message describing the change in attributes is printed out. (For example, the input module was specified as "reusable" and is now "not reusable.")

- Print out a directory of logged errors.

- Produce a cross-reference table if the XREF option is specified, and the cross-reference table was not produced during intermediate output processing.

- If the module has been marked "not executable," an error message is printed out.

- If a NAME card, not followed by end of file, terminated SYSLIN input, linkage editor processing is repeated, beginning with initialization.

- If end of file terminated SYSLIN input, linkage editor processing is completed.

Allocated main storage is released, and control is returned to the caller.

Error Logging

Whenever an error condition is detected during linkage editor processing, an indicator is set in an error logging map and a coded diagnostic message is printed out. During final processing, the error logging map is scanned. When an indicator is found "on" in the map, an associated list is used to build a diagnostic message.

Note: An example of error logging in level F is given in Figure 33. Each entry in the list contains a length indicator and a pointer to a phrase to be assembled into the message. (Phrases are stored to save main storage space; complete messages would require additional space due to repetition of identical phrases.) The diagnostic directory is then printed out, one or two lines to a message.

All error messages produced by the linkage editor are identified by a message ID having the format:

IEWDMMS

where:

IEW - identifies the message as a linkage editor error message.

D - contains a zero.

MM - is the message number.

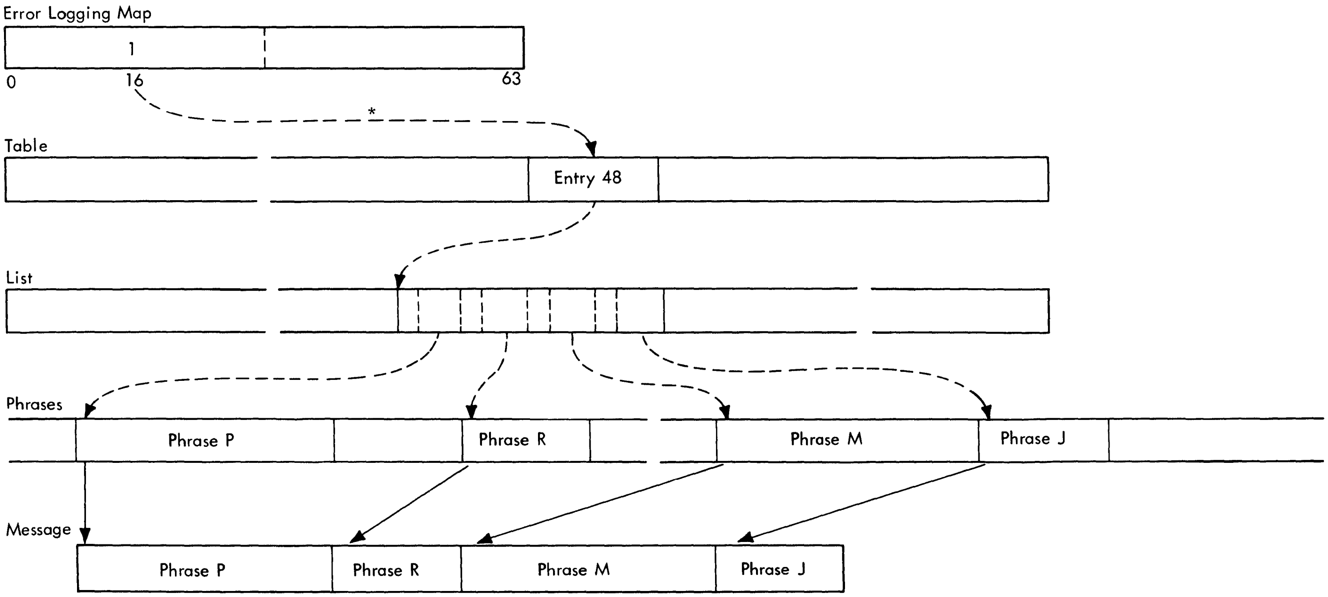
S - is the severity code.

The module in which an error message occurred is identified by the message number (MM). (Refer to Section 6 for an error message-module cross reference table.)

Cross-Reference Table

If the XREF option is specified, and the cross-reference table was not produced during intermediate output processing, the RLD records are read back from SYSLMOD, and the cross-reference table is built, as described in the discussion of intermediate processing.

[1] The STOW macro instruction is not issued if there was no valid input, if there were no ESDs, if nothing was written out on SYSLMOD, or if the run was terminated by a severity 4 error.



* This pointer is determined by subtracting the bit number from the length of the error logging map (64 - 16 = 48).

Figure 33. Building Error Messages

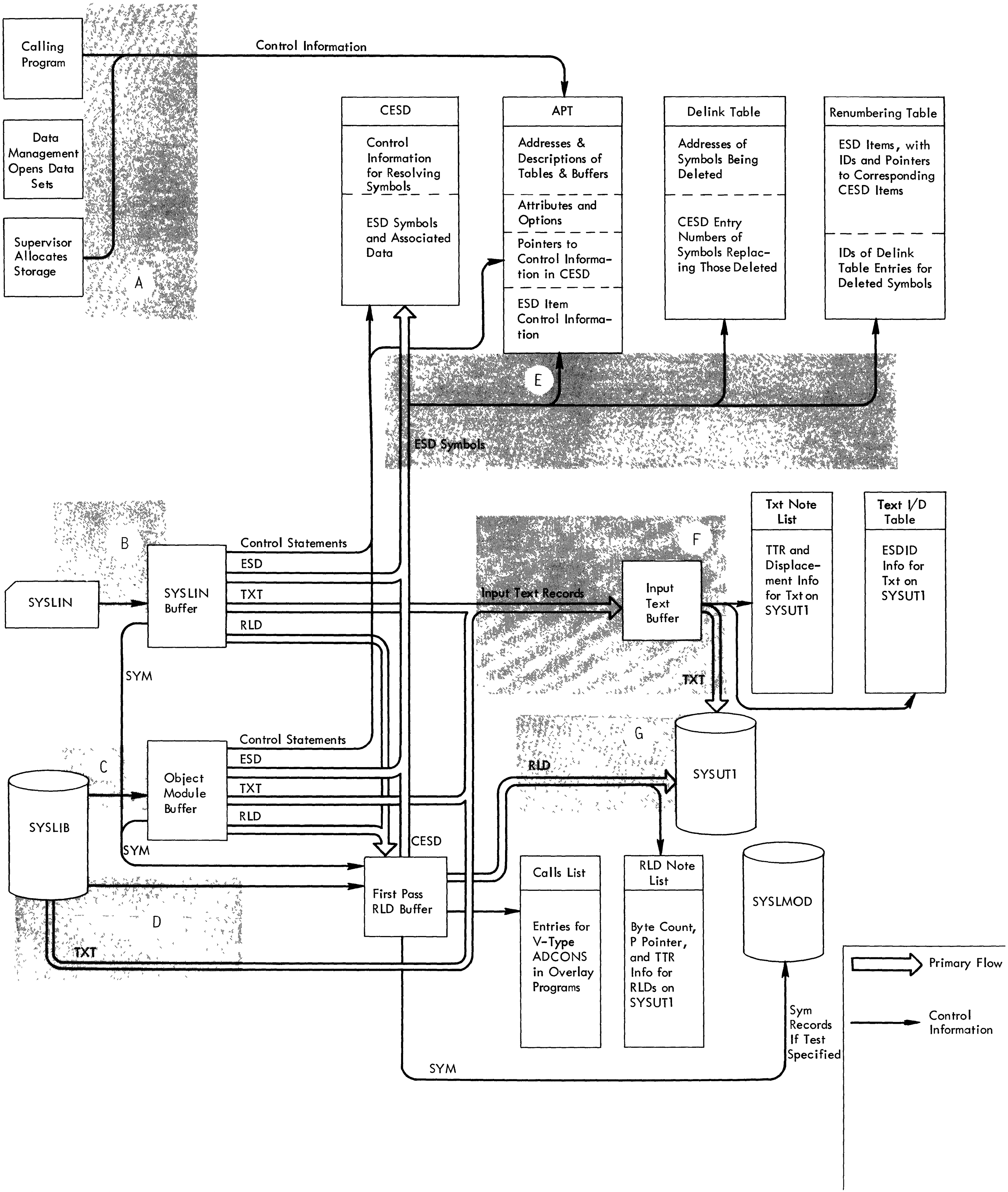

Calling Program
Data Management Opens Data Sets
Supervisor Allocates Storage
A
Control Information
CESD
Control Information for Resolving Symbols
ESD Symbols and Associated Data
APT
Addresses & Descriptions of Tables & Buffers
Attributes and Options
Pointers to Control Information in CESD
ESD Item Control Information
Delink Table
Addresses of Symbols Being Deleted
CESD Entry Numbers of Symbols Replacing Those Deleted
Renumbering Table
ESD Items, with IDs and Pointers to Corresponding CESD Items
IDs of Delink Table Entries for Deleted Symbols
E
ESD Symbols
B
SYSLIN
SYSLIN Buffer
Control Statements
ESD
TXT
RLD
SYM
Input Text Records
F
Input Text Buffer
TXT
Txt Note List
TTR and Displacement Info for Txt on SYSUT1
Text I/D Table
ESDID Info for Txt on SYSUT1
C
SYSLIB
Object Module Buffer
Control Statements
ESD
TXT
RLD
SYM
CESD
G
RLD
SYSUT1
First Pass RLD Buffer
D
TXT
Calls List
Entries for V-Type ADCONS in Overlay Programs
RLD Note List
Byte Count, P Pointer, and TTR Info for RLDs on SYSUT1
SYSLMOD
Sym Records If Test Specified
SYM
Primary Flow
Control Information

Figure 6. Operation Diagram A1 - Initial and Input Processing

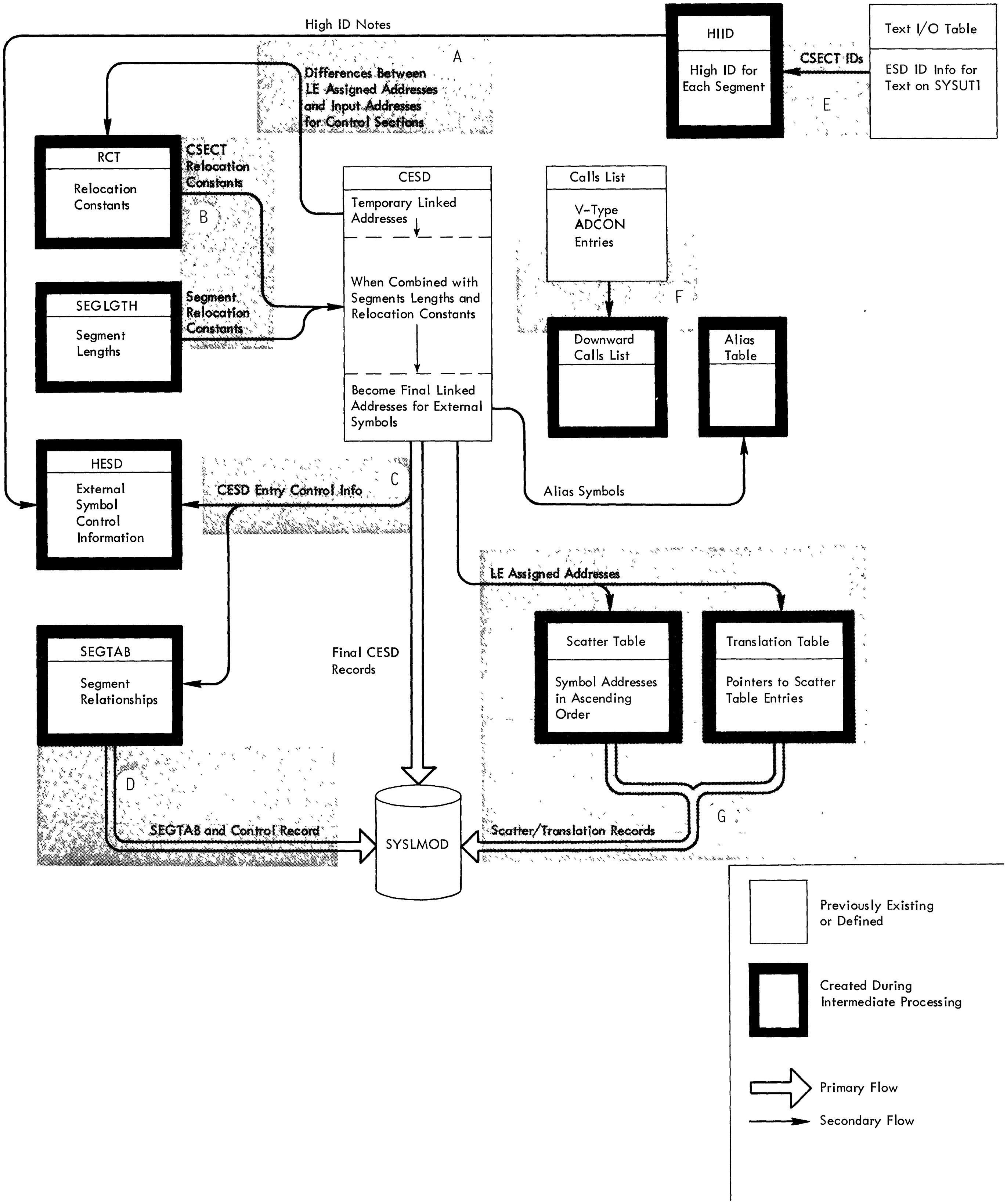

High ID Notes
HIID
High ID for Each Segment
Text I/O Table
ESD ID Info for Text on SYSUT1
CSECT IDs
E
A
Differences Between LE Assigned Addresses and Input Addresses for Control Sections
RCT
Relocation Constants
CSECT Relocation Constants
B
SEGLGTH
Segment Lengths
Segment Relocation Constants
CESD
Temporary Linked Addresses
When Combined with Segments Lengths and Relocation Constants
Become Final Linked Addresses for External Symbols
Calls List
V-Type ADCON Entries
F
Downward Calls List
Alias Table
Alias Symbols
HESD
External Symbol Control Information
CESD Entry Control Info
C
SEGTAB
Segment Relationships
Final CESD Records
LE Assigned Addresses
Scatter Table
Symbol Addresses in Ascending Order
Translation Table
Pointers to Scatter Table Entries
D
SEGTAB and Control Record
SYSLMOD
Scatter/Translation Records
G
Previously Existing or Defined
Created During Intermediate Processing
Primary Flow
Secondary Flow

Figure 7. Operation Diagram A2 - Intermediate Processing

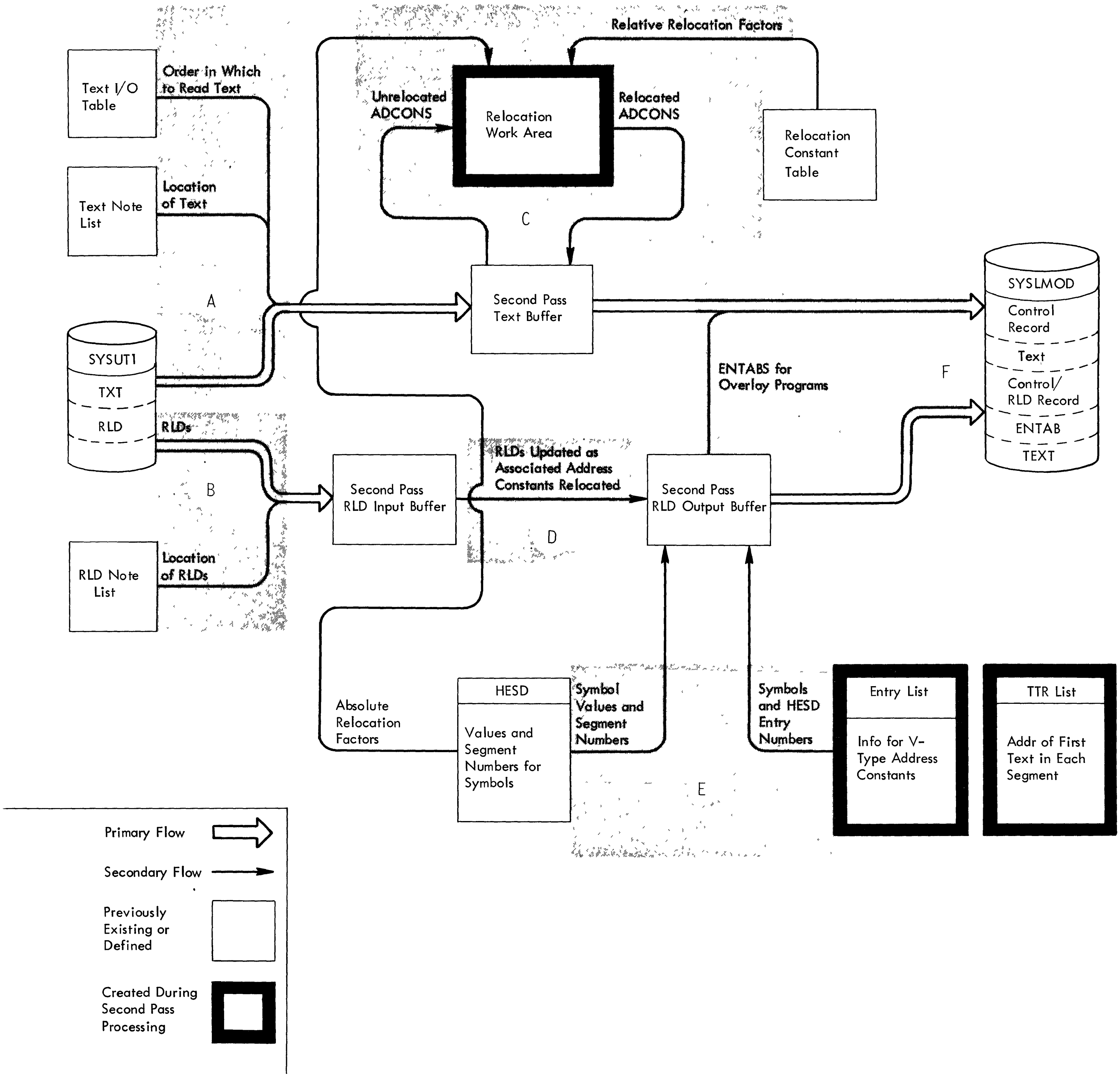

Relative Relocation Factors
Text I/O Table
Order in Which to Read Text
Unrelocated ADCONS
Relocation Work Area
Relocated ADCONS
Relocation Constant Table
Text Note List
Location of Text
C
A
Second Pass Text Buffer
SYSLMOD
Control Record
Text
Control/ RLD Record
ENTAB
TEXT
SYSUT1
TXT
RLD
RLDs
ENTABS for Overlay Programs
F
B
Second Pass RLD Input Buffer
RLDs Updated as Associated Address Constants Relocated
Second Pass RLD Output Buffer
D
RLD Note List
Location of RLDs
Absolute Relocation Factors
HESD
Values and Segment Numbers for Symbols
Symbol Values and Segment Numbers
Symbols and HESD Entry Numbers
Entry List
Info for V-Type Address Constants
TTR List
Addr of First Text in Each Segment
E
Primary Flow
Secondary Flow
Previously Existing or Defined
Created During Second Pass Processing

Figure 8. Operation Diagram A3 - Second Pass Processing

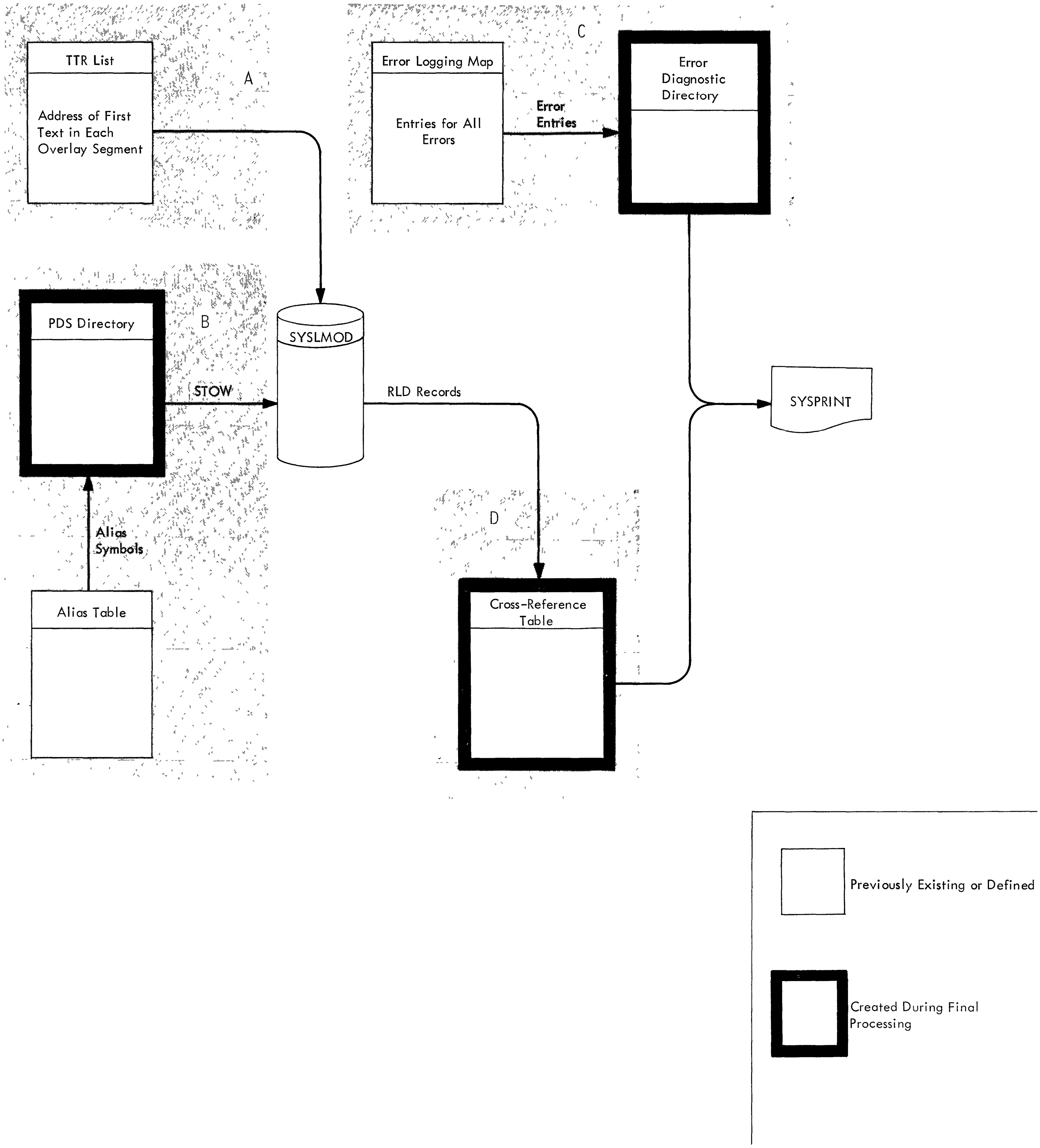
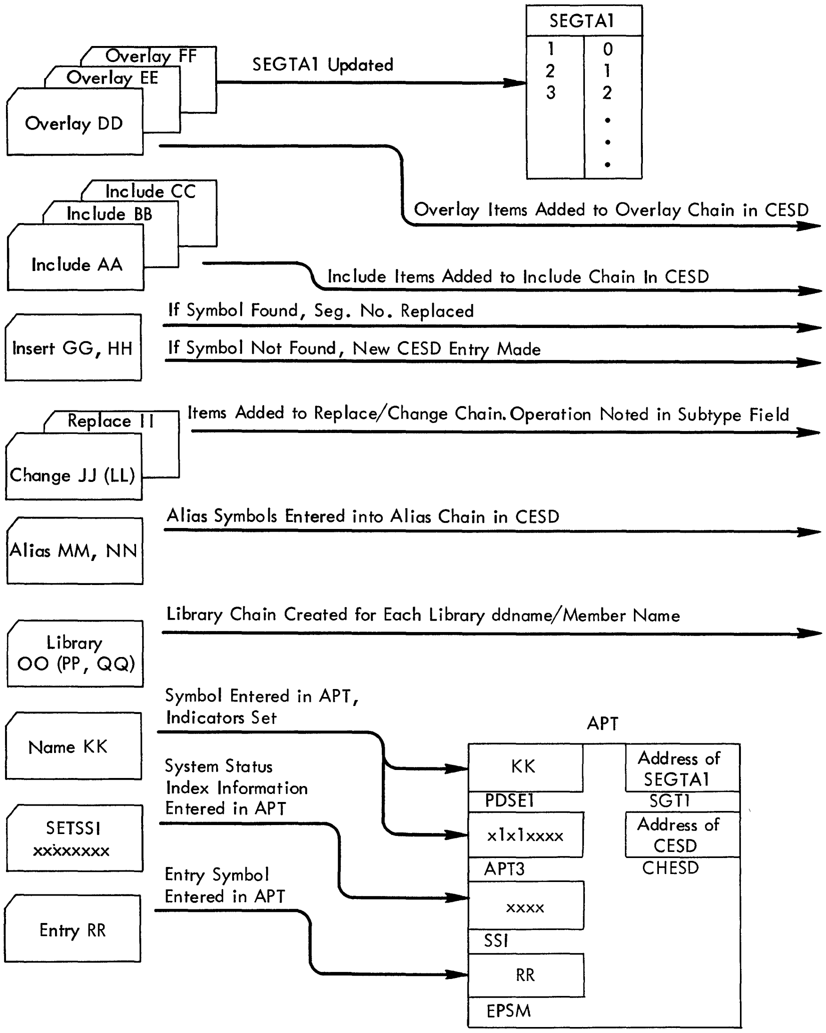


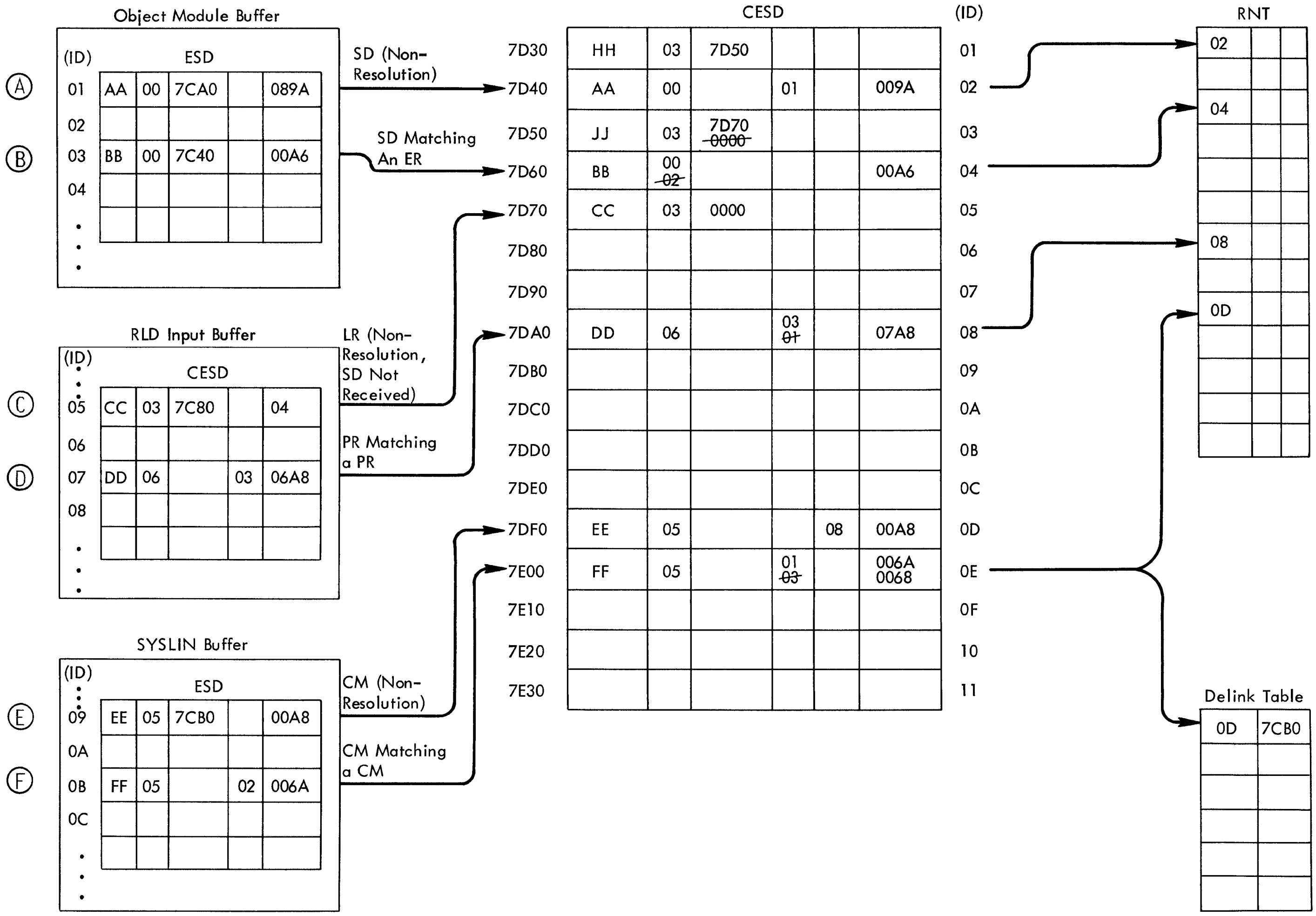
Figure 9. Operation Diagram A4 - Final Processing


Overlay FF
Overlay EE
Overlay DD
SEGTA1 Updated
SEGTA1
1 0
2 1
3 2
•
•
•
Overlay Items Added to Overlay Chain in CESD
Include CC
Include BB
Include AA
Include Items Added to Include Chain In CESD
Insert GG, HH
If Symbol Found, Seg. No. Replaced
If Symbol Not Found, New CESD Entry Made
Replace II
Change JJ (LL)
Items Added to Replace/Change Chain. Operation Noted in Subtype Field
Alias MM, NN
Alias Symbols Entered into Alias Chain in CESD
Library
OO (PP, QQ)
Library Chain Created for Each Library ddname/Member Name
Name KK
Symbol Entered in APT, Indicators Set
SETSSI
xxxxxxxx
System Status Index Information Entered in APT
Entry RR
Entry Symbol Entered in APT
APT
KK
PDSE1
x1x1xxxx
APT3
xxxx
SSI
RR
EPSM
Address of SEGTA1
SGT1
Address of CESD
CHESD

CESD

	Symbol	Type	Chain Addr/ Reverse Chain ID	Seg No.	Sub Type	Chain Pointer/ Chain ID/ Length
7C40	EE	02	7C60	02	90	
7C50	DD	02	7C40	01	90	
7C60	FF	02	0000	03	90	
7C70	AA	02	7CA0		D0	
7C80	GG	02	0000	06 ~~04~~	90	
7C90	HH	02	0000	06	90	
7CA0	BB	02	7CD0		D0	
7CB0	JJ	00	7CF0		08	
7CC0	LL	00	0000		F0	
7CD0	CC	02	0000		D0	
7CE0	MM	02	7D00		A0	
7CF0	II	02	0000		08	
7D00	NN	02	0000		A0	
7D10	OO	02	0000		B0	7D30
7D20						
7D30	PP	02	7D10		02	7D50
7D40						
7D50	QQ	02	7D30		02	0000
7D60						
7D70						
7D80						
7D90						
7DA0						

Figure 10. Operation Diagram B1 - Control Statement Processing



- The type of each input ESD item is determined
- The CESD is scanned for a matching symbol
- If no match is found, non-resolution processing is performed (A,C,E)
- If a match is found, resolution processing is performed (B,D,F)

Figure 16. Operation Diagram B2 - ESD Processing

- Text record IDs are renumbered (A)
- CSECT lengths obtained (B)
- Assuming there is space in TXTBUF1, text records are moved (C)
- Entries made in Text I/O Table and Text Note List (D)
- Contents of TXTBUF1 written onto SYSUT1 (E)
- TTR entered into Text Note List (F)

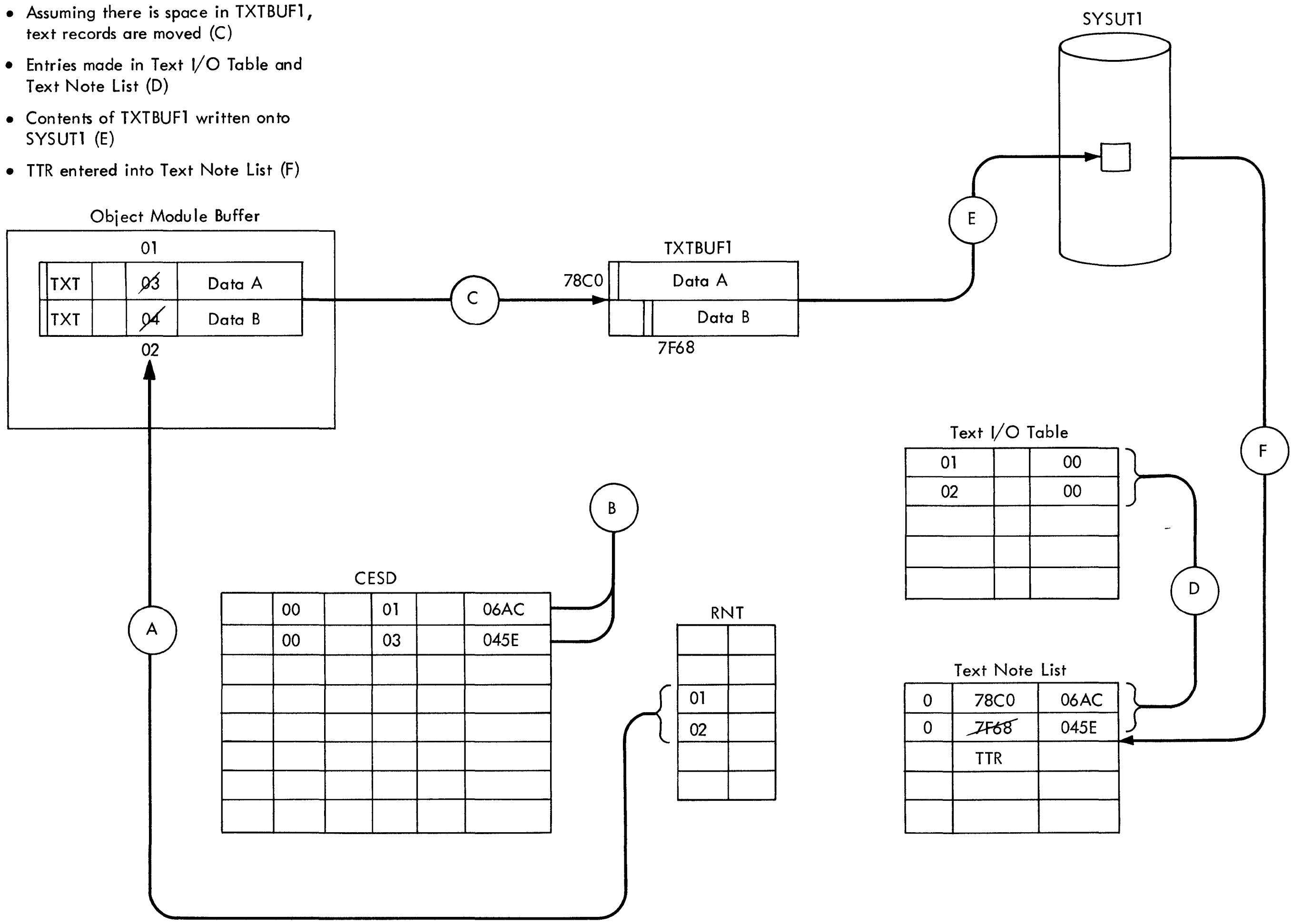
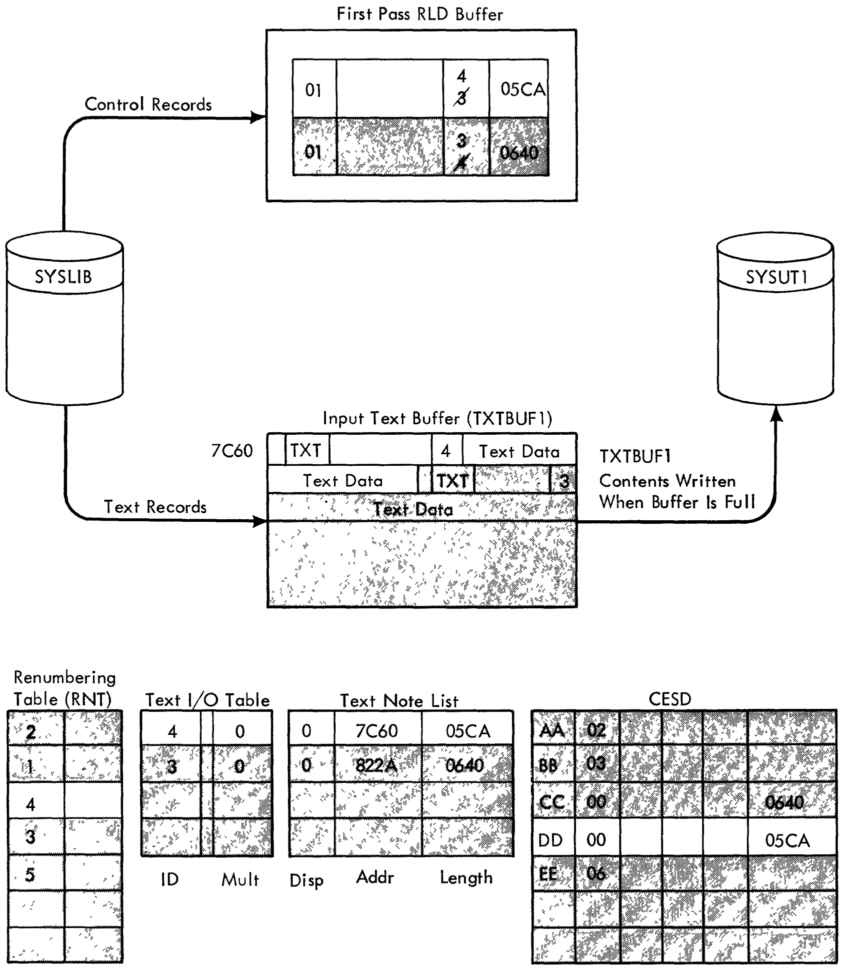


Figure 17. Operation Diagram B3 - Processing Object Module Text



Renumbering Table (RNT)

2	
1	
4	
3	
5	

Text I/O Table

ID	Mult
4	0
3	0

Text Note List

Disp	Addr	Length
0	7C60	05CA
0	822A	0640

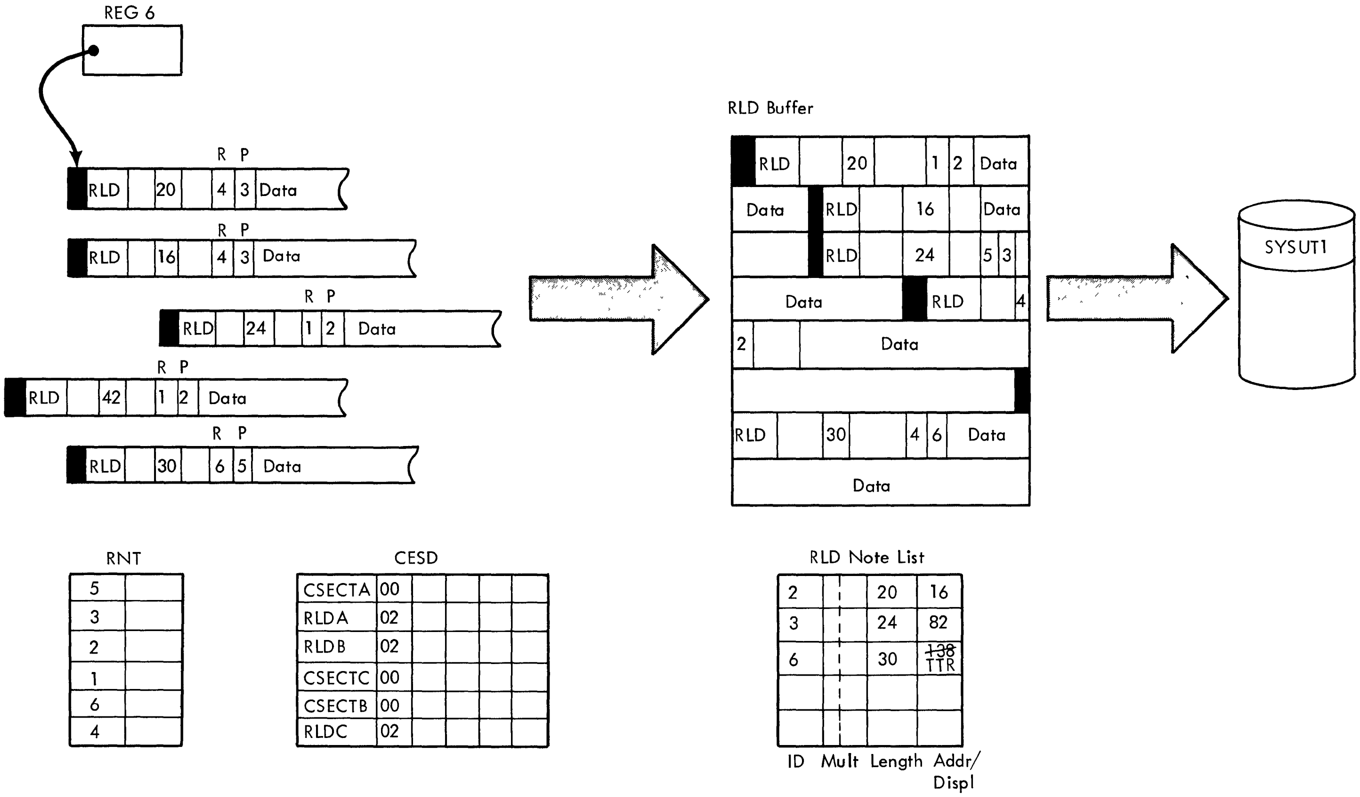
CESD

AA	02			
BB	03			
CC	00			0640
DD	00			05CA
EE	06			

Figure 18. Operation Diagram B4 - Processing Load Module Text Records

- The ID in the first control record is renumbered. The third line of the RNT contains a 4, so the ID is renumbered to refer to the fourth line of the CESD (CSECT DD).
- Assuming CSECT DD (CESD ID = 4) is not to be deleted, its length (in the control record) is checked.
- If the entire CSECT or a complete multiplicity will fit in TXTBUF1, the record containing text for DD is read into TXTBUF1, and entries are made in the text I/O table and the text note list*.
- Each subsequent control record is processed. Text records are read into TXTBUF1 until it becomes full, at which time its contents are written onto SYSUT1.

* In the two text records in this example, the multiplicity number is 0, since they are the first text records for their respective control sections.



RNT

5	
3	
2	
1	
6	
4	

CESD

CSECTA	00				
RLDA	02				
RLDB	02				
CSECTC	00				
CSECTB	00				
RLDC	02				

RLD Note List

ID	Mult	Length	Addr/ Displ
2		20	16
3		24	82
6		30	~~138~~ TTR

- Register 6 initially points to the first RLD input record.
- RLD records are grouped in the RLD buffer by P pointer. In this example, the first and second, and third and fourth RLD records are grouped.
- R and P pointers are renumbered, using the renumbering table, as RLD records are moved into the buffer.
- Entries for each RLD set are made in the RLD notelist. Length and displacement fields refer to the first record of the set.
- When the contents of the RLD buffer are written, the displacement field of the RLD note list entry for the last set included in the output record is replaced by the relative track address (TTR) of the SYSUT1 record.

Figure 19. Operation Diagram B5 - RLD Processing

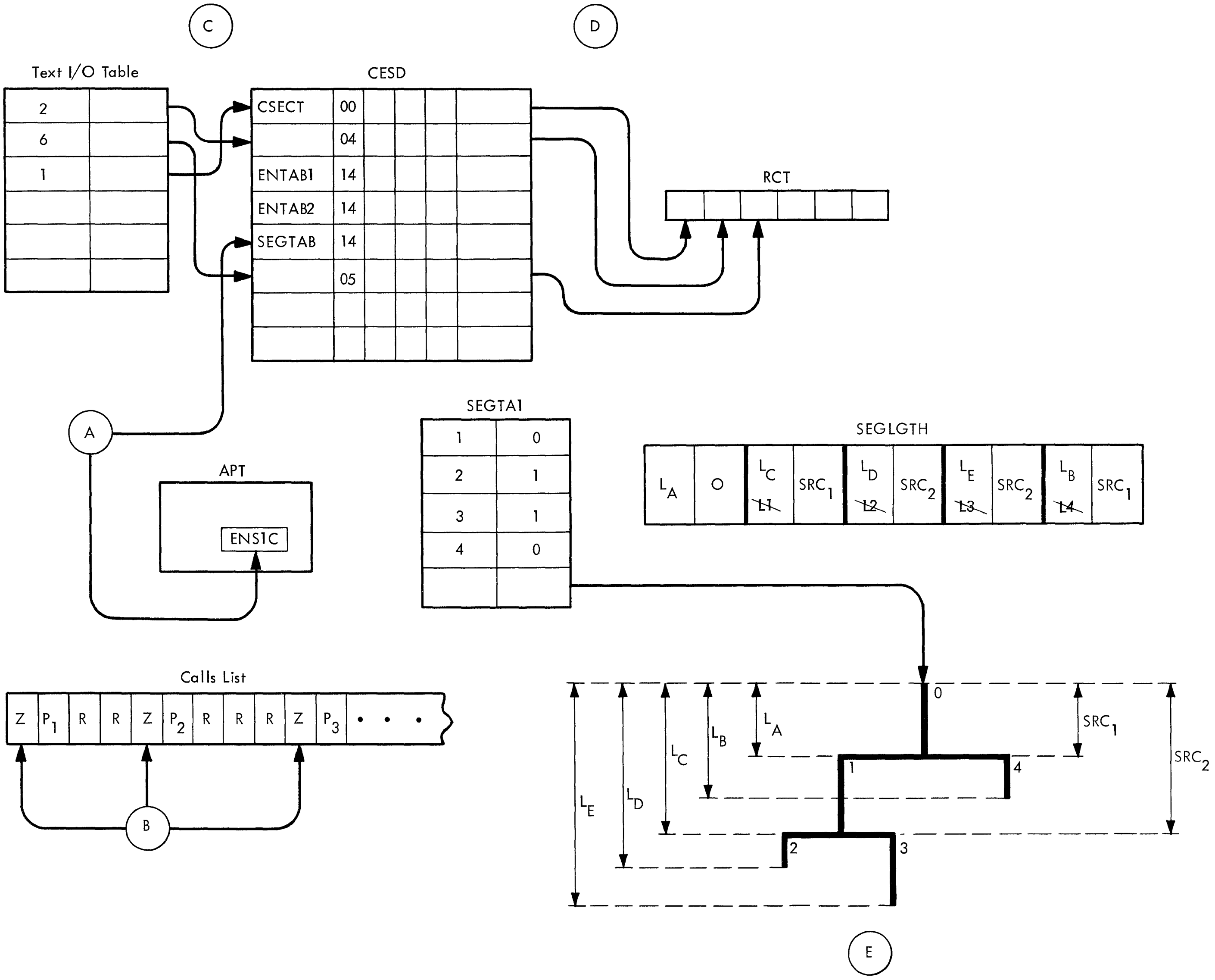


Figure 22. Operation Diagram C1 - Address Assignment

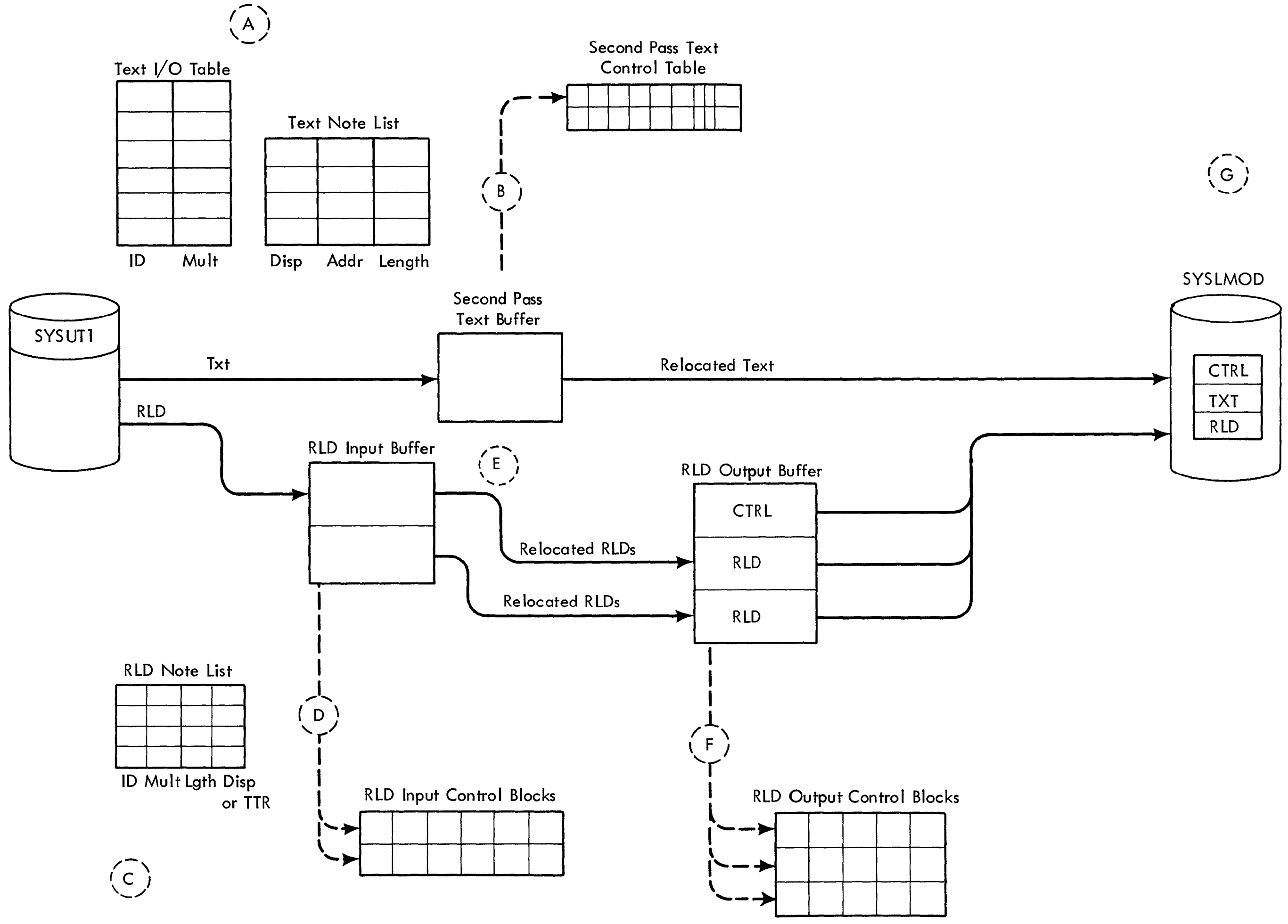


Figure 26. Operation Diagram D1 - Data Movement During Second Pass Processing

The following text and the flowcharts at the end of this section describe the processors (code modules, control sections, and routines) that accomplish the functions of Linkage Editor F. The organization of this section corresponds to the organization of the linkage editor; descriptions of all processors which constitute a phase of the linkage editor are grouped together. For each processor the symbolic name is given to facilitate use of program listings (see "Section 4: Michofiche Directory") and the descriptive name is given to facilitate reference to the Method of Operation" section (Section 2).

Figure 34 (a foldout) shows the overall organization of Linkage Editor F; this figure is designed to help determine relationships among the processors described in this section.

Refer to the microfiche directory (Section 4) for the chart numbers associated with each module.

INITIALIZATION AND INPUT PROCESSING

Initial Processor -- IEWLMINT (Chart IA)

Entrance: IEWLMINT is entered from IEWLMROU at the beginning of linkage editor processing.

Operation: IEWLMINT performs initialization functions, including: building the all purpose table (APT), opening data sets, analyzing attributes and options passed by the calling program, and allocating main storage for internal tables, buffers, and work areas.

Routines Called: IEWLMINT calls the attributes and options processor (IEWLMOPT) and the allocation routine (ALL001).

Exits: When initialization is completed, IEWLMINT passes control to the input processor (IEWLMINP).

Attributes and Options Processor -- IEWLMOPT

Entrance: IEWLMOPT is entered from the initial processor after all data sets except SYSLIB and SYSUT1 are opened.

Operation: IEWLMOPT analyzes the options requested and the attributes specified by the calling program, and notes this information in the APT.

Routines Called: None

Exits: When attribute and option processing is completed, IEWLMOPT returns control to the initial processor (IEWLMINT).

Allocation Processor -- AL001

Entrance: AL001 is entered from the initial processor after all data sets (except SYSLIB and SYSUT1) are opened.

Operation: AL001 issues the GETMAIN macro instruction and assigns storage to buffers. The remaining storage is assigned to tables, with variable tables being assigned as much storage as possible.

Routines Called: None

Exits: When allocation processing is completed, AL001 returns control to the initial processor (IEWLMINT).

Input Processor -- IEWLMINP (Chart JA)

Entrance: IEWLMINP receives control from the initial processor when all initialization functions are completed.

Operation: IEWLMINP reads and initially processes all linkage editor input. Input type (object module or load module) and input conditions are determined, and control is passed to appropriate processors.

Routines Called: IEWLMINP calls the following processors:

- Control statement scanner (IEWLMSCN) when a control statement is detected (blank in column 1).

- Object module processor (IEWLMMDI) when object module input is detected (SYSLIN input or F-format input from SYSLIB).

- Load module processor (INP270) when load module input is detected (U-format input from SYSLIB).

- Include processor (IEWLMINC) at end-of-input, if more modules must be included.

- Automatic library call processor (IEWLCAUT) at end-of-input on SYSLIN, if the NCAL option is not specified.

Exits: When input processing is completed, IEWLMINP passes control to the address assignment processor (IEWLMADA) if valid input was received. If no valid input was received, control is passed to the final processor (IEWLMFNL) to terminate linkage editor processing.

Control Statement Scanner -- IEWLMSCN (Charts JO, JP)

Entrance: IEWLMSCN is entered from the input processor when a control statement is detected.

Operation: Depending on the type of control statement being processed, the control statement scanner makes entries in the APT, SEGTA1, and the CESD. This information is used to control subsequent linkage editor processing.

Routines Called: IEWLMSCN calls the READ8 routine (Chart JQ) to process control statement operands.

Exits: When control statement processing is completed, IEWLMSCN passes control to the include processor (IEWLMINC) if an INCLUDE control statement was processed (include chain built in the CESD). Otherwise, IEWLMSCN returns control to the input processor.

Object Module Processor -- IEWLMMDI (Chart JB)

Entrance: IEWLMMDI is entered from the input processor when object module input is detected.

Operation: IEWLMMDI determines the input record type (SYM, TXT, RLD, ESD, END), loads input record information into general registers, and passes control to the appropriate processors.

Routines Called: Depending on input record type, IEWLMMDI calls the following processors:

- SYM Processor (IEWLMSYM)
- ESD Processor (IEWLMESD)
- END Processor (IEWLMEND)
- Text and RLD Processor (IEWLMRAT)

Exits: When object module processing is completed, IEWLMMDI returns control to the input processor.

Load Module Processor -- INP270 (Chart JC)

Entrance: INP270 is entered from the input processor when load module input is detected.

Operation: INP270 determines the input record type (TXT, CESD, scatter/translation, SYM, CCW, CCW/RLD, RLD), loads input record information into general registers, and passes control to the appropriate processors.

Routines Called: Depending on input record type, INP270 calls an associated processor, as shown in Table 8.

Exits: When load module processing is completed, INP270 returns control to the input processor.

Table 8. Load Module Record Types and Associated Processors

Record Type	Processor
TXT	IEWLMRAT
CESD	IEWLMESD
Scatter/Translation	(Ignored)
SYM	IEWLMSYM
CCW	IEWLMRAT
CCW/RLD	IEWLMRAT
RLD	IEWLMRAT
If end-of-module indicator is on:	
CCW	IEWLMEND
CCW/RLD	IEWLMEND
RLD	IEWLMEND

ESD Processor -- IEWLMESD (Charts JE, JF, JG)

Entrance: IEWLMESD is entered from the object module processor when an ESD record is detected, and from the load module processor when a CESD record is detected.

Operation: IEWLMESD combines ESDs in the linkage editor input into a composite ESD. Matching input symbols are resolved, and specified operations (replace, change, delete) are performed on the symbols. A renumbering table (RNT) is produced to allow input ESD IDs to be translated into CESD IDs.

Routines Called: None

Exits: When ESD processing is completed, IEWLMESD returns control to the routine from which it was entered (object module processor or load module processor).

SYM Processor -- IEWLMSYM (Chart JD)

Entrance: IEWLMSYM is entered from the object module processor when SYM records have been detected and the TEST option has been specified. If TEST is not specified, SYM records are ignored.

Operation: IEWLMSYM gathers SYM records in the RLD input buffer, and writes the buffer contents onto SYSLMOD when the first TXT record of a module is detected.

Routines Called: None

Exits: When SYM processing is completed, IEWLMSYM returns control to the object module processor.

Text and RLD Processor -- IEWLMRAT (Chart JH)

Entrance: IEWLMRAT is entered from the object or load module processors when a text or RLD record is detected.

Operation: IEWLMRAT determines record type (TXT or RLD), checks for error conditions (input record larger than buffer), and passes control to the appropriate processor.

Routines Called: Depending on the record type, IEWLMRAT passes control to either the text processor (IEWLMTXT) or the RLD processor (RLD001).

Exits: When text and RLD processing is completed, IEWLMRAT returns control to the object or load module processor.

Text Processor -- IEWLMTXT (Chart JI)

Entrance: IEWLMTXT is entered from the text and RLD processor when a text record is detected.

Operation: IEWLMTXT operation depends on whether text input is from object or load modules. Object module text is moved from the object module buffer to the input text buffer, and must be arranged in the proper order. Load module text input is already ordered, so IEWLMTXT reads it directly into the input text buffer. In either case, the input text ID is renumbered to refer to the CESD ID of the appropriate control section. When the input text buffer becomes full, its contents are written onto SYSUT1.

Routines Called: When the input text buffer is full, IEWLMTXT calls the text write routine (TXTBUF -- Chart JJ) to write the buffer contents onto SYSUT1.

Exits: When text processing is completed, IEWLMTXT returns control to the text and RLD processor.

RLD Processor -- RLD001 (Charts JK, JL)

Entrance: RLD001 is entered from the text and RLD processor when an RLD record is detected.

Operation: RLD001 groups RLD items in the RLD buffer and renumbers the R and P pointers to refer to appropriate CESD entries. Each RLD item is processed according to its flag and address (FA) field. RLD001 also creates an RLD note list, with entries for each set of RLDs (a set being all RLDs having the same P pointer). If either the RLD buffer or the RLD note list becomes full, the contents of the buffer and the note list are written onto SYSUT1.

Routines Called: When the RLD buffer or the RLD note list is full, RLD001 calls the RLD write routine (RLDBUF -- Chart JM) to write the note list and the buffer contents onto SYSUT1.

Exits: When RLD processing is completed, RLD001 returns control to the text and RLD processor.

End Processor -- IEWLMEND (Chart JN)

Entrance: IEWLMEND is entered from the object or load module processor when an END statement or the end of a load module is detected.

Operation: IEWLMEND resets tables involved in input processing, processes entry point information, deletes CESD lines marked CHAIN or DELETE, and enters into the CESD the length of control sections for which no length was previously indicated.

Routines Called: None

Exits: When end processing is completed, IEWLMEND returns control to the object or load module processor.

Include Processor -- IEWLMINC (Chart JR)

Entrance: IEWLMINC is entered from the input processor when "more includes" are indicated at end-of-input, and from the control statement scanner when an INCLUDE statement has been processed.

Operation: IEWLMINC examines the include chain in the CESD and selects the next module to be included. It opens the data set, determines the attributes of the module to be included, and initializes the DCB to allow the module to be read.

Routines Called: None.

Exits: When include processing is completed, control is returned to the input processor.

Automatic Library Call Processor -- IEWLCAUT (Charts JS, JT)

Entrance: IEWLCAUT is entered from the input processor at the end of SYSLIN input, or when a NAME statement has been detected (provided that the NCAL option was not specified).

Operation: IEWLCAUT first scans the CESD for unresolved ERs specified on LIBRARY statements. It attempts to resolve these ERs by searching the PDS directories of ddnames included in library chains, allowing the members found to be read. A second CESD scan attempts to resolve ERs not specified on LIBRARY statements by attempting to call them from SYSLIB.

Routines Called: After the first series of CESD scans, IEWLCAUT returns control to the input processor to read the members.

Exits: After the second series of CESD scans, IEWLCAUT passes control to the address assignment processor (IEWLMADA).

INTERMEDIATE PROCESSING

Address Assignment Processor -- IEWLMADA (Chart KA)

Entrance: IEWLMADA is entered from the input processor when input processing is completed.

Operation: IEWLMADA assigns linked addresses to all CESD entries, determines the size of SEGTAB if the program is in overlay, determines the number of ENTAB bytes required for each segment, builds the alias table, and determines an entry point for the program.

Routines Called: IEWLMADA call the ENTAB size determination routine (IEWMLENS -- Chart KB) to compute the size of ENTABs, and calls the entry processor (IEWLMENT -- Charts KC, KD) to build the alias table and determine an entry point.

Exits: When address assignment processing is completed, IEWLMADA passes control to the intermediate output processor (IEWLMOUT).

Intermediate Output Processor -- IEWLMOUT (Chart LA)

Entrance: IEWLMOUT is entered from IEWLMADA when address assignment processing is complete.

Operation: IEWLMOUT writes the following onto SYSLMOD: CESD, SEGTAB (for programs in overlay), and scatter/translation records (for programs to be scatter loaded).

If the MAP option has been specified, a module map is produced and written on SYSPRINT; if the XREF option has been specified and all RLDs are in storage, a cross-reference table is produced and written on SYSPRINT.

If the TXT and RLD note lists were placed on SYSUT1 during TXT and RLD processing, IEWLMOUT reads them back into storage, and builds the high ID table (HIID). The half ESD (HESD) is also built, after the CESD has been written.

Routines Called: IEWLMOUT calls the MAP/XREF processor (IEWLMMAP) to produce and write the module map and cross-reference table, if requested.

Exits: When intermediate output processing is completed, control is passed to the second pass processor (IEWLMSCD).

SECOND PASS PROCESSING

Second Pass Processor -- IEWLMSCD (Charts MA, MB)

Entrance: IEWLMSCD is entered from IEWLMOUT when intermediate output processing is completed.

Operation: IEWLMSCD performs the following functions:

- Reads text from SYSUT1.
- Relocates address constants contained in the text.
- Creates control/RLD records.
- Writes text and control/RLD records onto SYSLMOD in a format that can be loaded by program fetch.
- Creates ENTABs and associated RLD items for overlay modules.

Routines Called: During second pass processing, IEWLMSCD calls the following routines:

- Control section search routine (GETIDMUL -- Chart MC) to determine the next ID and multiplicity to be processed.
- Text and RLD read routines (RDTXT, RDRLD -- Chart MD) to read required text and RLDs from SYSUT1.
- Text write routine (WRTTXT -- Chart ME) to write text onto SYSLMOD.

- RLD/control record write routine (WRTCRRLD) to write RLDs and control records onto SYSLMOD.

- Relocation routine (RELOCATE -- Charts MF, MG, MH) to relocate address constants (branch type and non-branch type) in the text.

- Common path routine (IEWLCPTH) to determine common segments in an overlay path.

- ENTAB creation routine (SCDENTAB) to create ENTAB items for each segment.

Exits: When second pass processing is completed, control is passed to the final processor (IEWLMFNL).

FINAL PROCESSING

Final Processor -- IEWLMFNL (Chart NA)

Entrance: IEWLMFNL is entered from IEWLMSCD when second pass processing is completed.

Operation: IEWLMFNL performs the following "cleanup" functions:

- Writes the TTR list for overlay modules onto SYSLMOD.

- Places entries in the partitioned data set directory and issues a STOW macro instruction.

- Prints a directory of logged errors.

- Checks for more restrictive module attributes.

- Produces a cross-reference table if it was requested and not produced during intermediate processing.

Routines Called: During final processing, IEWLMFNL calls the following routines:

- Diagnostic message directory print routine (IEWLMBTP) which scans the error logging map produced throughout linkage editor processing by the error logging routine (IEWLMLOG -- Chart NC); IEWLMBTP builds and prints a directory of error messages.

- MAP/XREF processor (IEWLMMAP -- Chart LB) which produces a cross reference table if it was not produced during intermediate processing.

Exits: If end-of-file was not detected on a SYSLIN input, IEWLMFNL returns control to the initial processor (IEWLMINT), and linkage editor processing is repeated. Otherwise, linkage editor processing is terminated, and control is returned to the control program.

SYNAD Routine (Chart NB)

Entrance: The SYNAD routine may be entered from the following routines:

- From the control program when any I/O error has been detected.

- From the second pass processor, if an error is found after executing XDAP

Operation: Following are SYNAD considerations for linkage editor F:

- The SYNAD fields of the DCBs in IEWLMROU contain the address of the appropriate SYNAD entry point for the access method used with the data set.

- If the SYNAD routine is entered from the input processor because of incorrect length, the length of the incorrect input block is checked. If a valid short block (integral multiple of LRECL) is found, control is returned to the supervisor to continue processing; if not, processing is terminated with an error message and completion code of 16.

- If the SYNAD routine is entered while writing to the SYSPRINT data set, control is passed to the final processor, and execution is abnormally terminated with a condition code of 16.

- When the include processor opens the DCB for SYSLIB, the address of the appropriate SYNAD entry (for either BSAM or BPAM access methods) is moved into the SYNAD field.

- If the second pass processor finds an error after executing XDAP, it loads register 1 with the IOB address, loads register 15 with the SYNAD entry point for EXCP, and branches on register 15.

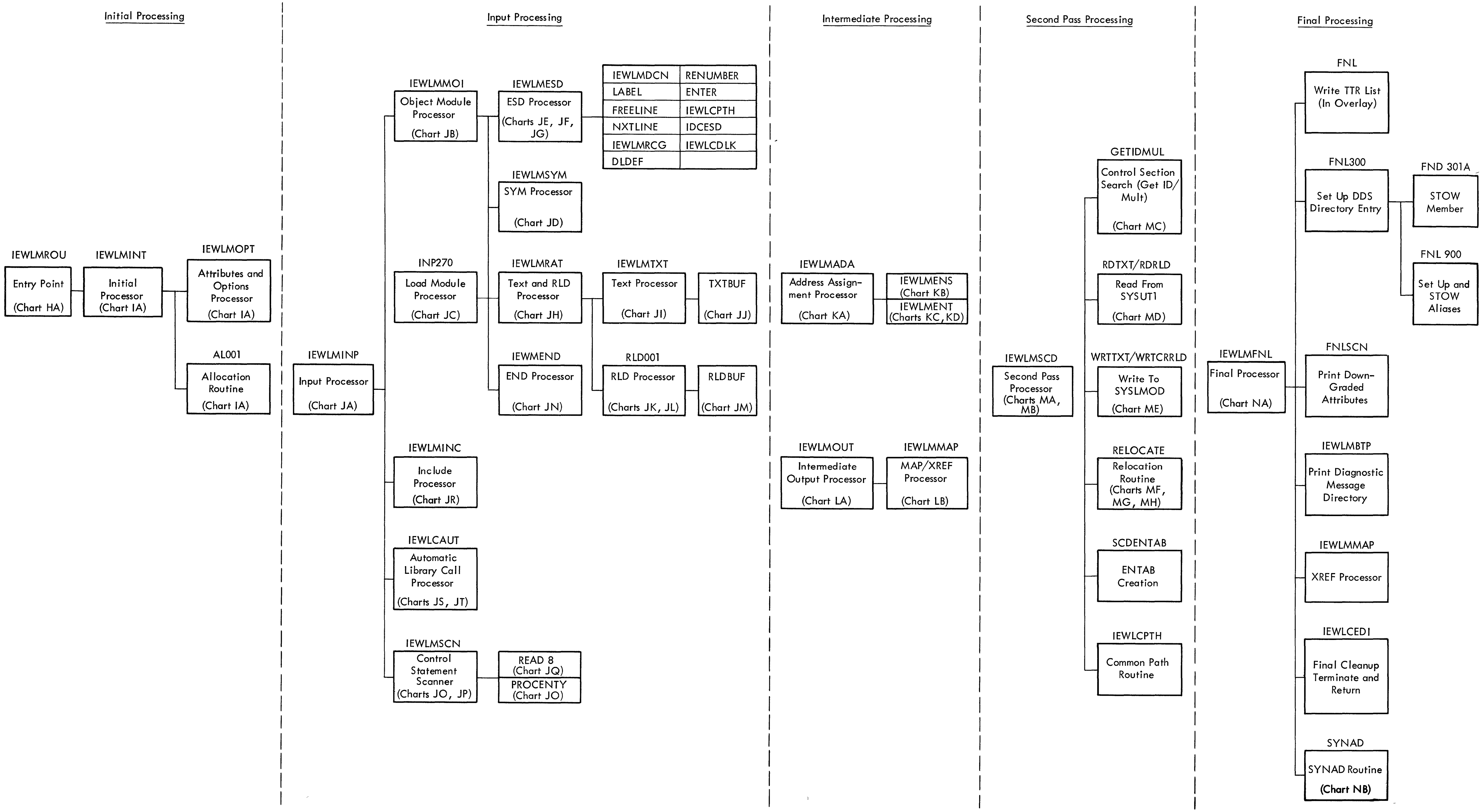


Figure 34. Linkage Editor F Organization

Chart HA. Level F Major Divisions

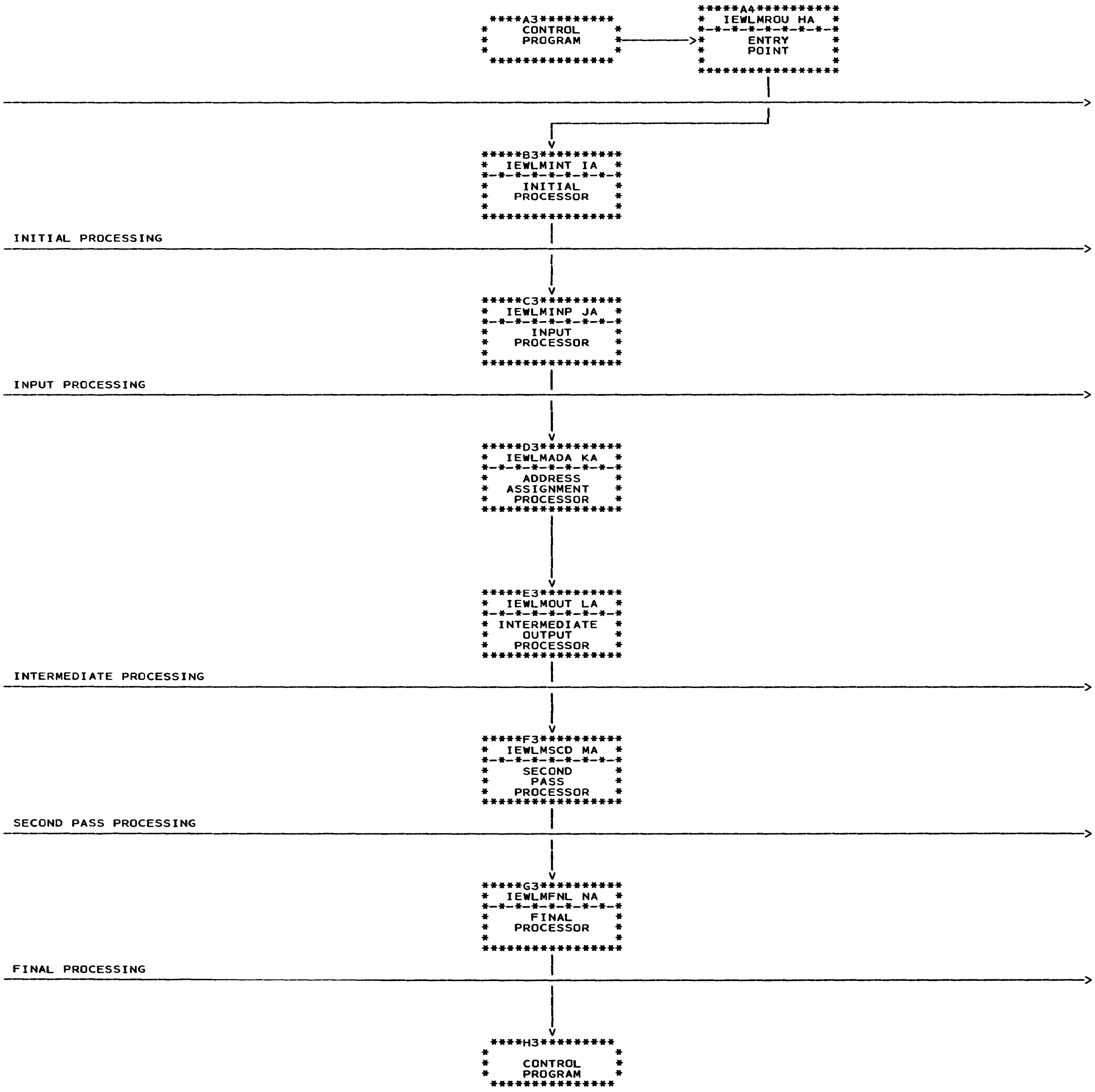
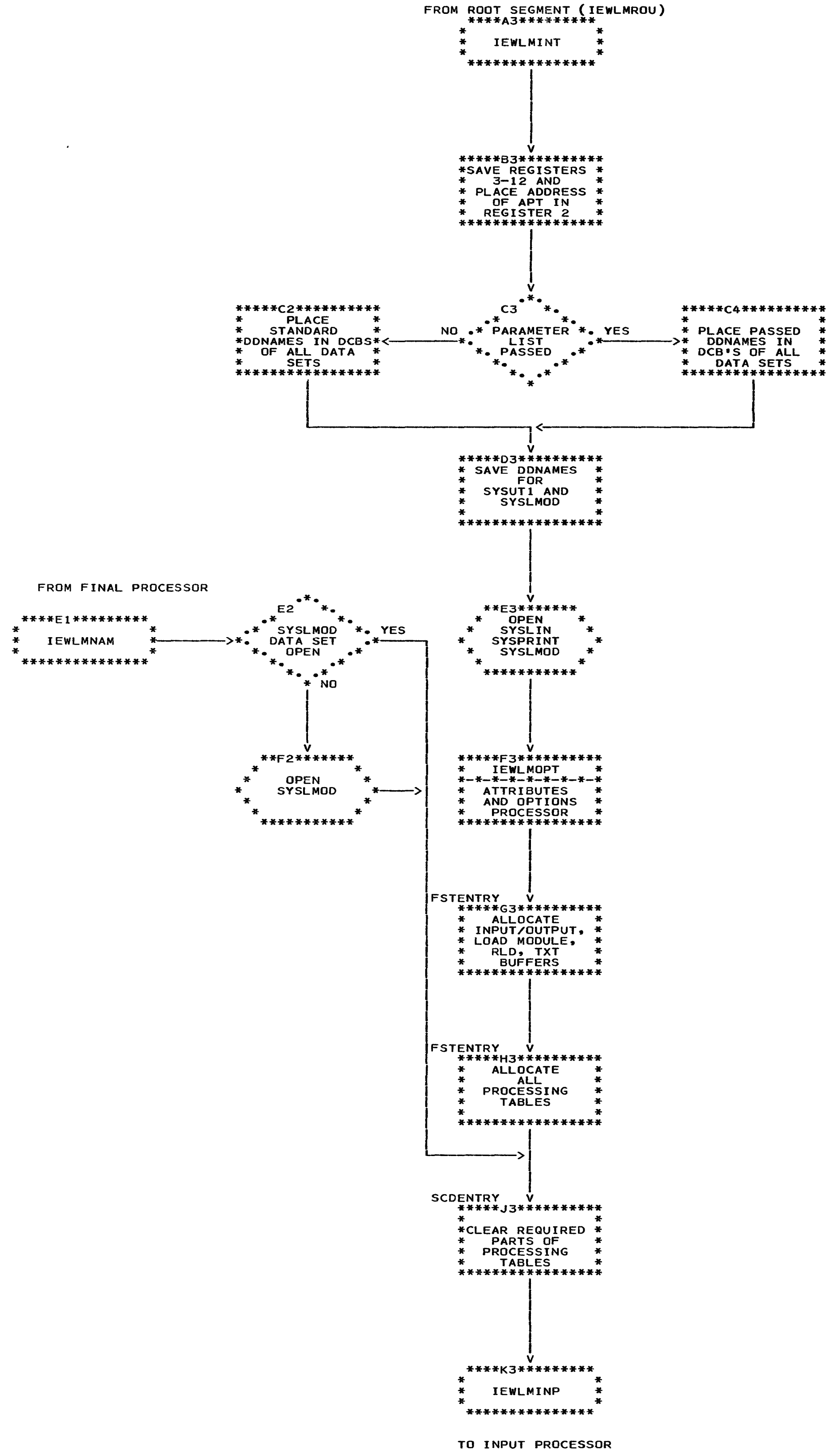


Chart IA. Initial Processor (IEWLMINT)



FROM ROOT SEGMENT (IEWLMROU)
A3
IEWLMINT
B3
SAVE REGISTERS 3-12 AND PLACE ADDRESS OF APT IN REGISTER 2
C2
PLACE STANDARD DDNAMES IN DCBS OF ALL DATA SETS
C3
PARAMETER LIST PASSED
NO
YES
C4
PLACE PASSED DDNAMES IN DCB'S OF ALL DATA SETS
D3
SAVE DDNAMES FOR SYSUT1 AND SYSLMOD
FROM FINAL PROCESSOR
E1
IEWLMNAM
E2
SYSLMOD DATA SET OPEN
YES
NO
E3
OPEN SYSLIN SYSPRINT SYSLMOD
F2
OPEN SYSLMOD
F3
IEWLMOPT
ATTRIBUTES AND OPTIONS PROCESSOR
FSTENTRY
G3
ALLOCATE INPUT/OUTPUT, LOAD MODULE, RLD, TXT BUFFERS
FSTENTRY
H3
ALLOCATE ALL PROCESSING TABLES
SCDENTRY
J3
CLEAR REQUIRED PARTS OF PROCESSING TABLES
K3
IEWLMINP
TO INPUT PROCESSOR

Chart JA. Input Processor (IEWLMINP)

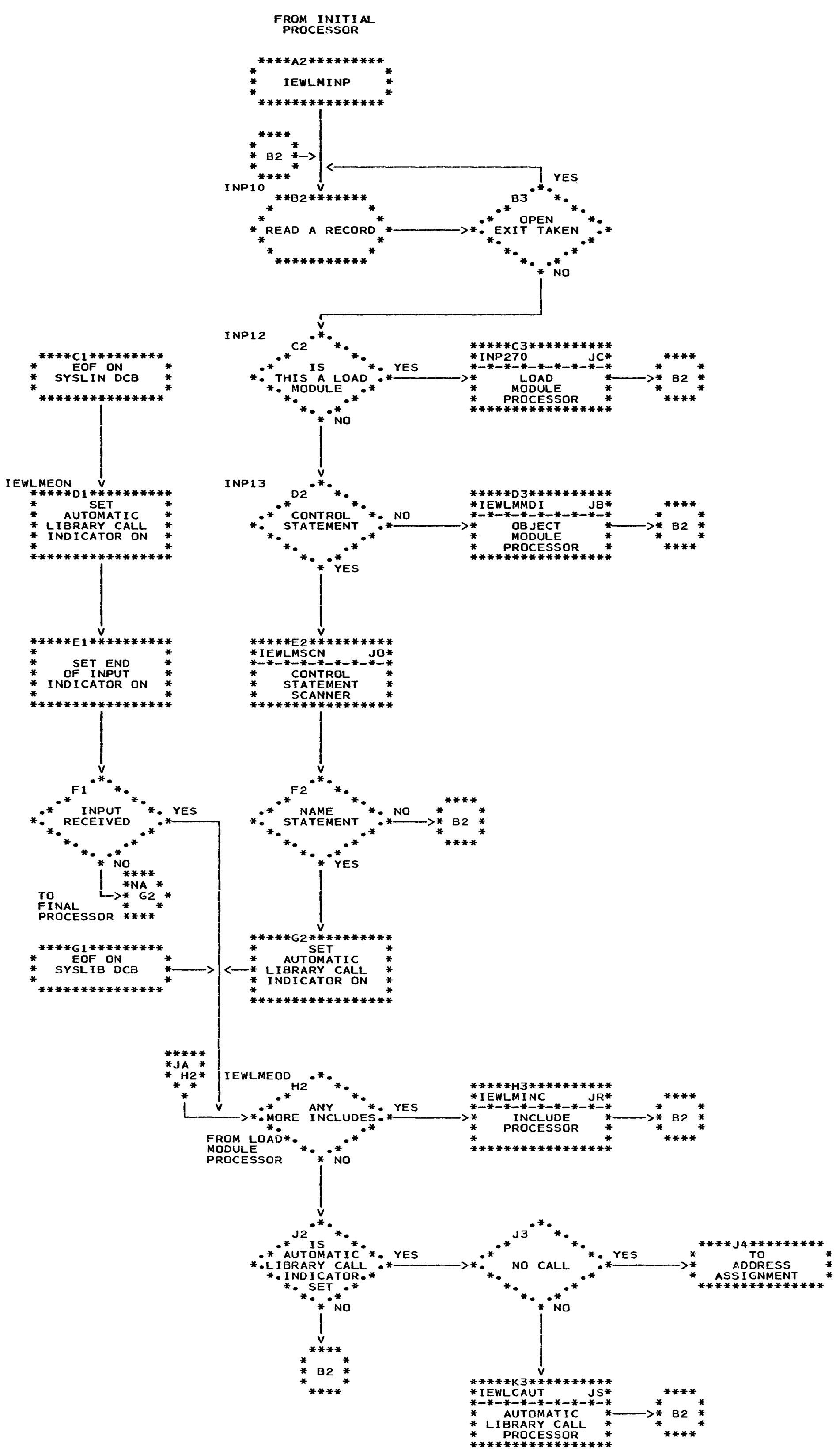


Chart JB. Object Module Processor (IEWLMMDI)

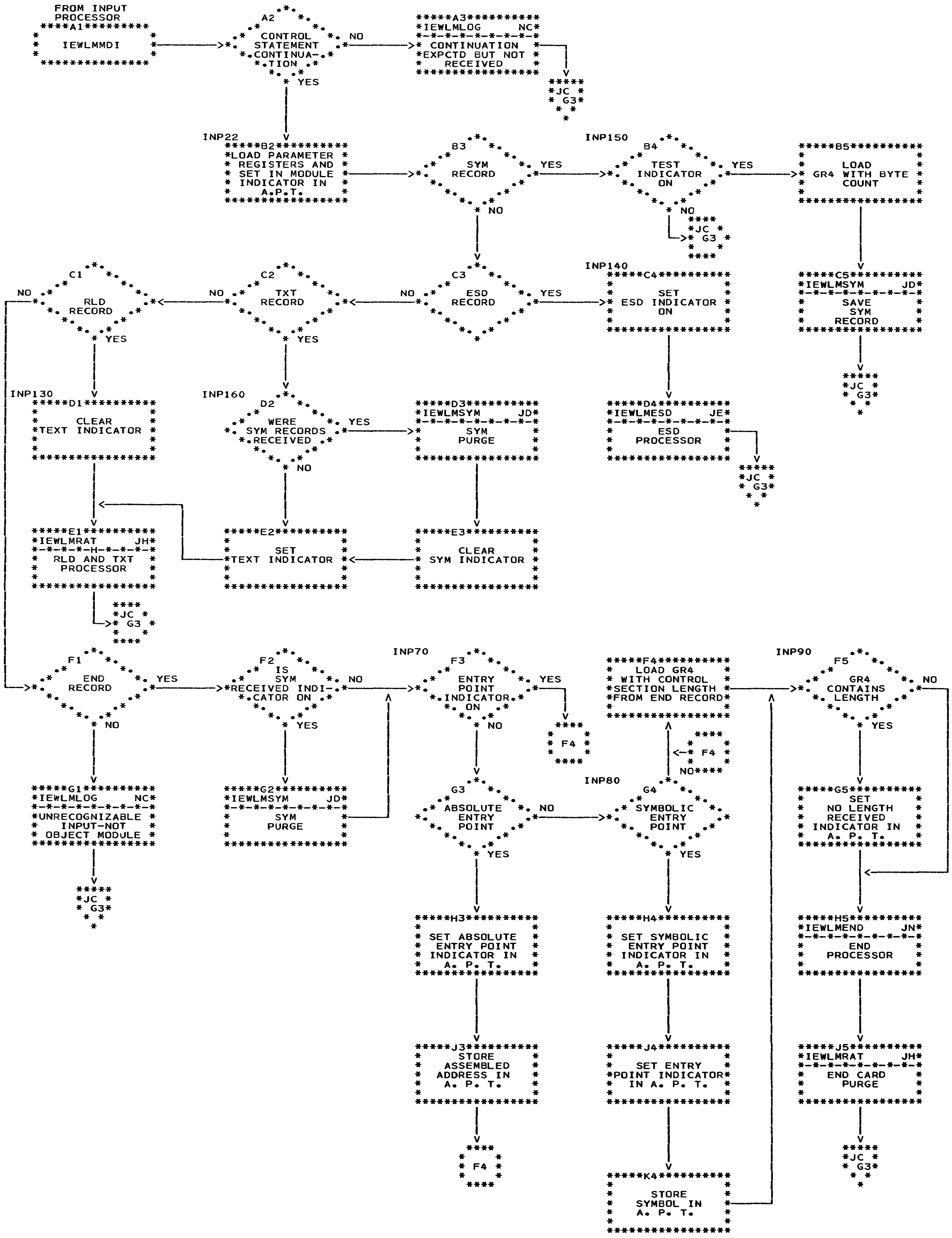


Chart JC. Load Module Processor (INP270)

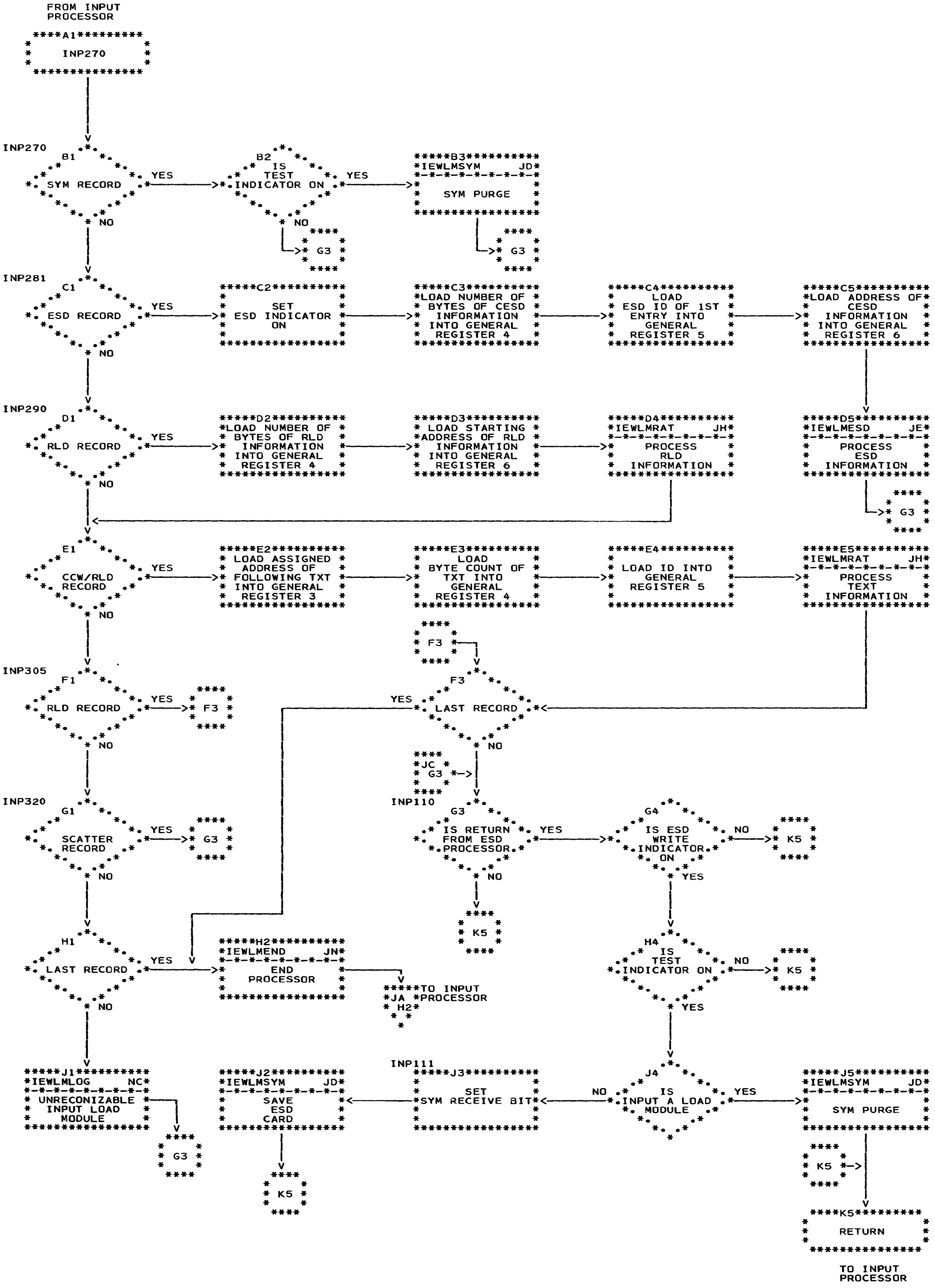


Chart JD. Sym Processor (IEWLMSYM)

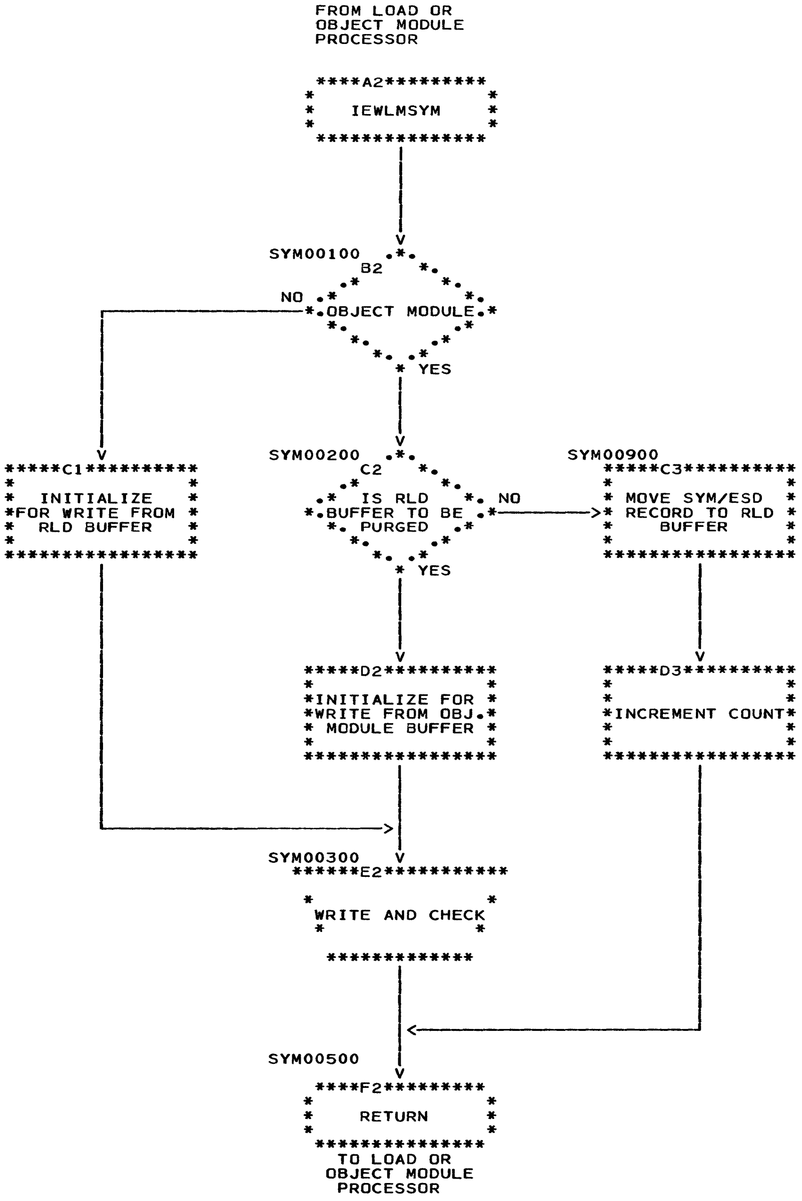
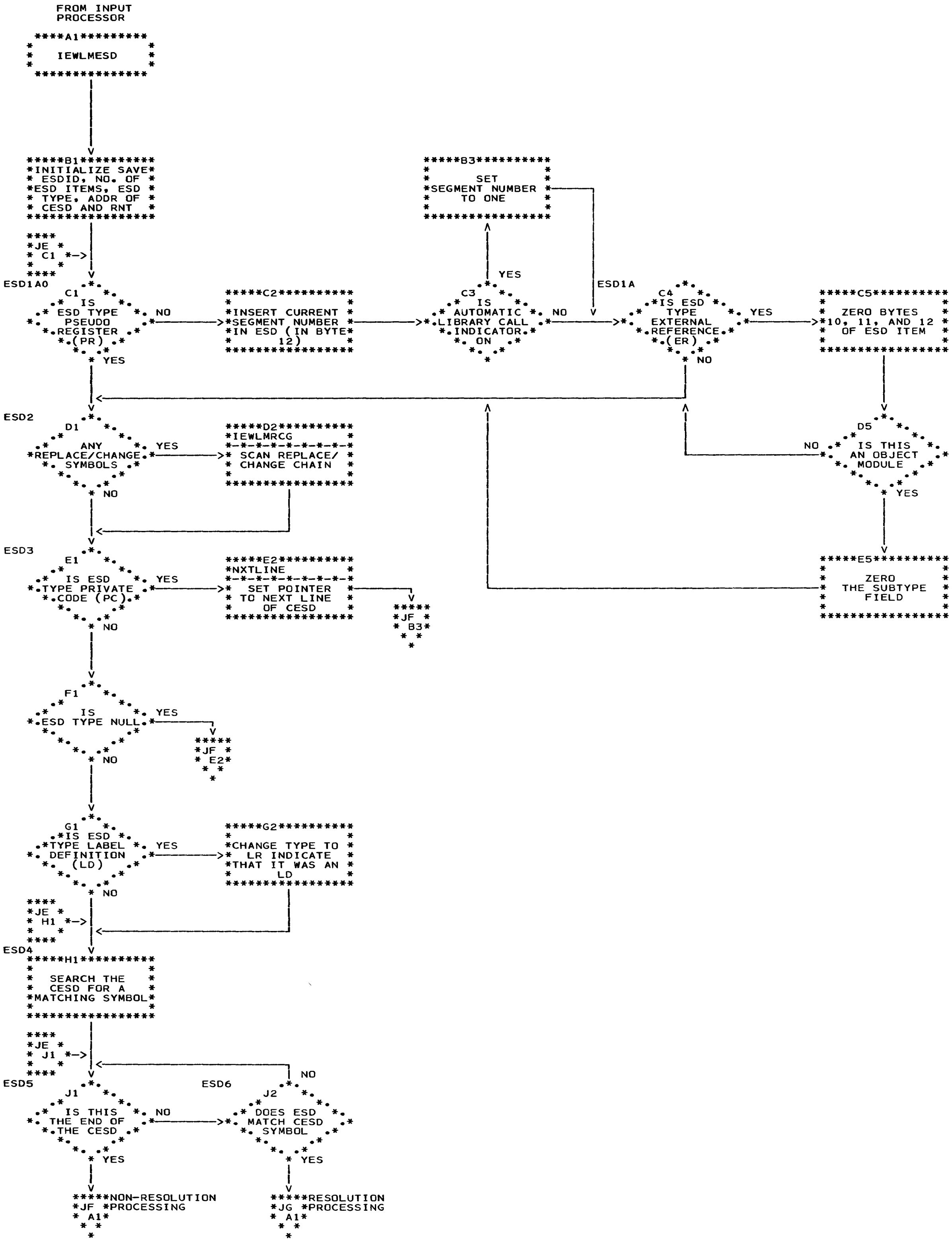

FROM LOAD OR OBJECT MODULE PROCESSOR
A2
IEWLMSYM
SYM00100
B2
OBJECT MODULE
NO
YES
C1
INITIALIZE FOR WRITE FROM RLD BUFFER
SYM00200
C2
IS RLD BUFFER TO BE PURGED
NO
YES
SYM00900
C3
MOVE SYM/ESD RECORD TO RLD BUFFER
D2
INITIALIZE FOR WRITE FROM OBJ. MODULE BUFFER
D3
INCREMENT COUNT
SYM00300
E2
WRITE AND CHECK
SYM00500
F2
RETURN
TO LOAD OR OBJECT MODULE PROCESSOR

Chart JE. ESD Processor (IEWLMESD)



FROM INPUT PROCESSOR
A1 IEWLMESD
B1 INITIALIZE SAVE ESDID, NO. OF ESD ITEMS, ESD TYPE, ADDR OF CESD AND RNT
JE C1
ESD1A0
C1 IS ESD TYPE PSEUDO REGISTER (PR)
NO
YES
C2 INSERT CURRENT SEGMENT NUMBER IN ESD (IN BYTE 12)
B3 SET SEGMENT NUMBER TO ONE
YES
C3 IS AUTOMATIC LIBRARY CALL INDICATOR ON
NO
ESD1A
C4 IS ESD TYPE EXTERNAL REFERENCE (ER)
YES
NO
C5 ZERO BYTES 10, 11, AND 12 OF ESD ITEM
ESD2
D1 ANY REPLACE/CHANGE SYMBOLS
YES
NO
D2 IEWLMRCG SCAN REPLACE/ CHANGE CHAIN
D5 IS THIS AN OBJECT MODULE
NO
YES
ESD3
E1 IS ESD TYPE PRIVATE CODE (PC)
YES
NO
E2 NXTLINE SET POINTER TO NEXT LINE OF CESD
JF B3
E5 ZERO THE SUBTYPE FIELD
F1 IS ESD TYPE NULL
YES
NO
JF E2
G1 IS ESD TYPE LABEL DEFINITION (LD)
YES
NO
G2 CHANGE TYPE TO LR INDICATE THAT IT WAS AN LD
JE H1
ESD4
H1 SEARCH THE CESD FOR A MATCHING SYMBOL
JE J1
NO
ESD5
J1 IS THIS THE END OF THE CESD
NO
YES
ESD6
J2 DOES ESD MATCH CESD SYMBOL
YES
JF A1 NON-RESOLUTION PROCESSING
JG A1 RESOLUTION PROCESSING

Chart JF. ESD Processor (IEWLMESD)

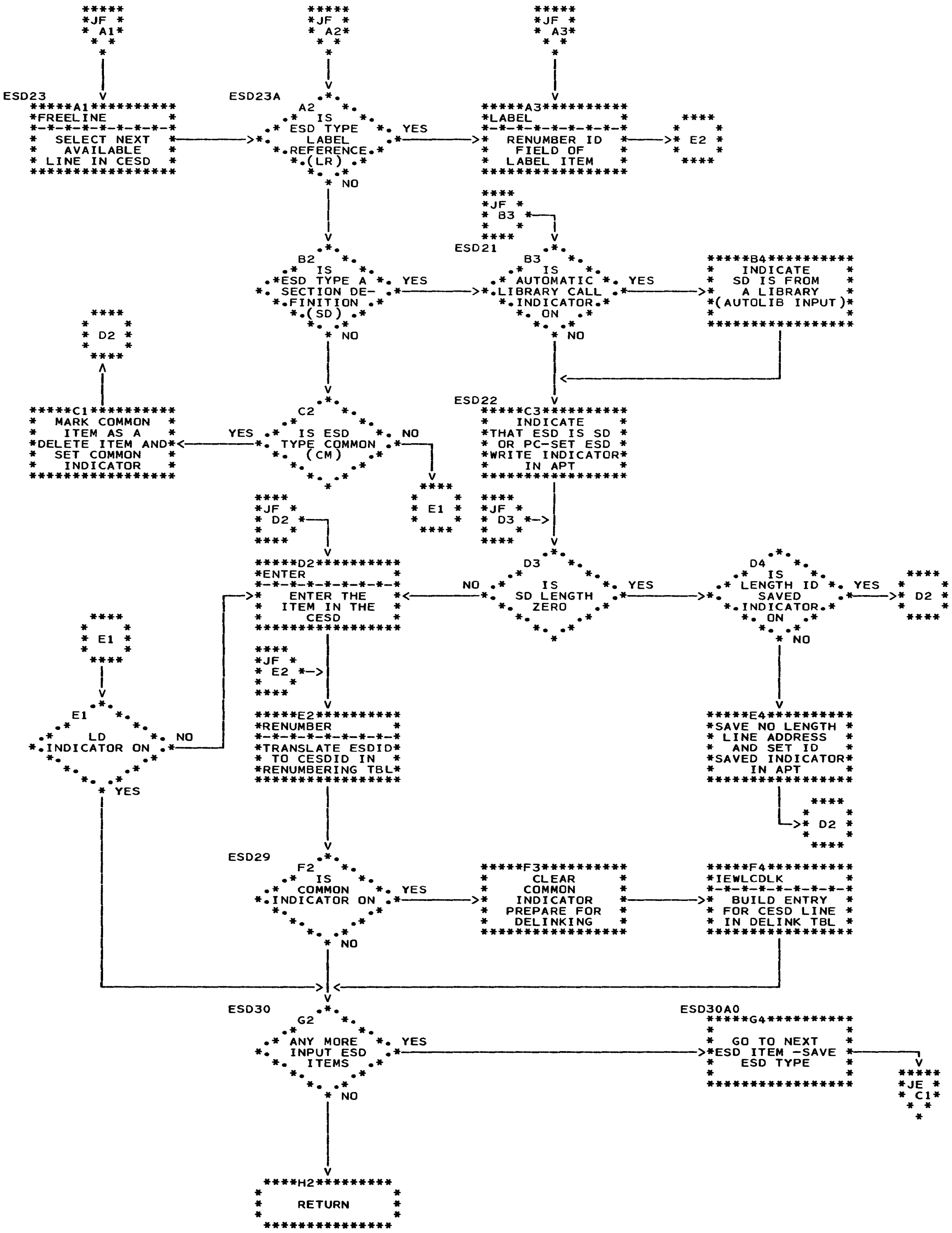
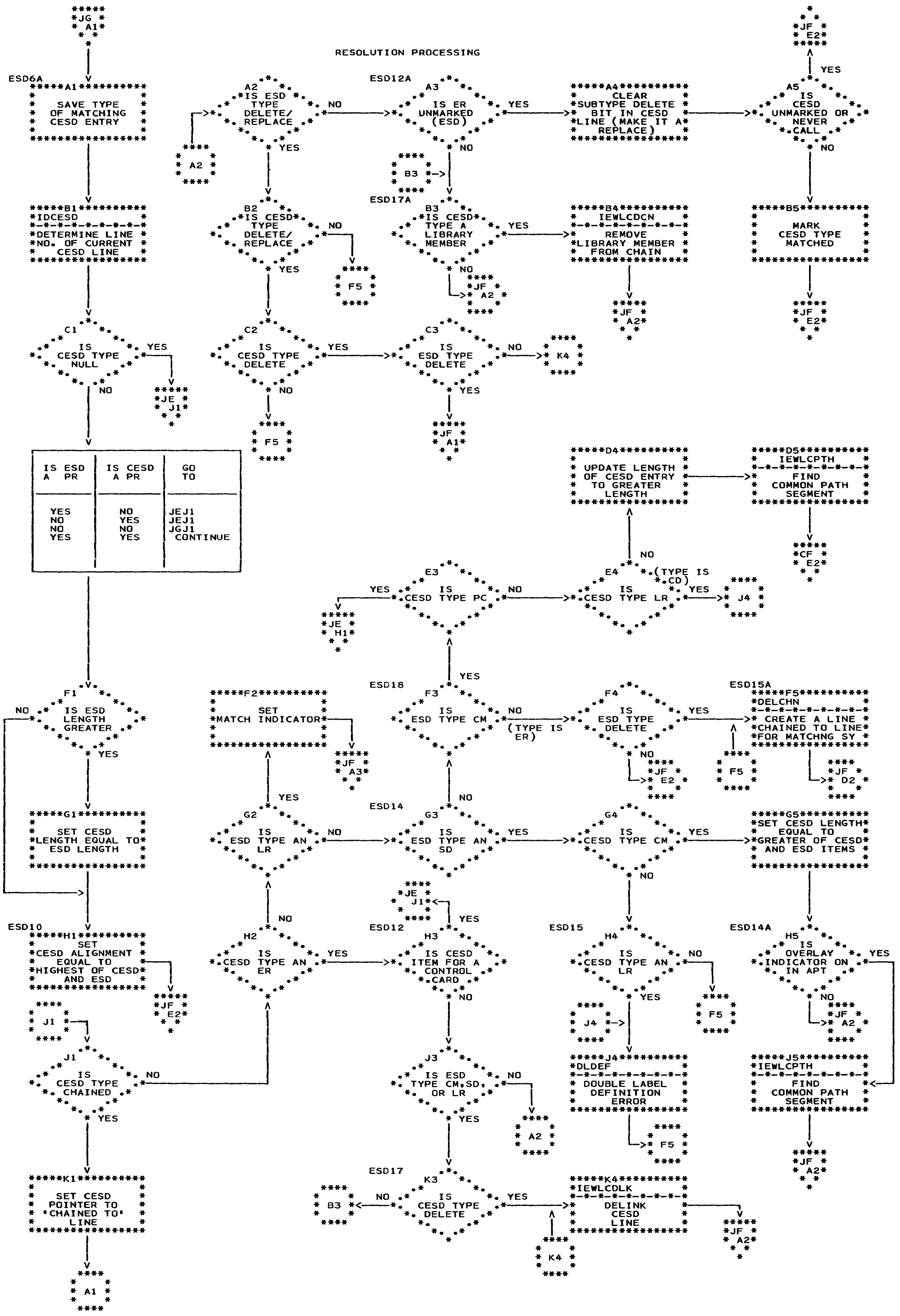

JF A1
JF A2
JF A3
ESD23
A1
FREELINE
SELECT NEXT AVAILABLE LINE IN CESD
ESD23A
A2
IS ESD TYPE LABEL REFERENCE (LR)
YES
NO
A3
LABEL
RENUMBER ID FIELD OF LABEL ITEM
E2
JF B3
ESD21
B2
IS ESD TYPE A SECTION DE- FINITION (SD)
YES
NO
B3
IS AUTOMATIC LIBRARY CALL INDICATOR ON
YES
NO
B4
INDICATE SD IS FROM A LIBRARY (AUTOLIB INPUT)
D2
ESD22
C1
MARK COMMON ITEM AS A DELETE ITEM AND SET COMMON INDICATOR
C2
IS ESD TYPE COMMON (CM)
YES
NO
C3
INDICATE THAT ESD IS SD OR PC-SET ESD WRITE INDICATOR IN APT
E1
JF D2
JF D3
D2
ENTER
ENTER THE ITEM IN THE CESD
D3
IS SD LENGTH ZERO
NO
YES
D4
LENGTH ID SAVED INDICATOR ON
YES
NO
D2
E1
JF E2
E1
LD INDICATOR ON
NO
YES
E2
RENUMBER
TRANSLATE ESDID TO CESDID IN RENUMBERING TBL
E4
SAVE NO LENGTH LINE ADDRESS AND SET ID SAVED INDICATOR IN APT
D2
ESD29
F2
IS COMMON INDICATOR ON
YES
NO
F3
CLEAR COMMON INDICATOR PREPARE FOR DELINKING
F4
IEWLCDLK
BUILD ENTRY FOR CESD LINE IN DELINK TBL
ESD30
G2
ANY MORE INPUT ESD ITEMS
YES
NO
ESD30A0
G4
GO TO NEXT ESD ITEM -SAVE ESD TYPE
JE C1
H2
RETURN

Chart JG. ESD Processor (IEWLMESD)



JG A1
JF E2
RESOLUTION PROCESSING
ESD6A
A1 SAVE TYPE OF MATCHING CESD ENTRY
A2 IS ESD TYPE DELETE/ REPLACE
NO
YES
A2
ESD12A
A3 IS ER UNMARKED (ESD)
YES
NO
B3
A4 CLEAR SUBTYPE DELETE BIT IN CESD LINE (MAKE IT A REPLACE)
A5 IS CESD UNMARKED OR NEVER CALL
YES
NO
B1 IDCESD DETERMINE LINE NO. OF CURRENT CESD LINE
B2 IS CESD TYPE DELETE/ REPLACE
F5
ESD17A
B3 IS CESD TYPE A LIBRARY MEMBER
JF A2
B4 IEWLCDCN REMOVE LIBRARY MEMBER FROM CHAIN
JF A2
B5 MARK CESD TYPE MATCHED
JF E2
C1 IS CESD TYPE NULL
JE J1
C2 IS CESD TYPE DELETE
F5
C3 IS ESD TYPE DELETE
K4
JF A1
IS ESD A PR
IS CESD A PR
GO TO
YES NO JEJ1
NO YES JEJ1
NO NO JGJ1
YES YES CONTINUE
D4 UPDATE LENGTH OF CESD ENTRY TO GREATER LENGTH
D5 IEWLCPTH FIND COMMON PATH SEGMENT
CF E2
E3 IS CESD TYPE PC
JE H1
E4 IS CESD TYPE LR
(TYPE IS CD)
J4
ESD18
F1 IS ESD LENGTH GREATER
F2 SET MATCH INDICATOR
JF A3
F3 IS ESD TYPE CM
(TYPE IS ER)
F4 IS ESD TYPE DELETE
JF E2
F5
ESD15A
F5 DELCHN CREATE A LINE CHAINED TO LINE FOR MATCHNG SY
JF D2
G1 SET CESD LENGTH EQUAL TO ESD LENGTH
ESD14
G2 IS ESD TYPE AN LR
G3 IS ESD TYPE AN SD
G4 IS CESD TYPE CM
G5 SET CESD LENGTH EQUAL TO GREATER OF CESD AND ESD ITEMS
JE J1
ESD10
H1 SET CESD ALIGNMENT EQUAL TO HIGHEST OF CESD AND ESD
JF E2
H2 IS CESD TYPE AN ER
ESD12
H3 IS CESD ITEM FOR A CONTROL CARD
ESD15
H4 IS CESD TYPE AN LR
F5
ESD14A
H5 IS OVERLAY INDICATOR ON IN APT
JF A2
J1
J4
J1 IS CESD TYPE CHAINED
J3 IS ESD TYPE CM,SD, OR LR
A2
J4 DLDEF DOUBLE LABEL DEFINITION ERROR
F5
J5 IEWLCPTH FIND COMMON PATH SEGMENT
JF A2
ESD17
K1 SET CESD POINTER TO 'CHAINED TO' LINE
A1
B3
K3 IS CESD TYPE DELETE
K4 IEWLCDLK DELINK CESD LINE
K4
JF A2

Chart JH. TXT and RLD Processor (IEWLMRAT)

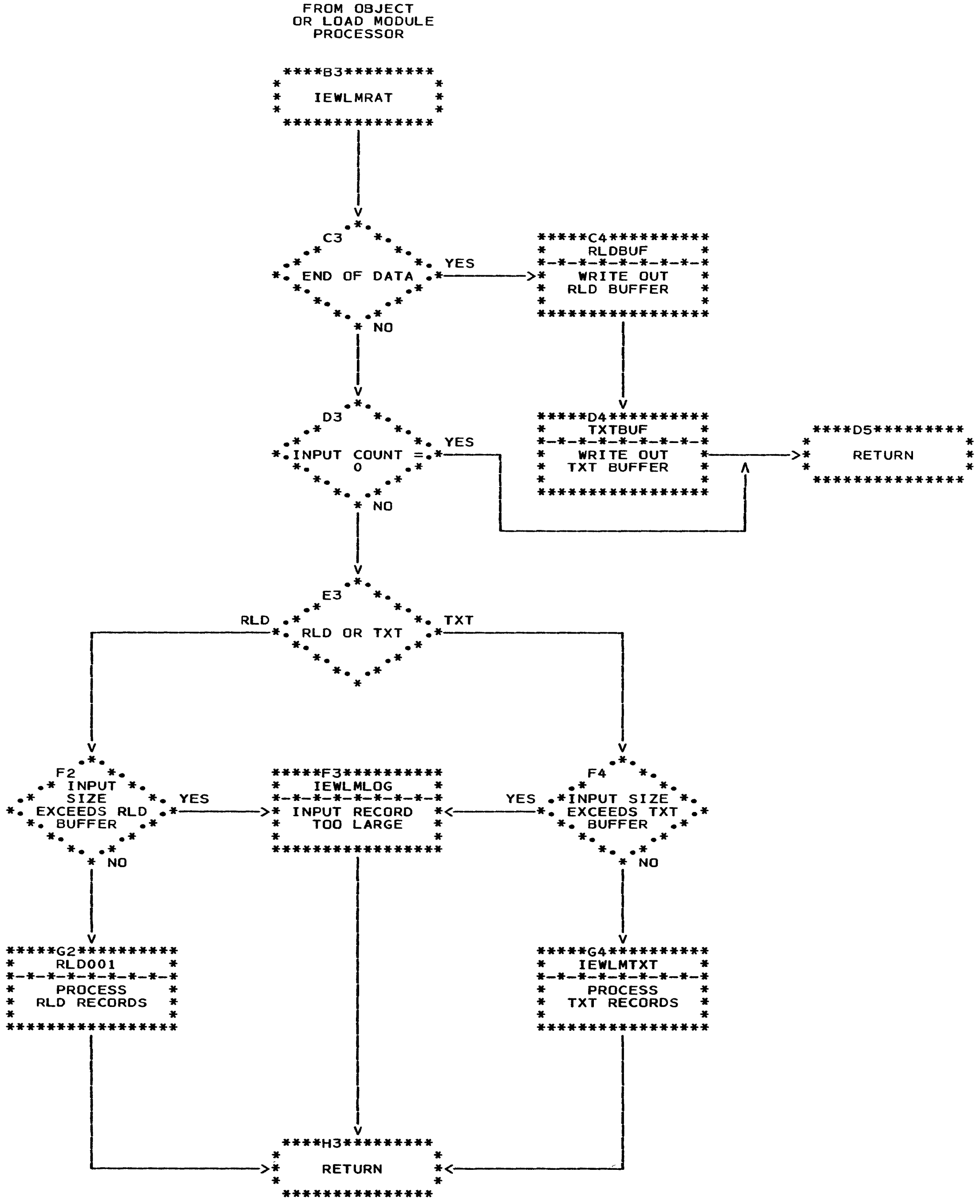


Chart JI. TXT Processor (IEWLMTXT)

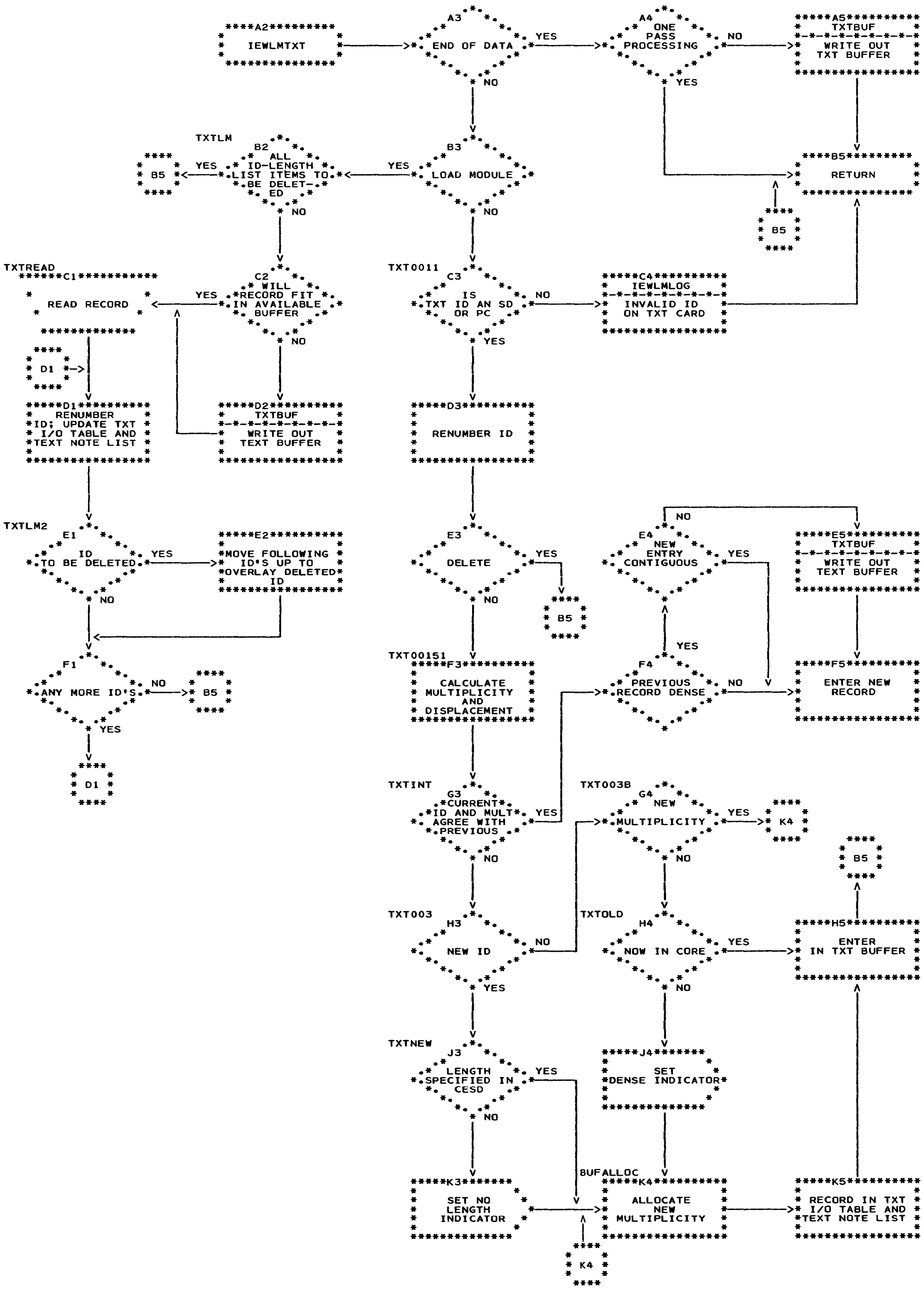


Chart JJ. Level F TXTBUF Routine

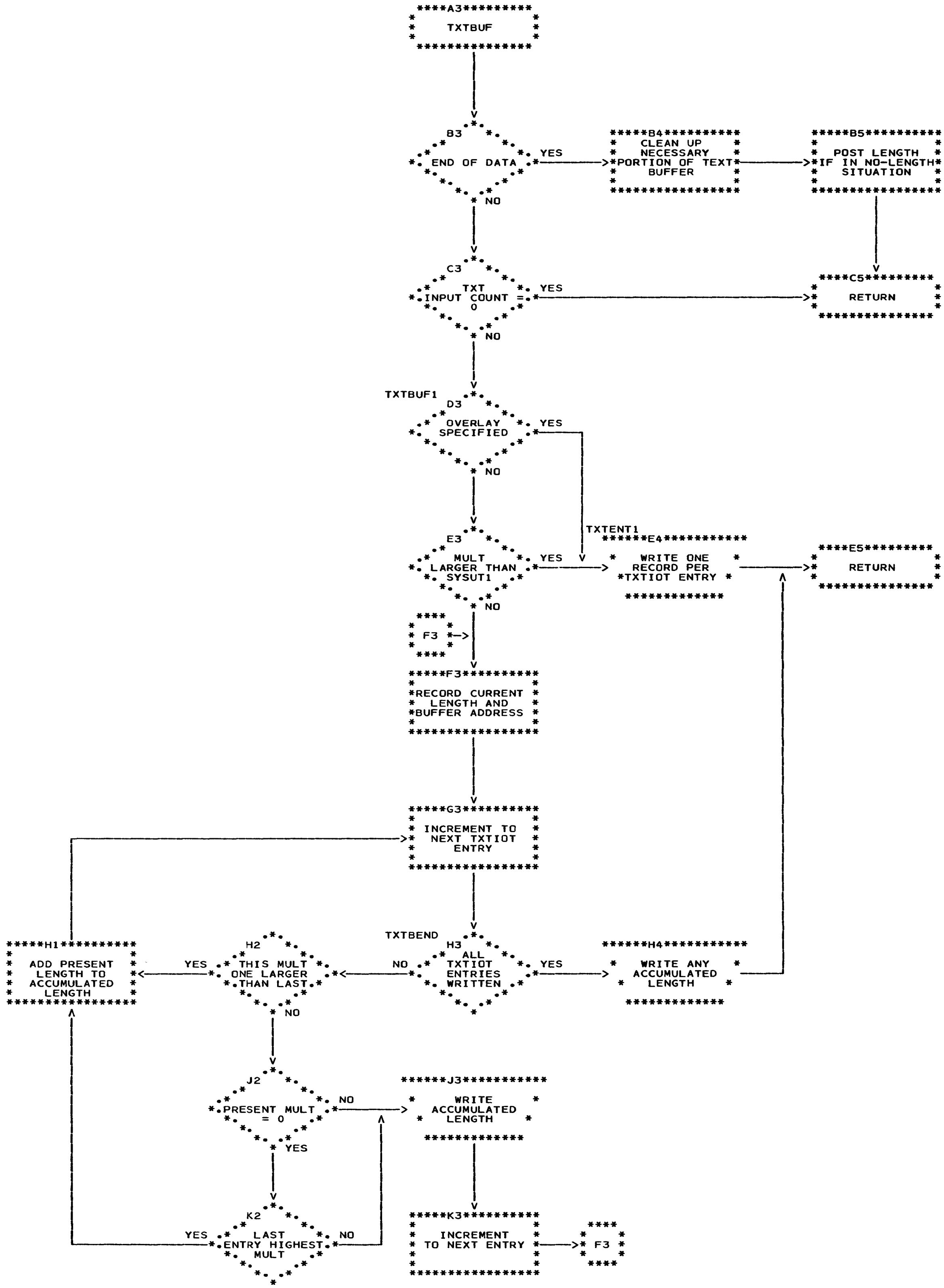

A3
TXTBUF
B3
END OF DATA
YES
NO
B4
CLEAN UP NECESSARY PORTION OF TEXT BUFFER
B5
POST LENGTH IF IN NO-LENGTH SITUATION
C3
TXT INPUT COUNT = 0
YES
NO
C5
RETURN
TXTBUF1
D3
OVERLAY SPECIFIED
YES
NO
TXTENT1
E3
MULT LARGER THAN SYSUT1
YES
NO
E4
WRITE ONE RECORD PER TXTIOT ENTRY
E5
RETURN
F3
F3
RECORD CURRENT LENGTH AND BUFFER ADDRESS
G3
INCREMENT TO NEXT TXTIOT ENTRY
H1
ADD PRESENT LENGTH TO ACCUMULATED LENGTH
H2
THIS MULT ONE LARGER THAN LAST
YES
NO
TXTBEND
H3
ALL TXTIOT ENTRIES WRITTEN
NO
YES
H4
WRITE ANY ACCUMULATED LENGTH
J2
PRESENT MULT = 0
NO
YES
J3
WRITE ACCUMULATED LENGTH
K2
LAST ENTRY HIGHEST MULT
YES
NO
K3
INCREMENT TO NEXT ENTRY
F3

Chart JK. Level F RLD Processor

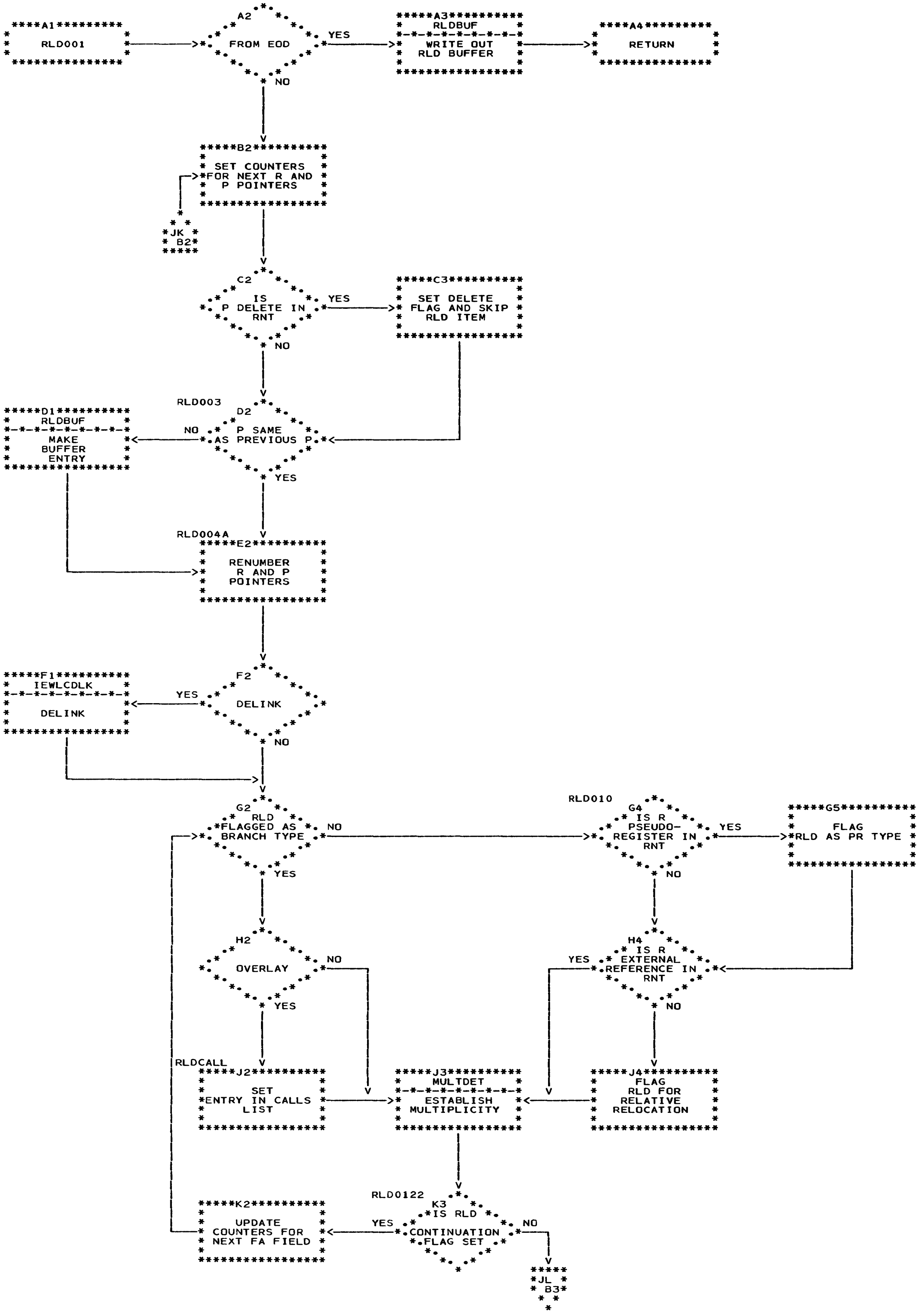


Chart JL. Level L RLD Processor (Continued)

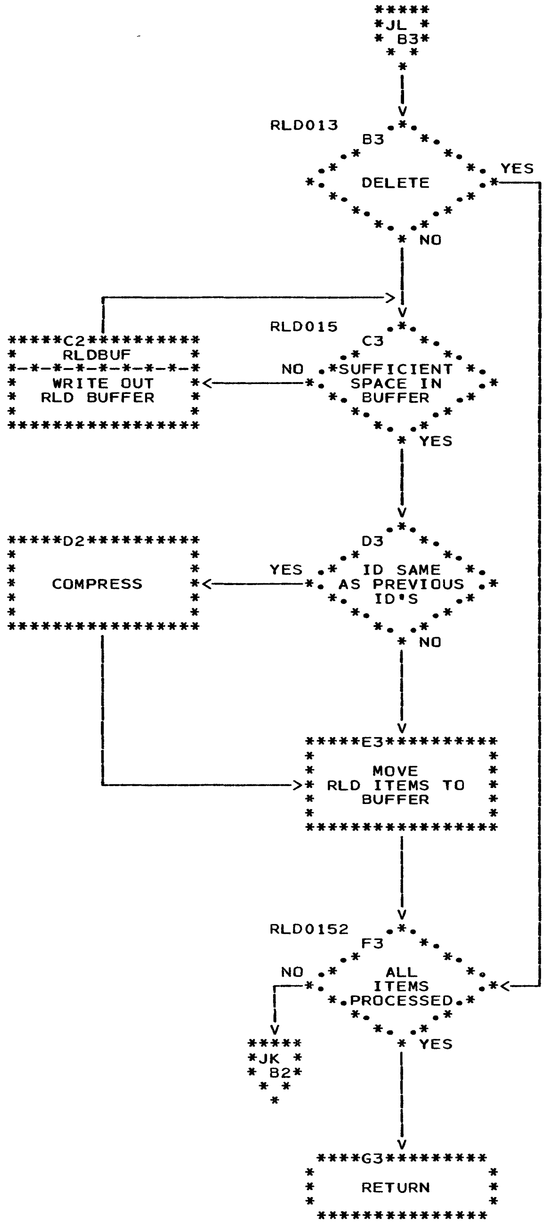


Chart JM. RLDBUF Routine

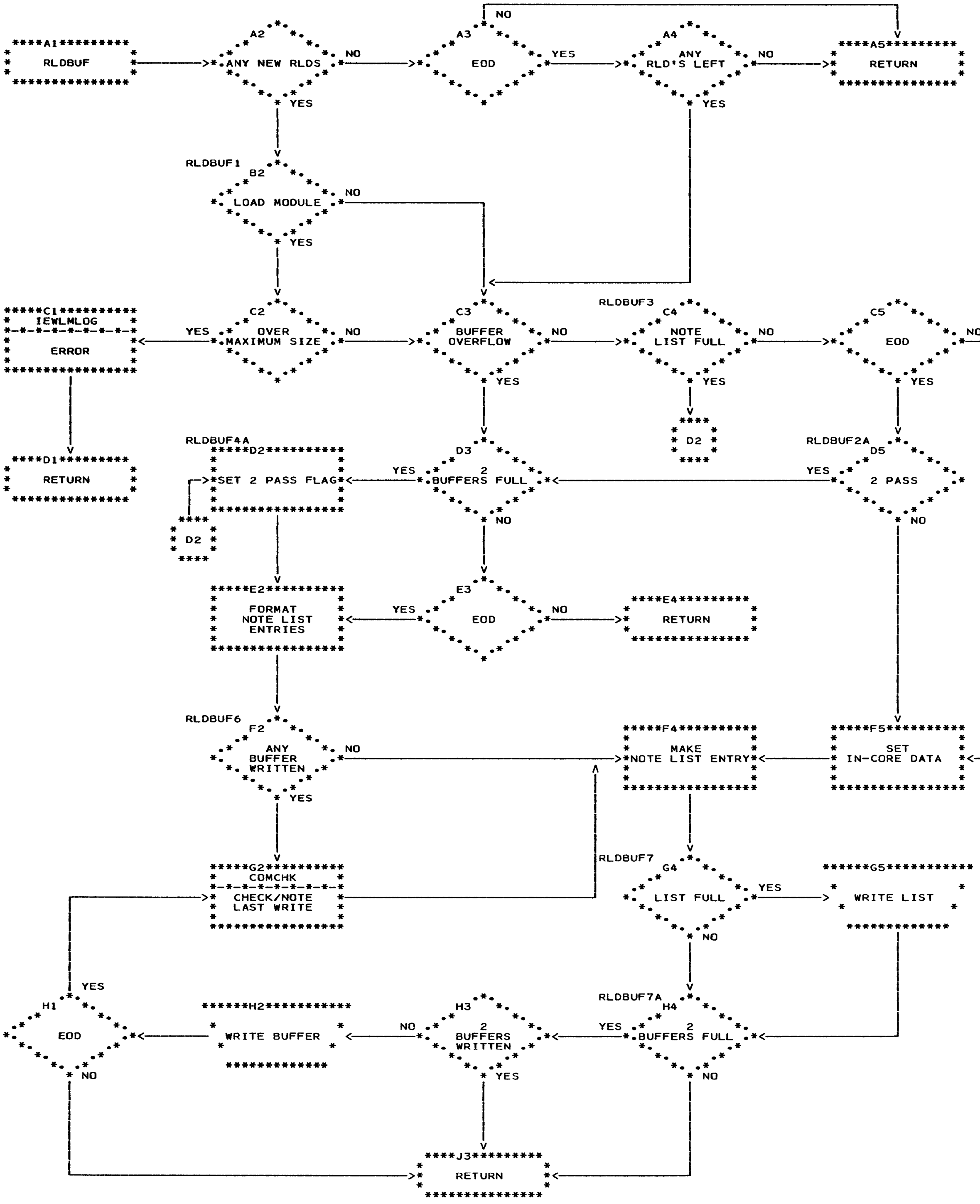


Chart JN. End Processor (IEWLMEND)

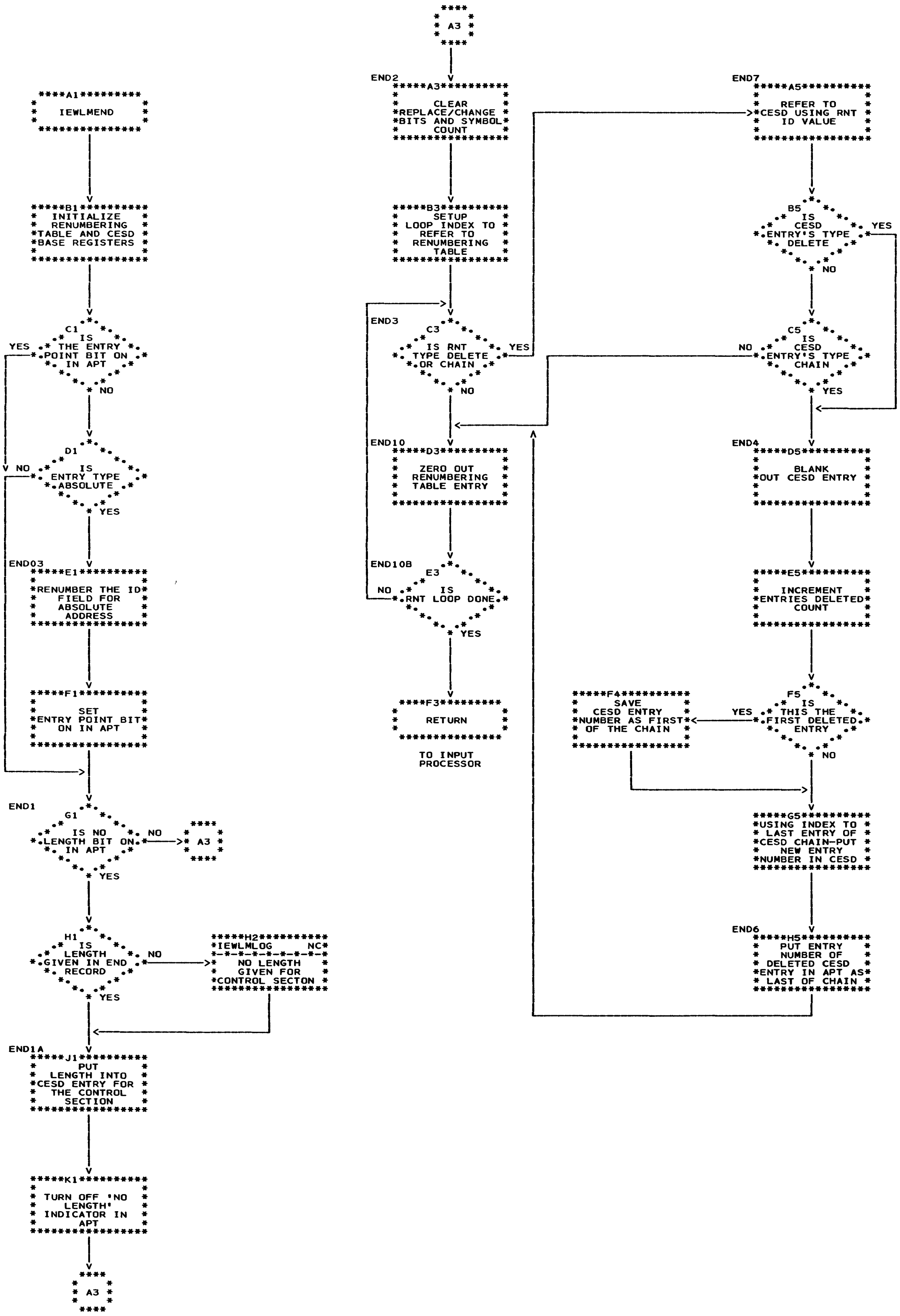

A1
IEWLMEND
B1
INITIALIZE RENUMBERING TABLE AND CESD BASE REGISTERS
C1
IS THE ENTRY POINT BIT ON IN APT
YES
NO
D1
IS ENTRY TYPE ABSOLUTE
NO
YES
END03
E1
RENUMBER THE ID FIELD FOR ABSOLUTE ADDRESS
F1
SET ENTRY POINT BIT ON IN APT
END1
G1
IS NO LENGTH BIT ON IN APT
NO
A3
YES
H1
IS LENGTH GIVEN IN END RECORD
NO
H2
IEWLMLOG NC
NO LENGTH GIVEN FOR CONTROL SECTON
YES
END1A
J1
PUT LENGTH INTO CESD ENTRY FOR THE CONTROL SECTION
K1
TURN OFF 'NO LENGTH' INDICATOR IN APT
A3
A3
END2
A3
CLEAR REPLACE/CHANGE BITS AND SYMBOL COUNT
B3
SETUP LOOP INDEX TO REFER TO RENUMBERING TABLE
END3
C3
IS RNT TYPE DELETE OR CHAIN
YES
NO
END10
D3
ZERO OUT RENUMBERING TABLE ENTRY
END10B
E3
IS RNT LOOP DONE
NO
YES
F3
RETURN
TO INPUT PROCESSOR
END7
A5
REFER TO CESD USING RNT ID VALUE
B5
IS CESD ENTRY'S TYPE DELETE
YES
NO
C5
IS CESD ENTRY'S TYPE CHAIN
NO
YES
END4
D5
BLANK OUT CESD ENTRY
E5
INCREMENT ENTRIES DELETED COUNT
F5
IS THIS THE FIRST DELETED ENTRY
YES
NO
F4
SAVE CESD ENTRY NUMBER AS FIRST OF THE CHAIN
G5
USING INDEX TO LAST ENTRY OF CESD CHAIN-PUT NEW ENTRY NUMBER IN CESD
END6
H5
PUT ENTRY NUMBER OF DELETED CESD ENTRY IN APT AS LAST OF CHAIN

Chart J0. Control Statement Scanner (IEWLMSCN)

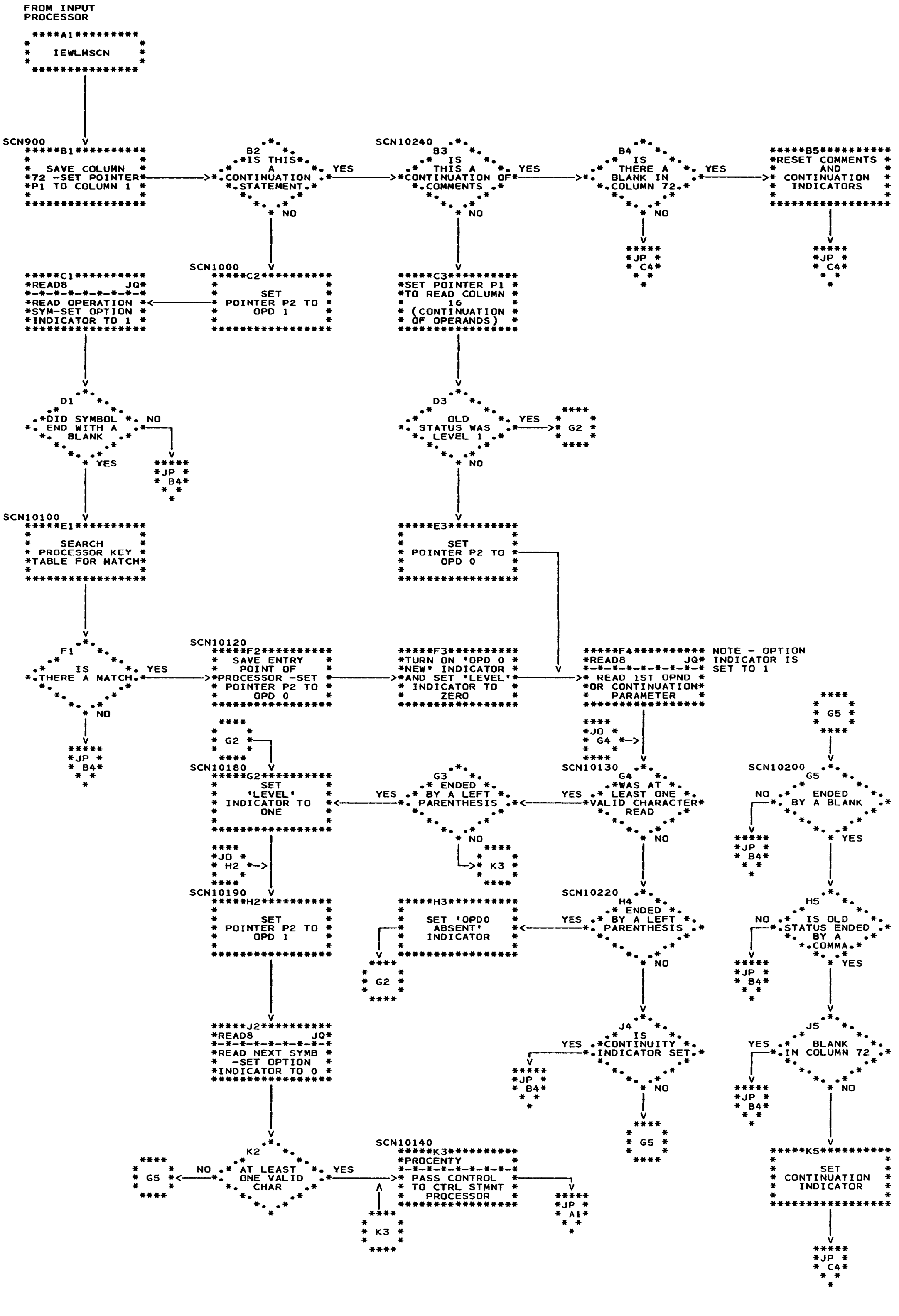

FROM INPUT PROCESSOR
A1
IEWLMSCN
SCN900
B1
SAVE COLUMN 72 -SET POINTER P1 TO COLUMN 1
B2
IS THIS A CONTINUATION STATEMENT
YES
NO
SCN10240
B3
IS THIS A CONTINUATION OF COMMENTS
YES
NO
B4
IS THERE A BLANK IN COLUMN 72
YES
NO
JP C4
B5
RESET COMMENTS AND CONTINUATION INDICATORS
JP C4
C1
READ8 JQ
READ OPERATION SYM-SET OPTION INDICATOR TO 1
SCN1000
C2
SET POINTER P2 TO OPD 1
C3
SET POINTER P1 TO READ COLUMN 16 (CONTINUATION OF OPERANDS)
D1
DID SYMBOL END WITH A BLANK
NO
YES
JP B4
D3
OLD STATUS WAS LEVEL 1
YES
G2
NO
SCN10100
E1
SEARCH PROCESSOR KEY TABLE FOR MATCH
E3
SET POINTER P2 TO OPD 0
F1
IS THERE A MATCH
YES
NO
JP B4
SCN10120
F2
SAVE ENTRY POINT OF PROCESSOR -SET POINTER P2 TO OPD 0
F3
TURN ON 'OPD 0 NEW' INDICATOR AND SET 'LEVEL' INDICATOR TO ZERO
F4
READ8 JQ
READ 1ST OPND OR CONTINUATION PARAMETER
NOTE - OPTION INDICATOR IS SET TO 1
G2
SCN10180
G2
SET 'LEVEL' INDICATOR TO ONE
G3
ENDED BY A LEFT PARENTHESIS
YES
NO
K3
JO G4
SCN10130
G4
WAS AT LEAST ONE VALID CHARACTER READ
YES
NO
G5
SCN10200
G5
ENDED BY A BLANK
NO
YES
JP B4
JO H2
SCN10190
H2
SET POINTER P2 TO OPD 1
H3
SET 'OPDO ABSENT' INDICATOR
G2
SCN10220
H4
ENDED BY A LEFT PARENTHESIS
YES
NO
H5
IS OLD STATUS ENDED BY A COMMA
NO
YES
JP B4
J2
READ8 JQ
READ NEXT SYMB -SET OPTION INDICATOR TO 0
J4
IS CONTINUITY INDICATOR SET
YES
NO
JP B4
G5
J5
BLANK IN COLUMN 72
YES
NO
JP B4
K2
AT LEAST ONE VALID CHAR
NO
YES
G5
SCN10140
K3
PROCENTY
PASS CONTROL TO CTRL STMNT PROCESSOR
JP A1
K3
K5
SET CONTINUATION INDICATOR
JP C4

Chart JP. Control Statement Scanner (IEWLMSCN) (Continued)

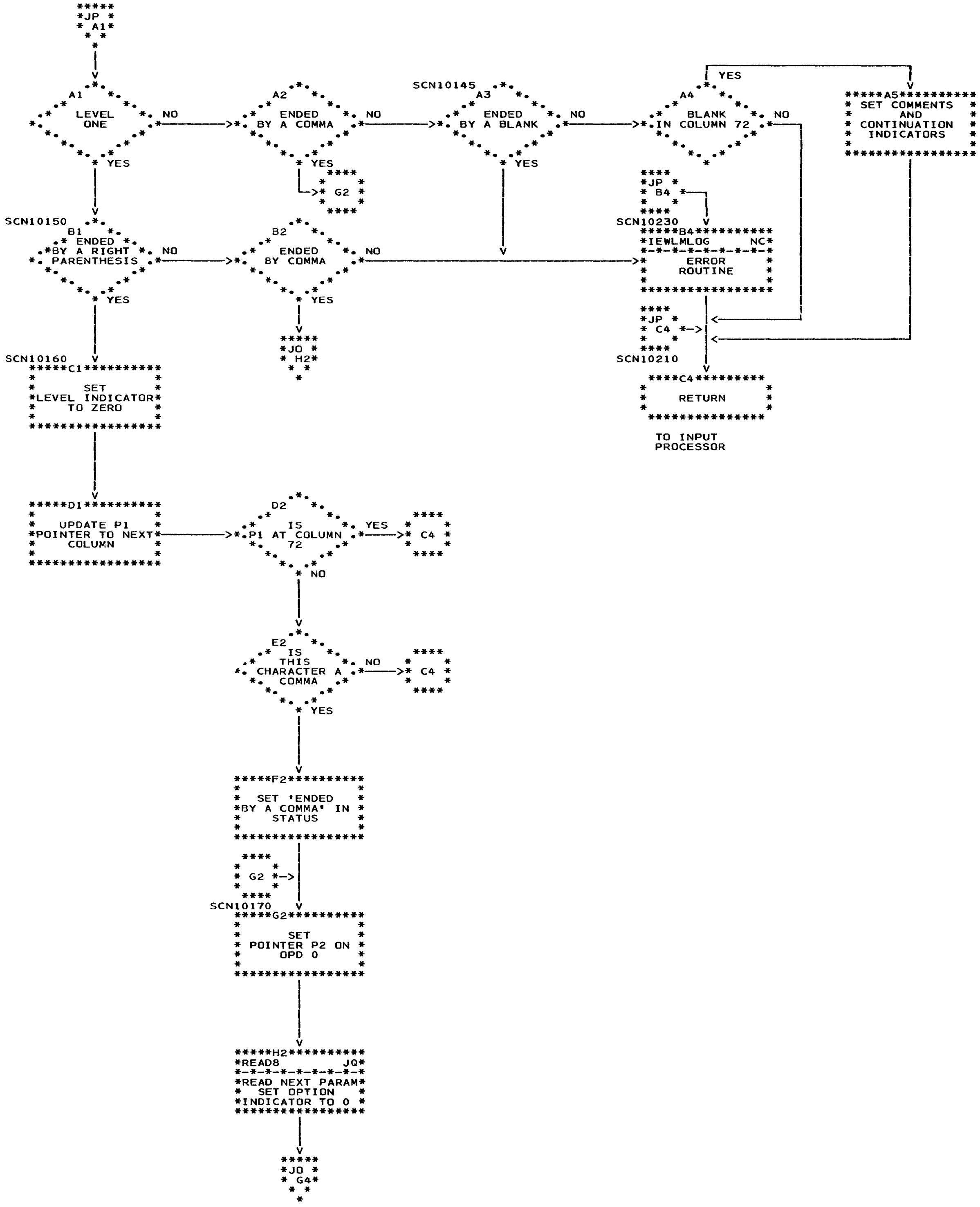


Chart JQ. READ8 Routine

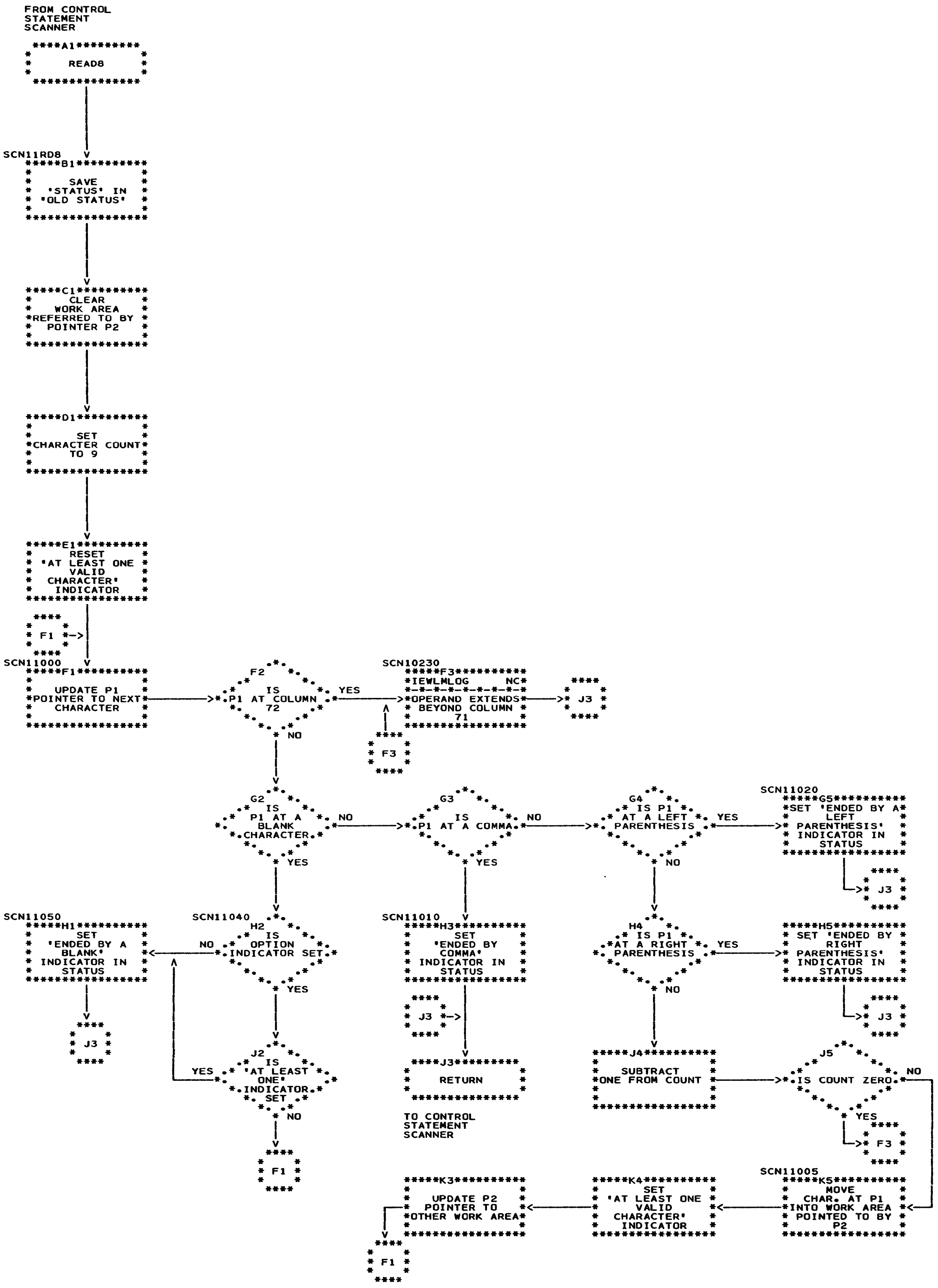

FROM CONTROL STATEMENT SCANNER
A1
READ8
SCN11RD8
B1
SAVE 'STATUS' IN 'OLD STATUS'
C1
CLEAR WORK AREA REFERRED TO BY POINTER P2
D1
SET CHARACTER COUNT TO 9
E1
RESET 'AT LEAST ONE VALID CHARACTER' INDICATOR
F1
SCN11000
F1
UPDATE P1 POINTER TO NEXT CHARACTER
F2
IS P1 AT COLUMN 72
YES
NO
SCN10230
F3
IEWLMLOG NC
OPERAND EXTENDS BEYOND COLUMN 71
J3
F3
G2
IS P1 AT A BLANK CHARACTER
NO
YES
G3
IS P1 AT A COMMA
NO
YES
G4
IS P1 AT A LEFT PARENTHESIS
YES
NO
SCN11020
G5
SET 'ENDED BY A LEFT PARENTHESIS' INDICATOR IN STATUS
J3
SCN11050
H1
SET 'ENDED BY A BLANK' INDICATOR IN STATUS
J3
SCN11040
H2
IS OPTION INDICATOR SET
NO
YES
SCN11010
H3
SET 'ENDED BY COMMA' INDICATOR IN STATUS
J3
H4
IS P1 AT A RIGHT PARENTHESIS
YES
NO
H5
SET 'ENDED BY RIGHT PARENTHESIS' INDICATOR IN STATUS
J3
J2
IS 'AT LEAST ONE' INDICATOR SET
YES
NO
F1
J3
RETURN
TO CONTROL STATEMENT SCANNER
J4
SUBTRACT ONE FROM COUNT
J5
IS COUNT ZERO
NO
YES
F3
SCN11005
K5
MOVE CHAR. AT P1 INTO WORK AREA POINTED TO BY P2
K4
SET 'AT LEAST ONE VALID CHARACTER' INDICATOR
K3
UPDATE P2 POINTER TO OTHER WORK AREA
F1

Chart JR. Include Processor (IEWLMINC)

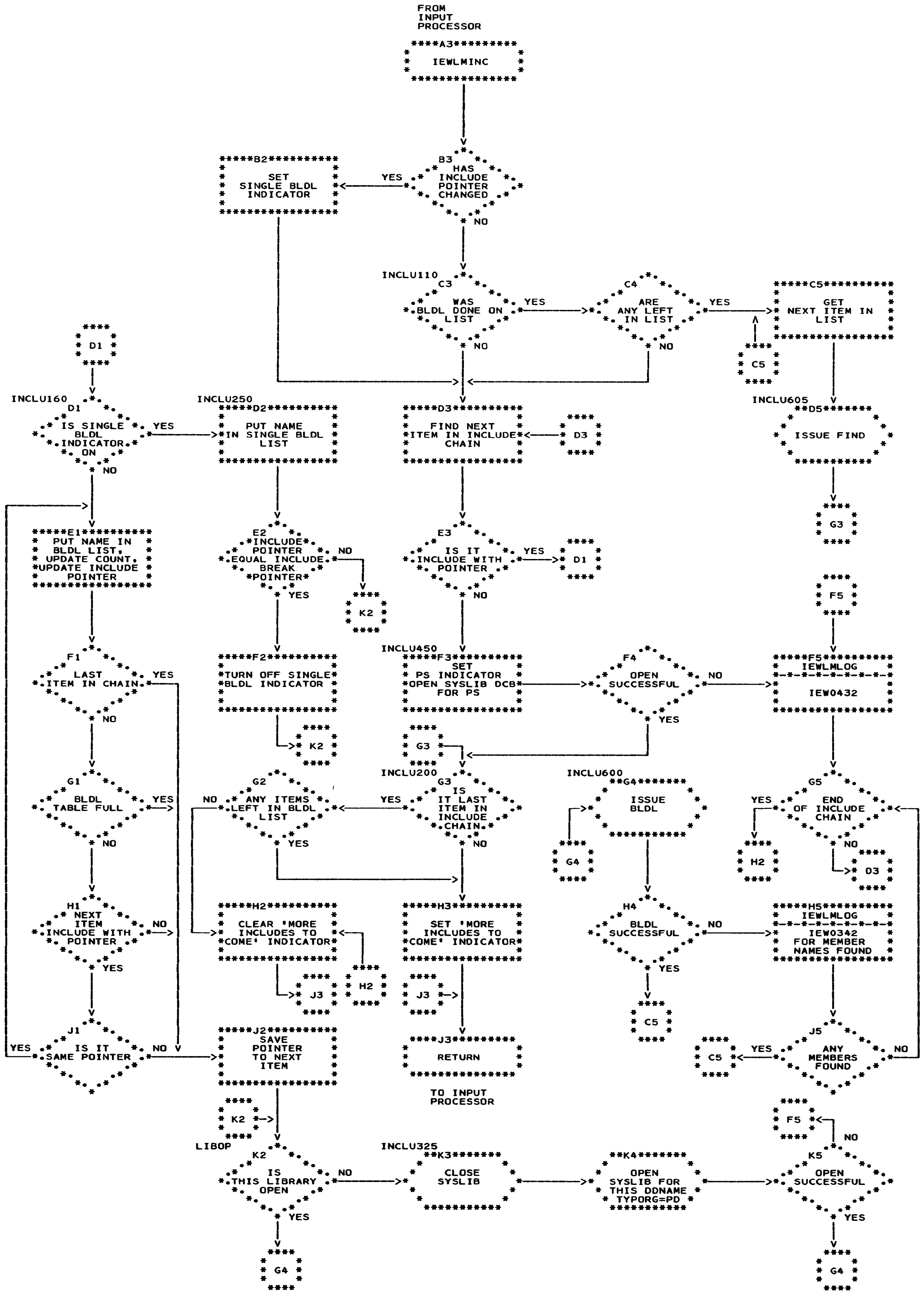

FROM INPUT PROCESSOR
A3 IEWLMINC
B3 HAS INCLUDE POINTER CHANGED
YES
NO
B2 SET SINGLE BLDL INDICATOR
INCLU110
C3 WAS BLDL DONE ON LIST
YES
NO
C4 ARE ANY LEFT IN LIST
YES
NO
C5 GET NEXT ITEM IN LIST
C5
D1
INCLU160
D1 IS SINGLE BLDL INDICATOR ON
YES
NO
INCLU250
D2 PUT NAME IN SINGLE BLDL LIST
D3 FIND NEXT ITEM IN INCLUDE CHAIN
D3
INCLU605
D5 ISSUE FIND
G3
E1 PUT NAME IN BLDL LIST, UPDATE COUNT, UPDATE INCLUDE POINTER
E2 INCLUDE POINTER EQUAL INCLUDE BREAK POINTER
NO
YES
K2
E3 IS IT INCLUDE WITH POINTER
YES
NO
D1
F5
F1 LAST ITEM IN CHAIN
YES
NO
F2 TURN OFF SINGLE BLDL INDICATOR
K2
INCLU450
F3 SET PS INDICATOR OPEN SYSLIB DCB FOR PS
F4 OPEN SUCCESSFUL
NO
YES
F5 IEWLMLOG IEW0432
G3
G1 BLDL TABLE FULL
YES
NO
G2 ANY ITEMS LEFT IN BLDL LIST
NO
YES
INCLU200
G3 IS IT LAST ITEM IN INCLUDE CHAIN
YES
NO
INCLU600
G4 ISSUE BLDL
G4
G5 END OF INCLUDE CHAIN
YES
NO
H2
D3
H1 NEXT ITEM INCLUDE WITH POINTER
NO
YES
H2 CLEAR 'MORE INCLUDES TO COME' INDICATOR
H3 SET 'MORE INCLUDES TO COME' INDICATOR
H4 BLDL SUCCESSFUL
NO
YES
H5 IEWLMLOG IEW0342 FOR MEMBER NAMES FOUND
J3
H2
J3
C5
J1 IS IT SAME POINTER
YES
NO
J2 SAVE POINTER TO NEXT ITEM
J3 RETURN
TO INPUT PROCESSOR
C5
J5 ANY MEMBERS FOUND
YES
NO
K2
F5
LIBOP
K2 IS THIS LIBRARY OPEN
NO
YES
INCLU325
K3 CLOSE SYSLIB
K4 OPEN SYSLIB FOR THIS DDNAME TYPORG=PD
K5 OPEN SUCCESSFUL
NO
YES
G4
G4

Chart JS. Automatic Library Call Processor (IEWLCAUT)

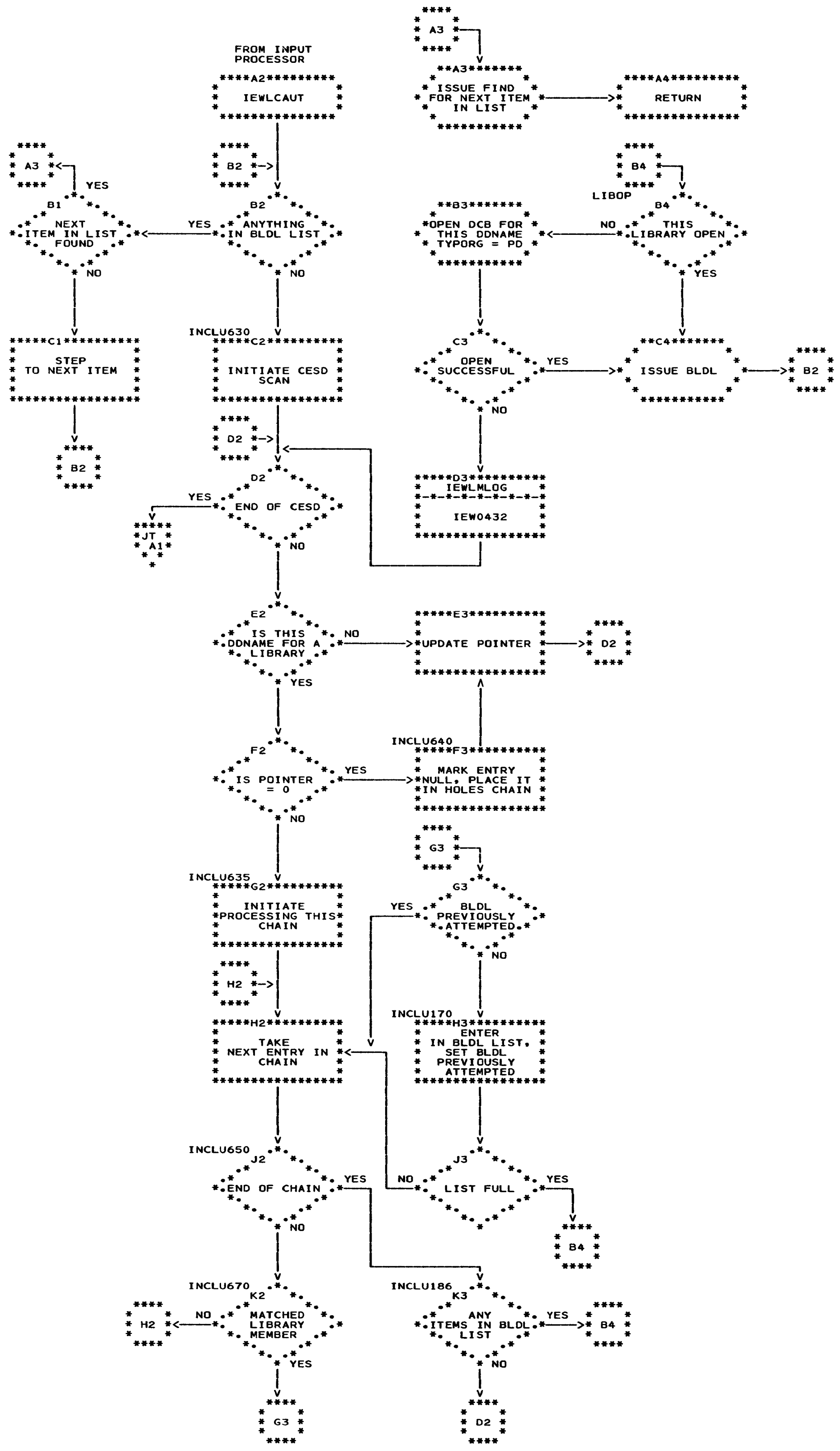


Chart JT. Automatic Library Call Processor (IEWLCAUT) (Continued)

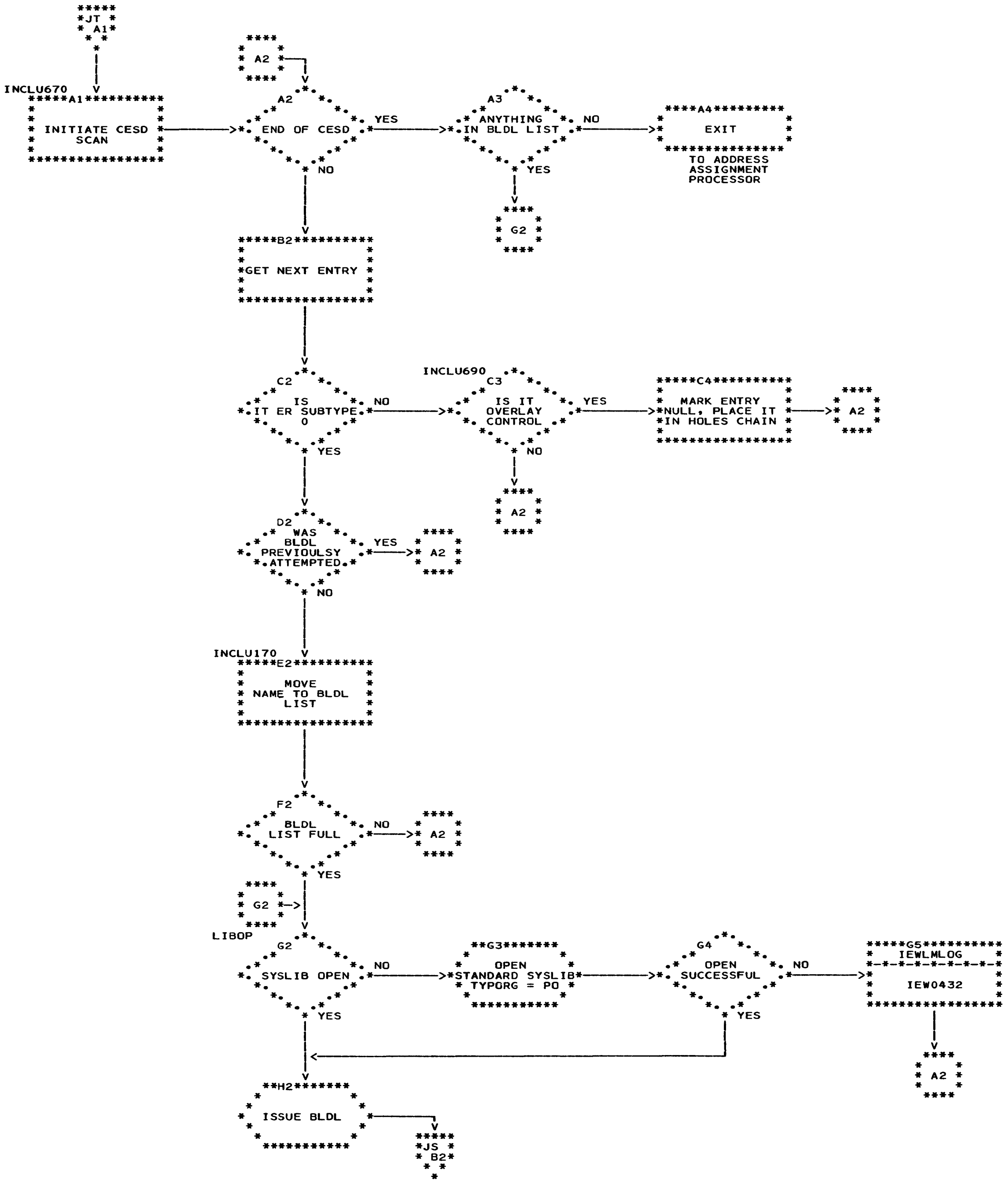


Chart KA. Address Assignment Processor (IEWLMADA)

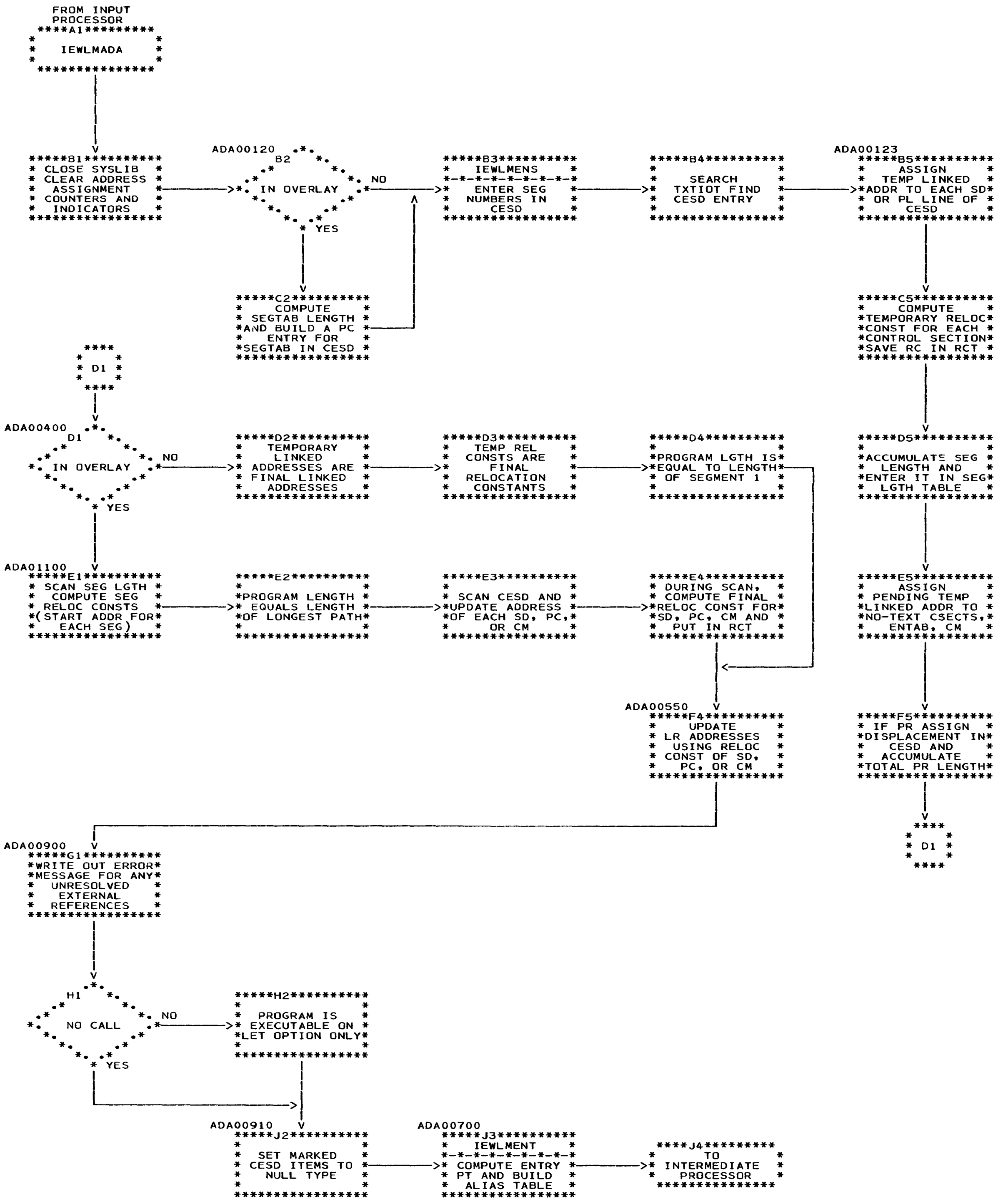

FROM INPUT PROCESSOR
A1
IEWLMADA
B1
CLOSE SYSLIB CLEAR ADDRESS ASSIGNMENT COUNTERS AND INDICATORS
ADA00120
B2
IN OVERLAY
NO
YES
C2
COMPUTE SEGTAB LENGTH AND BUILD A PC ENTRY FOR SEGTAB IN CESD
B3
IEWLMENS
ENTER SEG NUMBERS IN CESD
B4
SEARCH TXTIOT FIND CESD ENTRY
ADA00123
B5
ASSIGN TEMP LINKED ADDR TO EACH SD OR PL LINE OF CESD
C5
COMPUTE TEMPORARY RELOC CONST FOR EACH CONTROL SECTION SAVE RC IN RCT
D1
ADA00400
D1
IN OVERLAY
NO
YES
D2
TEMPORARY LINKED ADDRESSES ARE FINAL LINKED ADDRESSES
D3
TEMP REL CONSTS ARE FINAL RELOCATION CONSTANTS
D4
PROGRAM LGTH IS EQUAL TO LENGTH OF SEGMENT 1
D5
ACCUMULATE SEG LENGTH AND ENTER IT IN SEG LGTH TABLE
ADA01100
E1
SCAN SEG LGTH COMPUTE SEG RELOC CONSTS (START ADDR FOR EACH SEG)
E2
PROGRAM LENGTH EQUALS LENGTH OF LONGEST PATH
E3
SCAN CESD AND UPDATE ADDRESS OF EACH SD, PC, OR CM
E4
DURING SCAN, COMPUTE FINAL RELOC CONST FOR SD, PC, CM AND PUT IN RCT
E5
ASSIGN PENDING TEMP LINKED ADDR TO NO-TEXT CSECTS, ENTAB, CM
ADA00550
F4
UPDATE LR ADDRESSES USING RELOC CONST OF SD, PC, OR CM
F5
IF PR ASSIGN DISPLACEMENT IN CESD AND ACCUMULATE TOTAL PR LENGTH
D1
ADA00900
G1
WRITE OUT ERROR MESSAGE FOR ANY UNRESOLVED EXTERNAL REFERENCES
H1
NO CALL
NO
YES
H2
PROGRAM IS EXECUTABLE ON LET OPTION ONLY
ADA00910
J2
SET MARKED CESD ITEMS TO NULL TYPE
ADA00700
J3
IEWLMENT
COMPUTE ENTRY PT AND BUILD ALIAS TABLE
J4
TO INTERMEDIATE PROCESSOR

Chart KB. IEWLMENS

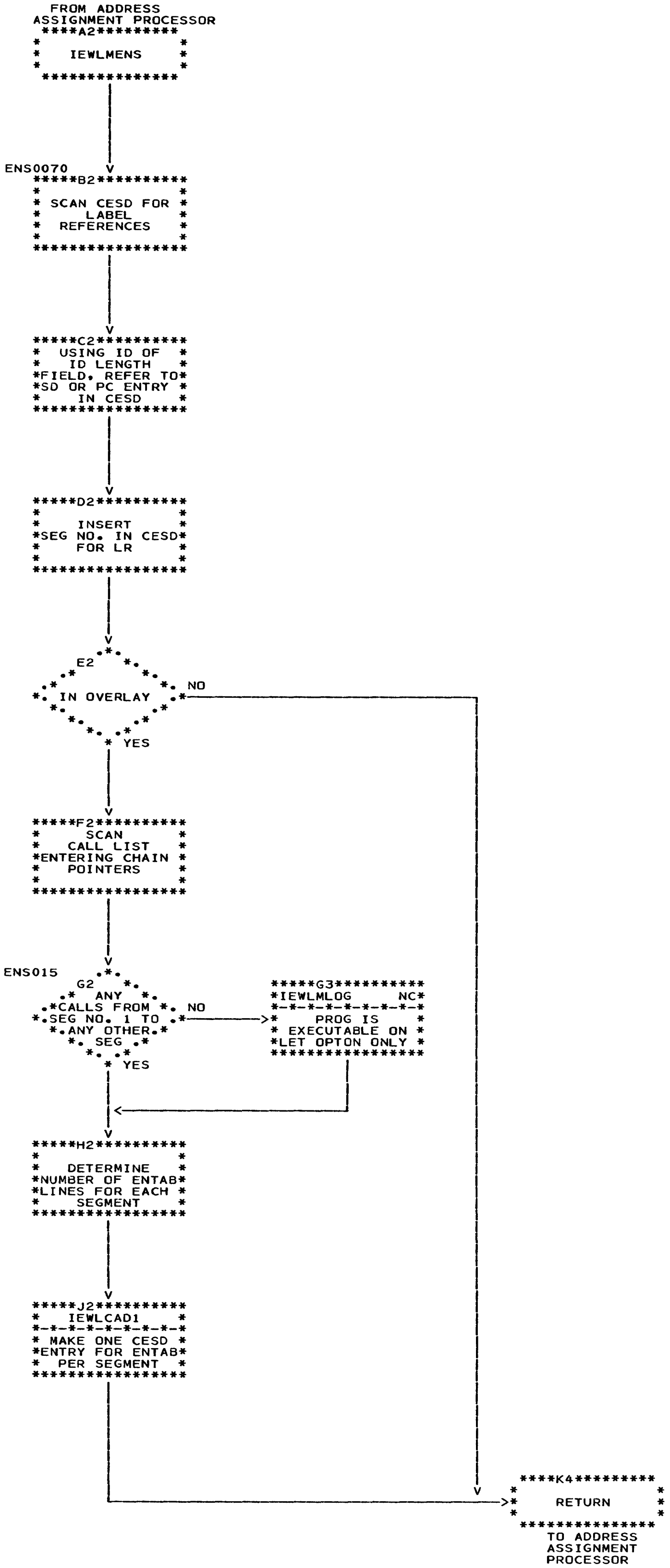

FROM ADDRESS ASSIGNMENT PROCESSOR
A2
IEWLMENS
ENS0070
B2
SCAN CESD FOR LABEL REFERENCES
C2
USING ID OF ID LENGTH FIELD, REFER TO SD OR PC ENTRY IN CESD
D2
INSERT SEG NO. IN CESD FOR LR
E2
IN OVERLAY
NO
YES
F2
SCAN CALL LIST ENTERING CHAIN POINTERS
ENS015
G2
ANY CALLS FROM SEG NO. 1 TO ANY OTHER. SEG
NO
YES
G3
IEWLMLOG NC
PROG IS EXECUTABLE ON LET OPTON ONLY
H2
DETERMINE NUMBER OF ENTAB LINES FOR EACH SEGMENT
J2
IEWLCAD1
MAKE ONE CESD ENTRY FOR ENTAB PER SEGMENT
K4
RETURN
TO ADDRESS ASSIGNMENT PROCESSOR

Chart KC. Entry Processor (IEWLMENT)

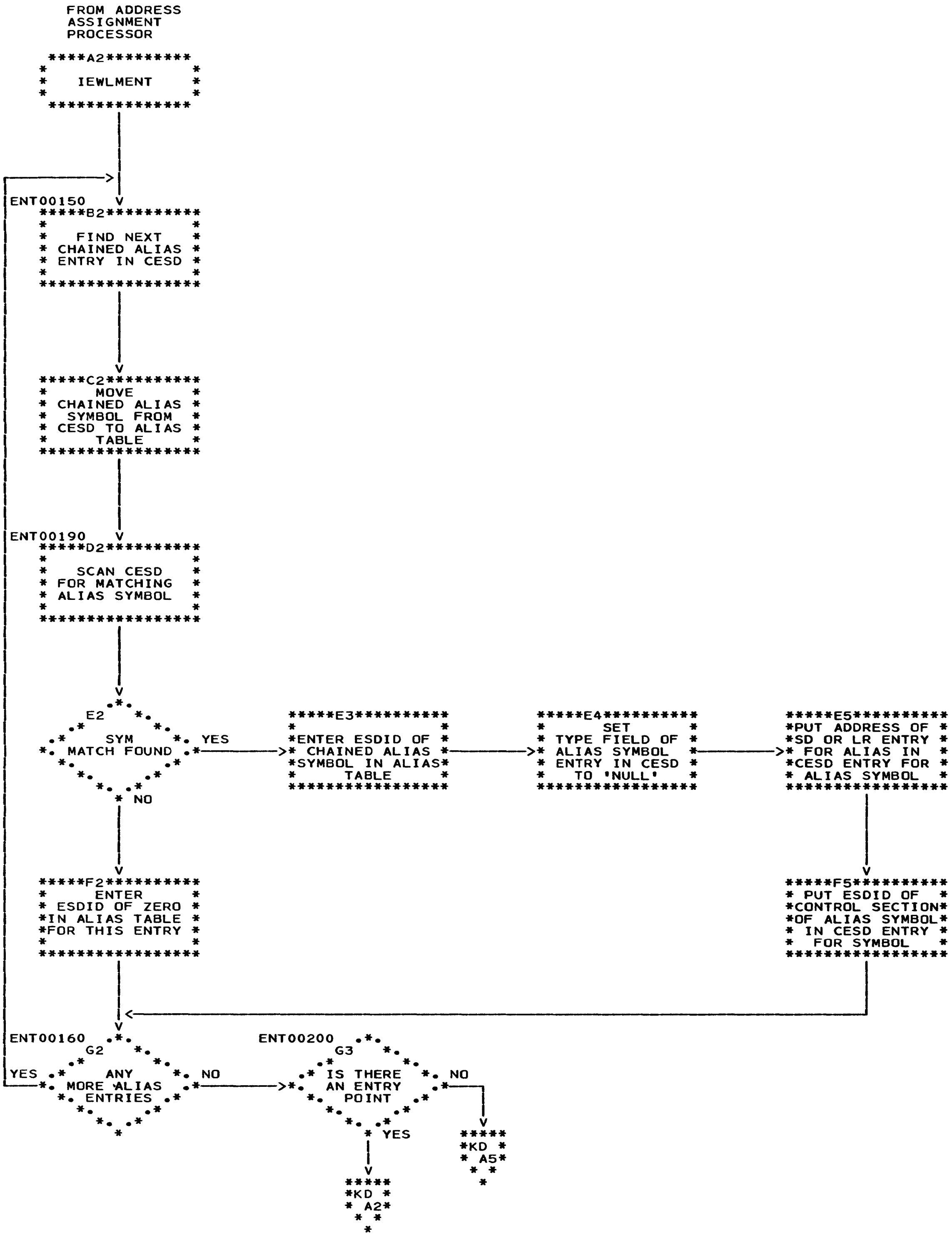

FROM ADDRESS ASSIGNMENT PROCESSOR
A2
IEWLMENT
ENT00150
B2
FIND NEXT CHAINED ALIAS ENTRY IN CESD
C2
MOVE CHAINED ALIAS SYMBOL FROM CESD TO ALIAS TABLE
ENT00190
D2
SCAN CESD FOR MATCHING ALIAS SYMBOL
E2
SYM MATCH FOUND
YES
NO
E3
ENTER ESDID OF CHAINED ALIAS SYMBOL IN ALIAS TABLE
E4
SET TYPE FIELD OF ALIAS SYMBOL ENTRY IN CESD TO 'NULL'
E5
PUT ADDRESS OF SD OR LR ENTRY FOR ALIAS IN CESD ENTRY FOR ALIAS SYMBOL
F2
ENTER ESDID OF ZERO IN ALIAS TABLE FOR THIS ENTRY
F5
PUT ESDID OF CONTROL SECTION OF ALIAS SYMBOL IN CESD ENTRY FOR SYMBOL
ENT00160
G2
ANY MORE ALIAS ENTRIES
YES
NO
ENT00200
G3
IS THERE AN ENTRY POINT
NO
YES
KD A5
KD A2

Chart KD. Entry Processor (IEWLMENT) (Continued)

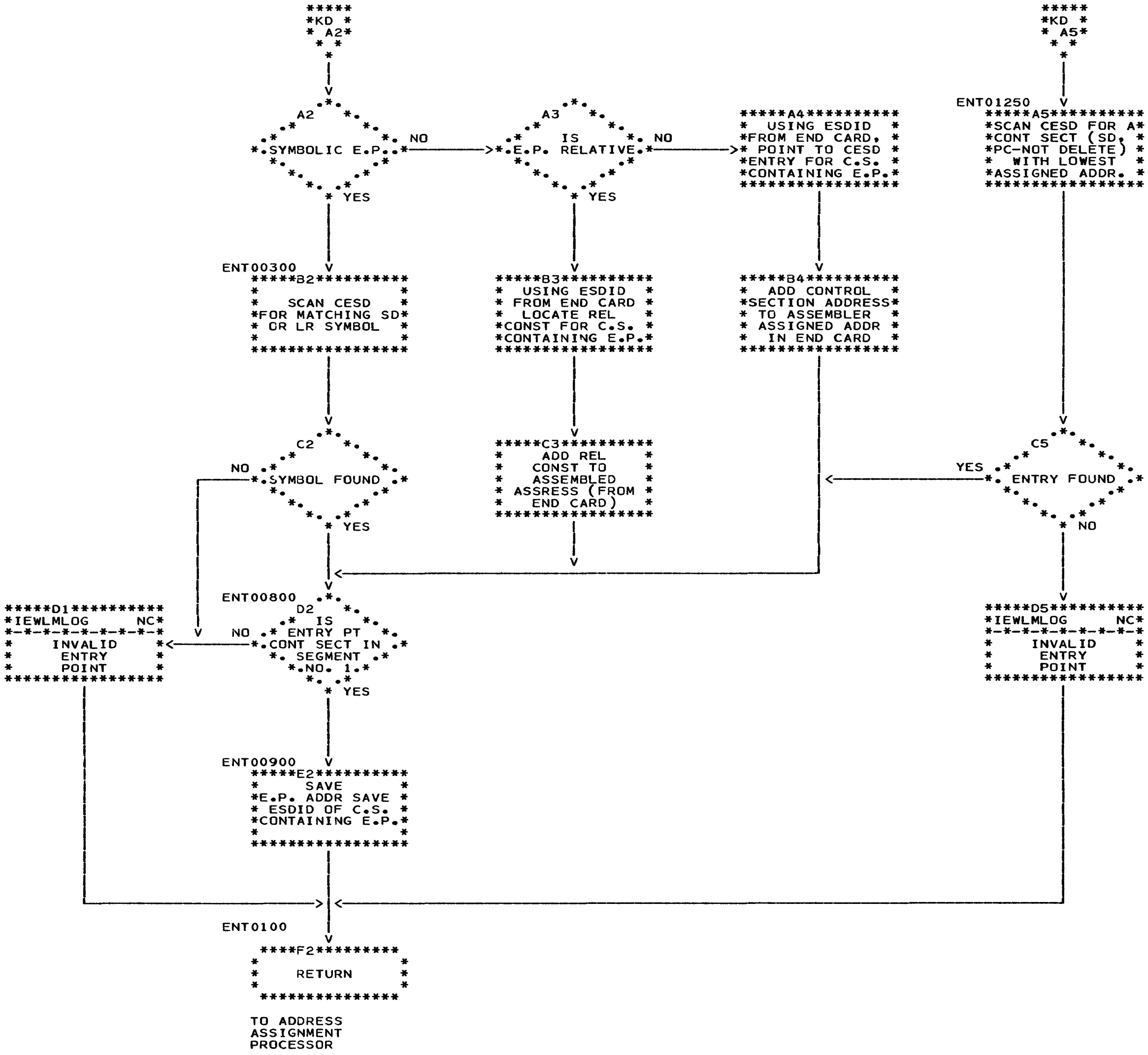

KD A2
A2 SYMBOLIC E.P.
NO
YES
A3 IS E.P. RELATIVE
NO
YES
A4 USING ESDID FROM END CARD, POINT TO CESD ENTRY FOR C.S. CONTAINING E.P.
KD A5
ENT01250
A5 SCAN CESD FOR A CONT SECT (SD, PC-NOT DELETE) WITH LOWEST ASSIGNED ADDR.
ENT00300
B2 SCAN CESD FOR MATCHING SD OR LR SYMBOL
B3 USING ESDID FROM END CARD LOCATE REL CONST FOR C.S. CONTAINING E.P.
B4 ADD CONTROL SECTION ADDRESS TO ASSEMBLER ASSIGNED ADDR IN END CARD
C2 SYMBOL FOUND
NO
YES
C3 ADD REL CONST TO ASSEMBLED ASSRESS (FROM END CARD)
C5 ENTRY FOUND
YES
NO
D1 IEWLMLOG NC
INVALID ENTRY POINT
ENT00800
D2 IS ENTRY PT CONT SECT IN SEGMENT NO. 1.
NO
YES
D5 IEWLMLOG NC
INVALID ENTRY POINT
ENT00900
E2 SAVE E.P. ADDR SAVE ESDID OF C.S. CONTAINING E.P.
ENT0100
F2 RETURN
TO ADDRESS ASSIGNMENT PROCESSOR

Chart LA. Intermediate Output Processor (IEWLMOUT)

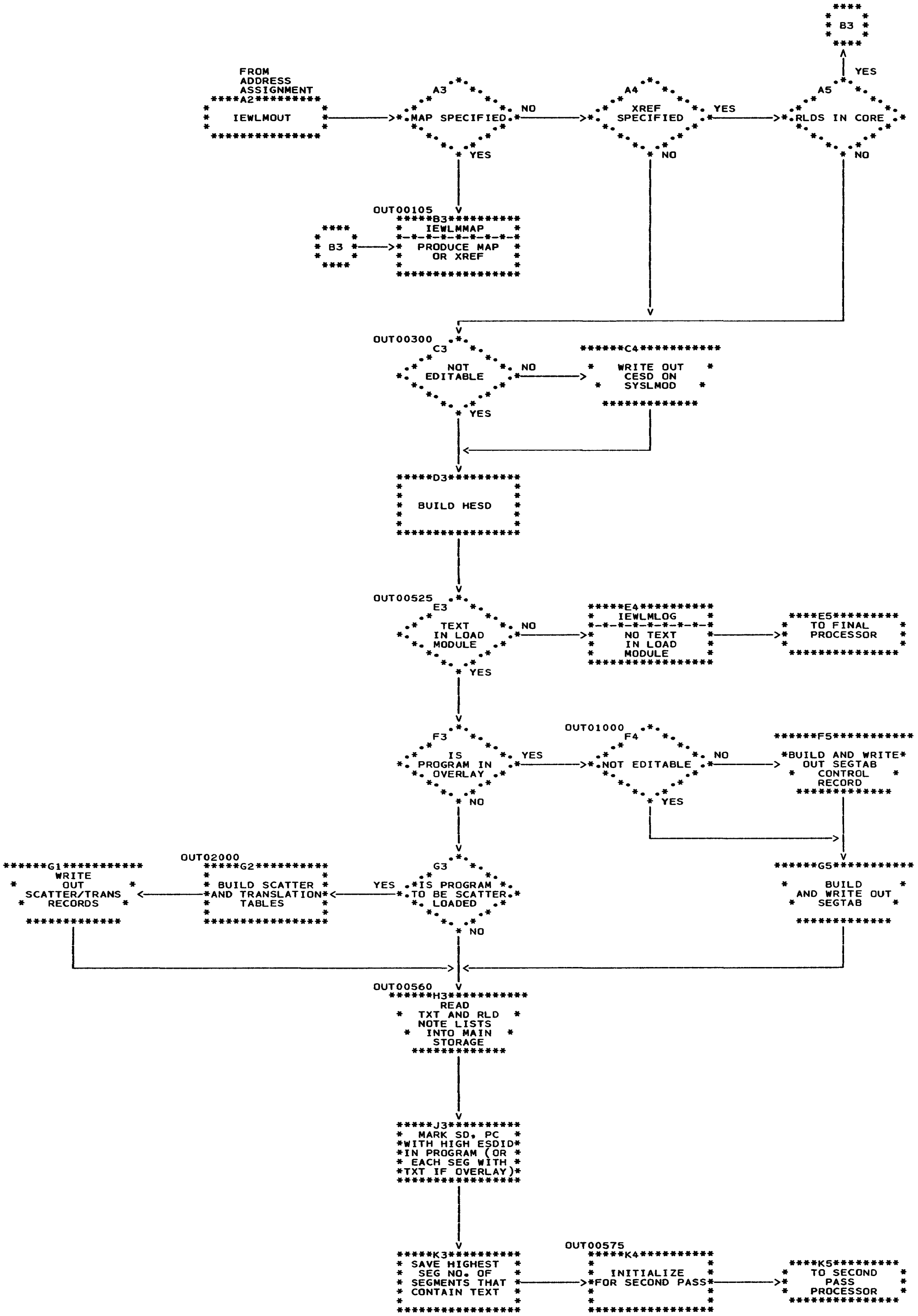


Chart LB. MAP/XREF Processor (IEWLMMAP)

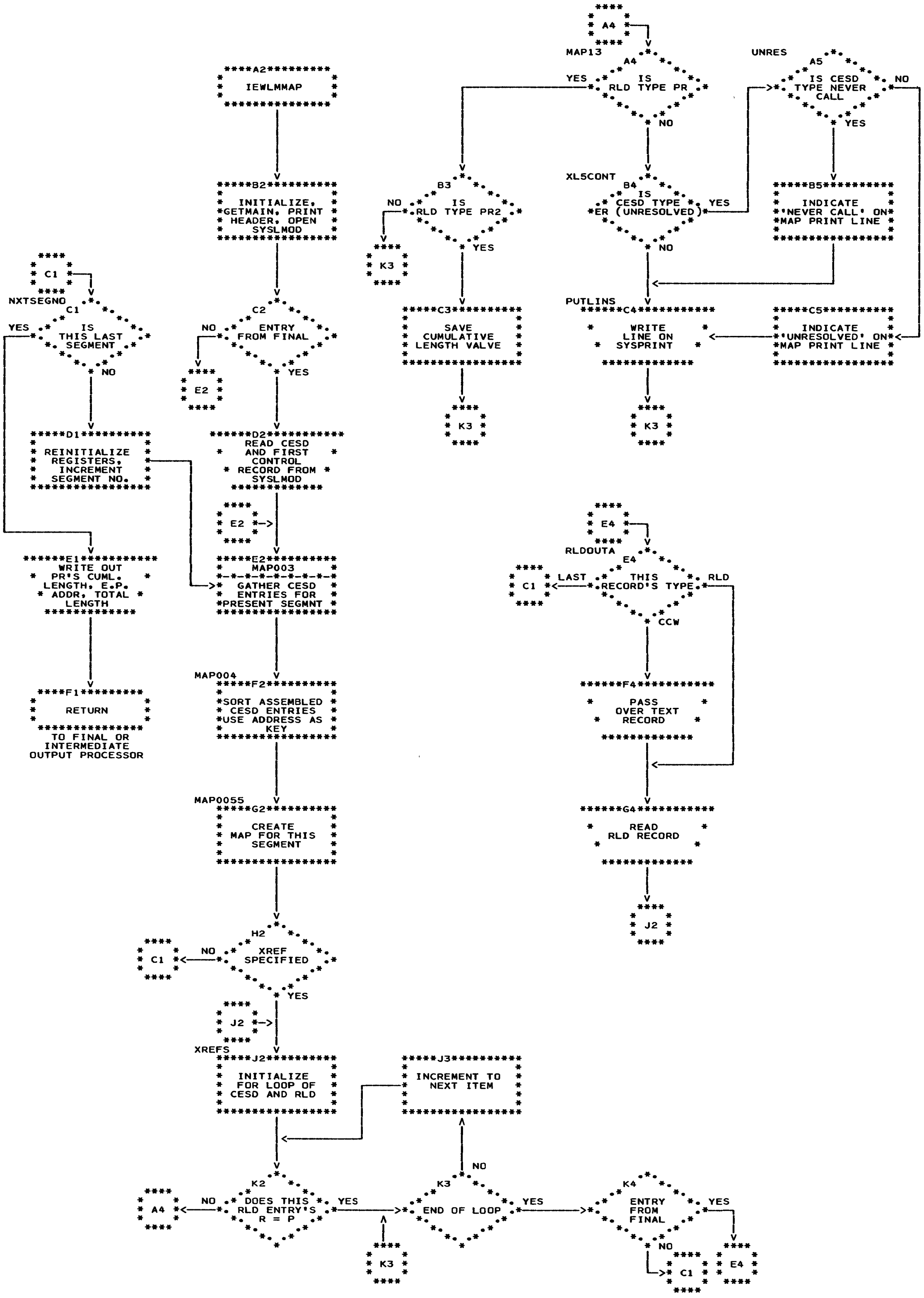

A4
MAP13
A4
IS RLD TYPE PR
YES
NO
UNRES
A5
IS CESD TYPE NEVER CALL
NO
YES
A2
IEWLMMAP
B2
INITIALIZE, GETMAIN, PRINT HEADER, OPEN SYSLMOD
B3
IS RLD TYPE PR2
NO
YES
K3
XL5CONT
B4
IS CESD TYPE ER (UNRESOLVED)
YES
NO
B5
INDICATE 'NEVER CALL' ON MAP PRINT LINE
C1
NXTSEGNO
C1
IS THIS LAST SEGMENT
YES
NO
C2
ENTRY FROM FINAL
NO
YES
E2
C3
SAVE CUMULATIVE LENGTH VALVE
K3
PUTLINS
C4
WRITE LINE ON SYSPRINT
K3
C5
INDICATE 'UNRESOLVED' ON MAP PRINT LINE
D1
REINITIALIZE REGISTERS, INCREMENT SEGMENT NO.
D2
READ CESD AND FIRST CONTROL RECORD FROM SYSLMOD
E2
E1
WRITE OUT PR'S CUML. LENGTH, E.P. ADDR, TOTAL LENGTH
E2
MAP003
GATHER CESD ENTRIES FOR PRESENT SEGMNT
E4
RLDOUTA
E4
THIS RECORD'S TYPE
LAST
C1
RLD
CCW
F1
RETURN
TO FINAL OR INTERMEDIATE OUTPUT PROCESSOR
MAP004
F2
SORT ASSEMBLED CESD ENTRIES USE ADDRESS AS KEY
F4
PASS OVER TEXT RECORD
MAP0055
G2
CREATE MAP FOR THIS SEGMENT
G4
READ RLD RECORD
J2
H2
XREF SPECIFIED
NO
C1
YES
J2
XREFS
J2
INITIALIZE FOR LOOP OF CESD AND RLD
J3
INCREMENT TO NEXT ITEM
K2
DOES THIS RLD ENTRY'S R = P
NO
A4
YES
K3
END OF LOOP
YES
NO
K3
K4
ENTRY FROM FINAL
YES
E4
NO
C1

Chart MA. Second Pass Processor

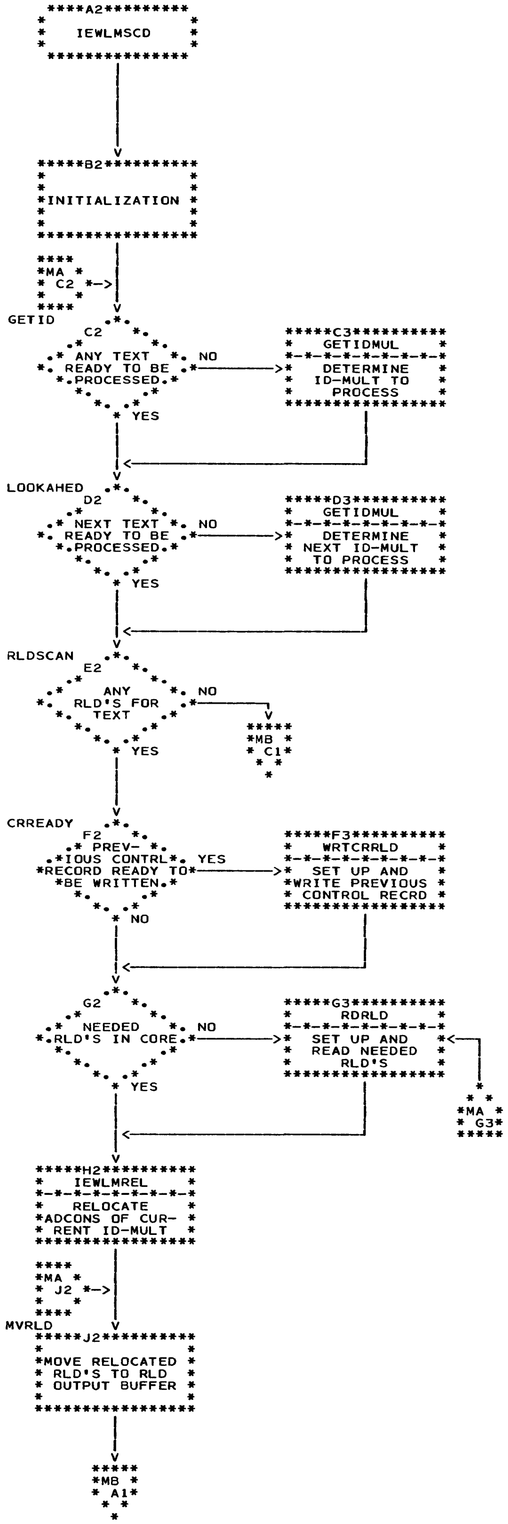

A2
IEWLMSCD
B2
INITIALIZATION
MA C2
GETID
C2
ANY TEXT READY TO BE PROCESSED
NO
YES
C3
GETIDMUL
DETERMINE ID-MULT TO PROCESS
LOOKAHED
D2
NEXT TEXT READY TO BE PROCESSED
NO
YES
D3
GETIDMUL
DETERMINE NEXT ID-MULT TO PROCESS
RLDSCAN
E2
ANY RLD'S FOR TEXT
NO
YES
MB C1
CRREADY
F2
PREV-IOUS CONTRL RECORD READY TO BE WRITTEN
YES
NO
F3
WRTCRRLD
SET UP AND WRITE PREVIOUS CONTROL RECRD
G2
NEEDED RLD'S IN CORE
NO
YES
G3
RDRLD
SET UP AND READ NEEDED RLD'S
MA G3
H2
IEWLMREL
RELOCATE ADCONS OF CUR-RENT ID-MULT
MA J2
MVRLD
J2
MOVE RELOCATED RLD'S TO RLD OUTPUT BUFFER
MB A1

Chart MB. Second Pass Processor (Continued)

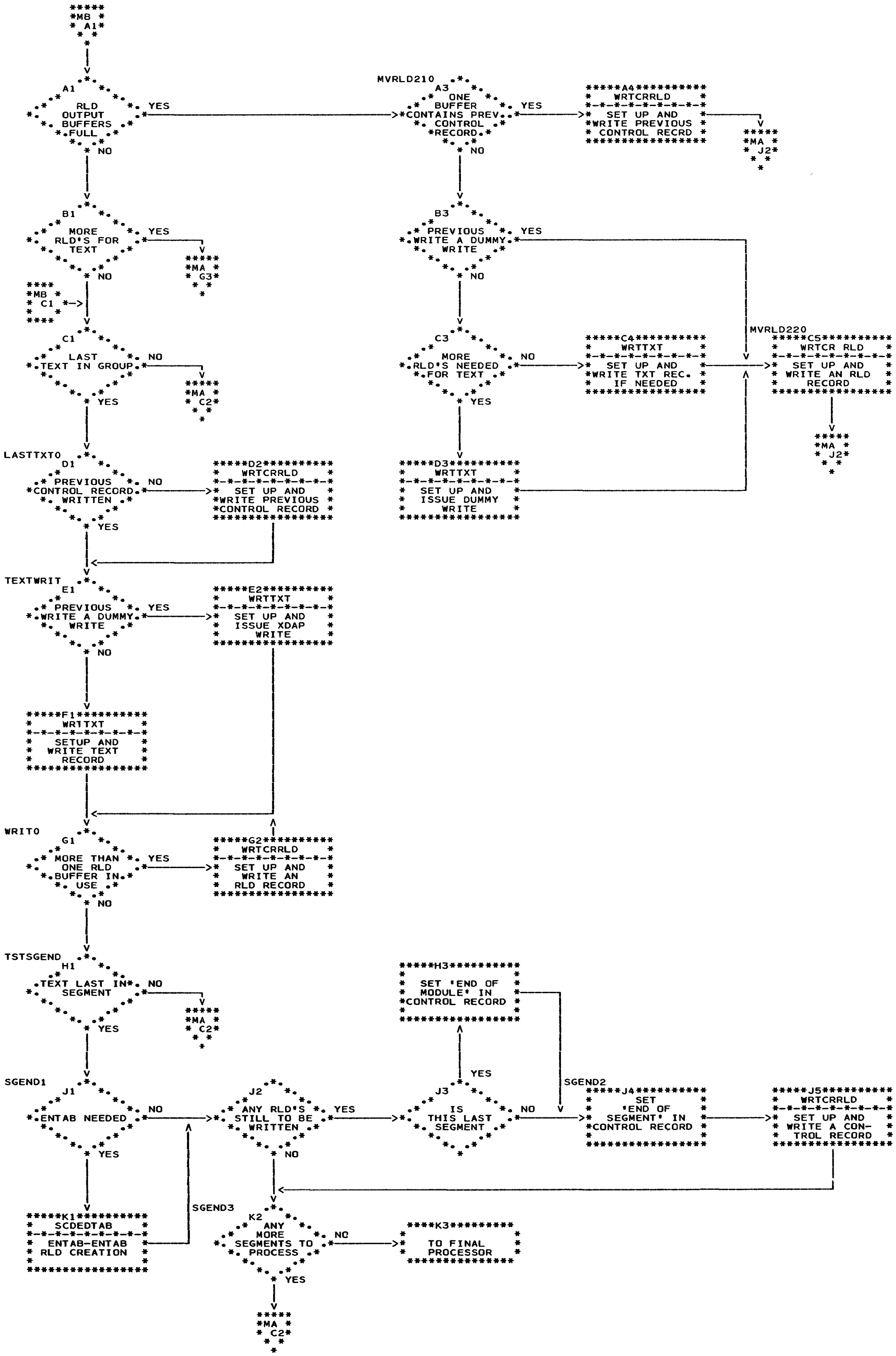

MB A1
A1 RLD OUTPUT BUFFERS FULL
YES
NO
MVRLD210
A3 ONE BUFFER CONTAINS PREV. CONTROL RECORD.
YES
NO
A4 WRTCRRLD
SET UP AND WRITE PREVIOUS CONTROL RECRD
MA J2
B1 MORE RLD'S FOR TEXT
YES
NO
MA G3
MB C1
B3 PREVIOUS WRITE A DUMMY WRITE
YES
NO
C1 LAST TEXT IN GROUP
NO
YES
MA C2
C3 MORE RLD'S NEEDED FOR TEXT
NO
YES
C4 WRTTXT
SET UP AND WRITE TXT REC. IF NEEDED
MVRLD220
C5 WRTCR RLD
SET UP AND WRITE AN RLD RECORD
MA J2
LASTTXTO
D1 PREVIOUS CONTROL RECORD WRITTEN
NO
YES
D2 WRTCRRLD
SET UP AND WRITE PREVIOUS CONTROL RECORD
D3 WRTTXT
SET UP AND ISSUE DUMMY WRITE
TEXTWRIT
E1 PREVIOUS WRITE A DUMMY WRITE
YES
NO
E2 WRTTXT
SET UP AND ISSUE XDAP WRITE
F1 WRITTXT
SETUP AND WRITE TEXT RECORD
WRITO
G1 MORE THAN ONE RLD BUFFER IN USE
YES
NO
G2 WRTCRRLD
SET UP AND WRITE AN RLD RECORD
TSTSGEND
H1 TEXT LAST IN SEGMENT
NO
YES
MA C2
H3
SET 'END OF MODULE' IN CONTROL RECORD
SGEND1
J1 ENTAB NEEDED
NO
YES
J2 ANY RLD'S STILL TO BE WRITTEN
YES
NO
J3 IS THIS LAST SEGMENT
YES
NO
SGEND2
J4 SET 'END OF SEGMENT' IN CONTROL RECORD
J5 WRTCRRLD
SET UP AND WRITE A CON- TROL RECORD
K1 SCDEDTAB
ENTAB-ENTAB RLD CREATION
SGEND3
K2 ANY MORE SEGMENTS TO PROCESS
NO
YES
K3 TO FINAL PROCESSOR
MA C2

Chart MC. GETIDMUL Routine

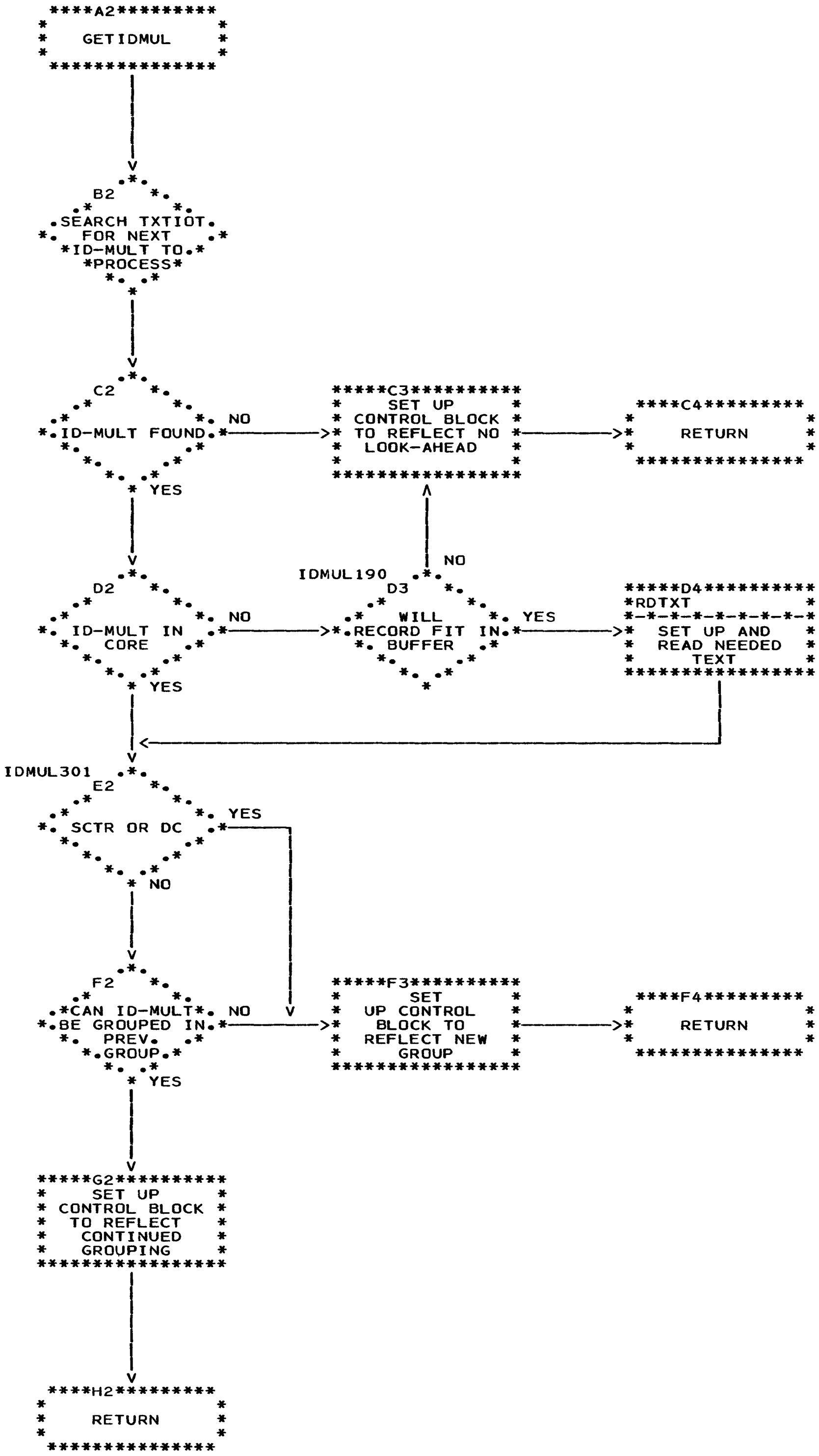

A2
GETIDMUL
B2
SEARCH TXTIOT FOR NEXT ID-MULT TO PROCESS
C2
ID-MULT FOUND
NO
YES
C3
SET UP CONTROL BLOCK TO REFLECT NO LOOK-AHEAD
C4
RETURN
D2
ID-MULT IN CORE
NO
YES
IDMUL190
D3
WILL RECORD FIT IN BUFFER
NO
YES
D4
RDTXT
SET UP AND READ NEEDED TEXT
IDMUL301
E2
SCTR OR DC
YES
NO
F2
CAN ID-MULT BE GROUPED IN PREV. GROUP
NO
YES
F3
SET UP CONTROL BLOCK TO REFLECT NEW GROUP
F4
RETURN
G2
SET UP CONTROL BLOCK TO REFLECT CONTINUED GROUPING
H2
RETURN

Chart MD. TXT/RLD Read Routines

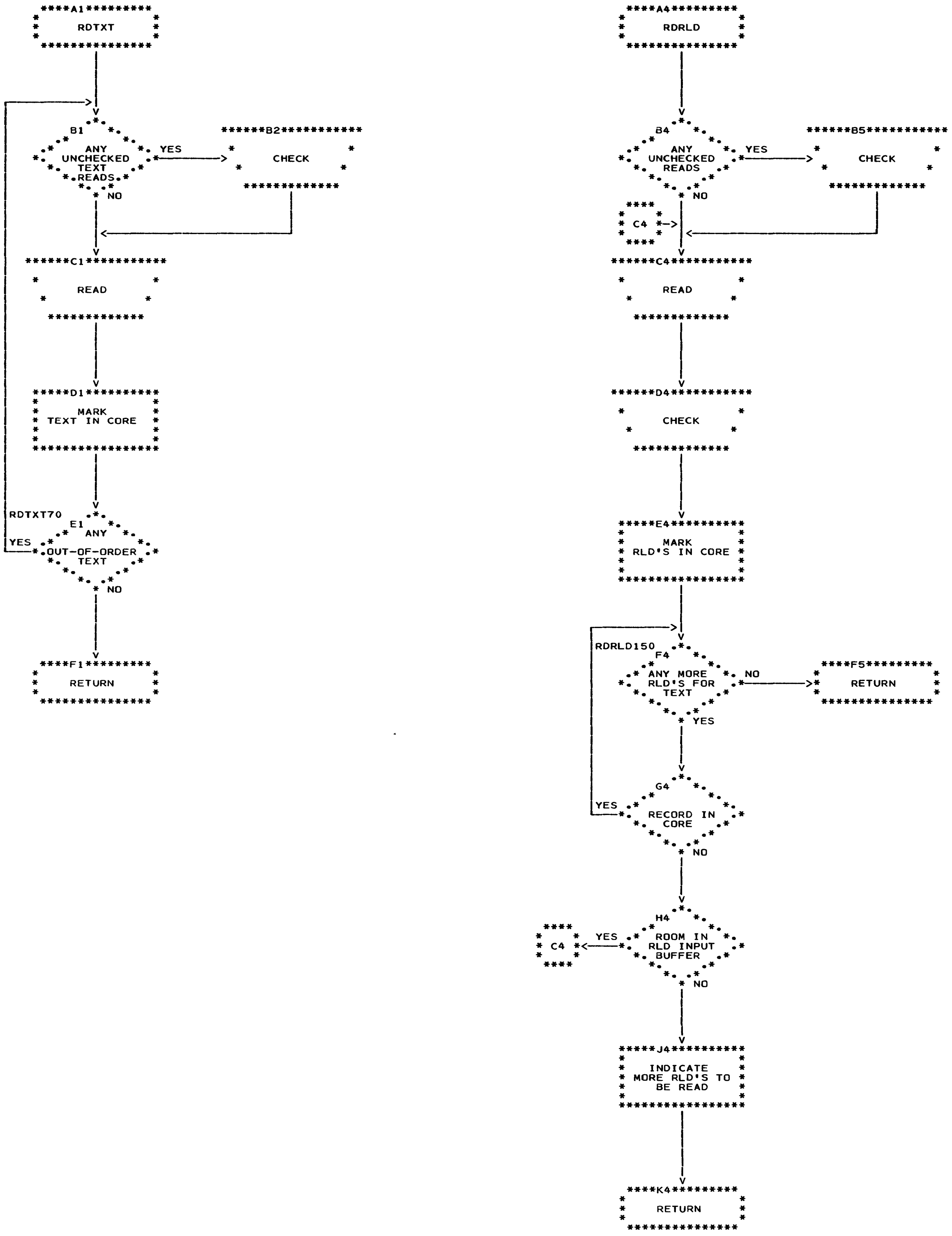


Chart ME. WRTTXT Routine

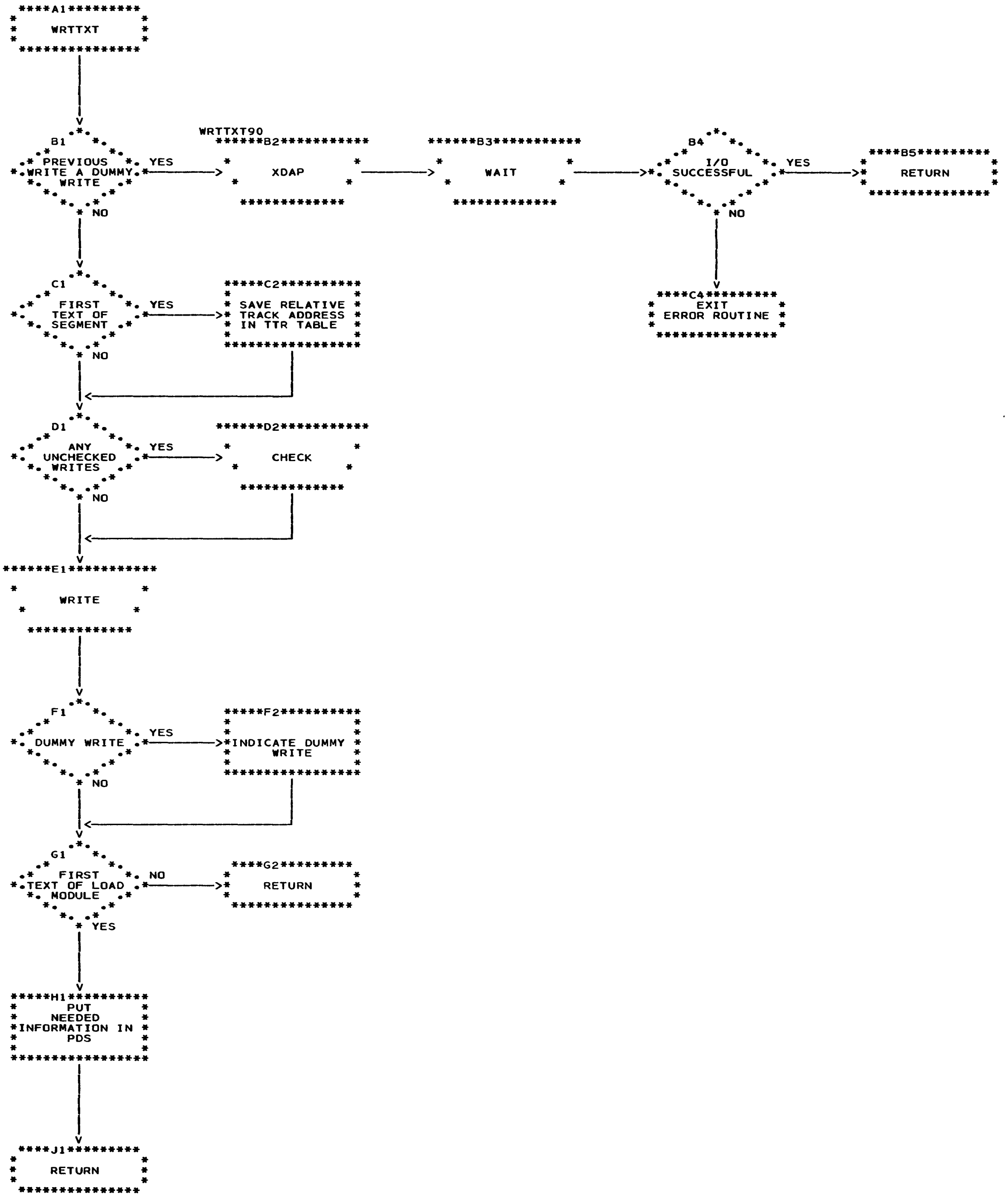


Chart MF. Relocation Routine (IEWLMREL)

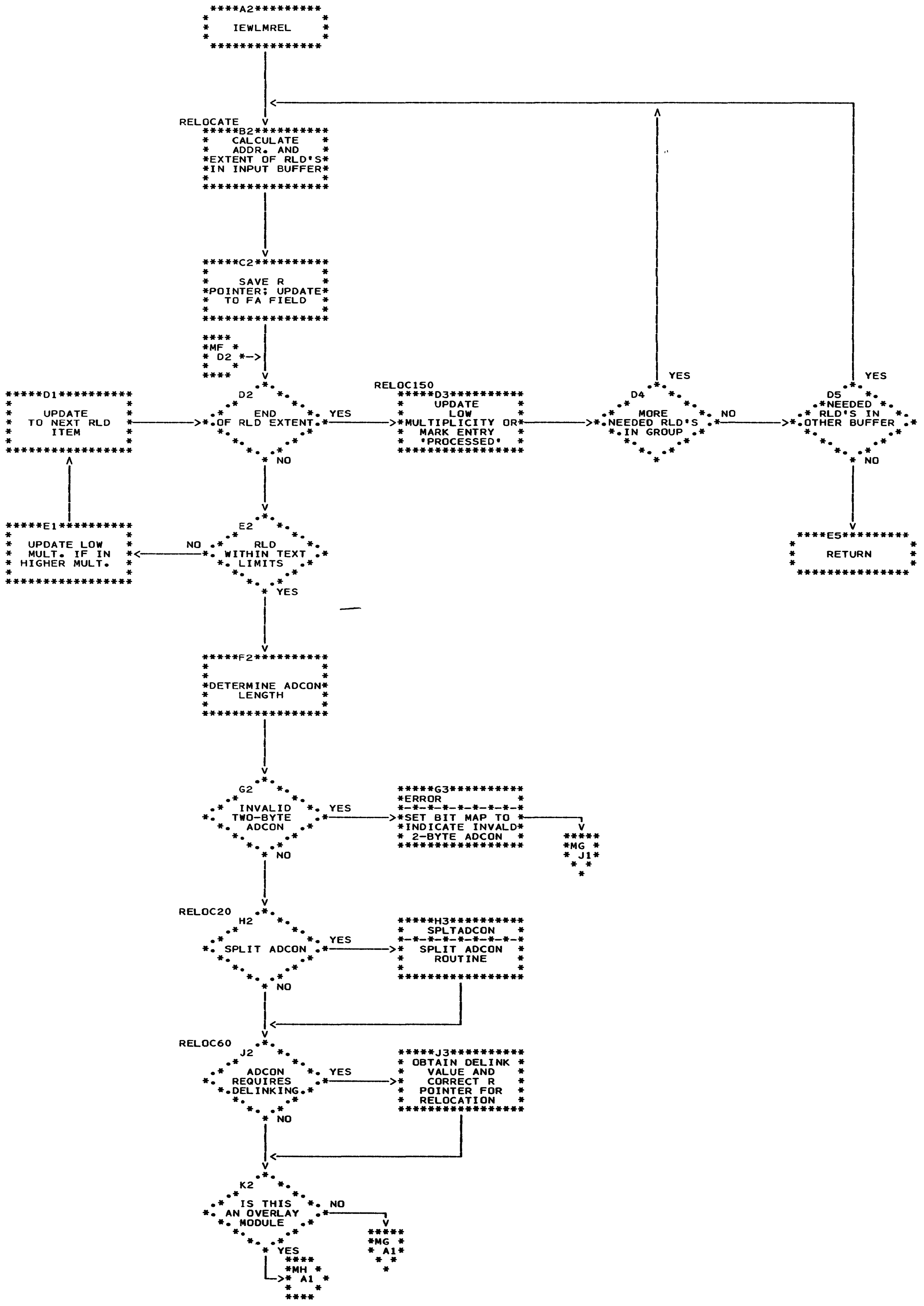

A2
IEWLMREL
RELOCATE
B2
CALCULATE ADDR. AND EXTENT OF RLD'S IN INPUT BUFFER
C2
SAVE R POINTER; UPDATE TO FA FIELD
MF D2
D1
UPDATE TO NEXT RLD ITEM
D2
END OF RLD EXTENT
YES
NO
RELOC150
D3
UPDATE LOW MULTIPLICITY OR MARK ENTRY 'PROCESSED'
D4
MORE NEEDED RLD'S IN GROUP
YES
NO
D5
NEEDED RLD'S IN OTHER BUFFER
YES
NO
E1
UPDATE LOW MULT. IF IN HIGHER MULT.
E2
RLD WITHIN TEXT LIMITS
NO
YES
E5
RETURN
F2
DETERMINE ADCON LENGTH
G2
INVALID TWO-BYTE ADCON
YES
NO
G3
ERROR
SET BIT MAP TO INDICATE INVALD 2-BYTE ADCON
MG J1
RELOC20
H2
SPLIT ADCON
YES
NO
H3
SPLTADCON
SPLIT ADCON ROUTINE
RELOC60
J2
ADCON REQUIRES DELINKING
YES
NO
J3
OBTAIN DELINK VALUE AND CORRECT R POINTER FOR RELOCATION
K2
IS THIS AN OVERLAY MODULE
NO
YES
MG A1
MH A1

Chart MG. Relocation Routine (IEWLMREL) (Continued)

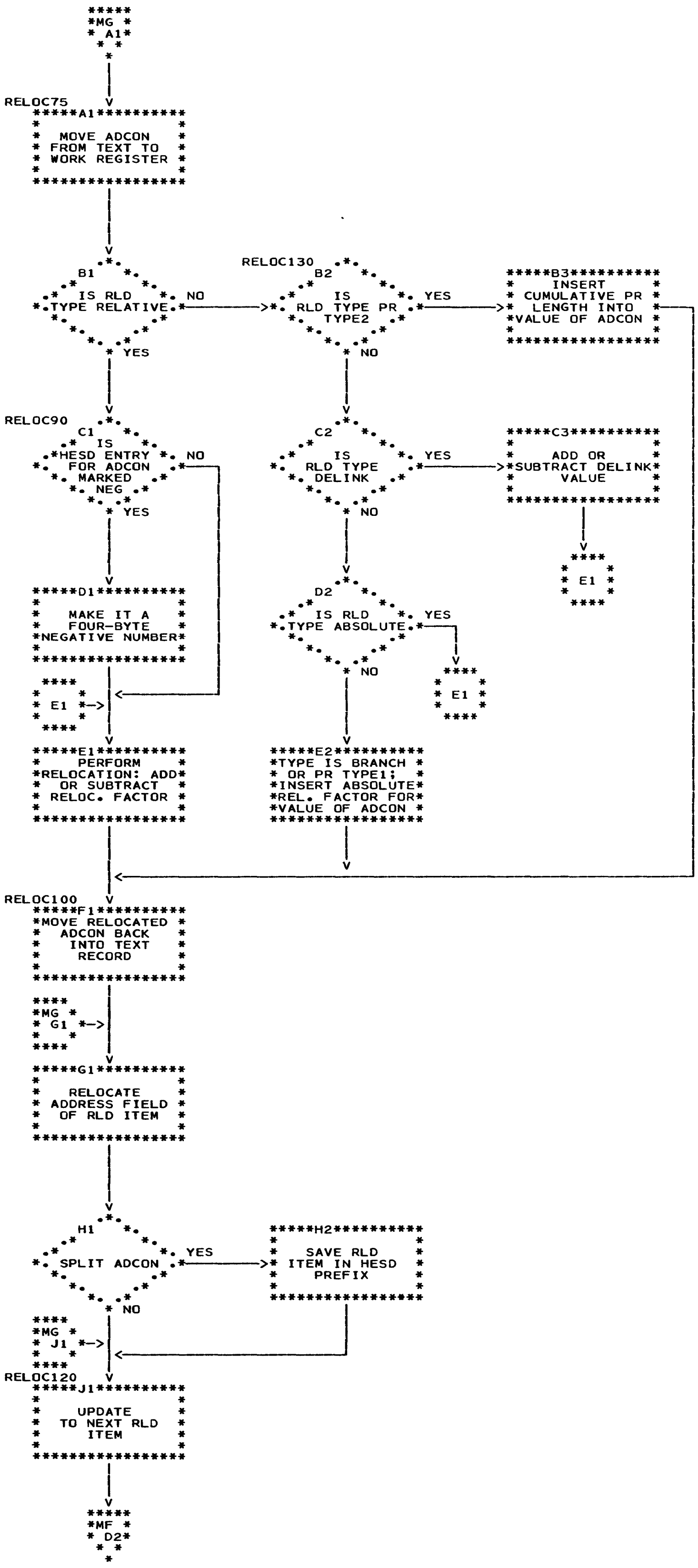

MG A1
RELOC75
A1
MOVE ADCON FROM TEXT TO WORK REGISTER
B1
IS RLD TYPE RELATIVE
NO
YES
RELOC130
B2
IS RLD TYPE PR TYPE2
YES
NO
B3
INSERT CUMULATIVE PR LENGTH INTO VALUE OF ADCON
RELOC90
C1
IS HESD ENTRY FOR ADCON MARKED NEG
NO
YES
C2
IS RLD TYPE DELINK
YES
NO
C3
ADD OR SUBTRACT DELINK VALUE
E1
D1
MAKE IT A FOUR-BYTE NEGATIVE NUMBER
D2
IS RLD TYPE ABSOLUTE
YES
NO
E1
E1
E1
PERFORM RELOCATION: ADD OR SUBTRACT RELOC. FACTOR
E2
TYPE IS BRANCH OR PR TYPE1; INSERT ABSOLUTE REL. FACTOR FOR VALUE OF ADCON
RELOC100
F1
MOVE RELOCATED ADCON BACK INTO TEXT RECORD
MG G1
G1
RELOCATE ADDRESS FIELD OF RLD ITEM
H1
SPLIT ADCON
YES
NO
H2
SAVE RLD ITEM IN HESD PREFIX
MG J1
RELOC120
J1
UPDATE TO NEXT RLD ITEM
MF D2

Chart MH. Relocation Routine (IEWLMREL) (Continued)

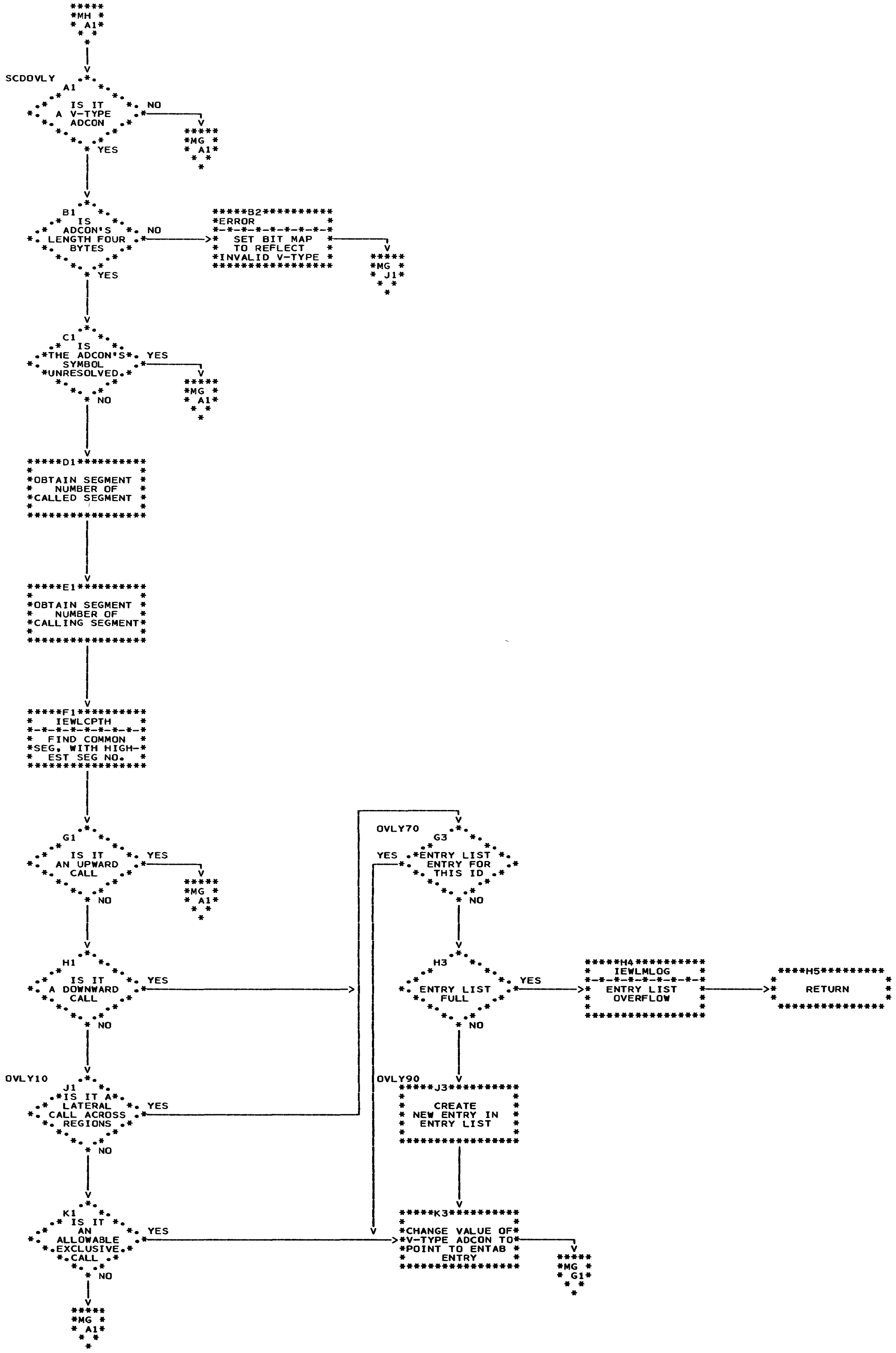

MH A1
SCDOVLY
A1
IS IT A V-TYPE ADCON
NO
MG A1
YES
B1
IS ADCON'S LENGTH FOUR BYTES
NO
B2
ERROR
SET BIT MAP TO REFLECT INVALID V-TYPE
MG J1
YES
C1
IS THE ADCON'S SYMBOL UNRESOLVED
YES
MG A1
NO
D1
OBTAIN SEGMENT NUMBER OF CALLED SEGMENT
E1
OBTAIN SEGMENT NUMBER OF CALLING SEGMENT
F1
IEWLCPTH
FIND COMMON SEG. WITH HIGH-EST SEG NO.
G1
IS IT AN UPWARD CALL
YES
MG A1
NO
H1
IS IT A DOWNWARD CALL
YES
NO
OVLY10
J1
IS IT A LATERAL CALL ACROSS REGIONS
YES
NO
K1
IS IT AN ALLOWABLE EXCLUSIVE CALL
YES
NO
MG A1
OVLY70
G3
ENTRY LIST ENTRY FOR THIS ID
YES
NO
H3
ENTRY LIST FULL
YES
H4
IEWLMLOG
ENTRY LIST OVERFLOW
H5
RETURN
NO
OVLY90
J3
CREATE NEW ENTRY IN ENTRY LIST
K3
CHANGE VALUE OF V-TYPE ADCON TO POINT TO ENTAB ENTRY
MG G1

Chart NA. Final Processor

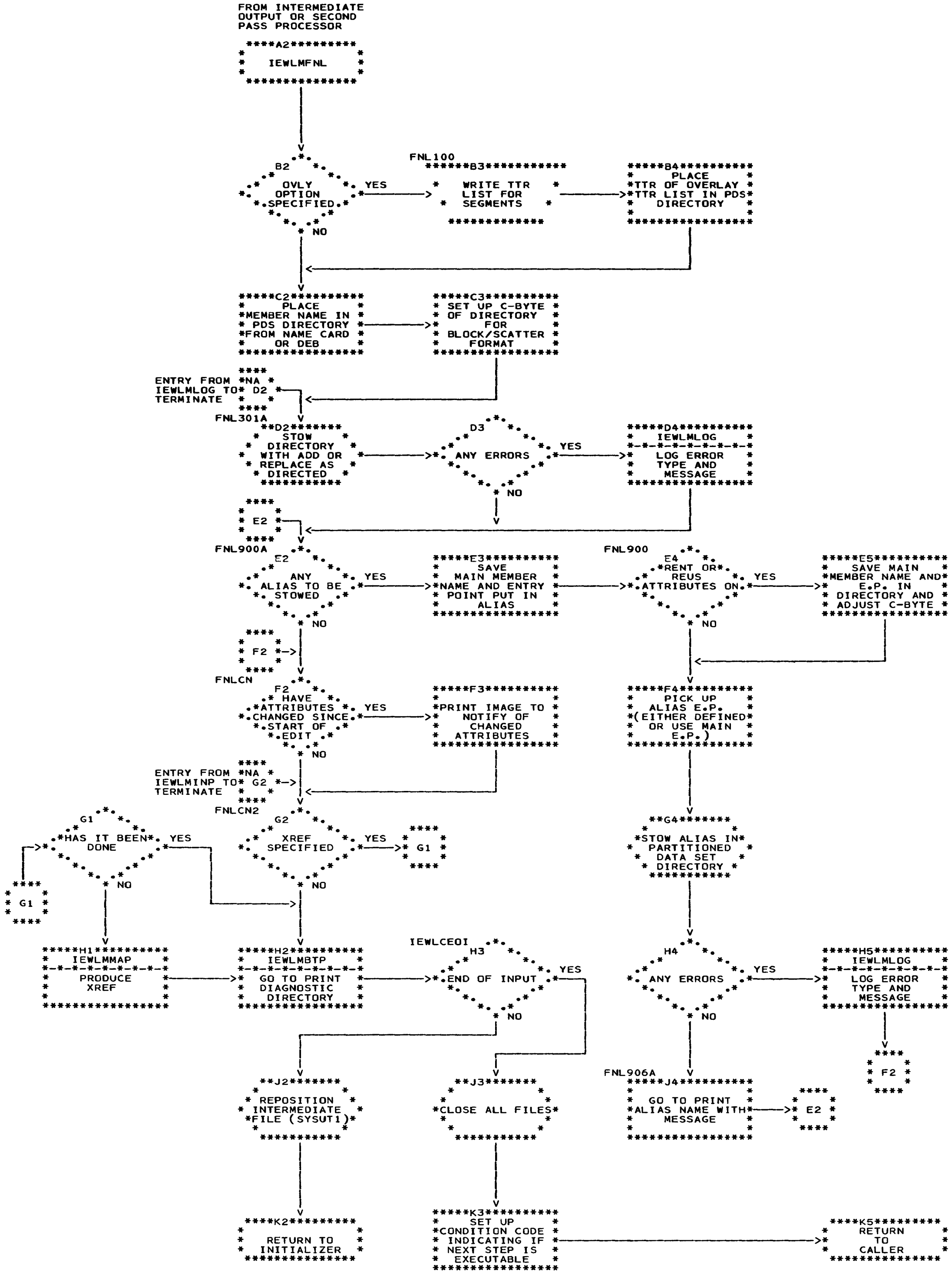

FROM INTERMEDIATE OUTPUT OR SECOND PASS PROCESSOR
A2
IEWLMFNL
B2
OVLY OPTION SPECIFIED
YES
NO
FNL100
B3
WRITE TTR LIST FOR SEGMENTS
B4
PLACE TTR OF OVERLAY TTR LIST IN PDS DIRECTORY
C2
PLACE MEMBER NAME IN PDS DIRECTORY FROM NAME CARD OR DEB
C3
SET UP C-BYTE OF DIRECTORY FOR BLOCK/SCATTER FORMAT
ENTRY FROM IEWLMLOG TO TERMINATE
NA D2
FNL301A
D2
STOW DIRECTORY WITH ADD OR REPLACE AS DIRECTED
D3
ANY ERRORS
YES
NO
D4
IEWLMLOG
LOG ERROR TYPE AND MESSAGE
E2
FNL900A
E2
ANY ALIAS TO BE STOWED
YES
NO
E3
SAVE MAIN MEMBER NAME AND ENTRY POINT PUT IN ALIAS
FNL900
E4
RENT OR REUS ATTRIBUTES ON
YES
NO
E5
SAVE MAIN MEMBER NAME AND E.P. IN DIRECTORY AND ADJUST C-BYTE
F2
FNLCN
F2
HAVE ATTRIBUTES CHANGED SINCE START OF EDIT
YES
NO
F3
PRINT IMAGE TO NOTIFY OF CHANGED ATTRIBUTES
F4
PICK UP ALIAS E.P. (EITHER DEFINED OR USE MAIN E.P.)
ENTRY FROM IEWLMINP TO TERMINATE
NA G2
FNLCN2
G1
HAS IT BEEN DONE
YES
NO
G1
G2
XREF SPECIFIED
YES
NO
G1
G4
STOW ALIAS IN PARTITIONED DATA SET DIRECTORY
H1
IEWLMMAP
PRODUCE XREF
H2
IEWLMBTP
GO TO PRINT DIAGNOSTIC DIRECTORY
IEWLCEOI
H3
END OF INPUT
YES
NO
H4
ANY ERRORS
YES
NO
H5
IEWLMLOG
LOG ERROR TYPE AND MESSAGE
F2
J2
REPOSITION INTERMEDIATE FILE (SYSUT1)
J3
CLOSE ALL FILES
FNL906A
J4
GO TO PRINT ALIAS NAME WITH MESSAGE
E2
K2
RETURN TO INITIALIZER
K3
SET UP CONDITION CODE INDICATING IF NEXT STEP IS EXECUTABLE
K5
RETURN TO CALLER

Chart NB. SYNAD Routine

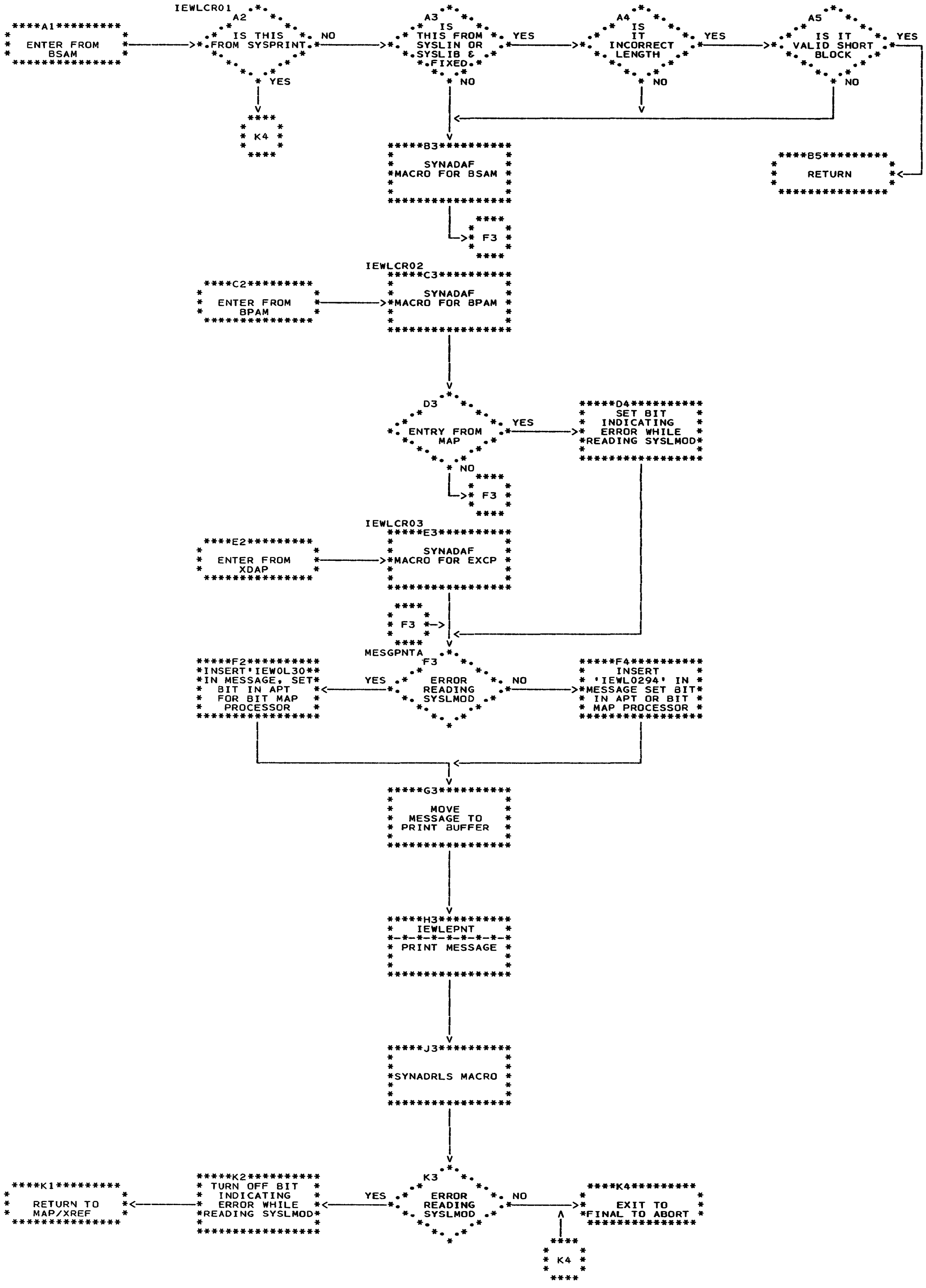
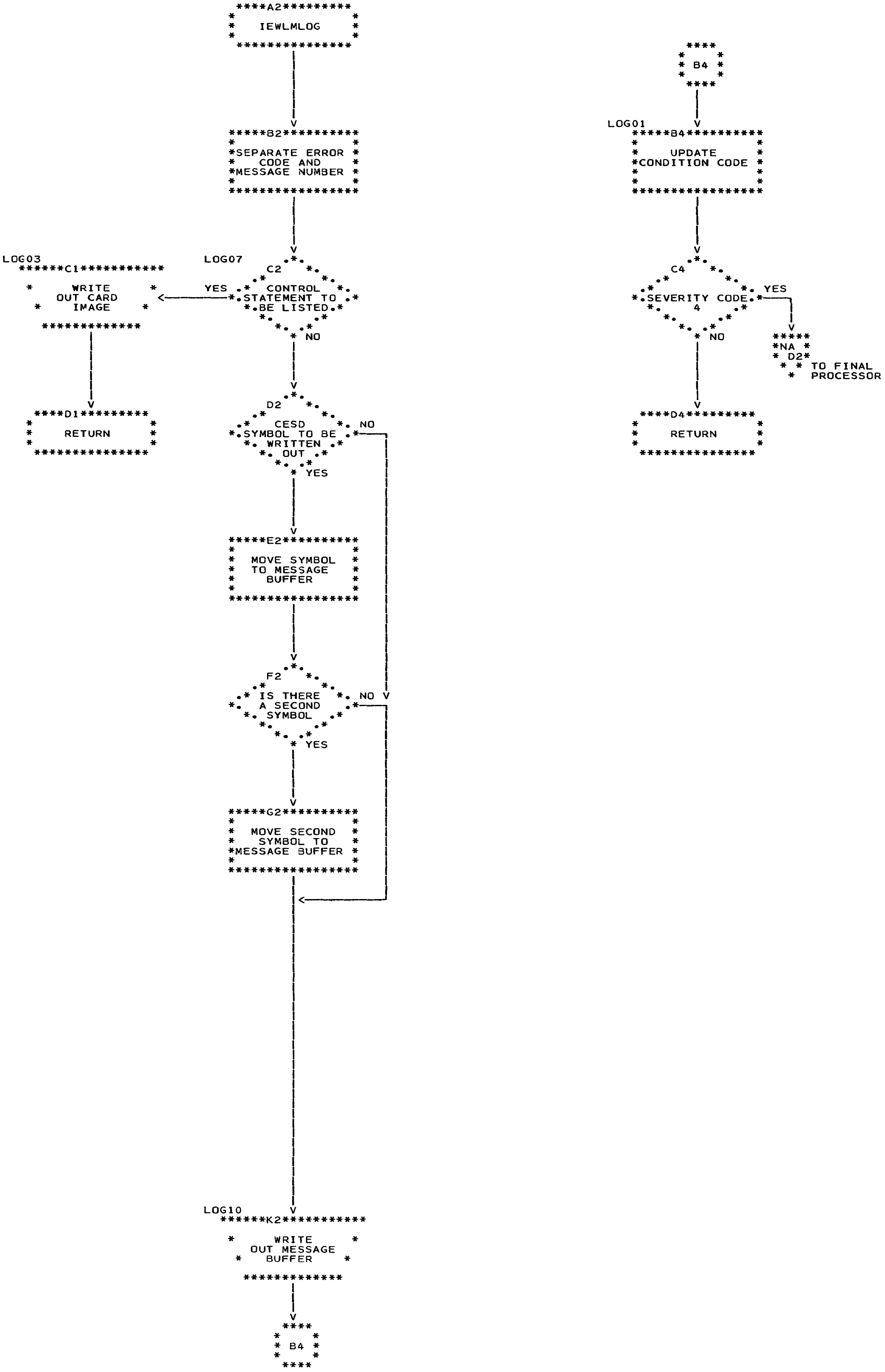


Chart NC. ERR. Logging RTN-IEWLMLOG



SECTION 4: MICROFICHE DIRECTORY

The microfiche directory is designed to help you find named areas of code in the program listing, which is contained on microfiche cards at your installation. Microfiche cards are filed in alphameric order by object module name. If you wish to locate a control section, entry point, table, or routine on microfiche, find the name in column one and note the associated object module name. You can then find the item on microfiche, via the object module name; for example, the Alias Table is on card IEWLMENT. The other columns provide a description of the item, its flowchart ID (if applicable), its overlay segment number, and a synopsis of its function (or its contents, if a table).

This section also contains:

- A CSECT-module cross-reference table.
- Diagrams of the overlay tree structures for the 44K and 88K versions of Linkage Editor F.

Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment **	Chart ID	Synopsis
Alias Table	Table	IEWLMENT	IEWLMENT	7,3	--	ALIAS symbols from CESD
All Purpose Table	Table	IEWLMAPT	IEWLMAPT	1,1	--	Major communication area
Calls List	Table	IEWLMRAT	IEWLMRAT	3,2	--	Entries for V-type ADCONS
CESD	Table	IEWLMESD	IEWLMESD	5,2	--	ESD control information
Delink Table	Table	IEWLMINP	IEWLMINP	3,2	--	Entries for symbols being deleted
Downward Calls List	Table	IEWLMENS	IEWLMENS	7,3	--	Downward calls from V-type ADCONS
Entry List	Table	IEWLMREL	IEWLMREL	8,3	--	Control information for V-type ADCONS
FSNX	Entry Point	IEWLMFNL	IEWLMFNL	9,3	--	Synchronous file error exit
GETIDMUL	Lookahead/ Readahead Routine	IEWLMSCD	IEWLMSCD	8,3	MC	Get next ID/multiplicity
HESD	Table	IEWLMOUT	IEWLMOUT	7,3	--	ESD control information
High ID Table	Table	IEWLMOUT	IEWLMOUT	7,3	--	High ID for each segment
IEWLCAD1	Entry Point	IEWLMADA	IEWLMADA	7,3	--	Compute number of ENTAB bytes per segment
IEWLCAUT	Entry Point	IEWLMINC	IEWLMINC	3,2	JS,JT	Automatic library call processing

(Continued)

(Continued)

Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment **	Chart ID	Synopsis
IEWLCDCN	Entry Point	IEWLMRCG	IEWLMRCG	5,2	*JG	Remove CESD item from library chain
IEWLCDLK	Entry Point JK	IEWLMINP	IEWLMINP	3,2	*JF,JG,	Builds delink table
IEWLCEOD	Entry Point	IEWLMINP	IEWLMINP	3,2	--	EOD for SYSLIB
IEWLCFAB	Entry Point	IEWLMFNL	IEWLMFNL	9,3	--	Termination processing
IEWLCPTH	Entry Point	IEWLMRCG	IEWLMRCG	5,2	--	Determine common segment in Overlay path
IEWLCRBB	Entry Point	IEWLMAPT	IEWLMAPT	1,1	--	Define SYSLIB DECB
IEWLCRBN	Entry Point	IEWLMAPT	IEWLMAPT	1,1	--	Define SYSLIN DECB
IEWLCRO1	Entry Point	IEWLMROU	IEWLMROU	1,1	--	SYNAD routine
IEWLCSDB	Label	IEWLMROU	IEWLMROU	1,1	--	SYSLIN DCB
IEWLEEON	Entry Point	IEWLMINP	IEWLMINP	3,2	--	EOD for SYSLIN
IEWLERDM	Entry Point	IEWLMINP	IEWLMINP	3,2	--	Read Routine
IEWLMADA	CSECT	IEWLMADA	IEWLMADA	7,3	KA	Address assignment
IEWLMAPT	CSECT	IEWLMAPT	IEWLMAPT	1,1	--	All purpose table
IEWLMBTP	CSECT	IEWLMBTP	IEWLMBTP	9,3	*NA	Print Error Messages
IEWLMDEF	CSECT	IEWLMDEF	IEWLMDEF	1,1	--	Default values for SIZE
IEWLMEND	CSECT	IEWLMEND	IEWLMEND	5,2	JN	END Statement Processing
IEWLMENS	CSECT	IEWLMENS	IEWLMENS	7,3	KB	ENTAB size determination
IEWLMENT	CSECT	IEWLMENT	IEWLMENT	7,3	KC,KD	ENTRY statement processing
IEWLMESD	CSECT	IEWLMESD	IEWLMESD	5,2	JE,JF,JG	ESD record processing
IEWLMFNL	CSECT	IEWLMFNL	IFWLMFNL	9,3	NA	Final processing
IEWLMINC	CSECT	IEWLMINC	IEWLMINC	3,2	JR	Include processing
IEWLMINP	CSECT	IEWLMINP	IEWLMINP	3,2	JA	Input processing
IEWLMLDB	Label	IEWLMROU	IEWLMROU	1,1	--	SYSLIB DCB
IEWLMLOG	Error Diag. and Log Routine	IEWLMROU	IEWLMROU	1,1	NC	Print error messages and log control cards
IEWLMMAP	CSECT	IEWLMMAP	IEWLMMAP	6,3	LB	MAP/XREF processing
IEWLMMDI	Entry Point	IEWLMINP	IEWLMINP	3,2	JB	Object module processing

(Continued)

(Continued)

Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment **	Chart ID	Synopsis
IEWLMOPT	CSECT	IEWLMOPT	IEWLMOPT	2,2	*IA	Determine attributes and options
IEWLMOUT	CSECT	IEWLMOUT	IEWLMOUT	7,3	LA	Intermediate output processing
IEWLMRAT	CSECT	IEWLMRAT	IEWLMRAT	3,2	JH	TXT and RLD processing
IEWLMRCG	CSECT	IEWLMRCG	IEWLMRCG	5,2	--	Replace/change processing
IEWLMREL	CSECT	IEWLMREL	IEWLMREL	8,3	MF,MG,MH	Relocate address constants
IEWLMROU	CSECT	IEWLMROU	IEWLMROU	1,1	--	Linkage editor F entry point
IEWLMSCD	CSECT	IEWLMSCD	IEWLMSCD	8,3	MA,MB	Second pass processing
IEWLMSCN	CSECT	IEWLMSCN	IEWLMSCN	4,2	JO,JP	Control statement scanning
IEWLMSYM	CSECT	IEWLMSYM	IEWLMSYM	5,2	JD	SYM processing
IEWLMTXT	CSECT	IEWLMRAT	IEWLMTXT	3,2	JI	TXT processing
INP270	Label	IEWLMINP	IEWLMINP	3,2	JC	Load module processing
RDRLD	RLD Read Routine	IEWLMSCD	IEWLMSCD	8,3	MD	Read RLDs from SYSUT1
RDTXT	Text Read Routine	IEWLMSCD	IEWLMSCD	8,3	MD	Read TXT from SYSUT1
Relocation Constant Table	Table	IEWLMADA	IEWLMADA	7,3	--	Relocation constants
Renumbering Table	Table	IEWLMESD	IEWLMESD	3,2	--	ESD - CESD item resolution
RLD I/O Control Table	Table	IEWLMRAT	IEWLMRAT	3,2	--	Description of RLDs on SYSUT1
RLD Note List	Table	IEWLMRAT	IEWLMRAT	3,2	--	Location of RLDs on SYSUT1
RLD001	RLD Processing Routine	IEWLMRAT	IEWLMRAT	3,2	JK,JL	RLD processing
Scatter Table	Table	IEWLMOUT	IEWLMOUT	7,3	--	Ordered symbol addresses
SCDENTAB	ENTAB and ENTAL RLD Creation Routine	IEWLMREL	IEWLMREL	6,3	*MB	Build and write ENTABs and ENTAL RLD to SYSLMOD

(Continued)

(Continued)

Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment **	Chart ID	Synopsis
Segment Length Table	Table	IEWLMADA	IEWLMADA	7,3	--	Segment lengths
SEGTAB	Table	IEWLMOUT	IEWLMOUT	7,3	--	Segment relationships
SPLTADCN	Split ADCON Routine	IEWLMREL	IEWLMREL	6,3	*MF	Relocate split ADCONs
SYSLMOD	Label	IEWLMROU	IEWLMROU	1,1	--	SYSLMOD DCB
SYSPRINT	Label	IEWLMROU	IEWLMROU	1,1	--	SYSPRINT DCB
SYSUT1	Label	IEWLMROU	IEWLMROU	1,1	--	SYSUT1 DCB
Test I/O Control Table (TTR List)	Table	IEWLMRAT	IEWLMTXT	3,2	--	Address of first text in each segment
Text I/O Table	Table	IEWLMRAT	IEWLMTXT	3,2	--	Description of TXT on SYSUT1
TEXT Note List	Table	IEWLMRAT	IEWLMTXT	3,2	--	Location of TXT on SYSUT1
Translation Table	Table	IEWLMOUT	IEWLMOUT	7,3	--	Pointers to Scatter Table entries
TXTBUF	TXT Write Routine	IEWLMRAT	IEWLMTXT	3,2	JJ	Write test to SYSUT1
WRTCRRLD	RLD/Control Record Write Routine	IEWLMSCD	IEWLMSCD	6,3	*MA,MB	Write RLDs and control - records
WRTTXT	TXT Write Routine	IEWLMSCD	IEWLMSCD	6,3	ME	Write TXT to SYSLMOD
RLDBUF	RLD Write Routine	IEWLMRAT	IEWLMRAT	3,2	JM	Write RLDs and RLD Note List to SYSUT1

*Mentioned only; not a complete chart for this routine.

**The first number refers to the segment in the 44K overlay structure; the second number refers to the segment in the 88K overlay structure.

Table 9. Level F Module -- CSECT Cross Reference Table

Module Name	CSECT Name
IEWLMADA	IEWLMADA
IEWLMAPT	IEWLMAPT
IEWLMBTP	IEWLMBTP
IEWLMEND	IEWLMEND
IEWLMENS	IEWLMENS
IEWLMENT	IEWLMENT
IEWLMESD	IEWLMESD
IEWLMFNL	IEWLMFNL
IEWLMINC	IEWLMINC
IEWLMINP	IEWLMINP
IEWLMMAP	IEWLMMAP
IEWLMOUT	IEWLMOUT
IEWLMRAT	IEWLMRAT,IEWLMTXT
IEWLMRCG	IEWLMRCG
IEWLMREL	IEWLMREL
IEWLMROU	IEWLMROU
IEWLMSCD	IEWLMSCD
IEWLMSCN	IEWLMSCN
IEWLMSYM	IEWLMSYM,IEWLMDEF

Overlay Tree Structure for the 44K Version of Linkage Editor F

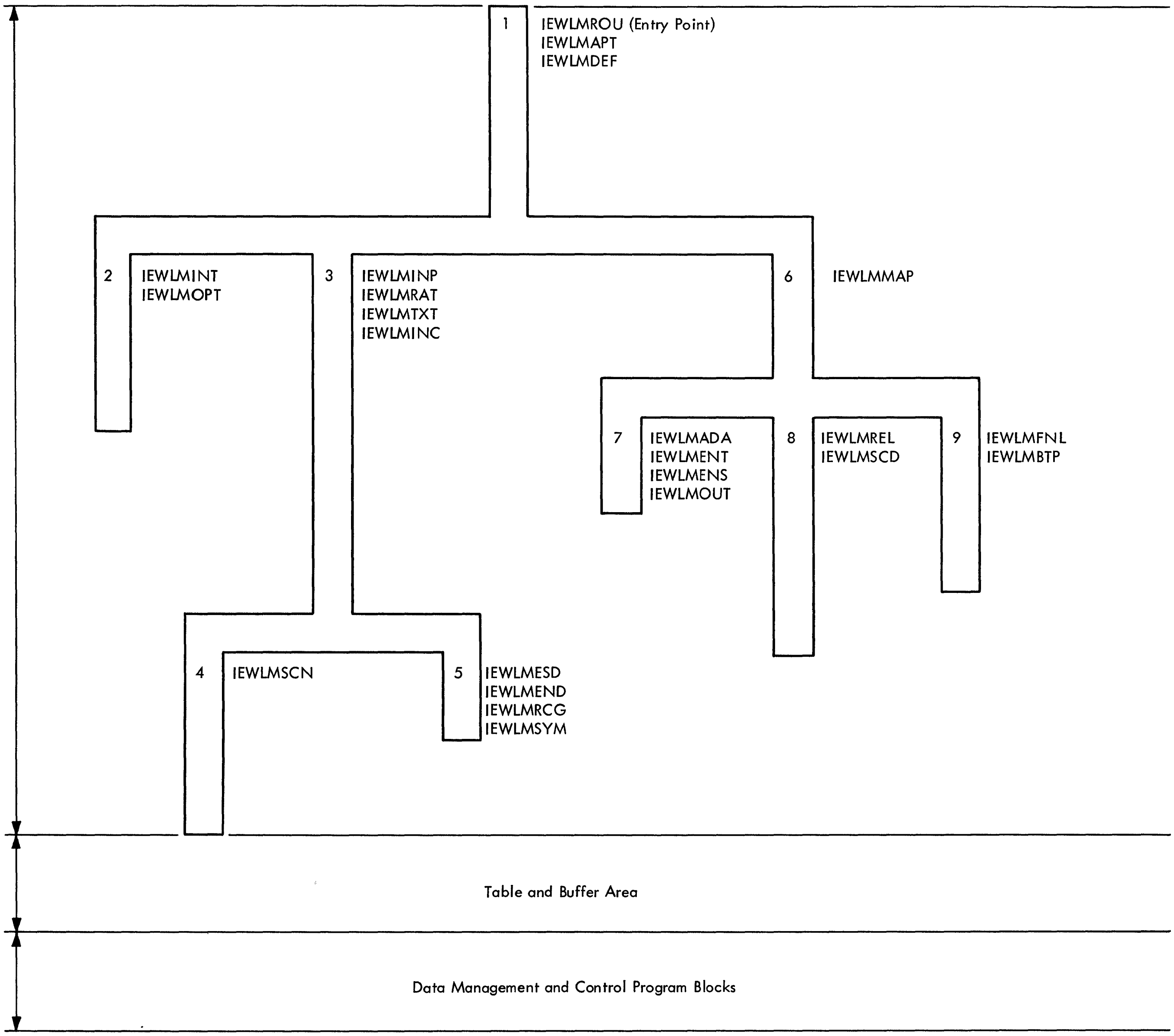


Figure 35. Overlay Tree Structure for Linkage Editor F (44K)

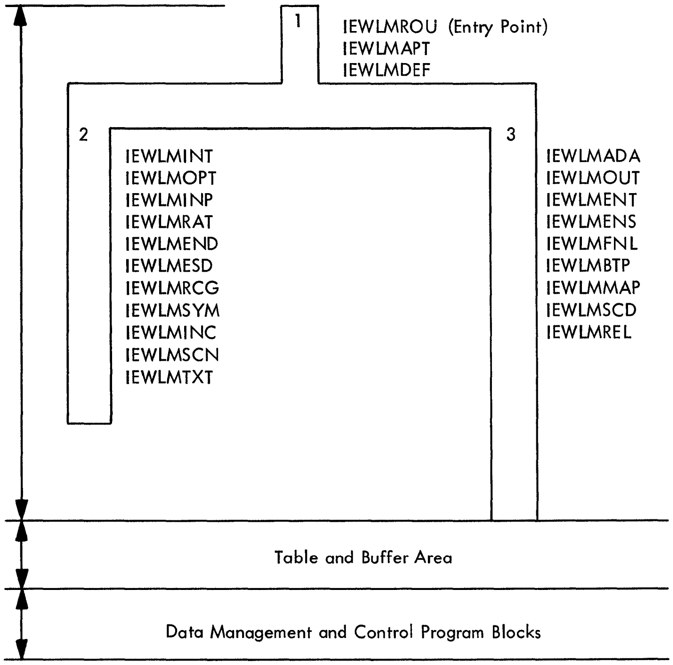


Figure 36. Overlay Tree Structure for Linkage Editor F (88K)

This section provides detailed layouts of internal tables used during Linkage Editor F processing. Table 10 indicates the modules in which tables are initialized and used or modified. Tables described in this section are included alphabetically except for the All Purpose Table, which is shown first.

Table 10. Table Construction and Usage

Table	Built by	Used and/or Modified by
Alias Table	IEWLMENT	IEWLMFNL
All Purpose Table	IEWLMINT	**
Calls List	LEWLMRAT	IEWLMENS
CESD	IEWLMESD	IEWLMRAT,IEWLMSCN,IEWLMINC,IEWLMAUT, IEWLMENS,IEWLMENT,IEWLMOUT,IFWLMTXT
Delink Table	IEWLMFSD	IEWLMRAT,IEWLMSCD
Downward Calls List	IEWLMENS	*
Entry List	IEWLMSCD	*
Entry Table	IEWLMSCD	*
Half ESD	IEWLMOUT	IEWLMSCD
Half ESD Prefix	IEWLMSCD	*
High ID Table	IEWLMOUT	*
Main Storage Allocation Table	IEWLMINT	*
Relocation Constant Table	IEWLMADA	IEWLMOUT,IEWLMSCD
Renumbering Table	IEWLMESD	IEWLMRAT,IEWLMTXT
RLD Input Control Blocks	IEWLMSCD	*
RLD Note List	IEWLMRAT	IEWLMOUT,IEWLMSCD
RLD Output Control Blocks	IEWLMSCD	*
Second Pass Text Control Blocks	IEWLMSCD	*
SEGLGTH Table	IEWLMADA	*
Segment Table (SEGTA1)	IEWLMOUT	IEWLMSCD
Text I/O Table	IEWLMTXT	IEWLMOUT,IEWLMSCD
Text Note List	IEWLMTXT	IFWLMOUT,IEWLMSCD
TTR List (TXT I/O Control Table)	IEWLMSCD	IFWLMSCD

*Built and processed entirely within one routine.
**Major communications area throughout linkage editor processing.

Table 11. All Purpose Table (APT)

All Purpose Table (APT)

0	PDSE1					PDS Directory
8	PDSE2	PDSE3	PDSE4			
16	PDSE5	PDSE6	PDSE7	PDSE8	PDSE9	
24	PDSE9	PDSE10	PDSE11	PDSE12		
32	PDSE12	PDSE13	PDSE14	PDSE15	PDSE16	
40	PDSE16	PDSE17	PDSE18			
48	PDSE18	REGSA				Register Save Area for IOCS
56	REGSA					
64	REGSA					
72	REGSA					
80	REGSA					
88	REGSA					
96	REGSA					
104	REGSA					
112	REGSA					
120	REGSA	IOCT				I/O Control Table
128	IOCT					
136	IOCT					
144	IOCT	APT0	APT1	APT2	APT3	
152	CTTR	CSNO	CRNO			
160	PRAL	FLCD				Addresses of Tables
168	RCCE	RCCB				
176	ALCB	OVCMBGAD				
184	SGT1	CLLT				
192	TNT1	RNT1				
200	RLDINPAD	RECNT				
208	TXTIO	ALAS				
216	DLKT	CHESD				
224	SELST	TNLS2				
232	RNLS2	TTRLIST				
240	RLDOUTBF	HIARADD				

248	BITMAP					
256	LINECNT	HISEV	INCBRKPT			
264	CRRTINCL	ENCDX	ENT1X			Capacities of Tables
272	ENR1X	ENT2X	ENR2X	ENT0X		
280	ENCLX	ENDTX	ENS1X	BUFSIZ		
288	HESD	ENELTX	ENRLD2X			
296	ENSPX		LSTS			
304	EPSM					
312	ENT1C	ENR1C	ENITC	ENIRC		Current Usage of Tables
320	ENTOC	ENCLC	ENS1C	ENASC		
328	ENDTC	ENCDC	ENELTC	ENT2C		
336	ENR2C	ENSPC	SYSRTN			
344	SYSRTN	Spaces				Save Area
352	Spaces					
360	Spaces					
368	Spaces					
376	Spaces					
384	Spaces					
392	Spaces					
400	Spaces					
408	Spaces					
416	Spaces	ERDIG				
424	SSI	FFCADR				
432	LIBNAME					
440	LIBOPEN					
448		SAVATS	APTSWS	NEWSW	NEWSW2	
456	MAXBF		IEWLCRBB			I/O Control Block for SYSLIB
464	IEWLCRBB					
472	IEWLCRBB					
480	IEWLCRBB					

488	IEWLCRBN		I/O Control Block for SYSLIN
496	IEWLCRBN		
504	IEWLCRBN		
512	IEWLCRBN	IEWLCWBB	I/O Control Block for SYSPRINT
520	IEWLCWBB		
528	IEWLCWBB		
536	IEWLCWBB		
544	RLDOUT1	RLDOUT2	
552	TXTBFBEG	TXTBFEND	
560	MULTSIZE	UT1SIZE	
568	SZSYSUT1	RLDSIZE	
576	VALUE1	VALUE2	
584	MSGONE	MSGTWO	
592	MSGTHREE	IEWLCLAC	
600	DECBLIN		SYSLIN DECB
608	DECBLIN		
616	DECBLIN	DECBLIB	SYSLIB DECB
624	DECBLIB		
632	DECBLIB		
640	NEGATE		

Explanation of APT Entries -- Level F

PDSE1 Member or alias name of module being creat

PDSE2 Relative disk address (TTR) of first record of module on SYSLMOD

PDSE3 C-byte. Initial value 0

Bit 0 Alias indicator

Bits 1-2 Number of TTRs in user's data

Bits 3-7 Lengh of user's data in halfwords

PSDE4 Relative disk address (TTR0) of first text record

PDSE5 Relative disk address (TTR) of note list or scatter-translation record

PDSE6 "L" byte: number of TTRs in note list if present

PDSE7 First attribute byte.

Module Attribute	Initial Value
Bit 0 - Reenterable	0
Bit 1 - Reusable	0
Bit 2 - Overlay	0
Bit 3 - Test	0
Bit 4 - Only loadable	0
Bit 5 - Block/scatter	0
Bit 6 - Executable	1
Bit 7 - 1 text record, no RLDs	0

PDSE8 Second attribute byte

Bit 0 - Compatibility: on indicates not DC	1
Bit 1 - Origin of first text record is zero	1
Bit 2 - Entry point assigned by linkage editor is 0	1
Bit 3 - Module contains no RLDs	1
Bit 4 - Module can be reprocessed by linkage editor	0
Bit 5 - Module does not contain symbol cards	0
Bit 6 - Spare	0
Bit 7 - Module is refreshable	0

PDSE9 Total contiguous main storage requirements of this module

PDSE10 Length of first text record

PDSE11	Entry point address	
PDSE12	Assigned origin of first text record	
PDSE13	Length, in bytes, of scatter list	
PDSE14	Length, in bytes, of translation table	
PDSE15	ESDID of the first text record	
PDSE16	ESDID of the control section containing the entry point	
PDSE17	Entry point of main member name	
PDSE18	Member name of module	
REGSA	Register save area for IOCS	
IOCT	I/O control table	
APTO	<u>All Purpose Indicators</u>	<u>Initial Value</u>
	Bit 0 - NCAL	0
	Bit 1 - XREF	0
	Bit 2 - MAP	0
	Bit 3 - LET	0
	Bit 4 - LOG	0
	Bit 5 - XCAL	0
	Bit 6 - TXT/RLD	0
	Bit 7 - A library card has been read	0
APT1	All purpose indicators	
	Bit 0 - More include input to come	0
	Bit 1 - Automatic library call in operation	0
	Bit 2 - Object or load module	0
	Bit 3 - Delete indicator	0
	Bit 4 - Entry point received	0
	Bit 5 - Symbolic or absolute entry point	1
	Bit 6 - Entry card received	0
	Bit 7 - ESD Write indicator	0
APT2	All purpose indicators	
	Bit 0 - No length received	0
	Bit 1 - No length indication	0
	Bit 2 - First text record	0
	Bit 3 - Status indicator received	0
	Bit 4 - Include previously initiated	0

	Bit 5 - I/O overlap bit	0
	Bit 6 - In module indicator	0
	Bit 7 - Programmer punched card continuation	0
APT3	All purpose indicators	
	Bit 0 - End of file	0
	Bit 1 - Name card received; end of input for load module	0
	Bit 2 - End of SYSLIN input -- (/*)	0
	Bit 3 - To STOW as replacement	0
	Bit 4 - First text of load module	0
	Bit 5 - First text of segment	0
	Bit 6 - RLDs for group	0
	Bit 7 - SYSLIB opened	0
CTTR	TTR0 of first CESD record on SYSLMOD, if MAP or XREF is specified.	
CSNO	Current segment number	
CRNO	Current region number	
PRAL	Pseudo register accumulative length	
FLCD	Address of first deleted CESD entry	
RCCE	Address of end of replace/change chain	
ALCB	Address of alias chain beginning	
OVCMBGAD	Address of beginning of overlay chain	
SGT1	Address of SEGTA1 - 1	
CLLT	Address of calls list table	
TNT1	Address of text note list 1	
RNT1	Address of RLD note list 1	
RLDINPAD	Address of RLD input buffer	
RECNT	Address of relocation constant table and renumbering table - 1	
TXTIO	Address of test I/O table	
ALAS	Address of alias table	
DLKT	Address of delink table - 1	
CHESD	Address of composite ESD - 16	
SELST	Address of second pass entry list	
TNLS2	Address of text note list 2	
RNLS2	Address of RLD note list 2	

TTRLIST	Address of TTR list
RLDOUTBF	Address of RLD output buffers
HIARADD	Address of hierarchy table
BITMAP	Switches denoting error messages logged
LINECNT	Lines on this page
HISEV	Highest severity message
INCBRKPT	Address of breaking point in include chain
CRRTINCL	Address of currently included ESD item
ENCDX	Maximum number of entries in CESD/HESD tables
ENT1X	Maximum number of entries in test note list 1
ENR1X	Maximum number of entries in RLD note list 1
ENT2X	Maximum number of entires in test note list 2
ENR2X	Maximum number of entries in RLD note list 2
ENTOX	Maximum number of bytes in text I/O table
ENCLX	Maximum number of entries in calls list
ENDTX	Maximum number of entries in delink table
ENS1X	Maximum number of segments
BUFSIZ	Size of load module input buffer
HESD	Address of HESD Table - 8
ENELTX	Maximum number of entries in second pass entry list
ENRLD2X	Maximum size on input RLD buffer
ENSPX	Used by IEWLMOUT
LSTS	Last segment in each region (regions 1 - 4)
EPSM	Entry point symbol or end card address/symbol
ENT1C	Current number of entries in text note list 1
ENR1C	Current number of entries in RLD note list 1
ENITC	Current number of bytes in text I/O control table
ENIRC	Current number of bytes in RLD I/O control table
ENTOC	Current number of bytes in text I/O table
ENCLC	Current number of bytes in calls list
ENS1C	Current number of entries in SEGTAB
ENASC	Current number of entries in alias table
ENDTC	Current number of entries in delink table
ENCDC	Current number of entries in HESD/CESD table

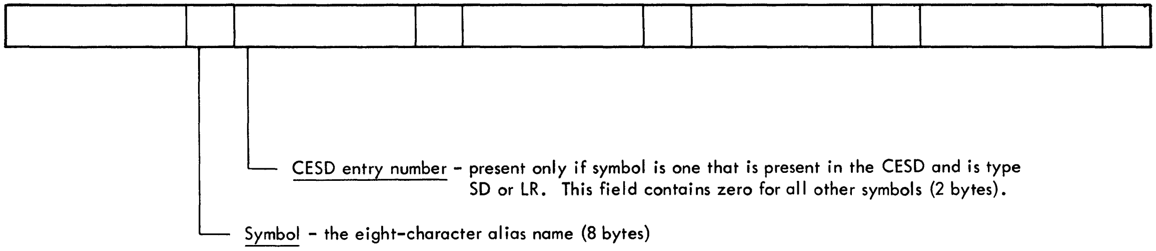
ENELTC	Current number of entries in 2nd pass entry list	
ENT2C	Current number of entries in text note list 2	
ENR2C	Current number of entries in RLD note list 2	
ENSPC	Highest segment number with text	
SYSRTN	Save area for registers 13 and 14 for return to scheduler	
SPACES	Save area	
ERDIG	Address of IEWLMLOG	
SSI	System status indicator	
FFCADR	Highest address retained by allocator	
LIBNAME	Name of library for automatic library call	
LIBOPEN	Name of library currently open	
SAVATS	Attributes save area	
APTSWS	<u>Switches</u>	<u>Initial Value</u>
	Bits 0 - 3 space	
	Bit 4 - Bit map processed	0
	Bit 5 - Linkage editor input received	0
	Bit 6 - SYM received	0
	Bit 7 - ESD received	0
NEWSW	Switches for determining control	
	Bit 0 - If 0, first time in initial processing	0
	Bit 1 - If 1, MAP/XREF entered from intermediate processor If 0, entered from final processor	1
	Bit 2 - If 0, all RLDs in core If 1, RLDs not in core	0
	Bit 3 - If 0, MAP/XREF not in control If 1, MAP/XREF is in control	0
	Bit 4 - If 0, normal printing on SYSPRINT If 1, abort immediately, no printing	0
	Bit 5 - Hierarchy	
	Bits 6 Spare	
	Bit 7 - If one, purge TEXT/RLD buffer	
NEWSW2	<u>Switches for Second Pass Processing</u>	<u>Initial Value</u>
	Bit 0 - More RLDs exist for current ID	0
	Bit 1 - Split RLD in output buffer	0
	Bit 2 - R and P pointers have been saved	0

	Bit 3 - If 0, relative relocation factor needed If 1, absolute relocation factor needed	0
	Bit 4 - Split RLD saved in HESD prefix	0
	Bit 5 - No RLDs exist for last text of segment or last text of module	0
	Bit 6 - RLDs are to be grouped with previous RLDs	0
	Bit 7 - R and P pointers for current chain are in buffer	0
MAXBF	Maximum blocking factor	
IEWLCRBB	Control block for SYSLIB	
IEWLCRBN	Control block for SYSLIN	
IEWLCWBB	Control block for SYSPRINT	
RLDOUT1	Address of first RLD output buffer	
RLDINBF1	Address of first RLD input buffer	
RLDOUT2	Address of second RLD output buffer	
RLDINBF2	Address of second RLD input buffer	
TXTBFBEG	Address of start of text buffer	
TXTBFEND	Address of end of text buffer	
MULTSIZE	Size of SYSLMOD multiplicity or record	
UT1SIZE	Size of SYSUT1 record	
SZSYSUT1	SYSUT1 maximum bytes per track	
RLDSIZE	Size of each RLD buffer: 1st pass output, 2nd pass input	
VALUE1	Size (value1) for available linkage editor storage	
VALUE2	Size (value2) for load module buffer	
MSGONE	Indicates first message from IEWLMCPT	
MSGTWO	Indicates second message from IEWLMOPT	
MSGTHREE	Indicates third message from IEWLMOPT	
IEWLCLAC	Address of current read block	
DECBLIN	DECB for SYSLIN	
DECBLIB	DECB for SYSLIB	
NEGATE	End flag for all purpose table	

Note: The following areas are used by IEWLMSCD and IEWLMREL for other purposes: IOCT, SPACES, EPSM

Alias Table

Built by: Entry Processor
Referred to by: Final Processor



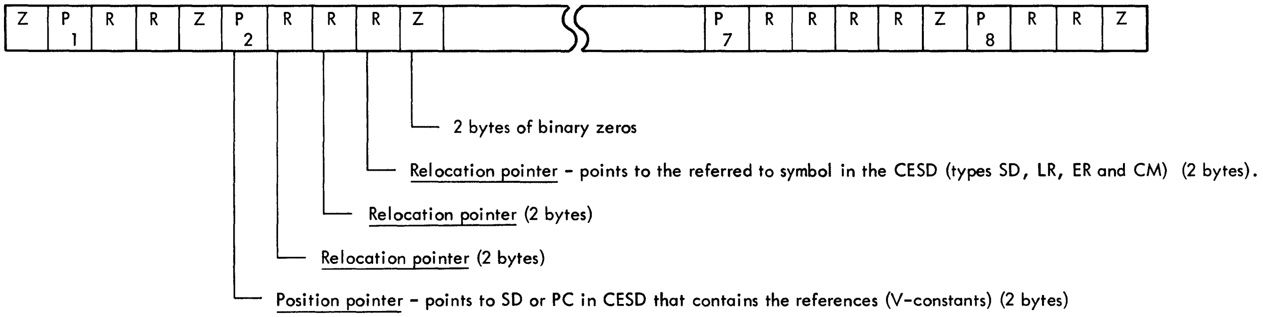
CESD entry number - present only if symbol is one that is present in the CESD and is type SD or LR. This field contains zero for all other symbols (2 bytes).

Symbol - the eight-character alias name (8 bytes)

Figure 37. Alias Table

Calls List

as built by RLD processor



2 bytes of binary zeros

Relocation pointer - points to the referred to symbol in the CESD (types SD, LR, ER and CM) (2 bytes).

Relocation pointer (2 bytes)

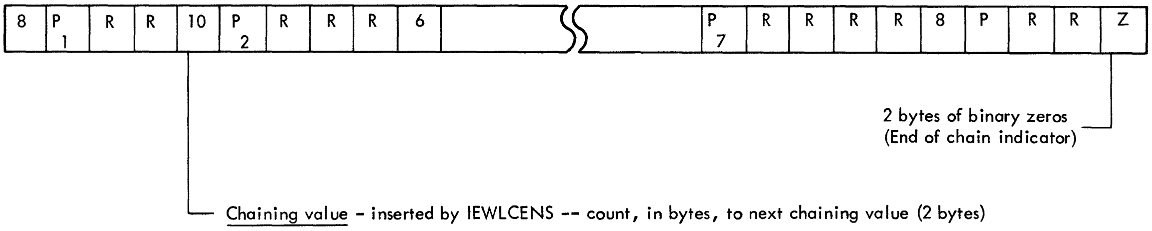
Relocation pointer (2 bytes)

Position pointer - points to SD or PC in CESD that contains the references (V-constants) (2 bytes)

Figure 38. Calls List (As built by RLD Processor)

Calls List

As altered and used by ENTAB size determination (IEWLCENS)



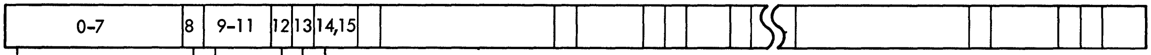
2 bytes of binary zeros
(End of chain indicator)

Chaining value - inserted by IEWLCENS -- count, in bytes, to next chaining value (2 bytes)

Figure 39. Calls List (As altered and Used by ENTAB Size Determinations)

Composite External Symbol Dictionary (CESD) - Internal Format

Built by: ESD Processor and Control Statement Processors
Modified by: Address Assignment Processor



Chain pointer/chain ID/length - Chain pointer when the entry type is: ER-Include w/pointer or an ER-ddname that was extracted from a LIBRARY control statement

Chain ID when the entry type is:
ER-Library (the symbol was extracted from a LIBRARY control statement).

Length of control section for type:
SD, PC, PR, or CM (2 bytes)

Subtype -

		Hex
ER	0000 0000	00
ER-Control change	1111 0000	F0
ER-Control replace	1110 0000	E0
ER-Control delete	1110 1000	E8
ER-Control include w/ pointer	1101 0000	D0
ER-Control include w/o pointer	1100 0000	C0
ER-ddname	1011 0000	B0
ER-Alias	1010 0000	A0
ER-Overlay	1001 0000	90
ER-Unmatched library member	0000 0010	02
ER-Matched library member	0000 0011	03
ER-Unmatched no call	0000 0100	04
ER-Matched no call	0000 0101	05
ER-Never call	0000 0110	06
ER-Delete	0000 1000	08
ER-Replace	0000 0000	00

(1 byte)

Segment number - this symbol appears in (1 byte). When type is PR, this byte contains the alignment value (See Half ESD).

Chain address/reverse chain ID - used to create a chain of CESD entries (3 bytes).

Type -

		Subclassification -	
Section definition (SD)	xxxx 0000	Delete	xxx1 xxxx
Label reference (LR)	xxxx 0011	Replace	xxx1 xxxx
Private code (PC)	xxxx 0100	Insert	xx1x xxxx
Common (CM)	xxxx 0101	Chain	x1xx xxxx
Pseudo register (PR)	xxxx 0110	Map	1xxx xxxx
Null	0000 0111		
External reference (ER)	xxxx 0010		

(1 byte)

NOTE: = Not applicable

Symbol - the eight-character symbolic name (8 bytes)

Figure 40. Composite External Symbol Dictionary (CESD) -- Internal Format

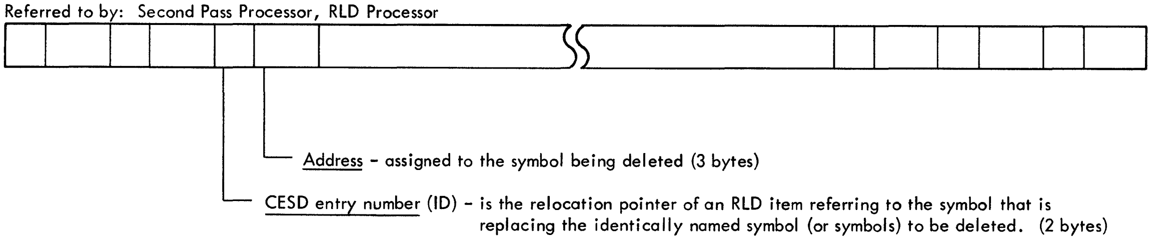
Table 12. Normal Combination of Internal CESD Types

CESD Entry Type	Type Field (byte 8)	Chain Address/ Chain ID (bytes 9-11)	Segment Number (byte 12)	ER Subtype (byte 13)	ddname Pointer/ Chain ID/Length (bytes 14-15)
Section Definition	xxxx x000		1 to 64	Length of control section	
Private Code	xxxx x100		1 to 64	Length of control section	
Common	xxxx x101		1 to 64	Length of common area	
Pseudo Register	xxxx x110		Alignment value (1)	Length of pseudo register	
External Reference	xxxx 0010	Hex 00 or 80		0000 0000	
Label Reference	xxxx x011		1 to 64		CESD entry no. of SD or FC (ID)
NULL	0000 0111				
Replace	xxx1 xxxx			0000 0000	
Insert	xx1x xxxx				
Chain	x1xx xxxx				
Map	1xxx xxxx				
Delete	xxx1 xxxx			0000 1000	
ER - Unmatched Library Member Name	0000 0010	Reverse chain ID		0000 0010	CESD entry no. of next item (ID)
ER - Matched Library Member Name	0000 0010	Reverse chain ID(2)		0000 0011	CESD entry no. of next item (ID)
ER - Unmatched No Call Name	0000 0010			0000 0100	
ER - Matched No Call	0000 0010			0000 0101	
ER - Never Call	0000 0010			0000 0110	
ER - Overlay Control Statement	0000 0010	Address of next item in the chain		1001 0000	
ERE - Alias Control Statement	0000 0010	Address of next item in the chain		1010 0000	
ERE - ddname from Library or Include Statement	0000 0010			1011 0000	Forward chain PTR (Library only)
ER - Include Control Statement w/o Pointer	0000 0010	Address of next item in the chain		1100 0000	
ER - Include Control Statement with Pointer	0000 0010	Address of next item in the chain		1101 0000	Pointer to library's ddname
ER - Replace Control Statement (3)	0000 0010	Address of next item in the chain		1100 0000	
ER - Control Delete (4)	0000 0010	Address of next item in the chain		1110 1000	
ER - Change Control Statement (3)	0000 0010	Address of next item in the chain		1111 0000	

1. Alignment Value - Specifies boundary alignment
   of the pseudo register.
   00 = byte alignment
   01 = halfword alignment
   03 = full-word alignment
   07 = double-word alignment
2. BLDL has been issued for this member name if bit 64 is set to 1.
3. Two CESD entries are made for each Replace or Change control statement, one entry for each symbol.
4. This entry results from a Replace or Change control statement containing only a single symbolic name.

Delink Table

Built by: RLD Processor (Delink Routine),
Referred to by: Second Pass Processor, RLD Processor



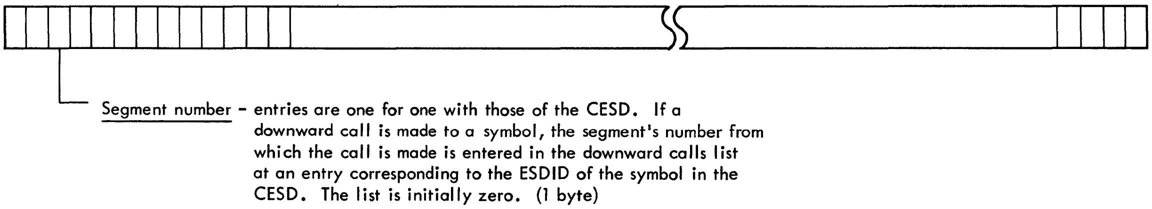
Address - assigned to the symbol being deleted (3 bytes)

CESD entry number (ID) - is the relocation pointer of an RLD item referring to the symbol that is replacing the identically named symbol (or symbols) to be deleted. (2 bytes)

Figure 41. Delink Table

Downward Calls List

Built by and referred to by IEWLCENS routine

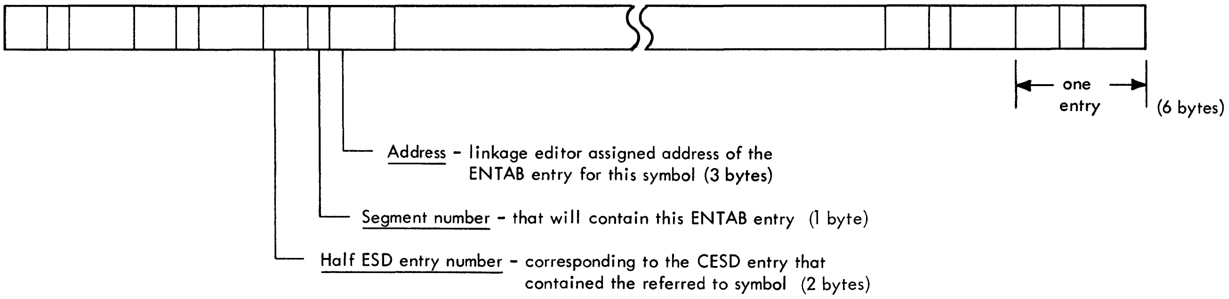


Segment number - entries are one for one with those of the CESD. If a downward call is made to a symbol, the segment's number from which the call is made is entered in the downward calls list at an entry corresponding to the ESDID of the symbol in the CESD. The list is initially zero. (1 byte)

Figure 42. Downward Calls List

Entry List

Built by and referred to by Second Pass Processor



Address - linkage editor assigned address of the ENTAB entry for this symbol (3 bytes)

Segment number - that will contain this ENTAB entry (1 byte)

Half ESD entry number - corresponding to the CESD entry that contained the referred to symbol (2 bytes)

Figure 43. Entry List

Entry Table (ENTAB)

Built by Second Pass Processor

<table>
<tr><td colspan="2">Unconditional branch to last entry BC 15, DISP (15,0)</td><td colspan="2">Address of referred to symbol</td><td>"to" seg number</td><td>Previous Caller (zero initially)</td></tr>
<tr><td colspan="2">Unconditional branch to last entry BC 15, DISP (15,0)</td><td colspan="2">Address of referred to symbol</td><td>"to" seg number</td><td>Previous Caller (zero initially)</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td></td><td></td></tr>
<tr><td colspan="2">Unconditional branch to last entry-BC 15, DISP (15,0)</td><td colspan="2">Address of referred to symbol</td><td>"to" seg number</td><td>Previous Caller (zero initially)</td></tr>
<tr><td>SVC 45</td><td colspan="2">L 15, 4 (0,15) Loads GR15 with the value of the ADCON</td><td>BCR 15,15</td><td>"from" seg no</td><td>Address of segment table (SEGTAB)</td></tr>
<tr><td>2 bytes</td><td>2 bytes</td><td>2 bytes</td><td>2 bytes</td><td>1 byte</td><td>3 bytes</td></tr>
</table>

DISP -- is the displacement, in bytes, of this entry from the last entry.

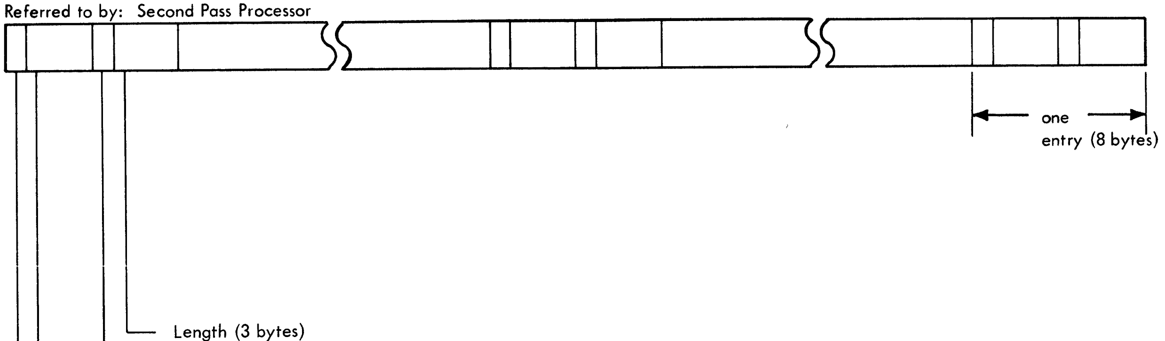
"to" segment number -- is the number of the segment containing the symbol being referred to.

"from" segment number -- is the number of the segment that contains this entry table.

Figure 44. Entry Table (ENTAB)

Half External Symbol Dictionary

Built by: Intermediate Output Processor
Referred to by: Second Pass Processor



Length (3 bytes)

Segment Number* - segment in which this symbol appears. Segment number =
1 in non-overlay programs. (1 byte)

Linkage Editor assigned address - of this symbol (absolute value of the address constant) (3 bytes)

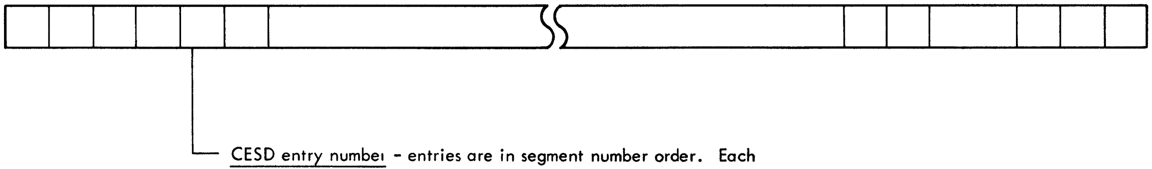
Indicator-Type - Bit zero is not used. Bits 1, 2 and 3 are used as an indicator field that applies to:
SD,PC - Bit 1 = 0 -- this control section (SD or PC) does not have
the highest CESD entry number with text in this segment
= 1 -- this control section (SD or PC) has the
highest CESD entry number in this segment
SD,PC or CM - Bit 2 = 0 -- relative relocation constant is a positive value
= 1 -- relative relocation constant is in
complemented form
PC delete - Bit 3 = 1 -- indicates that this unnamed control section
is a SEGTAB or ENTAB.

Bits 4, 5, 6 and 7 are used to specify the entry type:
0000 = Section Definition (SD)
0010 = External Reference (ER) - all fields are zero except type
0011 = Label Reference (LR)
0100 = Private Code (PC)
0101 = Common (CM)
* 0110 = Pseudo Register (PR) - the segment number field contains a byte alignment value as follows:
0 = byte alignment
1 = half word alignment
3 = full word alignment
7 = double word alignment
0111 = Null - all fields are zero except type

Figure 45. Half External Symbol Dictionary

High ID Table

Built and referred to by Intermediate Output Processor



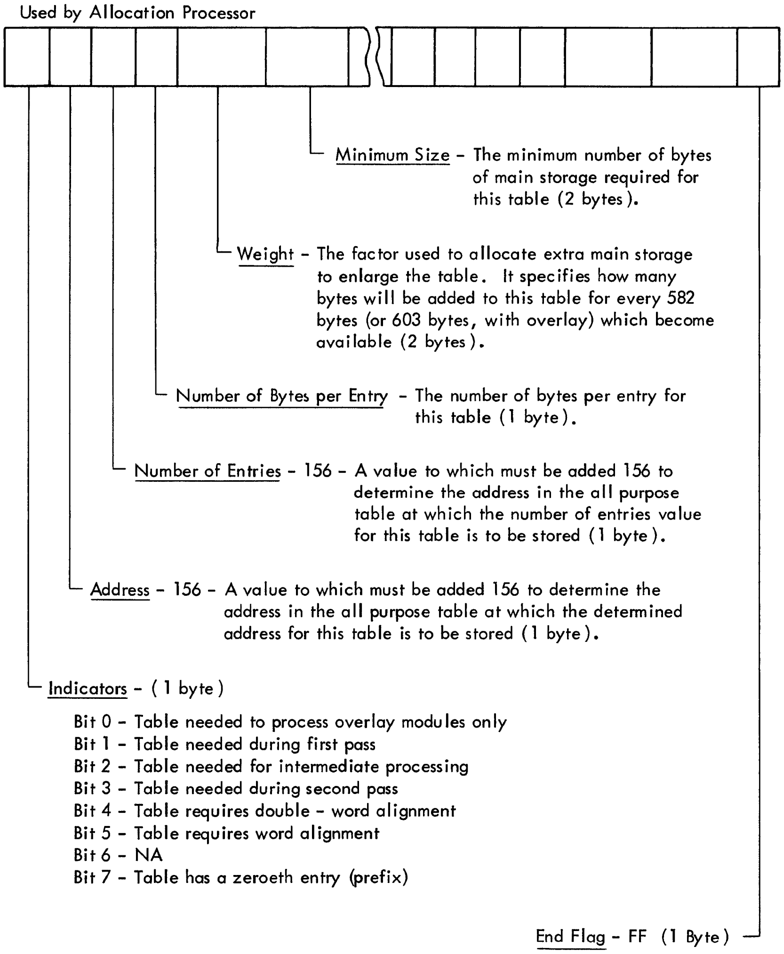
CESD entry numbei - entries are in segment number order. Each
entry contains the highest CESD entry number
(ID) assigned to a section definition (SD or PC)
within that segment. (2 bytes)

Note: If segment does not contain text, its corresponding entry contains zero.

Figure 46. High ID Table

Level F Main Storage Allocation Table

Used by Allocation Processor



Minimum Size - The minimum number of bytes of main storage required for this table (2 bytes).

Weight - The factor used to allocate extra main storage to enlarge the table. It specifies how many bytes will be added to this table for every 582 bytes (or 603 bytes, with overlay) which become available (2 bytes).

Number of Bytes per Entry - The number of bytes per entry for this table (1 byte).

Number of Entries - 156 - A value to which must be added 156 to determine the address in the all purpose table at which the number of entries value for this table is to be stored (1 byte).

Address - 156 - A value to which must be added 156 to determine the address in the all purpose table at which the determined address for this table is to be stored (1 byte).

Indicators - (1 byte)

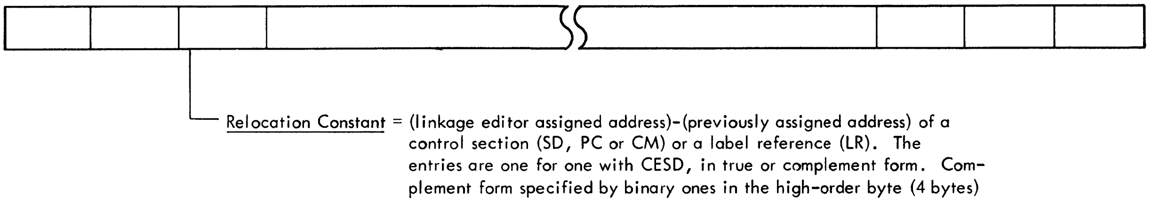
Bit 0 - Table needed to process overlay modules only
Bit 1 - Table needed during first pass
Bit 2 - Table needed for intermediate processing
Bit 3 - Table needed during second pass
Bit 4 - Table requires double - word alignment
Bit 5 - Table requires word alignment
Bit 6 - NA
Bit 7 - Table has a zeroeth entry (prefix)

End Flag - FF (1 Byte)

Figure 47. Level F Main Storage Allocation Table

Relative Relocation Constant Table

Built by and referred to by Address Assignment Processor



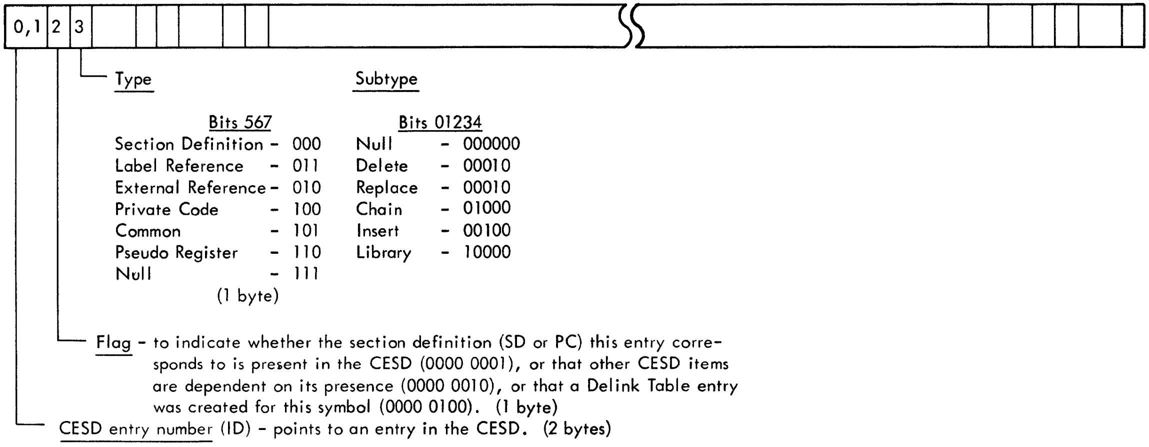
Relocation Constant = (linkage editor assigned address)-(previously assigned address) of a control section (SD, PC or CM) or a label reference (LR). The entries are one for one with CESD, in true or complement form. Complement form specified by binary ones in the high-order byte (4 bytes)

Figure 48. Relative Relocation Constant Table

Renumbering Table

Built by: ESD Processor

Referred to by: TXT, RLD, END and ESD Processor



Type

Subtype

Bits 567		Bits 01234	
Section Definition	- 000	Null	- 000000
Label Reference	- 011	Delete	- 00010
External Reference	- 010	Replace	- 00010
Private Code	- 100	Chain	- 01000
Common	- 101	Insert	- 00100
Pseudo Register	- 110	Library	- 10000
Null	- 111		
(1 byte)			

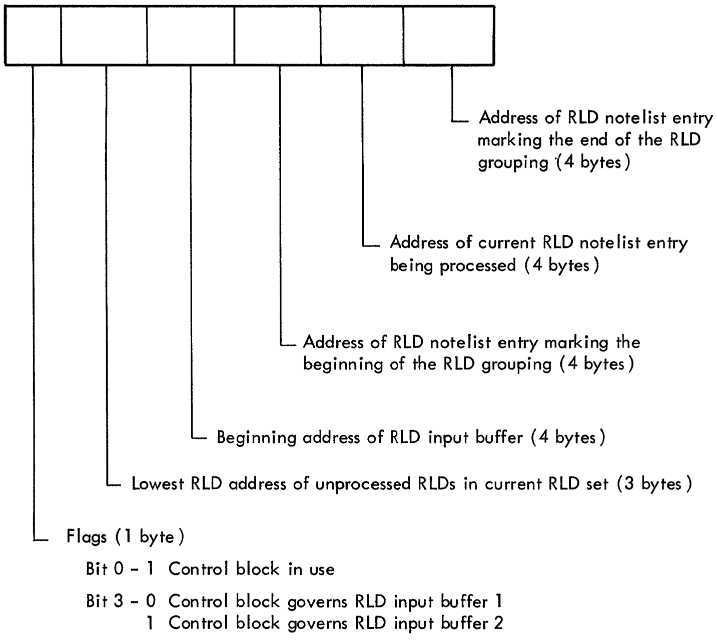
Flag - to indicate whether the section definition (SD or PC) this entry corresponds to is present in the CESD (0000 0001), or that other CESD items are dependent on its presence (0000 0010), or that a Delink Table entry was created for this symbol (0000 0100). (1 byte)

CESD entry number (ID) - points to an entry in the CESD. (2 bytes)

Figure 49. Renumbering Table

RLD Input Control Block*

Build and referred to by second pass RLD processor



Address of RLD notelist entry marking the end of the RLD grouping (4 bytes)

Address of current RLD notelist entry being processed (4 bytes)

Address of RLD notelist entry marking the beginning of the RLD grouping (4 bytes)

Beginning address of RLD input buffer (4 bytes)

Lowest RLD address of unprocessed RLDs in current RLD set (3 bytes)

Flags (1 byte)

Bit 0 - 1 Control block in use

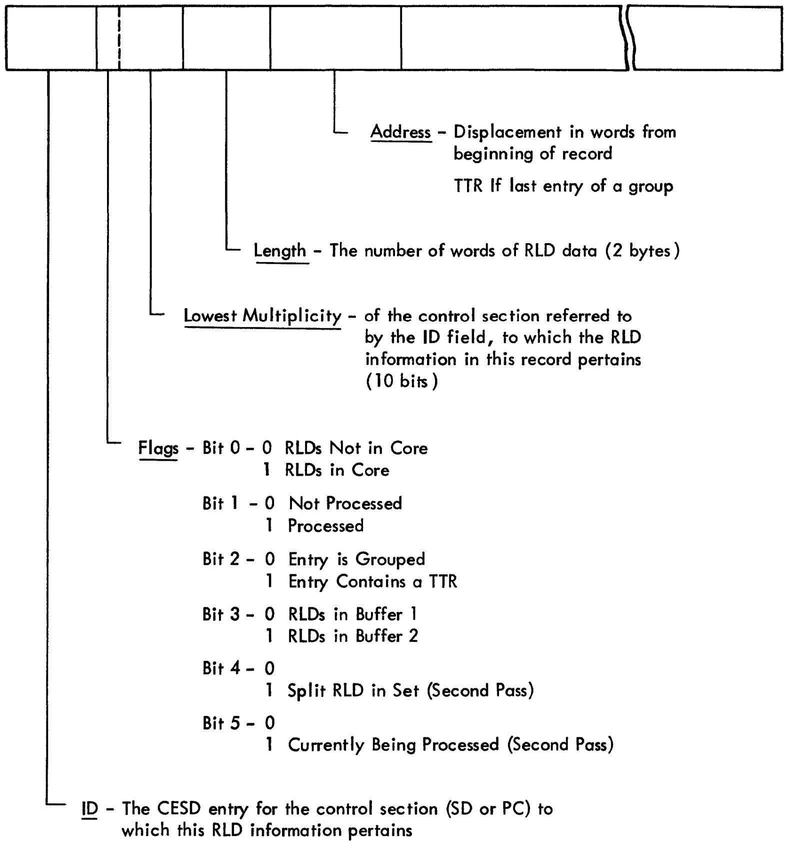
Bit 3 - 0 Control block governs RLD input buffer 1
1 Control block governs RLD input buffer 2

* There is a control block for each of two input buffers.

Figure 50. RLD Input Control Block

Level F RLD Note List

Built and referred to by First Pass RLD Processor



Address - Displacement in words from beginning of record

TTR If last entry of a group

Length - The number of words of RLD data (2 bytes)

Lowest Multiplicity - of the control section referred to by the ID field, to which the RLD information in this record pertains (10 bits)

Flags - Bit 0 - 0 RLDs Not in Core
1 RLDs in Core

Bit 1 - 0 Not Processed
1 Processed

Bit 2 - 0 Entry is Grouped
1 Entry Contains a TTR

Bit 3 - 0 RLDs in Buffer 1
1 RLDs in Buffer 2

Bit 4 - 0
1 Split RLD in Set (Second Pass)

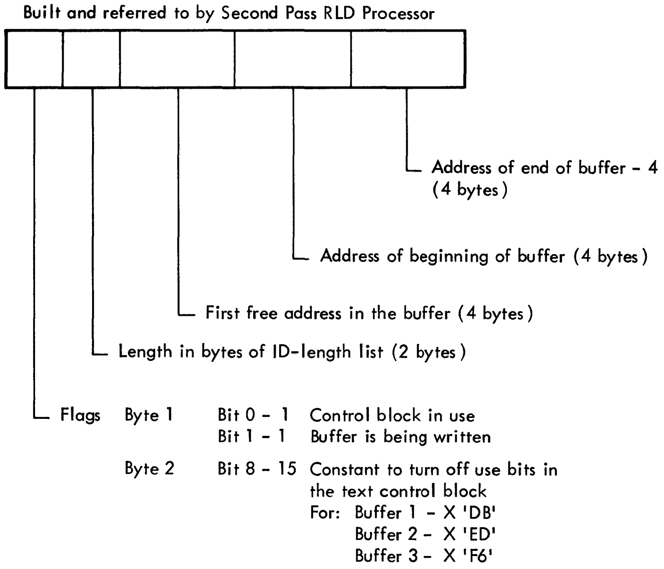
Bit 5 - 0
1 Currently Being Processed (Second Pass)

ID - The CESD entry for the control section (SD or PC) to which this RLD information pertains

Figure 51. Level F RLD Note List

RLD Output Control Block *

Built and referred to by Second Pass RLD Processor

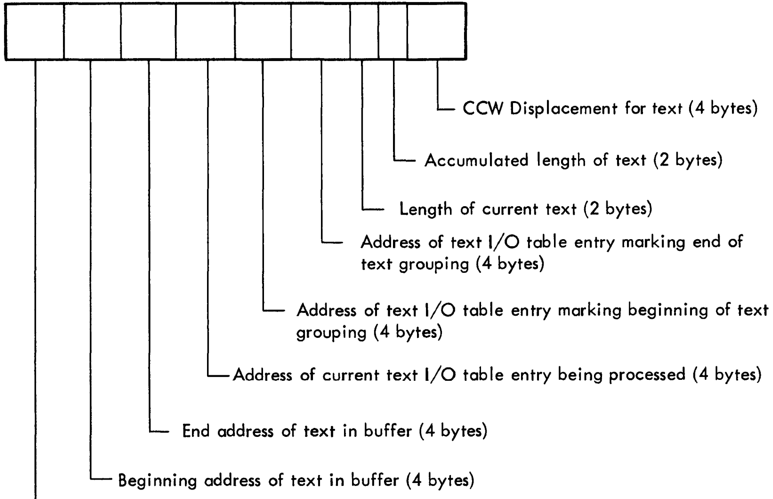

Address of end of buffer - 4 (4 bytes)
Address of beginning of buffer (4 bytes)
First free address in the buffer (4 bytes)
Length in bytes of ID-length list (2 bytes)
Flags Byte 1 Bit 0 - 1 Control block in use
Bit 1 - 1 Buffer is being written
Byte 2 Bit 8 - 15 Constant to turn off use bits in the text control block
For: Buffer 1 - X 'DB'
Buffer 2 - X 'ED'
Buffer 3 - X 'F6'

* There is a control block for each of three RLD output buffers.

Figure 52. RLD Output Control Block

Second Pass Text Control Block*

Built and referred to by Second Pass Text Processor



CCW Displacement for text (4 bytes)

Accumulated length of text (2 bytes)

Length of current text (2 bytes)

Address of text I/O table entry marking end of text grouping (4 bytes)

Address of text I/O table entry marking beginning of text grouping (4 bytes)

Address of current text I/O table entry being processed (4 bytes)

End address of text in buffer (4 bytes)

Beginning address of text in buffer (4 bytes)

Flags - (4 bytes)	Byte 1	Bit 0 - 1	Control block in use
		1 - 1	Text being written
		2 - 1	Text being read
		3 - 1	Text has RLDs
		4 - 1	Text is first of group,
		5 - 1	Text is last of group
		6 - 1	Text is last in segment
		7 - 1	Text is last in load module
	Byte 2	Bit 0 - 1	XDAP write needed
		1 - 1	Dummy write needed
		2 - 1	RLD output buffer 1 is being used
		3 - 1	RLD output buffer 2 is being used
		4 - 1	RLD output buffer 3 is being used
		5 - 1	RLD output buffer 1 contains ID-length list for this text
		6 - 1	RLD output buffer 2 contains ID-length list for this text
		7 - 1	RLD output buffer 3 contains ID-length list for this text
	Byte 3	Bit 0 - 1	RLD input buffer 1 contains RLDs for this text
		1 - 1	RLD input buffer 1 contains processed RLDs for this text
		2 - 1	RLD input buffer 2 contains RLDs for this text
		3 - 1	RLD input buffer 2 contains processed RLDs for this text
		4 - 1	There is more text to process after current text

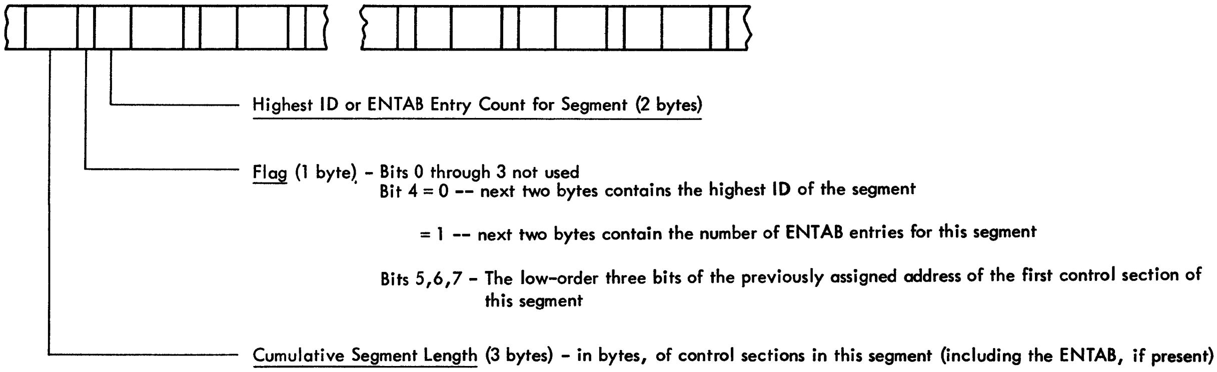
* There are two text control blocks - - one for current text being processed, another for next text to be processed or text just processed.

Figure 53. Second Pass Text Control Block

Segment Length Table

Built and referred to by address assignment processor

Appearance of table after assignment of control section addresses



Highest ID or ENTAB Entry Count for Segment (2 bytes)

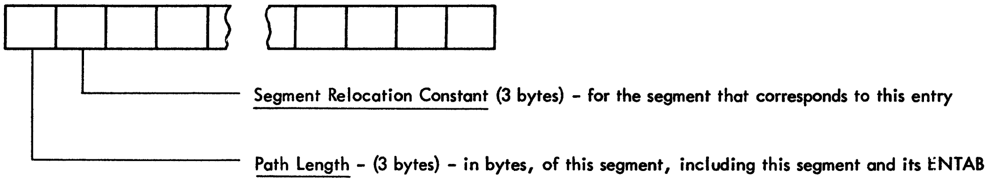
Flag (1 byte) - Bits 0 through 3 not used
Bit 4 = 0 -- next two bytes contains the highest ID of the segment

= 1 -- next two bytes contain the number of ENTAB entries for this segment

Bits 5,6,7 - The low-order three bits of the previously assigned address of the first control section of this segment

Cumulative Segment Length (3 bytes) - in bytes, of control sections in this segment (including the ENTAB, if present)

Appearance of table after segment addresses are determined



Segment Relocation Constant (3 bytes) - for the segment that corresponds to this entry

Path Length - (3 bytes) - in bytes, of this segment, including this segment and its ENTAB

Figure 54. Segment Length Table

Segment Table (SEGTAB)
Built by Intermediate Output Processor

<table>
<tr><td>TEST Indicator</td><td></td><td colspan="4">Address of Data Control Block (DCB) used to load module *</td></tr>
<tr><td colspan="2"></td><td colspan="4">Address of note list *</td></tr>
<tr><td colspan="2">Last segment number of region 1</td><td>Highest segment no. in storage-region 1</td><td>Last segment number of region 2</td><td colspan="2">Highest segment no. in storage-region 2</td></tr>
<tr><td colspan="2">Last segment number of region 3</td><td>Highest segment no. in storage-region 3</td><td>Last segment number of region 4</td><td colspan="2">Highest segment no. in storage-region 4</td></tr>
<tr><td colspan="2">Zero</td><td colspan="4">(Not used in the Fixed-Task Supervisor) *</td></tr>
<tr><td colspan="6">(Not used in the Fixed-Task Supervisor) *</td></tr>
<tr><td colspan="2">Previous segment * number for segment 1</td><td colspan="3">Zero</td><td>Status Indicator</td></tr>
<tr><td colspan="2">Previous segment number for segment 2</td><td colspan="3">Address of entry table entry (when caller chain exists) *</td><td>Status Indicator</td></tr>
<tr><td colspan="2">•
•</td><td colspan="3">•
•</td><td>•
•</td></tr>
<tr><td colspan="2">Previous segment number for segment N</td><td colspan="3">Address of entry table entry (when caller chain exists) *</td><td>Status Indicator</td></tr>
</table>

4 bytes

TEST indicator -- specifies that this module is "under test" using TESTRAN. (Bit 1) Initialized by program fetch.
Highest segment no. in storage -- is initially set to 00 except for region 1 which is initially set to 01 by linkage editor.

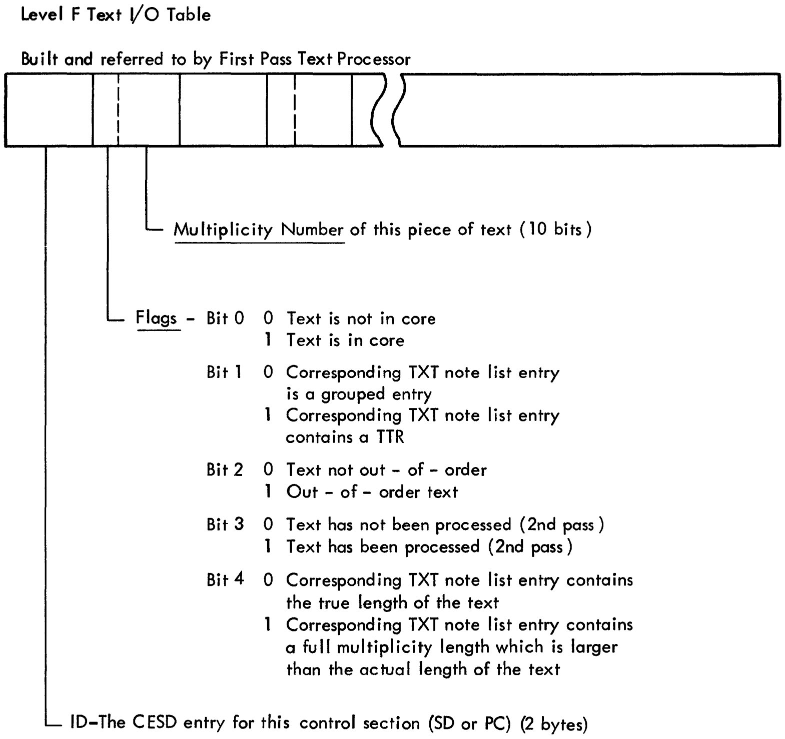
Status indicator -- indicates the status of this segment with the two last bits of the entry table address field as follows:

00 -- segment is in main storage as a result of a branch to the segment.
10 -- segment is in main storage, no caller chain exists.
01 -- segment is not in main storage, but is scheduled to be loaded.
11 -- segment is not in main storage.

The status indicator for segment 1 is initially set to 10, all the rest are initially set to 11.

* set to zero by linkage editor

Figure 55. Segment Table (SEGTAB)



Level F Text I/O Table

Built and referred to by First Pass Text Processor

Multiplicity Number of this piece of text (10 bits)

Flags - Bit 0 0 Text is not in core
1 Text is in core

Bit 1 0 Corresponding TXT note list entry is a grouped entry
1 Corresponding TXT note list entry contains a TTR

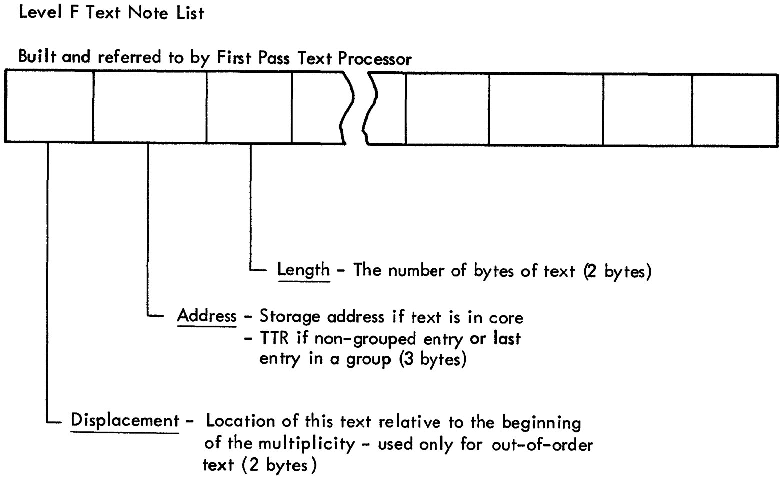
Bit 2 0 Text not out - of - order
1 Out - of - order text

Bit 3 0 Text has not been processed (2nd pass)
1 Text has been processed (2nd pass)

Bit 4 0 Corresponding TXT note list entry contains the true length of the text
1 Corresponding TXT note list entry contains a full multiplicity length which is larger than the actual length of the text

ID-The CESD entry for this control section (SD or PC) (2 bytes)

Figure 56. Level F Text I/O Table



Level F Text Note List

Built and referred to by First Pass Text Processor

Length - The number of bytes of text (2 bytes)

Address - Storage address if text is in core
- TTR if non-grouped entry or last entry in a group (3 bytes)

Displacement - Location of this text relative to the beginning of the multiplicity - used only for out-of-order text (2 bytes)

Figure 57. Level F Text Note List

Partitioned Organization Directory Record
As received from BLDL

Byte				
0	Name of Load Module (Member or Alias Name)			
4				
8	Relative (to beginning of data set) track address of module (TTR)			Concatenation number
12	Byte of binary zeros. *	Alias indicator and miscellaneous info	Relative (to beginning of data set) track address of first text record	
16	Continuation of track address	Byte of binary Zeros	Relative (to beginning of data set) track address of note list or scatter-	
20	translation record	Number of entries in note list **	Module attributes 0,1,2,3,4,5,6,7,8,9,10,11,12,13,R,15	
24	Total contiguous quantity of main storage required by the module			Length (in bytes) of first text record
28	Continuation of length	Module's linkage editor assigned entry point address		
32	Linkage editor assigned origin of first text record			
	For load modules in scatter format add:			Length of Scatter
36	List (in bytes)	Length of translation table (in bytes)		ESDID (CESD entry number of control
40	section name) for first text record	ESDID (CESD entry number of control section name) containing entry point		
	For load modules with RENT or REUS attribute and alias names add;			Entry point address
44	of the member name		Member Name	
48				
52				
	SSI Bytes - Aligned on a halfword boundary at the end of the PDS record			

Alias indicator and miscellaneous information:

1. Alias indicator -- 0 signifies none, 1 signifies alias -- bit 0
2. Number of relative disk addresses (TTR) in user data field -- bits 1,2
3. Length of user data field (in halfwords) -- bits 3-7

PODS Directory Record size:

Block format	36 bytes	(with alias names, 46 bytes)
Scatter format	44 bytes	(with alias names, 54 bytes)

For SSI, add 4 bytes to sizes given above

*This is normally a zero byte inserted to maintain halfword boundaries. If the DCB operand was specified as zero and the name was found in the link library, this byte will contain a 1; if the name was found in the job library, this byte will contain a 2.

**This byte contains zero if load module is not in overlay.

R=Reserved

Figure 58. Partitioned Organization Directory Record

Module Attributes

Bit Number	Attributes	Bit Setting	Indication
0	RENT	0	Not re-enterable
		1	Re-enterable
1	REUS	0	Not reusable
		1	Reusable
2	OVLY	0	Not an overlay module
		1	Overlay module
3	TEST	0	Not under test
		1	Under test
4	LOAD	0	Not only loadable
		1	Only loadable *
5	Format	0	Block format
		1	Scatter Format
6	Executable	0	Not executable
		1	Executable
7	Format	0	Module contains more than one text record and/or RLD record(s).
		1	Module contains only one text record and no RLD record.
8	Compatibility	0	Module can be processed by all levels of linkage editor.
		1	Module cannot be reprocessed by linkage editor E.
9	Format	0	Linkage editor assigned origin of first text record is not zero.
		1	Linkage editor assigned origin of first text record is zero.
10	Format	0	Linkage editor assigned entry point is not zero.
		1	Linkage editor assigned entry point is zero.
11	Format	0	Module contains RLD record(s)
		1	Module does not contain an RLD record.
12	Editability	0	Module can be reprocessed by linkage editor.
		1	Module cannot be reprocessed by linkage editor.
13	Format	0	Module does not contain TESTRAN symbol records.
		1	Module contains TESTRAN symbol records.
14	Reserved		
15	REFR	0	Module is not refreshable
		1	Module is refreshable

*Module can be loaded only with the LOAD macro-instruction. When the module is in main storage, it is entered directly, not through the use of a XCTL, LINK or ATTACH macro-instruction.

Figure 59. Module Attributes

Partitioned Organization Directory Record

As built by linkage editor

Byte				
0 4	Name of load module (Member or alias name)			
8	Relative (to beginning of data set) track address of module. (TTR)			Alias indicator and miscellaneous info.
12	Relative (to beginning of data set) track address of first text record. (TTR)			Byte of binary zeros.
16	Relative (to beginning of data set) track address of note list or Scatter/translation record. (TTR)			Number of entries in note list.*
20	Module Attributes (see below) 0,1,2,3,4,5,6,7,8,9,10,11,12,13,R,15		Total contiguous main storage required	
24	for the module.	Length (in bytes) of first text record.		Module's linkage
28	editor assigned entry point address		Linkage editor assigned origin of	
32	first text record			

For load modules in scatter format add:

Byte				
		Length of scatter list (in bytes)		Length of transla-
36	tion table (in bytes)	ESDID (CESD entry number of control section name) for first text record.		ESDID (CESD entry number of control
40	section name) containing entry point.			

For load modules with RENT or REUS attribute and Alias names add:

Byte				
		Entry point address of the member name		
44 48	Member name			

SSI Bytes - Aligned on a half-word boundary at the end of the PDS record.

Alias indicator and miscellaneous information:

1. Alias indicator -- 0 signifies none, 1 signifies alias -- bit 0
2. Number of relative track addresses in user data field -- bits 1,2
3. Length of user data field (in halfwords) -- bits 3-7

Note: The I/O conventions and record formats for Linkage Editor (E) and Linkage Editor (F) are the same.

PODS Directory Record size:

Block format	34 bytes	(when rounded to a half-word boundary)
Block format with alias names	44 bytes	
Scatter format	42 bytes	
Scatter format with alias names	52 bytes	

For SSI, add 4 bytes to sizes given above

R=Reserved

*This byte contains zero if load module is not in overlay.

Figure 60. Partitioned Organization Directory Record

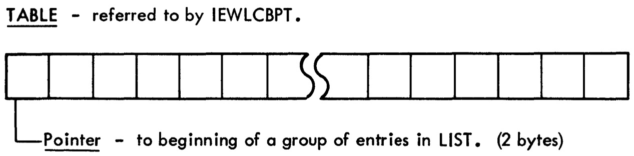


TABLE - referred to by IEWLCBPT.
Pointer - to beginning of a group of entries in LIST. (2 bytes)

Figure 61. Table - Referred to by IEWLCBPT

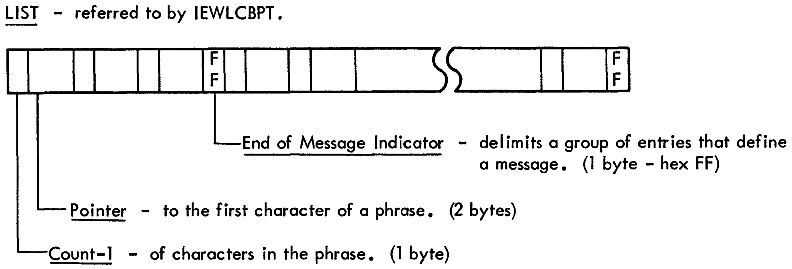

LIST - referred to by IEWLCBPT.
FF
FF
End of Message Indicator - delimits a group of entries that define a message. (1 byte - hex FF)
Pointer - to the first character of a phrase. (2 bytes)
Count-1 - of characters in the phrase. (1 byte)

Figure 62. LIST - Referred to by IEWLCBPT

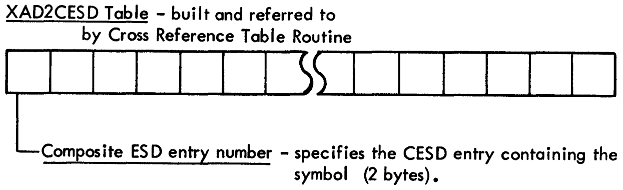

XAD2CESD Table - built and referred to by Cross Reference Table Routine
Composite ESD entry number - specifies the CESD entry containing the symbol (2 bytes).

Figure 63. XAD2CESD Table -- Built and Referred to by Cross Reference Table Routine

This section contains information that may be useful in diagnosing difficulties with the linkage editor program. Included are: register contents at entry to modules, charts describing buffer and table allocation, and an error message -- module cross reference table.

Table 13. General Register Contents at Entry to Module

Module Entry Point	Register Contents
IEWLMADA	2 -- Address of all purpose table
IEWLMBTP	2 -- Address of all purpose table 14 -- Return address 15 -- Entry point address
IEWLMEND	2 -- Address of all purpose table 4 -- Length of any no-length control section 5 -- ID of the assembled address of the module entry point 13 -- Address of save area 14 -- Return address 15 -- Entry point address
IEWLMENS	2 -- Address of all purpose table 13 -- Save area address 14 -- Return address
IEWLMENT	2 -- Address of all purpose table 13 -- Save area address 14 -- Return address
IEWLMESD	2 -- Address of all purpose table 4 -- Byte count of ESD information 5 -- ID of first ESD item to be processed 6 -- Address of first ESD item to be processed 13 -- Save area address 14 -- Return address 15 -- Entry point address
IEWLMFNL	2 -- Address of all purpose table
IEWLMINC	2 -- Address of all purpose table 15 -- Entry point address
IEWLMINP	2 -- Address of all purpose table 15 -- Entry point address
IEWLMINT	1 -- Pointer to parameter list 13 -- Save area address 14 -- Return address
IEWLMMAP	2 -- Address of all purpose table 14 -- Return address if entry is from IEWLMOUNT 15 -- Address of entry point
IEWLMOPT	1 -- Pointer to parameter list 2 -- Address of all purpose table 15 -- Entry point address

(Continued)

Table 13. General Register Contents at Entry to Module (Continued)

Module Entry Point	Register Contents
IEWLMOUNT	2 -- Address of all purpose table
IEWLMRAT	2 -- Address of all purpose table
	4 -- Byte count of RLD input
	6 -- Storage address of RLD input
	14 -- Return address
IEWLMRCG	2 -- Address of all purpose table
	6 -- Address of ESD item being processed
	10 -- Address of beginning of REPLACE/CHANGE chain
	13 -- Entry point address
IEWLMREL	1 -- HESD address of either first ENTAB entry (if overlay) or last HESD entry + 8
	2 -- Address of all purpose table
	3 -- Address of text control block for current text
	12 -- Address of IEWLMSCD
	13 -- Address of APT register save area (REGSA)
	14 -- Return address
	15 -- Entry point address (IEWLMREL)
SCDENTAB	2 -- Address of all purpose table
	3 -- Address of text control block for current text
	8 -- Address of control block for previous control record
	12 -- Address of IEWLMSCD
	13 -- Address of APT register save area (REGSA)
	14 -- Return address
	15 -- Entry point address (IEWLMREL)
IEWLMROU	1 -- Pointer to parameter list
	14 -- Return address
	15 -- Entry point address (IEWLMROU)
IEWLCR01	0 -- DECB address
IEWLCR02	1 -- DCB address
	2 -- Address of all purpose table
	14 -- Return address
	15 -- Entry point address (IEWLCR01 or IEWLCR02)
IEWLCR03	1 -- Address of IOB for XDAP
	2 -- Address of all purpose table
	14 -- Return address
	15 -- Entry point address (IEWLCR03)
IEWLEPNT	2 -- Address of all purpose table
	14 -- Return address
	15 -- Entry point address (IEWLEPNT)
IEWLMLOG	0 -- Error Code
	1 -- Address of first symbol (optional)
	2 -- Address of all purpose table
	13 -- Address of second symbol (optional)
	14 -- Return address

(Continued)

Table 13. General Register Contents at Entry to Module (Continued)

Module Entry Point	Register Contents
IEWLMSCD	1 -- Address of first HESD ENTAB entry (overlay) or first entry beyond HESD 2 -- Address of all purpose table
GETIDMUL	0 -- Indicator: 0 - prime read; 1 - lookahead 1 -- Text control block address
RDTXT	1 -- Current TXTIOT entry address
RDRLD	1 -- Current RLD notelist entry address
GETIDMUL RDTXT RDRLD	2 -- Address of all purpose table 3 -- Address of control block for current text 4 -- Address of control block for previous or next text
WRTCRRLD	1 -- Address of control block for buffer to be written
WRTCRRLD	12 -- Address of IEWLMSCD 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 -- Address of IEWLMREL
IEWLMSCN	1 -- Address of column 1 of input record 2 -- Address of all purpose table 15 -- Entry point address
IEWLMSYM	2 -- Address of all purpose table 13 -- Save area address 14 -- Return address 15 -- Entry point address
IEWLMTXT	2 -- Address of all purpose table 3 -- Assembled address of first byte of text 6 -- TD of current text record 7 -- Base register of IEWLMTXT (entry register) 12 -- Base register of IEWLMRAT 14 -- Return address

Table 14. Buffer Allocation

BUFFER ALLOCATION - (LINKAGE EDITOR F)

<table>
<tr><th>Initial and Input Processing</th><th>Intermediate Processing</th><th>Second Pass Processing</th></tr>
<tr><td colspan="3">Object Module Buffer 1
3200 bytes (max.) or 800 bytes or 400 bytes (min.)</td></tr>
<tr><td colspan="3">Object Module Buffer 2
3200 bytes (max.) or 800 bytes or 400 bytes (min.)</td></tr>
<tr><td colspan="3">SYSLIN Buffer 1
3200 bytes (max.) or 800 bytes or 400 bytes (min.)</td></tr>
<tr><td colspan="3">SYSLIN Buffer 2
3200 bytes (max.) or 800 bytes or 400 bytes (min.)</td></tr>
<tr><td colspan="3">Print Buffer 1
4840 bytes (max.) or 1216 bytes or 608 bytes (min.)</td></tr>
<tr><td colspan="3">Print Buffer 2
4840 bytes (max.) or 1216 bytes or 608 bytes (min.)</td></tr>
<tr><td colspan="3">RLD Buffer Area
1024 bytes</td></tr>
<tr><td colspan="3">Text Buffer Area
102400 bytes (max.) or 6144 bytes (min.)</td></tr>
</table>

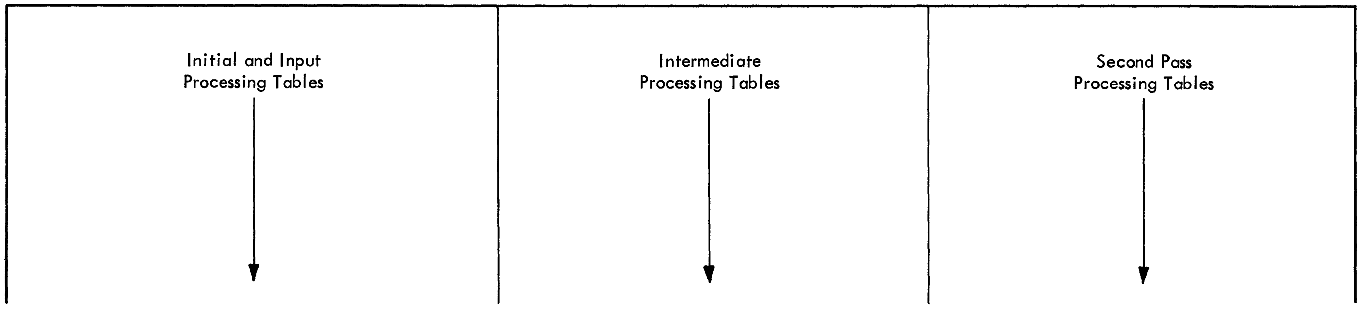

Initial and Input Processing Tables
Intermediate Processing Tables
Second Pass Processing Tables

Table 15. Table Allocation

Table Name	OVLY Only	Order in Coding	Bytes/ Entry	Weight	Present In			Prefix	Align	Size (in bytes)	
					Inp. Proc.	Int. Prcc.	2nd Pass			Min.	Max.
Alias Table	No	4	1	0	No	Yes	Yes	No	Byte	50	50
Calls List	Yes	18	1	20	Yes	Yes	No	No	Word	1368	*
Composite ESD	No	10	16	352	Yes	No	No	Yes	Dlbwd	5600	#
Delink Table	No	3	5	30	Yes	Yes	Yes	Yes	Byte	500	*
Entry List	Yes	12	6	6	No	No	Yes	No	Byte	600	*
Half ESD	No	6	8	176	No	Yes	Yes	No	Dblwd	2800	*
Half ESD Prefix	No	5	8	0	No	Yes	Yes	No	Dblwd	8	8
First Pass RLD Buffer	No	9	1	0	Yes	No	No	No	Word	256	256
Second Pass RLD Buffer	No	11	1	0	No	No	Yes	No	Word	768	768
Relocation Constant Table	No	17	4	88	No	Yes	Yes	Yes	Word	1400	*
Renumbering Table	No	16	4	88	Yes	No	No	Yes	Word	1400	*
Renumbering Table Prefix	No	16	1	0	Yes	Yes	No	No	Word	4	4
RLD Note List I	No	15	9	28	Yes	Yes	No	No	Byte	450	*
RLD Note	No	8 List II	9	112	No	Yes	Yes	No	Byte	1800	*
SEGTA1	Yes	2	1	0	Yes	Yes	Yes	Yes	Byte	256	256
Text I/O	No	1	4	48	Yes	Yes	Yes	No	alqs1i1		

*Maximum is determined by main storage availability

#CESD maximum = $(2^{16}-1) \times 2^4$

Table 16. Error Message -- Module Cross Reference Table

MMS	Module Where Error Occurred
012	IEWLMSCD
022	IEWLMSCD
033	IEWLMENT
053	IEWLMENT
063	IEWLMENT
073	IEWLMENT
083	IEWLMENT
093	IEWLMENT
102	LEWLMEND
113	IEWLMENT
123	IEWLMADA
132	IEWLMADA
143	IEWLMOUT
152	IEWLMENS
161	IEWLMENS
161	IEWLMENS
172	IEWLMENS
182	IEWLMENS
192	IEWLMADA
202	IEWLMADA
212	IEWLMINP
222	IEWLMESD,IEWMINP,IEWLMRAT
232	IEWLMESD,IEWLMINP,IEWLMRAT
241	IEWLMESD
254	IEWLMESD,IEWLMADA
264	IEWLMESD
274	IEWLMINC
284	IEWLMSCN,IEWLMINT
294	IEWLMINT,IEWLMFNL
302	IEWLMSCN
314	IEWLMSCN
324	IEWLMSCN
332	IEWLMSCN
342	IEWLMINC
354	IEWLMRAT
364	IEWLMRAT
374	IEWLMRAT
383	IEWLMRAT
394	IEWLMFNL
404	IEWLMFNL
414	IEWLMFNL
421	IEWLMFNL
432	IEWLMINC
444	IEWLMSCD
461	IEWLMADA
473	IEWLMENT
484	IEWLMINP
492	IEWLMSCN
504	IEWLMFNL
512	IEWLMINC
522	IEWLMINC
532	IEWLMINC
543	IEWLMFNL
594	IEWLMINT,IEWLMINP
611	IEWLMRAT
630	IEWLMMAP
611	IEWLMRAT

This section contains linkage editor input conventions and record formats. The I/O conventions and record formats for Linkage Editor E and Linkage Editor F are the same.)

INPUT CONVENTIONS

Input modules (object or load) to be processed in a single execution of linkage editor must conform with a number of input conventions. Violations of the following are treated as errors by linkage editor:

- All text records of a control section must follow the ESD record containing the SD or PC entry that describes the control section.
- The end of every input module must be marked by an end record (END in object modules, LAST in load modules).
- Each input module may contain only one no-length control section (a control section whose length field in its SD- or PC-type ESD entry contains zeros). The length must be specified on the END record of any module that contains a no-length control section.
- After processing the first text record of a no-length control section, linkage editor will not accept a text record of a different control section within the same input module.
- Any RLD item must be read after the ESD item to which it refers; if it refers to a label within a different control section, it must be read after the ESD item for that control section.
- The language translators must gather RLD items in groups of identical position pointers. No two RLD items having the same P pointer can be separated by an RLD item having a different P pointer.
- Each record of text[1] and each LD- or LR-type ESD record must refer to an SD or PC entry in the ESD.
- The position pointer of every RLD record must point to an SD- or PC-type entry in the ESD.
- No LD or LR may have the same name as an SD or CM.
- All SYM records must be placed at the beginning of an input module. The ESD for an input module containing test translator statements must follow the SYM records and precede the TXT records.
- Linkage editor accepts TXT records that are out of order within a control section, even though linkage editor processing may be affected. TXT records are accepted even though they may overwrite previous text in the same control section. Linkage editor does not eliminate any RLD records that correspond to overwritten text.
- During a single execution of linkage editor, if two or more control sections having the same name are read in, only the first control section is accepted; the subsequent control sections are deleted.
- Linkage editor interpretes common (CM) ESD items (blank or with the same name) as references to a single control section, whose length is the maximum length specified in the CM items of that name (or blank). No text may be contained in a common control section.
- Within an input module, linkage editor does not accept an SD- or PC-type ESD item after the first RLD item is read.

To avoid unnecessary scanning and I/O operations, input modules should conform with the following conventions. Although violations of these rules are not treated as errors, avoiding them will improve the efficiency of linkage editor processing.

- Within an input module, no LD or SD may have the same name as an ER.
- Within an input module, no two ERs may have the same name.
- Within an input module, TXT records may be in the order of the addresses assigned by the language translator. (If TXT records are not in address sequence, each reorigin operation may require additional linkage editor processing time.)
- SYSUT1 record size should be at least as large as SYSLMOD.

[1] A common (CM) control section cannot contain text or external references.

RECORD FORMATS

The following are the card image and load module record formats for the level F version of the linkage editor.

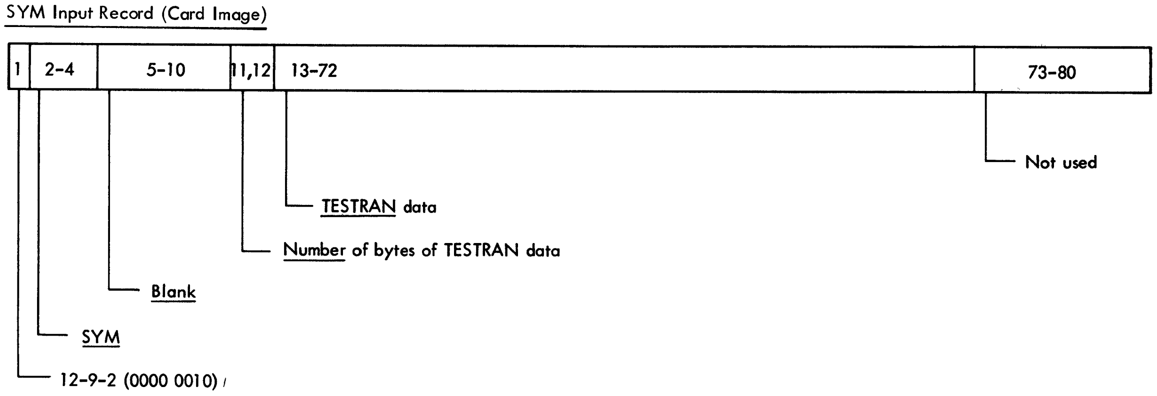


Figure 64. SYM Input Record (Card Image)

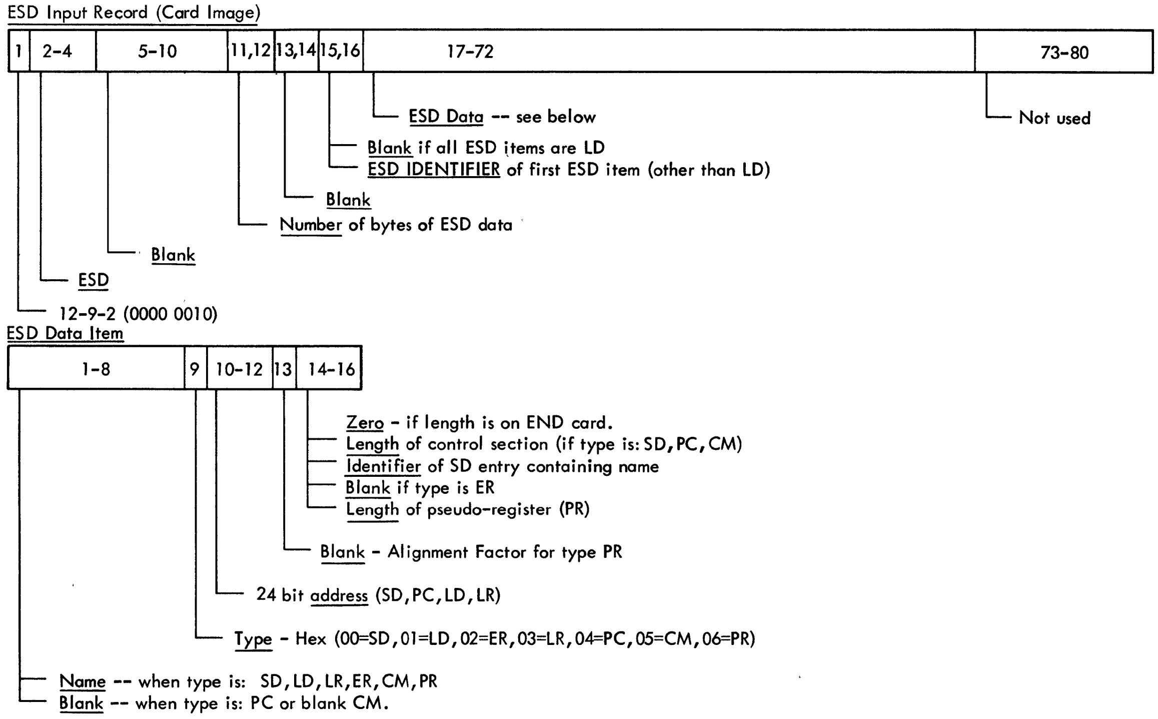

ESD Input Record (Card Image)
1
2-4
5-10
11,12
13,14
15,16
17-72
73-80
ESD Data -- see below
Not used
Blank if all ESD items are LD
ESD IDENTIFIER of first ESD item (other than LD)
Blank
Number of bytes of ESD data
Blank
ESD
12-9-2 (0000 0010)
ESD Data Item
1-8
9
10-12
13
14-16
Zero - if length is on END card.
Length of control section (if type is: SD, PC, CM)
Identifier of SD entry containing name
Blank if type is ER
Length of pseudo-register (PR)
Blank - Alignment Factor for type PR
24 bit address (SD, PC, LD, LR)
Type - Hex (00=SD, 01=LD, 02=ER, 03=LR, 04=PC, 05=CM, 06=PR)
Name -- when type is: SD, LD, LR, ER, CM, PR
Blank -- when type is: PC or blank CM.

Figure 65. ESD Input Record (Card Image)

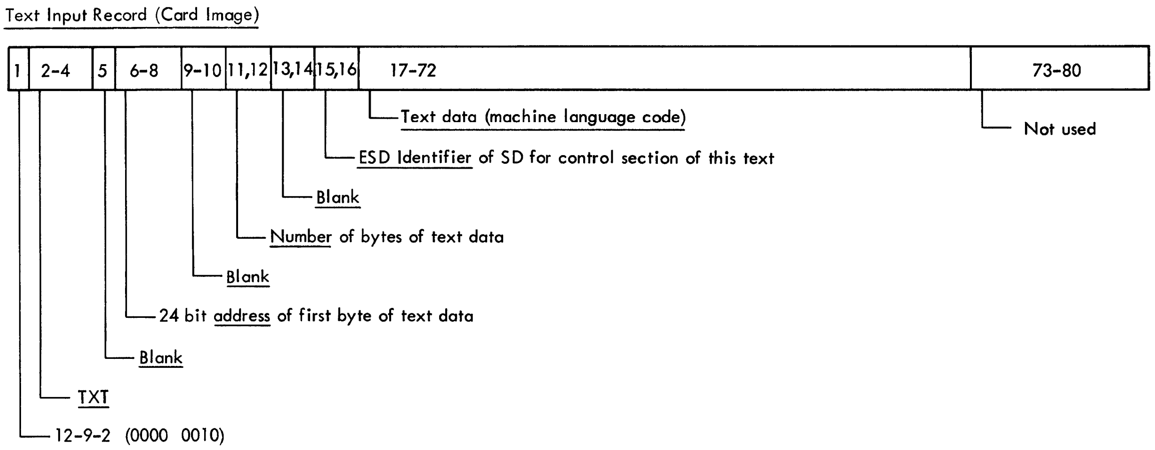

Text Input Record (Card Image)
1
2-4
5
6-8
9-10
11,12
13,14
15,16
17-72
73-80
Text data (machine language code)
Not used
ESD Identifier of SD for control section of this text
Blank
Number of bytes of text data
Blank
24 bit address of first byte of text data
Blank
TXT
12-9-2 (0000 0010)

Figure 66. Text Input Record (Card Image)

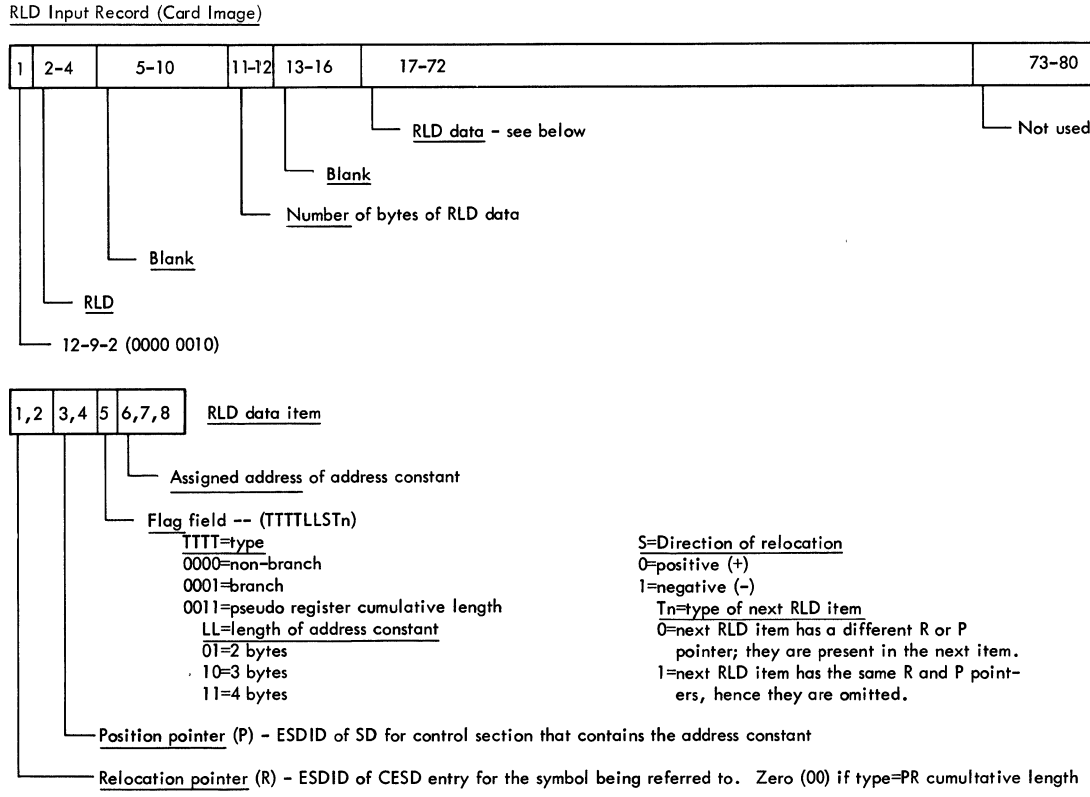
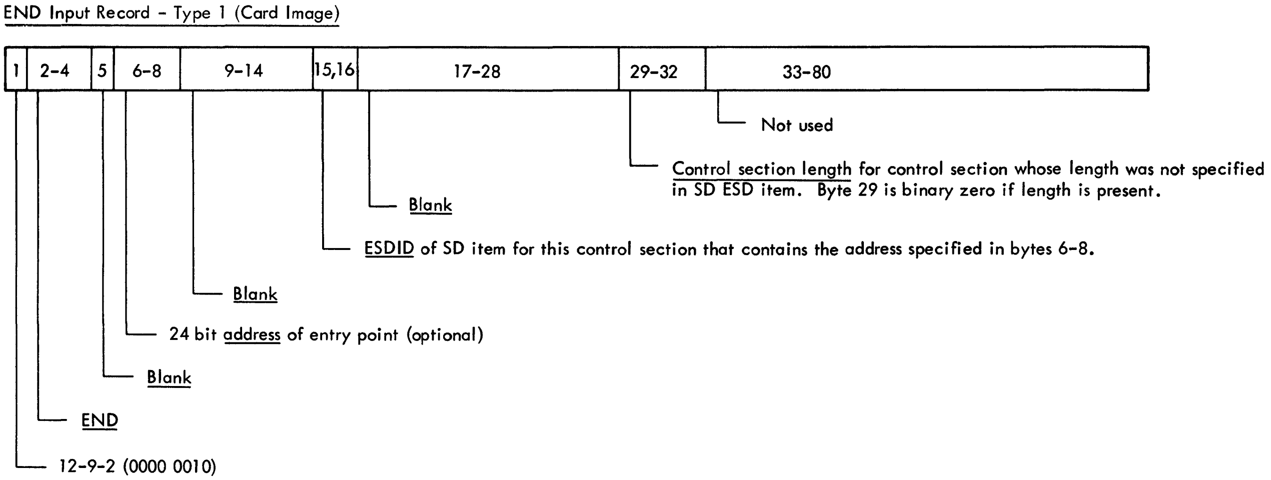

RLD Input Record (Card Image)
1
2-4
5-10
11-12
13-16
17-72
73-80
RLD data - see below
Not used
Blank
Number of bytes of RLD data
Blank
RLD
12-9-2 (0000 0010)
1,2
3,4
5
6,7,8
RLD data item
Assigned address of address constant
Flag field -- (TTTTLLSTn)
TTTT=type
0000=non-branch
0001=branch
0011=pseudo register cumulative length
LL=length of address constant
01=2 bytes
10=3 bytes
11=4 bytes
S=Direction of relocation
0=positive (+)
1=negative (-)
Tn=type of next RLD item
0=next RLD item has a different R or P pointer; they are present in the next item.
1=next RLD item has the same R and P pointers, hence they are omitted.
Position pointer (P) - ESDID of SD for control section that contains the address constant
Relocation pointer (R) - ESDID of CESD entry for the symbol being referred to. Zero (00) if type=PR cumultative length

Figure 67. RLD Input Record (Card Image)



Not used

Control section length for control section whose length was not specified in SD ESD item. Byte 29 is binary zero if length is present.

Blank

ESDID of SD item for this control section that contains the address specified in bytes 6-8.

Blank

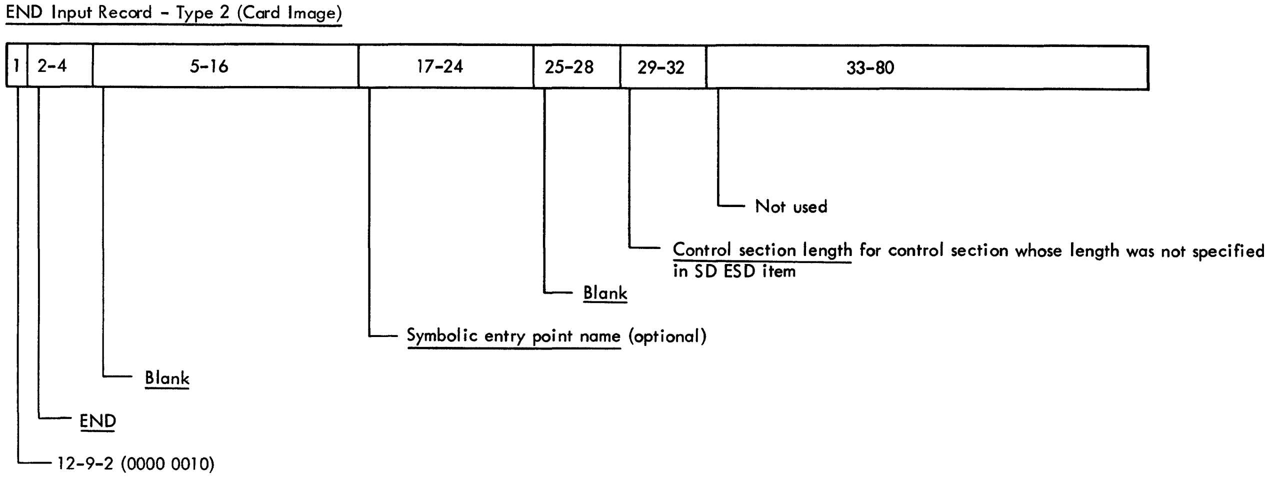
24 bit address of entry point (optional)

Blank

END

12-9-2 (0000 0010)

Figure 68. END Input Record - Type 1 (Card Image)



Not used

Control section length for control section whose length was not specified in SD ESD item

Blank

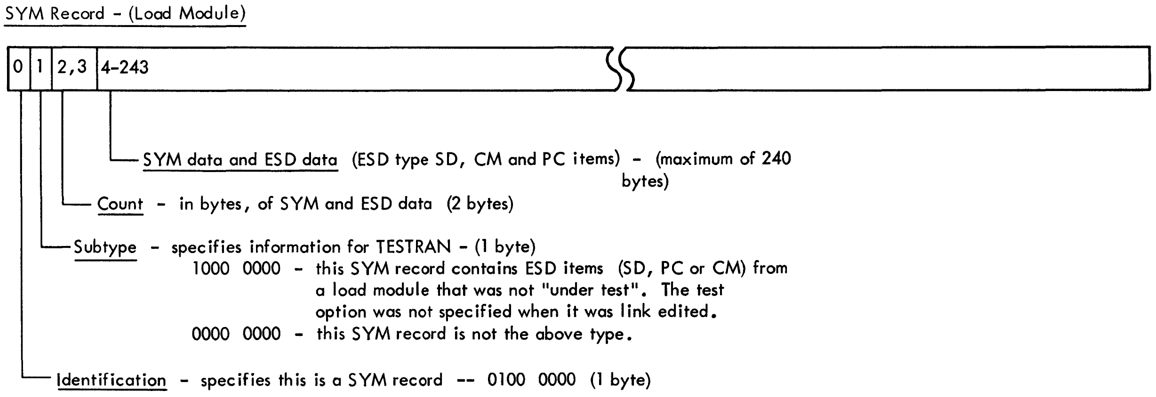
Symbolic entry point name (optional)

Blank

END

12-9-2 (0000 0010)

Figure 69. END Input Record - Type 2 (Card Image)



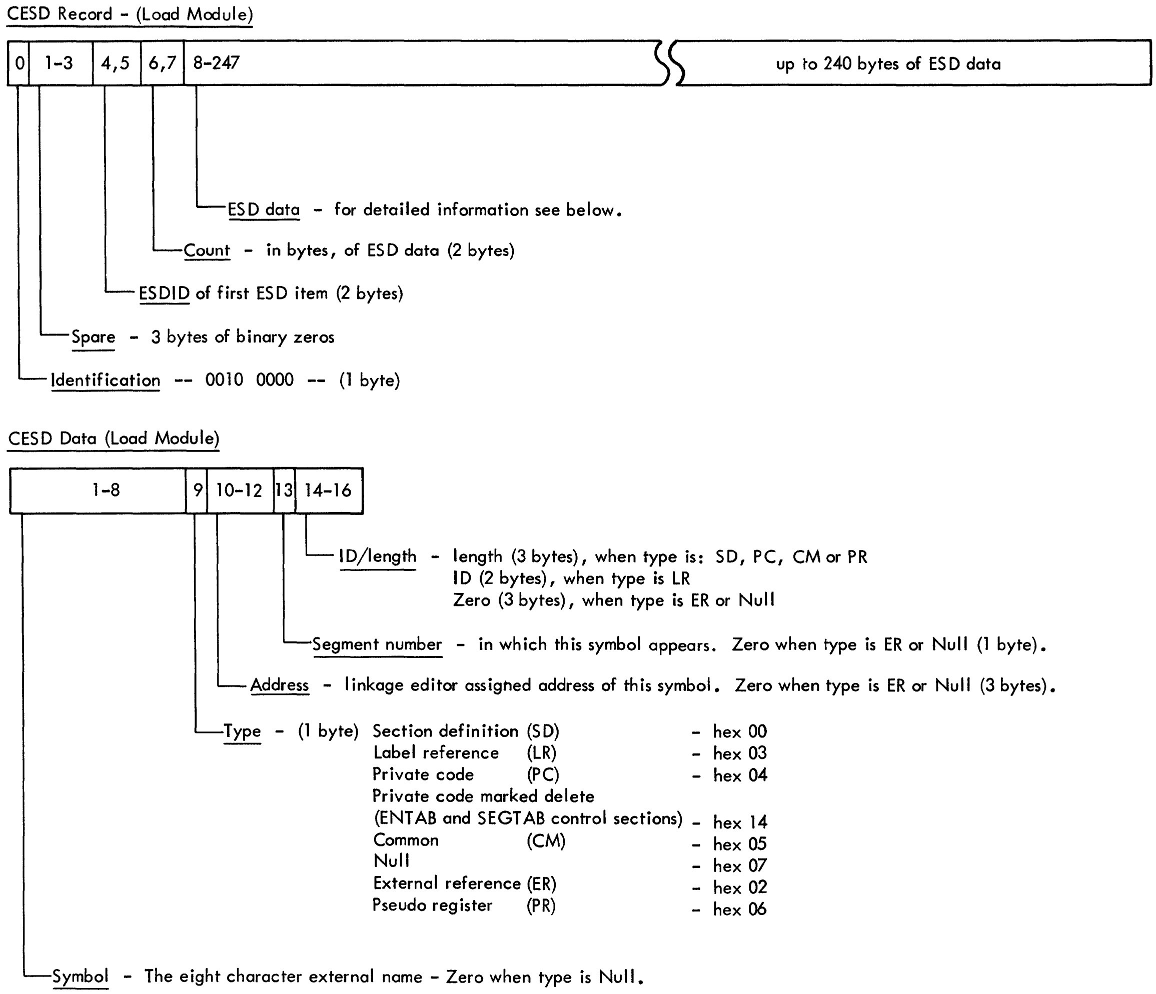
SYM data and ESD data (ESD type SD, CM and PC items) - (maximum of 240 bytes)

Count - in bytes, of SYM and ESD data (2 bytes)

Subtype - specifies information for TESTRAN - (1 byte)
1000 0000 - this SYM record contains ESD items (SD, PC or CM) from a load module that was not "under test". The test option was not specified when it was link edited.
0000 0000 - this SYM record is not the above type.

Identification - specifies this is a SYM record -- 0100 0000 (1 byte)

Figure 70. SYM Record - (Load Module)



CESD Record - (Load Module)

0	1-3	4,5	6,7	8-247	up to 240 bytes of ESD data

ESD data - for detailed information see below.

Count - in bytes, of ESD data (2 bytes)

ESDID of first ESD item (2 bytes)

Spare - 3 bytes of binary zeros

Identification -- 0010 0000 -- (1 byte)

CESD Data (Load Module)

1-8	9	10-12	13	14-16

ID/length - length (3 bytes), when type is: SD, PC, CM or PR
ID (2 bytes), when type is LR
Zero (3 bytes), when type is ER or Null

Segment number - in which this symbol appears. Zero when type is ER or Null (1 byte).

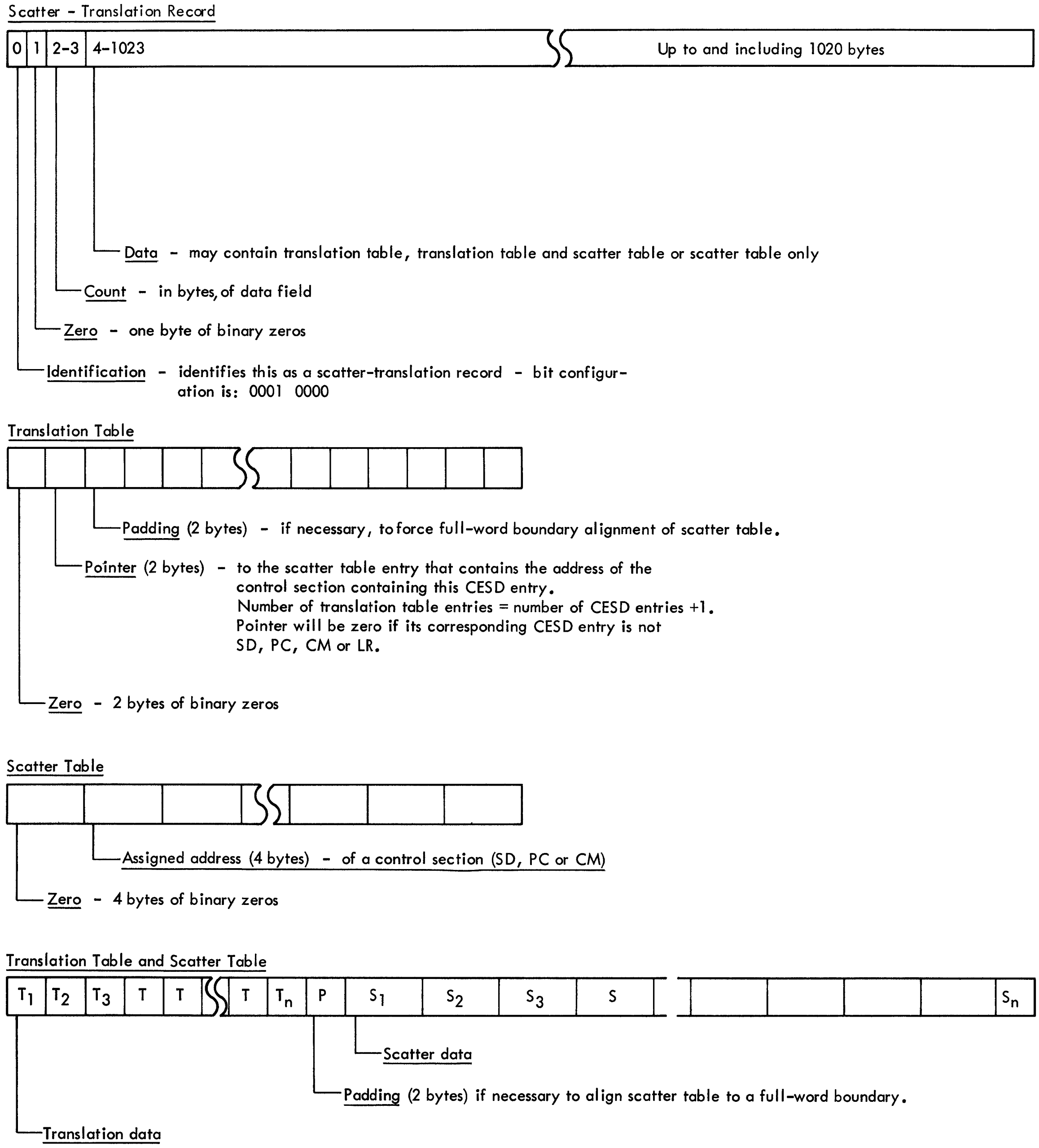
Address - linkage editor assigned address of this symbol. Zero when type is ER or Null (3 bytes).

Type - (1 byte)

Section definition	(SD)	- hex 00
Label reference	(LR)	- hex 03
Private code	(PC)	- hex 04
Private code marked delete (ENTAB and SEGTAB control sections)		- hex 14
Common	(CM)	- hex 05
Null		- hex 07
External reference	(ER)	- hex 02
Pseudo register	(PR)	- hex 06

Symbol - The eight character external name - Zero when type is Null.

Figure 71. CESD Record - (Load Module)



Scatter - Translation Record

Data - may contain translation table, translation table and scatter table or scatter table only

Count - in bytes, of data field

Zero - one byte of binary zeros

Identification - identifies this as a scatter-translation record - bit configuration is: 0001 0000

Translation Table

Padding (2 bytes) - if necessary, to force full-word boundary alignment of scatter table.

Pointer (2 bytes) - to the scatter table entry that contains the address of the control section containing this CESD entry. Number of translation table entries = number of CESD entries +1. Pointer will be zero if its corresponding CESD entry is not SD, PC, CM or LR.

Zero - 2 bytes of binary zeros

Scatter Table

Assigned address (4 bytes) - of a control section (SD, PC or CM)

Zero - 4 bytes of binary zeros

Translation Table and Scatter Table

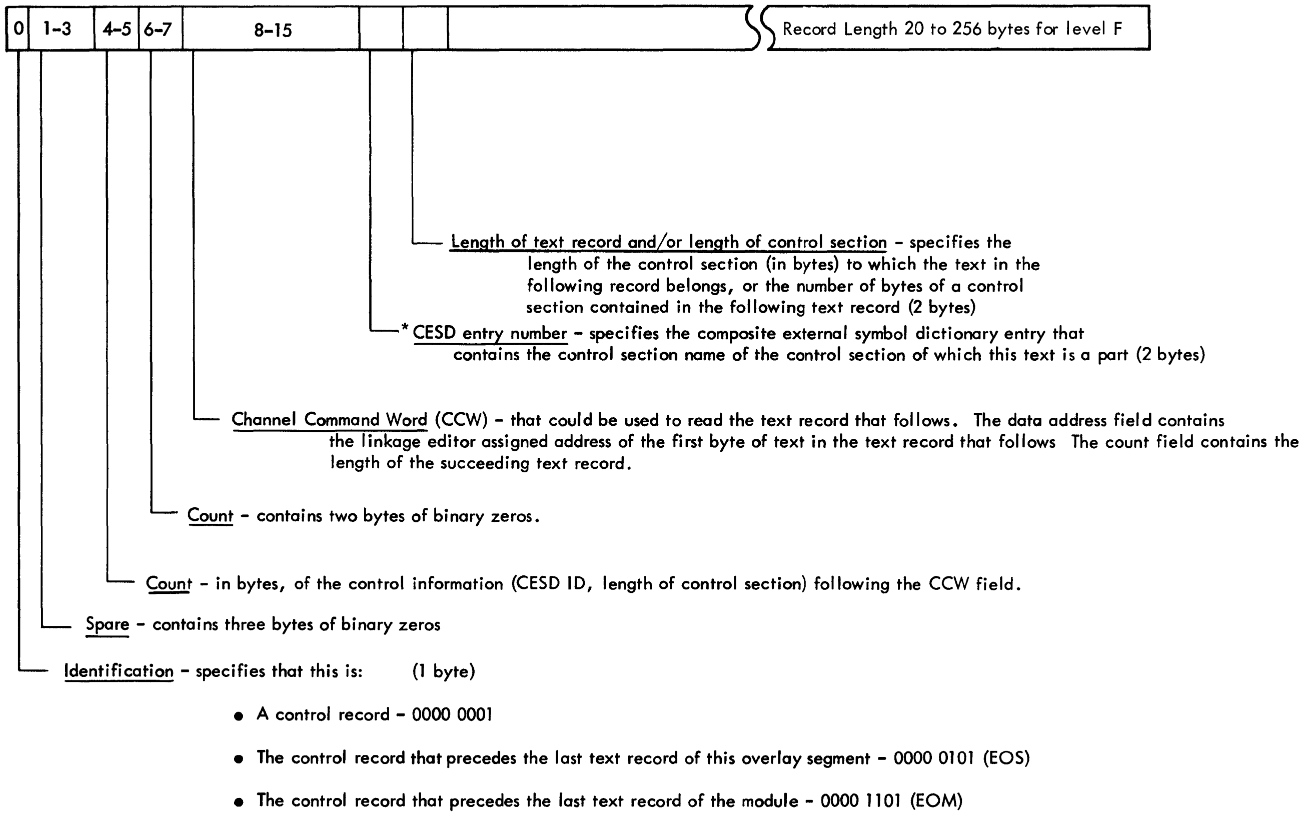
Scatter data

Padding (2 bytes) if necessary to align scatter table to a full-word boundary.

Translation data

Figure 72. Scatter-Translation Record

Control Record - (Load Module)



Length of text record and/or length of control section - specifies the length of the control section (in bytes) to which the text in the following record belongs, or the number of bytes of a control section contained in the following text record (2 bytes)

* CESD entry number - specifies the composite external symbol dictionary entry that contains the control section name of the control section of which this text is a part (2 bytes)

Channel Command Word (CCW) - that could be used to read the text record that follows. The data address field contains the linkage editor assigned address of the first byte of text in the text record that follows The count field contains the length of the succeeding text record.

Count - contains two bytes of binary zeros.

Count - in bytes, of the control information (CESD ID, length of control section) following the CCW field.

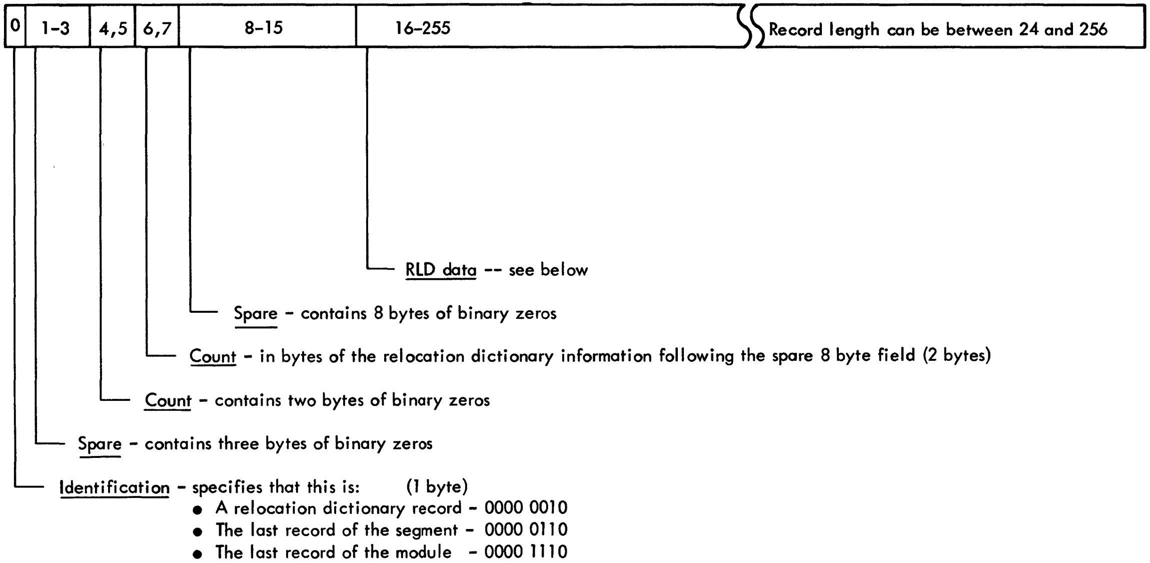
Spare - contains three bytes of binary zeros

Identification - specifies that this is: (1 byte)

- A control record - 0000 0001
- The control record that precedes the last text record of this overlay segment - 0000 0101 (EOS)
- The control record that precedes the last text record of the module - 0000 1101 (EOM)

Figure 73. Control Record - (Load Module)

Relocation Dictionally Record - (Load Module)



0	1-3	4,5	6,7	8-15	16-255	Record length can be between 24 and 256

RLD data -- see below

Spare - contains 8 bytes of binary zeros

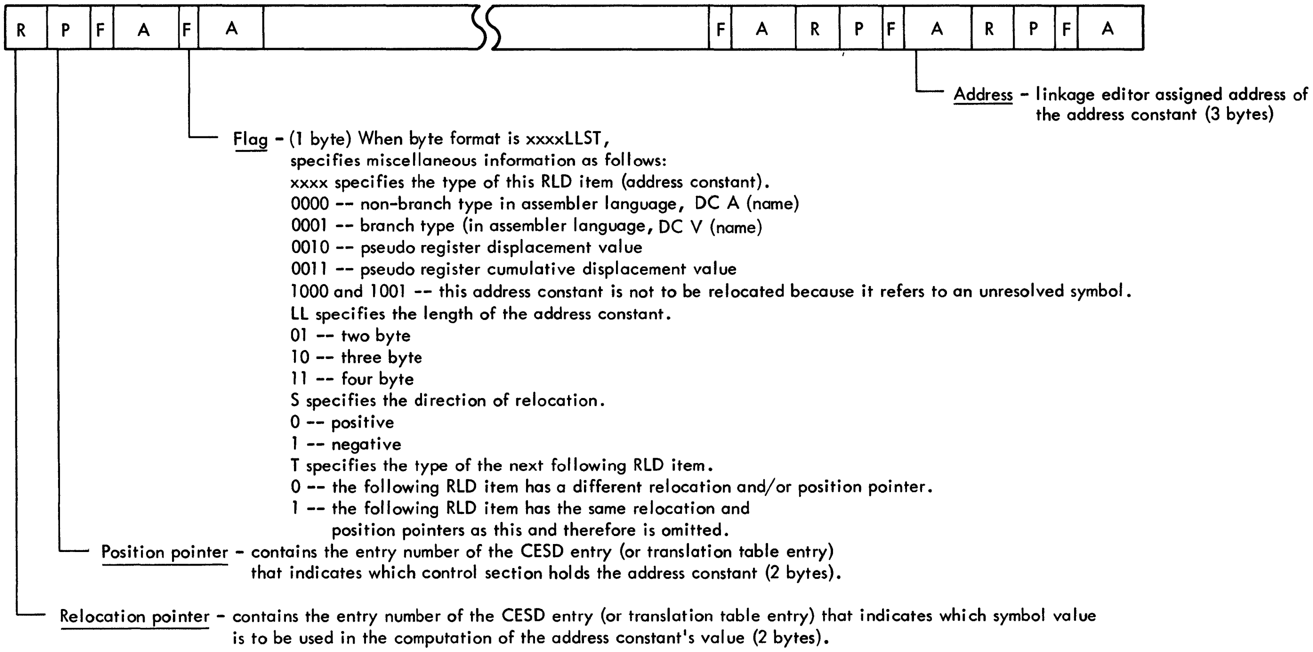
Count - in bytes of the relocation dictionary information following the spare 8 byte field (2 bytes)

Count - contains two bytes of binary zeros

Spare - contains three bytes of binary zeros

Identification - specifies that this is: (1 byte)
- A relocation dictionary record - 0000 0010
- The last record of the segment - 0000 0110
- The last record of the module - 0000 1110

RLD Data



R	P	F	A	F	A			F	A	R	P	F	A	R	P	F	A

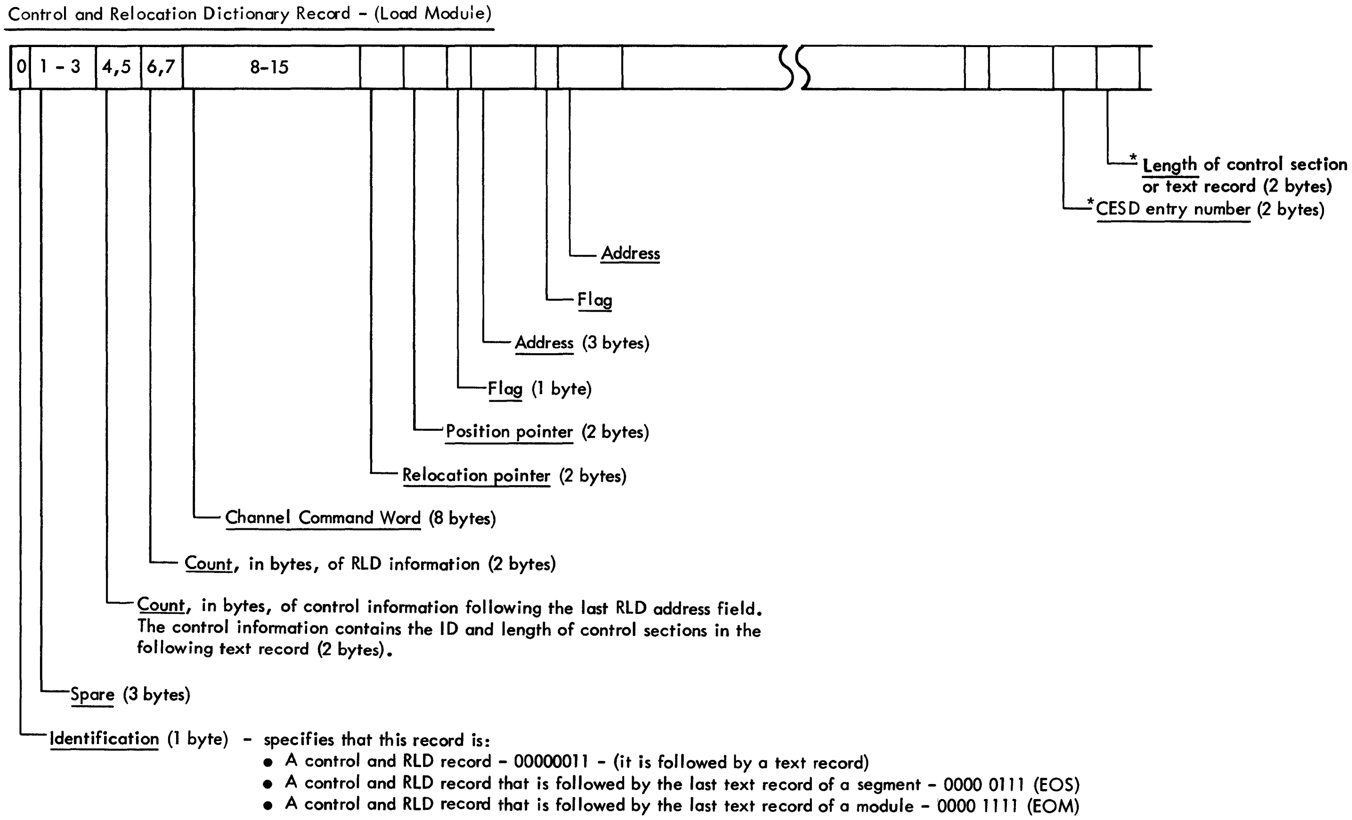
Address - linkage editor assigned address of the address constant (3 bytes)

Flag - (1 byte) When byte format is xxxxLLST,
specifies miscellaneous information as follows:
xxxx specifies the type of this RLD item (address constant).
0000 -- non-branch type in assembler language, DC A (name)
0001 -- branch type (in assembler language, DC V (name)
0010 -- pseudo register displacement value
0011 -- pseudo register cumulative displacement value
1000 and 1001 -- this address constant is not to be relocated because it refers to an unresolved symbol.
LL specifies the length of the address constant.
01 -- two byte
10 -- three byte
11 -- four byte
S specifies the direction of relocation.
0 -- positive
1 -- negative
T specifies the type of the next following RLD item.
0 -- the following RLD item has a different relocation and/or position pointer.
1 -- the following RLD item has the same relocation and position pointers as this and therefore is omitted.

Position pointer - contains the entry number of the CESD entry (or translation table entry) that indicates which control section holds the address constant (2 bytes).

Relocation pointer - contains the entry number of the CESD entry (or translation table entry) that indicates which symbol value is to be used in the computation of the address constant's value (2 bytes).

Figure 74. Relocation Dictionary Record - (Load Module)


Control and Relocation Dictionary Record - (Load Module)
0
1 - 3
4,5
6,7
8-15
* Length of control section or text record (2 bytes)
*CESD entry number (2 bytes)
Address
Flag
Address (3 bytes)
Flag (1 byte)
Position pointer (2 bytes)
Relocation pointer (2 bytes)
Channel Command Word (8 bytes)
Count, in bytes, of RLD information (2 bytes)
Spare (3 bytes)
Identification (1 byte)

Count, in bytes, of control information following the last RLD address field. The control information contains the ID and length of control sections in the following text record (2 bytes).

Identification (1 byte) - specifies that this record is:

- A control and RLD record - 00000011 - (it is followed by a text record)
- A control and RLD record that is followed by the last text record of a segment - 0000 0111 (EOS)
- A control and RLD record that is followed by the last text record of a module - 0000 1111 (EOM)

Note: For detailed descriptions of the data fields see Relocation Dictionary Record, and Control Record.

The record length varies from 20 to 260 bytes in the level E linkage editor

Figure 75. Control and Relocation Dictionary Record - (Load Module)

INDEX

READER'S COMMENT FORM

IBM System/360 Operating System
Linkage Editor F
Program Logic Manual

Form Y28-6667-0

- Is the material:

	Yes	No
Easy to read?	□	□
Well organized?	□	□
Complete?	□	□
Well illustrated?	□	□
Accurate?	□	□
Suitable for its intended audience?	□	□

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

- Please check the items that describe your position:
  - □ Customer personnel
  - □ IBM personnel
  - □ Manager
  - □ Systems Analyst
  - □ Operator
  - □ Programmer
  - □ Customer Engineer
  - □ Instructor
  - □ Sales Representative
  - □ Systems Engineer
  - □ Trainee
  - Other

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

Explanation:

- Thank your for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

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

POSTAGE WILL BE PAID BY

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM®

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

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

Printed in U.S.A. Y28-6667-0

IBM®

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

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