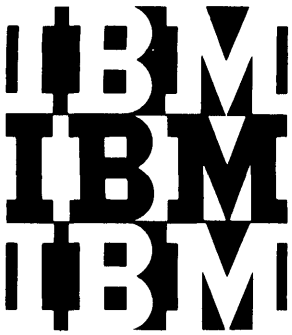


File No. S360-31
Form Y28-6610-2

Program Logic

IBM System/360 Operating System

Linkage Editor (E)

Program Logic Manual

Program Number 360S-ED-510

This publication describes the internal logic of the 15K and 18K versions of the level E linkage editor. The linkage editor combines and edits modules to produce a single load module that can be loaded into main storage by the control program. The linkage editor operates as a processing program rather than as a part of the control program.

This program logic manual is directed to the IBM customer engineer who is responsible for program maintenance. It can be used to locate specific areas of the program, and it enables the reader to relate these areas to the corresponding program listings. Because program logic information is not necessary for program operation and use, distribution of this manual is restricted to persons with program-maintenance responsibilities.

Form Y28-6610-2
Page revised 7/23/69 by TNL Y28-6400

PREFACE

This publication provides customer engineers and other technical personnel with information describing the internal organization and logic of the level E linkage editor. 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, Y28-6605

IBM System/360 Operating System: Concepts and Facilities, C28-6535

IBM System/360 Operating System: Assembler Language, C28-6514

The reader should also refer to the co-requisite publication: IBM System/360 Operating System: Linkage Editor and Loader, C28-6538

This manual consists of three parts:

1. An Introduction, describing 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 section describing each major division of the 15K and 18K versions of linkage editor E. Each major division is discussed in sufficient detail to enable the reader to understand its basic functions, and to provide a frame of reference for the comments and coding supplied in the program listing. Common data, such as tables, control blocks, and work areas, are discussed only to the extent required to understand the logic of the major divisions. Flowcharts are included at the end of this section.

3. An Appendix, containing:

   a. The input conventions and record formats for the linkage editor.

   b. The layouts of tables, which may not be essential for an understanding of the basic logic of the program, but are essential for analysis of storage dumps.

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

Third Edition (June 1967)

The specifications contained in this publication as amended by Technical Newsletter Y28-6400, dated July 23, 1969, correspond to Release 18 of IBM System/360 Operating System.

Changes are periodically made to the specifications herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

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

Comments may be addressed to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020.

CONTENTS

FIGURES

TABLES

CHARTS

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 E linkage editor is available in 15K and 18K versions; they differ in speed, table sizes, and overlay structure. 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 Call Processing. 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 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). A control dictionary entry is generated whenever an external symbol, an address constant, or the beginning of a control section is processed by a language translator.

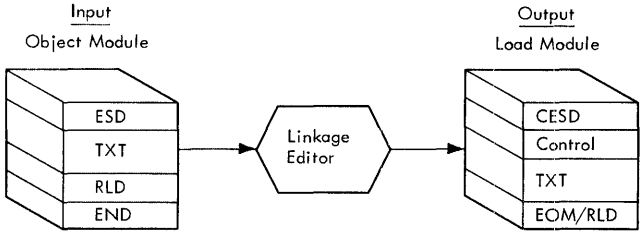


Figure 1. Linkage Editor Processing - Simple Case

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,

---

[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.

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 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. X refers to the ESD item for the control section in which it resides (CSECTA); therefore, both pointers of its associated RLD item refer to the ESD entry for the control section (line 1). Y refers to an external reference symbol (CSECTC); 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.

Note: Figure 2 is intended to show only the relationship between ESD, text, and RLD items before and after linkage editor processing; the output module structure shown applies only to the level E linkage editor.

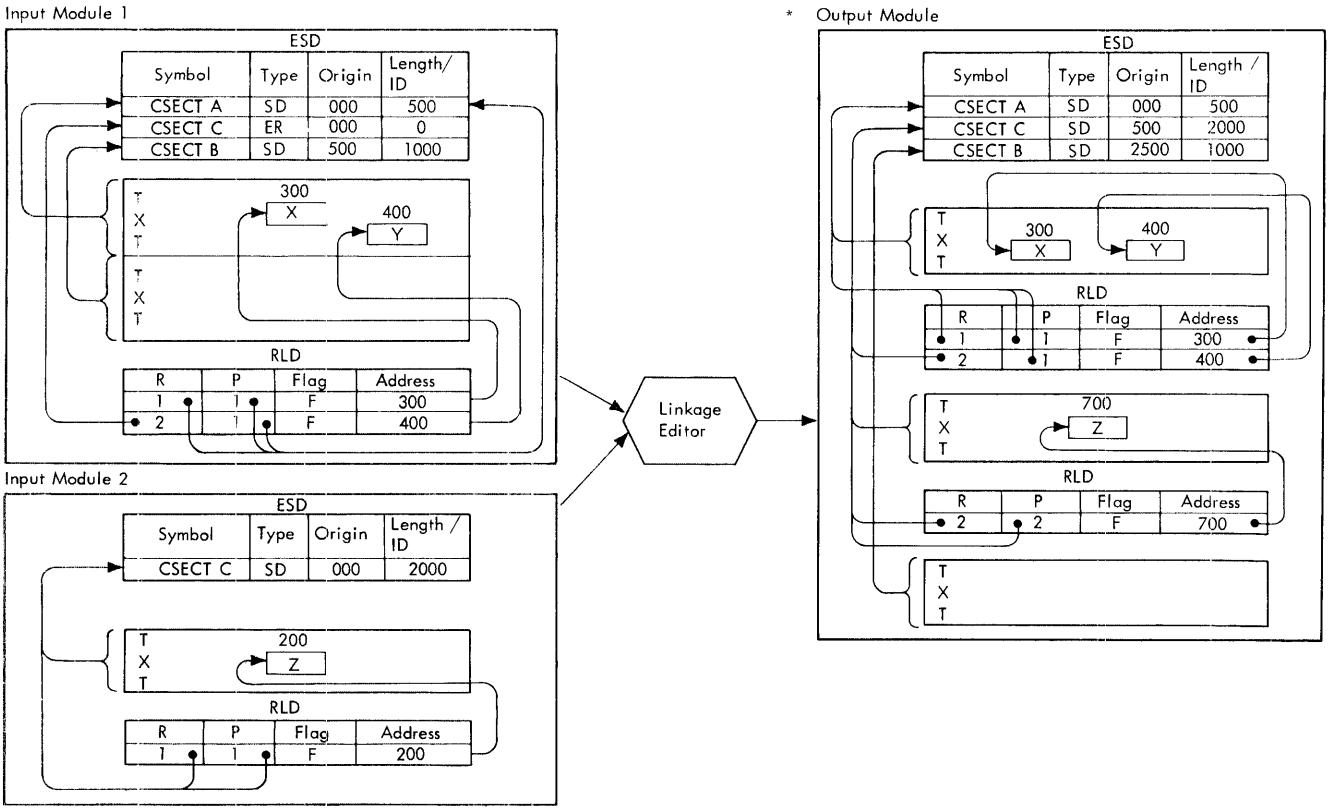
Options

Module structure 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 have been specified on the execute (EXEC) statement and overlay con-

---

[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 Processor.") Such ESD items are excluded from the numbering system.



• Figure 2. Combining Control Dictionaries

trol statements have been 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 TEST 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.[1] The control portion may contain:

  1. An end of segment (EOS) indication, if the following text record is the last text record of an overlay segment.[2]

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

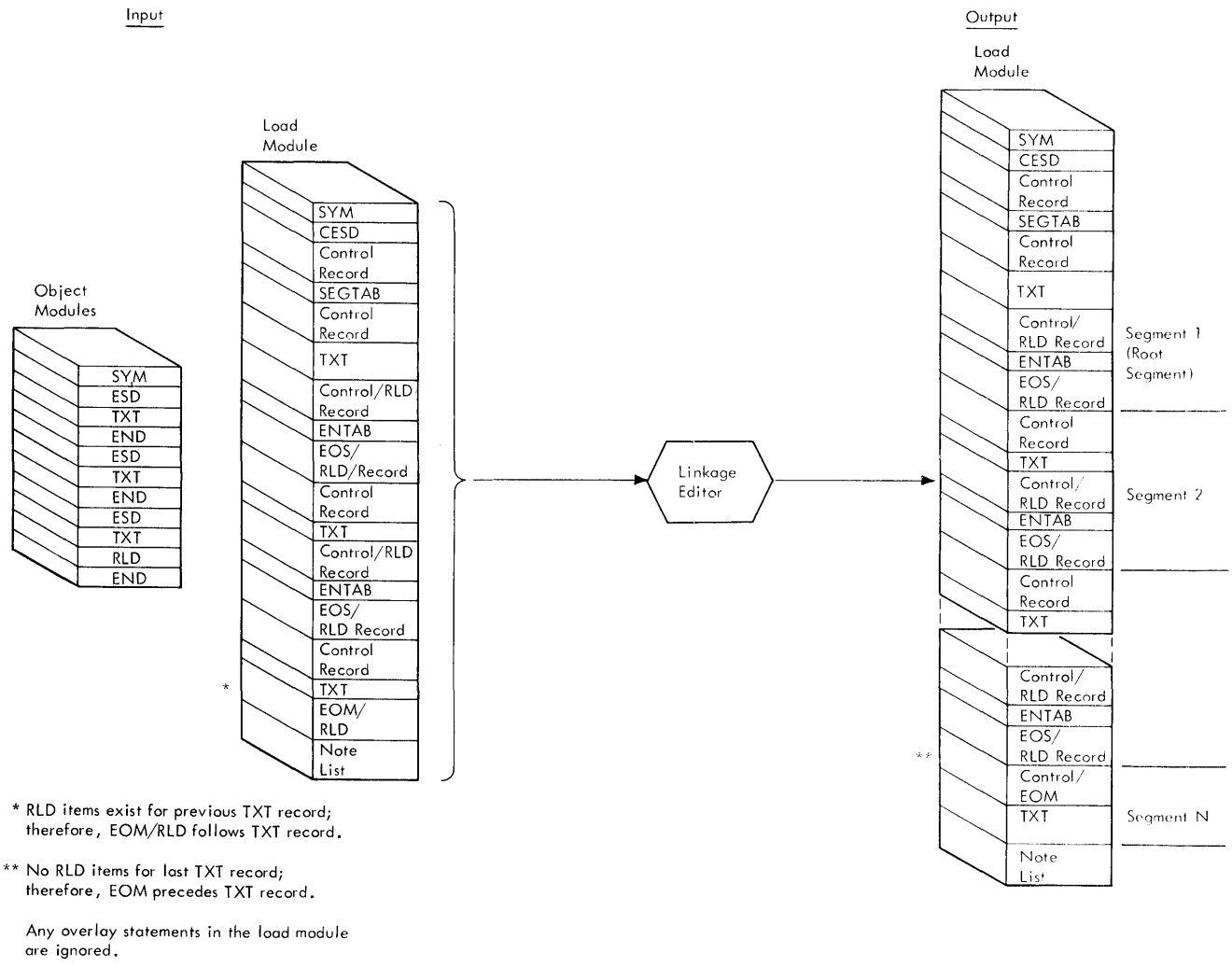
  3. The number of bytes of RLD information that follow, if it is a composite control/RLD record.

  4. The number of bytes of control information.

---

[1]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.

[2]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

The control portion also contains the IDs of the control sections in the following text record, the number of bytes of text for each ID, 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.

- Text for each control section. Text records contain the code 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 in 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

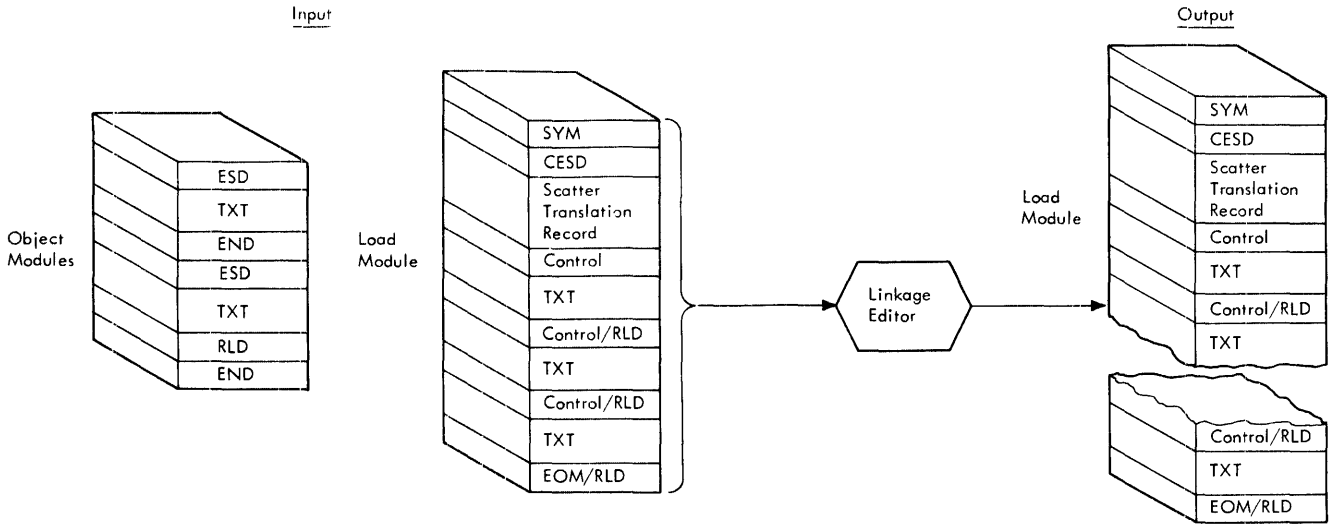


Figure 4. Linkage Editor Processing - Using Scatter Load and Test Options

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 3) 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.

- 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, refreshable, hierarchy, and scatter attributes cannot be present.
- Hierarchy format. When a HIARCHY statement is detected, the "number" and "name" operand values are used in building the scatter table and translation table. The high-order byte of each CSECT address entry contains the hierarchy number that is included in the GETMAIN request for main storage for program loading.

- 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.

- *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.

- 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 the results of processing. (For details on recovery management, refer to the publication: IBM System/360 Operating System: Concepts and Facilities, Form C28-6535.)

MAIN STORAGE HIERARCHY SUPPORT

If Main Storage Hierarchy Support for IBM 2361 Models 1 and 2 is included in the system, the linkage editor produces load modules which can be loaded into either processor storage or IBM 2361 Core Storage by the control program. If the HIAR parameter is specified in the PARM field of the EXEC statement, the linkage editor is initialized to accept the HIARCHY control statement. This statement specifies the storage hierarchy (0 for processor storage and 1 for IBM 2361 Core Storage) into which the CSECTs identified in the statement are to be loaded.

MAJOR DIVISIONS OF LINKAGE EDITOR

Linkage editor processing consists of five sequential operations:

1. Initial processing.
2. Input processing.
3. Intermediate processing.
4. Second pass processing.
5. Final processing.

INITIAL PROCESSING

Initial processing begins when the control program passes control to the linkage editor. During this operation, the linkage editor prepares for all subsequent operations. The initial processor:

- Uses data management facilities to open data sets to be used during linkage editor processing.
- Interprets the options and attributes specified by the programmer and saves them in an all purpose table (APT).
- Uses task management facilities to obtain main storage space for internal tables, work areas, and input/output buffer areas used in linkage editor processing.

After initial processing, control is passed to the input processor.

INPUT PROCESSING (FIRST PASS)

All input to the linkage editor is processed during the first pass. Input records are read, checked for validity, identified, and processed as required. The text and RLD items that are to be part of the output load module are written on the intermediate data set (SYSUT1). Linkage editor control statements are interpreted and processed and the CESD is built in main storage. SYM records in the input are gathered and written out directly on the output device as part of the output load module. After all input has been received and processed, control is passed to intermediate processing.

INTERMEDIATE PROCESSING

Intermediate processing consists basically of two operations: address assignment and intermediate output processing. Relative machine addresses are assigned to all external symbols that are to be contained in the output load module, to the module entry point, and also to any alternative entry points defined by the user with ALIAS statements. The intermediate output processor places the CESD and, if required, the SEGTAB or scatter translation table in the output module library.

SECOND PASS PROCESSING

During second pass processing, the text and RLD items are read from the intermediate data set, address constants in the text are relocated, and the records that make up the output module are written on the output module library (SYSLMOD).

FINAL PROCESSING

Final processing completes the library directory entry for the output load module and places it on the output module library. If the module is structured for overlay, the final processor writes out on SYSLMOD a note list that indicates the location of each segment in the output module library. If any coded diagnostic messages were written out on SYSPRINT during linkage editor processing, a directory explaining these coded messages is written. If specified, a module map or cross-reference table is produced. If a multiple execution of the linkage editor is specified, control returns to initial processing; otherwise, control is returned to the caller.

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. 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.[1]
- 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.
- 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

---

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

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.

If the TEST option has been selected, SYM records are written during input processing; text and RLD items are written sequentially on SYSUT1. 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.

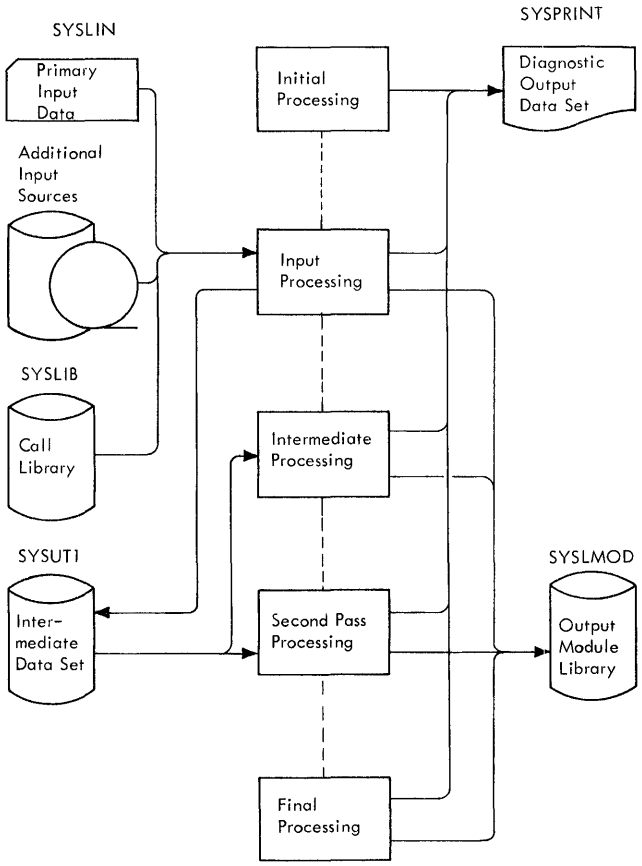


Figure 5. Input/Output Flow

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 lists for the text and RLD items on SYSUT1 are read into main storage.

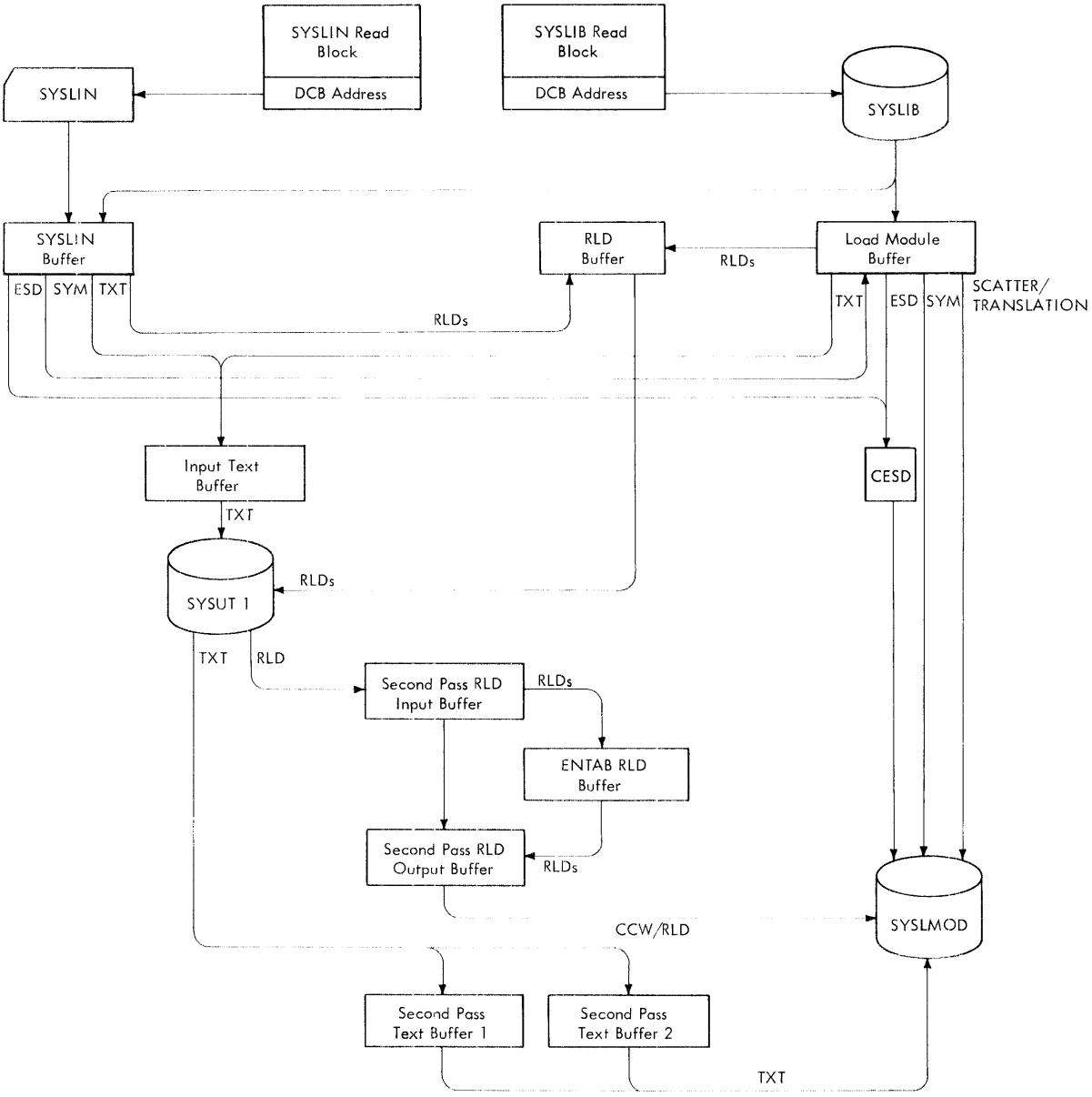
During second pass processing, text and RLD records are read into main storage from SYSUT1 in the order of assigned addresses within each segment (using the note lists to find the records) and are written out on SYSLMOD.

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 module map or cross-reference table was requested, SYSLMOD is closed, and then reopened as an input source. The CESD is read into main storage from SYSLMOD to be used in producing the module map. If a cross-reference table was requested, the RLD items are also read from SYSLMOD; at the end of final processing, SYSLMOD is closed. 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, control is returned to initial processing and SYSLMOD is opened for output. 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, SYSLMOD remains open for output.

INTERNAL DATA FLOW

A generalized representation of internal data flow during linkage editor processing is given in Figure 6. A pointer in the SYSLIN or SYSLIB read block indicates the input source from which data is to be read. The input data is then read in and processed in the following manner:

Input records from SYSLIN are read into the SYSLIN buffer. (SYSLIN contains only object modules.) Object modules from SYSLIB are read into the SYSLIN buffer, whereas load modules from SYSLIB are read into the load module buffer. During input processing, SYM information is gathered in the load module buffer, text is gathered in the input text buffer, RLD records are processed in the input RLD buffer, and ESD records are combined into the composite ESD. Text and RLDs are written out on SYSUT1, while SYM records are written directly on SYSLMOD. CESD, SEGTAB, and scatter/translation records are written out on SYSLMOD during intermediate processing. During second pass processing, any RLD items that were placed on SYSUT1 are read back into the second pass RLD buffer; any text that was stored on SYSUT1 is read back into the second pass text buffer. (Two second pass text buffers are used for input/output overlap.) For overlay modules, ENTAB RLD items are produced in the ENTAB RLD buffer. (The ENTAB itself is built in the second pass text buffer.) After address constants in the text have been relocated, text, RLD, and ENTAB records are written out on SYSLMOD.



• Figure 6. Internal Data Flow

The following text and the associated flowcharts at the end of this section describe the major divisions of the 15K and 18K versions of linkage editor E. Each major division is further subdivided and described to explain the general organization and operation of linkage editor.

The major divisions of linkage editor E are shown in chart AA.

- Initial processor.
- Input processor.
- Address assignment processor.
- Intermediate output processor.
- Second pass processor.
- Final processor.

The overall organization of linkage editor E is shown in Figure 7.

INITIAL PROCESSOR

The initial processor builds an all purpose table (APT), 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.

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 initial processor interprets the control information, checks it for validity, and saves it for later use in linkage editor processing.

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

The 15K and 18K level E initial processor (IEWLEINT) (Chart BA) operates in the following manner:

- After the standard ddnames (or passed ddnames) have been entered into the data control blocks of the data sets used by the linkage editor, the initial processor opens all data sets except SYSLIB and SYSLMOD using data management facilities. (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.)

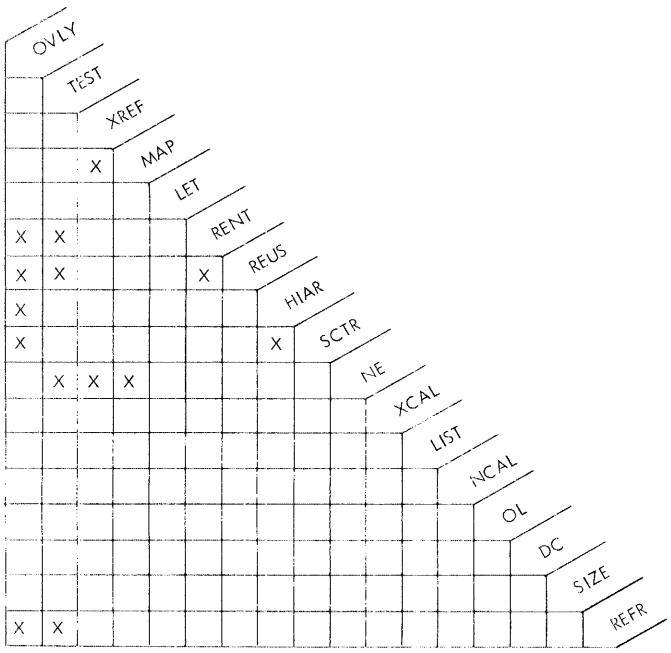
- The initial processor sets an "unlike attributes" indicator in the SYSLIN DCB. This indicates to the open routine that SYSLIN may be a concatenation of data sets stored on different devices.

- The attribute and option routine scans and analyzes 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).

---

[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 and Loader.

Table 1. Combinations of Module Attributes and Program Options



Note: An X indicates incompatible attributes: the attribute that appears lower on 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.

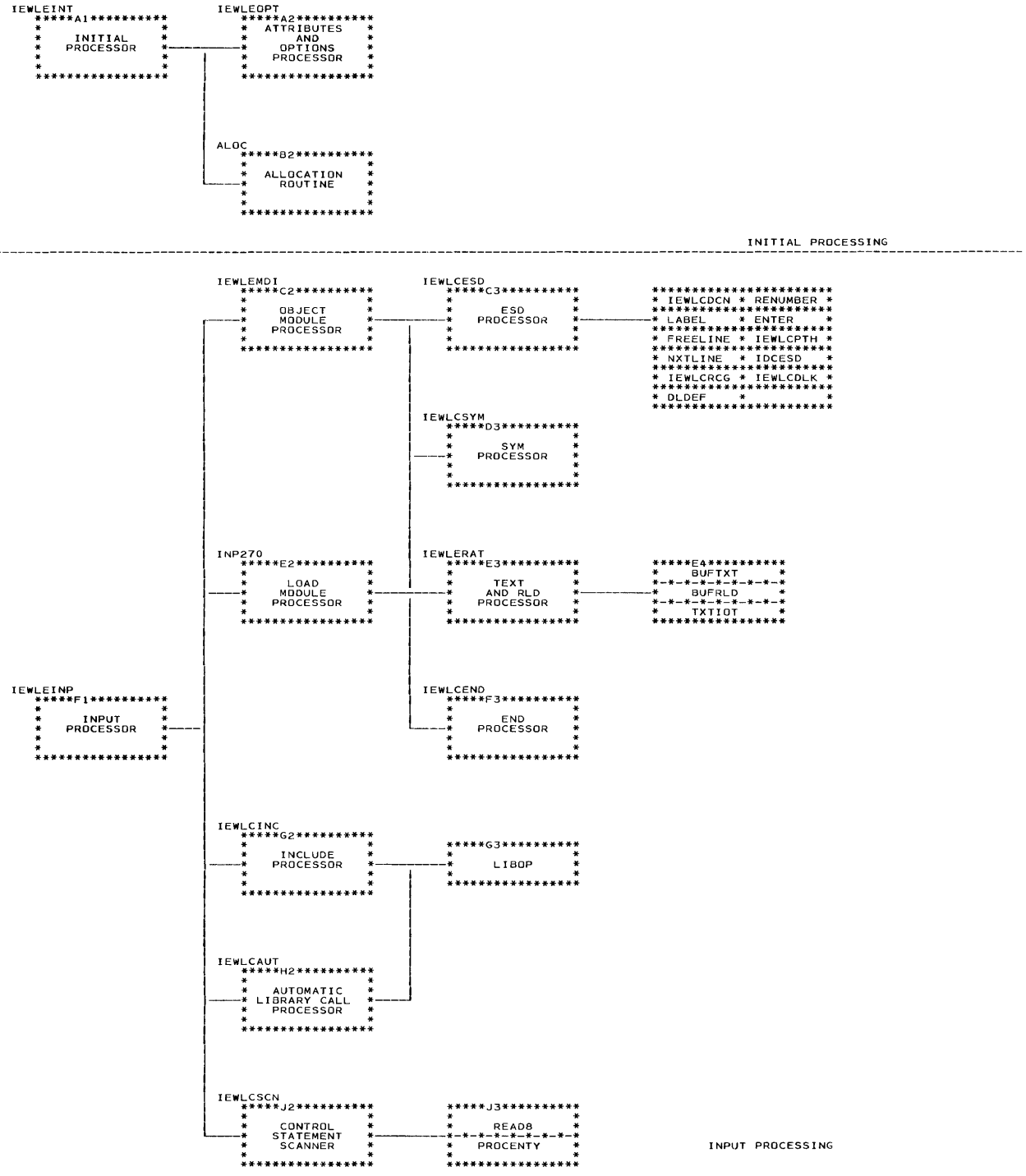


Figure 7. Level E Linkage Editor Organization

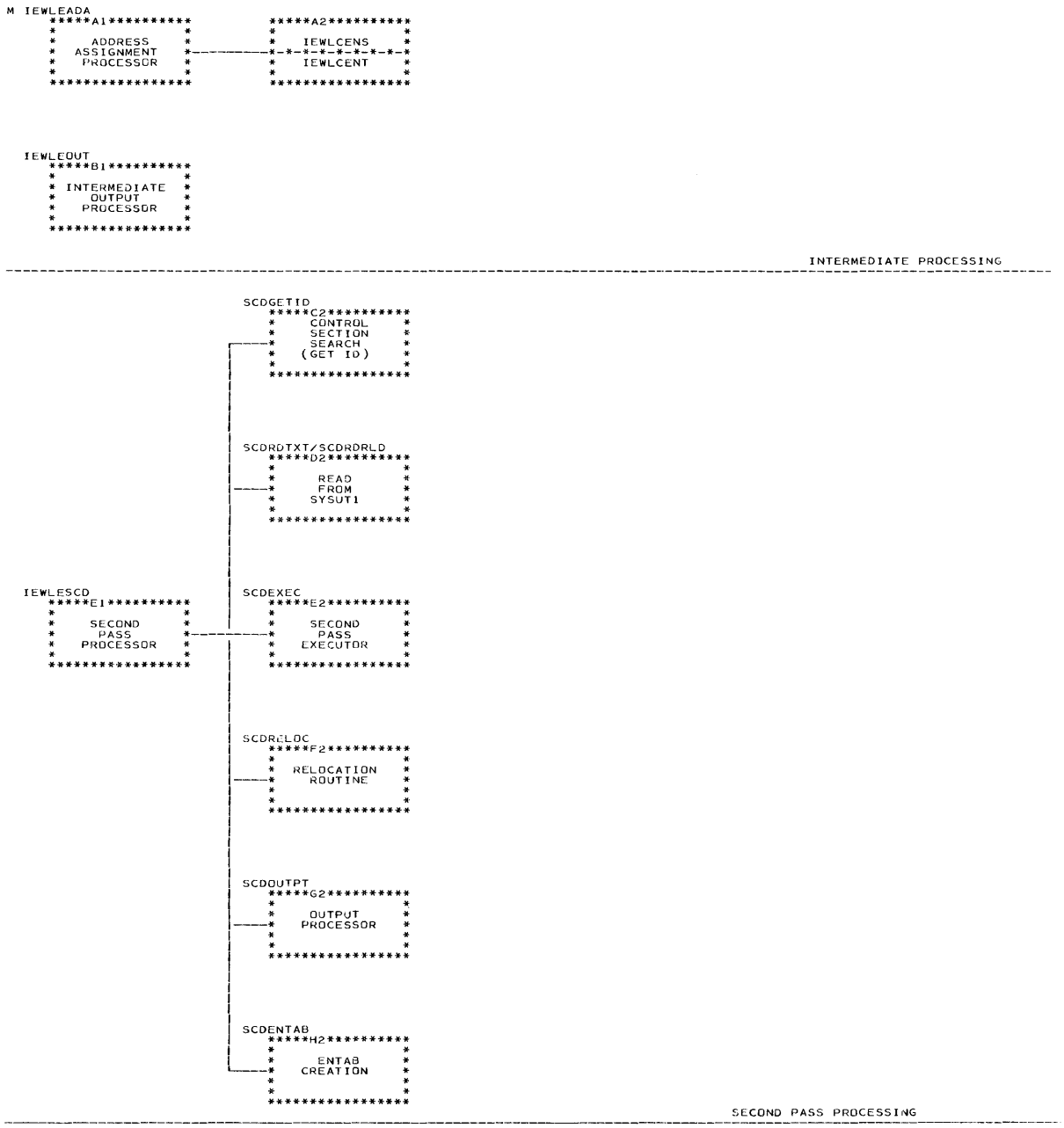
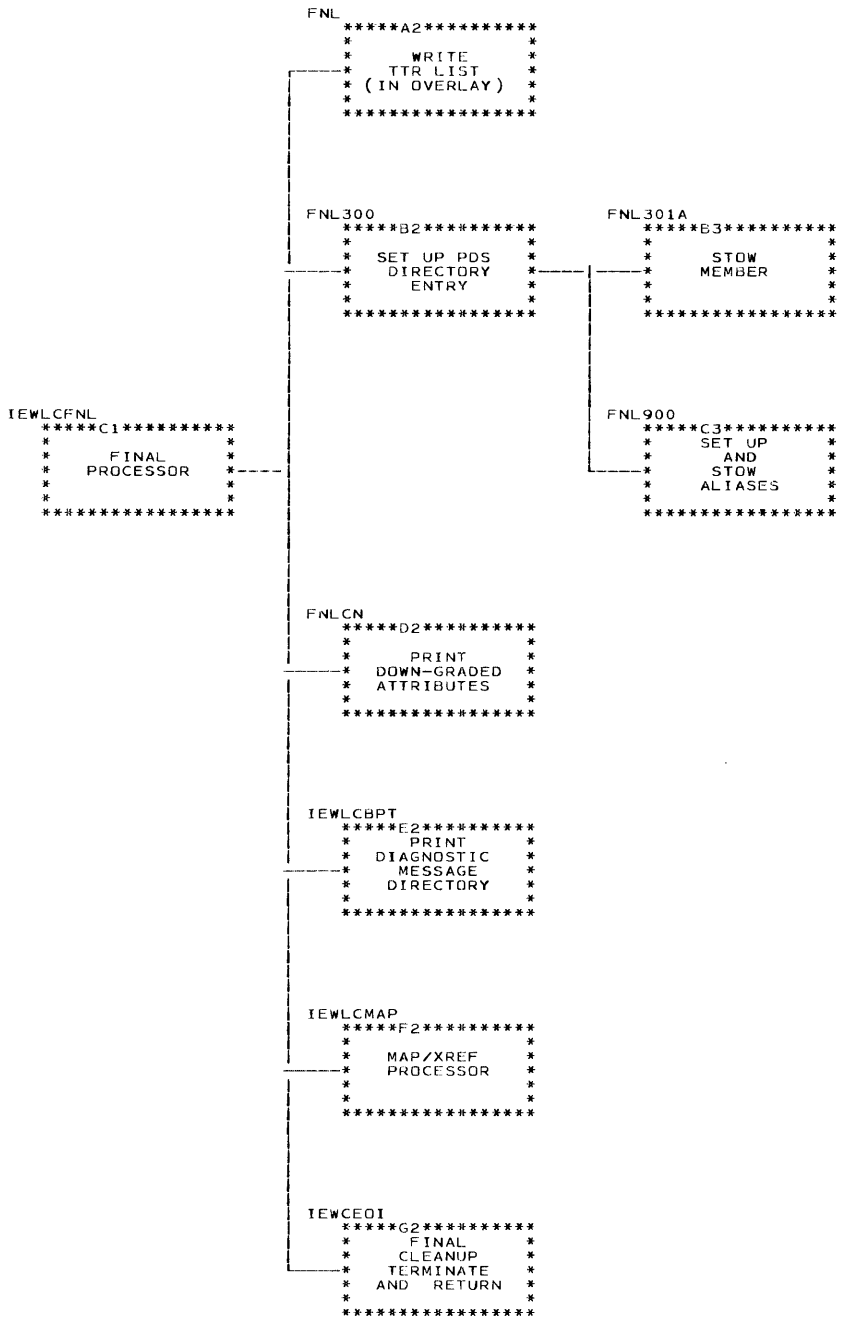


Figure 7. Level E Linkage Editor Organizaticn (Continued)



FINAL PROCESSING

Figure 7. Level E Linkage Editor Organization (Continued)

- SYSLMOD is opened, and the allocation processor requests 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 linkage editor.

The following paragraphs describe the allocation process in the level E version of the linkage editor.

MAIN STORAGE ALLOCATION - 15K AND 18K LEVEL E

To obtain the required main storage space, the allocation processor (ALOC):

1. Determines the excess of main storage space allocated by the supervisor.
2. Divides the total excess by the total weight factor. 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.
3. Multiplies the quotient obtained in step 2 (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 used by the current module.
4. Divides the total byte count for each table by the number of bytes per entry, and saves the result in the all purpose table.
5. Computes the addresses for the tables.
6. Releases excess main storage space, noting the last address used.

When the required main storage space has been allocated, tables are initialized, and control is passed to the input processor.

INPUT PROCESSOR

After initial processing, control is passed to the input processor. The input processor performs a control function; the operations performed depend on the nature of the input. The input type and input conditions are analyzed, and control is passed to the appropriate processing routine. At the end of input, control passes to the intermediate processor.

The 15K and 18K level E input processor (IEWLEINP) is shown in Chart CA; it operates in the following manner:

- Each input record is read, using one of two read blocks. The first read block contains the address of the SYSLIN module buffer, the address of the SYSLIN DCB, and the block size and logical record length. The second read 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 block is to be used for the input record. Initially, the pointer is set to the SYSLIN read block. If input is to be read from a library, the include processor (Chart CO) or automatic library call processor (Chart CP) may move the pointer to the library read block at any end-of-data condition. The reading of input is therefore not restricted to a particular DCB and buffer.
- If SYSLIN is a concatenation of data sets, the current READ is reissued when a data set boundary is crossed.
- Control is given to the control statement scanner (Charts CL and CM) for all object module records whose first column character is a blank, provided that the record is not encountered "in module." (Control statements encountered within a module cause an error indication.)
- Control is given to either the object module processor (Chart CB) or load module processor (Chart CC), depending on the input module type. (All input via include or automatic library call is identified by record format. F format indicates object modules; U format indicates load modules. Only object modules are read from SYSLIN.)

- At any end-of-input (from SYSLIN or SYSLIB), the input processor determines if control should be given to the include processor or to the automatic library call processor. The include processor is given control if more modules must be included before resuming normal processing. The automatic library call processor receives control if the NCAL option (no automatic library calls) was not selected and an end-of-input on SYSLIN has occurred. If the NCAL option was selected, control is passed to the address assignment processor.

- If a NAME statement, which may indicate a multiple execution of linkage editor, was detected by the control statement scanner, processing proceeds as if an end-of-input had occurred on SYSLIN (the automatic library call processor receives control). However, no end-of-input indication is made so that control will be returned to the initial processor at the end of final processing.

- If an end-of-input occurs on SYSLIN, but no valid input was received, control is passed to the final processor (Chart GA) to terminate linkage editor processing.

OBJECT MODULE PROCESSOR

The level E object module processor is shown in Chart CB. Object module processing consists essentially of three operations:

1. Determination of record type.
2. Setup of general registers.
3. Special event processing.

The record type is determined by examining columns 2 through 4 of each logical input record. For each record type, control is passed to an associated processor, as follows:

Record Type	Processor	Chart
SYM	IEWLCSYM	CD
ESD	IEWLCESD	CE,CF,CG
TXT	IEWLERAT	CH,CJ
RLD	IEWLERAT	CH,CI
END	IEWLCEND	CK

The general registers are loaded with input record information to be used by the selected processor, as described in Table 2.

Table 2. General Register Information - Object Module Processing

Input Record Type (See Appendix A 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	

Following is a description of special event processing:

- When an END statement is detected, the RLD and TXT processor is entered so that any data still contained in the input RLD buffer or the input text buffer can be written out on SYSUT1.

- If the TEST option is selected, the SYM records from the object module are gathered by the SYM processor in the load module 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 SYM processor is entered so that the contents of the load module buffer can be written out on SYSLMOD (see Chart CD).

- When control is returned from the ESD processor, 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 SYM processor is entered to block the ESD record with any other ESD records in the input text 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 by the error logging routine. 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 by the error logging routine. The input record is then ignored.

LOAD MODULE PROCESSOR

The level E load module processor is shown in Chart CC. Load modules included in the input to linkage editor by the include processor or the automatic library call processor are processed in the following manner:

- The input record type is determined by an identification field (byte 1 of the record), and control is passed to an associated processor, as shown in Table 3.

- The parameter registers are loaded with input record information to be used by the selected processor, as described in Table 4.

- 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, the END processor performs cleanup functions and control returns to the input processor.

- When a CCW record is detected, the following TXT record is immediately read into the input text buffer before the TXT and RLD processor is entered.

- If the test option was specified on the EXEC statement and a SYM record is received, control is passed to the SYM processor to write out the record as test translation data from the load module buffer (see Chart CD).

Table 3. Record Types and Associated Processors

Record Type	Identifier	Processor	Chart
TXT	*	IEWLERAT	CH,CJ
CESD	hex '20'	IEWCESD	CE,CF,CG
Scatter/ Translation	hex '10'	(Ignored)	
SYM	hex '40'	IEWLCSYM	CD
CCW	hex '01'	IEWLERAT	CH,CI
CCW/RLD	hex '03'	IEWLERAT	CH,CI
RLD	hex '02'	IEWLERAT	CH,CI
If end of module indication is on:			
CCW	hex 'OD'	IEWLCEND	CK
CCW/RLD	hex 'OF'	IEWLCEND	CK
RLD	hex 'OE'	IEWLCEND	CK

*Identified by preceding control record.

The following paragraphs describe the functions, during object and load module processing, of the ESD processor, the TXT and RLD processor, and the END processor.

Table 4. 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	Assigned address of first byte of text in following record	Level E-Byte count text in following record	ESDID of CSECT to which text belongs	
RLD		Byte count of RLD items in record		Address of first RLD item in buffer

ESD PROCESSOR

When the object or load module processor detects an ESD record, it gives control to the ESD processor (Charts CE, CF, and CG).

The main function of the ESD processor is symbol resolution. It combines the individual ESDs in the input to linkage editor into a composite ESD, which contains all symbols in the input which were not changed, deleted, or replaced. The ESD processor refers to a chained REPLACE/CHANGE list (produced by the control card scanner) to determine which ESD items are to be changed, deleted, or replaced. The ESD processor also produces a renumbering table (RNT), which is used by the TXT, RLD, and END processors to translate the ESD ID of the input ESD items to CESD IDs.

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 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 sec-

tion. 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 Section 3.

ESD Processing

Upon receiving control from the input processor, the ESD processor saves 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 load module buffer). Byte 10 must be cleared because the automatic library call processor 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 routine IEWLCRCG examines the REPLACE/CHANGE chain that was built in the CESD by the control statement scanner and makes the appropriate modifications. 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 ESD processor changes the input ESD item symbol from A to B.

- 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, the ESD processor changes it 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; refer to "LR (or LD) Items." For this reason, the ESD processor sets an internal indicator when it changes the type to LR.)

After determining the ESD type, the ESD processor scans the CESD for a matching symbol. If no match is found, non-resolution processing proceeds as shown on Chart CF. If the input ESD symbol matches a symbol in the CESD, resolution processing is performed as shown on Chart CG. 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 (CHART CF): 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:

- 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.)

- The ESD processor determines if automatic library calls are being processed. 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 control returns to the input processor, the write indicator is tested. If it is on and the TEST option was specified, routine IEWLCSYM will save ESD records containing SDs, PCs, or CMs, block them into 244-byte records (including four bytes of control information), and write them out on SYSLMOD.

- In any input object module the ESD processor saves the CESD line number of the first SD entry whose length is zero. The END processor uses this CESD line to enter the length specified on the END card. (Typical FORTRAN input has the control section length 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.

LR (or LD) Items: If the input ESD item is an LR or LD:

- The LABEL routine determines, when processing an LR 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.

- Since LDs are not referred to by RLDs, they are not numbered in language translator output; therefore, LDs are not renumbered. 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 CE). Column 12 of a PR item may contain an alignment value which indicates that the PR must be aligned to a half-word, full-word, or double-word boundary. The PR is then processed by the freeline, enter, and renumber routines, as described previously.

CM Items: If the input ESD item is CM, 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.

RESOLUTION PROCESSING (CHART CG): 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. 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, the greater length and the most "constrictive" boundary alignment are placed in the CESD entry. (A double word alignment is more constrictive than full word alignment; full word is more constrictive than half word; 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; the END processor 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, 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 Processor."

  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," routine IEWLCDCN removes the CESD item from the chain of library members and the input ER item is entered into the CESD and renumbered.

TXT AND RLD PROCESSOR - 15K AND 18K LEVEL E

When the input processor detects a TXT[1] or RLD record, it gives control to the TXT and RLD processor, passing control information in the general registers. TXT processing is shown on Chart CJ; RLD processing is shown on Charts CH and CI.

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.")

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. Whereas control sections vary in length, the text buffer, into which they are read has a fixed length (1024 bytes). Therefore, a control section longer than 1024 bytes must be divided into portions of 1024 bytes. (The last portion may be less than 1024 bytes.) Each division is called a "multiplicity." For example, a 4100-byte control section contains five multiplicities.

When the first text record of an object module is read, the input text buffer is "established" for the ID of the text record and the multiplicity in which the first byte of text falls. The ID is "renumbered," using the renumbering table, so that it refers to the CESD entry for that control section in the output module. The TXTIOT routine enters this ID and muliplicity into the text I/O table. Input text records of the same multiplicity and ID are moved into the input text buffer at their proper location, relative to their position in that multiplicity, until a change of multiplicity or ID occurs. When the ID or multiplicity changes, the BUFTXT routine writes out the contents of the input text buffer on SYSUT1, and the buffer is established for the new ID or multiplicity.

If an input record contains text which spans two multiplicities, the first part is read into the buffer. BUFTXT then writes out the contents of the buffer onto SYSUT1 and the remainder of the record is moved into the buffer, which is now established to reflect the second of the two multiplicities. Whenever BUFTXT writes out the contents of the input text buffer onto SYSUT1, an entry is made in the text note list (for each entry in the text I/O table there is a corresponding entry in the text note list). The text I/O table keeps a record of each occurrence of a multiplicity and ID which has been encountered in the input. The text note list contains the displacement of the record from the beginning of the text buffer and its relative track address (TTR) on SYSUT1. The text I/O table and the text note list will be used for finding the TXT on SYSUT1 during second pass processing. The text note list may itself be written out on SYSUT1 a maximum of three times if processing a large program causes it to overflow (a fourth portion may remain in main storage); in this case, the TTR of each part of the text note list on SYSUT1 is entered into the text I/O control table.

Since TXT records belonging to load modules have been previously processed, they are written out on SYSUT1 as soon as they are read into the text buffer. Entries are made in the text I/O table as described above. If an input TXT record ID in a load module is marked for deletion or replacement in the renumbering table (RNT), or contains an invalid ID, control is immediately returned to the object or load module processor. (The record is skipped, thereby deleting it.)

Note: When the END statement of an input object module is processed, the object module processor gives control to the TXT processor so that the BUFTXT routine can write out any TXT items still in the input text buffer. This is called an "END statement purge." An input text buffer purge is not required at the end of an input load module.

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

---

[1]Identified by the CCW/RLD record preceding the text record in a load module.

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). The text processor must build records of contiguous text on SYSUT1 so that the second pass processor can place the text into its proper position, within its ID and multiplicity, in the second pass text buffer.

Each byte of 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 non-zero value in the displacement field. When the second pass processor reads back the text from SYSUT1 into the second pass text buffer, it uses this displacement to place the text in its proper position within its ID and multiplicity.

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.

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, the BUFRLD routine writes out (on SYSUT1) all RLD items for the previous P that are in the RLD buffer. The relative track address of the record on SYSUT1 is noted by entering it in the RLD note list. If the entry referred to by the P pointer is marked for deletion in the renumbering table, the RLD items for that control section are not written out on SYSUT1 because the associated text has been skipped.

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 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, control passes to the ESD processor, which places an entry in the delink table and then returns control to the TXT and RLD processor. 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 RLD processor also saves (in the renumbering table) the ID of the delink table entry for the deleted symbol, and sets a "delink value saved" indicator. The ID of the identically-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

---

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

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 the second pass processor must handle PRs in a special manner). If the renumbering table entry is not an ER, marked delete or common, the RLD flag field is marked for relative relocation. This indicates to the second pass processor 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, marked delete or common, the RLD flag field is not marked. This indicates to the second pass processor that the address constant is to be relocated by absolute relocation; the second pass processor 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 Processor.")
- 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 by the address assignment processor 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. 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 5 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 the second pass processor; refer to Table 6.) After all FA fields have been processed, the RLD processor determines if the input RLD record is part of an object module or a load module.

- If the input RLD record is part of an object module, RLD items are placed in the RLD buffer and the next input RLD record is processed. The BUFRLD routine writes out RLD data on SYSUT1 whenever the RLD buffer is full or when there is a change in the P pointer. Each time the contents of the buffer are written out, an entry is made in the RLD note list; the entry contains the renumbered ID of the control section containing the RLD items, the number of bytes of RLD information, and the relative track address of the record on SYSUT1. (For a large program, the RLD note list may itself be written out on SYSUT1 a maximum of three times. The TTR of each portion of the note list on SYSUT1 is saved in the I/O control table.) When the END card of an input object module is processed, the object module processor gives control to the RLD processor so that the BUFRLD routine can write out any RLD items still in the RLD buffer. This is called an "END card purge."
- RLD records in an input load module are read directly into the RLD buffer and are processed there, without moving them. When the RLD data is fully processed, it is written out on SYSUT1 (provided that the control section to which they belong is not being deleted). No RLD buffer purge is necessary at the end of an input load module.

---

[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).

Table 5. Flag Field Processing

Input		Action Performed	Output	
≠ Flag	Type		Flag	Type
0000LLST	Not PR, ER, CM, or delete	Marked for relative relocation	*0100LLST	Relative
0000LLST	ER ('02' in renumbering table)	Marked for absolute relocation	0000LLST	Absolute
0000LLST	Delete or CM ('05')	Marked for absolute relocation if 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	*1000	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	*1000LLST	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 3: Appendix.

END PROCESSOR

When an END statement or the end of an input load module is detected, control is passed to the END processor (Chart CK). The END processor:

- Resets tables that were involved in the processing of the input module (such as the renumbering table).
- Processes entry point information.
- Deletes any CESD lines marked CHAIN or DELETE, and keeps 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).

CONTROL STATEMENT SCANNER

When the input processor detects a control statement (blank in column one), it passes control to the control statement scanner (Charts CL, CM, and CN). The control statement scanner analyzes the statement, detects any errors in format, checks for continuation of comments or operands, and scans a vector table to determine the appropriate control statement processor. Control is then passed to the INCLUDE, REPLACE, LIBRARY, CHANGE, INSERT, OVERLAY, ENTRY, ALIAS, NAME, SETSSI, or HIARCHY control statement processor.

The general format for linkage editor control statements is shown in Figure 8. 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 must 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.

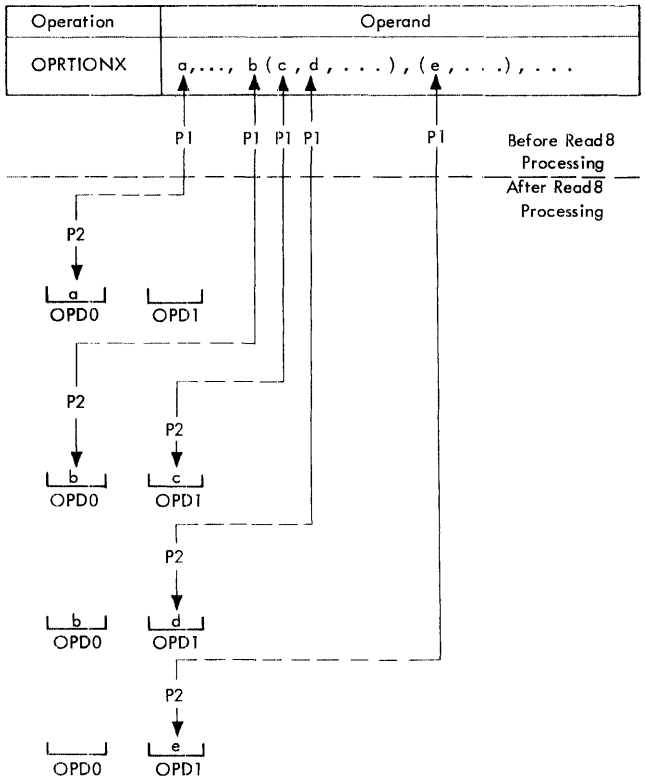


Figure 8. Control Statement Scanner Operation

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 9, 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 10, 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

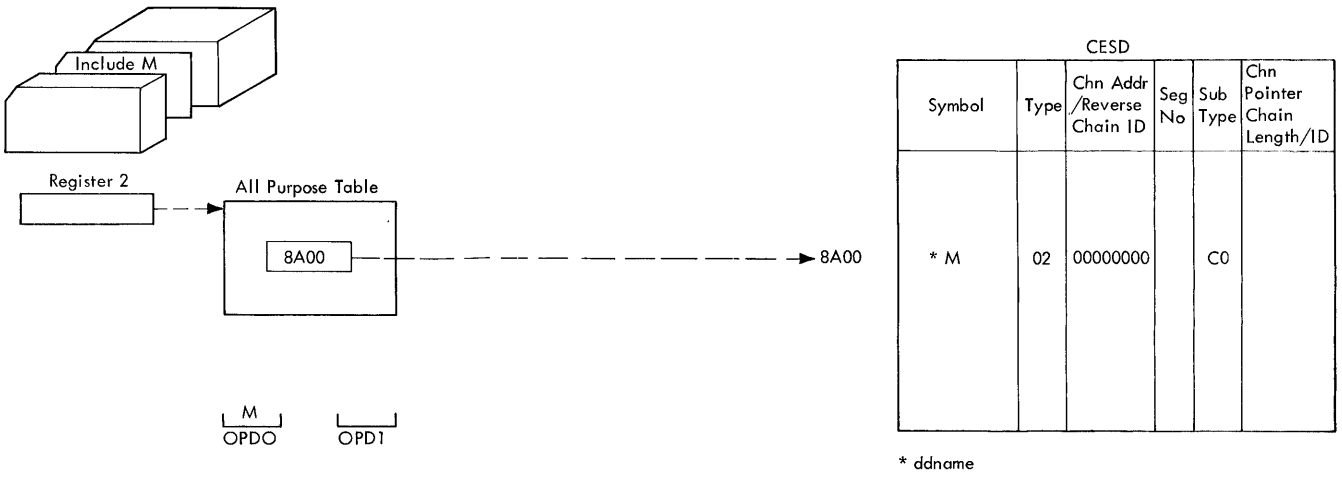


Figure 9. Include Statement Processing for a Sequential Data Set

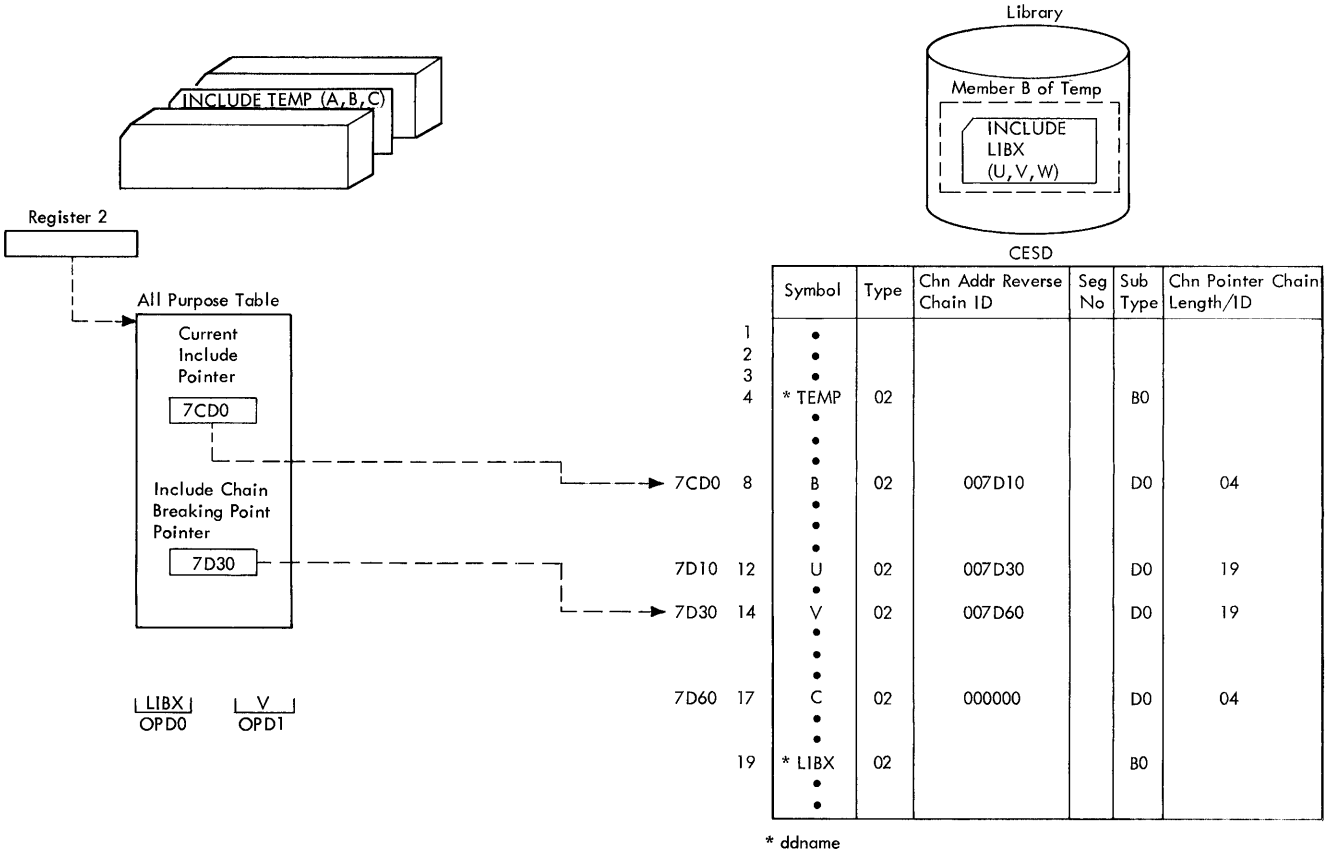
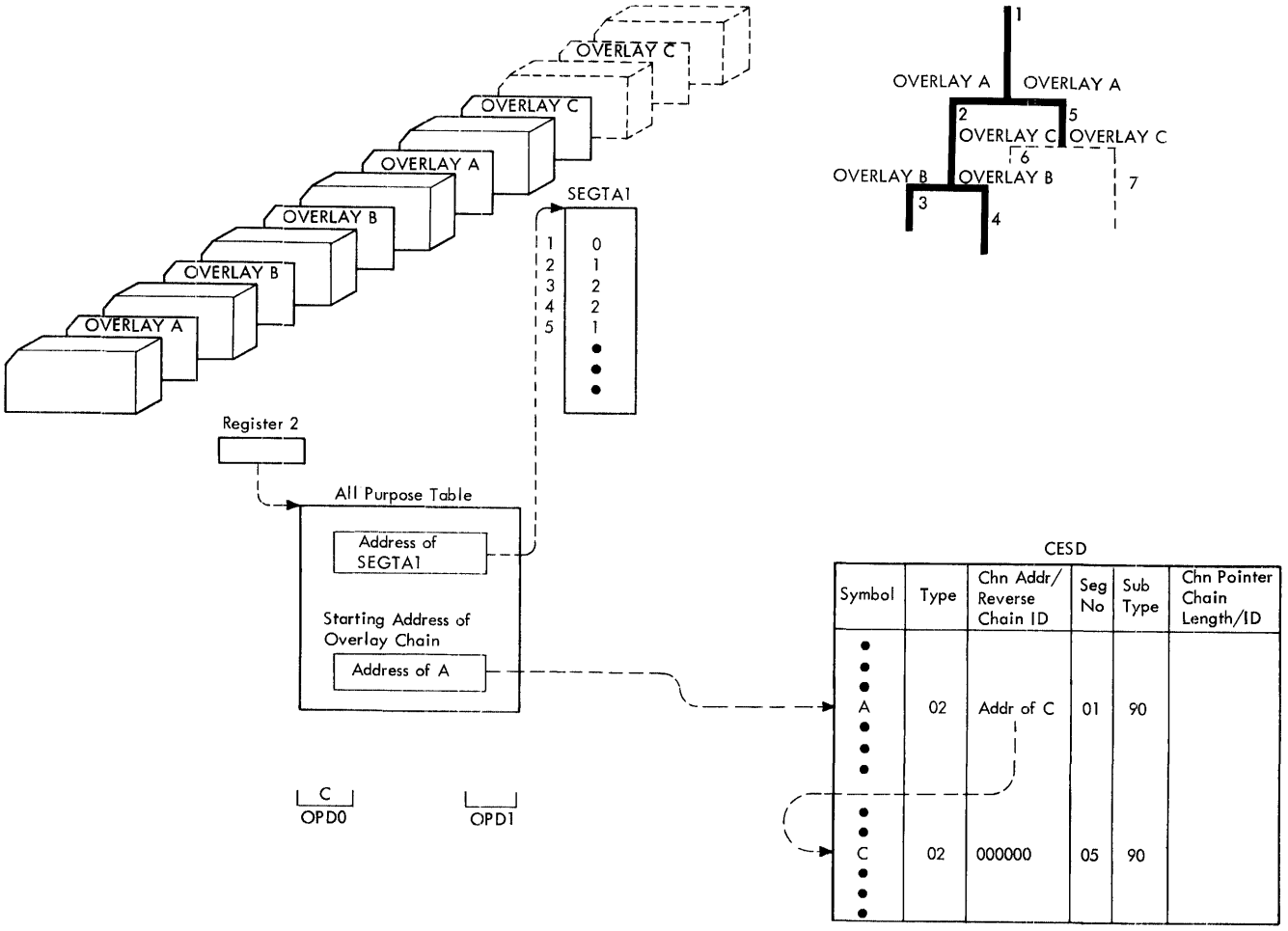


Figure 10. Include Statement Processing With Nested Members

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 CO).

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 11). Entry n in SEGTA1 contains the number of the segment that precedes the nth segment of the overlay



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

Figure 11. Overlay Statement Processing

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 11, 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.

HIARCHY STATEMENT PROCESSOR: The HIARCHY routine first determines whether the hierarchy number is valid. If it is invalid, the statement is printed, an error message is written, and the remainder of the statement is ignored. If the number is valid, it is converted to binary and saved for the Scan routine.

Processing of the statement continues with the collection of the next symbol (up to a comma or a blank). The CESD is searched for this symbol; the location in the hierarchy table corresponding to the CESD item is set to the hierarchy number specified. (The hierarchy table is built during initialization if HIAR was specified on the EXEC statement. The hierarchy table consists of one byte per entry in a one-to-one correspondence with the number of items allocated to the CESD. The address of this table is kept in a full word in the all purpose table.)

If the symbol does not appear in the CESD, the symbol is entered in an unused entry in the CESD, marked external reference, and the hierarchy number is stored in the corresponding entry in the hierarchy table. This procedure is repeated for each additional symbol in the HIARCHY statement.

The intermediate output routine uses the hierarchy table to place the hierarchy number associated with each CESD item in the scatter/translation table.

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.

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 12).

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."

INCLUDE PROCESSOR

The include processor (Chart CO) receives control when:

1. The control statement scanner has detected an INCLUDE statement and the include statement processor has built an include chain.

2. The input processor has detected an end-of-input, and the "more includes" indicator in the all purpose table is on.

The include processor chooses from the include chain the name of the next module to be included. It performs preparatory functions (OPEN, BLDL, and FIND), using the library open (LIBOP) routine, so that the input processor can read in the module.

The LIBOP routine (Chart CQ):

1. Sets an input pointer to the library read block, an area in main storage.

2. Closes the SYSLIB DCB (unless it is open for a PDS currently being used).

3. Changes the data set organization field of the DCB from partitioned to physical sequential if a sequential data set is to be included, and updates the ddname field.

4. Opens the DCB (unless the DCB is already open and in use).

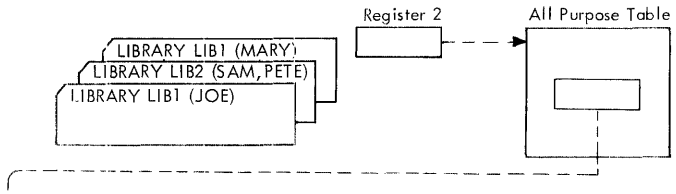
5. Tests the record format field (RECFM) in the DCB to determine if the included module is a load module (U format) or an object module (F format). If it is a load module, the LIBOP routine sets the "load module" indicator in the all purpose table. This indicator is tested by the input processor to determine the type of module being read.

6. Uses the BLDL macro instruction to obtain the attributes of the included module (if it is a load module) and may "downgrade" the attributes of the output load module in the APT accordingly.

7. Uses the FIND macro instruction and the directory entry obtained from BLDL to set a pointer in the DCB to the first record of the member (if it is a load module).

An example of include processing is given in Figure 13. 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 included. The subtype 'D0' indicates that A is a member of a partitioned data set. The pointer 000D refers to the data set DATASETX. 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 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 the member, using the FIND macro instruction, and the "include initiated" indicator is set in the all purpose table. The chain pointer field of the CESD entry for A is then tested. Since, in the example shown, this field does not contain zeros, the "more includes" indicator in the all purpose table is set, and control returns to the input processor to read this data.



	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

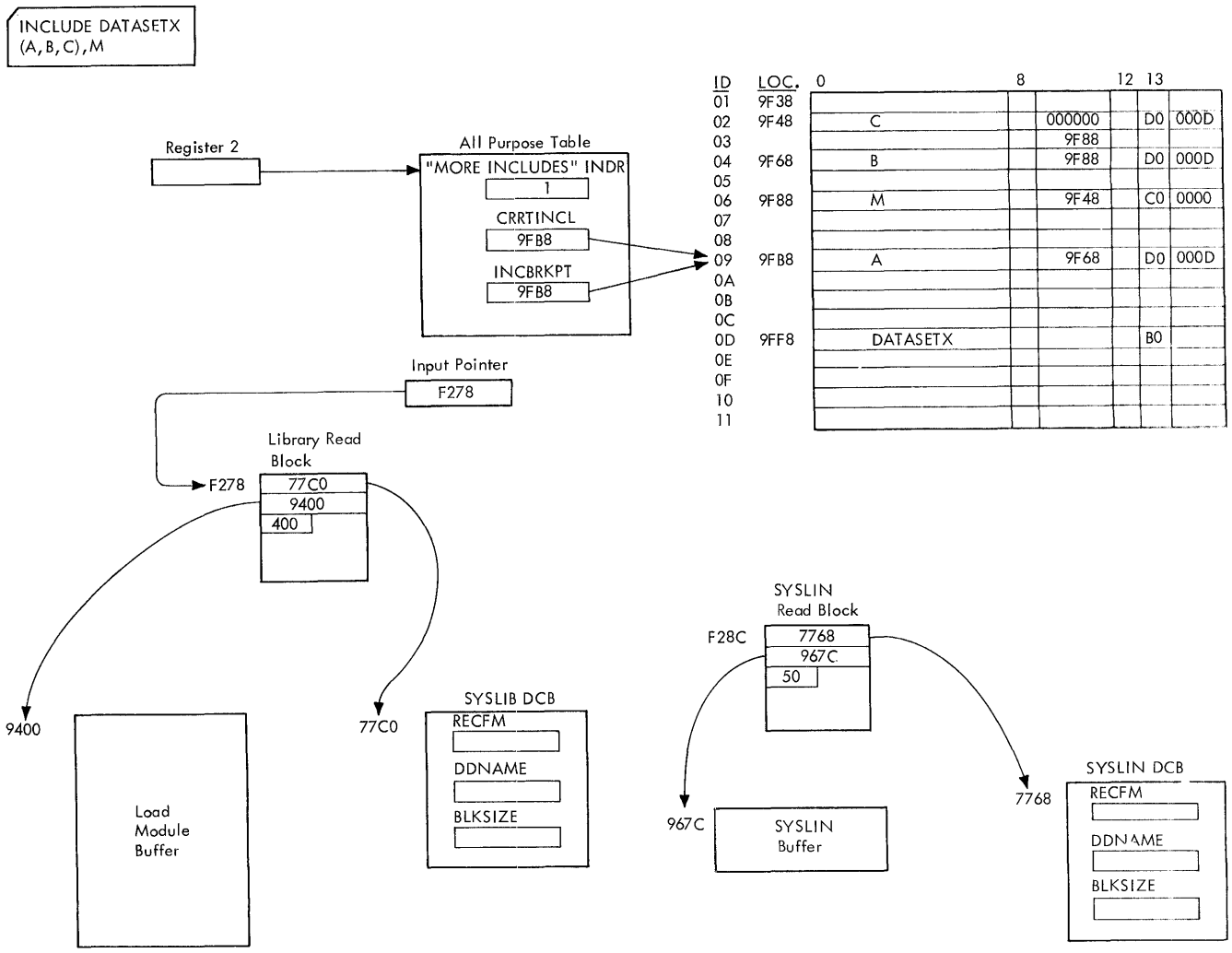
	Symbol	Type	Chn Addr /Reverse Chain ID	Seg No	Sub Type	Chn Pointer Chain Length/ID
01						
02						
03						
04	JOE	00				
05						
06	LIB2	02	00		B0	07
07	SAM	02	06		02	08
08	PETE	02	07		03	00
09						
0A	MARY	02	0C		03	00
0B						
0C	LIB1	02	00		B0	0A

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 12. Library Statement Processing



• Figure 13. Include Processing

The input processor reads member A using the input pointer and library read block. Module A is then processed. When the end of module A is reached, the input processor again calls the include processor because the "more includes" indicator in the APT is set.

When the include processor receives control again, the chain address field of the CESD entry for A is used to find item B; item B is then processed in the same manner as A. Item A is deleted from the chain, and the CESD line is marked "null."

Note: If the item to be included is a sequential data set (such as M, in Figure 10), there is no chain pointer in the CESD entry. Differences in processing for this type of include item are shown in Charts CO and CQ.

AUTOMATIC LIBRARY CALL PROCESSOR

The input processor passes control to the automatic library call processor (Chart CP) at the end of SYSLIN input, or when a NAME statement has been detected (provided that the NCAL option was not specified and no more includes are to be processed).

The automatic library call processor performs 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. The second series of scans 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 14. 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 the ESD processor, and the LIB1 chain ID now points to the line containing TOM.) The automatic library call processor operates on the library chains, as modified by the ESD processor (diagram B).

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	LIB1	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 14. Automatic Library Call Processing

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). The LIBOP routine opens LIB2 and uses the BLDL macro instruction 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 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. LIBOP uses the FIND macro instruction to set a pointer in the SYSLIB DCB to the member PETE; control is then returned to the input processor to read in PETE.

When the input processor returns control again to the automatic library call processor, 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. (The input processor may have read in object modules with additional LIBRARY statements.) When the automatic library call processor reaches the last line of the CESD, it begins the second series of scans.

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 the automatic library call processor receives control from the input processor during the second series of scans, it resumes the scan at the beginning of the CESD (because ER items from a library member may have been entered in any available CESD line).

When the automatic library call processor completes the second series of scans, control is passed to the address assignment processor.

ADDRESS ASSIGNMENT PROCESSOR

At the conclusion of input processing, when all automatic calls have been processed, control is passed to the address assignment processor (charts DA through DD). The address assignment processor performs the following operations:

- Closes the SYSLIB DCB if it was opened during input processing. It deletes CESD entries for ER items marked included, called, ddname, or overlay in the subtype field. These lines are marked "null" and are deleted if the

---

[1]SYSLIB is the standard library whenever the linkage editor is executed as a job step. If another program calls the linkage editor via the LINK macro instruction, the ddname of the standard library is passed in a parameter list.

module is processed again in a subsequent execution of the linkage editor.

- Computes, for programs in overlay, the size of SEGTAB[1], enters the size in the all purpose table, and places 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.

- Uses the ENTAB size determination routine (Chart DB) to enter segment numbers for label references in the CESD. If the program is in overlay, this routine also scans the calls list (built during RLD processing), entering pointers from one chain of calls to the next chain; determines the number of ENTAB bytes[2] for each segment; and places a PC-delete type entry in the CESD for each ENTAB. (Refer to "ENTAB Size Determination Routine.")

- Scans the CESD and assigns temporary linked addresses to SD-, PC-, and CM-type entries. 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.

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

- Accumulates 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.

- Determines 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.

- Processes the SEGLGTH table (if the program is in overlay) to determine the starting address of each segment, relative to the beginning of the program. 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. (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 for the segment, 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.

- Performs 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.

- Makes a final scan of the CESD to assign a final linked address to each label reference. (If in overlay, this is the third scan of the CESD; if not in overlay, it is the second scan.) 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.

---

[1]SEGTAB size = 24 + (4 x number of segments).

[2]ENTAB size = 12 + (12 x number of unique downward calls per segment).

- Marks the program as not executable if there are still unresolved external references and if neither the no call option nor the LET option has been specified.
- Uses the entry processor (Charts DC and DD) to build the alias table and compute an entry point for the program. (Refer to "Entry Processor.")

ENTAB Size Determination Routine

The ENTAB size determination routine computes 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 the second pass processor. 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 the ENTAB size determination routine is given in Figure 15. 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. The ENTAB size determination routine:

- Scans the CESD for LR-type entries and enters their segment numbers. In Figure 16, 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.
- Scans the calls list, inserting chaining values that point from one group of R and P pointers to the next group.

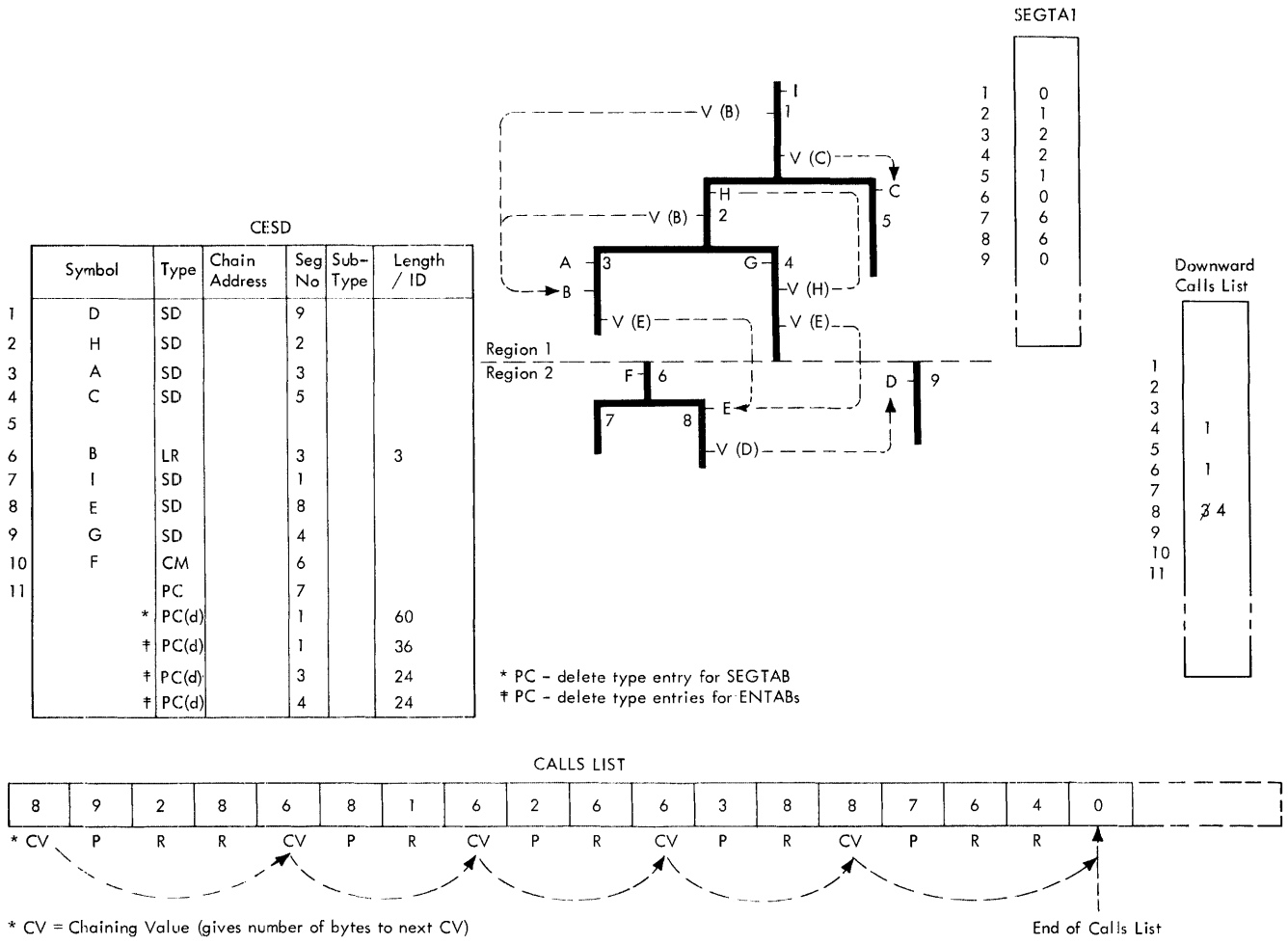


Figure 15. ENTAB Size Determination

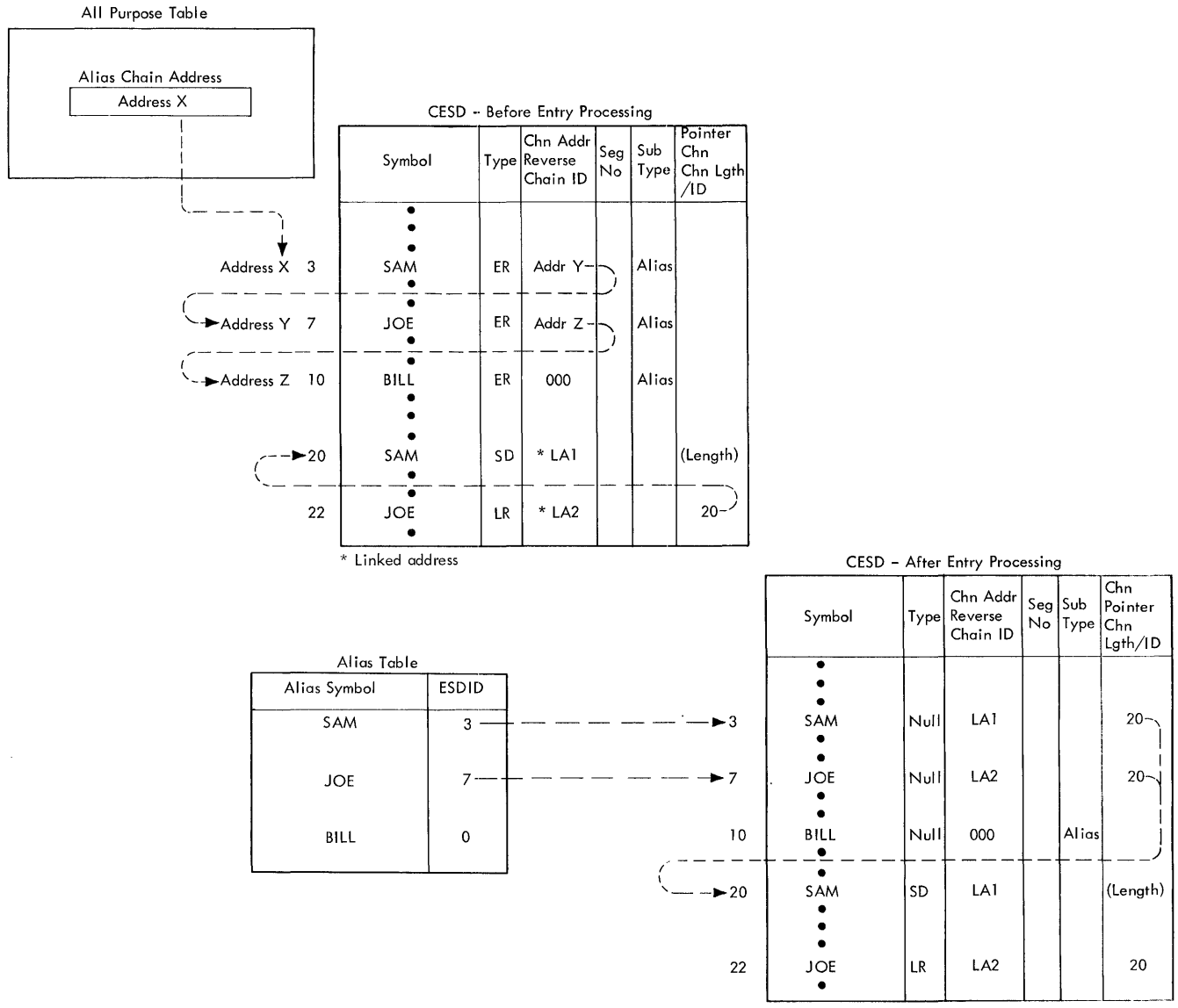


Figure 16. Processing of Alias Symbols by the ENTRY Processor

- Scans the calls list, for each segment (starting with segment 1), to 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 control is passed to routine IEWL-CAD1, which enters a PC-delete type entry for the ENTAB in the CESD. Referring to Figure 15, 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 ENTAB size determination routine examines the associated R pointers (which refer to referenced symbols) to determine the segment in which each referenced symbol resides. In Figure 16, 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 15, 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 routine IEWLCAD1 creates a PC-delete type entry for the ENTAB in the CESD.

2. When the scan for segment 1 is completed, the ENTAB size determination routine scans the calls list for P pointers that refer to segment 2. In Figure 15, the third P pointer in the calls list refers (via the CESD) to segment 2. The associated R pointer 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 ENTAB size determination routine scans the calls list for P pointers that refer to segment 3. In Figure 15, 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 IEWLCAD1 creates a PC-delete type entry for the ENTAB in the CESD.

4. The ENTAB size determination routine scans the calls list for P pointers that refer to segment 4. In Figure 15, 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 entry is created in the CESD.

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

Entry Processor

The entry processor (Charts DC and DD):

- 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 processor 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 16.) 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 PROCESSOR

The intermediate output processor (Chart EA):

- 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. Relative relocation factors are later moved into the HESD when the length/ID field is no longer needed. (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.[1] 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 records as scatter/translation information. Storage hierarchy designations are included in the tables if the HIAR bit is set. 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 17.)

- Completes the HESD by moving in relative relocation factors from the relocation constant table for SD-, PC-, CM-, or LR-type HESD entries. Each relocation constant is a 3-byte value; the value may be negative[2] because it is the difference between the address assigned to a symbol by the linkage editor and the address of the symbol in the input module. Unused HESD space is made available to the second pass RLD input buffer (by decreasing the starting addresses of the RLD buffer and the TXT and RLD note lists, which are located between the HESD and the RLD buffer).

- 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 text with the highest ESD ID 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 the second pass processor 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 high ID (HIID) table along with its associated segment number.

---

[1]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.

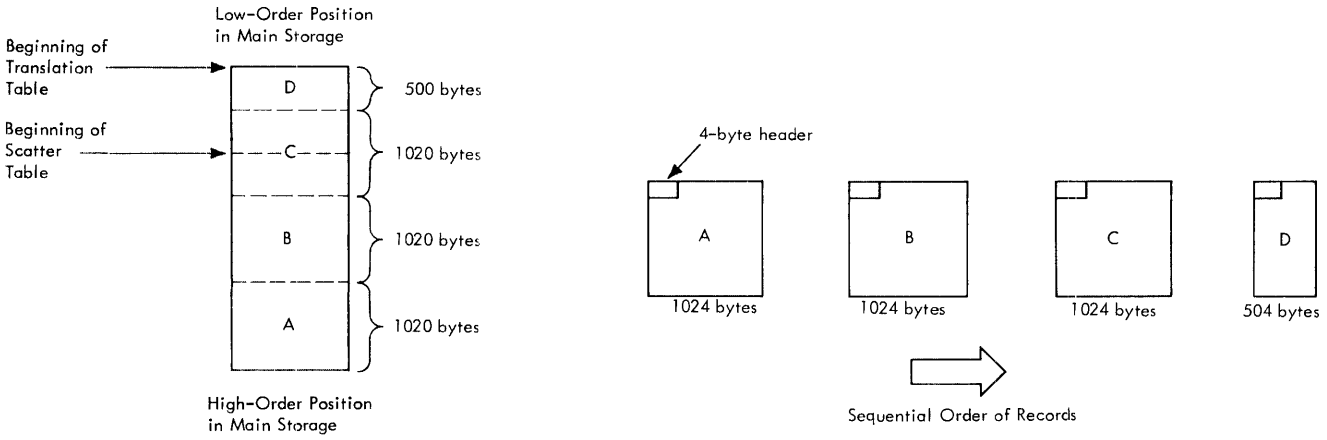


Figure 17. Writing Scatter/Translation Records

SECOND PASS PROCESSOR

After intermediate output processing, the second pass processor reads back TXT and RLD records from the intermediate data set (SYSUT1) into the second pass text buffer and second pass RLD input buffer. Address constants contained in the text are relocated and control/RLD records are created. The TXT and control/RLD records are then written out on SYSLMOD in a format that can be loaded by program fetch. The second pass processor also creates information required by the overlay supervisor and program fetch for the processing of an overlay load module; this information consists of ENTABs and associated RLD items used to relocate the address constants. The general operation of the second pass processor is described in the following paragraph. The method used to relocate address constants is described in "Relocation of Address Constants" and "Relocation Routine."

SECOND PASS OPERATION - 15K AND 18K LEVEL E

When the second pass processor (Charts FA through FE) receives control, it performs the following operations:

- The half ESD (HESD) table is searched for a PC- or SD-type entry to determine the ID of the first control section to be processed in the current segment. (The current segment is initially segment number one.) The multiplicity is initialized to zero and the text input/output table (TXTIOT) is then scanned for this ID and multiplicity; the entry containing this ID and the corresponding item in the text note list are used to find the location of that multiplicity of text on SYSUT1.
- All text records that pertain to the current multiplicity are read from SYSUT1 into the second pass text buffer. The second pass text buffer consists of two 1K areas; each area can hold a single multiplicity of text (1024 bytes). Two areas (output text buffer 1 and output text buffer 2) are used to provide for input/output overlap and processing of "split" address constants. When text is read in, it is placed in only one of these 1K areas; simultaneously, text may be written out of the other 1K area (unless a split address constant is being processed). The length of text read into the buffer is determined by checking the residual byte count in the input/output block (IOB). (Maximum size - residual byte count = size of record.)
- All RLD records associated with the control section currently being processed are read into the second pass RLD input buffer. The RLD input buffer length is a multiple of 244; RLD records are read into it 244 bytes apart to simplify recognition of the beginning and end of RLD records when they are relocated. (Relocation is performed by record.) The second pass processor searches the RLD note list for the current ID, and uses the associated TTR to find the location of the RLD records on SYSUT1. RLD records are read back from SYSUT1 only if the RLD note list entry contains a multiplicity equal to or lower than the current multiplicity. If all RLD records to be read in cannot fit in the buffer, those in the buffer are processed. The contents of the RLD input buffer are "compressed" by scanning the buffer and eliminating any record that has been completely processed. Additional RLD records are then read into the available space. If

sufficient space cannot be made available, the last record in the buffer is processed and overwritten until every RLD record has been read in and processed by the relocation routine. An overwritten record may contain RLD items for a later multiplicity of text; if it does, it must be read in again for that multiplicity.

- A control record is written out on SYSLMOD (the control information pertains to the text that follows it).
- The buffer relocation constant is computed for all RLDs associated with the multiplicity of text in the second pass

text buffer. This constant is added to the address field of an RLD item to determine if an address constant is contained in the second pass text buffer.

- The RLDs are relocated, using either relative or absolute relocation factors (refer to "Relocation of Address Constants"). As each RLD item is relocated, it is moved to the second pass RLD output buffer. The RLD output buffer can hold 30 8-byte RLD items. If an address constant has been relocated within the second pass text buffer and its corresponding RLD item cannot fit in the RLD output buffer, the contents of the RLD output buffer must be written out. However, because the contents of the text buffer must be written out first, a "dummy" text record (whose size is the same as the relocated text record) must be written out to reserve space for the text on SYSLMOD. The contents of the RLD output buffer are then placed on SYSLMOD. The dummy record is written out only for the first overflow of the RLD buffer, for a given multiplicity of text.

- When all RLDs pertaining to the text in the second pass text buffer have been processed, the text is written out on SYSLMOD. If a dummy text record was written, the text in the second pass text buffer will overwrite it, using XDAP ("execute direct-access program"), to maintain the proper output load module format.

- If another multiplicity of text is to be processed for the same control section, the operations described above are repeated for the new multiplicity. The RLD items are written out in a control/RLD record after the text to which they pertain. (The control information pertains to the next text record to be placed on SYSLMOD; the RLDs pertain to the previous text.)

- 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 of text" indicator in the all purpose table), the second pass processor sets indicators 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 control is passed to the final processor.

  Note: If the output load module is to be structured for overlay, the second pass processor creates a list of relative track addresses (TTR list) 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. The second pass processor also produces a PC-type control section containing ENTAB entries in each segment where the text requires them. This process is described in the paragraphs "ENTAB Creation" and "Relocation of V-Type Address Constants in Overlay." The second pass processor also creates the RLD records required by program fetch to relocate address constants contained in the ENTABs.

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).

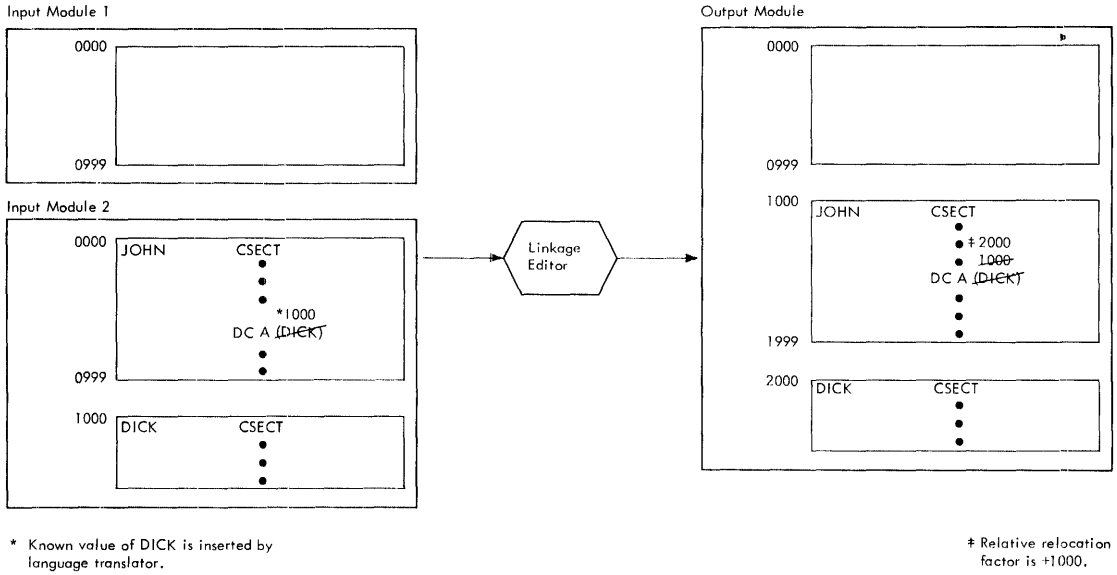
The value of a branch type or non-branch type address constant depends on a symbol in the CESD. To adjust an address constant to its proper value in the output load module, the linkage editor uses an absolute or relative relocation factor. The absolute relocation factor is the address assigned by linkage 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 18. 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.

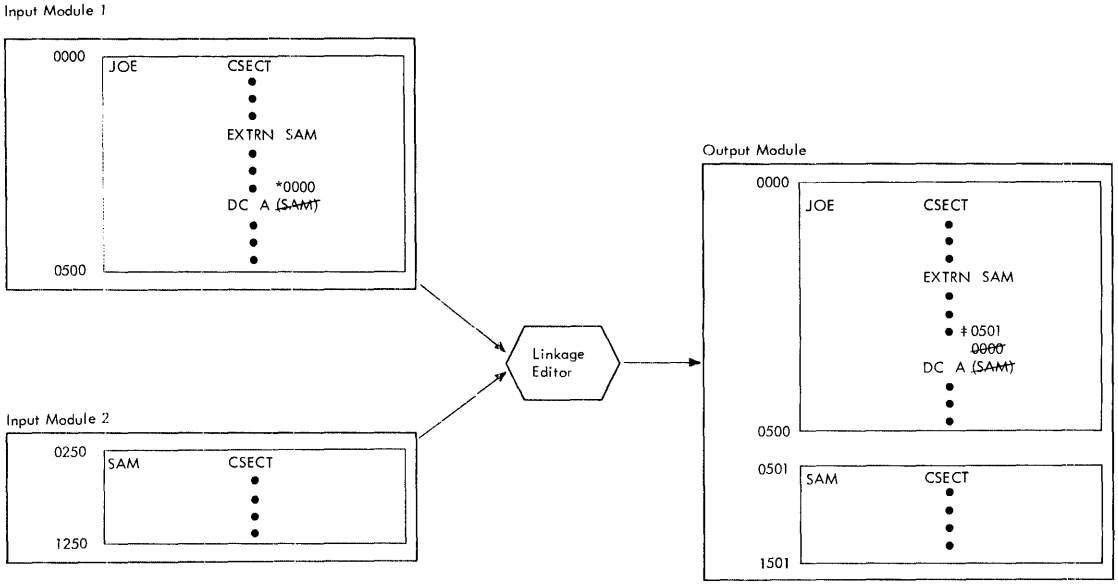
An absolute relocation factor is used for a non-branch type address constant if the symbol referred to by the address constant does not have a defined value within the same input module. (The R pointer of the RLD item refers to an external reference.) An example of absolute relocation of a non-branch type constant is shown in Figure 19. 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.



* Known value of DICK is inserted by language translator.

† Relative relocation factor is +1000.

Figure 18. Non-Branch Type Address Constants - Relative Relocation



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

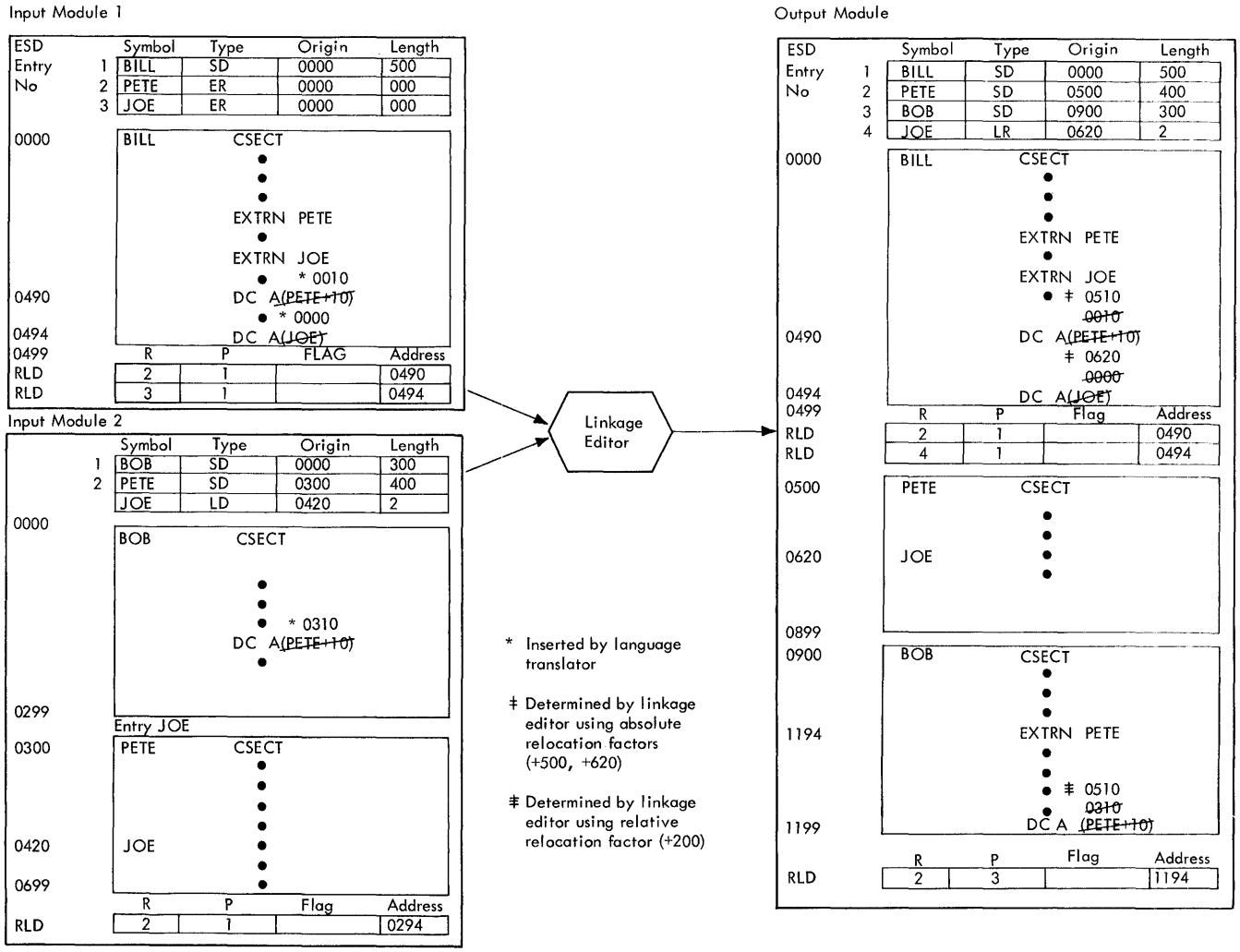
† Actual address of SAM in the output module (0501) is added to value of address constant. (Note that the relative relocation factor of SAM is +251.)

● Figure 19. Non-Branch Type Address Constants - Absolute Relocation

Figure 20 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 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



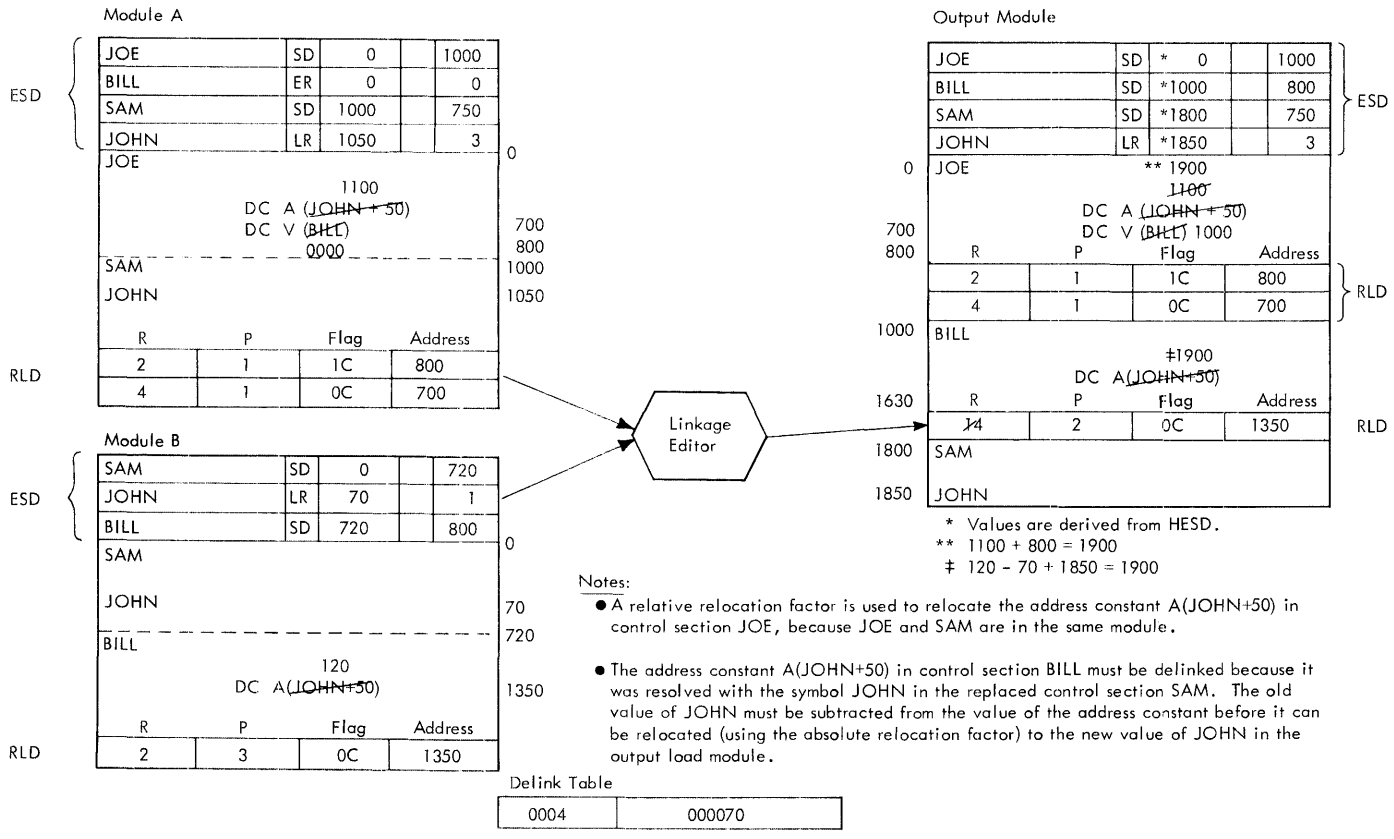
•Figure 20. Non-Branch Type Address Constants - Absolute and Relative Relocation

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.)

Delinking of an A-type address constant is shown in Figure 21. Input load modules A and B both contain control section SAM.



• Figure 21. Example of Delinking

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 and 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.

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 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 a V-type address constant unless:

1. It is in the same segment as the symbol to which it refers.

2. It refers to a symbol in another region.

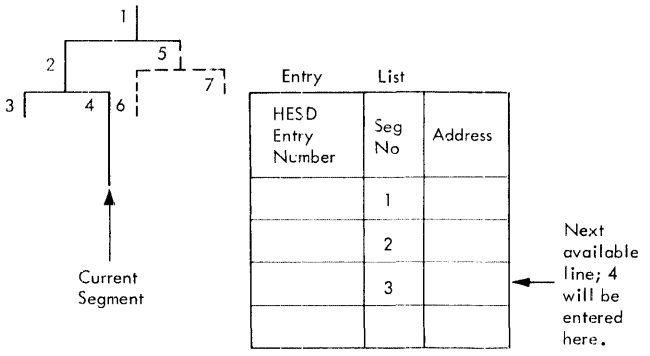
3. It also appears in a segment higher in the path.

In case 3, an ENTAB entry already exists for the V-type address constant. (The entry was created when it was encountered in the higher segment.) Any recurrence of the V-type address constant in a lower segment is resolved to the existing entry. 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 the second pass processor 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 22, segment 4 is being processed; the entry for segment 3 is removed since it is not higher in the path of 4.)

---

[1]All address constants 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.)



• Figure 22. 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 than 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 than 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[1] and an ENTAB 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.)

[1]Whenever a line is added to the entry list, an RLD item is created in the ENTAB RLD buffer so that the address in the ENTAB entry can be relocated when the segment is loaded by program fetch for execution.

ENTAB Creation

The ENTAB creation routine uses the number of RLD items in the ENTAB RLD buffer 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 Appendix A.

ENTAB creation is shown in Figure 23. 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.

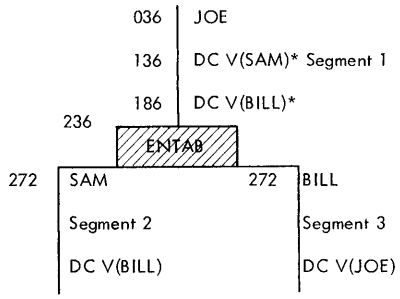
Diagram A.

HESD

	Type	L.E. Assigned Address	Seg	Relative Relocation Constants
JOE	SD	36	1	200
SAM	SD	272	2	500
BILL	SD	272	3	500
SEGTAB	PC	0	1	36
ENTAB	PC	236	1	36

RLD	R	P	Flag	Address
	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 Buffer

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
248	47FF 0012	00000 272	03	000000
260	Standard Last ENTAB Entry			

} ENTAB

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 Buffer

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 Buffer

None

Entry List

*

* Same as after processing segment 1

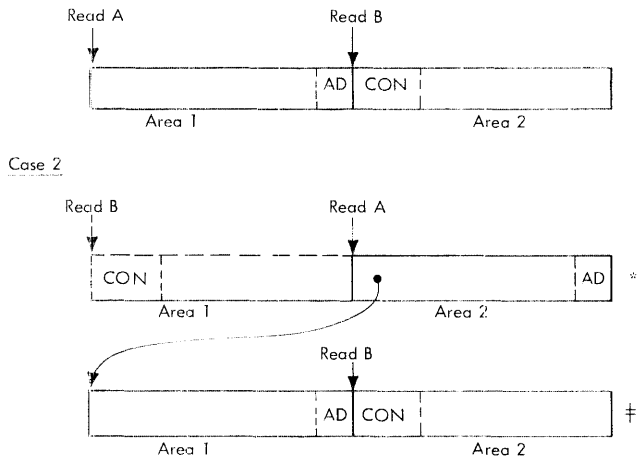
•Figure 23. ENTAB Creation

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 from 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.

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

"Split" Address Constants - Level E

Since each of the two 1K areas of the second pass text buffer can hold only one multiplicity of text, an address constant in a control section containing more than one multiplicity of text may be "split" across a buffer boundary. This situation is shown in Figure 24. In case 1, the address constant is split across the boundary between areas 1 and 2 of the second pass text buffer; this presents no problem, because the two parts are back-to-back, in their proper sequence.



* If Read B placed text in ..e half of buffer, portions of address constant would not be contiguous.

‡ Text is moved in buffer so that after Read B, portions of address constant are contiguous.

- Figure 24. Split Address Constants in the Second Pass Text Buffer

In case 2 however, the two parts of the split address constant would not be contiguous; read B (from SYSUT1) would place the second part in area 1 of the buffer. To avoid this situation, after read A the contents of area 2 are moved to area 1. Read B then places the next multiplicity of text in area 2; the address constant can now be relocated and the text in area 1 can be completely processed. The RLD cannot be placed in a control/RLD record until the text in area 2 has been written out on SYSLMOD. Therefore, the RLD for the split address constant is saved in the HESD prefix, an 8-byte area that precedes the half ESD. (The RLD is saved because an RLD describes a relocatable address constant and cannot be written out until the text containing the address constant is available.) After the text in area 2 is written out on SYSLMOD, the RLD is moved to the output buffer.

RELOCATION ROUTINE - LEVEL E

The relocation of address constants is performed by the relocation routine (Charts FC, FD, and FE); the routine operates on the following input data:

- The address of an RLD record in the RLD input buffer.
- The address of the next available location for an RLD record in the RLD input buffer.
- The address of the next available entry in the RLD output buffer.
- The buffer relocation constant (BRC) where:

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

The relocation routine operates in the following manner:

1. The size of the RLD record to be processed is determined.
2. Each RLD item is scanned to determine if:
   a. It describes an address constant currently in the TXT buffer (BRC + address contained in RLD address field falls within the boundaries of TXT buffer).
   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 6). 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.[1]

6. Steps 2 through 5 are repeated until all RLD items have been scanned in the RLD record being processed.

7. If there are more RLD records in the input buffer to be processed, the address of the next record is determined and steps 1 through 6 are performed. When there are no more RLD records to be processed for the current multiplicity of text, the relocation routine determines which RLD items must remain in the RLD input buffer for the next text record. It then adjusts the contents of the input RLD buffer and determines where the subsequent RLD items are to be read in.

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). 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. (The last record in the buffer is overwritten to provide space for the incoming record.)

When the relocation routine scans an RLD record in the input RLD buffer it determines if the record contains RLD items belonging to a later multiplicity of text in the current control section. If all RLD items in an input RLD record have been processed, the record is marked for deletion from the input RLD buffer to make room for more input RLD records. If the RLD record contains RLD items pertaining to text that has not yet been read in from SYSUT1, the record is marked "in core" in the record length field of the RLD note list, indicating that it is not to be deleted.

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.) Before such records are read in, the input RLD buffer is scanned again to eliminate all records marked for deletion and a "pushup" routine packs the remaining records (those marked "in core") so that they are contiguous from the beginning of the buffer. The records to be read in are then placed in the empty portion of the buffer; these records are processed in the same manner as those already residing in the buffer. This process is repeated until all records that may contain RLD items pertaining to the current multiplicity of text have been scanned and processed.

If there are no records in the input RLD buffer that are marked for deletion, and additional RLD records for the current multiplicity of text must be read in, a record is read in so that it overwrites the last record in the buffer. Each record is read in, scanned, and processed in this manner until all RLD records for the current multiplicity of text have been processed.

When all RLD records for a given multiplicity of text have been processed, the "pushup" routine eliminates all records marked for deletion and RLD records for the next multiplicity of text are read into the buffer.

To avoid processing the same RLD record twice for the same multiplicity of text, a "processed" indicator may be set in the record length field of the RLD note list when a record is overwritten. When a new multiplicity of text is to be relocated, the RLD note list is scanned sequentially

---

[1] 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.

Table 6. 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
0100LLST	Relative	Relative relocation factor is added to value of address constant	0000LLST	A-type
1000LLST	Delink	Delink value is subtracted from address constant and absolute relocation factor is added to 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.

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. However, such a record may be removed from the buffer so that other records can be read in; such a record is marked "processed" when the scan has not reached its entry in the note list. (When the scan reaches the entry it will be ignored and the processed indicator will be reset.)

FINAL PROCESSOR - 15K AND 18K LEVEL E

The final processor (Chart GA) performs "cleanup" functions, and is the last operation of linkage editor processing. The final processor:

- Writes the TTR note list, created by the second pass processor, 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.
- Places each entry in the proper format for the partitioned data set directory, modifies it if there are alias symbols, and issues a STOW macro instruction[1] for the member name and each alias.
- Checks attributes (reusable and reentrant). If the attributes have

[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.

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.")

- Passes control to the diagnostic directory print routine to print out a directory of logged errors.
- Releases main storage space that was allocated to linkage editor.
- Checks for any final options and passes control to the module map routine if a module map or cross reference table was requested.
- If the module has been marked "not executable," an error message is printed out. Control is then returned to the caller, or, if a NAME card terminated SYSLIN input for the load module being processed, to the initial processor.

ERROR LOGGING

Whenever an error condition (other than input/output errors) is detected during linkage editor processing, the error logging routine (Chart GB) sets an indicator in an error logging map and prints out a coded diagnostic message. During final processing, the diagnostic directory print routine scans the error logging map. When an indicator is found "on" in the map, this routine refers to an associated list (via a table of pointers) which is used to build a diagnostic message.

Note: An example of error logging in level E is given in Figure 25. 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) as shown in Table 7.

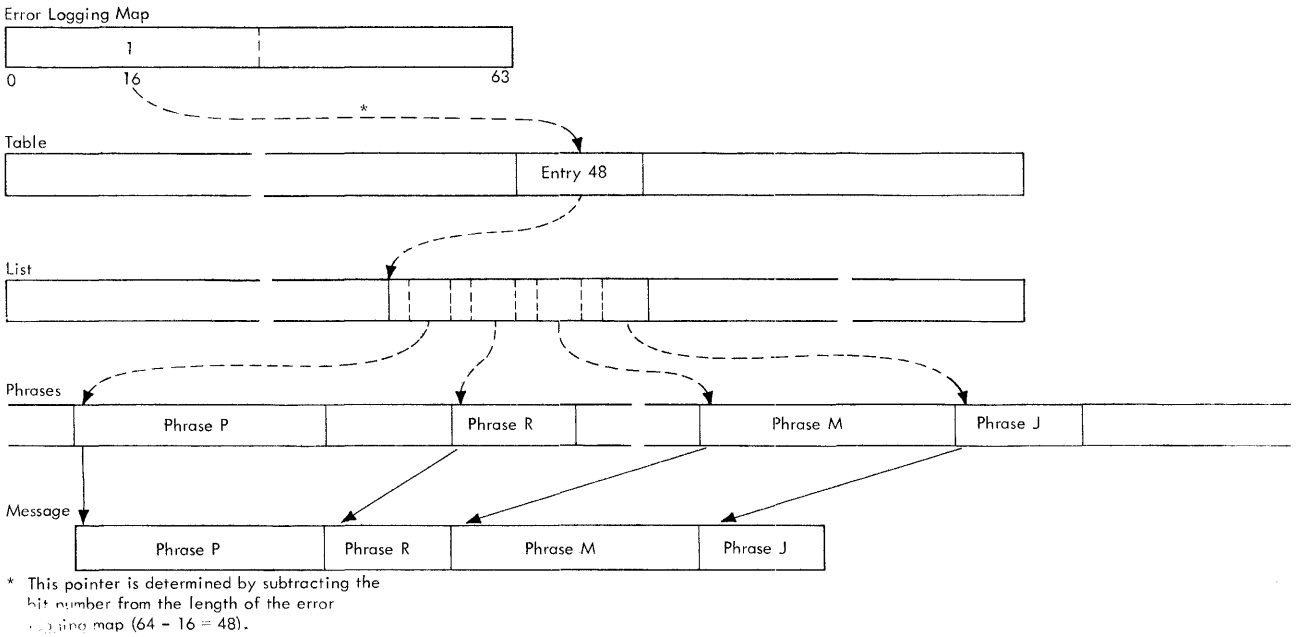


Figure 25. Building Error Messages (Level E)

INPUT/OUTPUT ERROR HANDLING

When the control program detects an input/output error during linkage editor processing or when the second pass processor finds an error following execution of an XDAP macro instruction, the SYNAD routine (chart GD) is entered for an analysis of the error and for production of a message describing the error.

Error analysis depends on whether BSAM, BPAM or XDAP was in use at the time of the I/O error. Thus, the SYNAD routine has three entry points. The proper SYNAD entry point is assembled into the SYNAD field of the DCBs for the SYSLMOD, SYSLIN, SYSUT1, and SYSPRINT data sets. The SYNAD field for the SYSLIB DCB is filled by the include processor when the DCB is opened. (For SYSLIB the entry point corresponds to BSAM or BPAM.)

Upon entry from BPAM or XDAP, the SYNAD routine issues the SYNADAF macro instruction for error analysis and message construction. The routine moves the SYNAD-generated message into the TEXT I/O table area, inserts the proper ID into the message, and then prints the completed message. The SYNAD routine then either returns control to continue linkage editor processing (in the case of an input failure from SYSLMOD during module map processing) or passes control to the final processor to terminate the edit without output.

When the SYNAD routine is entered from BSAM, a check is made to determine whether the error occurred while writing to SYSPRINT. If this is the case, the macro instruction is not issued; instead, control is immediately returned to the caller. If the error did not occur while writing to SYSPRINT, SYNAD operation is the same as for BPAM and XDAP.

• Table 7. Error Message -- Module Cross Reference Table

MMS	Module Where Error Occurred
012	IEWLESCD
022	IEWLESDC
033	IEWLCENT
043	IEWLCENT
053	IEWLCENT
063	IEWLCENT
073	IEWLCENT
083	IEWLCENT
093	IEWLCENT
102	IEWLCEND
113	IEWLCENT
123	IEWLEADA
132	IEWLEADA
143	IEWLEOUT
152	IEWLCENS
161	IEWLCENS
172	IEWLCENS
182	IEWLCENS
193	IEWLEADA
201	IEWLEADA
212	IEWLEINP
222	IEWLCESD,IEWLEINP,IEWLETXR
232	IEWLCESD,IEWLEINP,IEWLETXR
241	IEWLCESD
254	IEWLCESD,IEWLEADA
264	IEWLCESD
272	IEWLCINC
284	IEWLCSCN,IEWLEINT
294	IEWLCFNL
302	IEWLCSCN
314	IEWLCSCN
324	IEWLCSCN
332	IEWLCSCN
342	IEWLCINC
354	IEWLETXR
364	IEWLETXR
374	IEWLETXR
382	IEWLETXR
394	IEWLCFNL
404	IEWLCFNL
412	IEWLCFNL
421	IEWLCFNL
432	IEWLCINC
444	IEWLESCD
454	IEWLESCD
461	IEWLEADA
472	IEWLCENT
484	IEWLEINP
492	IEWLCSCN
502	IEWLCFNL
512	IEWLCINC
522	IEWLCINC
532	IEWLCINC
543	IEWLCFNL
630	IEWLCMAP

MODULE MAP AND CROSS-REFERENCE TABLE

If the MAP option is specified, the final processor passes control to the module map processor (Chart GC). The module map processor requests main storage space, opens SYSLMOD for input, and reads in ESD records. The ESD 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 external symbols within the control sections. Control sections in an overlay output module are grouped by segment and are listed in ascending order of their assigned addresses within the segment.

If the XREF option is specified, the module map processor also reads back the RLD records from SYSLMOD and builds the cross-reference table. 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.

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, routine ALL001 is on card IEWLEINT. 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).

Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment (15K,18K)	Chart ID	Synopsis
Alias Table	Table	IEWLCENT	IEWLCENT	9,6	--	ALIAS symbols from CESD
ALL001	Allocation Routine	IEWLEINT	IEWLEINT	3,2	BA	Allocate main storage
All Purpose Table	Table	IEWLEAPT	IEWLEAPT	1,1	--	Major communications area
Calls List	Table	IEWLETXR	IEWLERAT	6,3	--	Entries for V-type ADCONS
CESD	Table	IEWLCESD	IEWLCESD	12,7	--	ESD control information
Delink Table	Table	IEWLEINP	IEWLEINP	4,3	--	Entries for symbols being deleted
Downward Calls List	Table	IEWLCENS	IEWLCENS	9,6	--	Downward calls from V-type ADCONS
ENTER	Enter Routine	IEWLCESD	IEWLCESD	12,7	CF	Enter ESD item in CESD
Entry List	Table	IEWLESCD	IEWLESCD	12,7	--	Control information for V-type ADCONS
ENTAB RLD Buffer	Table	IEWLESCD	IEWLESCD	12,7	--	ENTAB records built here
FREELINE	Freeline routine	IEWLCESD	IEWLCESD	12,7	CF	Select next available line in CESD
Half ESD	Table	IEWLEOUT	IEWLEOUT	9,6	--	ESD control information
High ID Table	Table	IEWLEOUT	IEWLEOUT	9,6	--	High ID for each segment
IEWLCAD1	Entry Point	IEWLEADA	IEWLEADA	9,6	DB	Make CESD entries for ENTABs
IEWLCAUT	Entry Point	IEWLCINC	IEWLCINC	8,4	CP	Automatic library call processing
IEWLCBTP	CSECT	IEWLCBTP	IEWLCBTP	10,8	GA	Print diagnostic messages

(Continued)

Microfiche Directory (Continued)

Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment (15K,18K)	Chart ID	Synopsis
IEWLCDCN	Entry Point	IEWLCRCG	IEWLCRCG	7,3	CG	Removes CESD item from library chain
IEWLCDLK	Entry Point	IEWLEINP	IEWLEMDI	4,3	--	Builds Delink Table
IEWLCEND	CSECT	IEWLCEND	IEWLCEND	6,3	CK	END Statement Processing
IEWLCENS	CSECT	IEWLCENS	IEWLCENS	9,6	DB	ENTAB size determination
IEWLCENT	CSECT	IEWLCENT	IEWLCENT	9,6	DC,DD	ENTRY statement processing
IEWLCEOD	Entry Point	IEWLEINP	IEWLEINP	4,3	--	EOD for SYSLIB; also entered when ECM record read from load module
IEWLCESD	CSECT	IEWLCESD	IEWLCESD	12,7	CE,CF CG	ESD record processing
IEWLCFAB	Entry Point	IEWLCFNL	IEWLCFNL	10,8	--	Termination processing
IEWLCFNL	CSECT	IEWLCFNL	IEWLCFNL	10,8	GA	Final processing
IEWLCINC	CSECT	IEWLCINC	IEWLCINC	8,4	CO	Include processing
IEWLCLDB	CSECT	IEWLCLDB	IEWLCLDB	2,1	--	SYSLIB DCB
IEWLCMAP	CSECT	IEWLCMAP	IEWLCMAP	11,8	GC	Module map processing
IEWLCPTH	Entry Point	IEWLCRCG	IEWLCRCG	7,3	CG	Find common segment in overlay path
IEWLCRCG	CSECT	IEWLCRCG	IEWLCRCG	7,3	--	Replace/change processing
IEWLCSCN	CSECT	IEWLCSCN	IEWLCSCN	8,5	CI,CM	Control statement scan
IEWLCSDB	Symbol	IEWLEAPT	IEWLEAPT	1,1	--	SYSLIN DCB
IEWLCSNX	Entry Point	IEWLCFNL	IEWLCFNL	10,8	--	Synchronous file error exit
IEWLCSYM	CSECT	IEWLCSYM	IEWLCSYM	7,3	CD	SYM processing
IEWLEADA	CSECT	IEWLEADA	IEWLEADA	9,6	DA	Address assignment processing
IEWLEAPT	CSECT	IEWLEAPT	IEWLEAPT	1,1	--	All purpose Table
IEWLEEON	Entry Point	IEWLEINP	IEWLEINP	4,3	--	EOD for SYSPRINT
IEWLEINP	CSECT	IEWLEINP	IEWLEINP	4,3	CA	Input processing
IEWLEINT	CSECT	IEWLEINT	IEWLEINT	3,2	BA	Initial processing
IEWLELOG	CSECT	IEWLELOG	IEWLELOG	2,1	GB	Error logging
IEWLEMDI	CSECT	IEWLEINP	IEWLEMDI	4,3	--	Module input
IEWLEOPT	CSECT	IEWLEOPT	IEWLEOPT	3,2	BA	Attributes and options processing

(Continued)

Microfiche Directory (Continued)

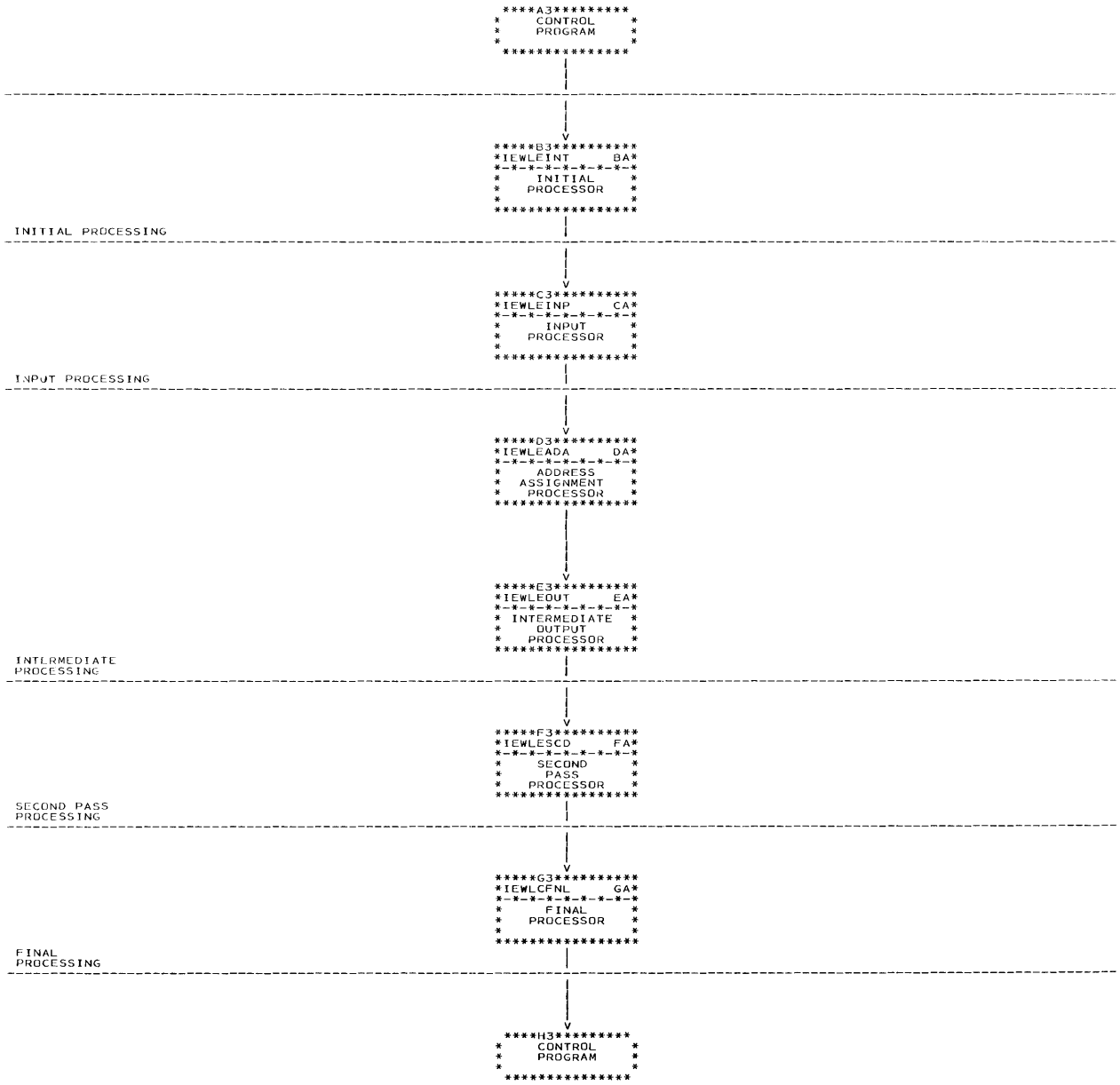
Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment (15K,18K)	Chart ID	Synopsis
IEWLEOUT	CSECT	IEWLEOUT	IEWLEOUT	9,6	EA	Intermediate output processing
IEWLERAT	CSECT	IEWLETXR	IEWLERAT	6,3	CH,CI CJ	TXT and RLD processing
IEWLERDM	Entry Point	IEWLEINP	IEWLEINP	4,3	--	Read routine
IEWLEROU	CSECT	IEWLEROU	IEWLEROU	1,1	--	Linkage editor E entry point
IEWLESCD	CSECT	IEWLESCD	IEWLESCD	12,7	FA,FB	Second pass processing
LABEL	Label Routine	IEWLCESD	IEWLCESD	12,7	CF	Renumber ID field of LABEL item
LIBOP	Library Open Routine	IEWLCINC	IEWLCINC	8,4	CQ	Opens libraries
NXTLINE	Next Line Routine	IEWLCESD	IEWLCESD	12,7	CE	Set pointer to next line in CESD
Relocatable Constant Table	Table	IEWLEADA	IEWLEADA	9,6	--	Relocation constants
RENUMBER	Renumber Routine	IEWLCESD	IEWLCESD	12,7	CF	Translate ESD ID to CESD ID
Renumbering Table	Table	IEWLCESD	IEWLCESD	12,7	--	ESD - CESD item resolution
RLDBUF	RLD Buffer Routine	IEWLETXR	IEWLERAT	6,3	CH,CI	Write RLDs to SYSUT1
RLD Note List	Table	IEWLETXR	IEWLERAT	6,3	--	Description, location of RLDs on SYSUT1
Scatter Table	Table	IEWLEOUT	IEWLEOUT	9,6	--	Ordered symbol addresses
SCDCUTLD	Split ADCON Routine	IEWLESCD	IEWLESCD	12,7	FC	Relocate split ADCONs
SCDRELOC	Relocation Routine	IEWLESCD	IEWLESCD	12,7	FC,FD FE	Relocate address constants
SEGLGTH Table	Table	IEWLEADA	IEWLEADA	9,6	--	Segment lengths
SEGTAB	Table	IEWLEOUT	IEWLEOUT	9,6	--	Segment relationships

(Continued)

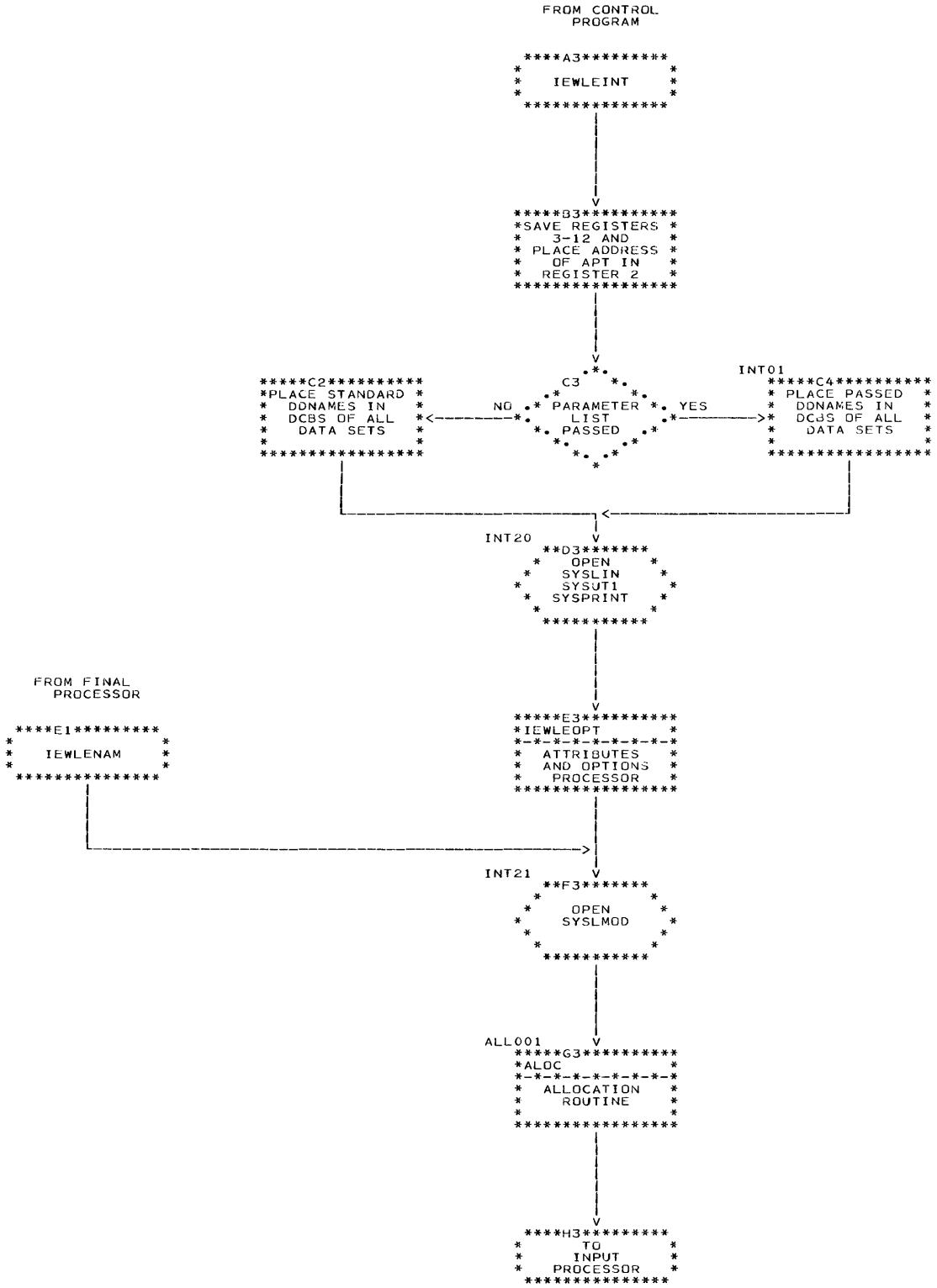
Microfiche Directory (Continued)

Name	Description	Object Module Name (Microfiche Name)	CSECT Name	Overlay Segment (15K,18K)	Chart ID	Synopsis
SYSLMOD	Symbol	IEWLEAPT	IEWLEAPT	1,1	--	SYSLMOD DCB
SYSPRINT	Symbol	IEWLEAPT	IEWLEAPT	1,1	--	SYSPRINT DCB
SYSUT1	Symbol	IEWLEAPT	IEWLEAPT	1,1	--	SYSUT1 DCB
Text I/O Table	Table	IEWLETXR	IEWLERAT	6,3	--	Description of text on SYSUT1
Text Note List	Table	IEWLETXR	IEWLERAT	6,3	--	Location of text on SYSUT1
Translation Table	Table	IEWLEOUT	IEWLEOUT	9,6	--	Pointers to Scatter Table entries
TTR List	Table	IEWLETXR	IEWLERAT	6,3	--	Address of first text in each segment
TXTBUF	TXT Buffer Routine	IEWLETXR	IEWLERAT	6,3	CH,CJ	Write TXT to SYSUT1

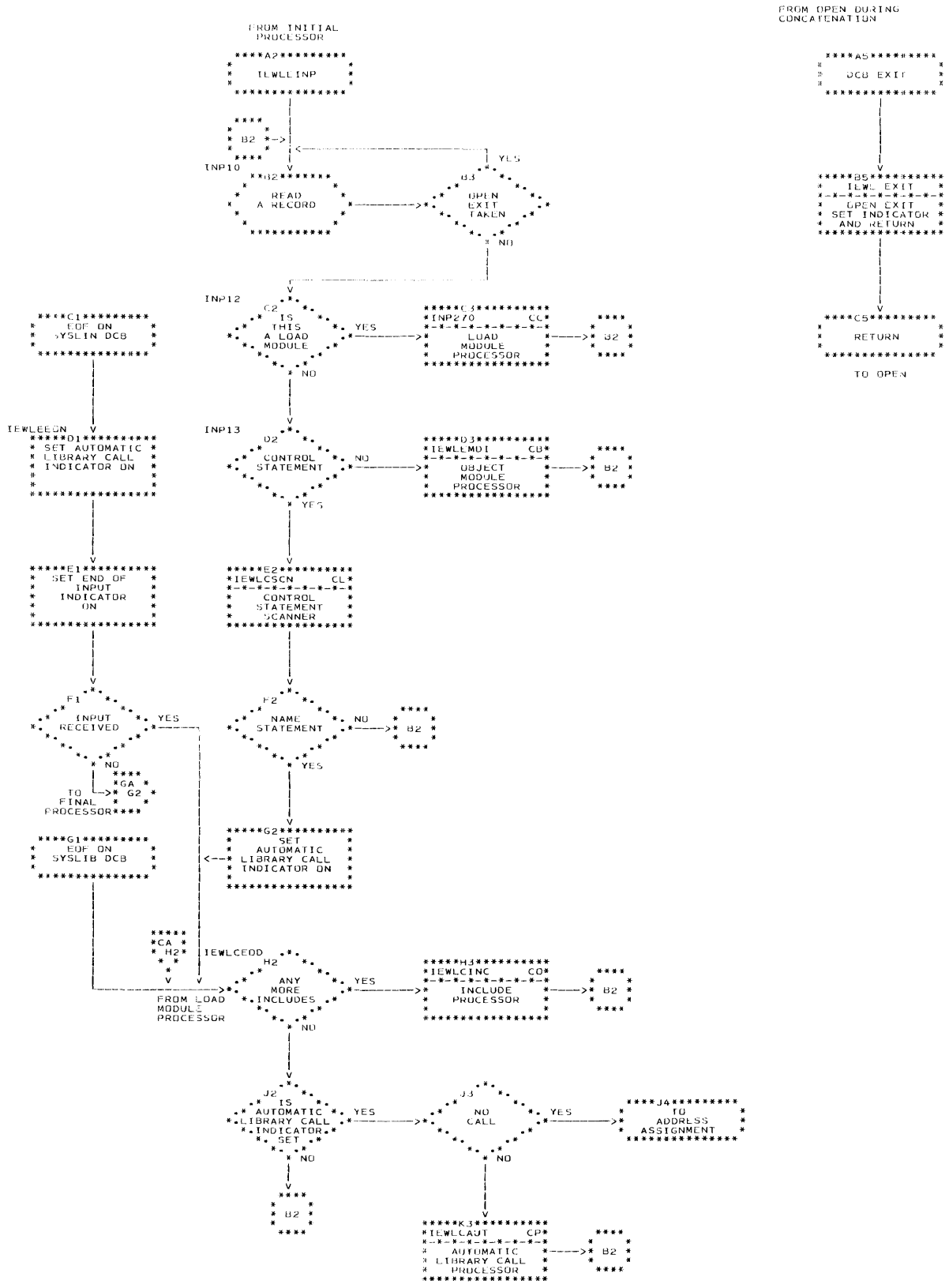
Chart AA. Major Divisions



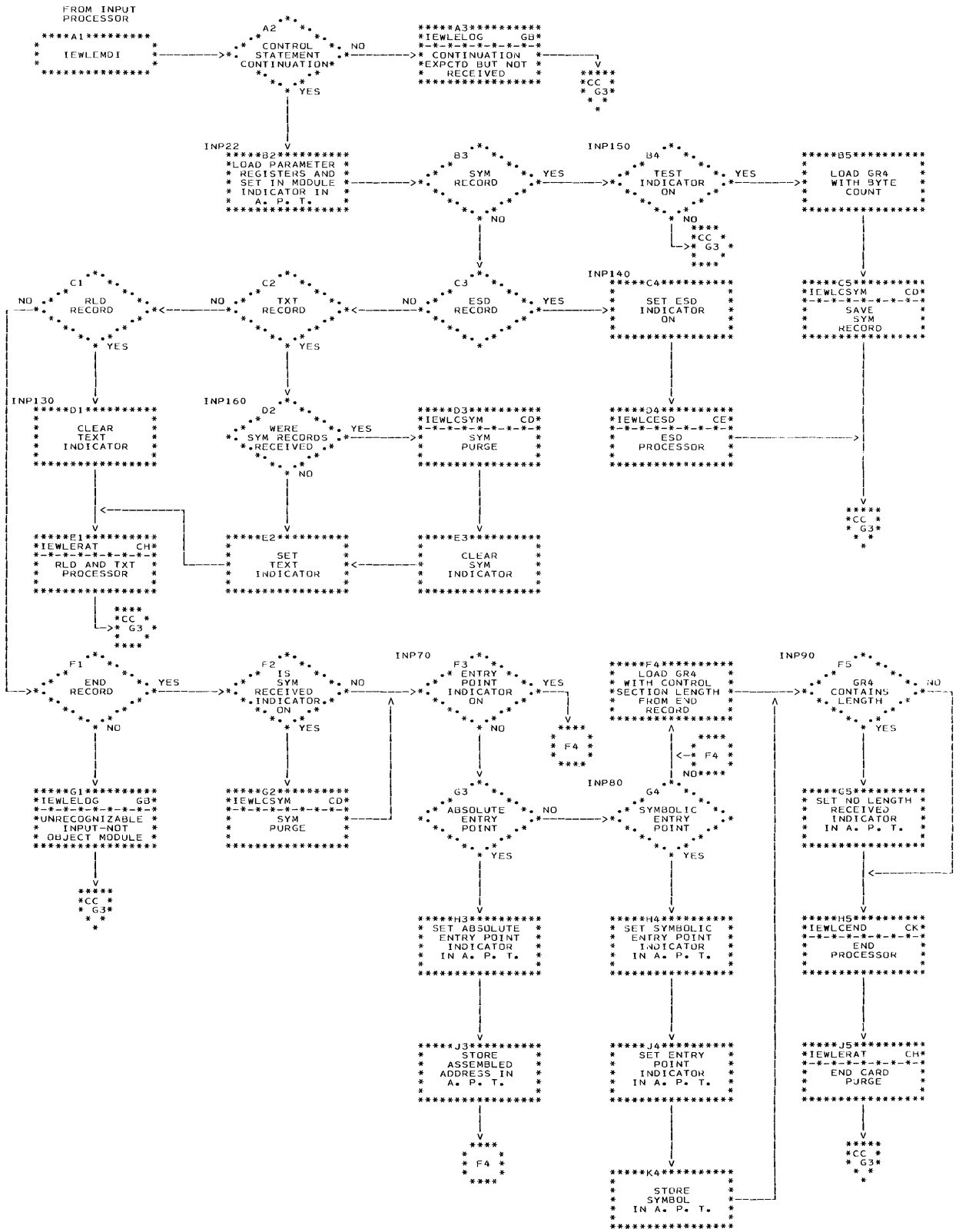
• Chart BA. Initial Processor (IEWLEINT)



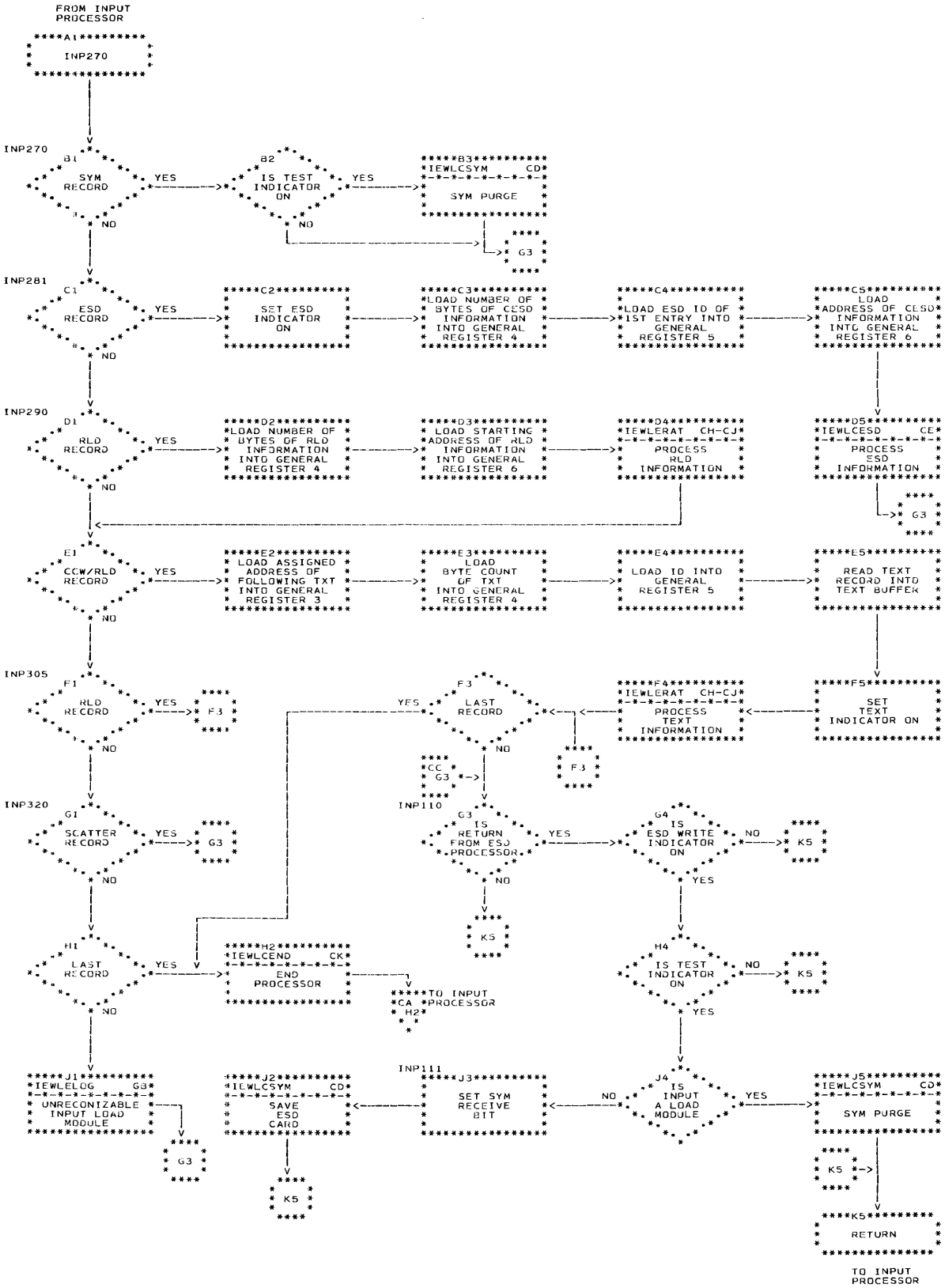
• Chart CA. Input Processor (IEWLEINP)



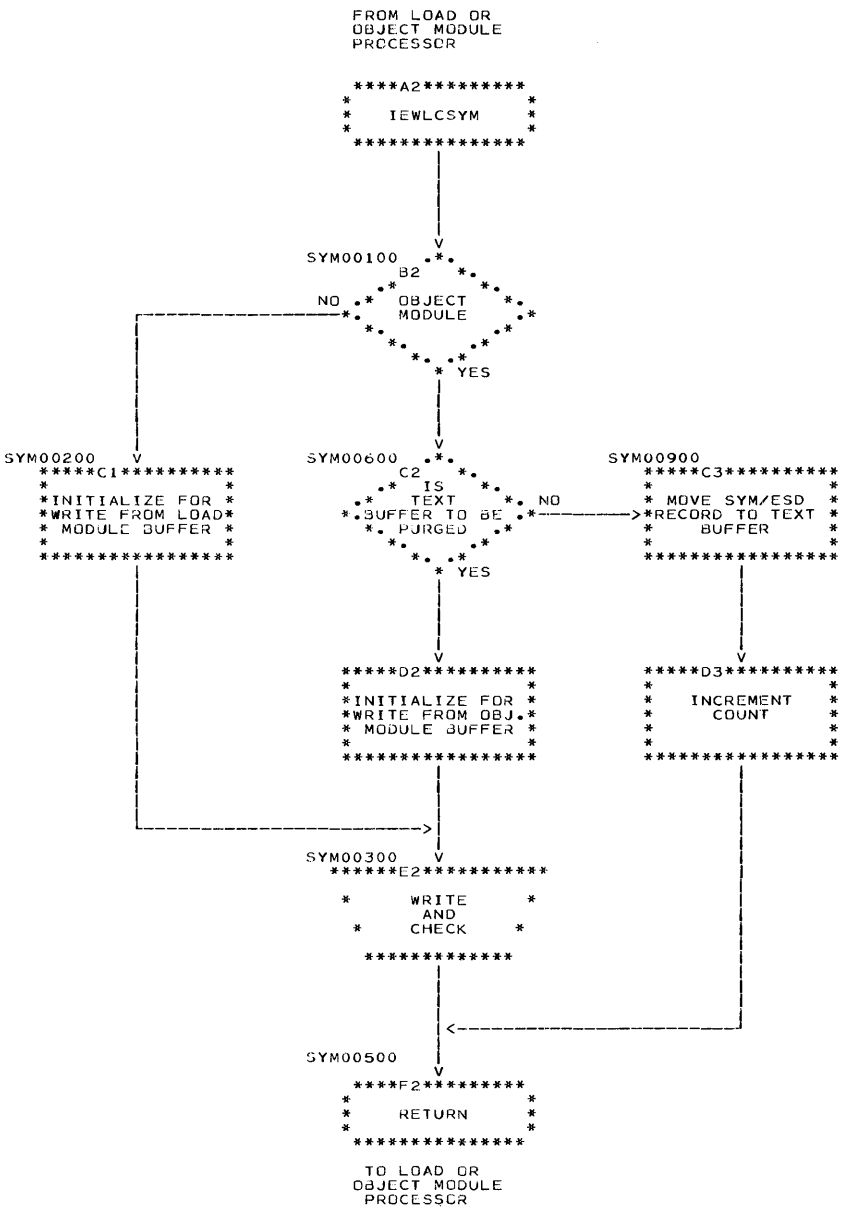
• Chart CB. Object Module Processor (IEWLEMDI)



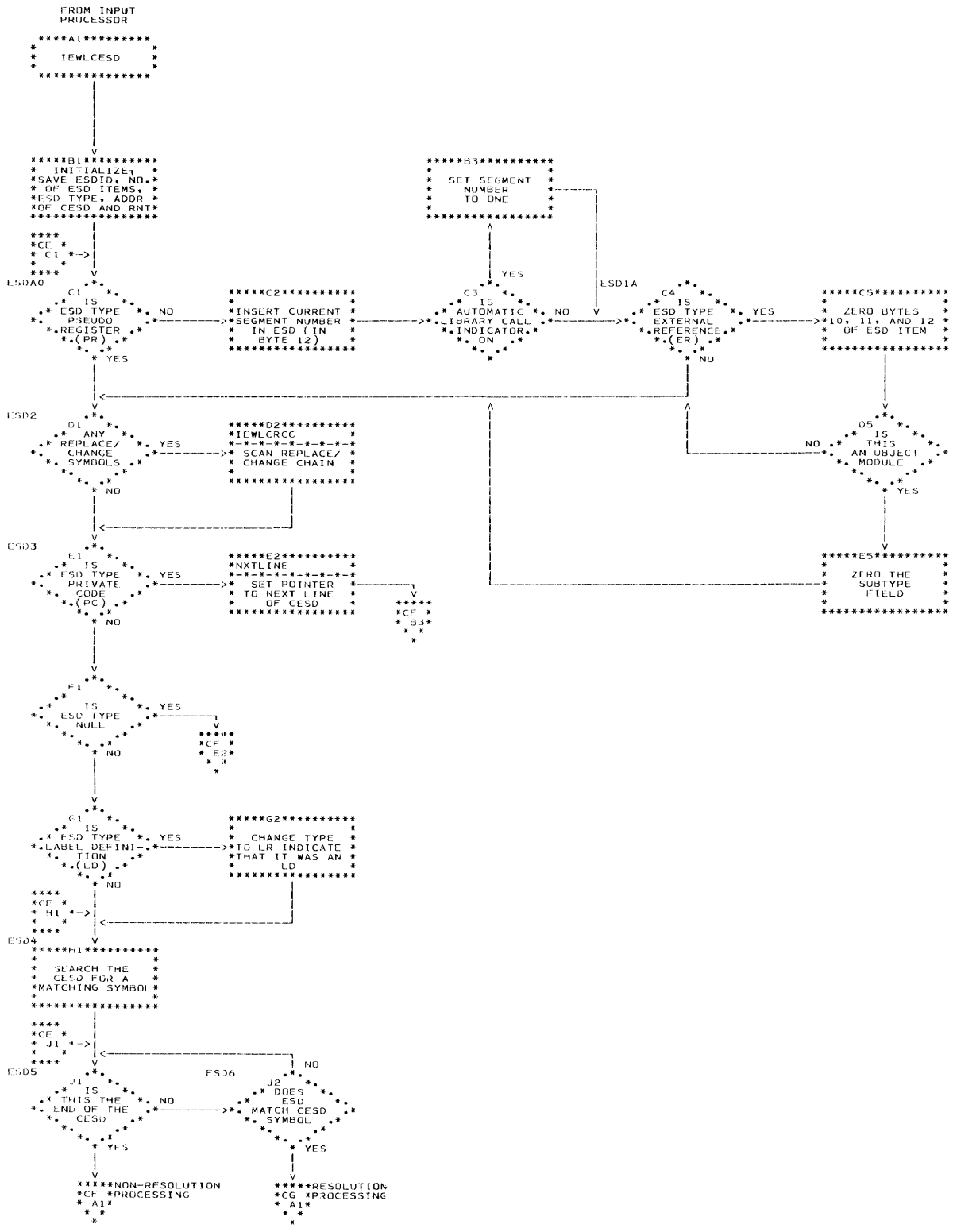
• Chart CC. Load Module Processor (INP270)



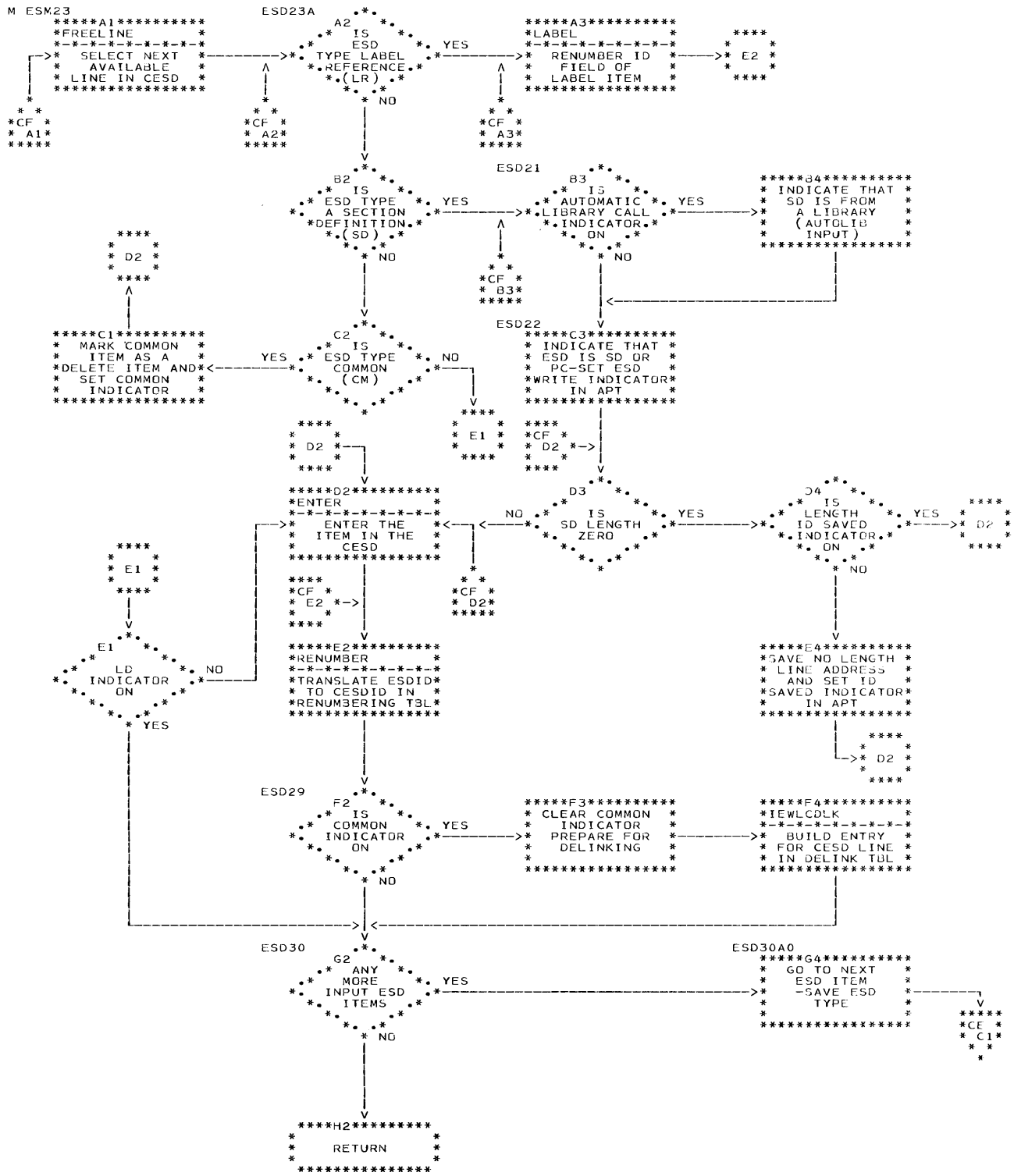
• Chart CD. SYM Processor (IEWLCSYM)



• Chart CE. ESD Processor (IEWLCESD)



• Chart CF. ESD Processor (IEWLCESD) (Continued)



• Chart CG. ESD Processor (IEWLCESD) (Continued)

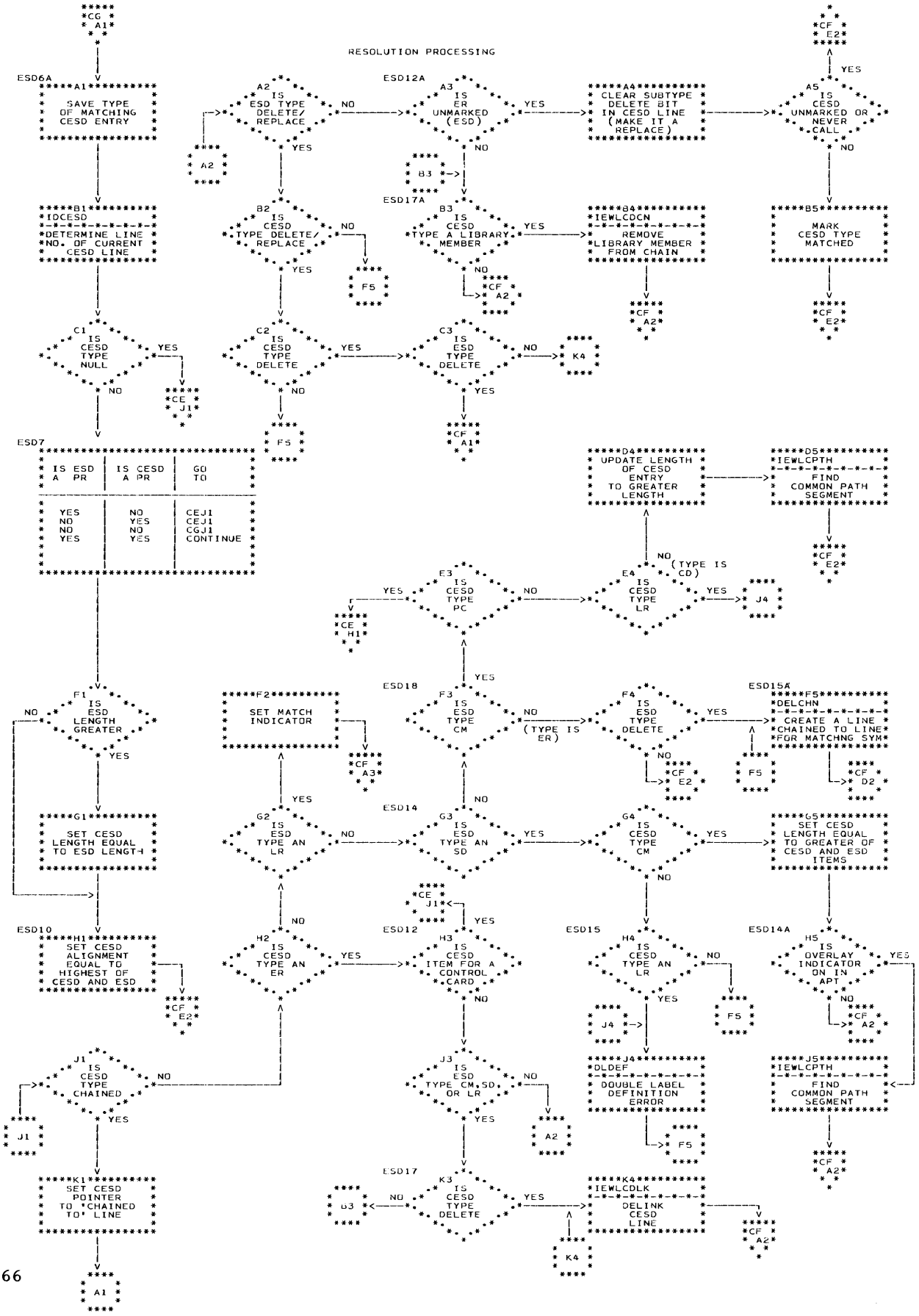


Chart CH. TXT and RLD Processor (IEWLERAT)

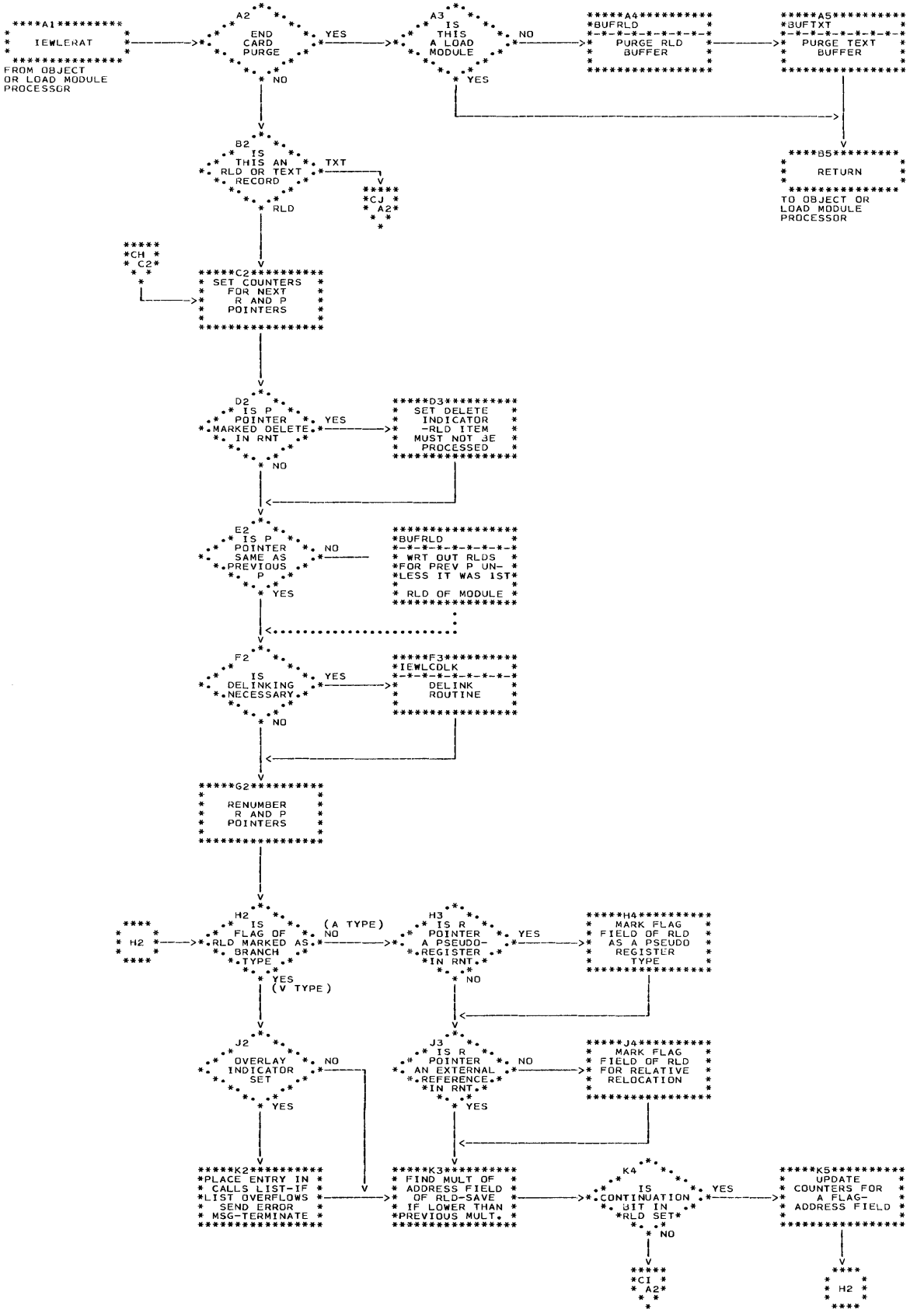


Chart CI. TXT and RLD Processor (IEWLERAT) (Continued)

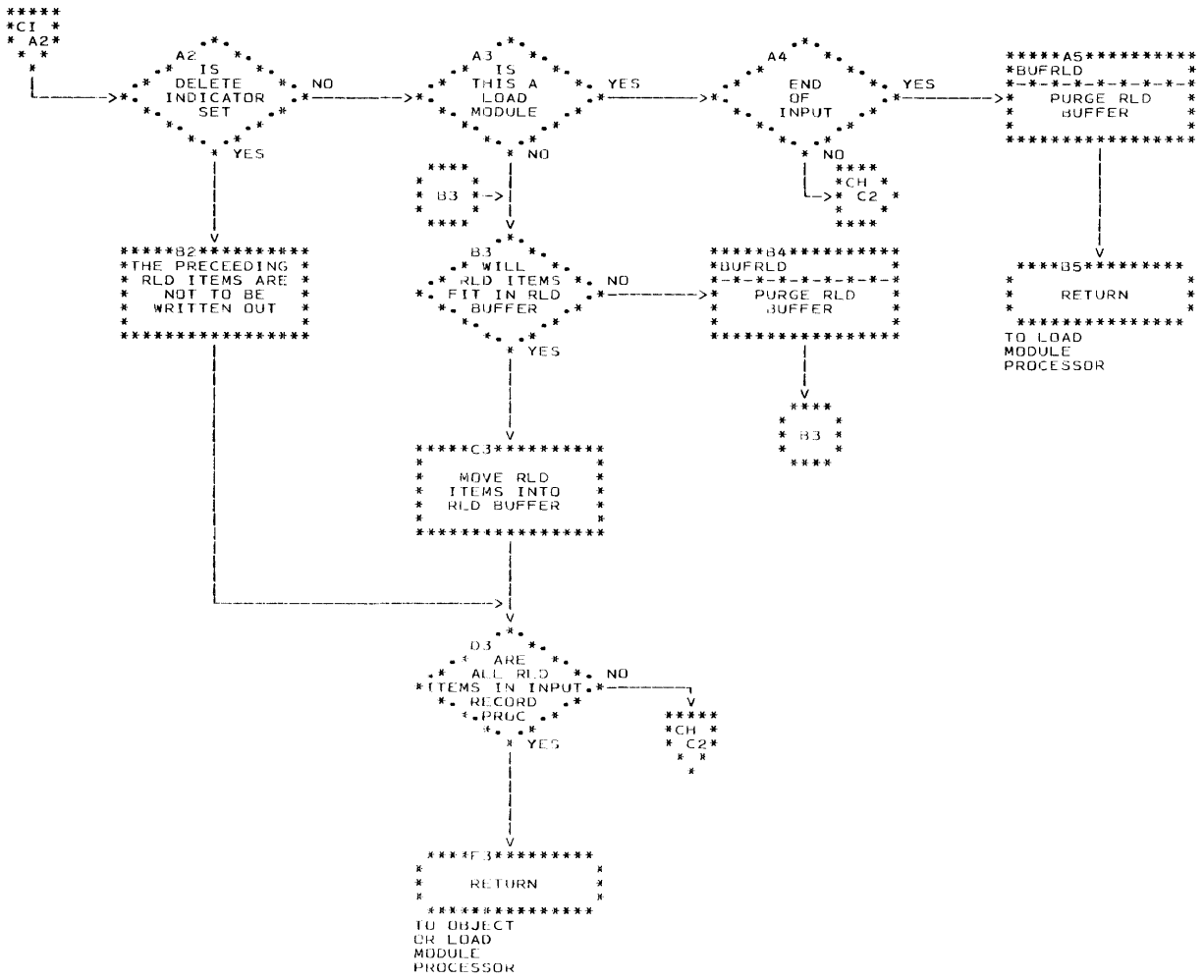
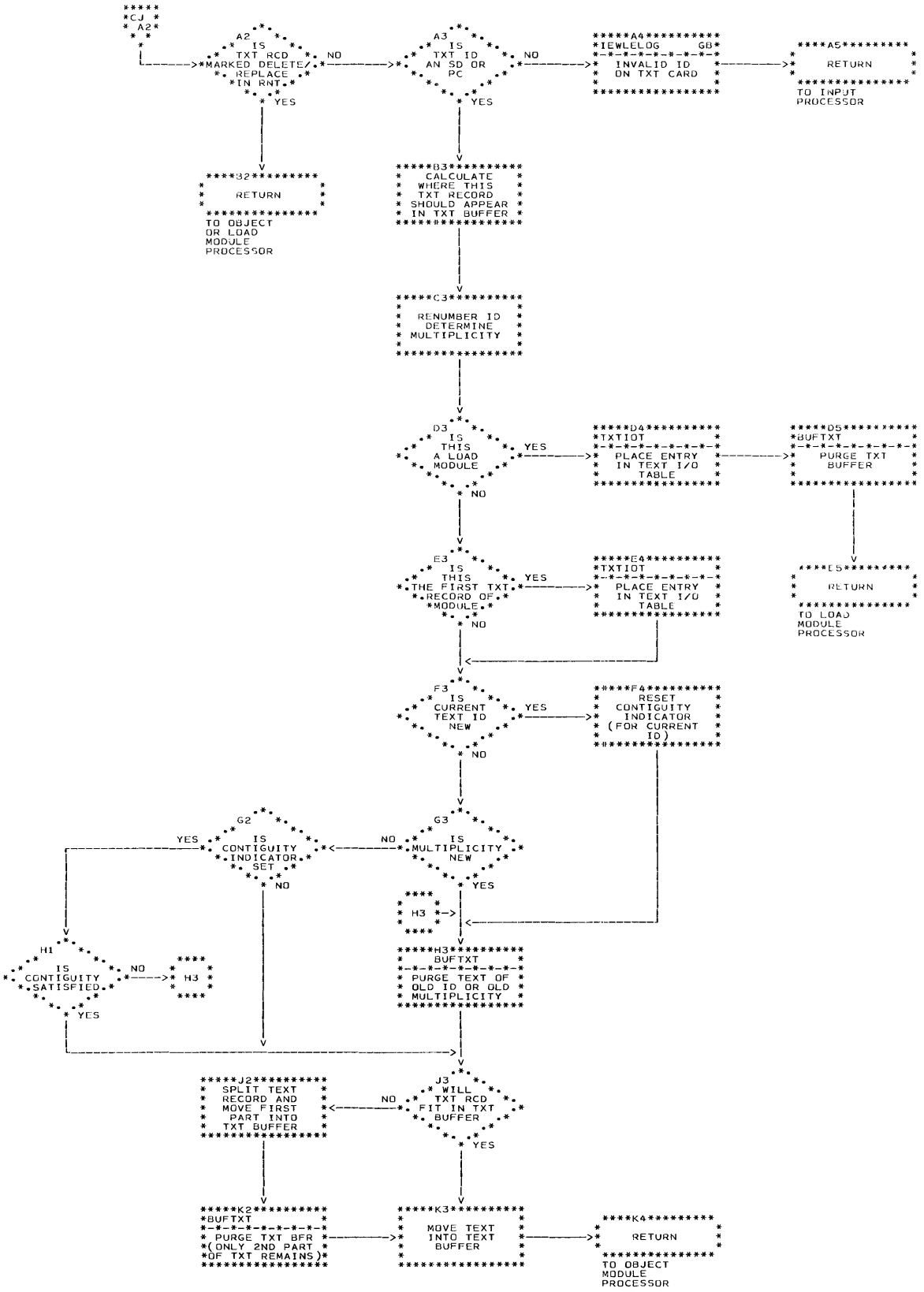
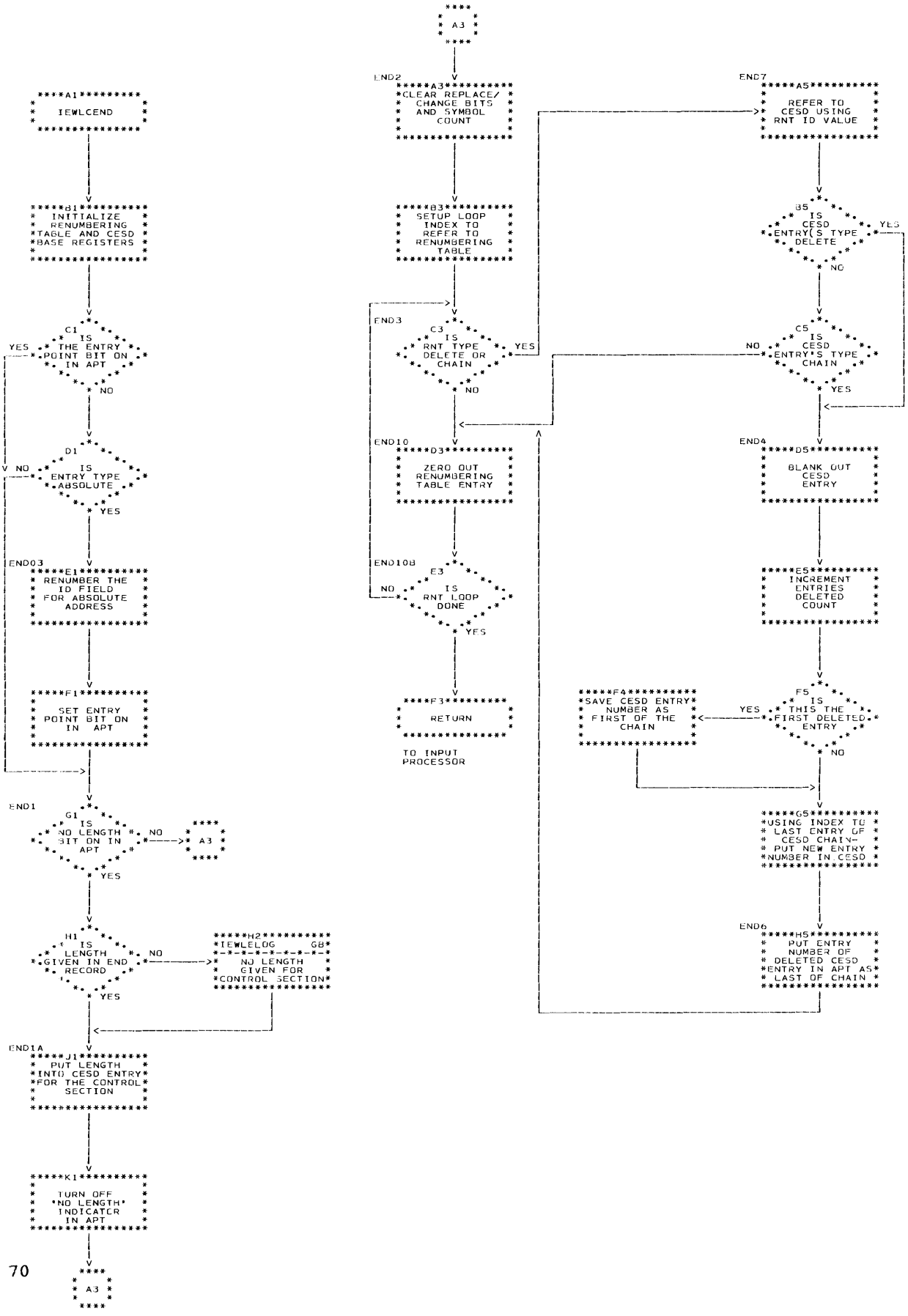


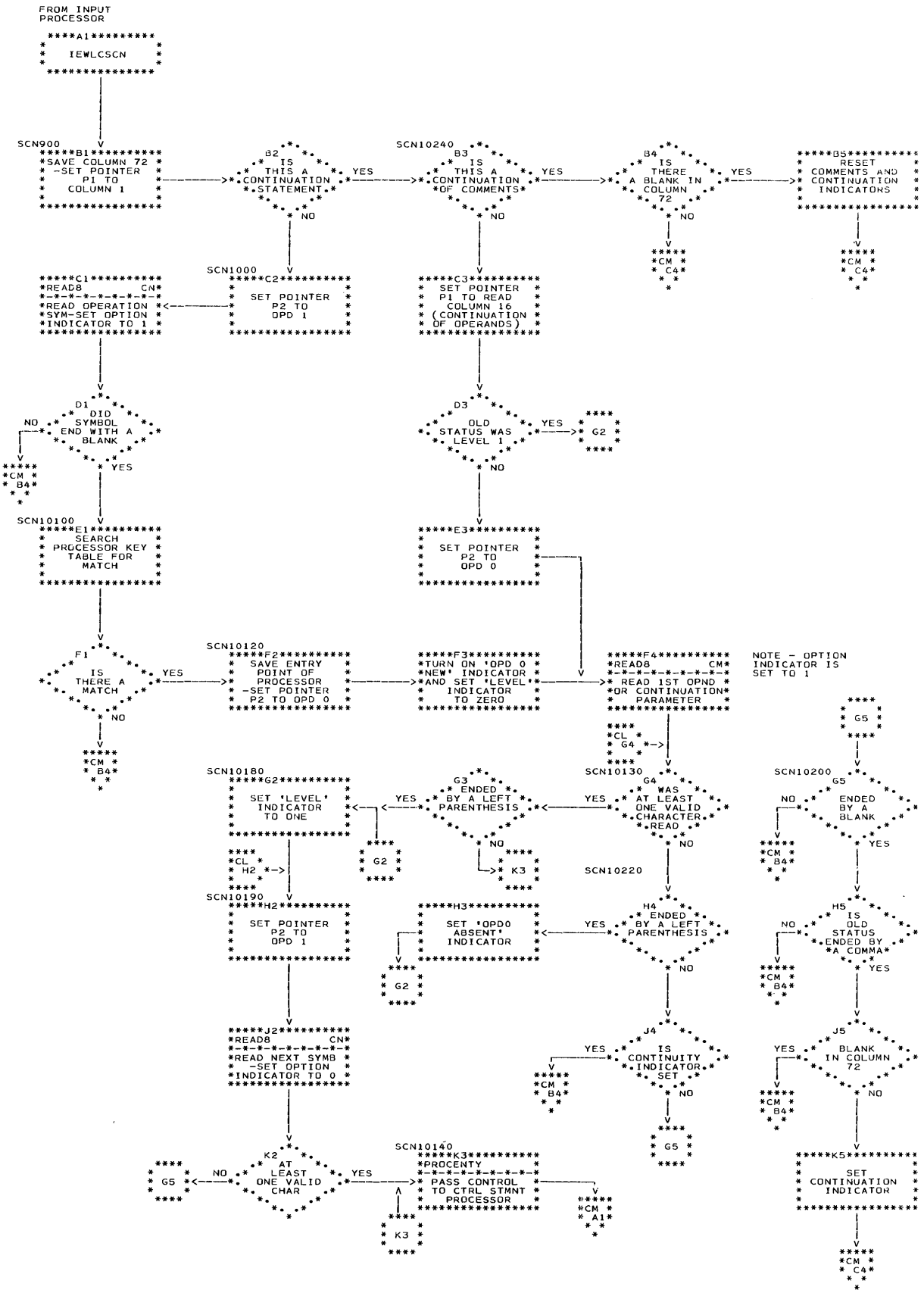
Chart CJ. TXT and RLD Processor (IEWLERAT) (Continued)



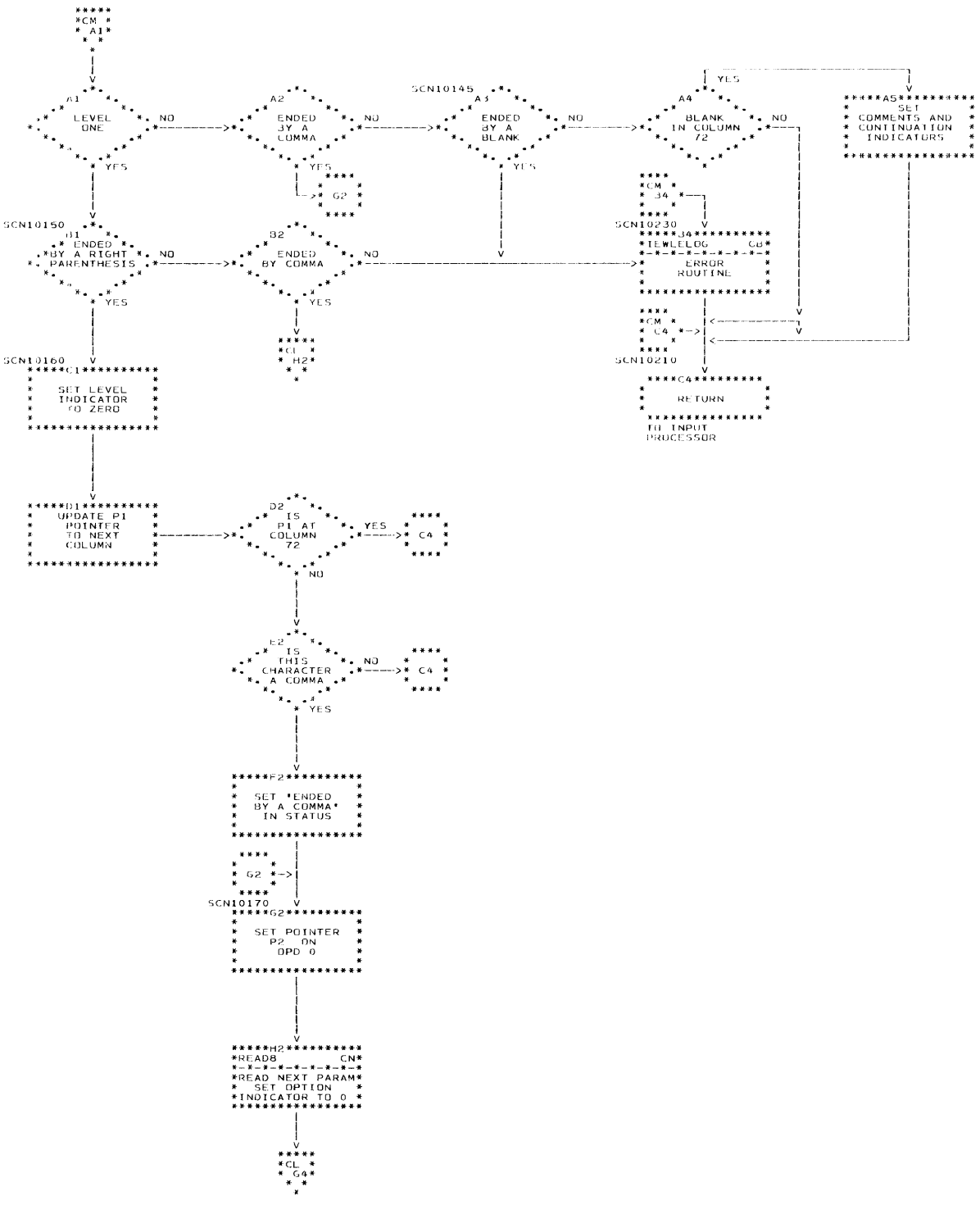
• Chart CK. END Processor (IEWLCEND)



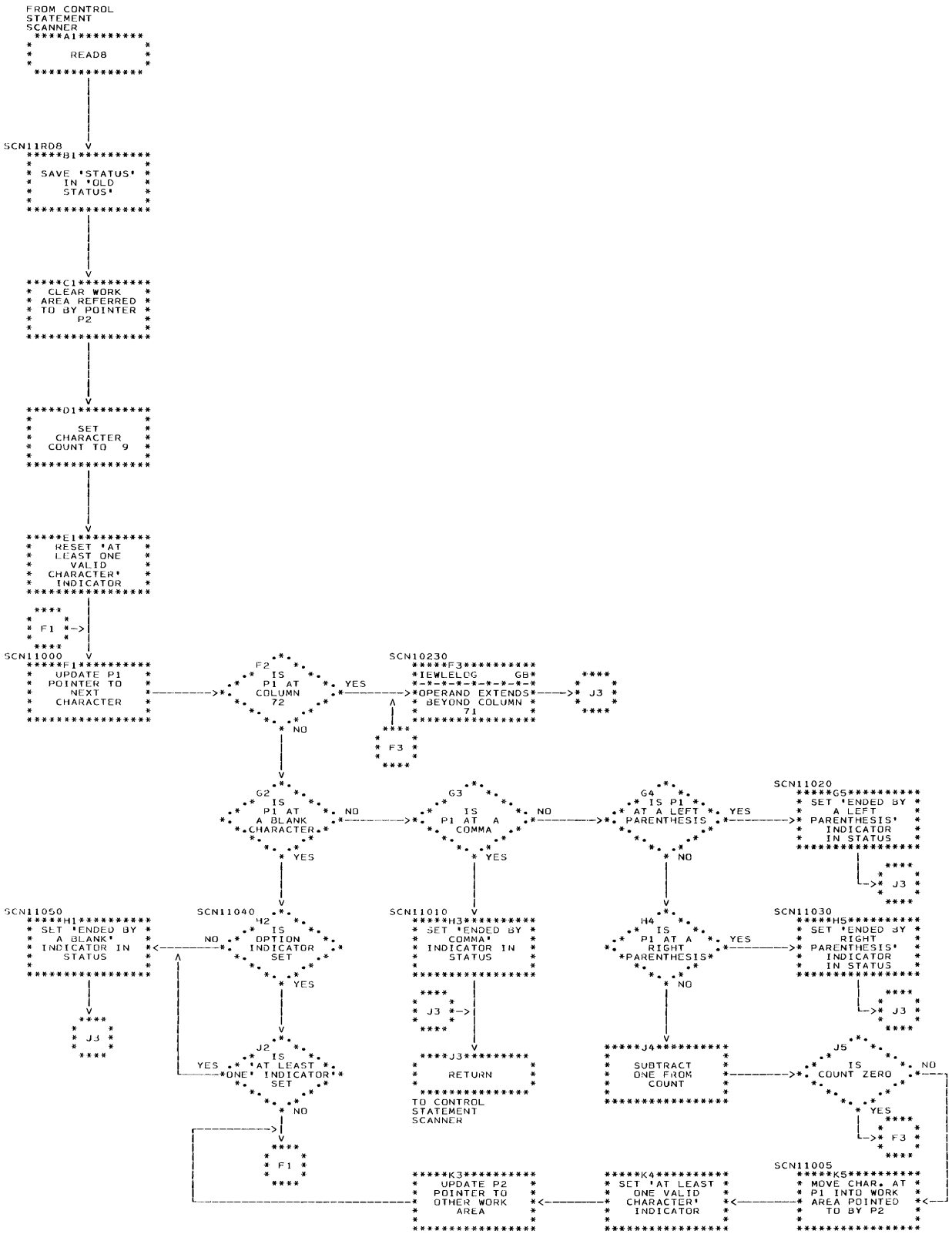
• Chart CL. Control Statement Scanner (IEWLCSCN)



• Chart CM. Control Statement Scanner (IEWLCSCN) (Continued)



• Chart CN. Read 8 Routine



• Chart CO. Include Processor (IEWLCINC)

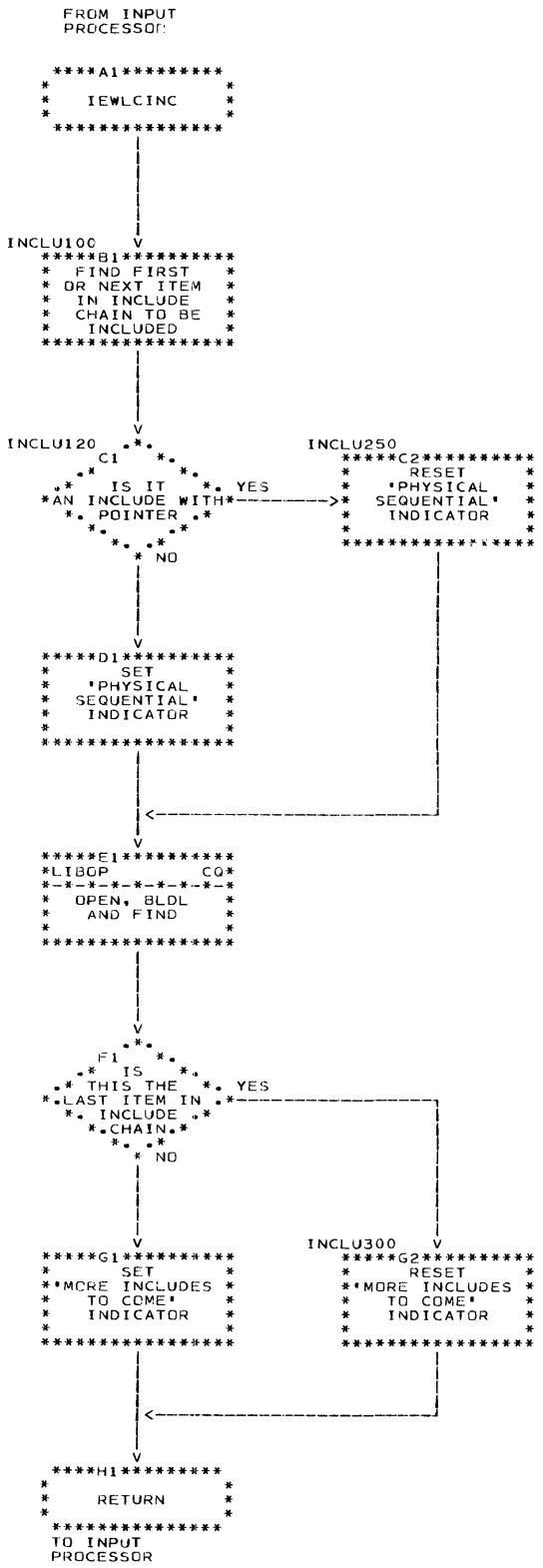
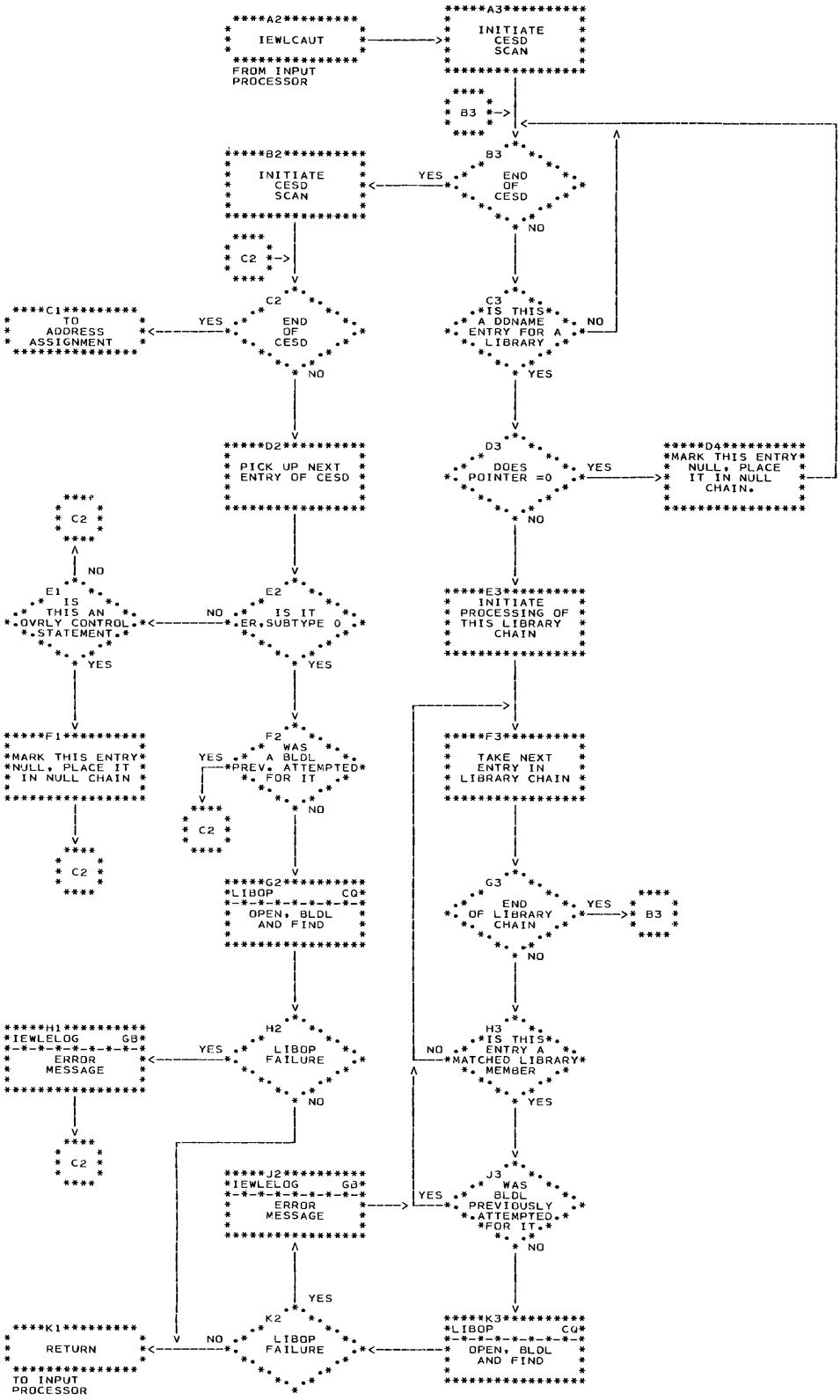
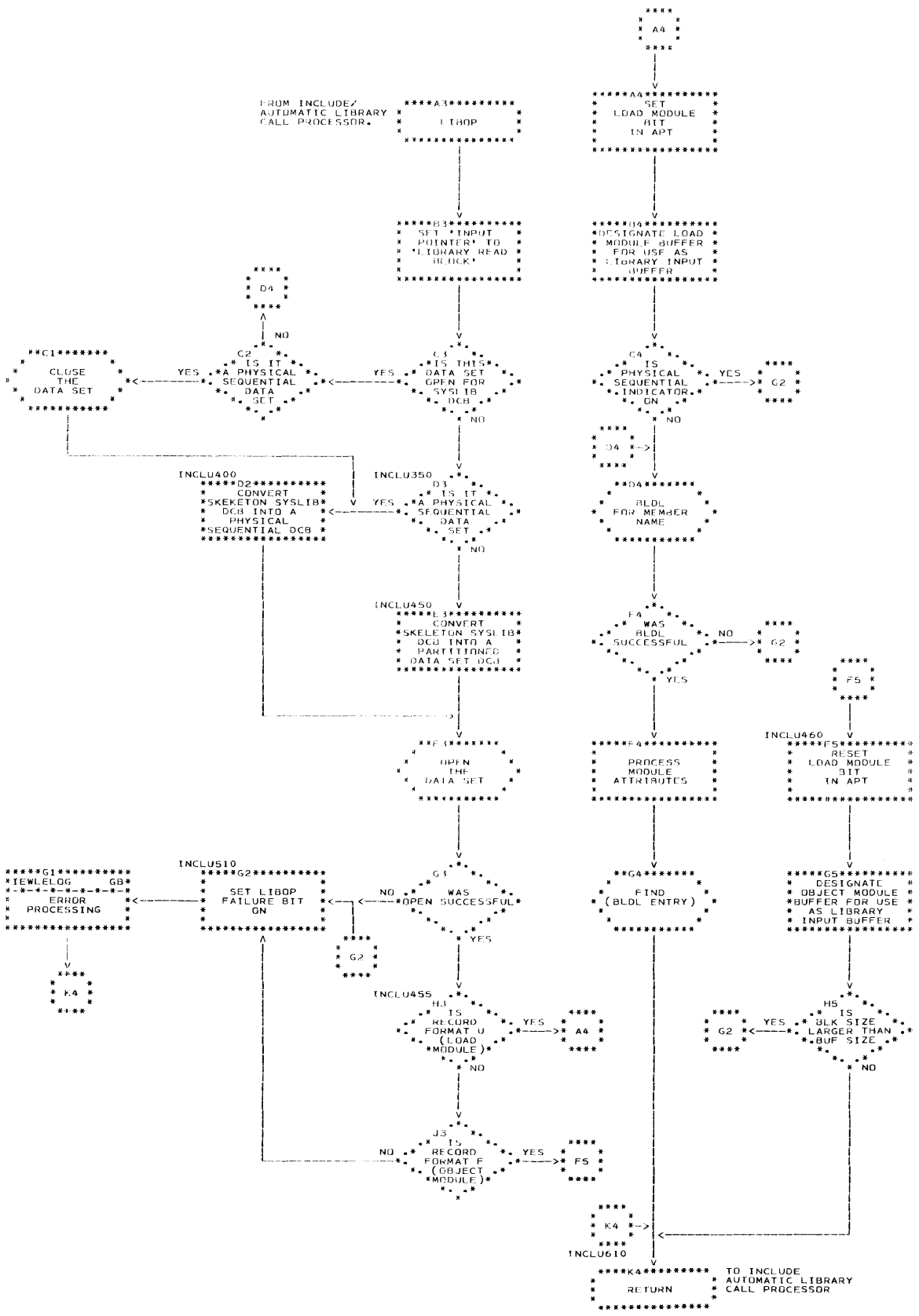


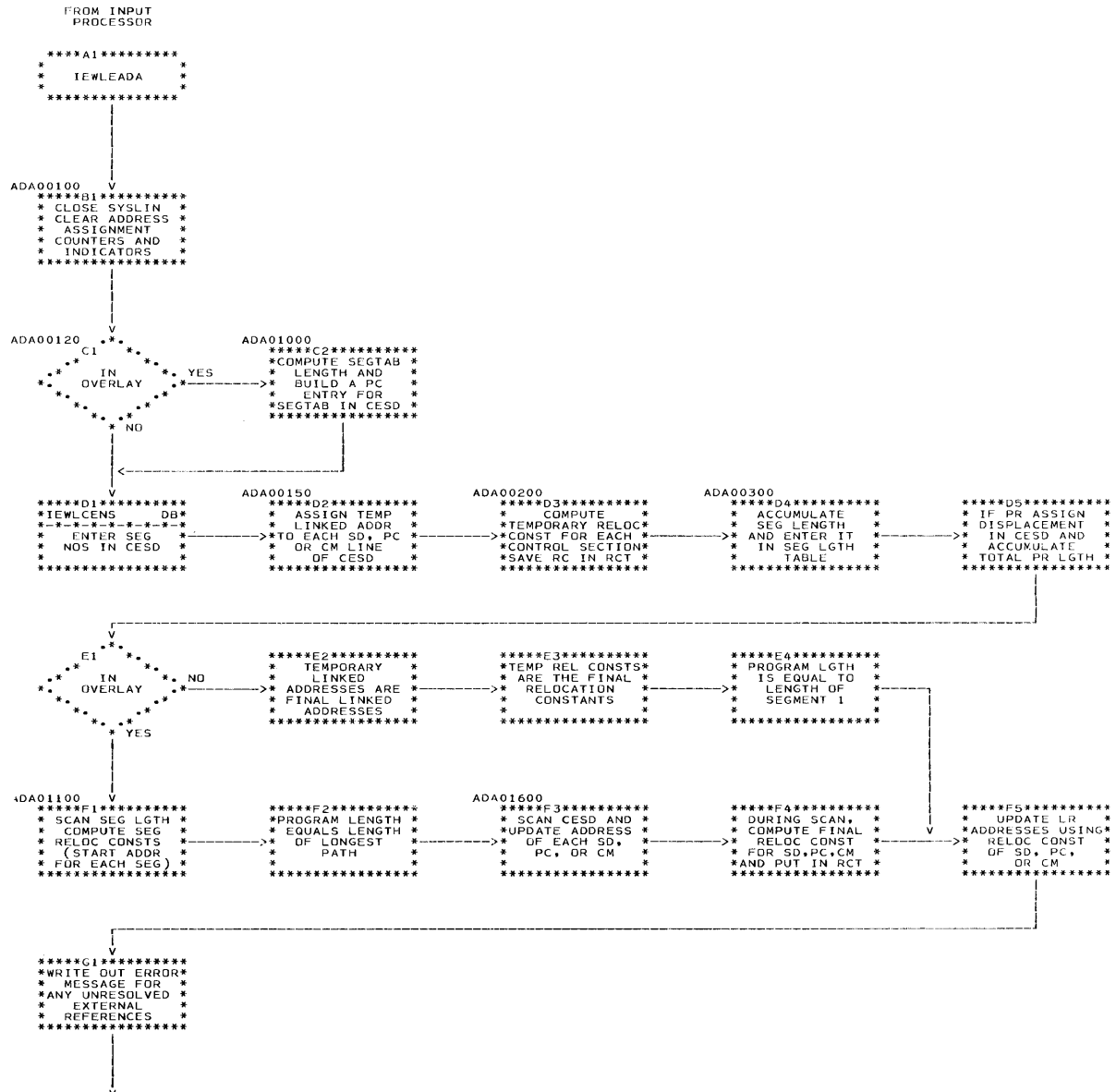
Chart CP. Automatic Library Call Processor (IEWLCAUT)



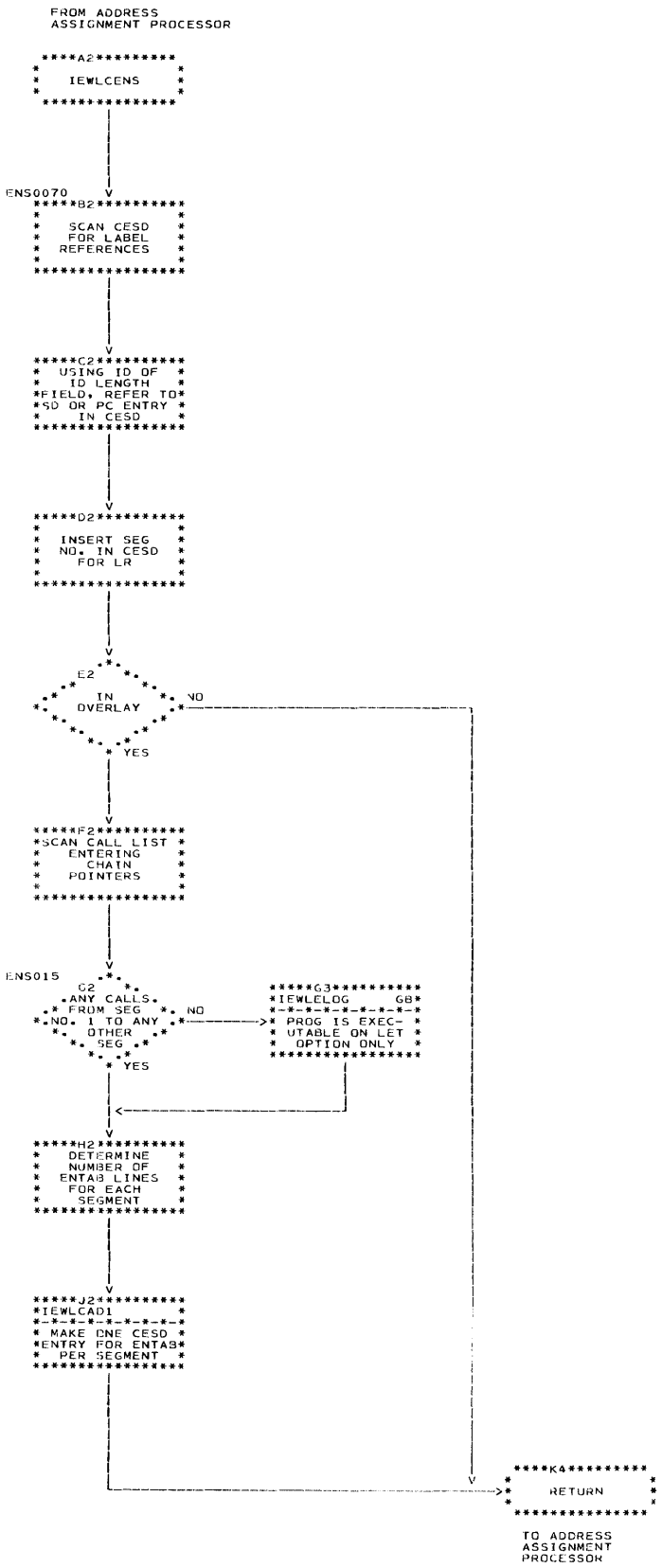
• Chart CQ. Library Open Routine (LIBOP)



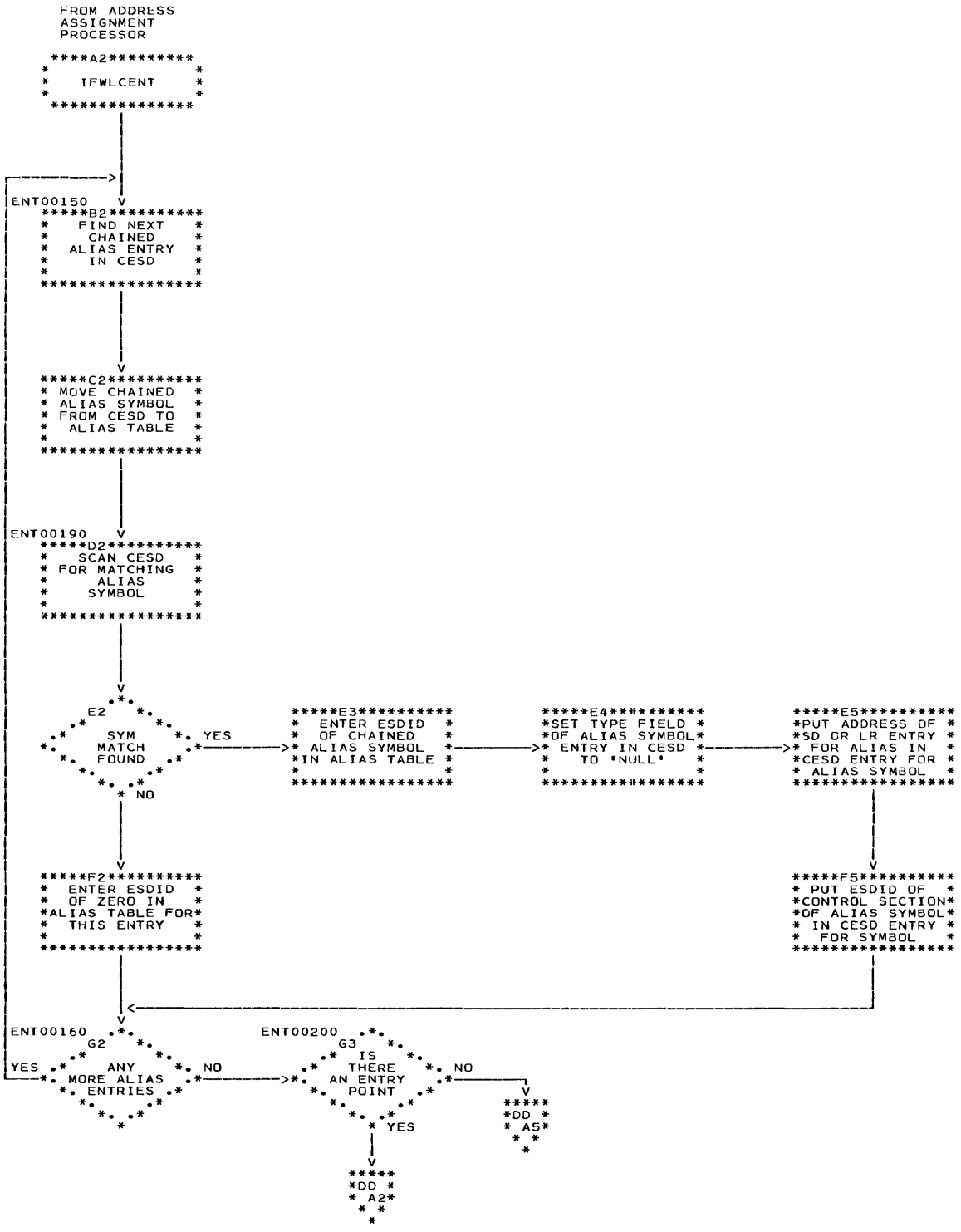
•Chart DA. Address Assignment Processor (IEWLEADA)



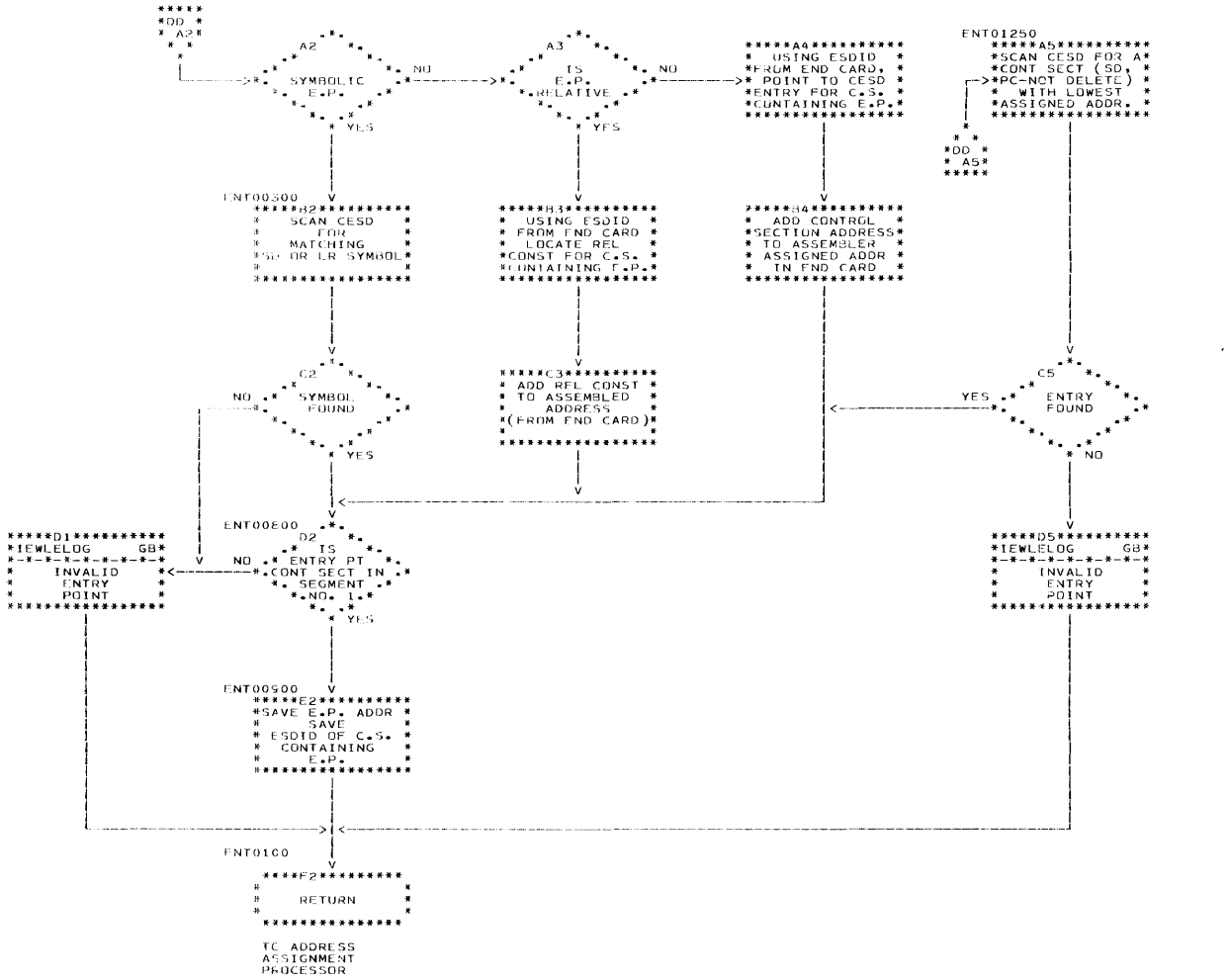
•Chart DB. IEWLCENS Routine



• Chart DC. Entry Processor (IEWLCENT)



● Chart DD. Entry Processor (IEWLCENT) (Continued)



•Chart EA. Intermediate Output Processor (IEWLEOUT)

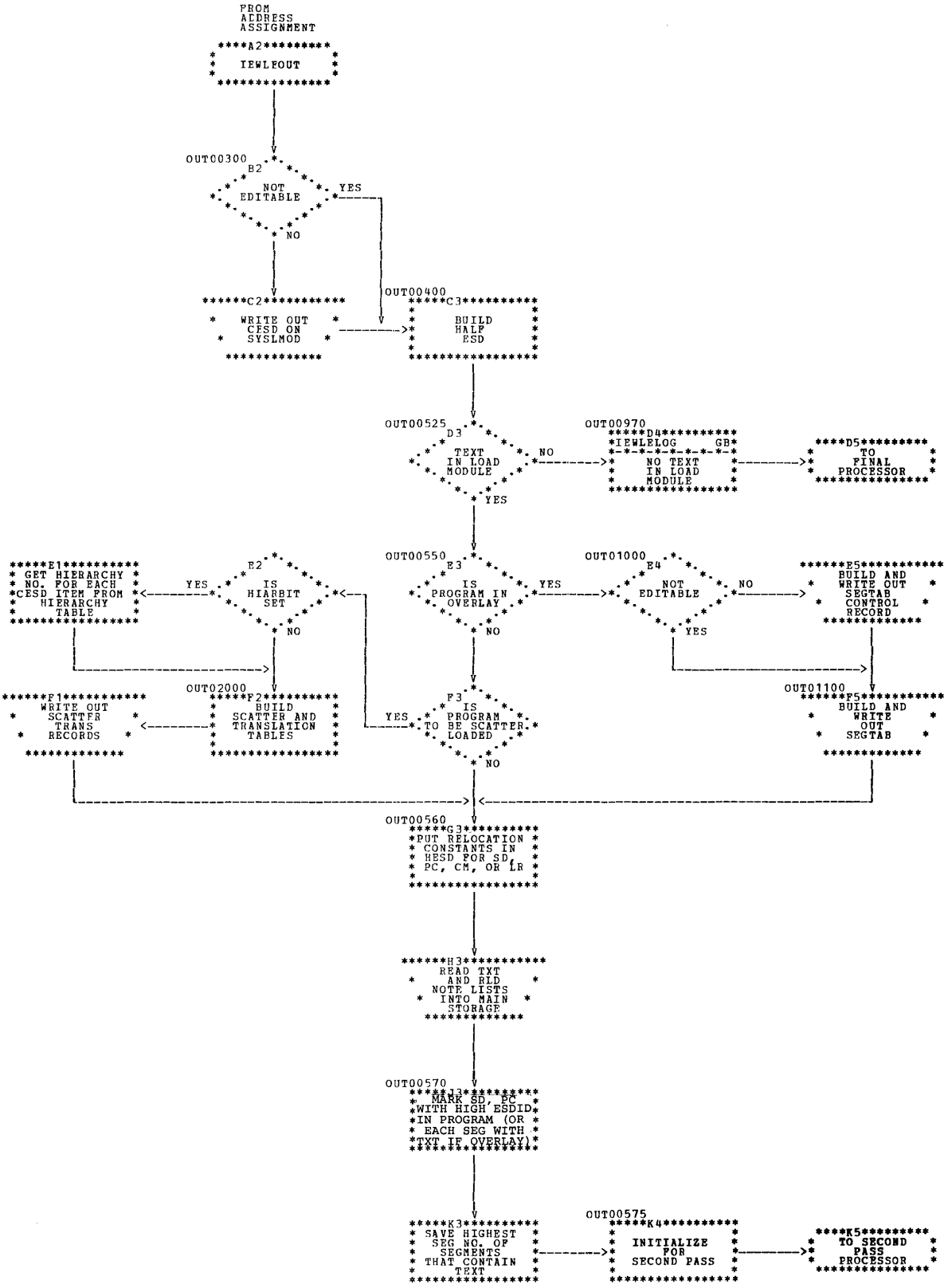


Chart FA. Second Pass Processor (IEWLESCD)

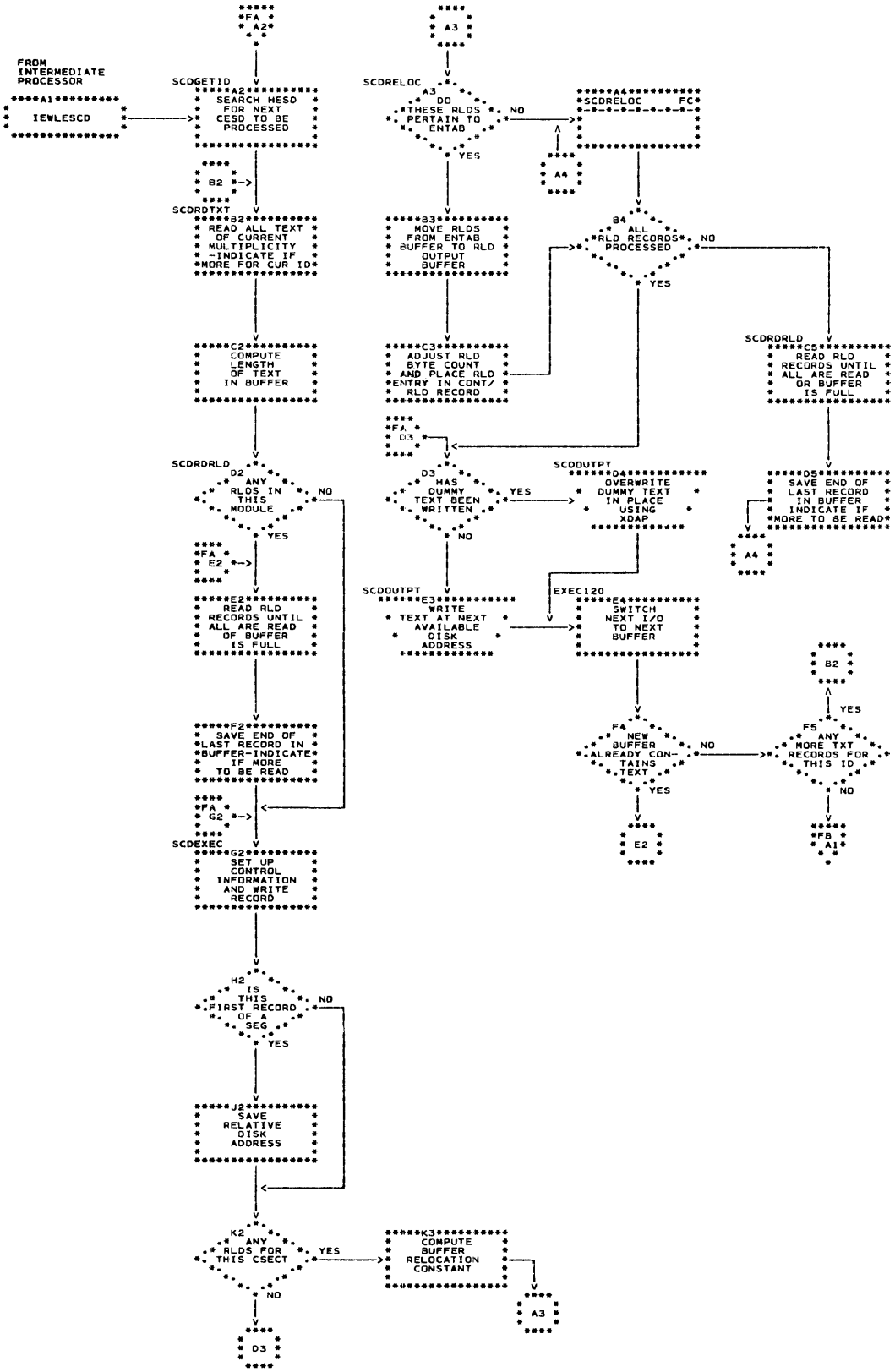
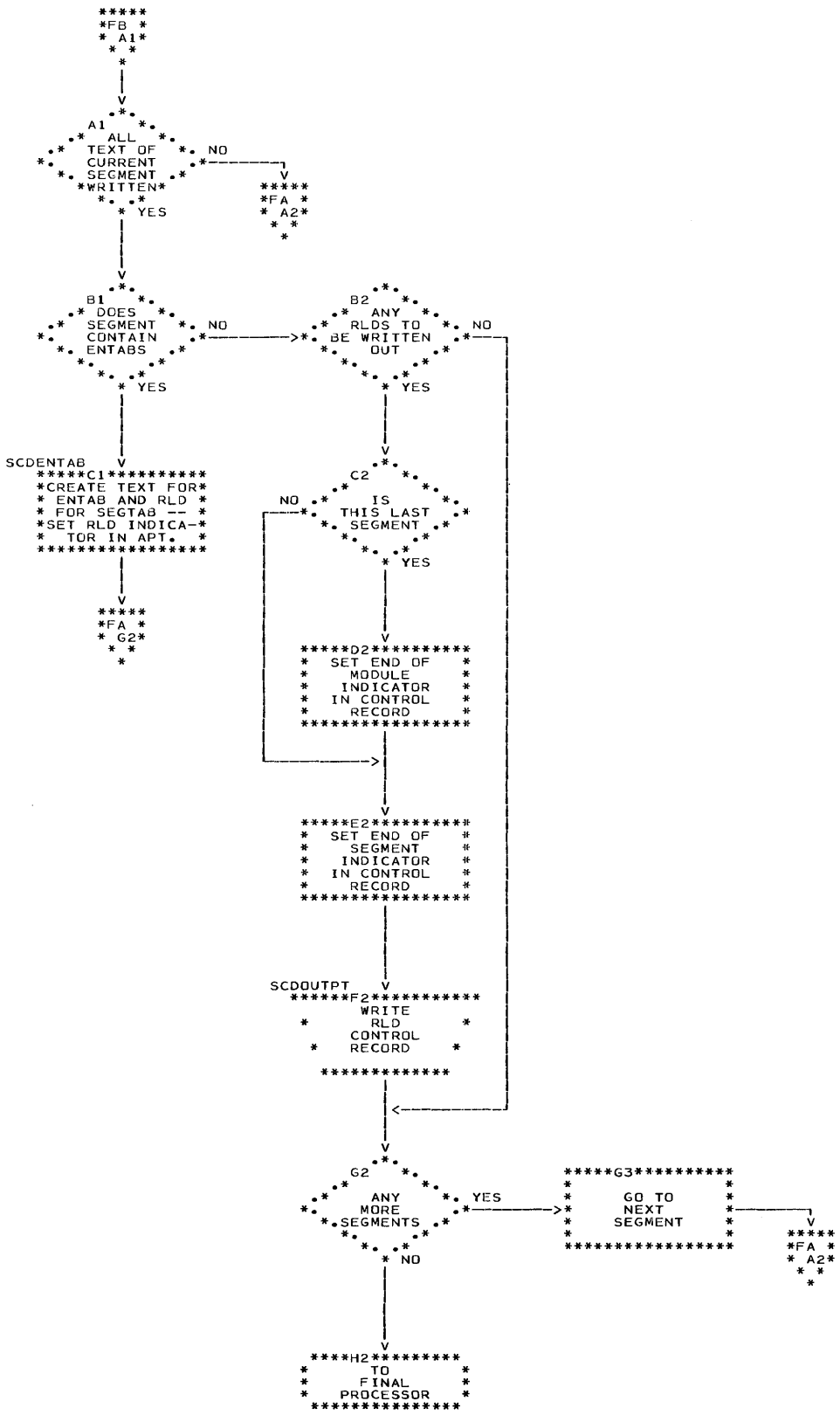
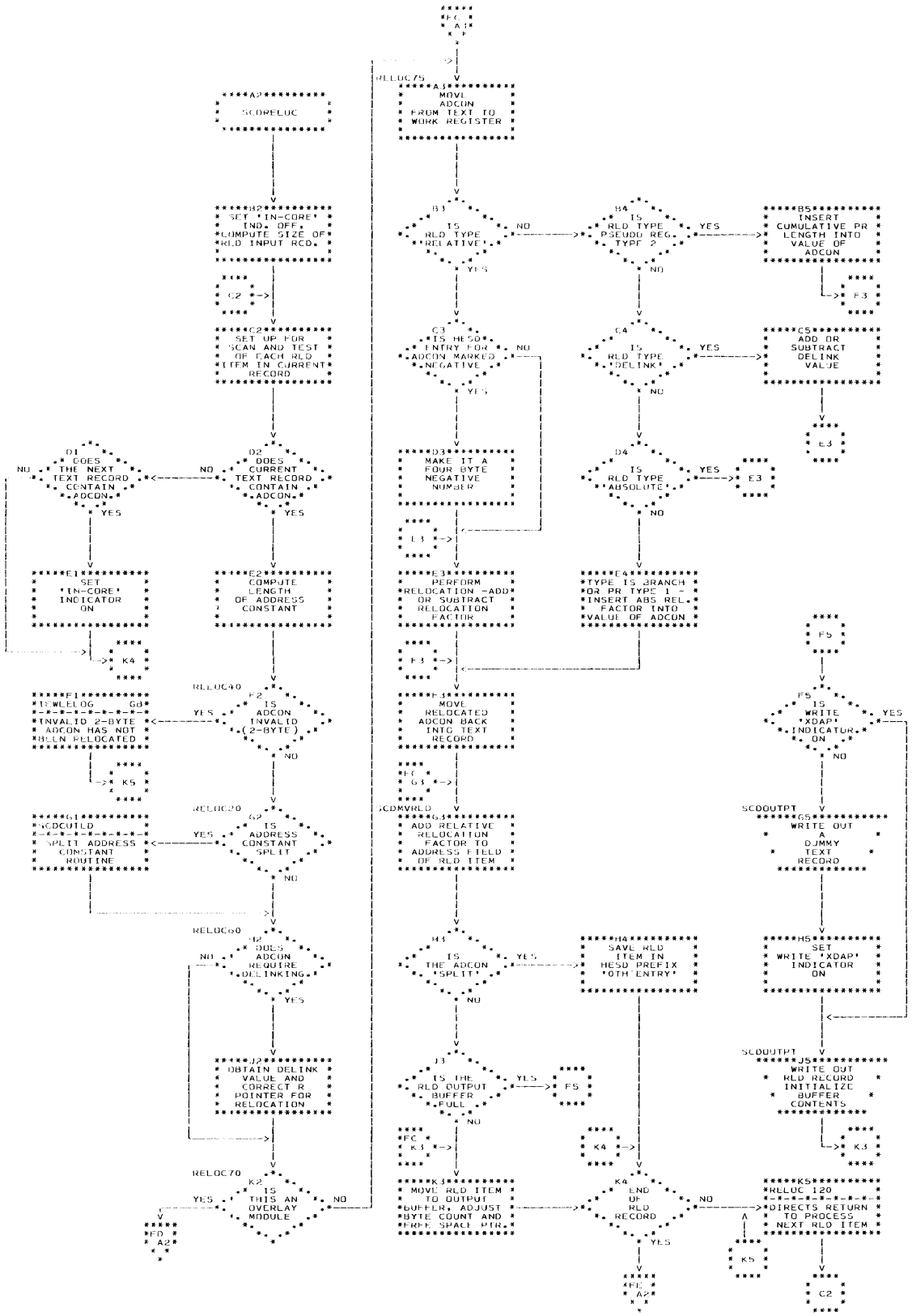


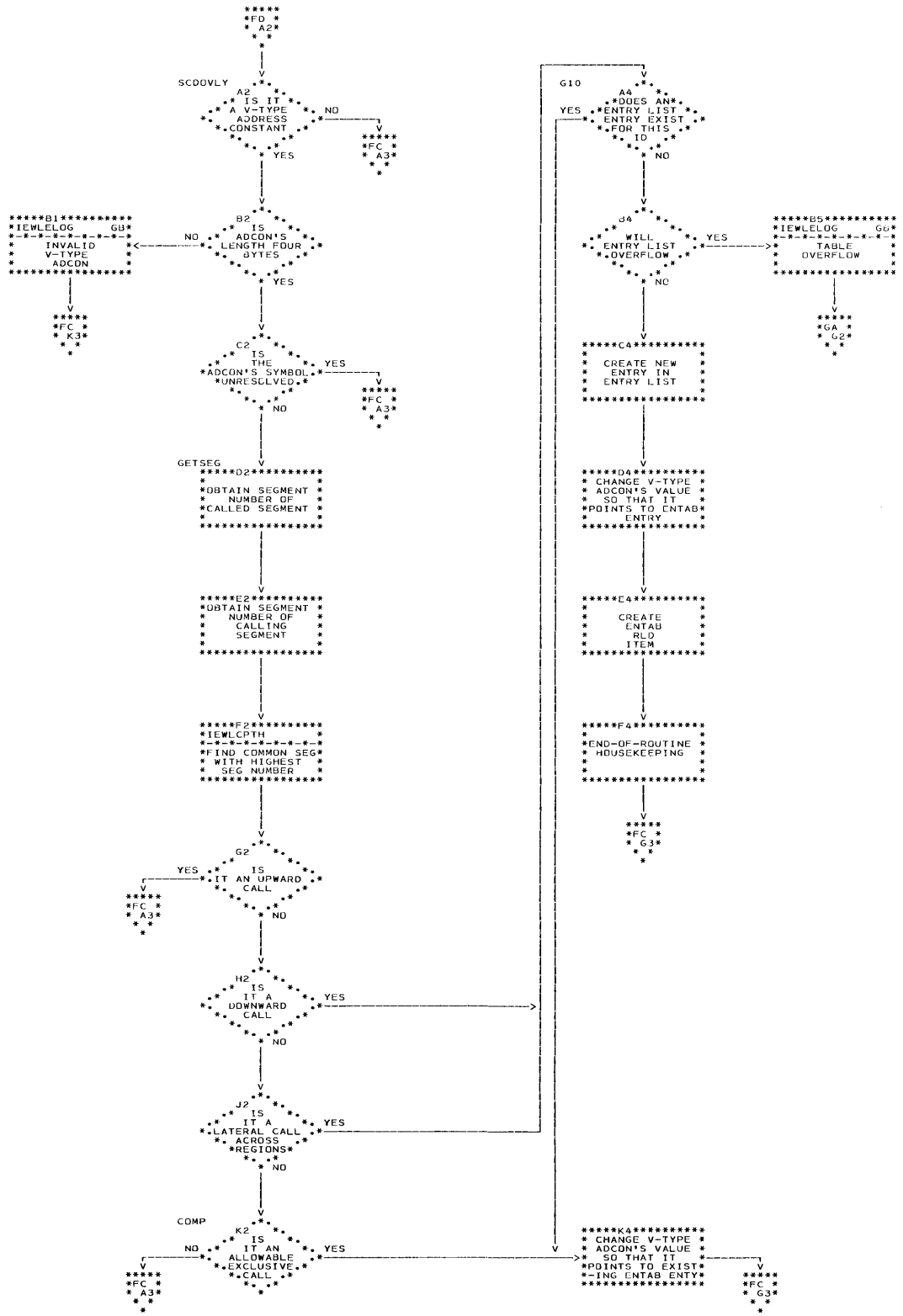
Chart FB. Second Pass Processor (IEWLESCD) (Continued)



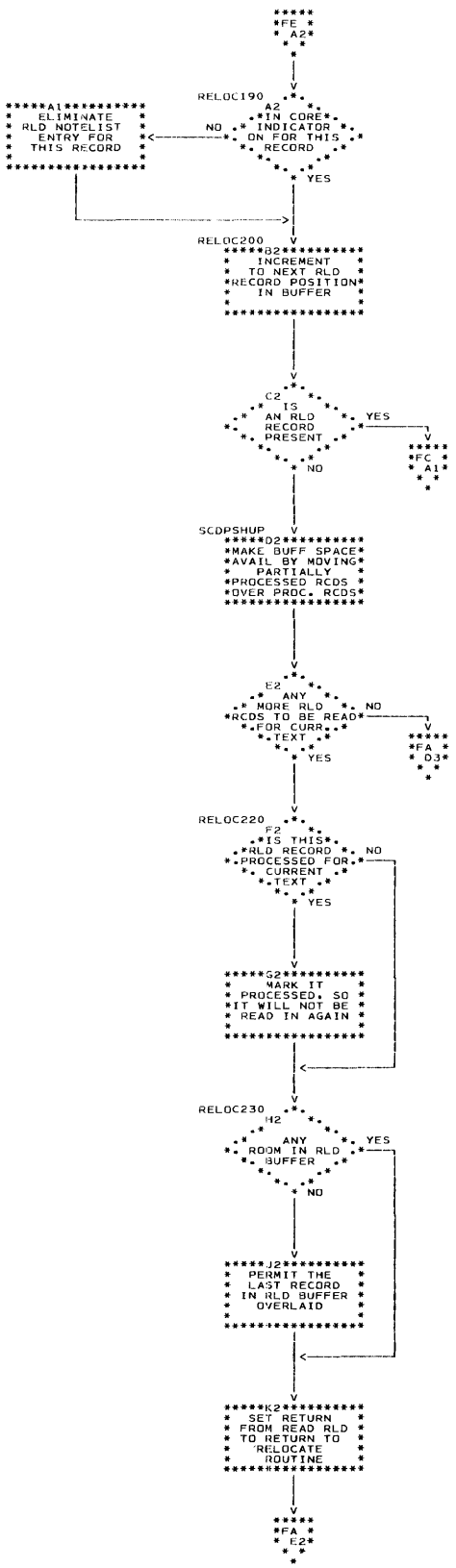
• Chart FC. Relocation Routine



• Chart FD. Relocation Routine (Continued)



• Chart FE. Relocation Routine (Continued)



• Chart GA. Final Processor (IEWLCFNL)

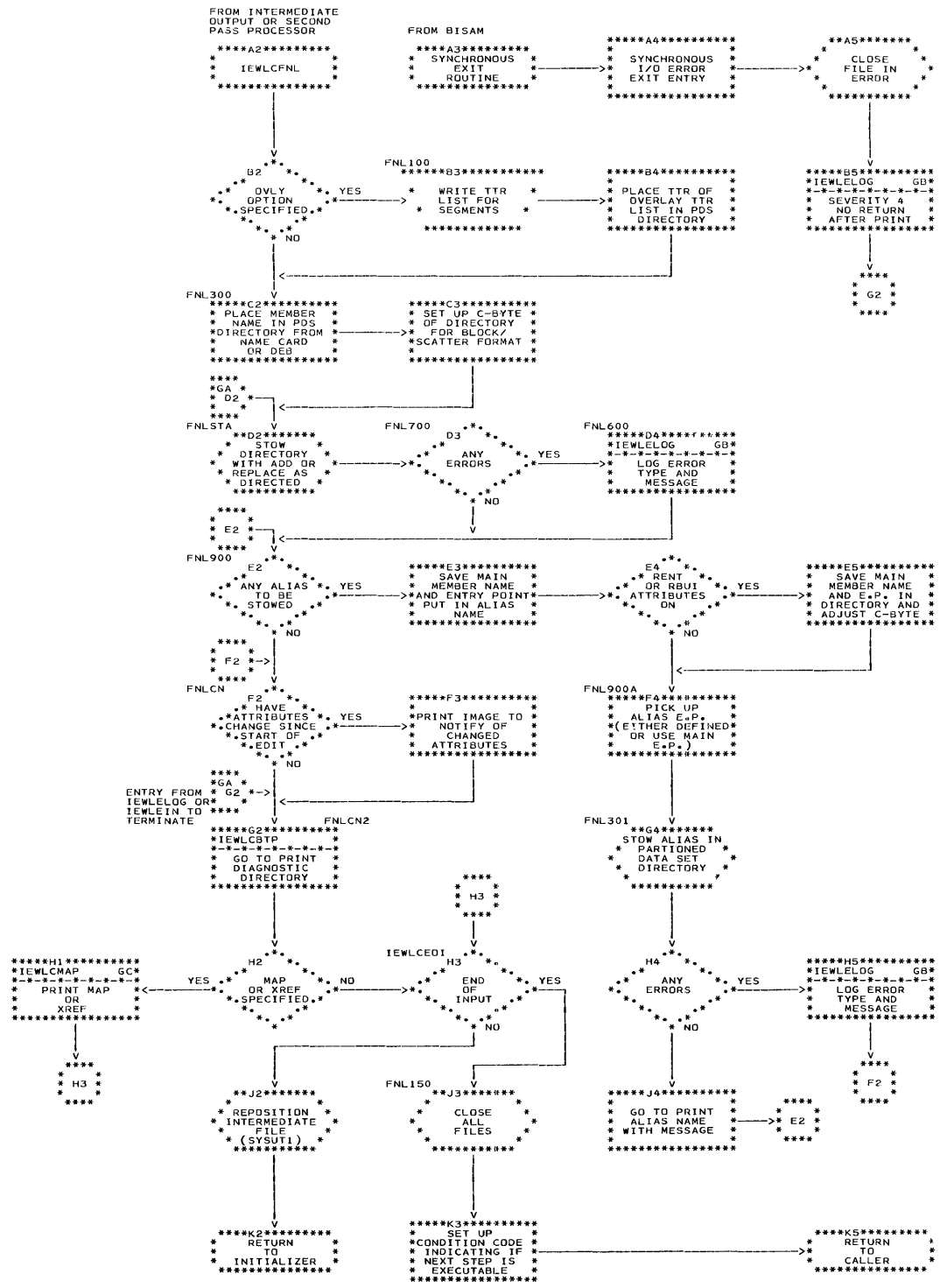


Chart GB. Error Logging Routine (IEWLELOG)

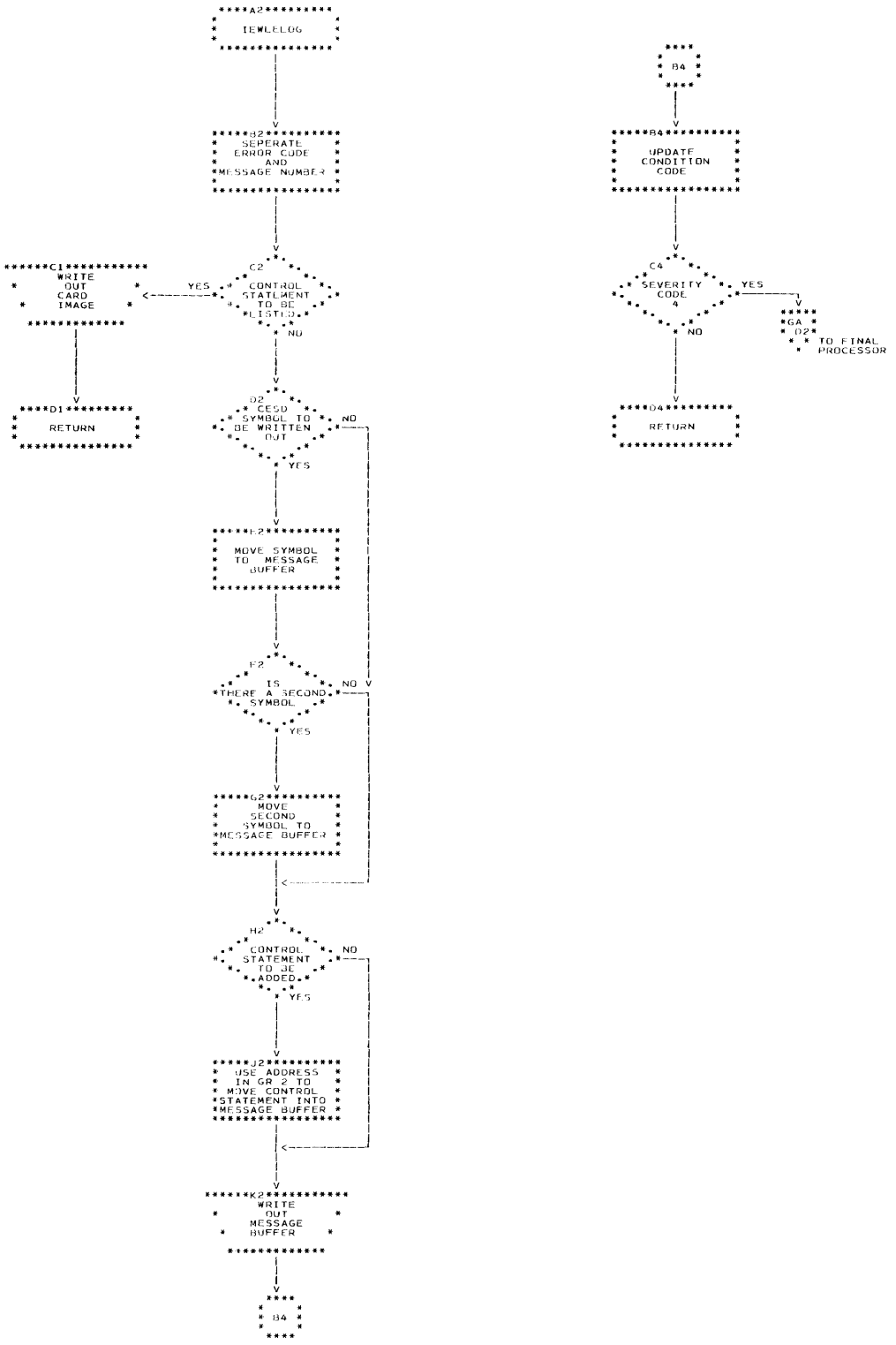
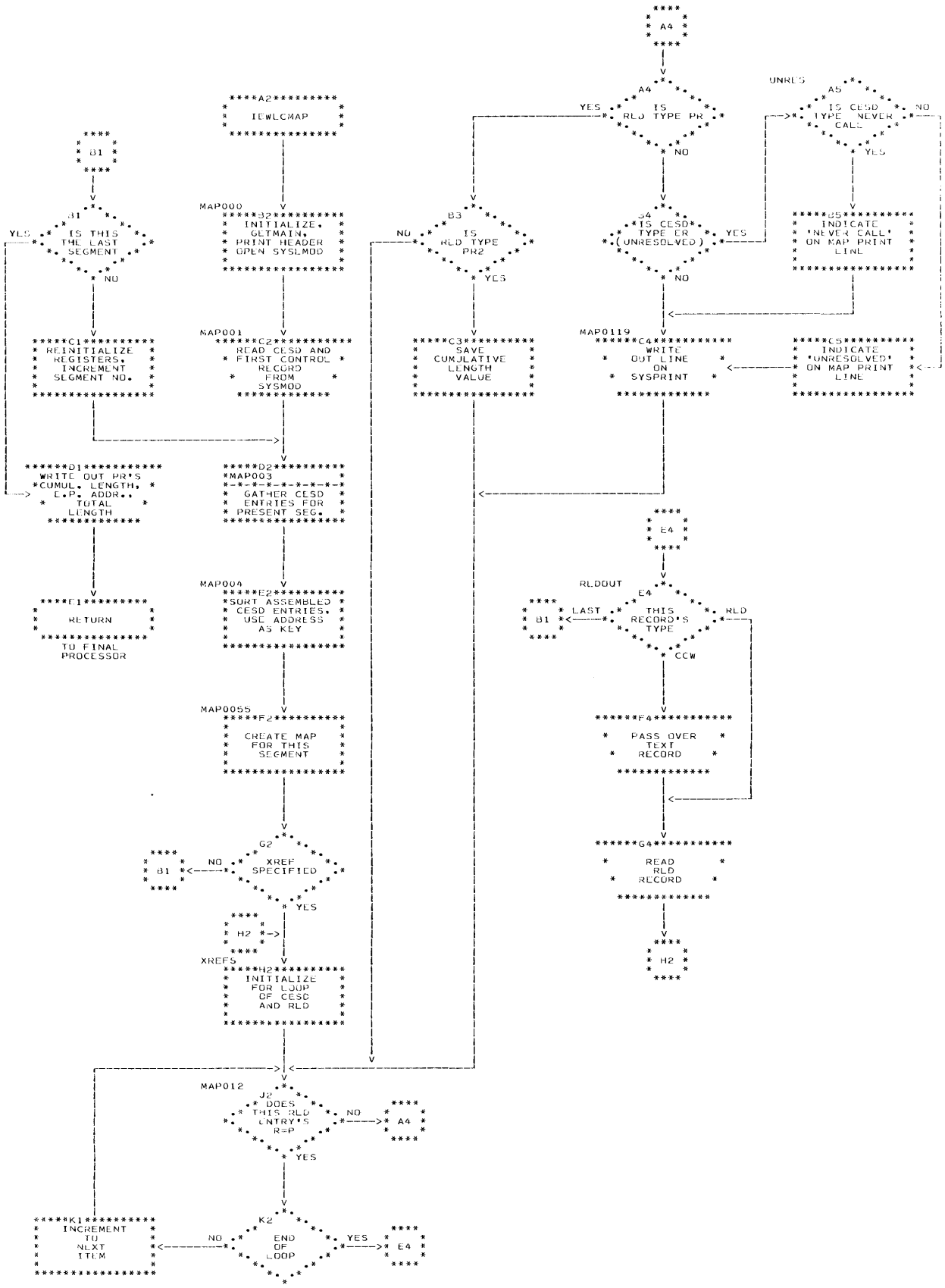
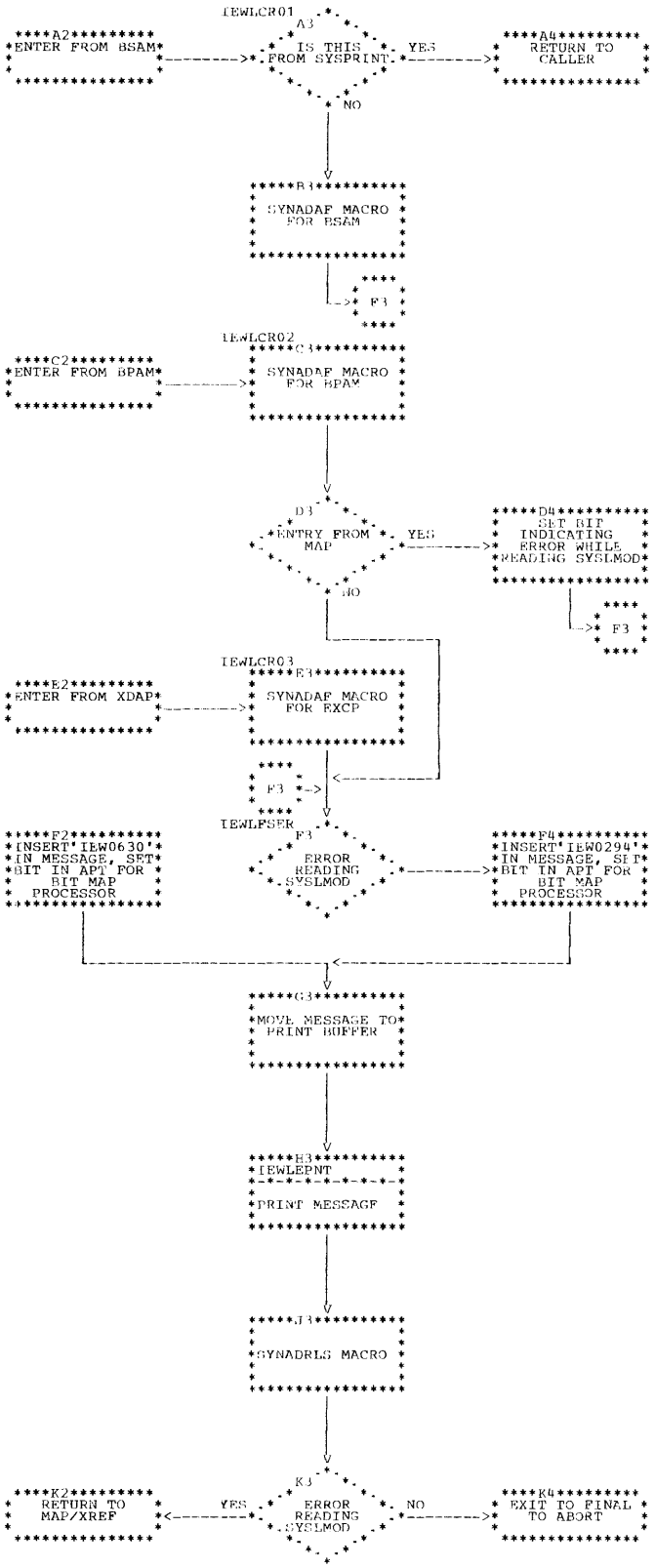


Chart GC. Module Map Processor (IEWLCMAP)



- Chart GD. SYNAD Routine



APPENDIX A: REFERENCE DATA FOR LEVEL E LINKAGE EDITOR

This section contains reference information, including linkage editor conventions, tables, and record formats, for the 15K and 18K level E linkage editor.

Note: 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 rules 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 the SD- or PC-type ESD entry that describes it 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 interprets 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 input/output operations, input modules should also conform with the following

---

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

conventions. Although violations of these rules are not treated as errors, compliance with them will improve the efficiency of linkage editor processing.

- Within an input module, no LD or SD should have the same name as an ER.
- Within an input module, no two ERs should have the same name.
- Within an input module, TXT records should 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.)

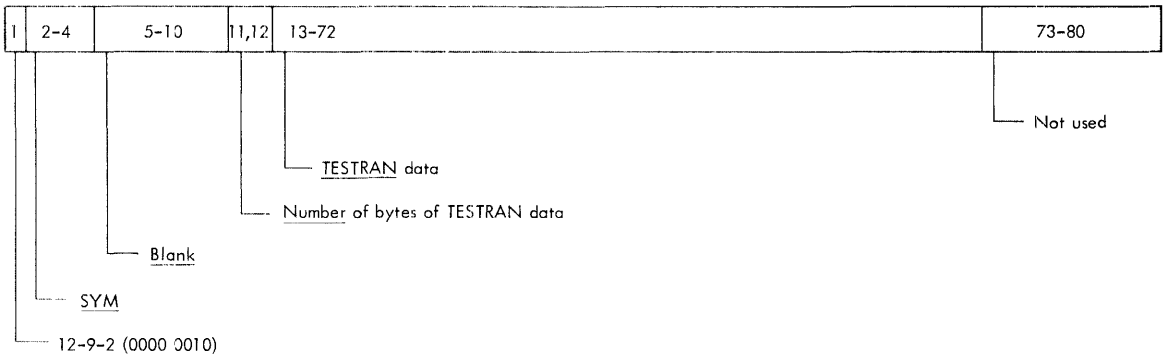
RECORD FORMATS

Following are the record formats produced during linkage editor processing.

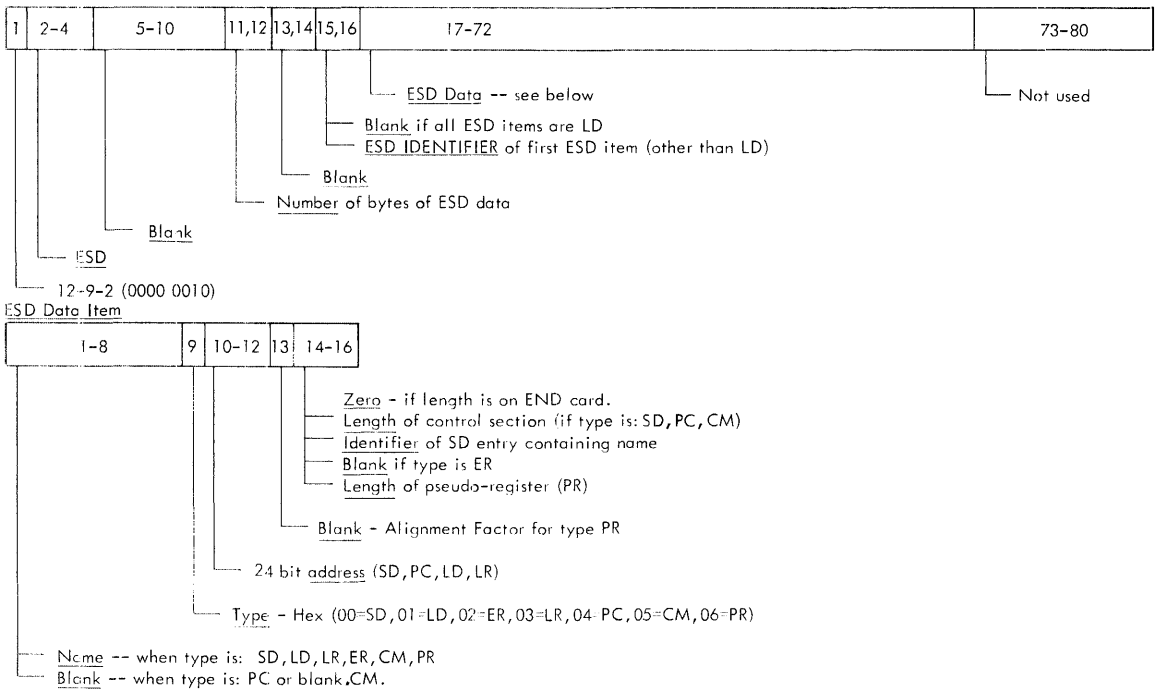
RECORD FORMATS - LEVEL E

The following are the card image load module record formats for the level E linkage editor.

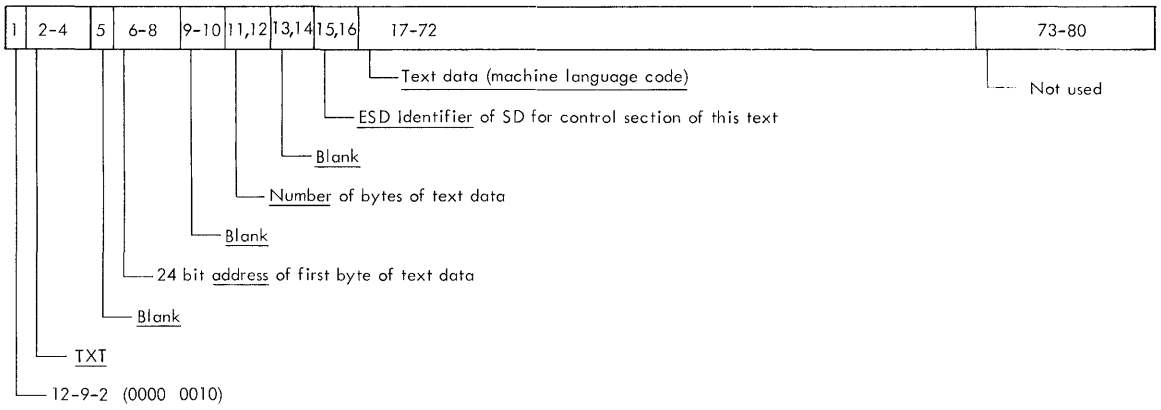
SYM Input Record (Card Image)



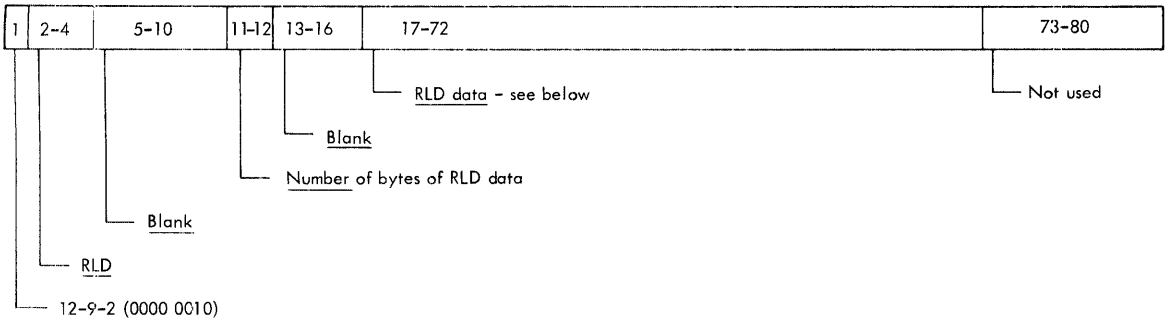
ESD Input Record (Card Image)

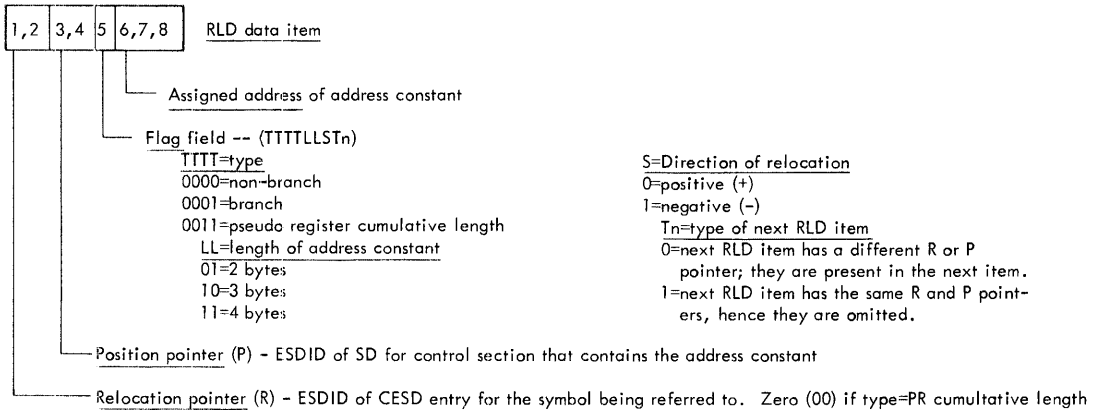


Text Input Record (Card Image)

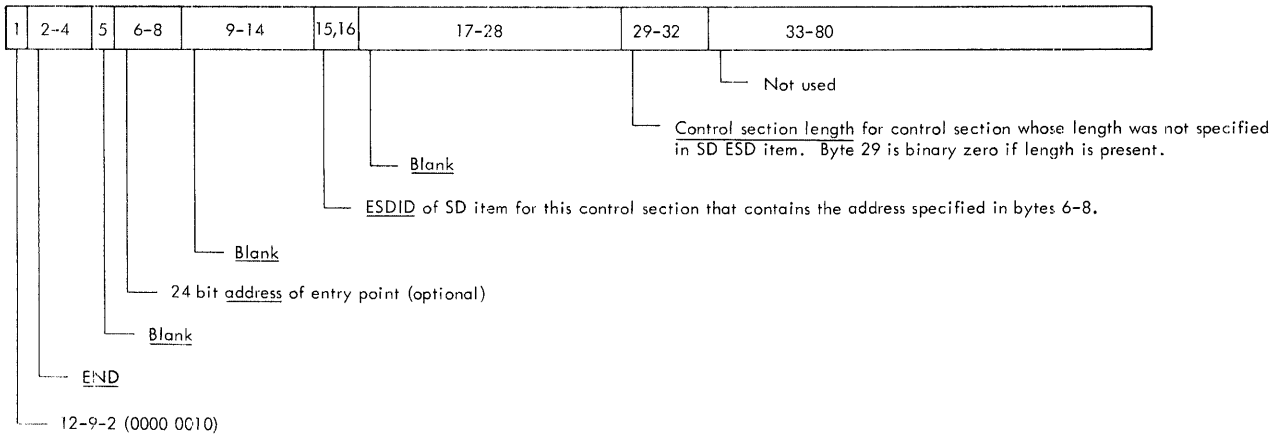


RLD Input Record (Card Image)

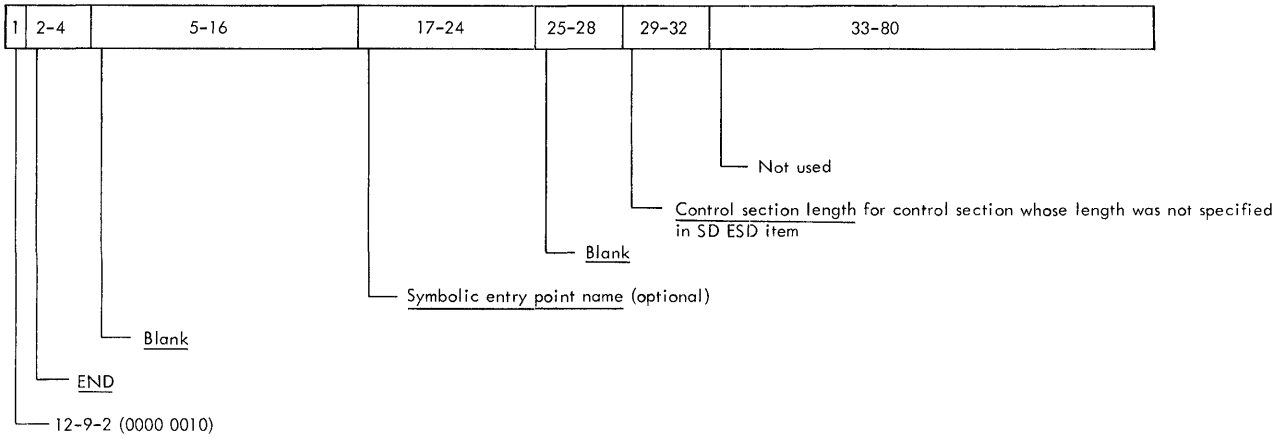




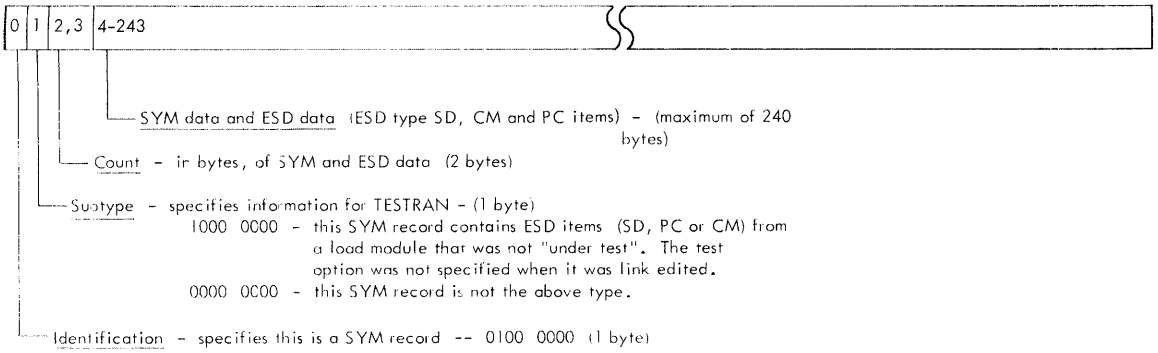
END Input Record - Type 1 (Card Image)



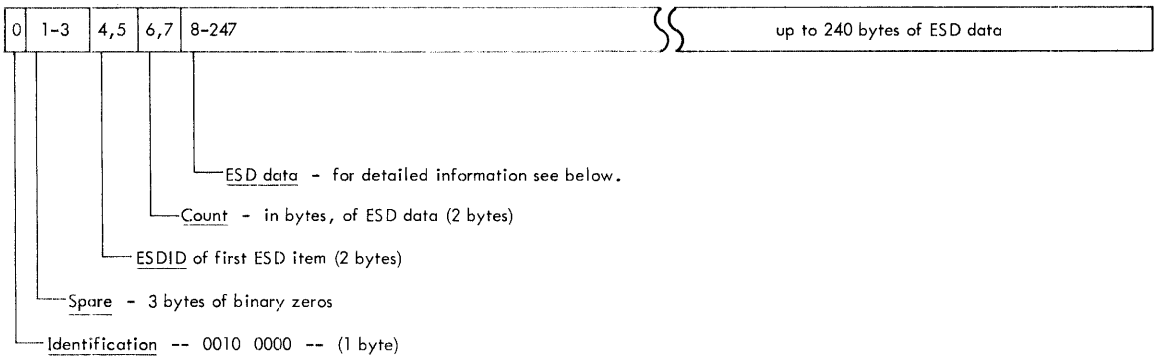
END Input Record - Type 2 (Card Image)



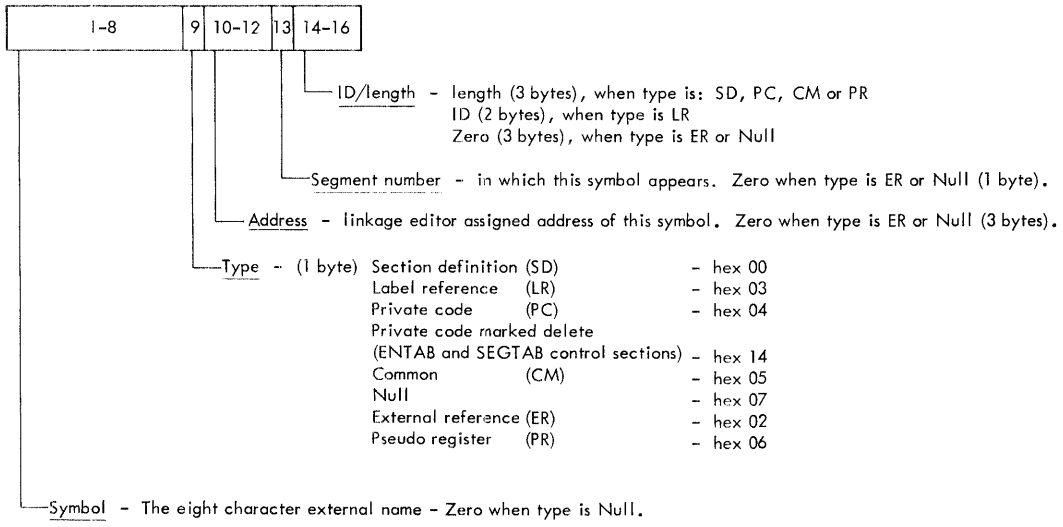
SYM Record - (Load Module)



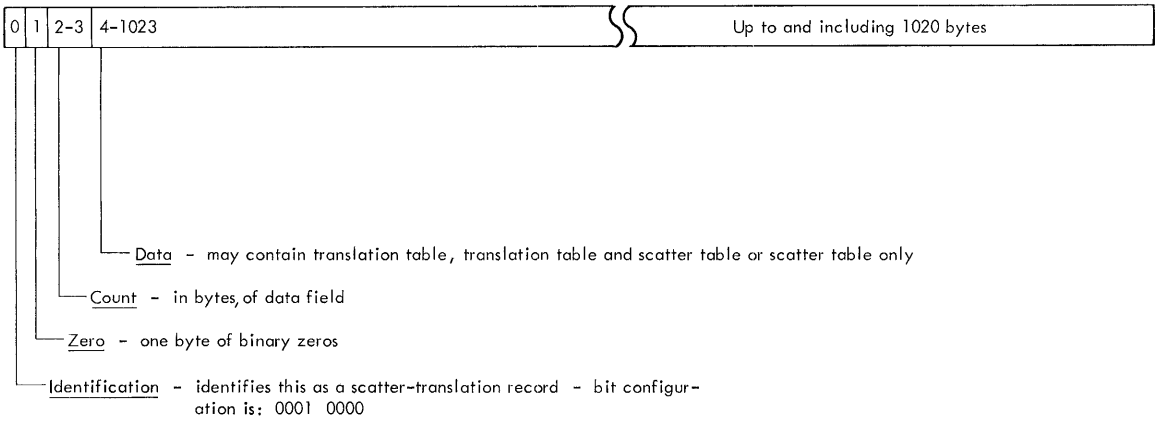
CESD Record - (Load Module)



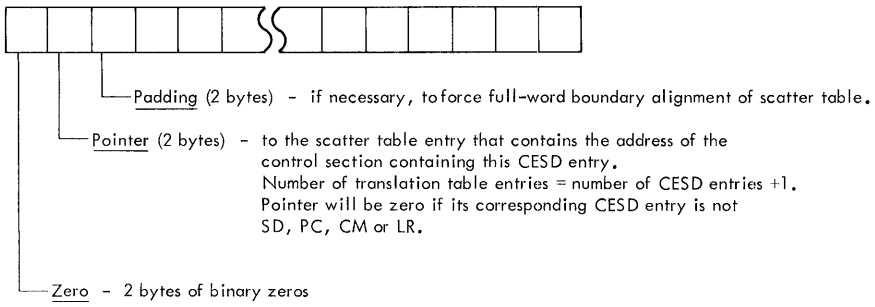
CESD Data (Load Module)



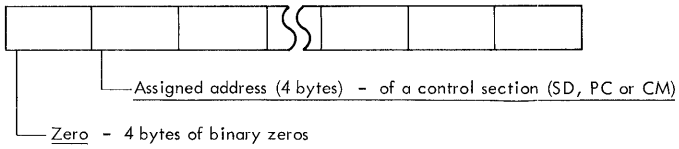
Scatter-Translation Record



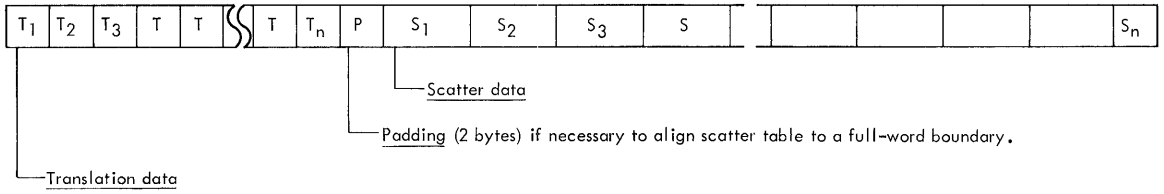
Translation Table



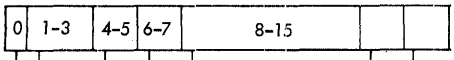
Scatter Table



Translation Table and Scatter Table



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.

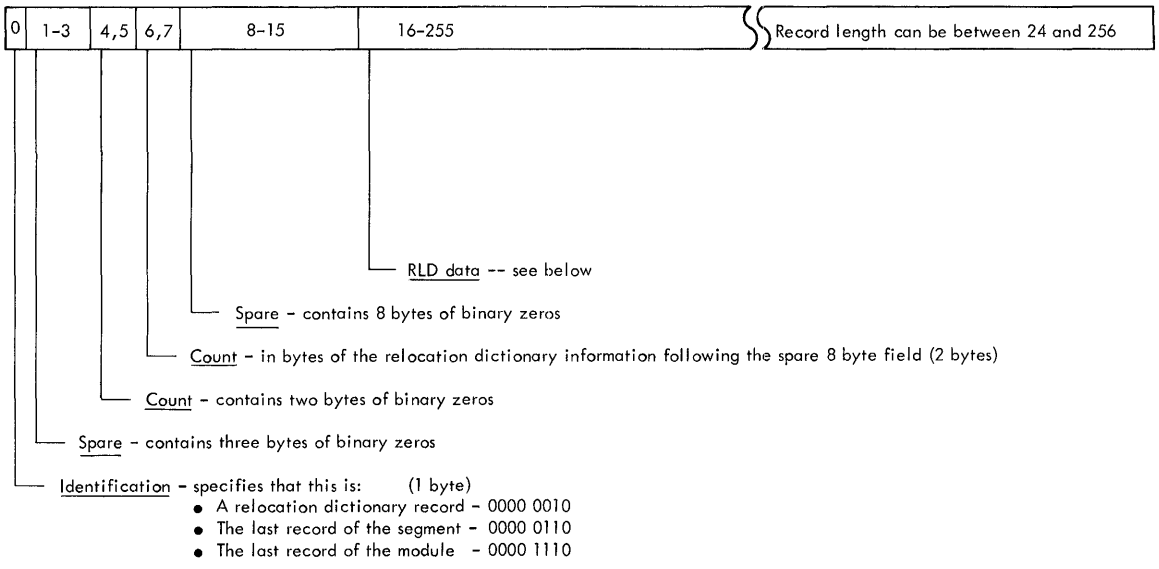
Count - in bytes, of the control information (CESD ID, length of control section) following the CCW field. The count is always 4 bytes when processed by the level E linkage editor.

Spare - contains three bytes of binary zeros

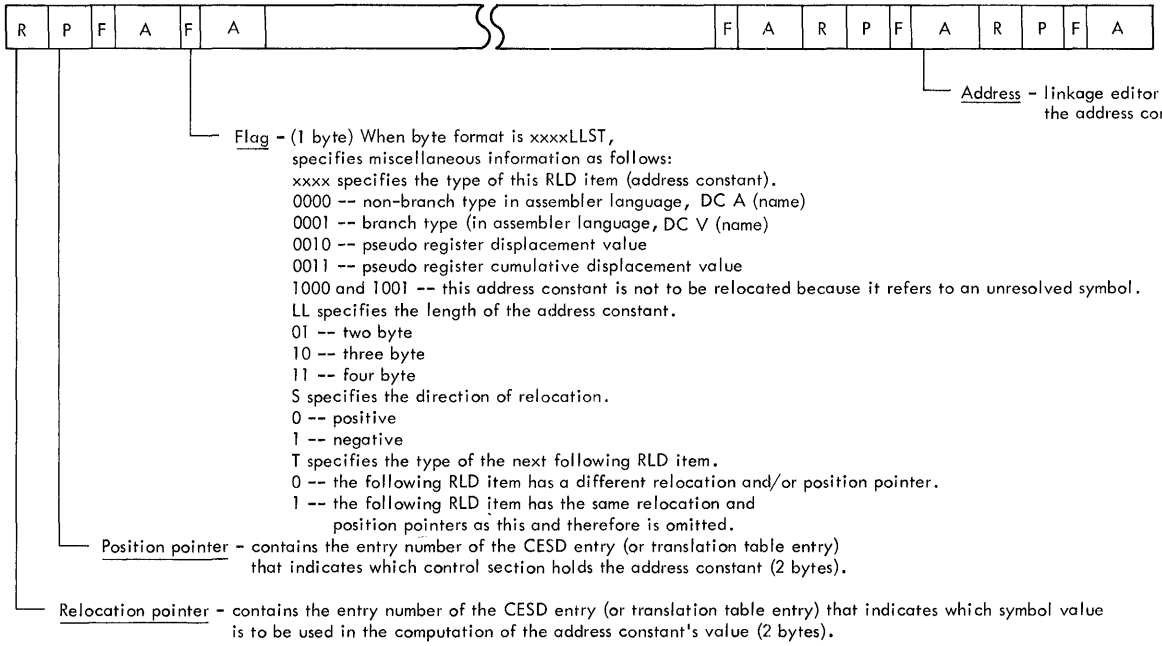
Identification - specifies that this is: (1 byte)

- 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)

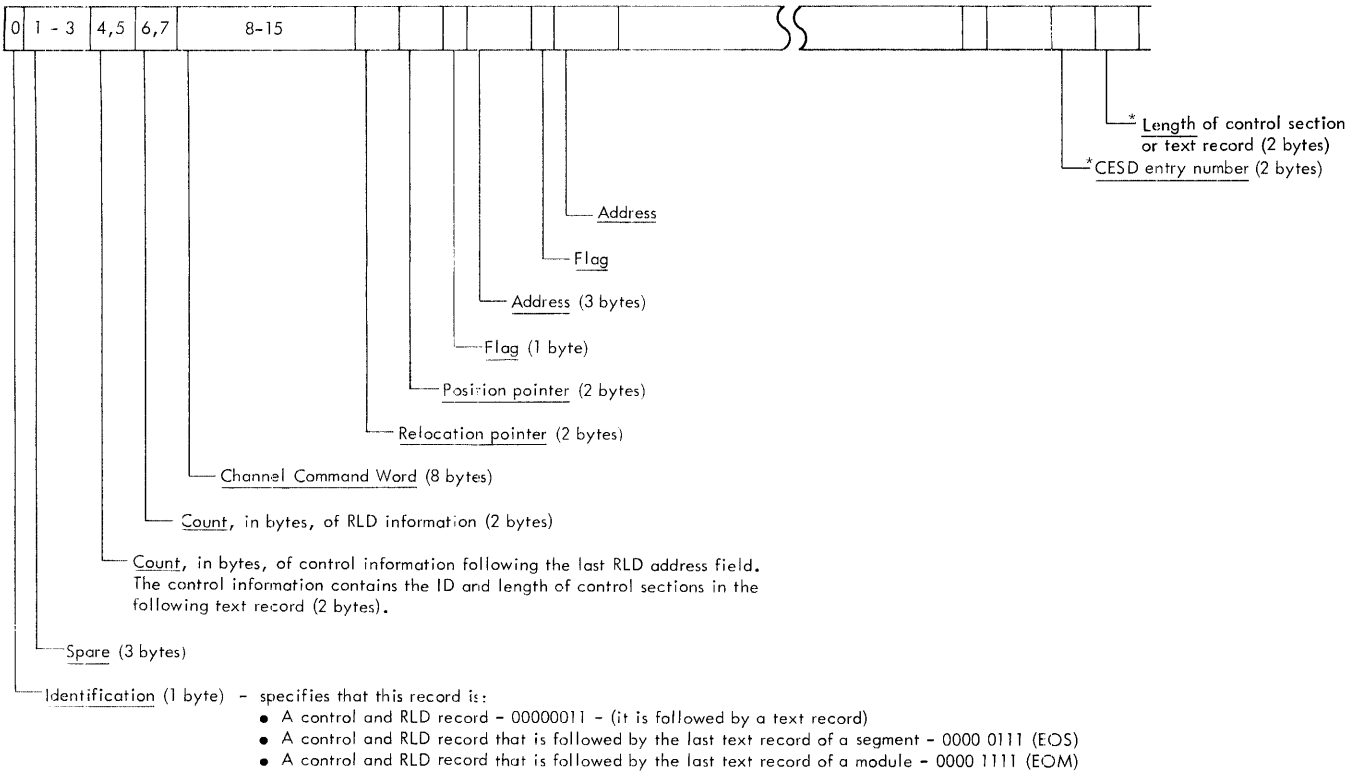
Relocation Dictionary Record - (Load Module)



RLD Data



Control and Relocation Dictionary Record - (Load Module)



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

REFERENCE DATA FOR INITIAL PROCESSING - 15K AND 18K LEVEL E

The following tables pertain to initial processing in the level E linkage editor.

All Purpose Table

0	PDSE1					
8	PDSE2	PDSE3	PDSE4			
16	PDSE5	PDSE6	PDSE7	PSDE8	PDSE9	
24	PDSE9 (continued)	PDSE10	PDSE11		PDSE12	
32	PDSE12 (continued)	PDSE13	PDSE14	PDSE15	PDSE16	
40	PDSE16 (continued)	PDSE17	PDSE18			
48	PDSE18 (continued)		REGSA			
56	REGSA (continued)					
120	REGSA (continued)		IOCT			
128	IOCT (continued)					
136	IOCT (continued)					
144	IOCT (continued)		APT0	APT1	APT2	APT3
152	CTTR		CSNO		CRNO	
160	SLNTB		PRAL			
168	FLCD		RCCE			
176	RCCB		ALCB			
184	OVCMBGAD		SGT1			
192	CLLT		TNT1			
200	RNT1		LSTS			
208	RECNT		LOGAREA			
216	SYINB		ERDIG			
224	TXTIO		ALAS			

232	DLKT		CHESD	
240	SELST		TNLS2	
248	RNLS2		TTRLIST	
256	CUTRLD		RLDB1	
264	INRLD		BITMAP (Error Logging Map)	
272	BITMAP (continued)		LINECNT	HISEV
280	INCBRKPT		CRRTINCL	
288	ENCDX	ENTIX	ENRIX	ENT2X
296	ENR2X	ENT0X	ENCLX	ENDTX
304	ENS1X	BUFSIZ	HESD	
312	ENRLDIX	ENRLD2X	ENELTX	ENSPX
320	SAVATS	APTSWS	EPSM	
328	EPSM (continued)		ENTIC	ENRIC
336	ENTIC	ENIRC	ENTOC	ENCLC
344	ENSIC	ENASC	ENDTC	ENCDC
352	ENELTC	ENT2C	ENR2C	ENSPC
360	SYSRTN			
368	SPACES			
440	IEWLCSCD		SSI	
448	LIBNAME			
456	MAXBLKSZ	APT000	HIARBIT	
464	DCB'S			
824	APTEXLST			
832	APTXLST1		APTXLIST	
840	APTREG3		HIARADD	

Explanation of APT Entries

PDSE1	Member or alias name of module being created.
PDSE2	Relative disk address (TTR) of module on SYSLMOD.
PDSE3	C - byte - see partitioned organization directory record, alias indicator and miscellaneous information. During second pass processing, PDSE2 and PDSE3 contain the end address of the RLD record currently in the second pass RLD input buffer.
PDSE4	Relative disk address (TTR0) of first text record.
PDSE5	Relative disk address of note list or scatter-translation record.
PDSE6	"L" byte, number of TTRs in note list if present.
PDSE7	Module attributes — 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 RLD — 0
PDSE8	Bit 0 - Compatibility - 0 Bit 1 - Origin of 1st text record is zero. 1 Bit 2 - Assigned entry point is 0 1 Bit 3 - Module contains RLD items 0 Bit 4 - Module can be reprocessed 0 Bit 5 - Module does not contain SYM records 0 Bit 6 - Spare 0 Bit 7 - Module is refreshable 0
PDSE9	Total contiguous main storage requirement of 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 control section containing the entry point.
PDSE17	Entry point of main member name.
PDSE18	Member name of module. During input processing, word 1 of PDSE18 contains the CESD address of the control section with no length given. During second pass processing, PDSE18 is used in the following manner: Word 1 = Address of next free entry in RLD output buffer. Word 2 = Address of address constant within the text buffer.
REGSA	Register save area for IOS
IOCT	Input/Output Control Table During second pass processing, IOCT contains the following data: Words 1-3 = Register save area (reg. 13-15) Word 4 = Address of next available entry in ENTAB Word 4 = Address of next available entry in ENTAB RLD buffer. Word 5 = Address of current entry in the Entry List. Word 6 = Address of the RLD record in the RLD input buffer that is currently being processed.

(Continued)

Explanation of APT Entries (Continued)

APT0	All Purpose Indicators Bit 0 - NCAL Bit 1 - XREF BIT 2 - MAP Bit 3 - LET Bit 4 - LOG Bit 5 - XCAL Bit 6 - Input record is text or RLD. Bit 7 - A library card has been read
APT1	All Purpose Indicators Bit 0 - More include input to come Bit 1 - Auto library call in operation Bit 2 - Object or load module. Bit 3 - Delete indicator. Bit 4 - Entry point has been received. Bit 5 - Symbolic or absolute entry point. Bit 6 - Entry card has been received. Bit 7 - ESD-Write indicator.
APT2	All Purpose Indicators Bit 0 - Length received in END item. Bit 1 - No length received in the SD record. Bit 2 - SYM records in load module. Bit 3 - Status indicator received. Bit 4 - Include processing previously initiated. Bit 5 - Input/Output overlap indicator. Bit 6 - In module indicator Bit 7 - Control statement continuation
APT3	All Purpose Indicators Bit 0 - End of file. Bit 1 - Name card received Bit 2 - End of input Bit 3 - Stow as a replacement Bit 4 - Split address constant to be output. Bit 5 - More RLDs to be processed Bit 6 - Current RLD for split address constant found. Bit 7 - SYSLIB data set is open.
CTTR	TTR0 of first CESD record on SYSLMOD, if MAP or XREF is specified.
CSNO	Current segment number.
CRNO	Current region number. During second pass processing, CRNO contains the last ID for the current segment.
SLNTB	Address of segment length table
PRAL	Pseudo register accumulative length.
FLCD	Address of first deleted CESD entry.
RCCE	Address of end of replace/change chain. During second pass processing, RCCE is used as a work area to perform address constant alignment.
RCCB	Address of beginning of replace/change chain. During second pass processing, RCCB contains the address of the address constant in the work area.
ALCB	Address of beginning of Alias chain. During second pass processing, ALCB contains the address of the next ENTAB entry in the HESD.
OVCMBGAD	Address of beginning of overlay chain During second pass processing, OVCMBGAD contains the address of the RLD note list entry for the currently processed RLD input record.
SGT1	Address of SEGTAB1
CLLT	Address of calls list. During second pass processing, CLLT contains the maximum size of the RLD input buffer.

(Continued)

Explanation of APT Entries (Continued)

TNT1	Address of text note list 1
RNT1	Address of RLD note list 1
LSTS	Last segment in each region
RECNT	Address of relocation constant table or renumbering table. During second pass processing, RECNT contains the buffer relocation constant.
LOGAREA	Address of 32 byte error logging area
SYSINB	Address of object module buffer
ERDIG	Address of error log routine
TXTIO	Address of text I/O table
ALAS	Address of alias table
DLKT	Address of delink table
CHESD	Address of composite ESD
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
OUTRLD	Address of second pass output RLD buffer
RLDB1	Address of ENTAB buffer
INRLD	Address of second pass input RLD buffer
BITMAP	Bit switches used to log error messages
LINECNT	Line count of lines printed on SYSPRINT
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
ENT1X	Maximum number of entries in text note list 1
ENR1X	Maximum number of entries in RLD note list 1
ENT2X	Maximum number of entries in text note list 2
ENR2X	Maximum number of entries in RLD note list 2
ENTOX	Maximum number of entries 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
ENRLD1X	Maximum size of first pass RLD buffer
ENRLD2X	Maximum size of second pass input RLD buffer
ENELTX	Maximum number of entries in second pass entry list
ENSPX	SEGTAB ID
SAVATS	Attribute save area
APTSWS	All purpose table switches Bit 0-2 - Spare Bit 3 - Linkage editor E=1, linkage editor F=3 Bit 4 - Bit map processed - Initial value =0 Bit 5 - Linkage editor input received - Initial value =0 Bit 6 - SYM received - Initial value =0 Bit 7 - ESD received - Initial value =0
EPSM	Entry point symbol or address During second pass processing, EPSM contains the following data: Bytes 1-4 = Address of next ENTAB entry to be built. Byte 5 = Segment number of the next segment that requires an ENTAB to be created for it. Byte 6 = Not used. Byte 7,8 = Length of address constant being processed.
ENT1C	Current number of bytes in text note list 1
ENR1C	Current number of bytes 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 During second pass processing, ENITC and ENIRC contains the linkage editor assigned address of current text record.

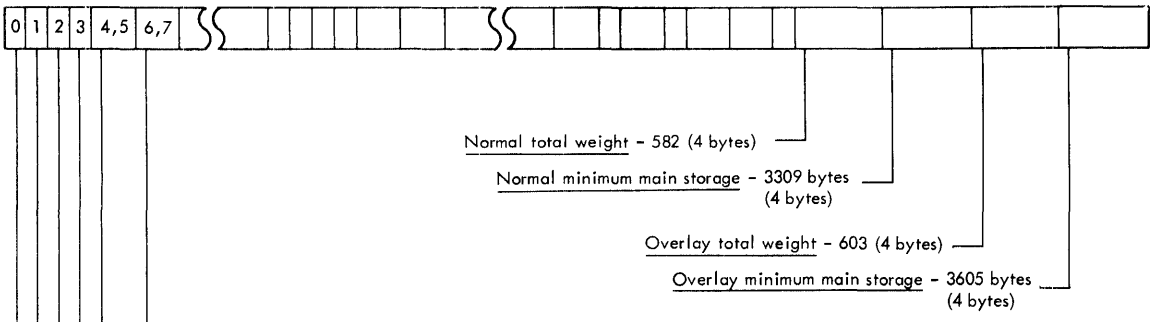
(Continued)

Explanation of APT Entries (Continued)

ENTOC	Current number of bytes in text I/O table
ENCLC	Current number of bytes in calls list During second pass processing, ENCLC contains the last R pointer obtained from the RLD buffer.
ENS1C	Current number of entries in SEGTAB1 During second pass processing, ENS1C contains the current segment number.
ENASC	Current number of entries in alias table
ENDTC	Current number of entries in delink table During second pass processing, ENDTC contains the next multiplicity number of the text to be read in for processing.
ENCDC	Current number of entries in CESD/HESD
ENELTC	Current number of entries in second pass entry list
ENT2C	Current number of entries in text note list 2 During second pass processing, ENT2C contains the last R pointer that was placed into the RLD output buffer.
ENR2C	Current number of entries in RLD note list 2
ENSPC	Highest segment number of segment containing text
SYSRTN	Save area for register 13 and 14 for return to job management
SPACES	Save area IEWLEDE1 = SPACES+32 IEWLEDE2 = SPACES+52 During second pass processing, SPACES contains the following data: Words 1,2 = Address and loop counter for next RLD note list entry to process for current text. Word 3 = Address of next available byte in the RLD input buffer. Words 4-7 = Temporary save area for BSAM disk address or track balance. Word 8 = Address of end of ENTAB RLD buffer. Words 9-13 = DECB for text buffer 1. Words 14-18 = DECB for text buffer 2.
IEWLCSCD	Address of second pass processor
SSI	System status indicator
FFCADR	Highest address retained by allocation routine
LIBNAME	Name of library for automatic library call
MAXBLKSZ	Maximum block size for linkage editor E (80 byte)
APT000	SYNAD for printer
HIARBIT	Storage hierarchies have been specified if high order byte contains X'01'.
DCBs	DCBs for linkage editor devices.
APTREG3	Save area.
HIARADD	Hierarchy table address.

Main Storage Allocation Table

Used by Allocation Processor



Normal total weight - 582 (4 bytes)

Normal minimum main storage - 3309 bytes (4 bytes)

Overlay total weight - 603 (4 bytes)

Overlay minimum main storage - 3605 bytes (4 bytes)

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)

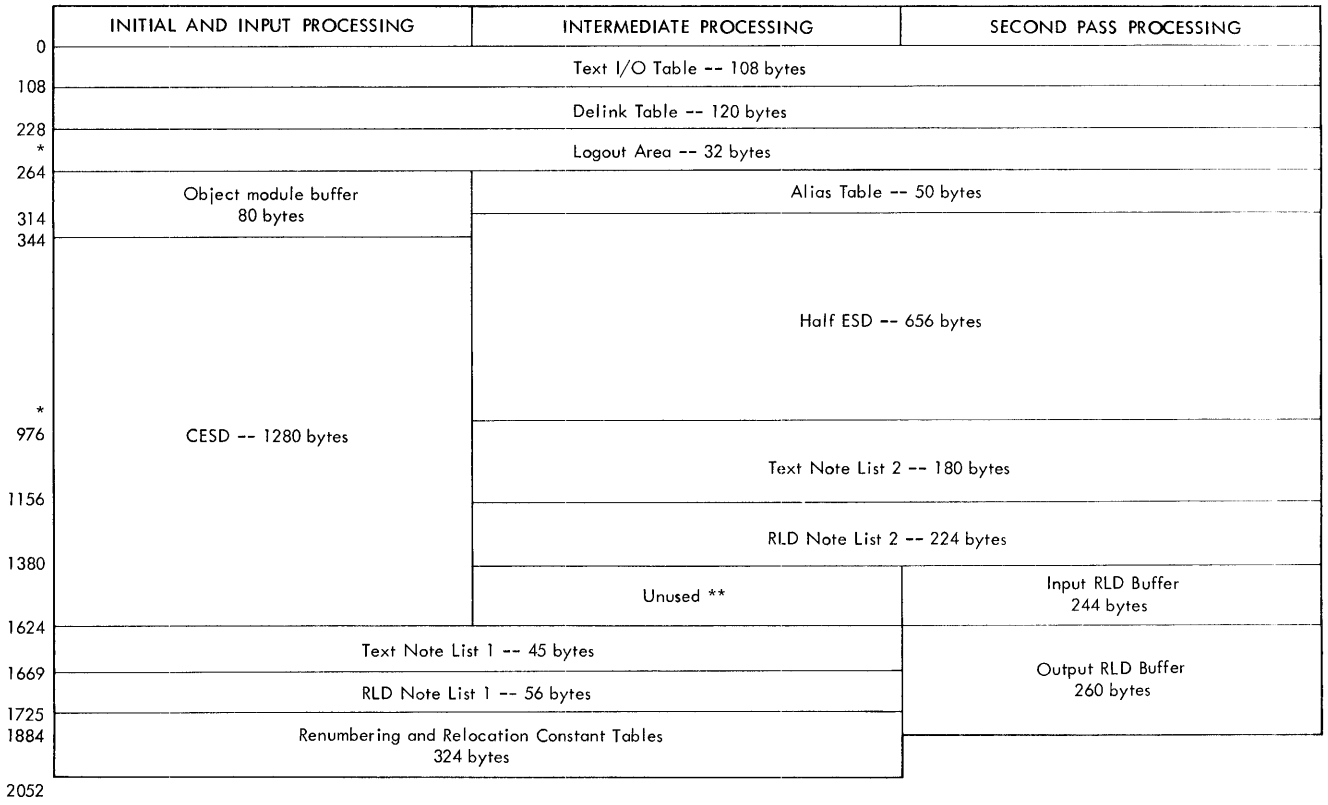
Number of Entries-156 - The number of entries value for this table found in the All Purpose Table, reduced by 156 (1 byte)

Address-156 - The address assigned to this table (buffer or area) found in the All Purpose Table reduced by 156 (1 byte)

Indicators - (1 byte)

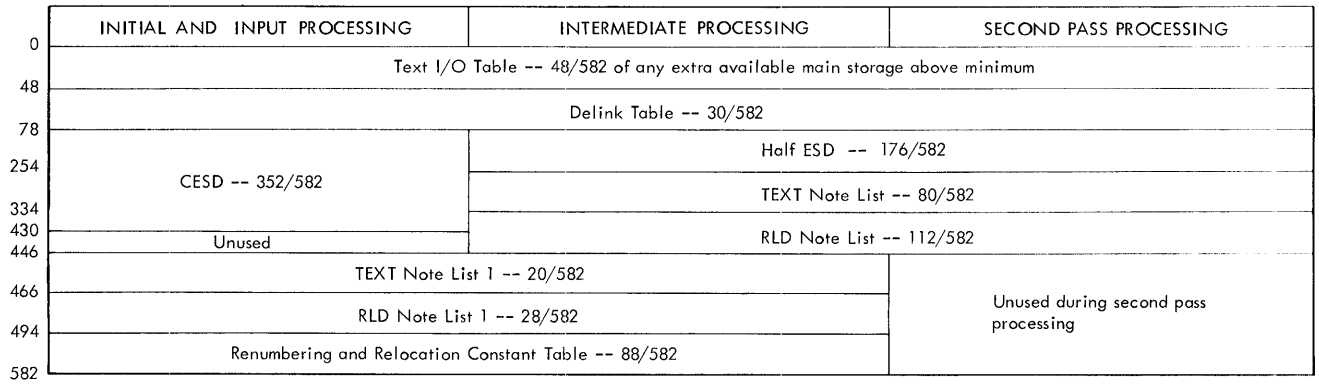
- Bit 0 - Table needed to process overlay modules only.
- Bit 1 - Table needed during first pass.
- Bit 2 - Table needed for intermediate processing.
- Bit 3 - Table needed during second pass.
- Bit 4 - Table requires double-word alignment.
- Bit 5 - Table requires word alignment.
- Bit 6 - Table has a maximum size of 240 or 64 bytes.
- Bit 7 - Table has a zeroeth entry (Prefix).

Minimum Table Area for Processing Non-Overlay Programs

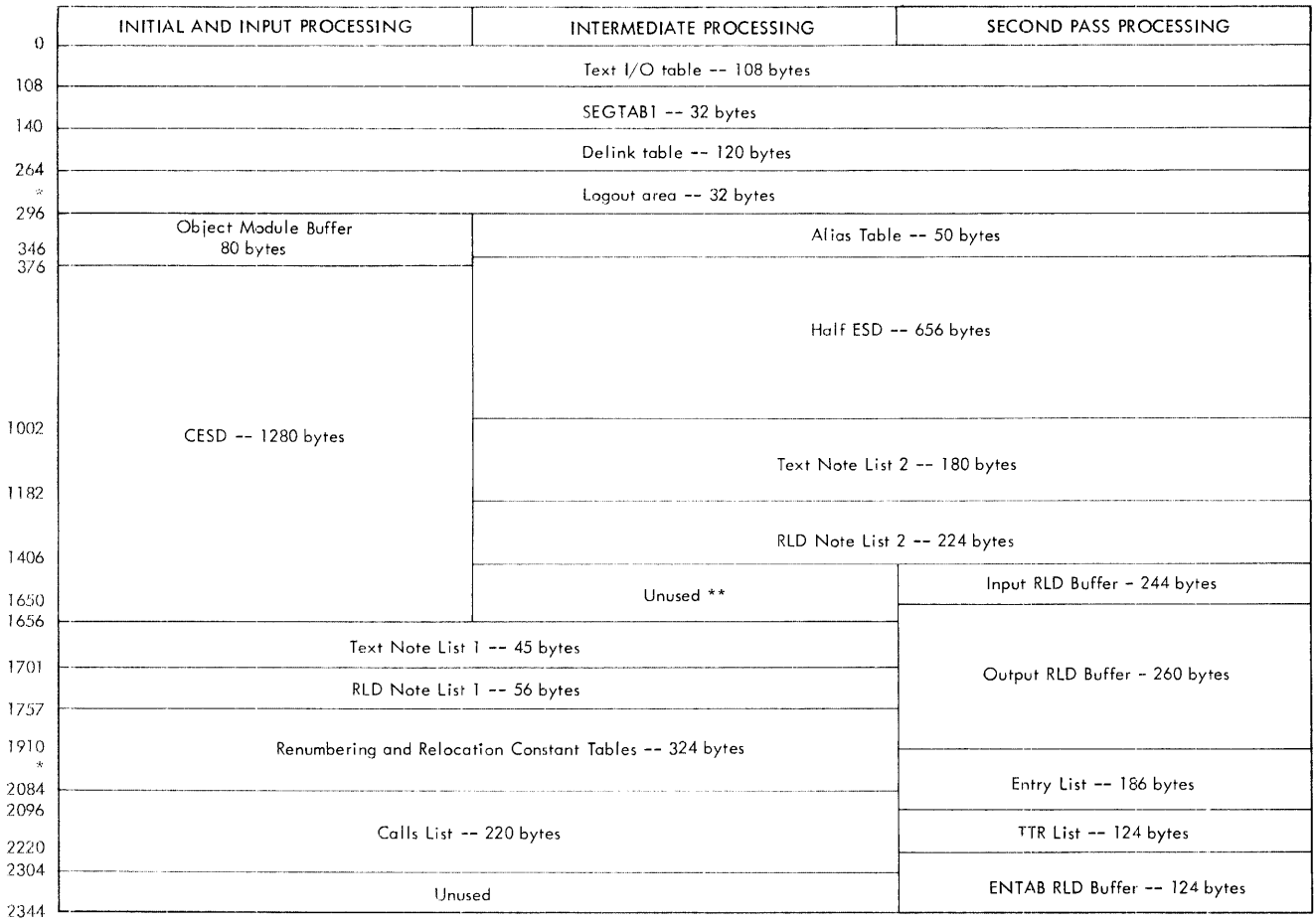


* The byte number below is not consecutive because of the necessity for proper boundary alignment.
** If an additional 9,312 bytes are available, there is no unused space during intermediate processing.

Expansion of Table Area Into Extra Available Main Storage (Non-Overlay Processing)



Minimum Table Area for Processing Overlay Programs



* The byte number below is not consecutive because of the necessity for proper boundary alignment.

** If an additional 10,251 bytes are available, there is no unused space during intermediate processing.

Expansion of Table Area Into Extra Available Main Storage (Overlay Processing)

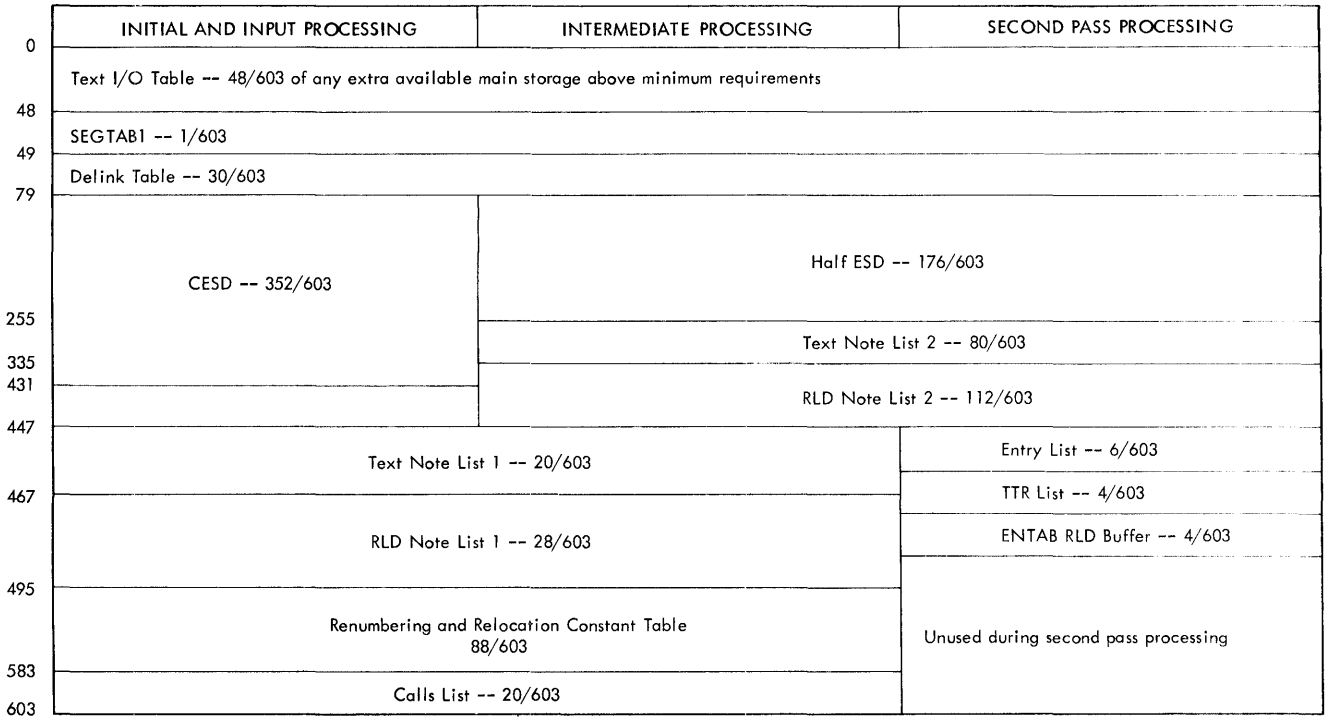


Table of Buffer Sizes and Table Sizes

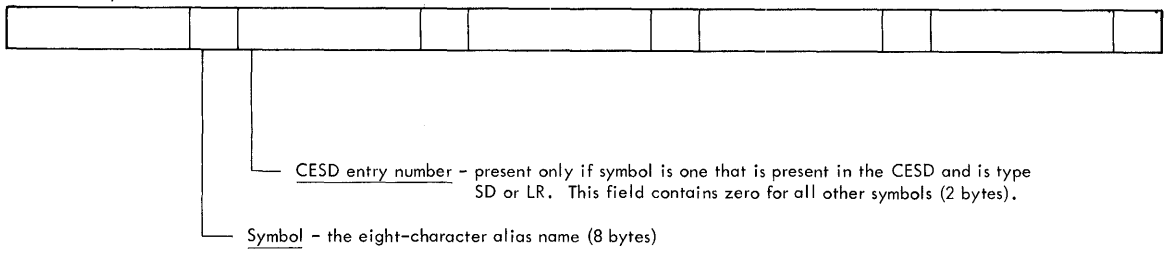
Table Name	OVLY Only	Order in Coding	Bytes/ Entry	Weight	Present in:			Prefix	Align	Size (in bytes)	
					1st Pass	Int Proc	2nd Pass			Min.	Max.
Alias Table	No	5	10	0	No	Yes	Yes	No	Byte	50	50
Calls List	Yes	20	2	20	Yes	Yes	No	No	Word	220	*
Composite ESD	No	11	16	352	Yes	No	No	Yes	Dblwd	1280	*
Delink Table	No	3	5	30	Yes	Yes	Yes	Yes	Byte	120	*
ENTAB RLD Buffer	Yes	16	1	4	No	No	Yes	No	Word	124	240
Entry List	Yes	14	6	6	No	No	Yes	No	Byte	186	*
Error Log Area	No	4	1	0	Yes	Yes	Yes	No	Word	32	32
Half ESD	No	7	8	176	No	Yes	Yes	No	Dblwd	648	*
Half ESD Prefix	No	6	1	0	No	Yes	Yes	No	Dblwd	8	8
Input RLD Buffer	No	12	1	0	No	No	Yes	No	Word	244	244
Object Module Buffer	No	10	1	0	Yes	No	No	No	Byte	80	80
Output RLD Buffer	No	13	1	0	No	No	Yes	No	Word	260	260
Relocatable Constant Table	No	19	4	88	No	Yes	No	Yes	Word	320	*
Renumbering Table	No	19	4	88	Yes	No	No	Yes	Word	320	*
Renumbering Table Prefix	No	18.5	4	0	Yes	Yes	No	No	Word	4	4
RLD Note List 1	No	18	7	28	Yes	Yes	No	No	Byte	56	*
RLD Note List 2	No	9	7	112	No	Yes	Yes	No	Byte	224	*
SEGTAB1	Yes	2	1	1	Yes	Yes	Yes	Yes	Byte	32	64
Text I/O Table	No	1	3	48	Yes	Yes	Yes	No	Byte	108	*
Text Note List 1	No	17	5	20	Yes	Yes	No	No	Byte	45	*
Text Note List 2	No	8	5	80	No	Yes	Yes	Yes	Byte	180	*
TTR List	Yes	15	4	4	No	No	Yes	Yes	Word	124	*

* Maximum is determined by availability of main storage

REFERENCE DATA FOR INPUT PROCESSING -- LEVEL E

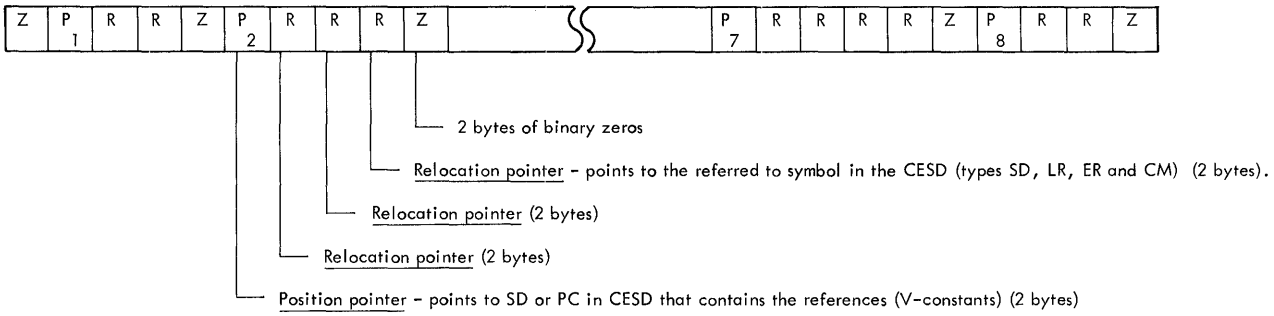
Alias Table

Built by: Entry Processor
Referred to by: Final Processor



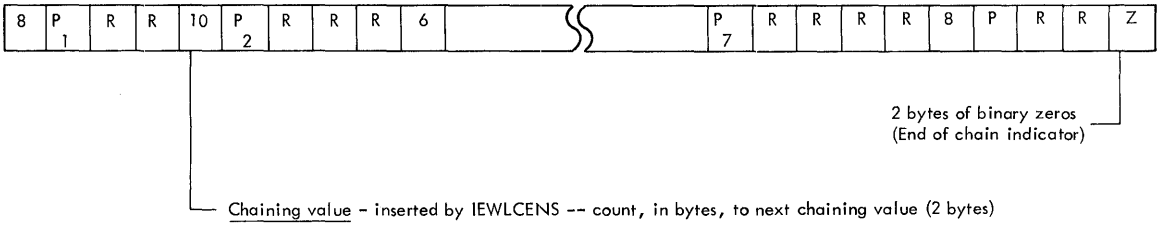
Calls List

as built by RLD processor



Calls List

As altered and used by ENTAB size determination (IEWLCENS)



Composite External Symbol Dictionary (CESD) -- Internal Format

The following tables are produced during input processing in the level E version of the linkage editor.

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 - ER

		Hex
ER	0000 0000	00
ER-Control change	1111 0000	F0
ER-Control replace	1110 0000	E0
ER-Control delete	1110 1000	F8
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)

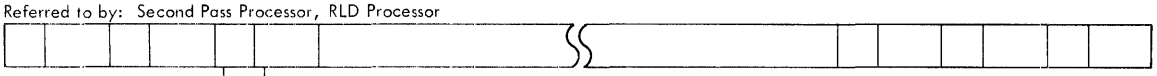
Normal Combination of Internal CESD Types

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

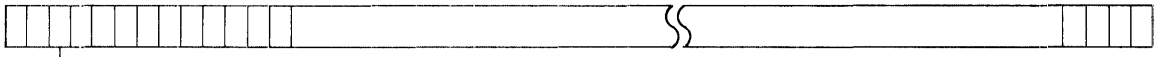


Address - assigned to the symbol being deleted (3 bytes)

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)

Downward Calls List

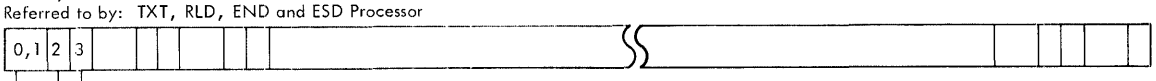
Built by and referred to by IEWLCENS routine



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)

Renumbering Table

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



Type

Bits 567	
Section Definition	000
Label Reference	011
External Reference	010
Private Code	100
Common	101
Pseudo Register	110
Null	111

(1 byte)

Subtype

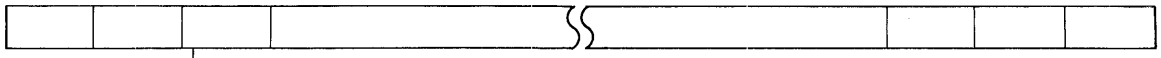
Bits 01234	
Null	000000
Delete	00010
Replace	00010
Chain	01000
Insert	00100
Library	10000

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)

Relative Relocation Constant Table

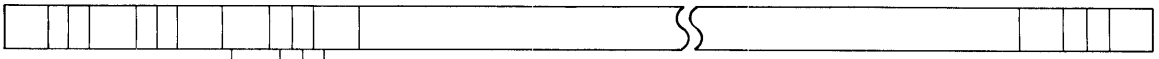
Built by and referred to by Address Assignment Processor



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)

RLD Note List

Built by: RLD Processor
Referred to by: Second Pass Processor



Relative Track Address (TTR) - of this RLD record on SYSUT1.

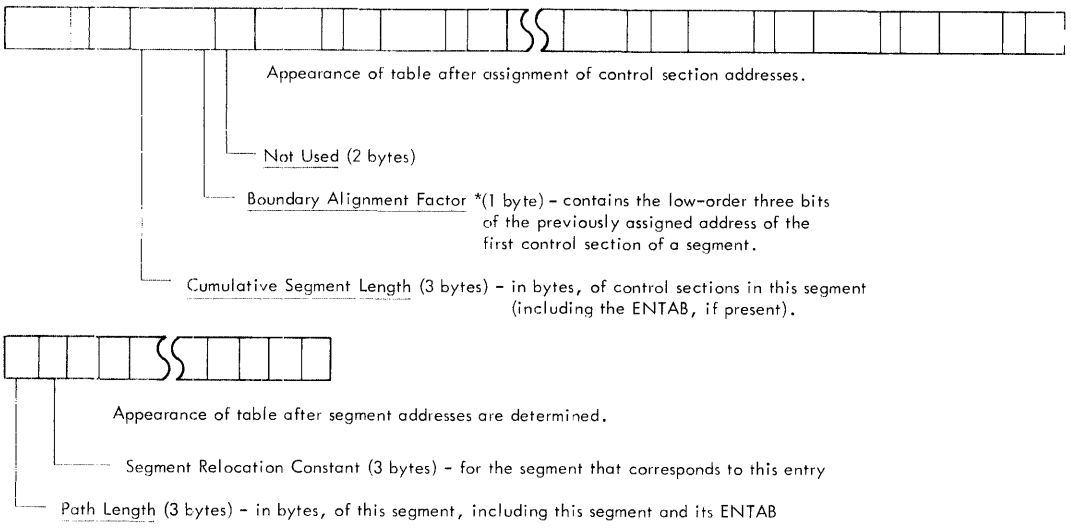
Record length - the number of words of RLD data.
During Second Pass Processing:
Bit 0 is an 'In Core' indicator.
Bit 1 is a 'Processed' indicator.

Lowest multiplicity - of the control section referred to by the ID field, to which the RLD information in this record pertains.

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

Segment Length Table

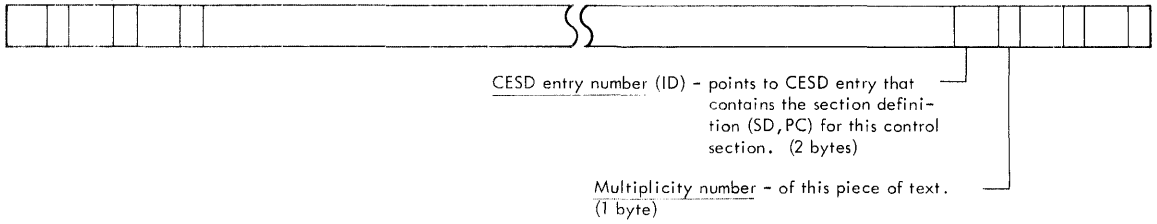
Built and referred to by Address Assignment Processor



* Note: The three low-order bits of the previously assigned address of the first control section of each segment are saved in the high-order byte of the segment relocation constant field. These bits are used to retain correct byte alignment when computing the segment relocation constant. When the computation is completed, the result will overwrite all three bytes of the segment relocation constant field.

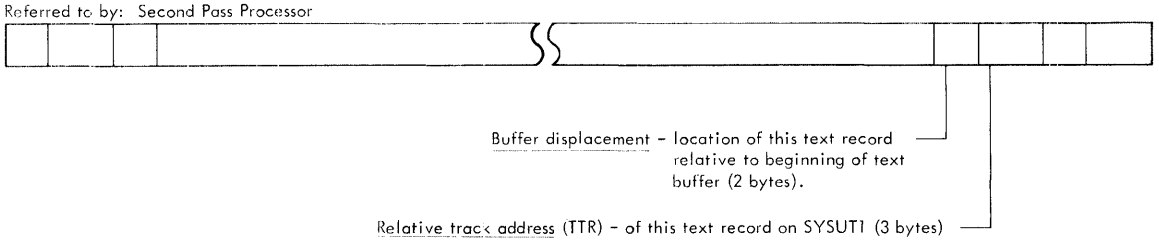
Text Input/Output Table

Built by: Text Processor
Referred to by: Second Pass Processor and Text Processor



Text Note List

Built by: Text Processor
Referred to by: Second Pass Processor



Note: There is a one-to-one correspondence between entries of text input/output table and the text note list.

REFERENCE DATA FOR INTERMEDIATE PROCESSING -- LEVEL E

The following table is produced during intermediate processing in the level E version of the linkage editor.

Segment Table (SEGTAB)

Built by Intermediate Output Processor

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

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:

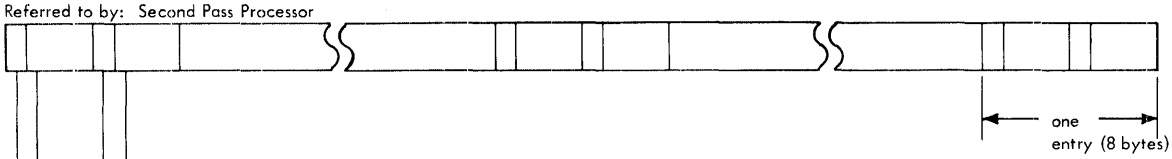
00 -- segment is in main storage as a result of a branch to the segment.
10 -- segment is in main storage, no caller chain exists.
01 -- segment is not in main storage, but is scheduled to be loaded.
11 -- segment is not in main storage.

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

Half External Symbol Dictionary

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



Relative Relocation Constant - not applicable to types ER, PR and Null (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 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

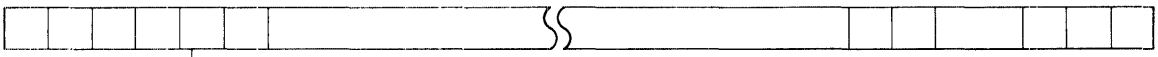
PC delete - Bit 3 = 1 -- indicates that this unnamed control section is a SEGTAB or ENTAB.

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

High ID Table

Built and referred to by Intermediate Output Processor



CESD entry number - 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)

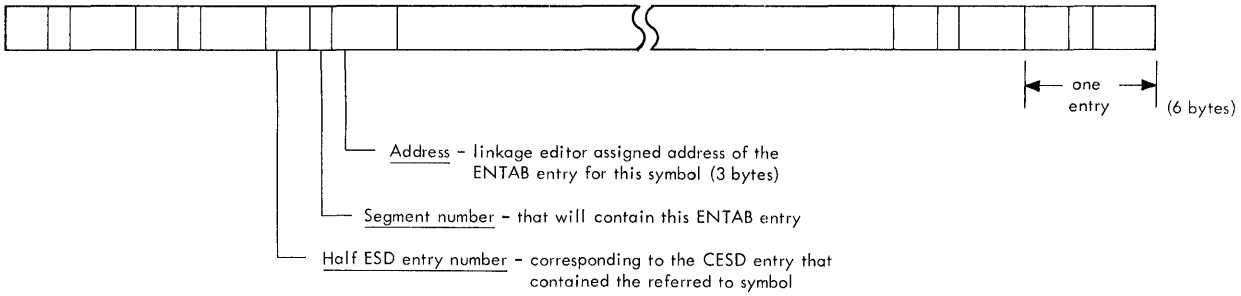
Note: If segment does not contain text, its corresponding entry contains zero.

REFERENCE DATA FOR SECOND PASS PROCESSING -- LEVEL E

The following tables are produced during second pass processing by the level E version of the linkage editor.

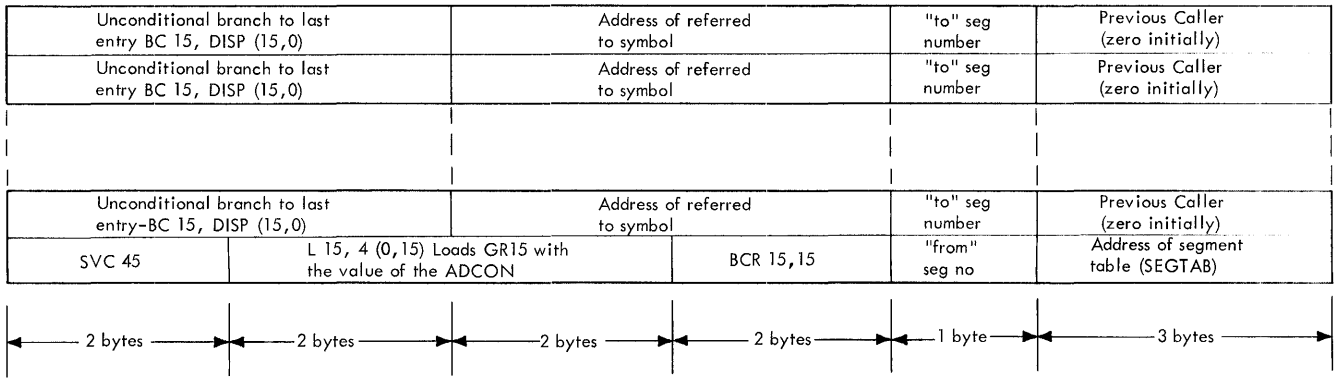
Entry List

Built by and referred to by Second Pass Processor



Entry Table (ENTAB)

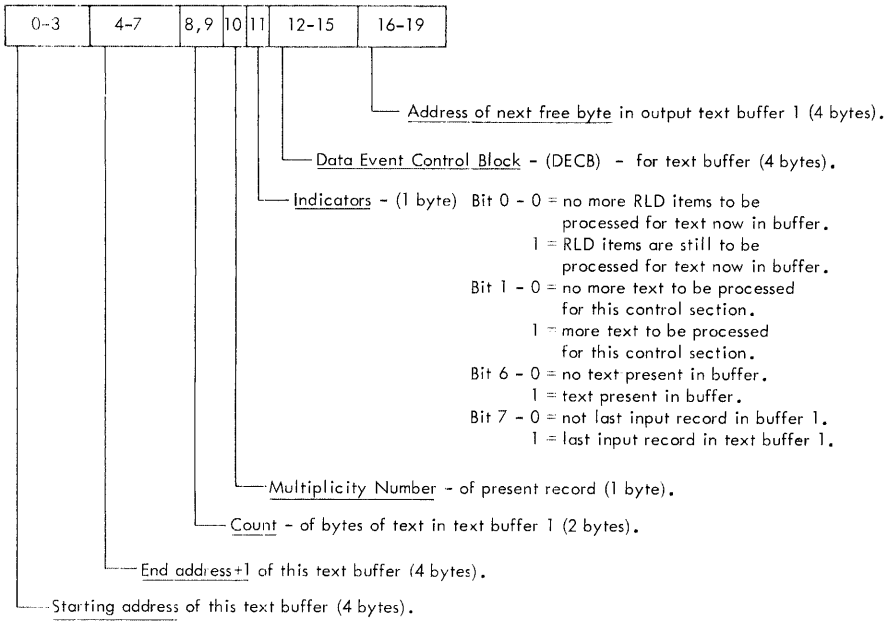
Built by Second Pass Processor



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.

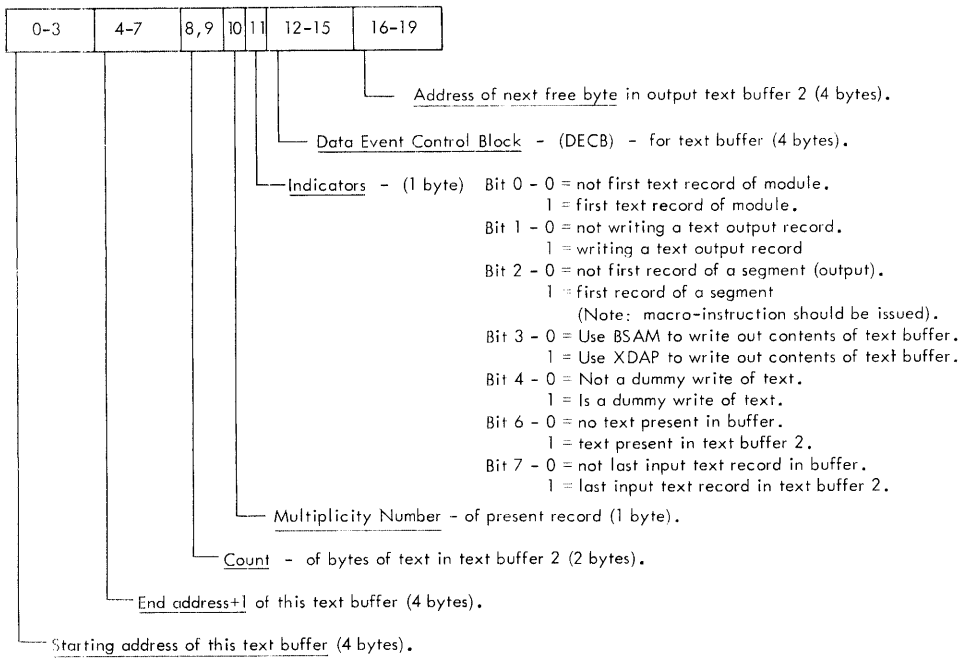
Text Table I

Built and referred to by Second Pass Processor



Text Table II

Built and referred to by Second Pass Processor

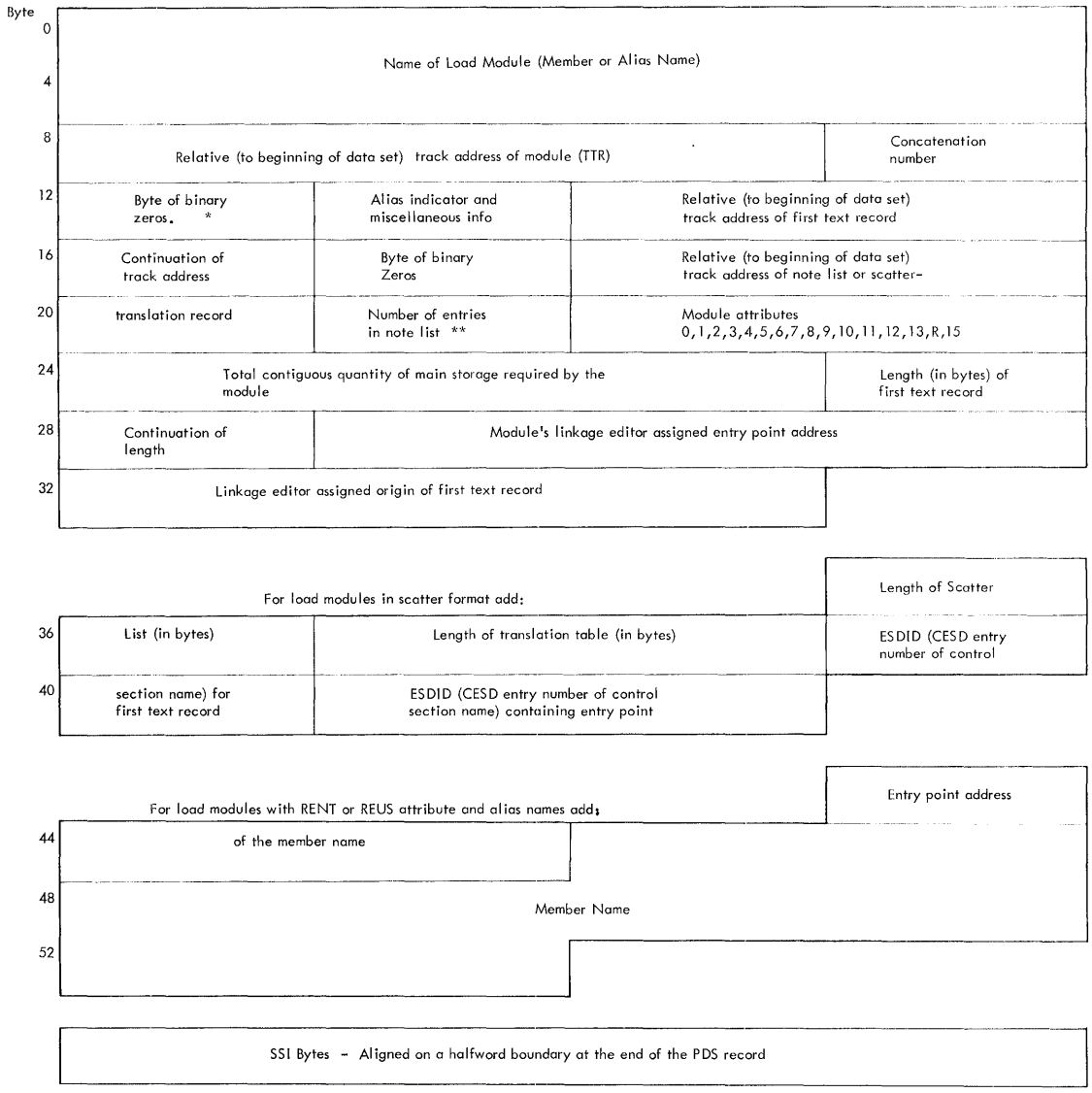


REFERENCE DATA FOR FINAL PROCESSING -- LEVEL E

The following reference data is used during final processing in the level E version of the linkage editor.

● Partitioned Organization Directory Record

As Received From BLDL



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

Module Attributes

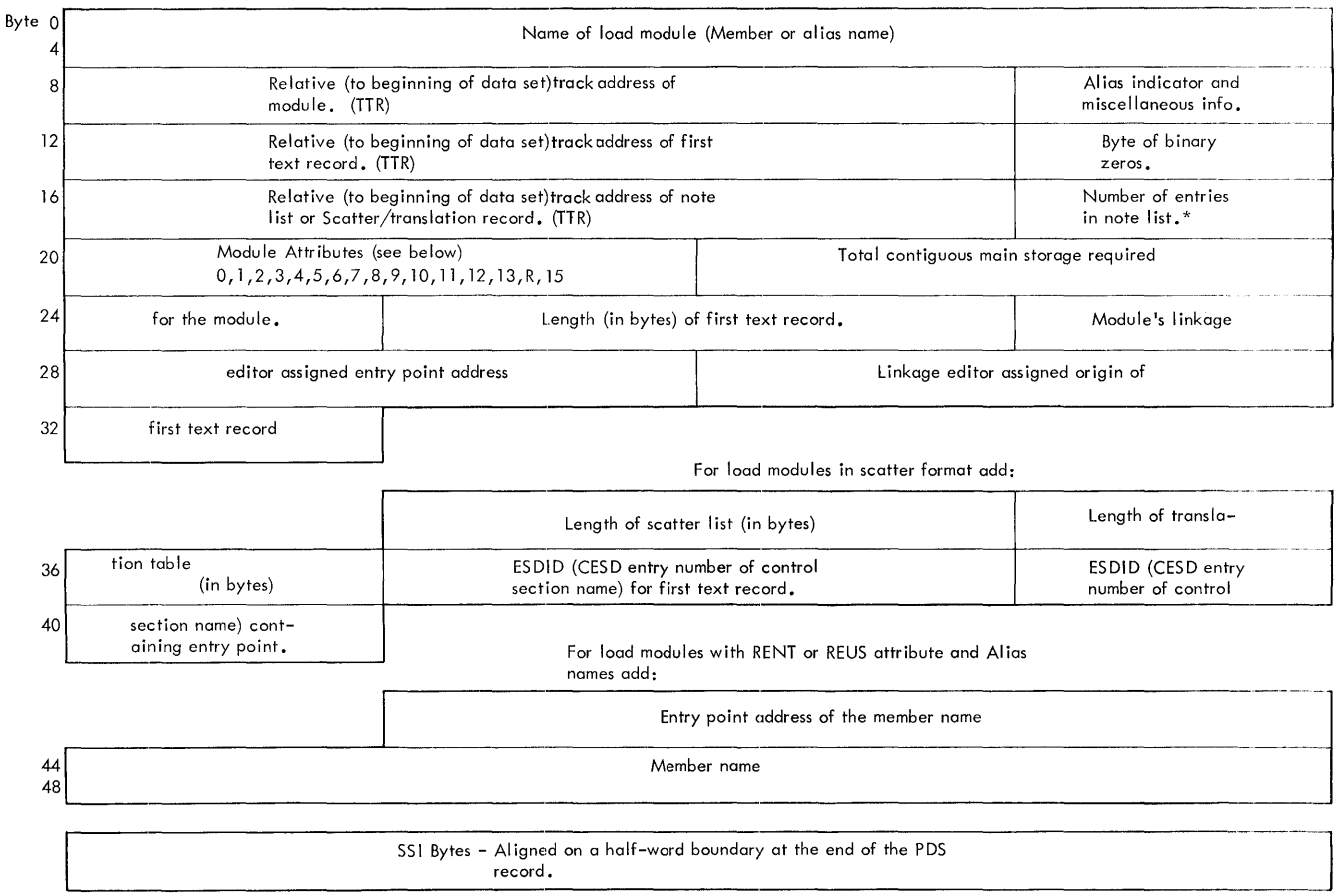
Bit Number	Attributes	Bit Setting	Indication
0	RENT	0	Not re-enterable
		1	Re-enterable
1	REUS	0	Not reusable
		1	Reusable
2	OVLY	0	Not an overlay module
		1	Overlay module
3	TEST	0	Not under test
		1	Under test
4	LOAD	0	Not only loadable
		1	Only loadable*
5	Format	0	Block format
		1	Scatter Format
6	Executable	0	Not executable
		1	Executable
7	Format	0	Module contains more than one text record and/or RLD record.
		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	Refreshable	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.

● Partitioned Organization Directory Record

As built by linkage editor

As built by linkage editor



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 record format shown above is the same as the corresponding record format for linkage editor F.

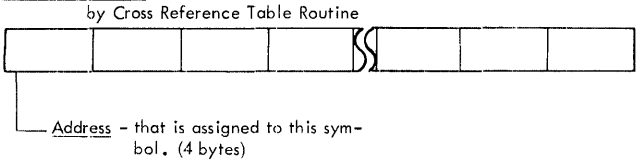
PODS Directory Record size:

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

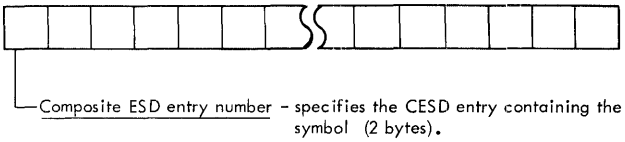
R=Reserved
*This byte contains zero if load module is not in overlay.

XADDCESD Table - built and referred to by Cross Reference Table Routine



Address - that is assigned to this symbol. (4 bytes)

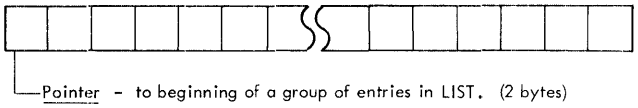
XAD2CESD Table - built and referred to by Cross Reference Table Routine



Composite ESD entry number - specifies the CESD entry containing the symbol (2 bytes).

Note: There is a one-to-one correspondence between entries in the above tables.

TABLE - referred to by IEWLCBPT.



Pointer - to beginning of a group of entries in LIST. (2 bytes)

LIST - referred to by IEWLCBPT.

End of Message Indicator - delimits a group of entries that define a message. (1 byte - hex FF)

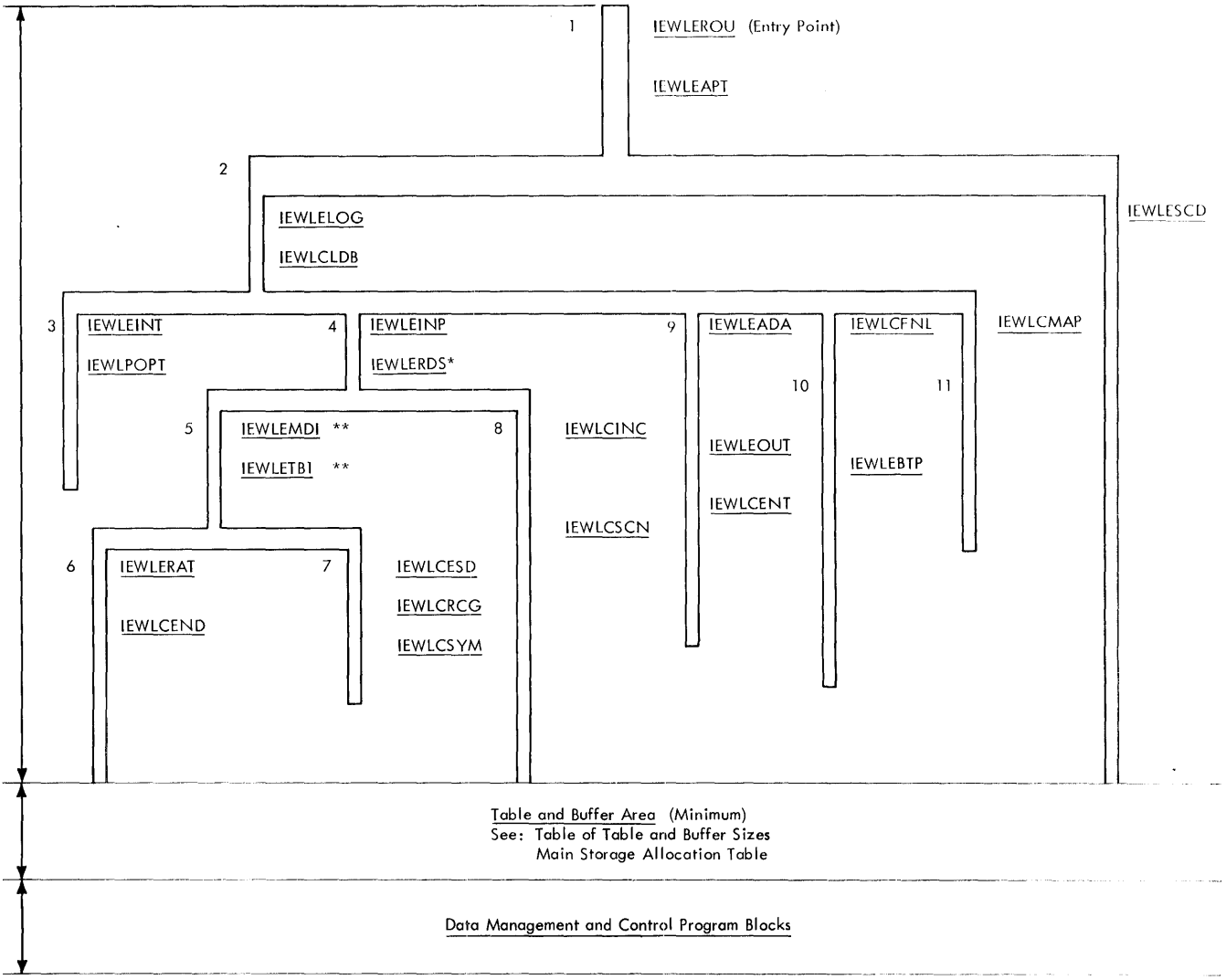
Pointer - to the first character of a phrase. (2 bytes)

Count-1 - of characters in the phrase. (1 byte)

OVERLAY TREE STRUCTURE -- LEVEL E

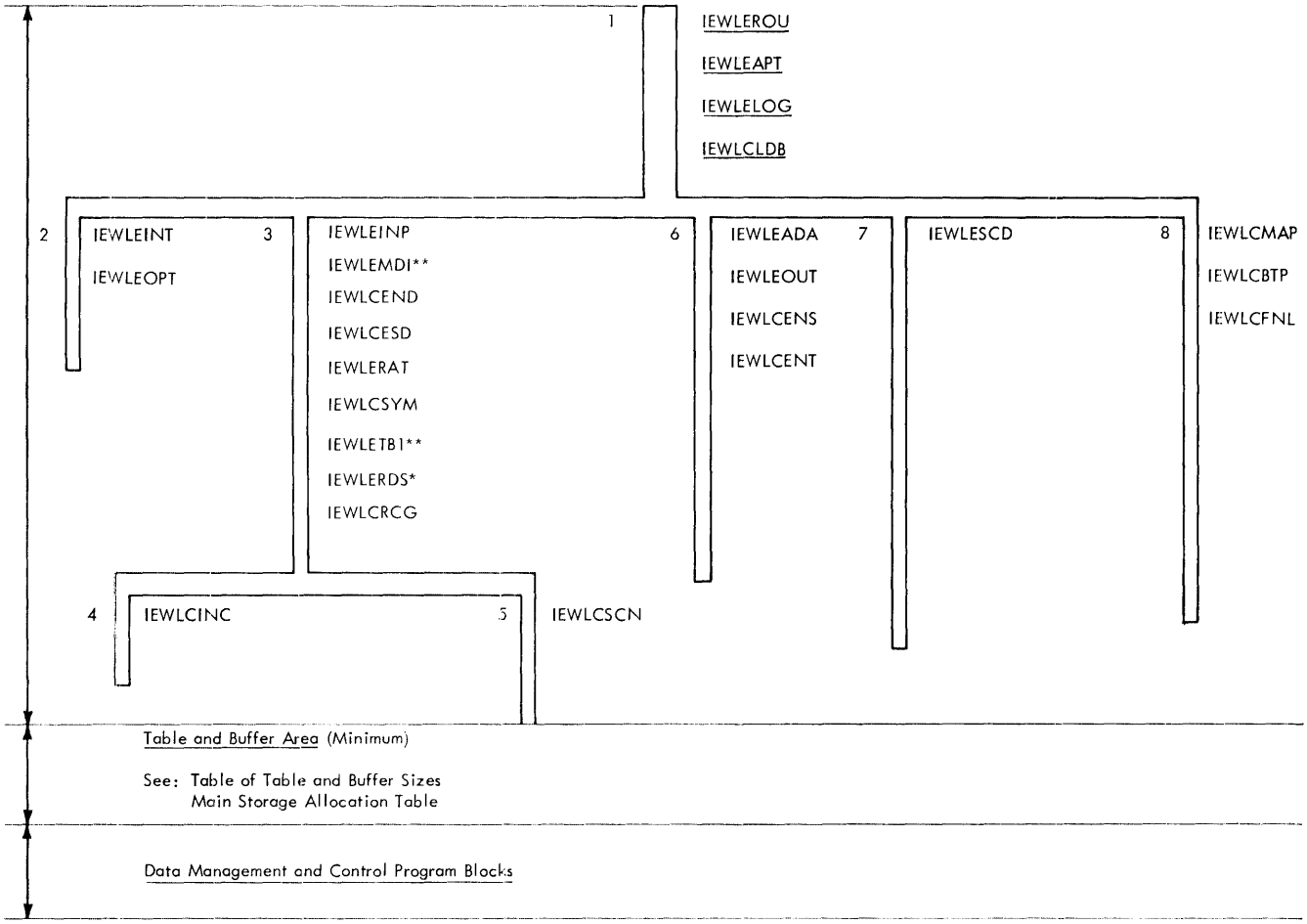
The following are the overlay tree structures for the 15K and 18K versions of the level E linkage editor.

LEVEL E LINKAGE EDITOR - 15K OVERLAY TREE STRUCTURE



* Csect within IEWLELOG
** Csects within IEWLEINP

LEVEL E LINKAGE EDITOR -- 18K OVERLAY TREE STRUCTURE



* Csect within IEWLELOG
** Csects within IEWLEINP

Object Module -- Control Section Cross Reference Table

Module Name	CSECT Name
IEWLCBTP	IEWLCBTP
IEWLCEND	IEWLCEND
IEWLCENS	IEWLCENS
IEWLCENT	IEWLCENT
IEWLCESD	IEWLCESD
IEWLCFNL	IEWLCFNL
IEWLCINC	IEWLCINC
IEWLCLDB	IEWLCLDB
IEWLCMAP	IEWLCMAP
IEWLCRCG	IEWLCRCG
IEWLCSCN	IEWLCSCN
IEWLCSYM	IEWLCSYM
IEWLEADA	IEWLEADA
IEWLEAPT	IEWLEAPT
IEWLEINP	IEWLEINP
	IEWLEMDI
	IEWLETB1
IEWLEINT	IEWLEINT
IEWLELOG	IEWLELOG
	IEWLERDS
IEWLEOPT	IEWLEOPT
IEWLEOUT	IEWLEOUT
IEWLEROU	IEWLEROU
IEWLESCD	IEWLESCD
IEWLETXR	IEWLERAT

General Register Contents at Entry to Modules -- Level E

Module Entry Point	Register Contents
IEWLCBTP	2 -- Address of all purpose table 13 -- Address of APT register save are (REGSA) 14 -- Return address 15 -- Entry point address
IEWLCEND	2 -- Address of all purpose table *3 -- Address of entry point, if present *4 -- CSECT length from END card, if present *5 -- ID of absolute entry point on END card, if present *6 -- Address of symbolic name, if present 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 -- Entry point address (IEWLCEND)
IEWLCENS	2 -- Address of all purpose table 13 -- Save area address 14 -- Return address 15 -- Entry point address

(Continued)

General Register Contents at Entry to Modules -- Level E (Continued)

Module Entry Point	Register Contents
IEWLCENT	2 -- Address of all purpose table 13 -- Save area address 14 -- Return address 15 -- Entry point address
IEWLCESD	2 -- Address of all purpose table 4 -- Byte count of ESD items to be processed *5 -- ID of first ESD item input *6 -- Address of first ESD item to be processed 7 -- Pointer to address specified within IEWLEMDI 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 -- Entry point address
IEWLCFNL	2 -- Address of all purpose table 15 -- Entry point address: IEWLCFNL or IEWLCFAB for normal processing; IEWLFSER for SYNAD exit
IEWLCINC	2 -- Address of all purpose table 13 -- Address of APT register save area (REGSA) 15 -- Entry point address
IEWLCMAP	2 -- Address of all purpose table 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 -- Entry point address
IEWLCSCN	1 -- Input record buffer address 2 -- Address of all purpose table 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 -- Entry point address
IEWLCSYM	2 -- Address of all purpose table 4 -- Byte count of TESTRAN data to be processed 6 -- Buffer address 7 -- Pointer to address specified within IEWLEMDI 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 -- Entry point address
IEWLEADA	2 -- Address of all purpose table 13 -- Address of APT register save area (REGSA) 15 -- Entry point address
IEWLEINP	2 -- Address of all purpose table 13 -- Address of APT register save area (REGSA) 15 -- Entry point address
IEWLEINT	1 -- Address of parameter list (first half word of parameter field is length of field, right justified) 13 -- Save area address 14 -- Return address 15 -- Entry point address

(Continued)

General Register Contents at Entry to Modules -- Level E (Continued)

Module Entry Point	Register Contents
IEWLELOG	0 -- Error code: bits 0-15 0 bits 16-19 disposition (1,2,3) bits 20-23 severity (1,2,3,4) bits 24-31 message number 1 -- Address of first symbol to be printed (optional) 13 -- Address of second symbol to be printed (optional) 14 -- Return address 15 -- Entry point address
IEWLEMDI	2 -- Address of all purpose table 13 -- Pointer to address specified within IEWLEMDI 15 -- Address specified within IEWLEMDI
IEWLEOPT	1 -- Address of parameter list (first half word of parameter field is length of field, right justified) 2 -- Address of all purpose table 13 -- Address of APT register save area (REGSA) 14 -- Return address (INT20A in IEWLEINT) 15 -- Entry point address
IEWLEOUT	2 -- Address of all purpose table 13 -- Save area address 14 -- Return address 15 -- Entry point address
IEWLEROU	1 -- Address of parameter list (first half word of parameter field is length of field, right justified) 13 -- Save area address 14 -- Return address 15 -- Entry point address
IEWLESCD	2 -- Address of all purpose table 13 -- HESD address of next ENTAB 15 -- Entry point address
IEWLETXR (Text Processing)	2 -- Address of all purpose table *3 -- Assembled address of first byte of text record *4 -- Byte count of text record *5 -- ID of text record 6 -- Storage address of this input text record 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 --Entry point address (IEWLERAT)
IEWLETXR (RLD Processing)	2 -- Address of all purpose table *4 -- Byte count of RLD record 6 -- Storage address of this RLD record 13 -- Address of APT register save area (REGSA) 14 -- Return address 15 -- Entry point address (IEWLERAT)

*Pertains to editor input.

Table Construction and Usage -- Linkage Editor E

TABLE	BUILT BY	USED AND/OR MODIFIED BY
Alias Table	IEWLCENT	IEWLCFNL
All Purpose Table	IEWLEINT	**
Calls List	IEWLERAT	IEWLCENS
CESD	IEWLCESD	IEWLERAT,IEWLCSCN,IEWLCINC,IEWLCAUT, IEWLCENS,IEWLCENT,IEWLEOUT
Delink Table	IEWLCESD	IEWLERAT
Downward Calls List	IEWLCENS	*
Entry List	SCDRELOC	*
Entry Table (ENTAB)	IEWLESCD	*
Half ESD	IEWLEOUT	IEWLESCD,SCDRELOC
Hafl ESD Prefix	SCDRELOC	*
High ID Table	IEWLEOUT	*
Relocation Constant Table	IEWLEADA	IEWLEOUT
Renumbering Table	IEWLCESD	IEWLERAT
RLD Note List	IEWLERAT	IEWLEOUT,IEWLESCD
Scatter Table	IEWLEOUT	*
SEGLGTH Table	IEWLEADA	*
SEGTAB	IEWLEOUT	SCDRELOC,IEWLCENS
Text Table I & II	IEWLESCD	*
Text I/O Table	IEWLERAT	IEWLEOUT,IEWLESCD
Text Note List	IEWLERAT	IEWLEOUT,IEWLESCD
Translation Table	IEWLEOUT	*
TTR List (TXT I/O Control Table)	IEWLERAT	IEWLEOUT,IEWLESCD
XADDCESD Table	IEWLCMAP	*
XAD2CESD Table	IEWLCMAP	*

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

IBM Technical Newsletter

File Number	S360-31
Re: Form No.	Y28-6610-2
This Newsletter No.	Y28-6400
Date	July 23, 1969
Previous Newsletter Nos.	Y28-2356 Y28-2301

IBM System/360 Operating System
Linkage Editor (E)
Program Logic Manual

This Technical Newsletter, a part of Release 18 of the System/360 Operating System, provides replacement pages for the Linkage Editor (E) Program Logic Manual, Form Y28-6610-2. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are listed below.

Cover, Preface
Contents, Illustrations
17,18
55,56,56.1
89,90,90.1
101,102
Index

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol • to the left of the caption.

Summary of Amendments

This amendment describes the improved error handling facility which uses the SYNADAF macro instruction, and corrects minor errors.

Please file this cover letter at the back of the publication to provide a record of changes.

IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, N.Y. 10020

PRINTED IN U.S.A.

Restricted Distribution

IBM Technical Newsletter

File Number S360-31
Re: Form No. Y28-6610-2
This Newsletter No. Y28-2356
Date November 15, 1968
Previous Newsletter Nos. Y28-2301

IBM SYSTEM/360 OPERATING SYSTEM
LINKAGE EDITOR (E)
PROGRAM LOGIC MANUAL

This Technical Newsletter, a part of release 17 of the System/360 Operating System, provides replacement pages for the Linkage Editor (E) Program Logic Manual, Form Y28-6610-2. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are listed below.

Contents
11-14.1
17-18.1
31-34.1
39-44.1
81,82
101,102
105,106
Index

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol • to the left of the caption.

Summary of Amendments

This amendment describes how the linkage editor can produce a load module capable of being loaded by the control program into either processor storage or 2361 Core Storage.

File this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Systems Publications, P.O. Box 390, Poughkeepsie, N.Y. 12602

PRINTED IN U.S.A.

IBM Technical Newsletter

File Number S360-31

Re: Form No. Y28-6610-2

This Newsletter No. Y28-2301

Date January 31, 1968

Previous Newsletter Nos. None

IBM SYSTEM/360 OPERATING SYSTEM
LINKAGE EDITOR (E)
PROGRAM LOGIC MANUAL

This publication corresponds to Release 15 and contains amendments to the Linkage Editor E PLM publication. Replacement and/or supplemental pages to be inserted in the publication are noted below. Corrections and additions to text and/or illustrations are indicated by a vertical bar to the left of the text or illustration and a bullet (•) to the left of the illustration caption.

Pages to Be Inserted	Pages to Be Removed
Cover, Preface	Cover, Preface
Contents, Illustrations	Contents, Illustrations
7-14.1	7-14
17,18	17,18
21,22	21,22
45,46	45,46
49,50	49,50
57-58.3	57,58
89,90	89,90
101,102	101,102
121-130	121-141
Index	Index

Summary of Amendments

This amendment deletes information pertaining to the 44K version of the level E linkage editor, and describes modifications for automatic system recovery (ASR), an optional feature that may be included in model 65 configurations using the MFT or MVT versions of the operating system (pages 13, 17, 102, 121-123).

This amendment also provides:

- Corrections to Figure 2 (page 10), Figure 19 (page 46), and Figure 22 (page 50).
- A microfiche directory (pages 57-58.3).
- A table describing the contents of registers when modules are entered (pages 127-129).
- A table describing where tables are constructed and used (page 130).
- A reorganization of Table 2 (page 22).
- A note that the I/O conventions and record formats for linkage editor E and linkage editor F are the same (pages 90, 123).

Note: Please file this cover letter at the back of the publication. Cover letters provide a quick reference to changes and a means of checking receipt of all amendments.

IBM Corporation, Programming Systems Publications, P.O. Box 390, Poughkeepsie, N.Y. 12602

PRINTED IN U.S.A.

READER'S COMMENT FORM

IBM System/360 Operating System
Linkage Editor
Program Logic Manual

Form Y28-6610-2

- 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 S/360 Printed in U.S.A. Y28-6610-2

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]

IBM S/360 Printed in U.S.A. Y28-6610-2

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]