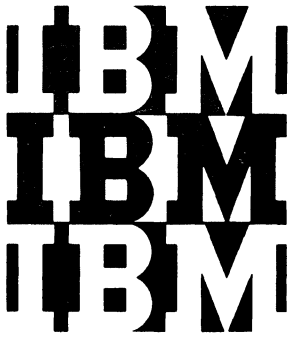


File No. S360-31
Order No. GY28-6714-1

Program Logic

IBM System/360 Operating System

Loader

Program Logic Manual

This publication describes the internal logic of the IBM System/360 Operating System Loader. The Loader functions as a processing program to combine and link input object and load modules in main storage and to pass control directly to the loaded program for its execution. This publication identifies areas of the program that perform specific functions and relates those areas to the program listing.

This publication is intended for persons involved in program maintenance, or system programmers who are altering the program design; it is not needed for normal use or operation of the program described.

The information on the Time Sharing Option (TSO) in this manual should be used for planning purposes only until such time as the option is available.

Second Edition (December, 1970)

This is a major revision of, and makes obsolete, Order No. GY28-6714-0 and Technical Newsletters GY28-2401 and GY28-6405. It contains information concerning a new Compiler/Loader interface, the new SYSTERM data set, and new Loader options. All changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition corresponds to Release 20 of the 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.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020.

SUMMARY OF AMENDMENTS -- RELEASE 20

Description	Pages
A new interface is defined between the Loader and a compiler which allows an internal data area to be used as input to the Loader. An open library data set can also be passed to the Loader.	9,10,12,14 16-19,22-24, 28-29,32,34-35, 40,42,43,78-80, 85,86,157-161
New main storage requirements for the Loader are given.	9
The Loader can be used with the Time Sharing Option (TSO).	9-10,17,18,40
The SYSTERM data set is added to the Loader data sets.	10,18,43,78,81, 82,85-89
The name of the diagnostic output data set is changed from SYSPRINT to SYSLOUT.	10,15,18,19,27, 43,78,81,84, 85-86,88
The name of the Load module processor is changed from IEWLOAD to IEWLODE.	77-79,81,84 85,86
The Loader can now identify the loaded program to the control program.	10,17,18,34,40, 42,43,82,84, 85-86,162-163
Two new options, TERM/NOTERM and NAME=, may be specified when using the Loader.	15,18,19,78,85, 86,88-89
The Loader processing portion can be invoked through a new entry point, IEWLOAD.	18,43,77
An exception is added to the statement that the CESD contains only one entry for each uniquely named text location.	24,32,73

This publication provides information describing the internal organization and logic of the Loader. It is part of an integrated library of IBM System/360 Operating System Program Logic Manuals. Other publications whose contents are required for an understanding of the Loader are:

IBM System/360 Operating System:

Introduction to Control Program Logic, Program Logic Manual, Order No. GY28-6605

Concepts and Facilities, Order No. GC28-6535

Linkage Editor and Loader, Order No. GC28-6538

Assembler Language, Order No. GC28-6514

The reader should also refer to the co-requisite publications:

IBM System/360 Operating System:

Storage Estimates, Order No. GC28-6551

System Control Blocks, Order No. GC28-6628

This publication has eight sections:

Section 1: Introduction. This section describes the Loader as a whole, including its relationship to the operating system. The major divisions of the program and the relationship between them are also described in this section.

Section 2: Method of Operation. This section provides: (a) an overview of, and an introduction to, the logic of the Loader, and (b) detailed descriptions of specific operations. Included are text and operation diagrams. The latter emphasize the flow of data within the Loader. The text and diagrams are correlated through a system of references. These references are of two levels. That is, the operation diagram for a function has an alphameric identification; within the diagram, specific points of reference have alphabetic labels. The text which describes the same function refers to the operation diagram as a whole and to the specific labeled references where appropriate. For example, the discussion of initialization refers to operation Diagram B1. Within the discussion, reference (B) refers to point (B) in Diagram B1. Also included in Section 2 are examples of the internal tables at strategic points in Loader processing. Both the diagrams and the table illustrations are designed as aids to quick recall.

Section 3: Program Organization. This section describes the organization of the Loader. Program components (modules, control sections, and routines) are described both in terms of their operation and their relation to other components. Flowcharts are included at the end of this section.

Section 4: Microfiche Directory. This section directs the reader to named areas of code in the program listing, which is contained on microfiche cards.

Section 5: Data Area Layouts. This section illustrates the layouts of tables and control blocks used by the Loader. These layouts may not be essential for an understanding of the basic logic of the program, but are essential for analysis of storage dumps.

Section 6: Diagnostic Aids. This section includes the general register contents at entry to program components, definitions of the internal error codes, and a list of serviceability aids available with the Loader.

Section 7: Appendix. This section includes input conventions, record formats, an Error Message/Issuer Cross-Reference Table and a description of the Compiler/Loader interface for passed data sets.

Section 8: Dictionary of Abbreviations and Acronyms. This section lists the expansions of abbreviations and acronyms used in the manual.

If more detailed information is required, the reader should see the comments and coding in the Loader program listing.

FIGURES

TABLES

CHARTS

DIAGRAMS

This section provides a general description of the Loader. Included are the purpose and functions of the program, its physical and environmental characteristics, and operational considerations necessary for its use. Also discussed in this section is a generalized theory of loading.

PURPOSE

The purpose of the Loader is to combine input object and load modules into an executable program in main storage. In this, the Loader performs the basic functions of the linkage editor and program fetch to obtain high-performance loading. (The Loader can be used only when special linkage editor processing, such as overlaying modules, is not required.)

Use of the Loader can provide advantages of increased system throughput and conservation of auxiliary storage space. System throughput can be increased through:

1. Elimination of scheduler overhead since loading and execution occur in a single job step.
2. Elimination of linkage editor I/O for intermediate and final output.
3. Elimination of certain linkage editor functions such as: control statement processing and overlay structuring.
4. Reduction of time required to read input through improved buffering techniques.
5. Reduction of time required for library search through use of link pack resident modules.
6. Elimination of time required to read input from an external device through use of an internal input data area prepared by a compiler.

Auxiliary storage space is conserved through:

1. Deferring inclusion of processor library routines until load time, thus reducing space required for the program. (This applies to a production environment where jobs are selected from a job library.)
2. Eliminating space needed for the linkage editor intermediate and output data sets.

FUNCTIONS

The Loader performs the basic logical functions of the linkage editor and of program fetch. Like the linkage editor, the Loader combines and links the input modules. In addition, the Loader assigns actual machine addresses to the resulting program and then passes control directly to the program for execution. In this, the Loader functions as does program fetch.

As part of the link-loading procedure, the Loader also automatically deletes duplicate copies of a module and can include modules from a system library.

MAIN STORAGE REQUIREMENTS

Loader operation requires about 21K bytes of main storage.[1] (This amount does not include the storage for the loaded program and the condensed symbol table.) The storage for Loader operation includes that for Loader code (about 14K bytes), for the data management access methods (about 4K bytes), and for Loader buffers and tables (about 3K bytes). Part of the Loader storage may be allocated from system storage if the access methods are resident and if the Loader code is resident in the link pack area. Figure 1 shows an example of Loader structure in main storage.

ENVIRONMENT

The Loader can be used with the PCP, MFT, and MVT options of the control program. The Loader can also be invoked under the Time Sharing Option (TSO) of the System/360

---

[1]The actual amount required depends on the type of input (e.g., input produced by the PL/I compiler requires a minimum of 10K bytes for Loader tables).

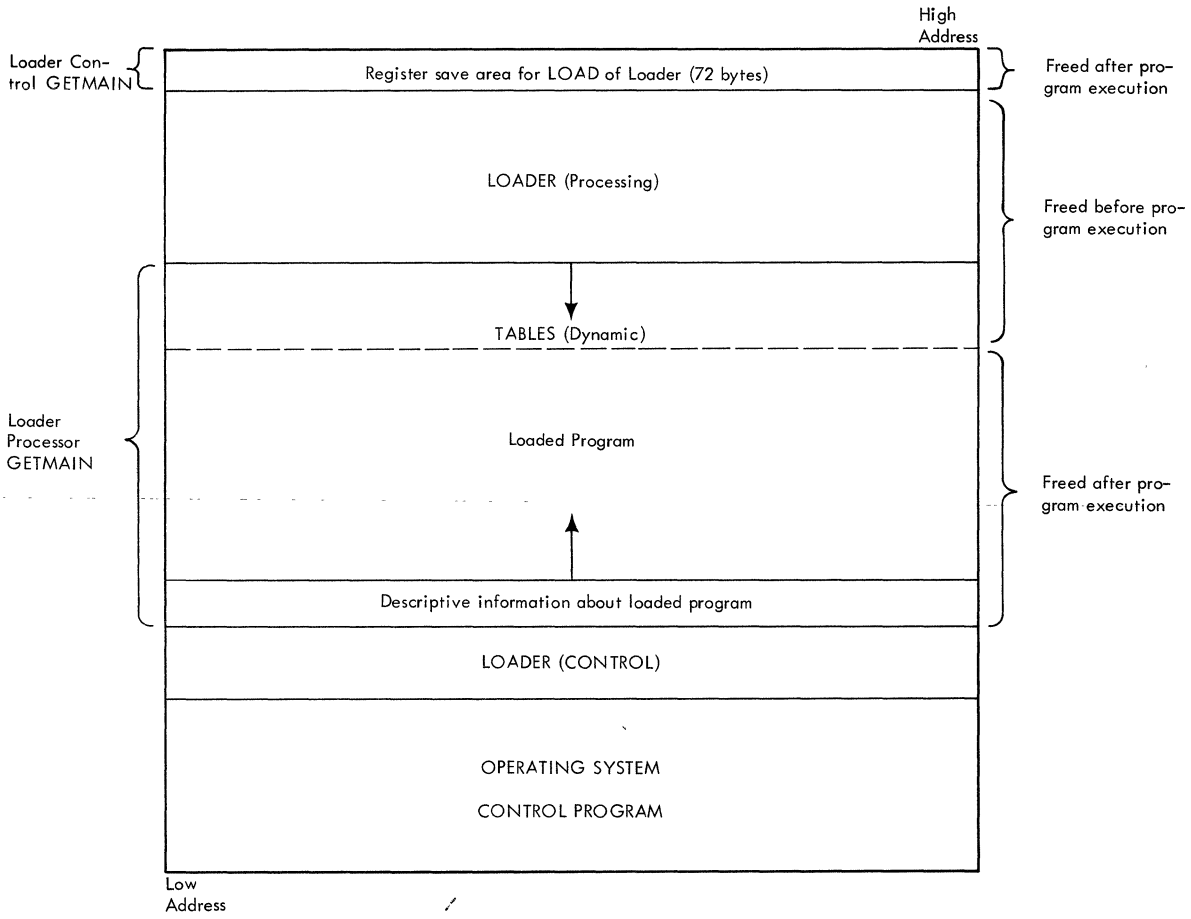


Figure 1. An Example of Loader Structure and Storage Map

Operating System. It can be used in one of three ways:

1. As a job step, when the Loader is specified on an EXEC job control statement in the input stream.

2. As a subprogram, via the 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 an ATTACH macro instruction.

Loader operation requires access to a primary input source, the SYSLIN data set. Input may be from a card reader, magnetic tape, a direct access device, or it may be a concatenation of data sets from different types of devices. Input may also be an internal input data area prepared by a compiler.

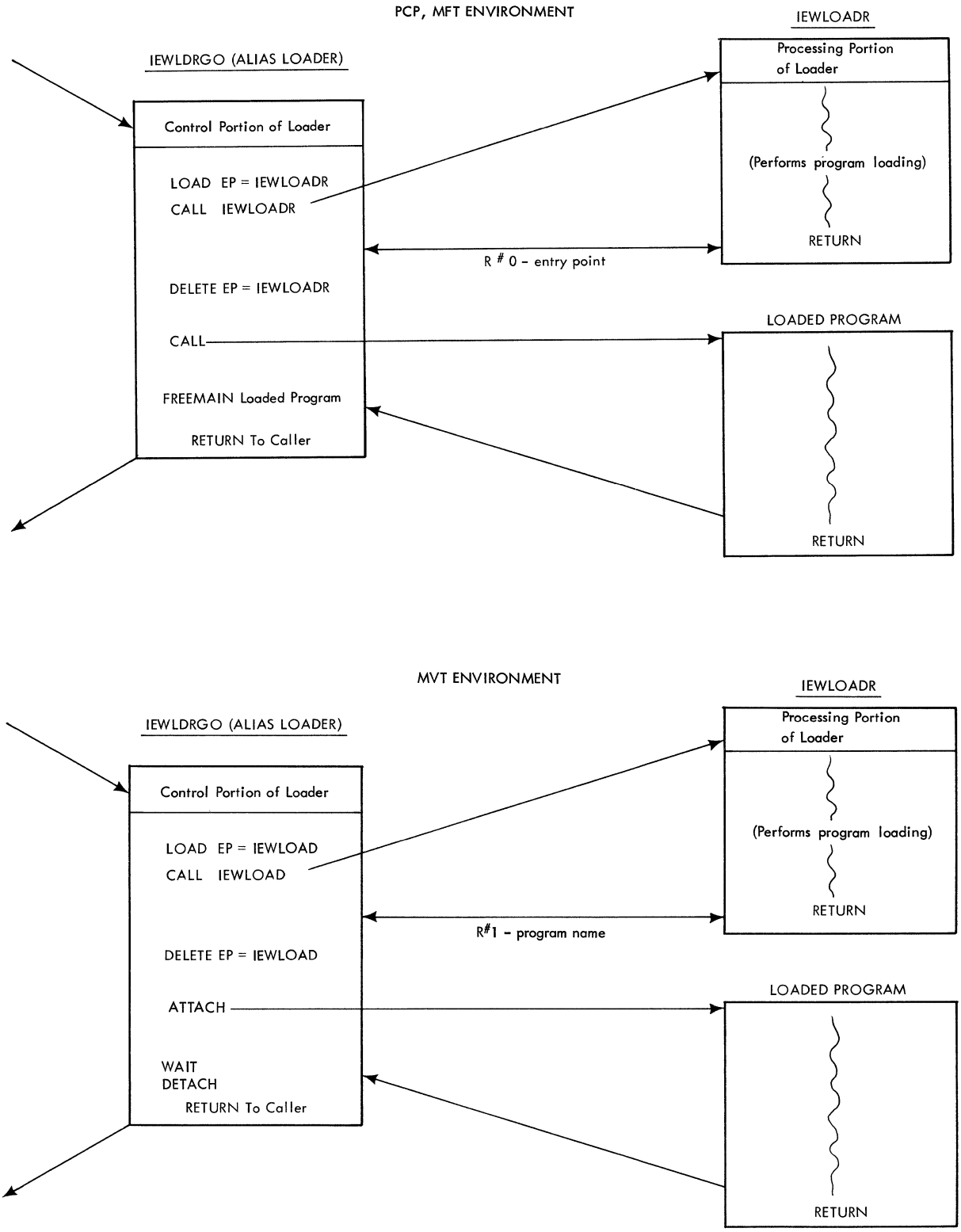
An automatic search of a system library can occur to complete the input. This requires use of the SYSLIB data set. It is defined only as a partitioned data set. SYSLIB may also be concatenated; however, SYSLIB input consists of object modules only or load modules only.

When the link pack area is available, the Loader can include in the loaded program resident modules listed in the contents directory entry queue.

The Loader uses the SYSLOUT data set for both diagnostic messages and module maps and the SYSTERM data set for diagnostic messages only. These data sets may be used in conjunction or separately.

PHYSICAL CHARACTERISTICS

The Loader consists of a control portion and a processing portion. The control portion handles linkages to and from the processing portion, which performs the actual program loading, and to and from the loaded program for its execution. The relationship between the portions of the Loader is illustrated in Figure 2.



• Figure 2. Loader Control Logic Flow

The Loader consists of two loads: the first is module IEWLCTRL, the control portion; and the other comprises control sections IEWLDDEF, IEWLIOCA, IEWLRELO, IEWLLIBR, and IEWLIDEN which together perform program loading. Because of the interrelationships among module functions, the Loader is not a candidate for overlay structuring.

OPERATIONAL CONSIDERATIONS

Loader operation is dependent on the type of input received and on user options that may be specified.

The input to the Loader may be load modules produced by the linkage editor and/or object modules produced by the following language processors: ALGOL, COBOL, FORTRAN, PL/I, the report program generator, and the assembler.[1] Input may be from an external device or it may be one or more internal object modules, that is, a data area which resides in main storage and consists of contiguous object module records. If input is an internal data area, the object module records containing the instructions and data of the program (text) can be omitted from the data area itself and replaced by passing a pointer to the text. The Loader then performs its usual functions of relocation and linkage on the text without having to read or move it.

If the Loader is processing an internal data area, input from an external device cannot be concatenated to it.

---

[1]If the input consists only of load modules, the user must specify the loaded program's entry point.

INPUT MODULE STRUCTURE

Object modules and load modules have basically the same logical structure (see Figure 3). Each consists of:

- Control dictionaries, containing the information necessary to resolve symbolic cross references between control sections of different modules.
- Text, containing the instructions and data of the program. If an internal object module is being processed, text prepared by a compiler may be omitted and replaced by a pointer to its location.
- End-of-module indication (END statement in object modules; EOM indicator in load modules).

The instructions and data of any module may contain symbolic references to specific areas of code. The symbols may be defined and referred to in the same module, or may be defined in one module and referred to in another. Thus, symbolic references are either internal or external with respect to the module in which they occur. A symbol which refers to external code is called an external reference (ER). External and internal references are made through address constants.

The Loader performs its function of changing all address constants to actual machine addresses by manipulating the input modules' control dictionaries.

Object modules usually contain two control dictionaries: an external symbol dictionary (ESD) and a relocation dictionary (RLD). An RLD is not present if the module contains no relocatable address constants.

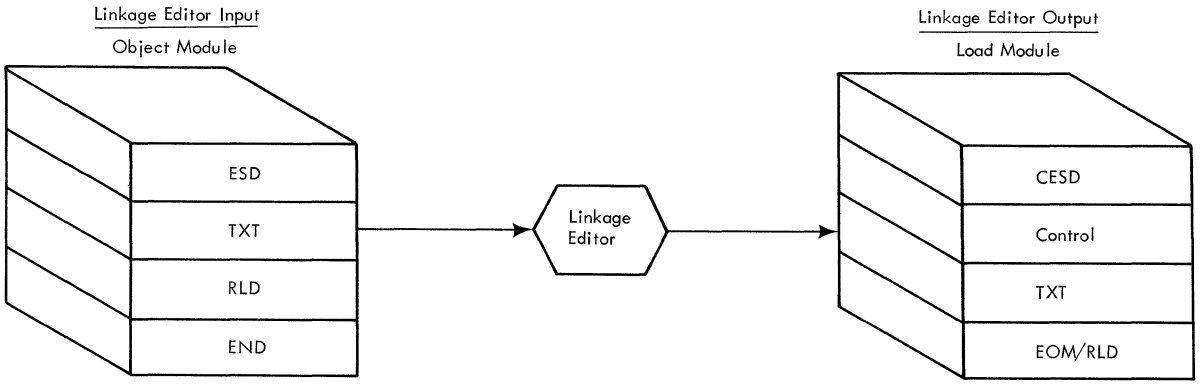


Figure 3. Object Module and Load Module Structure

Load modules are a composite of object modules, and, therefore, contain a composite ESD (CESD). Load modules contain RLDs also, unless there are no relocatable address constants. General descriptions of the control dictionaries follow. For detailed descriptions, see "Section 7: Appendix."

External Symbol Dictionary (ESD)

The external symbol dictionary contains entries for all external symbols defined or referred to within a module. Each entry indicates the symbol and its type and gives its position, if any, within the module. For example, there is an ESD entry for each control section, entry point, common area, and external dummy section. (An external dummy section defines a displacement within an area, obtained during execution of the input program via a GETMAIN macro instruction. External DSECTs are also referred to as pseudo registers.)

Relocation Dictionary (RLD)

The relocation dictionary contains at least one entry for every relocatable address constant (thus, for every external and internal reference) 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 determines the value of the address constant.

INTERRELATIONSHIP OF CONTROL DICTIONARIES

The control dictionaries and associated text are related through a system of numbers known as ESD identifiers (ESD IDs). An ESD ID is assigned to each external symbol according to its sequential appearance in an object module. The external symbol dictionary entries, as created by a compiler or assembler, have the same sequential order, so the ESD ID gives the dictionary entry number of an external symbol.[1] (The linkage editor renumbers the ESD IDs to maintain the ordered relationship when combining modules into a load module.)

Although the ESD IDs do not appear in the ESD entries, they are used in label definitions, text items, and RLD entries to refer to the symbols in the ESD.

In the RLD entries, the ESD IDs are used to show two relationships between the RLD and ESD entries, as follows:

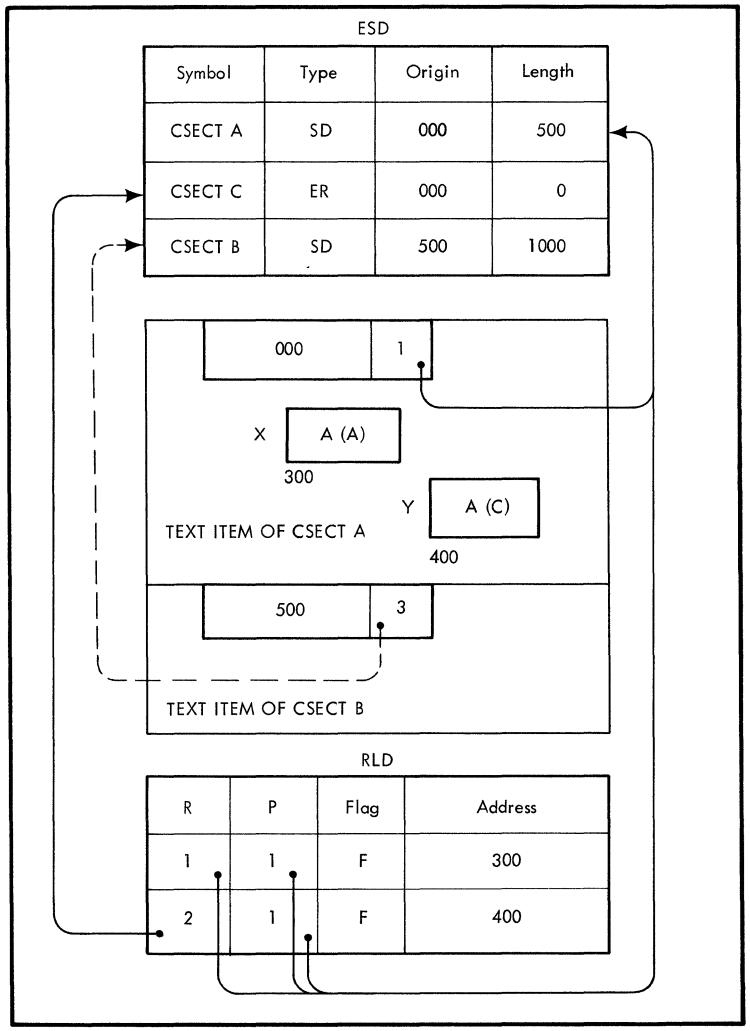
1. The RLD relocation pointer (R pointer) indicates the ESD ID for the symbol referred to by the address constant.

2. The RLD position pointer (P pointer) gives the ESD ID for the CSECT in which the address constant occurs.

Figure 4 illustrates the two cases of RLD pointers. The text of CSECT A contains two address constants, X and Y. X refers to a symbol within CSECT A. Therefore, both pointers of its associated RLD entry give the ESD ID of CSECT A. The value field of Y, however, refers to a symbol in a different control section, CSECT C. Thus, the R pointer of the entry for Y gives the ESD ID for CSECT C, the external reference; the P pointer gives the ESD ID for CSECT A.

---

[1] In an object module, an ESD item with type=LD can not have associated text or dependent address constants (see "ESD Processing") and so is excluded from the numbering system.



Note: The module above includes an external symbol dictionary, text, and a relocation dictionary. The entry in the ESD for CSECT C results from the reference to CSECT C in the text of CSECT A. This reference is at location 400. (CSECT B has no relocatable address constants.)

Figure 4. Example of an Input Module

LOADER OPTIONS

User options may be specified by parameters listed on the EXEC job control statement[1] or may be passed internally by a program requesting the Loader via LINK, LOAD, ATTACH, or XCTL macro instruction.[2] If the options are not user-specified, the defaults chosen through system generation are used.

If the options are passed internally, the user can also provide alternates for the standard ddnames and for the standard SYSLIN and SYSLIB DCBs.

Table 1 describes the Loader options. The parameters used are listed with the associated options. For some options, there are different parameters to specify either the choice or the refusal of the option. For example, NOCALL signifies that the library call option (CALL) is not to be used. (In this case, the third possible parameter has been retained for compatibility with the linkage editor option NCAL.) Table 1 also indicates the default options.

---

[1] See the publication IBM System/360 Operating System: Job Control Language Reference, Order No. GC28-6704.

[2] See the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Order No. GC28-6647.

• Table 1. Loader Options

Parameters	Options	SYSGEN Defaults
RES/NORES	The Loader searches the Link Pack Area queue for resident modules after primary input is complete, but before the SYSLIB data set is opened.	RES
MAP/NOMAP	The Loader produces a list of external names and their actual storage addresses.	NOMAP
CALL/ NOCALL/ NCAL	The Loader performs an automatic search of the SYSLIB data set for unresolved external names.	CALL
LET/NOLET	The Loader passes control to the loaded program despite the occurrence of a severity 2 error condition during loading.	NOLET
SIZE=	Specifies the maximum amount of dynamic storage to be obtained for loader processing.	SIZE=100K
EP=	Specifies an external name to be used as the entry point of the loaded program.	No SYSGEN default*
PRINT/ NOPRINT	The Loader attempts to open the SYSLOUT data set for diagnostic output.	PRINT
TERM/ NOTERM	Error messages are directed to the SYSTERM data set as well as the SYSLOUT data set.	No SYSGEN default**
NAME=	Specifies the name to be used as the name of the loaded program.	No SYSGEN default***

*The Loader assigns an entry point to the loaded program if none was specified.
**The Loader default is NOTERM.
***The Loader assigns the name **GO if none was specified.

GENERAL THEORY OF OPERATION

In processing the input modules, the Loader assigns main storage addresses to the control sections to be included in the loaded program and resolves external references in the CSECTs.

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.) The Loader assigns an address to the first control section and then assigns storage addresses, relative to this origin, to all other CSECTs.

Since cross references between CSECTs in different modules are symbolic, they are resolved (translated into machine addresses) relative to the main storage addresses assigned to the loaded program.

SECTION 2: METHOD OF OPERATION

This section describes the logic of the Loader. It contains an introduction which emphasizes the flow of primary data and control information through tables and buffers. This section also contains detailed functional descriptions of the Loader.

The logic introduction refers to the operation diagrams associated with a particular function. The detailed functional descriptions refer, through lettered references e.g., (A) to a portion of a diagram, to the corresponding steps of a function as shown in the operation diagrams. (The diagrams follow the text of this section.)

At the end of this section are illustrations of the internal Loader tables at strategic points in processing (Figures 14 and 15). These illustrations stress the changes to data as opposed to the diagrams, which stress movement of data. Used together, the two sets of figures offer quick recall.

STEPS OF THE LOADER OPERATION

The Loader control portion, which acts as an interface with the supervisor, loads the processing portion of the Loader and passes to it the parameter list received through the scheduler. The system interface is shown in Diagrams A1-A2. The Loader then performs loading through the following basic functions:

- Initialization
- Input control and buffer allocation
- Primary input processing
- Secondary input processing
- Final processing
- End of loading

After the processing portion has completed these functions, the Loader control portion passes control to the loaded program for execution.

The overall flow of data and control during loading is shown in Diagram A0.

Initialization (Diagram B1)

When the Loader begins processing, it performs initialization in preparation for all subsequent processing. The operations included in initial processing are:

- Analyzing control information
- Initializing main storage
- Initializing DCBs and opening data sets

Input Control and Buffer Allocation (Diagram C1)

The Loader reads input and allocates buffers as required for the current input module. Object modules from SYSLIN (primary input data set) and from SYSLIB (secondary input data set) are read into the object module buffers. (However, if input is an internal data area, buffers are not allocated and the data area itself is considered one buffer.) Control information from load modules (including ESD and RLD records) are read into the RLD buffer. Text from load modules is read directly into the loaded program's storage area.

Primary Input Processing (Diagrams D1, D2)

The Loader performs the following processing for all SYSLIN modules. (All overlay and scatter control statements from load modules and SYM records are ignored.)

External Symbol Dictionary Processing (Diagrams D3-D5)

The ESD records from object modules and CESD records from load modules describe symbols that have been defined for external use. The Loader makes entries for the symbols in the CESD and also makes entries in the translation table to allow the translation of the input ESD IDs to CESD addresses. The Loader calculates main storage addresses and stores them in the CESD entries.

Text Record Processing (Diagrams D7, D8)

For object modules, the Loader translates the ID of a text record to the proper CESD entry address. The CESD entry contains the storage address assigned to the CSECT. When the Loader finds the address for the text, it moves the text from the object module's buffer to the loaded program's storage. For load modules, the Loader translates the IDs of all CSECTs in a text record and thus finds their assigned main storage addresses. Then the Loader reads the record directly into the loaded program's storage area. When a CSECT read into main storage is to be deleted, the Loader adjusts the location of following CSECTs.

Relocation Dictionary Processing (Diagram D9)

The Loader builds its RLD table from information contained in the RLD records. The Loader processes the RLD records of object modules from the object module buffer and those of load modules from the RLD buffer. The Loader uses the relocation and position (R and P) pointers to determine the addresses of the adcons and uses the flag field to determine the method of address constant relocation required.

Address Constant Relocation Processing

When external references in the CESD are resolved, the Loader uses the RLD table entries chained to the CESD entry to relocate the related address constants in the loaded text.

Secondary Input Processing (Diagram E1)

If there are unresolved external references after all SYSLIN input has been processed, the Loader tries to resolve them from system library routines. If RES is specified, the Loader first tries to resolve the references from link pack area routines. When this is possible, the Loader uses the addresses of the referenced routines in the link pack area to resolve the adcons used to symbolically refer to them. Finally, the Loader opens the SYSLIB data set, if necessary. The Loader then loads any library modules that can be used to resolve ERs in the loaded program. The modules are located via the BLDL and FIND macro instructions. The Loader processes the modules, depending on whether they are object or load modules, in the same manner as it processes primary input.

Final Processing

After processing all the input for the loaded program, the Loader performs the following: assigns addresses for the common areas and for displacements in the external dummy section, issues messages for unresolved ERs, and determines the address of the loaded program's entry point.

Identifying Loaded Program[1]

If program loading is successful, the Loader issues an IDENTIFY macro instruction to pass the name of the program to be executed to the control program. At this time, a condensed symbol table may also be constructed for use during the program's execution by the test facilities available under the Time Sharing Option of the System/360 Operating System.

End of Loading

Before ending Loader processing, the Loader performs the following: writes out the diagnostic message dictionary and any remaining diagnostic messages, closes data set DCBs, sets up return information, and frees storage not required for the loaded program.

INITIALIZATION (IEWLIOCA)

When the Loader begins processing, it analyzes control information, performs initialization of main storage and of data sets, and allocates initial buffers for the data sets. See Diagram B1.

ANALYZING CONTROL INFORMATION

Loader operation depends on the control information consisting of the options, ddnames of the data sets, and the data control block addresses to be included in Loader processing. The Loader uses the information passed by the user or the defaults assigned by the system generator. (The defaults are contained in a control section assembled at system generation time. This CSECT is IEWLDDEF.)

---

[1]This processing is performed only when the Loader is used with the MVT option of the control program.

(A) To analyze the control information, the Loader obtains a temporary work area, INITMAIN. (See Section 5 for the contents of INITMAIN.) The Loader saves in the temporary work area the default ddnames and option indicators. An EXTRACT macro instruction is then issued to determine whether the Loader is currently operating under the Time Sharing Option, and an indicator is set in INITMAIN. If the processing portion of the Loader was invoked through the entry point IEWLOAD, another indicator is set to show that identification of the loaded program is desired. The Loader then scans the user's options and resets the default indicators in INITMAIN, when necessary.

If the SIZE option is specified, the associated user's value replaces the default value. However, if the option is incorrectly specified, the default value is used.

If the EP option is specified, the associated entry point name is saved in INITMAIN. There is no default entry point specified by the system generator.

If the NAME option is specified, the associated program name is saved in INITMAIN. Otherwise, the default name **GO is used.

The Loader then checks for user-specified ddnames to be used in specifying data sets. If present, these ddnames also replace the default names.

Finally, a check is made for the addresses of alternates for the data control blocks. A SYSLIN control block is accepted if it describes an internal data area. The address of this control block is saved and an indicator for an internal SYSLIN data area is set in INITMAIN. (The SYSLIN control block, which is not a data control block, is described in "Internal SYSLIN Control Block" under "Compiler/ Loader Interface for Passed Data Sets" in Section 7.) An alternate SYSLIB DCB is accepted if it describes a data set which has been opened. The address of this DCB is also saved and an indicator for an open library data set is set in INITMAIN.

INITIALIZING MAIN STORAGE

(B) The Loader obtains from the supervisor the storage required to process the input via the GETMAIN macro instruction. The request is conditional and variable. The maximum amount requested is that specified by the SIZE option; the minimum is 2K bytes. If the supervisor does not return storage, the Loader then issues an unconditional GETMAIN request for the minimum. If 2K bytes of storage is still unavailable, a system ABEND occurs.

If the supervisor returns main storage space, the Loader establishes its permanent communication area. (The communication area is described in Section 5.) The Loader then moves the information stored in INITMAIN to the communication area.

Save areas for use during loading are allocated and chained backward and forward. Finally, the INITMAIN area is returned to the system via a FREEMAIN macro instruction. The area is then available for data management functions required for loading.

READYING DATA SETS

(C) The Loader performs initialization requisite to use of its data sets. If the TERM option has been specified, space is reserved for a SYSTERM DCB, two DECBs, and two buffers. Unless an internal SYSLIN data set has been passed to the Loader, a SYSLIN DCB must be prepared and opened. Similarly, unless the NOPRINT option has been specified, a SYSLOUT DCB must be prepared and opened.

DCBs for the data sets are constructed using a model DCB contained in the Loader. The ddnames and basic attributes are placed into the constructed DCBs before the data sets are opened.

During opening, other data set attributes are checked. These include record format, record and block sizes, and the number of buffers to be allocated for the data set. If record and block sizes are not defined, the Loader uses the following defaults:

- For SYSLIN, both values are set to 80.
- For SYSLOUT, both values are normally set to 121. However, if the Loader is operating in time-sharing mode, the record length of the SYSLOUT data set is set to 81 so output can be easily directed to a terminal.

Since the Loader allocates buffers for its data sets, it does not require the buffer allocation supplied by the Open routine. The Loader indicates this by setting the DCBBUFNO field in the DCB to zero. The value that was found in the DCBBUFNO field is stored in DCBNCP.

The Loader determines whether the data sets opened successfully. If SYSLOUT is open, the Loader allocates the number of buffers and DECBs specified in the DCBNCP field in the DCB, and sets a flag indicating that the SYSLOUT data set is useable. The diagnostic output page heading is set up and printed. The Loader then constructs in the SYSLOUT buffer a list of the options used, the amount of main storage received for Loader processing, and the entry point and program names if specified. After printing this list, the Loader prints out any invalid options received and any errors encountered during the open procedure. Finally, if the MAP option was chosen, the MAP heading is constructed and printed.

If the opening of SYSLOUT was not successful, the MAP option indicator is set off and the storage allocated for the data set's DCB is released.

Next, the Loader determines whether the SYSLIN data set opened successfully. If an error occurred during opening of SYSLIN, loading is terminated. If SYSLIN opened properly, the Loader sets the "unlike attributes" indicator in the DCB to signify that SYSLIN may be a concatenation of data sets with unlike record formats. The buffers for the first input module are then allocated as described under "Buffer Allocation."

INPUT CONTROL AND BUFFER ALLOCATION

To read input, the Loader determines whether the current input consists of object or load modules and whether it resides on an external device or in main storage. This is indicated by indicators (CMFLAG3) in the communication area as well as the record format of the DCB. (The format is undefined (U) for load modules, fixed (F) for either object modules on an external device or internal object modules, and variable (V) for internal object modules.) If the input data set resides on an external device, buffers are allocated and primed. If the input data set is an internal data area consisting of internal object modules, no allocation or priming of buffers occurs and the data area itself is considered one buffer. In any case, the records are read and processed until the end of the current data set is recognized, either through the end-of-concatenation or end-of-file condition for a data set residing on an external device or through the end-of-buffer condition for an internal data area.[1] (No check for the END card or EOM indication is made during the reading procedure; the end condition is only recognized when the record is processed.) When the end of the current input is reached, the Loader checks for additional SYSLIN input[2]

Another data set in SYSLIN is indicated unless both the end-of-file and end-of-concatenation switches are on. When the Loader opens a new data set in SYSLIN input, the Loader determines the new attributes. This is accomplished by the same procedures used during Loader initialization for the first input data set.

BUFFER MANAGEMENT (IEWBUFFR)

In general, the Loader allocates storage individually for DECBs and buffers. Thus, for a single data set, buffer allocation actually consists of several separate allocations. These allocations are made from contiguous storage whenever feasible. All allocations are made from the highest available address in Loader processing storage. When no longer needed, allocated space is made available for subsequent modules.

Buffer Deallocation

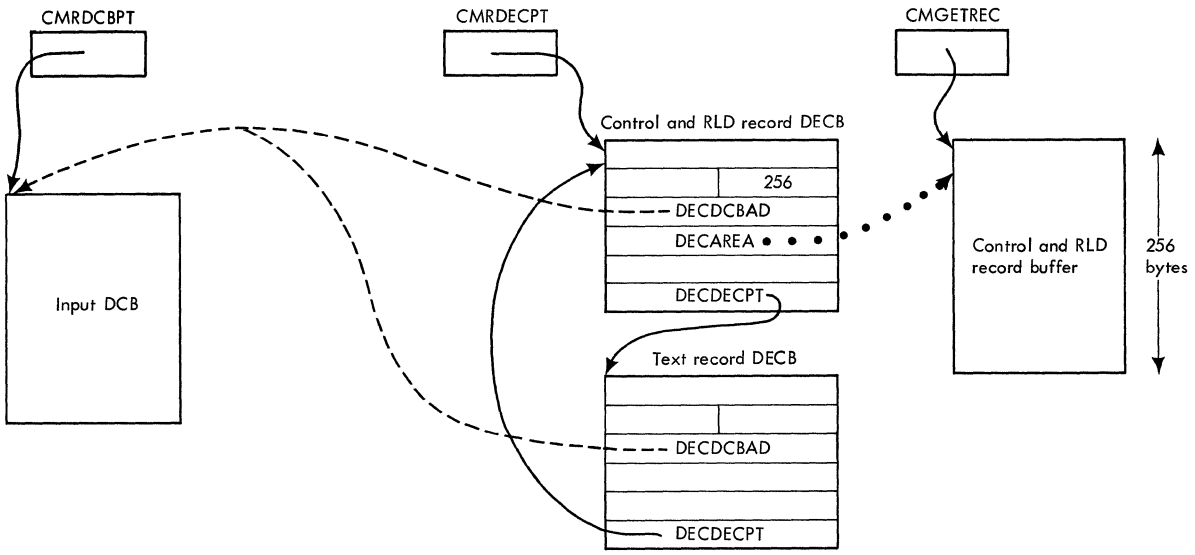
If both the current input and the previous input consist of load modules, the Loader uses the same buffer and DECBs. This is possible since the buffer-DECB requirement for load modules is constant. Figure 5 illustrates the buffer and DECBs required for reading load modules. If either the current or the previous data set consists of object modules, the Loader frees (deallocates) the storage used for the previous buffer-DECB allocation.

A pointer to the first freed area is maintained at CMFRECOR. (See Figure 6.) The first four bytes of each freed area are

---

[1]End-of-buffer signifies both end-of-file and end-of-concatenation for an internal data area.

[2]The end-of-concatenation switch is set during the data set opening if another data set is concatenated to the current one. If there is no other SYSLIN input, the end-of-concatenation and end-of-file switches are both set on. They are tested at the end of each module.



Note: CMRDCBPT, CMRDECPT, and CMGETREC are pointers in the communications area (IEWLDCOM).

Figure 5. Load Module Allocation

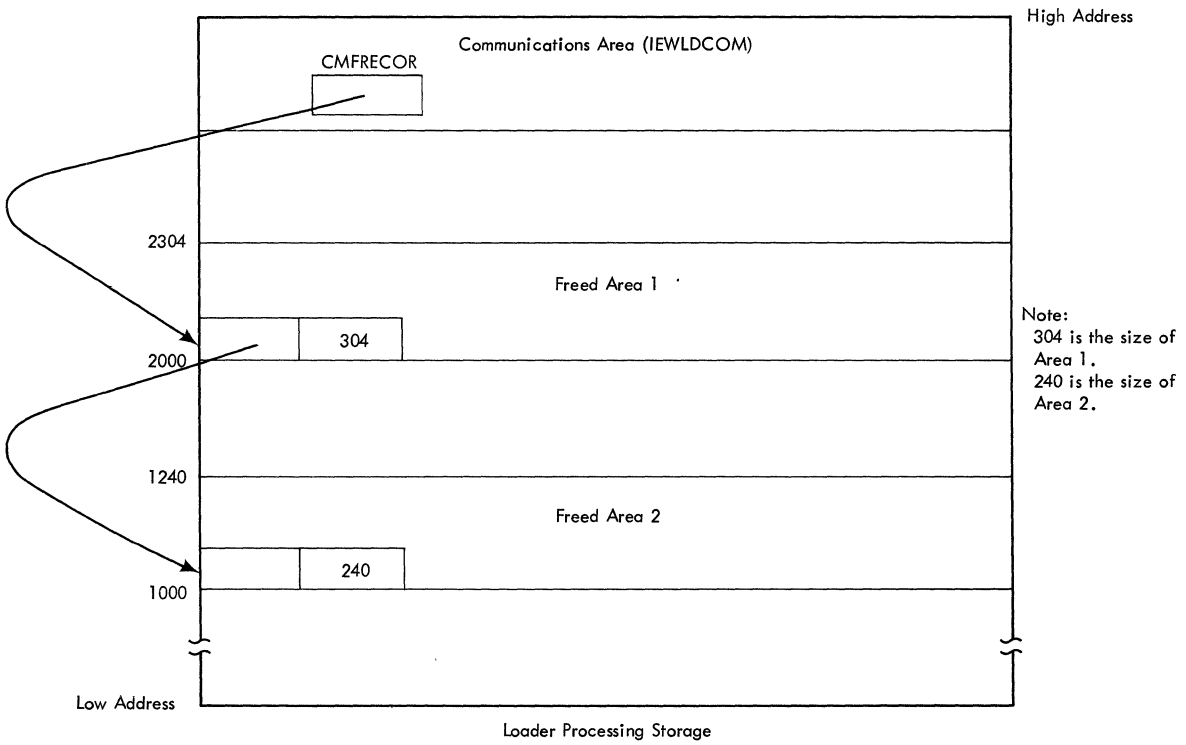


Figure 6. Freed Areas From Buffer-DECB Allocation

used to store a pointer to the next freed area in the chain. The second four bytes give the size of the current area. (The size is always rounded to doubleword value.) See Figure 6 for an illustration of freed area chaining.

Before chaining an area deallocated from a DECB or buffer, the Loader checks the area's location against the pointers of the other areas in the chain for contiguity. Contiguous freed areas are combined under a single pointer. For example, in Figure 6, Freed Area 1 could consist of areas from three separate deallocations: one for each DECB and one for the buffer.

Buffer Allocation

After freeing any previously used buffers, the Loader allocates DECBs and buffers for the current input module. For object module input, a DECB is allocated, cleared, and the address of the DCB is stored in it; then, the related buffer is allocated and its address stored in the DECB. (The size of the buffer is obtained from DECBBLKSI, the number from DCBNCP where the value from DCBBUFNO was stored.) The allocation procedure is repeated until the specified number of buffers has been allocated. However, after the first time, each DECB is chained to the one before. The last DECB is chained to the first. (See Figure 7 for an illustration of an allocation for object module input.) The Loader also sets a pointer to the DECB chain in the communication area at CMRDECPT, sets the I/O flags to indicate object module input, and saves the buffer size in the communication area for the later deallocation.

For load module input, the Loader allocates the required two DECBs, clears them, chains them together, and stores the address of the DCB in them. The required buffer, called the RLD buffer, is then allocated and its address stored in the first DECB. The Loader stores a pointer to this buffer in the communication area at CMGETREC, and a pointer to the first DECB in CMRDECPT. (No buffer is allocated for load module text which is read directly into the loaded program's storage area.) The RLD buffer size is stored in the DECB, and finally the I/O flags are set to indicate load module input.

In allocating buffers and DECBs for load or object module input, the Loader attempts to reuse any storage freed from previous allocations. The Loader examines each entry in the freed area chain to determine whether the related storage is sufficient for the current DECB or buffer.

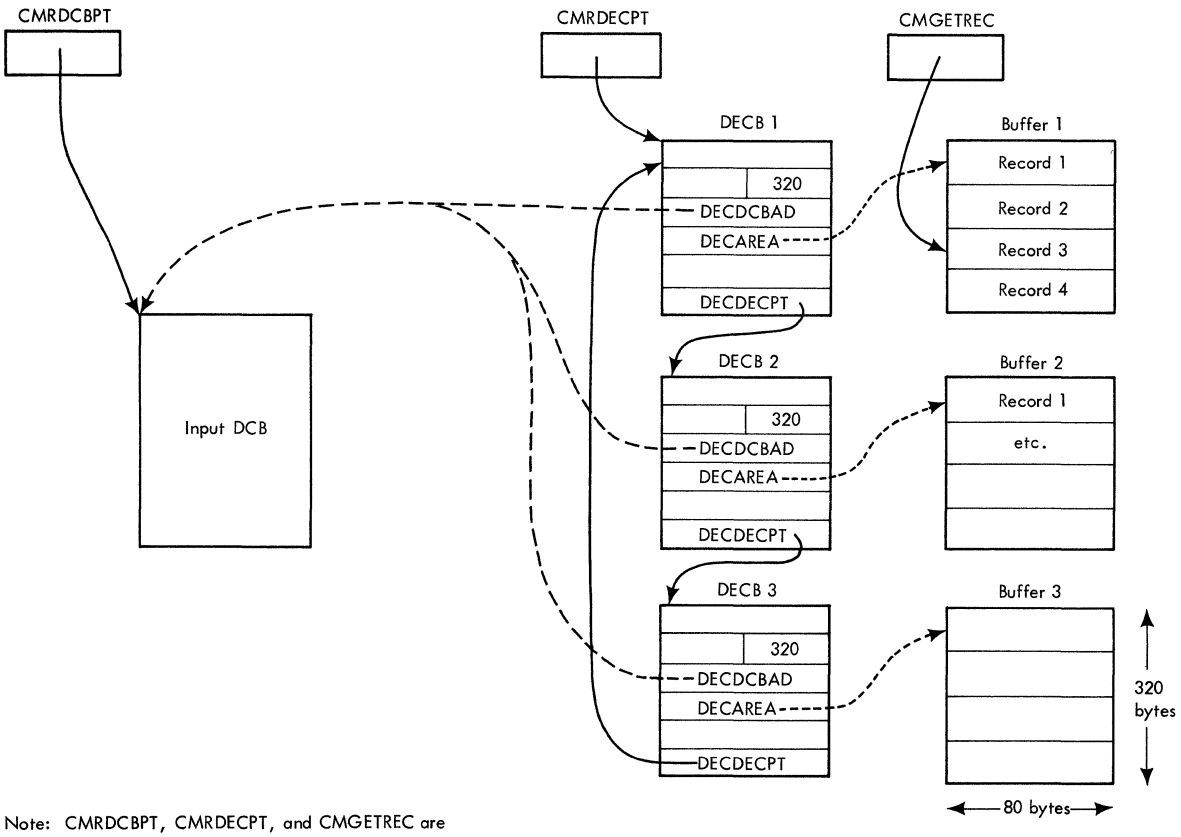
If the area is too small, the next entry is tested. If the size of an area equals the required size (rounded to doubleword value), the Loader unchains the area and constructs the buffer or DECB. If the freed area is greater than the required area, the chain pointer for that area is updated to show the size and location of the remainder.

If no area in the chain is adequate for the current buffer or DECB, the Loader makes the allocation from its processing storage not previously allocated (prime storage). If this allocation requires an area so large that it would overlap the loaded program's area, the loading process is terminated with a message printed to indicate that available storage was exceeded.

READING OBJECT MODULE INPUT FROM AN EXTERNAL DEVICE

Because of the fixed format of object module records, the Loader can initiate the reading of physical sequential blocks before they are actually needed for processing. To accomplish this, the Loader primes the buffers after allocating them for object modules. Priming consists of initiating READ macro instructions for all buffers except one. When the Loader requires the first record for processing, a READ macro instruction is issued for the unfilled buffer and a CHECK macro instruction is issued for the first buffer primed.

At the beginning of processing for a module, the DECB pointer (CMRDECPT) specifies the DECB associated with the first primed buffer. (See Figure 7.) The pointer to the current logical record also specifies the beginning of that buffer. As each record is processed, the Loader updates the logical record pointer to the next record. When all records in the buffer have been processed, the Loader updates the DECB pointer to the one for the next filled buffer, and issues a READ macro instruction for the completed buffer. The procedure is repeated until the end of the module is recognized.



Note: CMRDCBPT, CMRDECPT, and CMGETREC are located in IEWLDCOM. CMRDECPT points to the DECB/buffer being processed. CMGETREC points to the logical record being processed.

Figure 7. Allocation for Object Module Input

READING INTERNAL OBJECT MODULE INPUT

For internal object modules prepared by a compiler, record format may be fixed or variable. After initialization of the data area containing the internal object module records, the pointer to the current logical record points to the beginning of the data area. As each new logical record is requested, the Loader updates the pointer to the next record in the data area, using the DCBRECFM field in the SYSLIN control block to determine whether fixed or variable length records are being processed. The end of the module is recognized when the length of the processed records equals the length specified in the DCBBLKSI field. At this time, the end-of-file and end-of-concatenation switches are set on.

READING LOAD MODULE INPUT

For load modules, the record format is undefined, but the order in which record types may be processed is limited. For example, control records are required before the related text record can be read. All non-text records of load modules are read into the same buffer. This buffer, the RLD buffer, has the same length as the maximum length of non-text records processed by the Loader (256 bytes).

The Loader reads text records directly into the loaded program's assigned area; therefore, there is a DECB for reading text although no buffer is required. The Loader determines the address to receive the text during load module processing. At the time that a text record is read, the following record is also read, since that record is always non-text.

PRIMARY INPUT PROCESSING

After determining the current record type, the Loader performs one of the following types of processing for the primary input (object and/or load modules from the SYSLIN data set):

- External symbol dictionary (ESD) processing
- Text record processing
- Relocation dictionary (RLD) processing
- Address constant relocation processing
- End processing (including end-of-module and END card)
- MOD record processing

If an invalid record type is encountered, a diagnostic message is issued. In addition, if an internal input data area is being processed, the end-of-concatenation and end-of-file switches are set on so that no further input will be processed.

Table 2 shows the differences in processing for object and load modules. Input module processing for object and load modules is shown in Diagrams D1 and D2, respectively.

Load module record types include composite ESD, control, RLD, control/RLD, text, SYM, and scatter/translation. When the Loader recognizes a SYM or scatter/translation record, it simply ignores that record and requests another control record. Descriptions of those load module records processed by the Loader follow. (For detailed descriptions, see the record formats given in Section 7.)

- CESD: These records each contain no more than 15 ESD entries.[1] The first eight bytes give the following control information for the entries in that record: (1) the ESD ID of the first entry, and (2) the number of bytes occupied by the entries.
- Control: These records give control information about the module text on the following text record. Included are the related ESD IDs and the lengths of each control section in the following text record, and an indication of EOM, when pertinent. The control records also contain a channel command word (CCW) with the linkage editor-assigned relative address and total length of the text record. The Loader uses this information to read the text.

---

[1]The Loader can accept a maximum of 1024 ESD entries per input module.

Table 2. Object and Load Module Processing Differences

Type of Processing	Object Module	Load Module
ESD	1. Input is an ESD record. 2. The Loader performs preliminary processing for NULL, PC, and LD entries.	1. Input is a CESD record. 2. The Loader performs preliminary processing for SD, LR, PC, and NULL entries.
Text	The Loader processes text from the object module buffer one ID at a time.	The Loader reads text directly into loaded program's storage area after processing the entire ID/length list.
RLD	No Difference	
Relocation	No Difference	
End	The Loader processes the END statement for each CSECT, and performs end-of-module processing.	The Loader performs end-of-module processing.
MOD (internal object modules only)	The Loader determines the origin of the compiler-loaded text for the module and equates this address with what would normally be the Loader-assigned address.	Not Processed

- Text: These records contain the control sections with the instructions and data of the module. A text record can contain a maximum of 60 control sections.

- RLD: These records contain the RLD entries used to relocate address constants in the preceding text. When the text contains a large number of relocatable symbols, the related RLD entries may require several records.

- Control/RLD: These records combine a control and an RLD record into one physical block. They contain RLD entries related to a previous text record and the control information for the following text record.

The object module records, ESD, RLD, TXT, and END, contain information similar to that described previously. In addition, an internal object module can contain the MOD record. This record contains control information about the text of the module, which has already been loaded by a compiler or other text-generating processor. This information includes the main storage address of the text, the address of the byte following the estimated or actual end of the text, and optional extent information. If a MOD record appears as the first record of an internal object module, all succeeding text records are ignored until an END statement has been processed.

EXTERNAL SYMBOL DICTIONARY (ESD) PROCESSING (IEWLESD)

The Loader processes the input modules' external symbol dictionary (ESD) records to resolve the symbols used in internal and external addressing. Resolution insures that each named location in the text for the loaded program has a unique symbol.[1]

To resolve symbols, the Loader builds its composite ESD (CESD) from individual ESDs and CESDs in the input. The Loader's CESD entries are created as required during processing of the input entries. See Section 5 for a detailed description of CESD entries.

Because of ESD processing, the Loader's CESD contains only one entry for each uniquely named text location, regardless of the number of input ESD entries containing the symbol for that location.[2] For a single module, the Loader records multiple ESD entries for a symbol in the translation table.[3] Each entry in the translation table corresponds to one input ESD entry for a symbol and contains a pointer to the CESD entry for the symbol.

A translation table entry has the same position in the table as the identifying number (ESD ID) of the associated ESD entry. For example, if an input ESD entry has an ESD ID of three, its corresponding entry is the third one in the translation table. Using this relationship, the Loader converts input ESD IDs via the translation table into the appropriate CESD address.

The Loader's ESD processing depends on the function of each input entry. The function of an entry is identified by the type indication in the entry. Table 3 gives the function specified by each type indication. The table also indicates whether a particular type can occur in object and/or load module external symbol dictionaries.

When the Loader creates a CESD entry, it chains it to others with the same type indication. Then, in processing each new input entry, the Loader determines by searching the chains whether a CESD entry with the associated symbol already exists. (The Loader only searches those chains for types that could be related to the current input entry's type.) In certain cases, special preliminary processing is performed to delay or to bypass the CESD search.

ESD processing is shown in Diagrams D3-D5.

---

[1]Names for areas of private code or for external dummy section displacements need not be unique since they are treated specially. These are defined by PC and PR entries, respectively.

[2]The only exception involves control sections with identical names. In this case, two entries are kept in the CESD, one of which is flagged "delete."

[3]The Loader clears the translation table after processing each module.

Table 3. ESD Entry Types and Functions

Type	Function	Occurrence	Comments
SD (Section Definition)	Defines the beginning of a named CSECT.	Object & Load	
PC (Private Code)	Defines the beginning of an unnamed CSECT.	Object & Load	
PC (Private Code) marked "delete"	Defines the beginning of an unnamed CSECT not to be included in the loaded program. For example, a SEGTAB created by the linkage editor.	Load only	The delete indication means that the associated text and RLDs are to be deleted.
LD (Label Definition)	Defines a label by giving its location relative to the beginning of the CSECT containing the label.	Object only	The defined label cannot be referenced directly since the LD entry has no ESD ID. The Loader changes the type to LR in the CESD entry.
LR (Label Reference)	Defines a label by giving its location relative to the beginning of the CSECT containing the label.	Load only	An LR entry contains an ESD ID and can, therefore, be referenced by an RLD entry.
ER (External Reference)	Refers to a symbol not defined in the same module containing the reference.	Object & Load	
CM (Common)	Defines a common area whose main storage address is assigned during loading.	Object & Load	The area may be named or unnamed. An unnamed area is called "blank common."
PR (Pseudo Register)	Defines a displacement within an external dummy section.	Object & Load	The external DSECT defines the area obtained by the loaded program via a GETMAIN macro instruction.
NULL	Indicates that the entry is to be ignored.	Object & Load	Only one entry for NULL is made in the Loader's CESD.
WX (Weak External Reference)	Defines an external reference that is not to be resolved by automatic library call.	Object & Load	The Loader processes a WX entry as an ER entry with a "weak call" flag.

Preliminary ESD Processing

When the Loader processes load modules, it does not necessarily receive CESD entries in the same order as the linkage editor assigned the relative addresses. Therefore, no entries for symbols that define module text locations are processed until all entries for the module have been received.

The Loader delays the processing by placing on a temporary chain the CESD entries it constructs for the SD, LR, and PC (not marked "delete") entries. Before chaining an entry, the Loader places the ID and the segment number in the CESD entry.

The entries are chained in the order of their linkage editor-assigned addresses.

Besides the preliminary processing for load module location definitions, the Loader also determines whether an input entry type is NULL, PC, LD, LR, or WX. These entries, in both object and load modules, are handled as follows:

NULL
: The Loader does not perform a CESD search for NULL entries since these entries have no effect on ESD resolution. When the first NULL entry for a module is recognized, a CESD entry is created. This CESD entry is cleared and marked "delete". (See the CESD entry description in Section 5.) The Loader places a pointer to the entry in the communication area (CMNULCHN) and makes a translation table entry. See "Making a Translation Table Entry." For all following NULL entries, processing consists only of making a translation table entry which refers to the CESD entry pointed to by CMNULCHN.

PC
: The Loader does not perform a CESD search for PC entries since it treats them as unique. For each PC entry, the Loader creates a CESD entry. Processing continues as described under "No-Match Processing" for SD entries.

PC "delete"
: The Loader treats PC entries that are marked "delete" as NULLs.

LD and LR
: LD and LR entries depend on their related section definitions (SDs). Therefore, before performing the CESD search, the Loader inserts the CESD entry address for the SD in the LD or LR entry. The address is obtained by translating the SD ID contained in the LD or LR.

  If an object module is the input, it is possible (through physical rearrangement of an object deck) to receive an LD before the related SD. The SD's CESD entry address cannot be placed in the LD until the SD's entry is created. Whenever this occurs, the LD is placed on a temporary LD chain. At the end of each input ESD record, the temporary LD chain is processed to determine whether a required SD has been received. When the SD associated with an LD has been received, its CESD entry address is placed into the LD. The Loader then searches the CESD for a matching symbol.

WX
: The Loader treats WX entries as ER entries that are marked "weak call." The "weak-call" flag like the "never-call" flag specifies those external references that are not to be resolved by automatic library call. However, the following difference arises in match processing. If a WX entry matches an ER entry in the CESD, the "weak-call" flag is set off. If an ER entry with a "never-call" flag matches an ER entry in the CESD, the flag is left on.

CESD Searching

In general, an input ESD entry requires resolution processing. The Loader does this by searching the CESD for a matching symbol. To direct the search, the Loader uses two tables. These are:

- HIERTBLE, which specifies which CESD chains are to be searched for a particular entry type, and the order in which the chains are to be searched.
- CMTYPCHN, which contains the address of the first entry in each CESD chain.

Figure 8 shows the relationship between the two tables.

The Loader determines the type of an input ESD entry and begins to search the first chain specified by HIERTBLE. (If the type is LD, the Loader performs the search as if it were an LR.) The symbol from the input entry is compared to the symbol in each chained entry. If no matching symbol is found and end-of-chain is recognized, the next chain specified by HIERTBLE is searched.[1] If no matching symbol is found in any of the appropriate chains, a CESD entry for the symbol is created and chained. A translation table entry is also made, if appropriate. See "No-Match Processing." If a matching symbol is found, symbol resolution occurs. See "Match Processing."

---

[1]Whenever a new entry on a chain is examined, a pointer to that entry is stored in the communication area (CMPREVPT). Should the next entry on the chain be a match, the pointer at CMPREVPT is used to update the chain.

HIERTBLE

Input ESD Entry Type					
	SD	2	0	5	3
	LD	-	-	-	-
	ER	0	2	3	5
	LR	2	3	0	5
	PC	-	-	-	-
	CM	5	2	0	3
	PR	6	-	-	-
	NULL	-	-	-	-

Order of Type Chain Search →

CMTYPCHN

SD Chain Address	LD Chain Address	ER Chain Address	LR Chain Address	PC Chain Address	CM Chain Address	PR Chain Address	NULL Chain Address
0	1	2	3	4	5	6	7

Notes:

The HIERTBLE entries identify by number the CMTYPCHN entries. E.g., zero (0) in the HIERTBLE refers to the SD chain address in CMTYPCHN.

When more than one type chain can be searched for a symbol, the order is specified by HIERTBLE. E.g., if an input ESD entry is an SD, the HIERTBLE entry specifies that the ER, SD, CM, and LR chains are to be searched in that order.

Figure 8. Tables Used in the CESD Search

No-Match Processing

When a symbol is received for the first time, the Loader performs processing that depends on the type of the input entry for the symbol. This always includes the construction of the CESD entry, which differs by entry type. Except for LD entries, no-match processing also includes construction of a translation table entry.

If the user specified the MAP option, the Loader formats a map entry for each symbol (except ERs). See Section 5 for an example of map output. The Loader prints the map entries on the SYSLOUT data set.

Table 4 summarizes the processing performed for each input entry type.

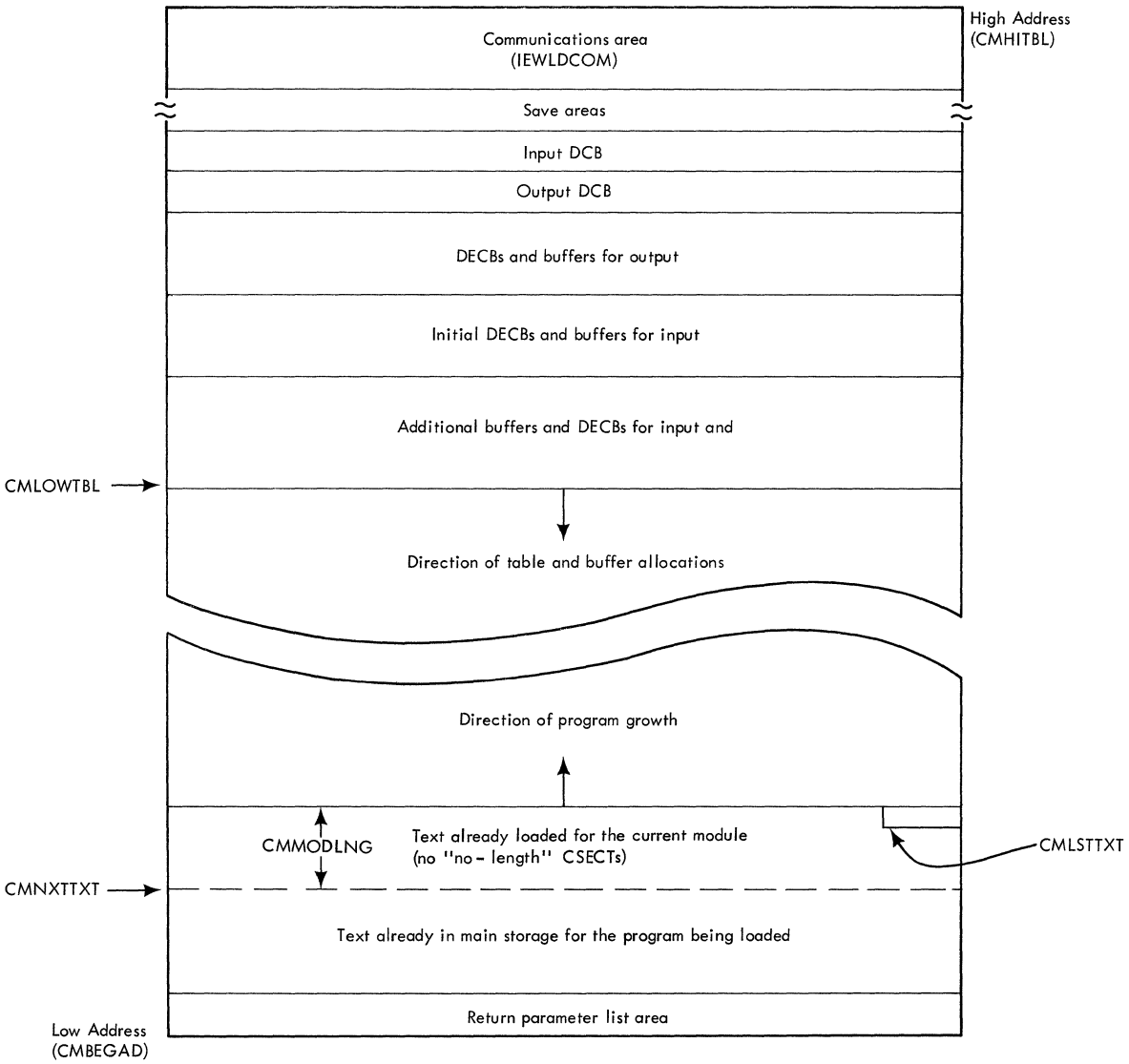
MAKING A CESD ENTRY: For each input entry type, the Loader makes a CESD entry. A WX entry type is treated as an ER input entry type with a "weak-call" flag. The Loader first obtains the storage required for the entry (20 bytes). Whenever possible, the Loader uses storage previously allocated for CESD entries which were later freed. (A CESD entry can be freed as a result of preliminary ESD or resolution processing.) The Loader chains freed entries together. A pointer to the chain resides in the communication area at CMESDCHN; the pointer is updated as the freed entries are used.

If there are no freed CESD entries, the Loader allocates storage for the entry from the highest available processing storage. (See Figure 9.) If the space required for the entry would exceed available storage, the loading process is terminated with an error message. The Loader makes this determination by comparing the pointer for the beginning of the Loader's tables (CMLOWTBL) to the pointer for the highest address used for the loaded program's text (CMLSTTXT).

Table 4. No-Match Processing Required for Input Entry Types

Input Entry Type	CESD Entry	Translation Table Entry	Map Entry
SD	X	X	X
LD	X		X
LR	X	X	X
ER	X	X	
CM*	X	X	X
PR*	X	X	X

*Since CM and PR entries are assigned addresses during final processing, they are also mapped at that time.



Note: CMBEGADR = Beginning address of Loader processing storage
CMHITBL = End address of Loader processing storage
CMLOWTBL = Lowest address allocated for buffers and tables
CMLSTTXT = Highest address already used for the loaded program's text
CMMODLNG = Length of text already loaded for the current module, not including "no-length" CSECTs
CMNXTTXT = Lowest address used for the current module

Figure 9. Storage Allocation

After obtaining storage for the CESD entry, the Loader stores descriptive information in the entry. The information stored depends on the input entry type. Handling of the various entry types is described below:

SD

The Loader moves the symbol from the input entry to the CESD entry. The Loader then assigns an address to the defined CSECT by adding the length of all previously defined CSECTs for this module to the Loader-assigned address of the first CSECT in the module. (In the communication area, the length of all previously defined CSECTs is found at location CMMODLNG; the Loader-assigned address of the first CSECT, if the CSECTs are being passed through text records, is found at location CMNXTTXT; and the Loader-assigned address of the first CSECT, if the CSECTs are being pointed

to by MOD records, is found at location CMCORE1.) For CSECTs pointed to by MOD records,the resulting address is stored in the CESD entry for the SD as the Loader-assigned address of the CSECT. For CSECTs passed through text records, however, the resulting address is compared to the beginning address of the Loader tables (CMLOWTBL). If there is no more unused storage, the loading process is terminated with an error message. Otherwise, the resulting address is stored in CESD entry for the SD as the Loader-assigned address of the CSECT.

Next, the Loader clears the CESD flag field, except for the entry's type indication, and computes the relocation constant. The relocation constant is computed by subtracting the input address (specified by the input SD entry) from the Loader-assigned address. The Loader stores the relocation constant in the CESD entry.

If the option to specify the entry point name for the loaded program was used, the Loader determines whether the SD with that name has already been received. If not, the Loader compares that name to the symbol for the currently defined CSECT (the symbol in the CESD entry). If the names are the same, the Loader-assigned address is stored as the entry point address in CMEPADDR.

For an SD entry, the Loader determines whether the CSECT length specified in the input entry equals 0. If so, the Loader sets the "no length" indicators in the communication area and in the CESD entry itself. If the length is positive, it is added to CMMODLNG to calculate the next CSECT address. If the MAP indicator is on, a MAP entry is made for the SD.

Finally, the Loader puts the CESD entry on the SD chain pointed to in the CMTYPCHN table. Chaining consists of storing the pointer to the last SD entry (found in CMTYPCHN) in the current CESD entry's chain pointer. Then the address of this entry becomes the current pointer in CMTYPCHN. After chaining the entry, a translation table entry is made.

LD or LR

The Loader processes input LD entries in the same manner as input LR entries. The name from the input entry is moved to the CESD entry. Then the Loader-assigned address for the defined label is determined by adding the relocation constant (found in the CESD entry for the related SD) to the input address of the LD or LR entry. If the instructions and data for the module have been passed through text records and if the Loader-assigned address exceeds available storage, the loading process is terminated with an error message. Otherwise, the address is stored in the CESD entry.

The Loader sets the type indication in the CESD entry to LR. Finally, the relocation constant is computed. This value equals the Loader-assigned address minus the input relative address. The relocation constant also is stored in the CESD. If the related SD entry was marked "delete", the Loader makes an ER entry instead of an LR and sets the "delink" flag in the entry to signify that all adcons referring to it should be adjusted.

CM

To make a CM entry, the Loader uses two separately obtained 20-byte areas. The first area obtained is used as an extension to the CM entry. In this portion, the Loader stores the length and the address assigned to the common area in the input. Then the Loader obtains the second 20-byte area and stores in it the name for the common area and the entry's type indication. (This area is the one pointed to by the translation table and the CM chain.) The Loader clears three bytes in the entry to be used as a pointer to related ERs and sets a pointer in it to the extended portion of the CM entry. Finally, a translation table entry is made.

PR

For a PR entry, the Loader moves the information describing the external DSECT from the input entry to the CESD entry. The 3-byte field to be used as a pointer to the related RLDs is cleared, and the entry is chained to the other PR entries. (PRs are chained according to their order in the input.) For a DSECT displacement definition, a translation table entry is also required.

ER

For an ER entry, the Loader moves the name and type from the input entry to the CESD entry. If the input ER entry is marked "never call," the Loader sets the "never-call" indication in the CESD entry. If the input ER entry is marked "weak call," the Loader similarly sets the "weak-call"

indication. Then the Loader chains the ER entry to the other ERs and makes a translation table entry.

MAKING A TRANSLATION TABLE ENTRY: The Loader uses the translation control table to direct building of the translation table.[1] The translation control table consists of 32 fullword entries beginning at location CMTRCTRL in the communication area. Each entry is a pointer to a possible 32-entry extent to be allocated for the translation table. The Loader allocates the extents as required, depending on the number of incoming ESD entries.

The entries of one extent correspond to consecutive ESD IDs in a single module. For example, the entries of the first extent correspond to ESD IDs from 1 to 31; those of the second extent correspond to IDs 32 to 63, etc. (The first extent contains only 31 translation table entries since the initial 4 bytes are used for indexing purposes.) Thus, the position designated for creation of a particular translation table entry depends on the ESD ID of the associated input entry.

Figure 10 is an illustration of the translation control table and the translation table.

To make a translation table entry, the Loader first determines whether the input ID is valid. (Diagram D6, reference (A).) If an ID is not valid, an error message is printed and loading continues with the next input ESD entry. An ID is not valid if it is less than 1 or greater than 1023.

If an ID is valid, the Loader then determines by examining the translation control table whether the extent for this ID has been allocated. If not, the Loader allocates an area for 32 4-byte entries and

---

[1]For each input module, the Loader reinitializes the translation table.

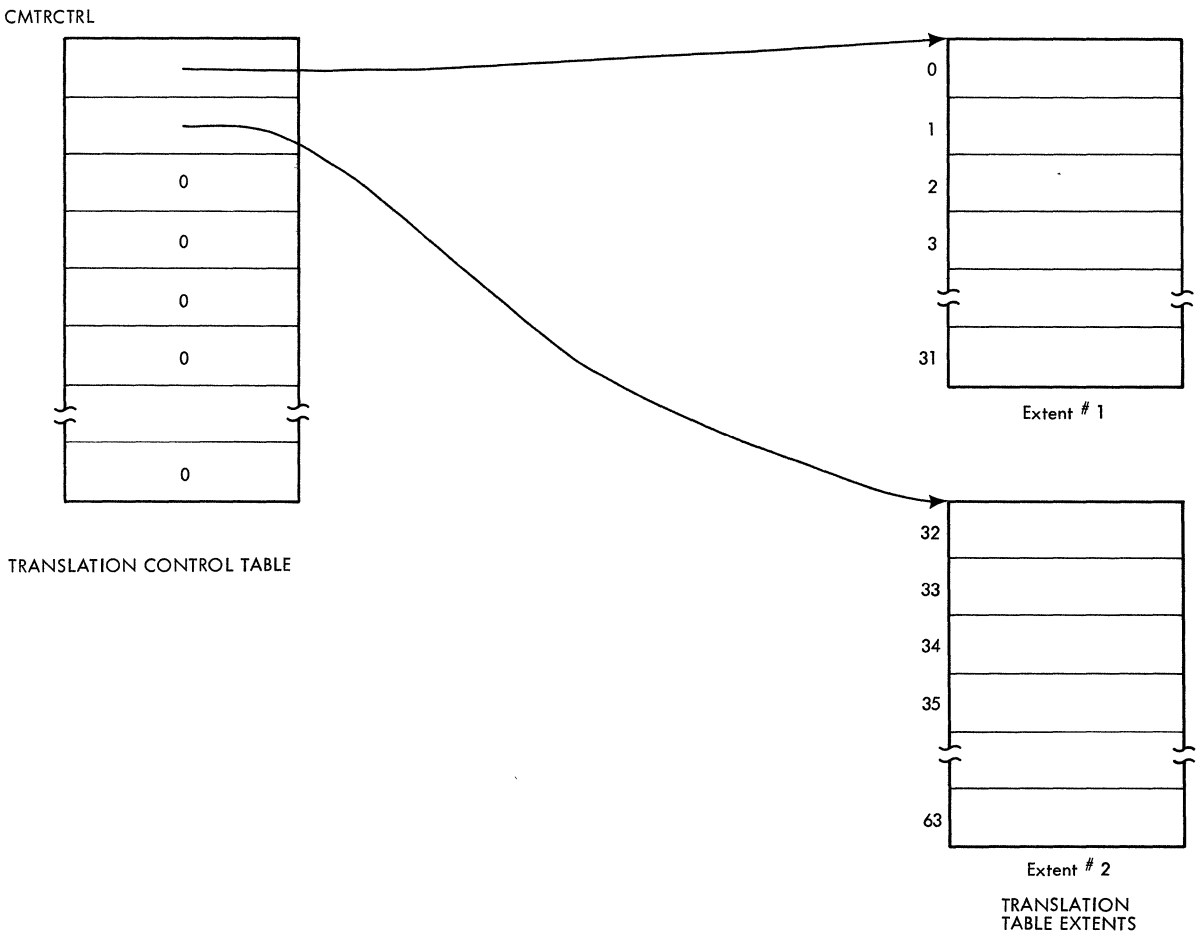


Figure 10. Translation Control Table and Translation Table

stores the beginning address of the area in the translation control table entry for this extent. The area is allocated from the highest available storage. If not enough Loader processing storage remains to make the allocation, loading is terminated with an error message.

After the extent allocation has occurred, the Loader clears the extent. Then the Loader calculates the entry address in the extent for this ID. The address of the CESD entry related to the input entry ID is stored in the translation table entry.

If the CESD entry is an ER, the Loader sets the high-order bit of the first byte to 1. (This indicates absolute relocation.)

Figure 11 shows the overall relationship of tables used in ESD processing.

Match Processing

If the Loader finds a match for an input symbol during the CESD search, the Loader performs symbol resolution. Through resolution, the Loader insures that each named location within the text of the loaded program has a unique symbol.[1] Also, all references to a named location are set to the correct Loader-assigned main storage address.

If two named locations have the same symbol, only one of them can be retained for the loaded program. The Loader determines which is retained on the basis of ESD entry type. The general rules used in symbol resolution follow.

If the entry already in the CESD has type:

SD, it is always retained.

LR, it is always retained.

CM, it is retained except when the input type is SD.

ER, it is always changed to the input type.

---

[1]This does not refer to PC and PR names, which need not be unique.

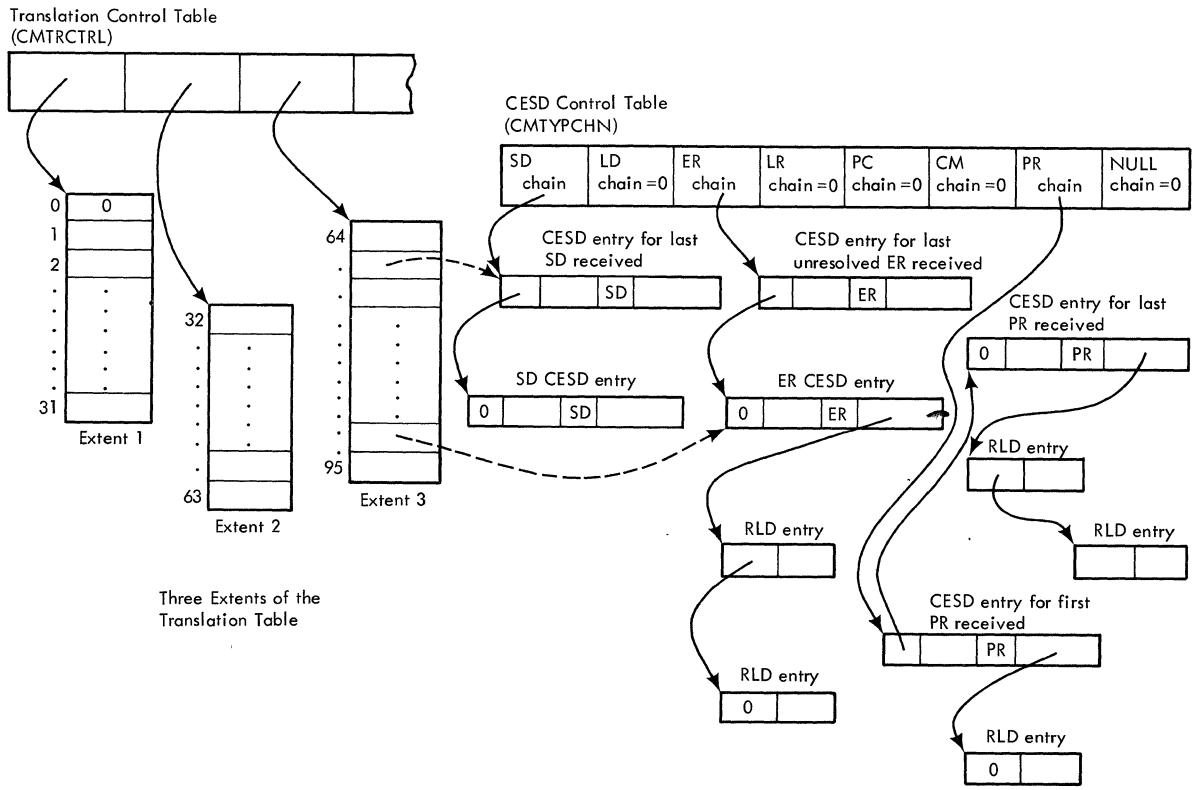


Figure 11. Overall Relationship of Tables

If two entries have matching symbols and have types that indicate they should be retained, the Loader retains the first entry received.

Table 5 gives a summary of symbol resolution.

Table 5. Symbol Resolution

Input Type	CESD Type	Result
SD	ER	SD
	SD	SD
	CM	SD
	LR	LR
CM	CM	CM
	ER	CM
	SD	SD
	LR	LR*
LD/LR	ER	LR
	LR	LR
	SD	SD
	CM	CM**
ER	SD	SD
	ER	ER
	LR	LR
	CM	CM

*Match results in an error.
**Match results in an error if the SD for the LD/LR is not marked "delete."

INPUT ENTRY TYPE IS SD:

CESD type is ER
The Loader changes the ER entry in the CESD to an SD entry. The entry is made as described under "No-Match Processing" for an SD entry. This includes: chaining the entry to other SDs, updating the cumulative length of the loaded program, determining whether this is the loaded program's entry point name, mapping the entry, and making a translation table entry. If RLDs were chained to the ER entry, they are relocated as described under "Relocation Processing." Also, the Loader takes the entry off the ER chain using the pointer to the previous entry on the chain (CMPREVPT). If there are no previous entries, the Loader sets the ER entry in the type chain table (CMTYPCHN) to 0.

CESD type is SD
If the orignial SD is not flagged "delete," the Loader obtains space for another CESD entry and moves the name and Loader-assigned address of the original entry into the new one. The relocation constant is then computed by substracting the input address from the Loader-assigned address. A "delete" indicator is set to show that text and RLDs related to the current input SD should be deleted. If the text for the CSECT has been pointed to by a MOD record rather than passed through text records, the text cannot be deleted and, thus, the cumulative module length (CMMODLNG) is updated to include this CSECT. Finally, the entry is chained to existing SD entries and a translation table entry is made. If the original SD is flagged "delete," the original entry is used.

CESD type is CM
The Loader changes the existing CM entry to an SD. Since the extended portion of the CM entry is no longer needed, the Loader chains it to the freed CESD entries (pointed to by CMESDCHN). First, however, the Loader obtains from the extended portion the length of the common area. For the SD entry, the Loader retains the greater between this length and the one specified in the input SD. To change the CM entry to an SD, the Loader performs the same processing described above for the SD-ER match.

CESD type is LR
The Loader sets the "delete" indicator in the CESD entry so the text associated with the input SD will not be loaded. The relocation constant is updated to reflect the difference between the relative address in the input entry and the Loader-assigned address in the CESD entry. The Loader makes a translation table entry referring to the existing LR entry in the CESD.

INPUT ENTRY TYPE IS CM:

CESD type is CM
The Loader determines the greater between the length in the extended portion of the CESD entry and the length specified in the input CM. The greater length is retained in the CESD entry. The Loader stores the new input address in the extended portion of the CM entry. Also, a translation table entry is made.

CESD type is ER
To change an ER entry to a CM, the Loader obtains a 20-byte area for the extended portion and chains it to the existing entry. The Loader stores the type, address, and length from the input entry in the extended portion of the CESD entry. The CM type

indication is set, and the entry is unchained from the ERs. The Loader chains the entry to the other CMs and makes a translation table entry.

CESD type is SD
The relocation factor in the CESD entry is updated to reflect the CM relative address, and a translation table entry is made.

CESD type is LR
The Loader issues an error message for matching symbols with conflicting types. Nevertheless, the relocation constant is updated and a translation table entry is made.

INPUT ENTRY TYPE IS LD OR LR: With one exception, LD and LR entries are processed in the same way. The difference is that since an LD entry has no ESD ID, the Loader does not make a translation table entry for an LD.

CESD type is ER
The Loader changes the ER entry to an LR. The Loader assigns a main storage address for the symbol by adding the relocation constant from the related SD entry to the relative address in the input LR. Next, the Loader calculates the relocation constant by subtracting the input address from the Loader-assigned address. Both the relocation constant and the Loader-assigned address are stored in the LR entry in the CESD. Any RLDs that were chained to the ER entry are relocated. The Loader checks the LR name for the user-specified entry point and makes a MAP entry if mapping is required. Then, the Loader takes the CESD entry off the ER chain and chains it to the LR chain. If the input entry was an LD, no translation table entry is made. Otherwise, the Loader makes a translation table entry.

CESD type is LR
If the SD entry pointed to by the LR is not marked "delete", the Loader issues an error message for matching symbols with conflicting types. In any case, the Loader updates the relocation constant in the existing CESD entry. The Loader makes a translation table entry referring to the LR in the CESD if the input entry was an LR from a load module. If not, a translation table entry is not required.

CESD type is SD
Processing is the same as that described above for an LD/LR-LR match.

CESD type is CM
The Loader saves the input address in the extended portion of the CM entry. The Loader makes a translation table entry only if the input entry was an LR from a load module. If the SD pointed to by the LR entry is not marked "delete," the Loader issues an error message for matching symbols with conflicting types.

INPUT ENTRY TYPE IS ER: Whenever the Loader makes a translation table entry for an input ER, it sets an indicator for later use. (The indicator signifies during RLD processing that the Loader-assigned address is to be used for relocation of any RLDs with this ID.)

CESD type is SD
The Loader makes a translation table entry referring to the SD entry.

CESD type is ER
If the input ER is marked "never call," the Loader sets the "never-call" indicator in the CESD entry also. If the "delink" indicator is on, the Loader sets the indicator off. In any case, a translation table entry is made referring to the ER entry in the CESD. If either ER is marked "weak call," the "weak-call" flag is set off. If both ERs are marked "weak call," the flag is left on.

CESD type is LR
The Loader makes a translation table entry referring to the LR entry.

CESD type is CM
The Loader sets the input address in the extended portion of the CM entry to zero, and makes a translation table entry referring to the CM entry.

INPUT ENTRY TYPE IS PR: A PR entry can only be matched to another PR entry. When two of these definitions of external DSECT displacements have matching symbols, the Loader sets the existing CESD entry to specify the greater of the two given displacement lengths. The Loader also determines the most restrictive boundary alignment specified in the two PR entries. (For example, doubleword alignment is more restrictive than fullword.) The PR entry in the CESD is changed, if necessary, to specify this alignment.

TEXT RECORD PROCESSING

Text record processing consists of loading those CSECTs required for the loaded program into their assigned locations. The Loader determines whether a CSECT is to be retained or deleted by examining the CESD entry for that CSECT's ID. The translation table is used to obtain the CESD entry.

The way the Loader processes text records depends on whether the current input is an object or a load module. If the input is an object module, the Loader reads all the records for the module, including text, into main storage buffer areas and then processes each record in turn. For load modules, the Loader uses the information in the text control records to process the text before reading it into its assigned storage.

Processing Object Module Text (IEWLTXT)

When a text record is recognized during processing of an object module, the ID contained in the record is translated into a CESD entry address. The Loader translates the ID by first insuring that the ID is valid and then using the translation control table to obtain the corresponding translation table entry.

The translation procedure is the same used prior to allocating a translation table extent. (See "Making a Translation Table Entry.")

In processing text, the Loader considers an ID invalid if no translation table entry exists for it. Thus, an ID between the allowable limits of 1 and 1023 is invalid if it was not received during ESD processing. For any invalid ID, the Loader issues an error message and then tries to process the next record.

(A) If a translation table entry does exist for an ID, the entry contains the address of the CESD entry for the related text. The Loader determines whether the CESD entry is marked "delete." If it is, the Loader skips the text record and tries to process the next record.

(B) If the CESD entry is not marked "delete," the Loader sets an indicator to show that some text has been received for this module. If the "no length" indicator in the CESD entry has been set on, an indicator is set in the communication area to show that text has been received for a "no length" CSECT. The Loader then calculates the address for this text in the loaded program's main storage area. The address equals the displacement of the text from the beginning of the input added to the relocation constant contained in the CESD entry.

(C) Next, the Loader checks whether the text would exceed available storage by adding the length of the text to the assigned main storage address. The resulting end address for the text is compared to the beginning address of the Loader's tables (CMLOWTBL). If the text would overlap the tables, loading is abnormally terminated.

If there is sufficient unused storage for the text, the Loader moves the text from the buffer area to the assigned address in the loaded program's area. Finally, the Loader updates the pointer to the highest address used for the loaded program's text (CMLSTTXT).

Object module text processing is shown in Diagram D7.

Processing Preloaded Text (IEWLMOD)

If a SYSLIN data area consisting of internal object modules is passed to the Loader, one MOD record may be substituted for all text records within a module. Upon encountering a MOD record, the Loader checks that an internal object module is being processed, that no ESD records have been received for the module, and that some control information is contained in the MOD record. If any of these conditions is not met, the record is ignored. Otherwise, indicators are set to show that a MOD record and text have been received for the module. If the origin of the first CSECT is specified, it is saved in the communication area at location CMCORE1. Similarly, the address of the byte following the estimated or actual end of the text is saved at location CMCORE2.

Extent information, used by the identification routine (IEWLIDEN), is saved in chained entries pointed to by location CMXLCHN in the communication area. These entries contain the address and length of the extent and a pointer to the next entry in the chain. The number of extents is saved at location CMNUMXS in the communication area. Later, the identification routine uses these entries to build a parameter list for the IDENTIFY macro instruction.

Finally, the address of the first extent is saved as the default entry point of the program if the entry point has not previously been defined.

Processing Load Module Text (LMTXT)

The Loader uses the text control (or control/RLD) record to process load module text. The control record contains an ID/length list with an entry for each CSECT in the following text record. By processing the IDs consecutively, the Loader determines which CSECTs from the record are to be retained as part of the loaded program.

Load module text processing is shown in Diagram D8.

PROCESSING THE ID/LENGTH LIST: (A) The Loader obtains each ID in turn from the list and attempts to translate each one via the translation control and translation tables to a CESD entry address. If the Loader determines during translation that an ID is invalid, the Loader skips over the record. If there are more records in the module, the Loader continues processing the module.

If the translation of the ID is successful, the Loader checks for the "delete" flag in the CESD entry (obtained by the translation). If the entry is marked "delete," the Loader adds the length from the ID/length list entry to the sum of the lengths of any immediately preceding CSECTs to be deleted.

The accumulated sum is used to truncate the text record when CSECTs at the end of the record are to be deleted. Therefore, only the sum of those consecutive CSECTs to be deleted at the end of the record is used. To accomplish this, the Loader reinitializes the sum of these lengths to zero whenever a following CSECT is to be retained. (CSECTs to be deleted can be scattered throughout a text record.)

If the CESD entry for a text ID is not marked "delete," the Loader determines whether the current CSECT is the first one to be retained from the text record. If it is, the Loader saves the relative relocation constant from the related CESD entry. (After completely processing the ID/length list, the Loader uses this relocation constant to calculate the proper main storage address for reading the text record.) After saving the relocation constant, the Loader sets an indicator to show that at least one CSECT from this record is to be retained and that its relocation constant has been saved. (Only one relocation constant per control record is used since the text record is read in as a whole.)

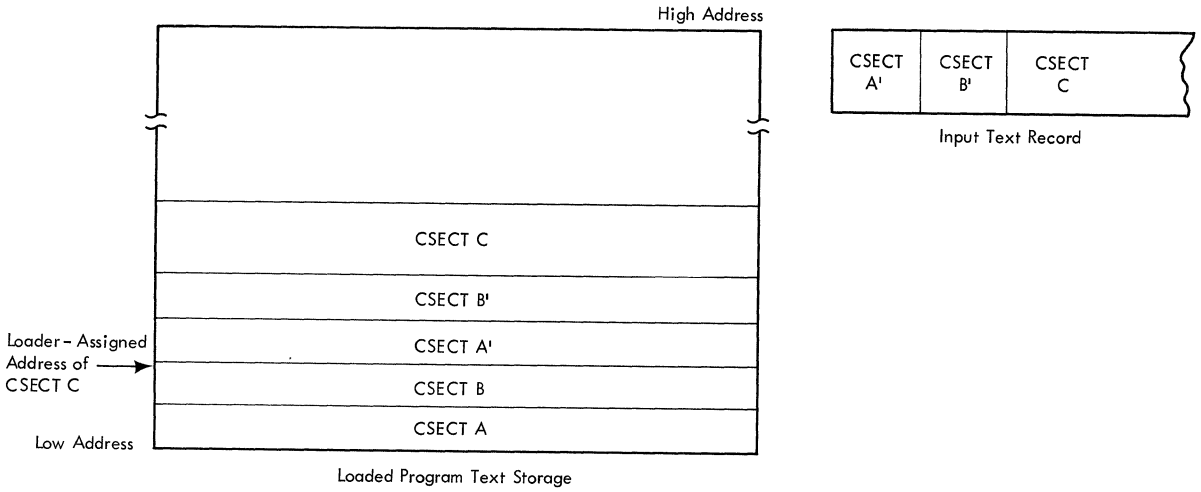
Each time the Loader recognizes a CSECT to be retained, it updates the pointer to the last address used for text (CMLSTTXT) by adding the length of the CSECT to the previous value of CMLSTTXT.

READING THE TEXT: (B) After processing all IDs in the ID/length list, the Loader prepares to read the text into the assigned storage. The Loader:

- Adds the relocation constant and beginning delete length to the CCW address from the text control record to obtain the Loader-assigned address of the text. (See Figure 12.)
- Subtracts the sum of the lengths of consecutive deleted CSECTs at the end of the text record from the text length in the control record to obtain the actual read count.
- Adds the read count to the Loader-assigned address to determine whether sufficient unused storage remains for the text. If not, an error message is issued and loading is terminated.
- Determines whether the text record is the last record in the module by examining the control record's type.

If the record is not the last, the Loader determines whether any CSECTs from the record are to be deleted. If not, the text record and the following control record are read. (The control record is read into the RLD buffer.)

If the text record is the last in the module or if any CSECTs from the record are to be deleted, the Loader reads in only the text record. If an end-of-file occurs, the Loader terminates module-text processing and issues an error message; then the Loader goes to process end-of-module.



Note:
CSECT A' and CSECT B' are to be deleted.
The text read address is, therefore, the Loader - assigned address of CSECT C.
During later text processing, the Loader moves CSECT C to its proper location over CSECT A' and CSECT B'.

Figure 12. Loading the Text from a Load Module Record

CHECKING CSECT STORAGE ADDRESSES: If CSECTs to be deleted were scattered among the CSECTs to be retained, the Loader deletes these scattered CSECTs after the text has been read into main storage.

The Loader insures that each CSECT is in the location determined during ESD processing. To do this, the Loader again translates each ID in the ID/length list to obtain the related CESD entry.

If a CESD entry for an ID is marked "delete," the Loader continues translating successive IDs until one is not marked "delete." The Loader determines whether the related CSECT is in the correct place by comparing the read address to the Loader-assigned address found in the CESD entry. If the text is correctly placed, the Loader continues to translate IDs.

If a CSECT is in the wrong place, the CSECT is moved to the Loader-assigned address. Before checking the next ID in the ID/length list, the read address is updated by the length of the current CSECT to get the read address of the next CSECT. When all CSECTs are in the correct location, the Loader continues processing the module with the next record.

If no CSECTs that were read into main storage are to be deleted, the Loader determines whether a control record was read at the same time as the text record. If so, the Loader continues processing the module with that control record. Otherwise, the end of the module has been reached, and the Loader goes to end-of-module processing.

RELOCATION DICTIONARY (RLD) PROCESSING (IEWLRLD)

Processing of relocation dictionary records consists of building the Loader's RLD table from information in the input RLD records. RLD record processing is the same for object and load module input. (Relocation of adcons is performed as the RLD is encountered unless the referenced CSECT is not in main storage.)

RLD record processing is shown in Diagram D9.

To build the RLD table, the Loader tests the R and P pointers of the entries in an RLD record for validity.[1] These pointers consist of ESD IDs describing an address constant. The P pointer gives the ESD ID of the control section containing the address constant; the R pointer gives the ESD ID of the symbol referred to by the address constant.

Since the pointers are IDs, they are valid if translation yields the address to a CESD entry for the ID. If an invalid ID is received, the Loader issues an error message and continues RLD record processing

---

[1]RLD entries for adcons referring to a cumulative pseudo register are only tested for a valid P pointer, because the R pointer is always zero (CXD type RLD).

with the next entry having different R and P pointers.

The Loader first translates the P pointer. If the CESD entry for that ID is marked "delete," the loader skips all RLD entries with the same R and P pointers. If the CESD entry is not marked "delete," the Loader checks the validity of the R pointer, unless the RLD entry is for a cumulative pseudo register (CXD type).

(A) After insuring that the RLD pointers are valid, the Loader makes an RLD table entry for the input entry. (The Loader uses the storage from a freed RLD entry if possible. Otherwise, storage for the entry is obtained from the highest available storage.)

The Loader stores in the RLD table entry the Loader-assigned address of the address constant. The address is obtained by adding the relocation constant from the CESD entry identified by the P pointer, to the value found in the address field of the input RLD entry. (If the RLD is for a cumulative external DSECT displacement, it is chained from location CMCXDPT in the Loader communication area; the next RLD entry is then processed.) The Loader moves the flag field from the input entry to the RLD table. If the translation table entry indicates that an ER entry is referenced by the R pointer, the Loader sets an indicator in the RLD table for absolute relocation.

After completing the RLD table entry, the Loader determines whether relocation is possible by determining the type of the CESD entry. Processing for the CESD entry types is as follows:

SD, PC, LR
: The Loader clears the chain field of the RLD table entry and relocates the address constant. (See "Relocating Address Constants.")

CM, ER created from LR
: The Loader delinks the RLD entry. That is, it subtracts the input address of the CM or ER from the value in the address constant. The RLD entry is then chained to the CM or ER entry for later relocation after the Loader-assigned address is defined.

PR, ER
: The RLD table entry is chained to the related CESD entry when the address for the CESD symbol is assigned. (See "Match Processing.")

(B) After processing an RLD entry, the Loader continues processing the entries in the RLD record until the end of the record is reached. If the R and P pointers for the next entry are the same as for the current entry, the Loader does not recheck them for validity. Instead, the RLD table entry is made directly. If the pointers for the next entry are different, the Loader performs the validity check.

RELOCATING ADDRESS CONSTANTS (IEWLERTN)

Address constant relocation is the replacement of an address constant in the text of the loaded program with the actual main storage address. The Loader relocates adcons as it encounters their RLD entries, whenever possible.

The Loader processes three types of relocatable address constants:

1. A-type constants, used to reference a location in the same CSECT as the constant.
2. V-type constants, used to reference a location in a different CSECT.
3. Q-type constants, used to reference a displacement in an external dummy section.

In general, the main storage address equivalent of an address constant is calculated by combining either the relative or the absolute relocation constant with the input value of the address constant.[1] The relative relocation constant is the difference between the Loader-assigned address and the input address of the referenced location. The absolute relocation constant is simply the Loader-assigned main storage address of the referenced location. Table 6 relates the types of relocation constants and of address constants to the types of relocation.

When the Loader resolves a CESD entry (e.g., a CESD ER matched with an SD), it relocates all address constants referring to the name. These are pointed to by RLD table entries chained from the CESD entry. The Loader processes each RLD entry in the following way.

First, the Loader insures that the address constant is not an invalid 2-byte adcon. (2-byte adcons can only be used to define external DSECT displacements.) If the adcon is invalid the Loader issues an error message and continues loading the

---

[1]The Loader does not compute the absolute addresses for PRs or CMs until all the text has been loaded.

Table 6. Relocation of Address Constants

Type of Relocation	Relocation Constant Usage	Type of Address Constant	Comments
Absolute Relocation	Absolute relocation constant replaces adcon value	V(symbol) where symbol is not a PR in CESD	Displacements are not valid in V-type address constants.
Relative Relocation	Relative relocation constant is added to or subtracted from adcon value	A(symbol) where symbol is not an ER or PR in CESD	Addition or subtraction is specified by indicators in RLD flag field. Also see comment below for Delinking.
	Absolute relocation constant is added to or subtracted from adcon value	A(symbol) where symbol is ER in CESD	Addition or subtraction is specified by indicators in RLD flag field.
Pseudo Register Relocation	Pseudo register displacement constant is moved in	Q(symbol) where symbol is PR in CESD	
Delinking	Input address of CM or LR/LD CESD entry is subtracted from value	A(symbol) where symbol is CM or ER created from LR/LD	The relocation of address constants pointing to CM CESD entries is a combination of 1) delinking and subsequent 2) relative relocation with the absolute relocation constant.

Note: Absolute relocation constant = Loader-assigned Address
Relative relocation constant = Loader-assigned Address minus the Input Address

program. Otherwise, the Loader moves the adcon from the text to a work area where it determines the type of relocation required.

If the RLD entry indicates absolute relocation, the Loader places the absolute relocation constant at the text address. The RLD entry is placed on the chain of freed RLD table entries (CMRLDCHN), and the next entry on the chain is processed. When the end of the RLD chain has been reached, the Loader continues its processing.

If the RLD entry indicates relative relocation, the Loader also determines the type of relocation constant required. If the location referenced by the adcon is an external reference, the Loader uses the absolute relocation constant. Otherwise, the Loader uses the relative relocation constant. The Loader tests the RLD entry to determine whether the relocation constant should be added to or subtracted from the input value of the address constant. After calculating the adcon value, the Loader moves it back to the text. Finally, the Loader frees the RLD entry and continues resolution.

If the RLD entry indicates delinking for a CM entry or for an LR entry converted to an ER, the Loader subtracts the input address of common or of the LR from the value of the adcon. The result is a reference to a displacement in the common area or input module. When these entries are resolved (i.e., CM address assigned or ER matched), absolute or relative relocation occurs.

If the RLD entry indicates a PR reference, the Loader performs absolute relocation as described above.

END PROCESSING

End processing includes END card processing for object module CSECTs and end-of-module processing for object and load modules.

END Card Processing

The Loader processes object module END cards for the length of the CSECT and for loaded program entry point definition. (Also, when an END card is recognized, the Loader issues messages for any remaining LD entries for which no SD entry has been received.) In setting the length of the current CSECT, the Loader determines

whether the CSECT is a "no-length" CSECT. If it is, the Loader uses the larger of the END card length and the length specified by the CESD SD entry as the CSECT length.[1] If the END card of a "no-length" CSECT does not specify a length and text has been received for the CSECT, the Loader issues an error message. (In this case, the length of the text is used.)

The Loader determines whether the loaded program's entry point name or address has already been received. If it has, the Loader does not process the END card for entry point. If not, the Loader examines the END card for an ID to be used for the entry point. If an ID is present, the Loader sets the entry point address to the address specified by the END card or to 0 if the END card specifies no address. The Loader translates the ID to a CESD entry address and saves the CESD address in location CMEPCESD. (If there is no CESD entry for the ID, an invalid ID message is issued.) The Loader creates an RLD entry for the entry point (at CMEPNAME). This entry is not treated as a regular RLD.

If the END card does not specify an ID but does give a symbolic name to be used as the entry point, the Loader saves the name at location CMEPNAME. If there is an SD or LR entry in the CESD with that name, the Loader uses the address specified as the program entry point address.

End-of-Module Processing

At end-of-module for a load or object module, the Loader initializes for processing the next input module. If text has been passed through text records, the Loader updates the text pointers, CMLSTTXT and CMNXTTXT, by the module length or, if no length was given, to the address of the last text received (rounded to doubleword value.) Then, the Loader determines whether the available storage has been exceeded. If so, an error message is issued and loading is terminated. Otherwise, the Loader clears the translation table and the module length counter (CMMODLNG). All flags except the END and LIB flags are set off. The Loader either begins processing another module from SYSLIN or, if end-of-file on SYSLIN is recognized, goes to process any secondary input.

---

[1]A "no-length" CSECT's SD can be matched by a CM entry, which defines an area larger than the CSECT.

SECONDARY INPUT PROCESSING (IEWACALL)

After the Loader has processed all primary input, it attempts to resolve remaining ERs in the CESD if CALL was specified. If there are no remaining ERs, the Loader performs final processing for the loaded program. (See "Final Processing for the Loaded Program.")

The Loader can resolve ERs from the link pack area and/or the SYSLIB data set. If the link pack area is available for resolution, and the RES option is specified, the Loader searches the contents directory entry queue for the ERs before attemptimg to resolve them from SYSLIB.[2]

Secondary input processing is shown in Diagram E1.

RESOLVING ERS FROM THE LINK PACK AREA

Before resolving ERs from the link pack area, the Loader obtains the address of the contents directory entry (CDE) queue from the communication vector table (CVTQLPAQ). Then, for each ER which is not marked "never call" or "weak call," the Loader searches the CDE queue for an entry with the same name.

(A) In searching the CDE queue, the Loader compares only the first half of the names in the ER and in the CDE. If the first halves match, the second halves are compared. If a comparison is unequal, the Loader continues searching the CDE queue for the name until the end of the queue is reached. Then the Loader searches for the next ER which is not marked "never call" or "weak call."

If the Loader finds a matching name for an ER in a CDE, it puts the entry point in the CESD entry and changes the entry's type to SD. The Loader then takes the entry off the ER chain, puts it on the SD chain, and makes a map entry for the SD if MAP is specified. Finally, the Loader relocates all RLD table entries which are chained to the CESD entry.

If there are still unresolved ERs after the CDE has been searched, the Loader performs library call processing. Otherwise, the Loader performs final processing for the loaded program. (See "Final Processing for the Loaded Program.")

---

[2]The Loader determines whether the system is MVT by checking for X'10' in location CVTDCB in the communication vector table.

RESOLVING ERS FROM THE SYSLIB DATA SET

Before resolving ERs from the SYSLIB data set, the Loader checks whether an open SYSLIB data set has been passed. (The fourth entry in the DCB list, which is passed to the Loader as a parameter, can point to an open SYSLIB DCB.) If an open SYSLIB DCB has been passed to the Loader, the exit addresses in the passed SYSLIB DCB are saved in the communication area and replaced by the Loader's own exit routine addresses. If a SYSLIB DCB has not been passed, a SYSLIB DCB is initialized and opened.[1]

(B) Otherwise, the Loader constructs two lists used for BLDL information in the available storage, which is defined by the CMNXTTXT and CMLOWTBL pointers. The two lists are the BLDL list and an address list. The Loader uses the address list to store pointers to the ER entries in the CESD for which it constructs BLDL entries. The entries in the two lists have a one-to-one correspondence relative to the ER entries. Figure 13 shows this relationship.

Before constructing the lists, the Loader determines the maximum number of entries possible by dividing the amount of available storage by the number of bytes required for an entry in the two lists (BLDL list entry size=16, address list entry size=4). Then, for each ER which is not marked "never call" or "weak call," the Loader makes an entry in the BLDL list including the name specified by the ER and the address of the ER.

After building the BLDL list, the Loader constructs the address list by moving the pointers to the ERs from the BLDL list. This preserves the pointers, which are overlaid in the BLDL list during BLDL operation.

Finally, the Loader issues the BLDL macro instruction. If an I/O error occurs during execution of the BLDL, the Loader logs the error and performs final processing for the loaded program.

(C) Otherwise, the Loader moves the relative track addresses (TTRs) returned in the BLDL list to the associated CESD entries. Each CESD entry for which a TTR was returned is marked to indicate that it contains an auxiliary storage address.

The Loader issues a FIND macro instruction for each ER entry marked "TTR received." The Loader processes each module located in the same way as it processes primary input modules.

Since SYSLIB contains only load modules or only object modules, processing for each module located is the same. If SYSLIB contains object modules, the Loader first primes the buffers and then performs object module processing. If SYSLIB contains load modules, the Loader performs load module processing. See "Primary Input Processing."

The Loader resolves as many ERs from SYSLIB as possible. Then the Loader performs final processing for the loaded program. (If during processing of one of these modules a program size error occurs, the loading procedure is terminated with an error message.)

FINAL PROCESSING FOR THE LOADED PROGRAM

After all possible ERs have been resolved, the Loader performs the following for the loaded program:

- Assigns addresses for common areas.
- Assigns addresses for displacements in the external DSECT (pseudo registers).
- Issues messages for all unresolved ERs.
- Finds the address of the program's entry point.
- Builds a condensed symbol table if the Loader is operating in time-sharing mode.[2]
- Identifies the loaded program to the system.[2]
- Writes out the diagnostic message dictionary.

---

[1] If the Loader has opened a SYSLIN data set, the Loader closes it before opening SYSLIB and reuses the DCB for SYSLIB.

[2] This processing is performed only when the Loader is used with the MVT option of the control program.

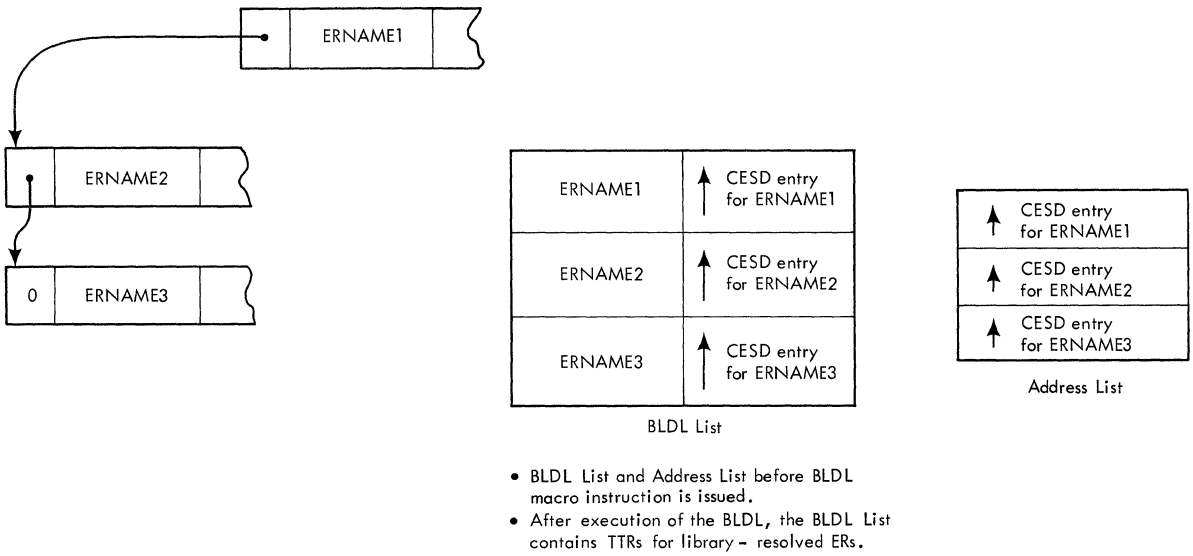


Figure 13. BLDL List and Address List

ASSIGNING ADDRESSES FOR COMMON AREAS (COMMON)

The Loader assigns addresses for the loaded program's common areas by processing entries on the CESD CM chain.

For each CM entry, the Loader assigns the next available storage address above the text of the loaded program. (The highest text address before the allocation of a common area is saved in the communication area at CMTOPCOD. This allows the Loader to continue using work space which may be overlapped with common areas.) The address contained in CMNXTTXT rounded to doubleword value is the address used. The Loader insures that there is enough available storage for the common area and then updates the pointer to available storage by adding the length from the current common entry to the CMNXTTXT value. (If there is not enough storage, an error message is issued and loading is terminated.) Next, the common area is mapped, if the MAP option was chosen. Finally, the Loader relocates the address constants referring to the current "common" definition. (The adcons are relocated through processing the RLDs chained from the current CESD CM entry.)

After all the CM entries in the CESD have been processed, the Loader assigns addresses for external DSECT displacements.

ASSIGNING ADDRESSES FOR EXTERNAL DSECT DISPLACEMENTS (PSEUDOR)

The Loader assigns contiguous storage for displacements in the loaded program's external DSECT by processing the CESD PR chain. (The storage for all DSECTs is obtained via one GETMAIN macro instruction, and the individual DSECTs are displacements within the area.)

For each entry on the chain, the Loader subtracts the alignment factor from hex "FFFF". The Loader adds the difference to the location counter for the PRs to obtain the assigned address of the current external DSECT. (The location counter = 0 at the beginning of PR processing.) After calculating the current address, the Loader updates the location counter by adding the length of the displacement specified in the CESD PR. Then the Loader maps the DSECT displacement and relocates all address constants referring to it. These are indicated by RLD table entries chained to the PR entry.

After processing all the PR entries, the Loader stores the value contained in the location counter (the cumulative length of all DSECTs) in all locations in the loaded program requesting it. These locations are chained from CMCXDPT in the communication area.[1] (If NCAL was specified, there is no CXD chain pointer in CMCXDPT.)

---

[1]See IBM System/360 Operating System: Assembler Language, Order No. GC28-6514, for the use of external DSECTs and the CXD statement.

ISSUING UNRESOLVED ER MESSAGES

For all ERs remaining in the CESD which are not marked "weak call," the Loader issues either error or warning messages. If NCAL is specified or if an ER is marked "never call," the Loader issues a warning message. Otherwise, an error message is issued. An error message is also issued if no text was loaded for the program.

CHECKING THE LOADED PROGRAM'S ENTRY POINT

After the loaded program has been processed, the Loader checks to determine whether the entry point name and address have been received. This is determined by testing the program flag field (CMPRMFLG). Processing for the possible conditions is as follows:

- Entry point name and address both received. No further entry point processing is required.
- Entry point name only received. If the entry point name was specified by the EP=parameter but no address for the name was ever received, the Loader issues an error message. Then if text for the SYSLIN data set was pointed to by MOD records instead of being passed through text records, the address of the first byte of the first extent described on a MOD record is assigned as the entry point. Otherwise, the Loader assigns the address of the first byte of Loader-constructed text (found in CMBEGADR) as the entry point.
- Entry point address only received. If the entry point address was received (CMEPADDR), the Loader determines whether the referenced symbol is an ER. If so, the Loader assigns the first byte of text as the entry point.
- Neither entry point nor address received. The Loader issues an error message and uses the first byte of text as the entry point.

After determining the entry point for the loaded program, the Loader calculates the program's total length. The length equals the difference between the address of the next available storage (CMNXTTXT) and the address of the first byte of text (CMBEGADR) added to the lengths of any extents that may be passed through MOD records. The Loader then prints out the entry point address and the total length of the loaded program.

IDENTIFYING LOADED PROGRAM[1]

If program loading is successful, the Loader prepares to identify the program to the control system. A parameter list is constructed to pass the program name, entry point address, and extent list information to the IDENTIFY macro instruction. (The extent list defines the storage that the loaded program occupies.) If storage is not available for this parameter list, an error message is issued and Loader processing is terminated.

The Loader initializes the parameter list with the program name, entry point address, and length and address of the Loader-constructed program (as the first extent). This information is found in the communication area. If the Loader is operating in time-sharing mode, it attempts to build a condensed symbol table for use during the program's execution. An entry is made in the table for each control section and common area in the program. This table becomes the second extent of the program and its address and length are placed in the extent list. If there is not enough storage for the entire table, it is not built and the second extent of the program is assigned a length of zero. The extent list is then completed with the extent information which was passed on MOD records and saved in the communication area.

Finally, the IDENTIFY macro instruction is issued. If identification processing is not successful, an error message is issued and Loader processing is terminated. Otherwise, a flag indicating that the program has been identified is set in the communication area.

END OF LOADING

After all processing for the loaded program is complete, the Loader processing portion performs termination processing and then passes control to the Loader control portion. The control portion then attempts to execute the loaded program.

---

[1]This processing is performed only when the Loader is used with the MVT option of the control program.

LOADER PROCESSING TERMINATION

If the SYSLOUT and/or SYSTERM data set was opened, the Loader prints a diagnostic dictionary describing the errors encountered during loading. (As errors occur, the Loader sets a flag indicating the type of the error in the bit map field (CMBITMAP) in the communication area.) The Loader determines the highest error severity indicated and returns it to the caller at termination.

Next the Loader insures that all diagnostic data has been written to SYSLOUT and then closes both the output and the current input data sets.[1]

The Loader then sets up the return parameter list. If the processing portion of the Loader was invoked through the entry point IEWLOAD, the name of the identified program is placed in this parameter list. Otherwise, the list contains the main storage address and size of the loaded program.

Finally, the Loader issues a FREEMAIN macro instruction for all its processing storage not assigned to the loaded program or the condensed symbol table. (If the completion code for loading is greater than 4, the storage occupied by the loaded program is also released including preloaded text passed through MOD records. If the loaded program was identified, the storage it occupied is released through the execution of the LOAD and DELETE macro instructions.) The Loader then returns control to the control portion.

---

[1]The current input data set is SYSLIB unless no library searching was done. The Loader closes SYSLIN when it opens SYSLIB. However, if a SYSLIB DCB marked open was passed to the Loader, SYSLIB is not closed.

LOADER CONTROL TERMINATION

Before attempting to execute the loaded program, the Loader control portion issues a DELETE macro instruction for the processing portion. Then, if the condition code for loading is not greater than 4, the Loader control portion passes the user's parameter list to the loaded program for its execution.

After the program's execution, the Loader control portion returns to the scheduler. If the loaded program is invoked through the execution of a branch and link instruction (MFT/PCP) rather than through the execution of an ATTACH macro instruction (MVT), the storage occupied by the loaded program (including preloaded text, if any) is freed by the Loader control portion before return is made to the scheduler.

LEGEND FOR DIAGRAMS

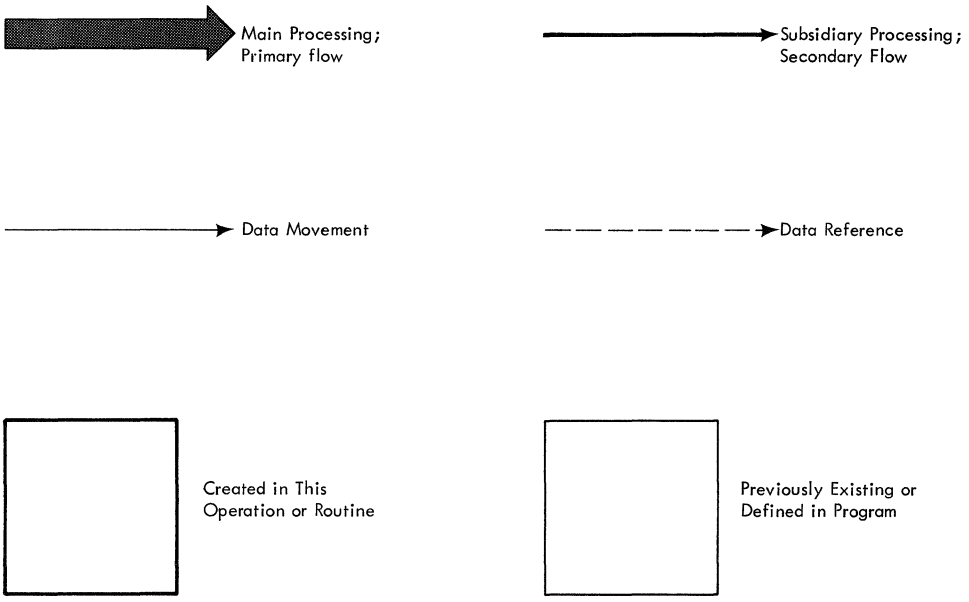
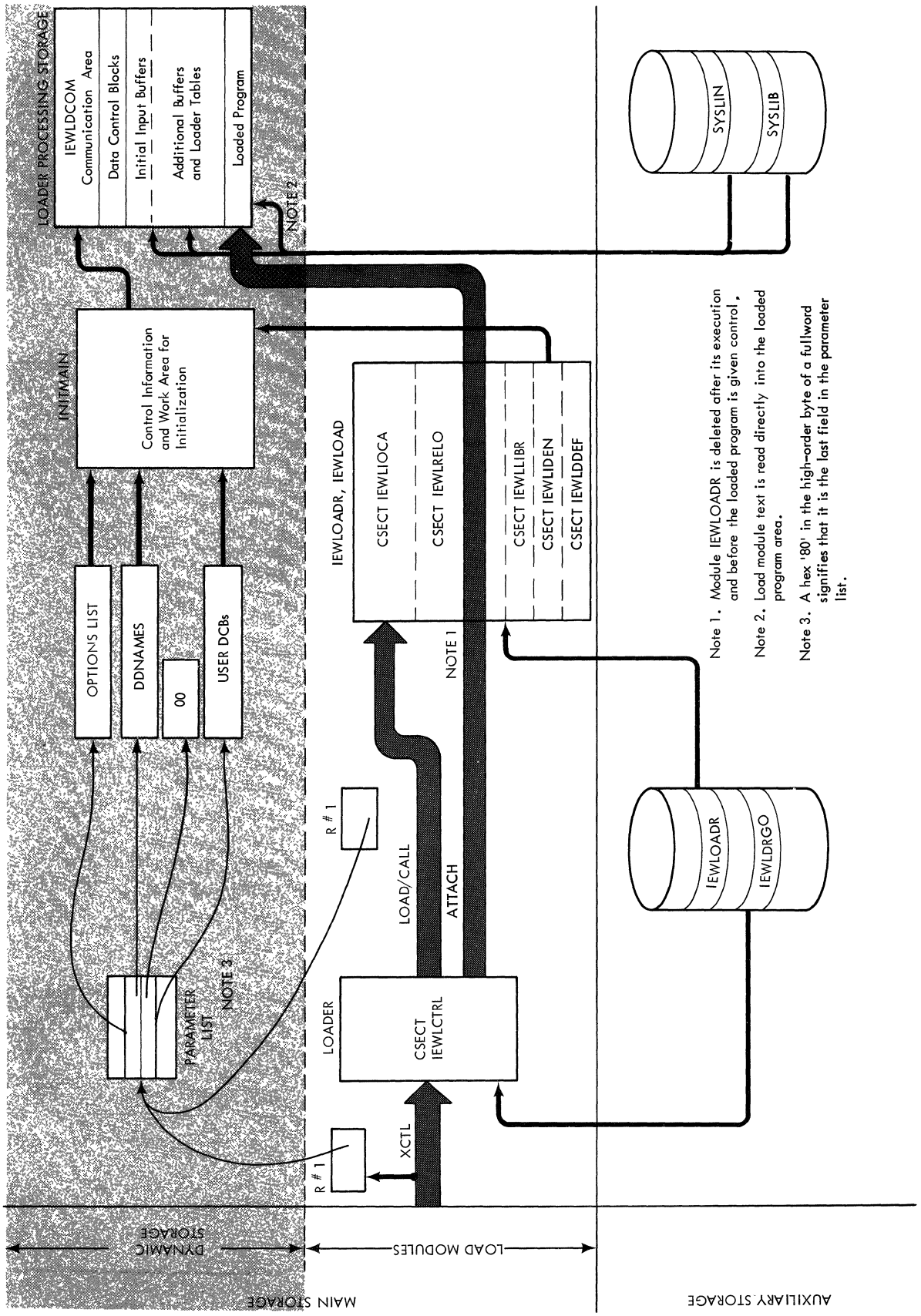


Diagram A0. Overall Loader Operation



• Diagram A1. System Generation

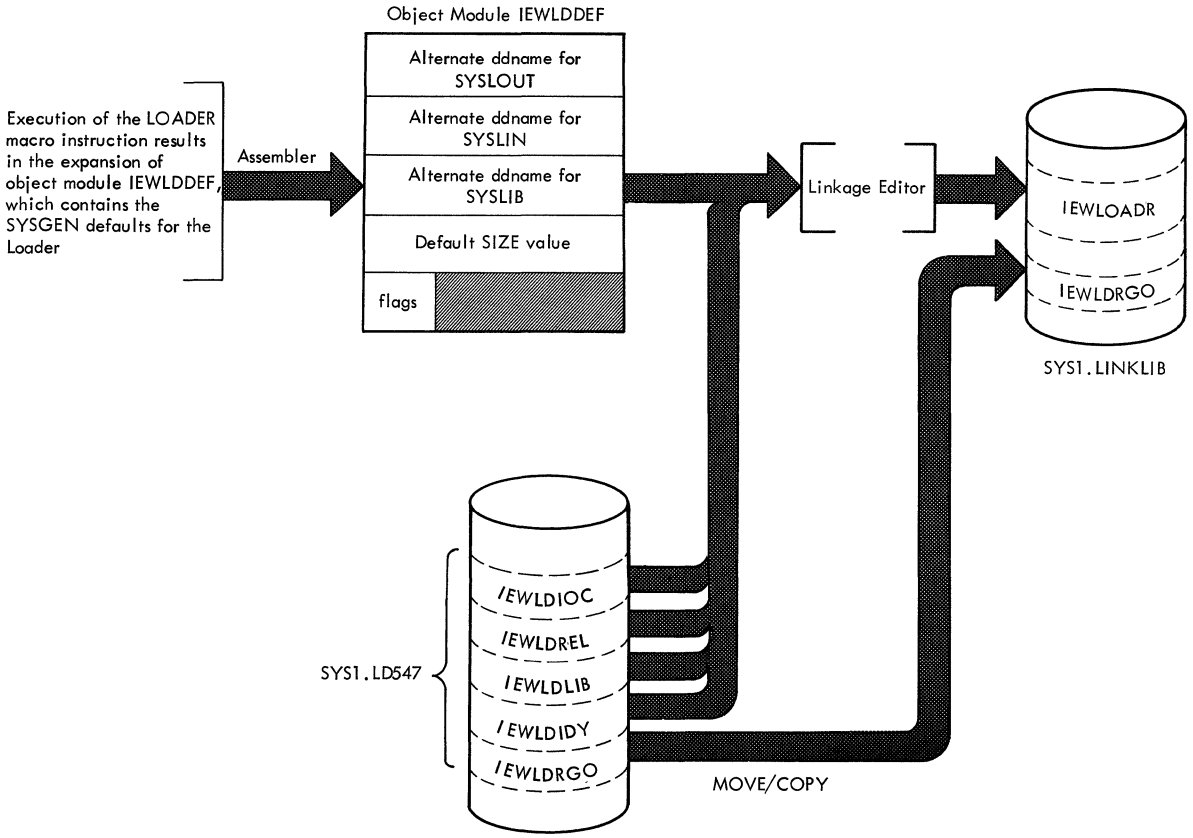
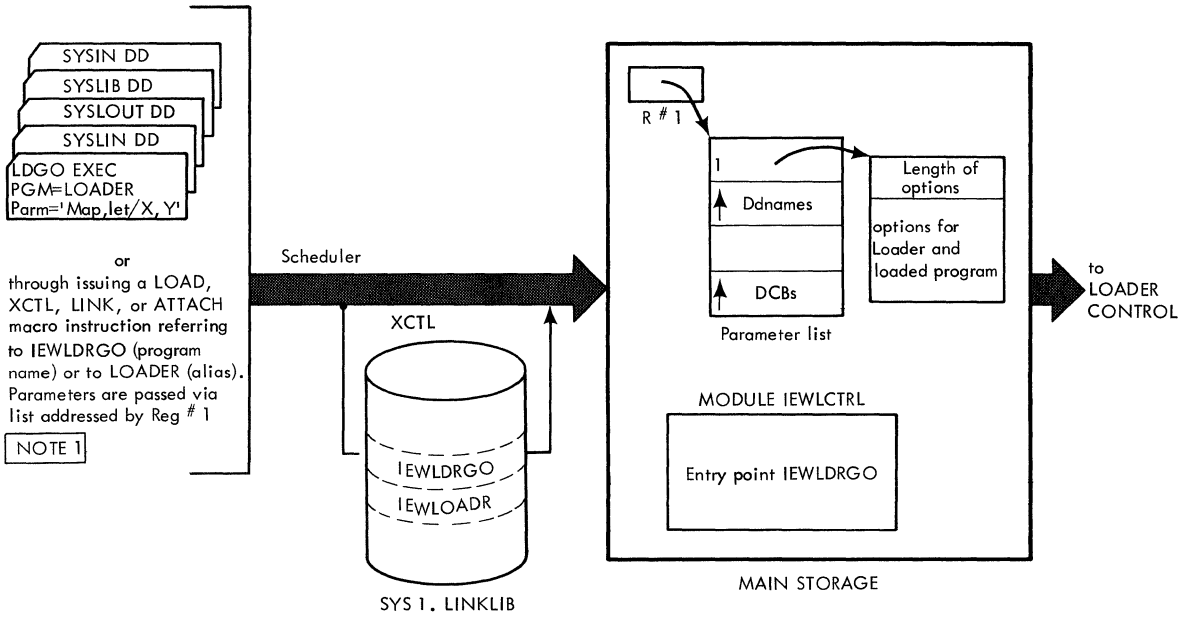


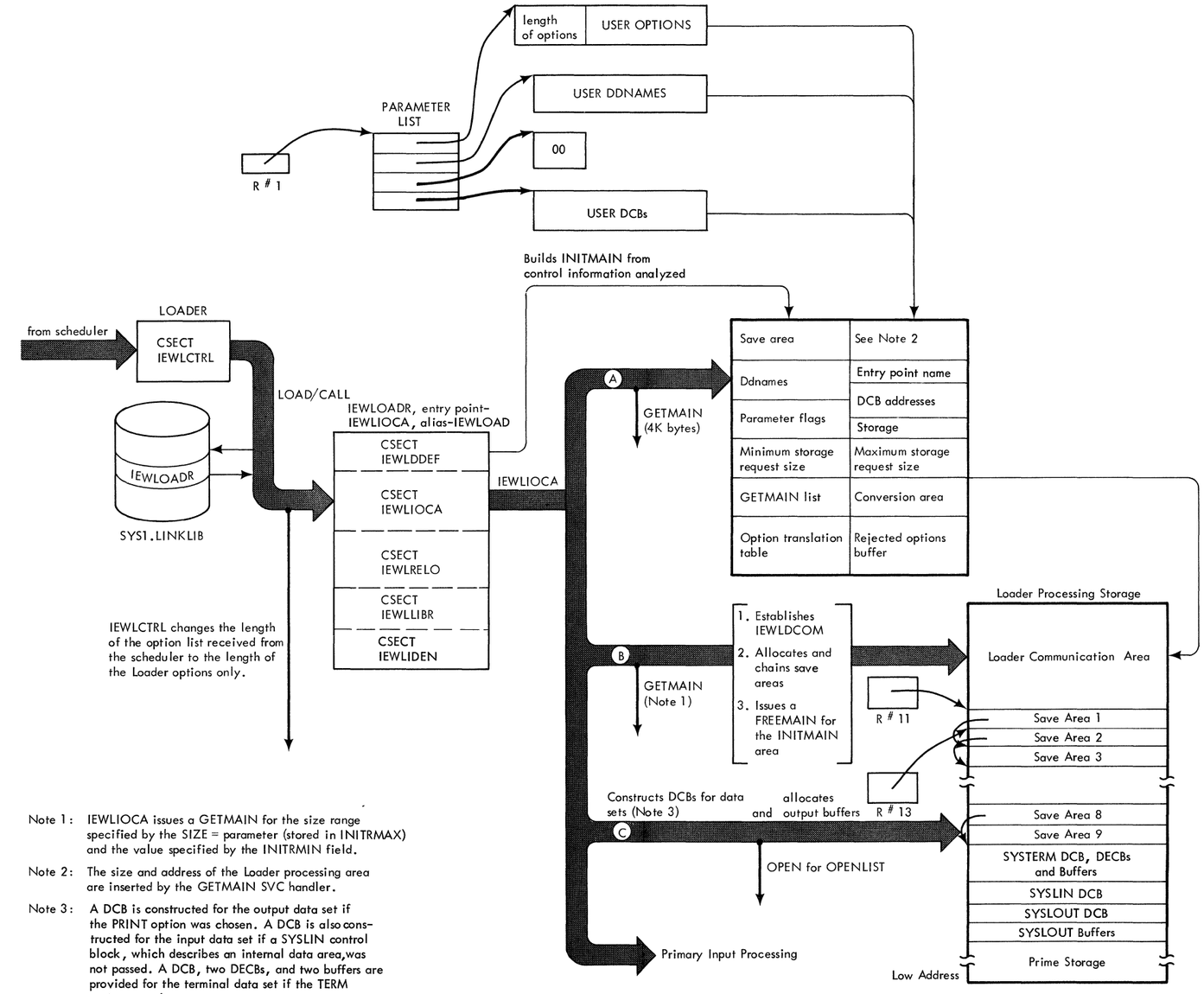
Diagram A2. Loader Invocation



NOTE 1

The user may invoke the Loader to load a program but not pass control to it. In this case, the user issues a LOAD and a CALL macro instruction referring to IEWLOADR (for loading without identification) or to IEWLOAD (for loading with identification). Entry point IEWLOAD may only be used when the MVT option of the control program is in operation.

• Diagram B1. Loader/Scheduler Interface and Initialization

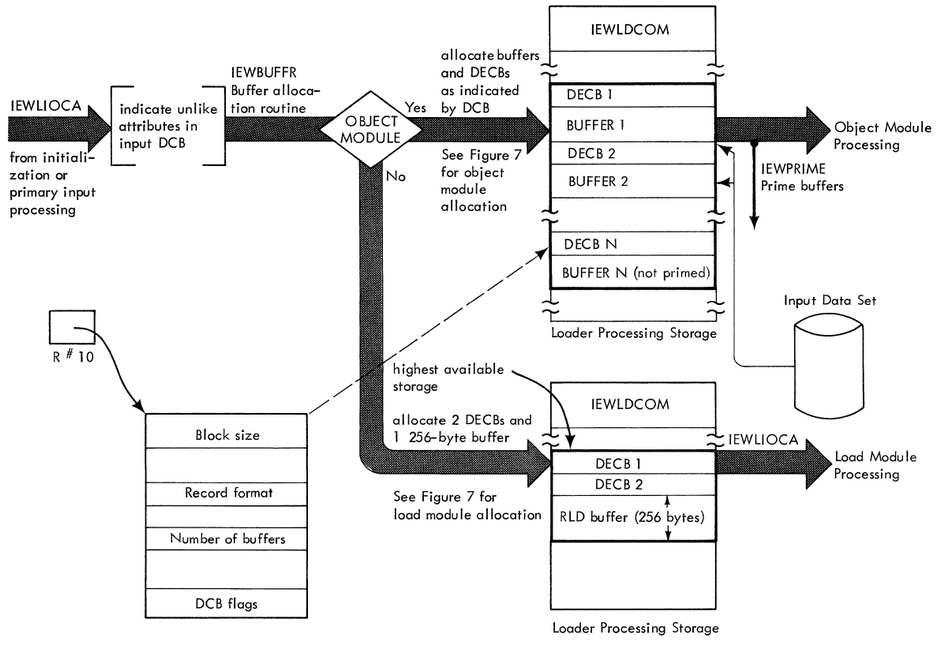


Note 1: IEWLIOCA issues a GETMAIN for the size range specified by the SIZE = parameter (stored in INITRMAX) and the value specified by the INITRMIN field.

Note 2: The size and address of the Loader processing area are inserted by the GETMAIN SVC handler.

Note 3: A DCB is constructed for the output data set if the PRINT option was chosen. A DCB is also constructed for the input data set if a SYSLIN control block, which describes an internal data area, was not passed. A DCB, two DECBs, and two buffers are provided for the terminal data set if the TERM option was chosen.

Diagram C1. Primary Input Control and Buffer Allocation



• Diagram D1. Object Module Processing

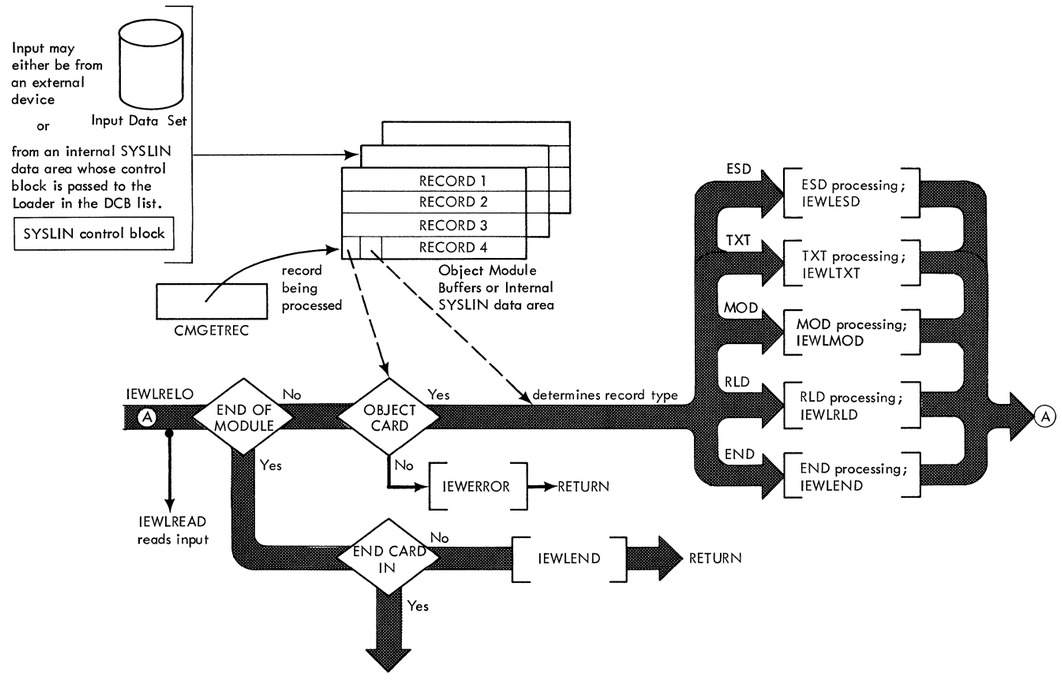
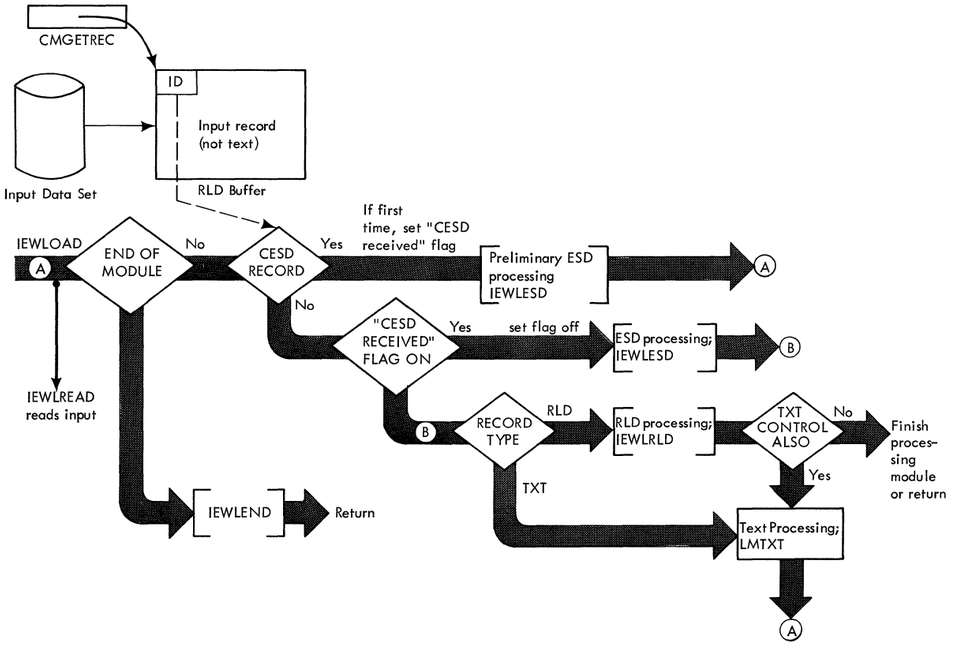
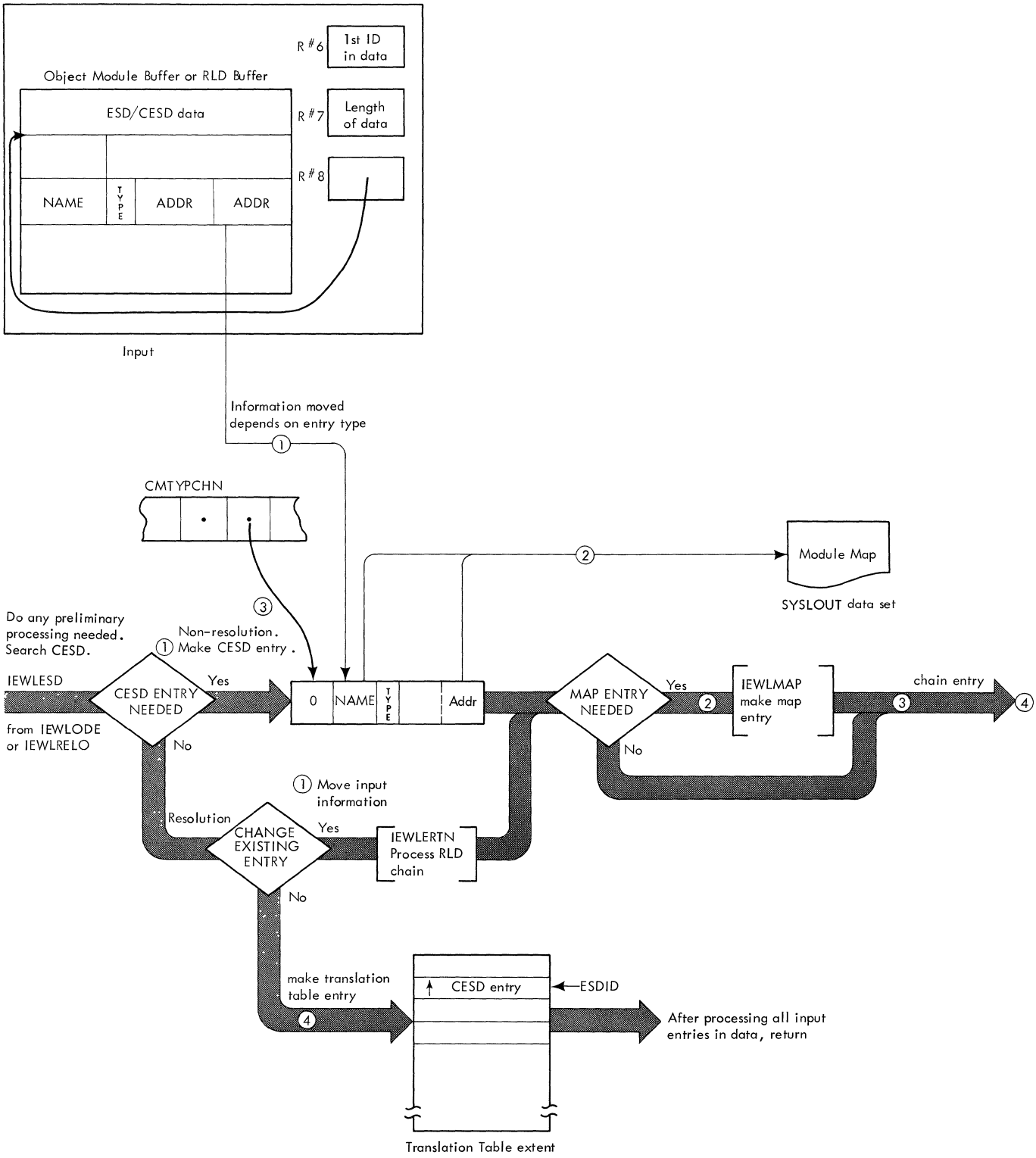


Diagram D2. Load Module Processing

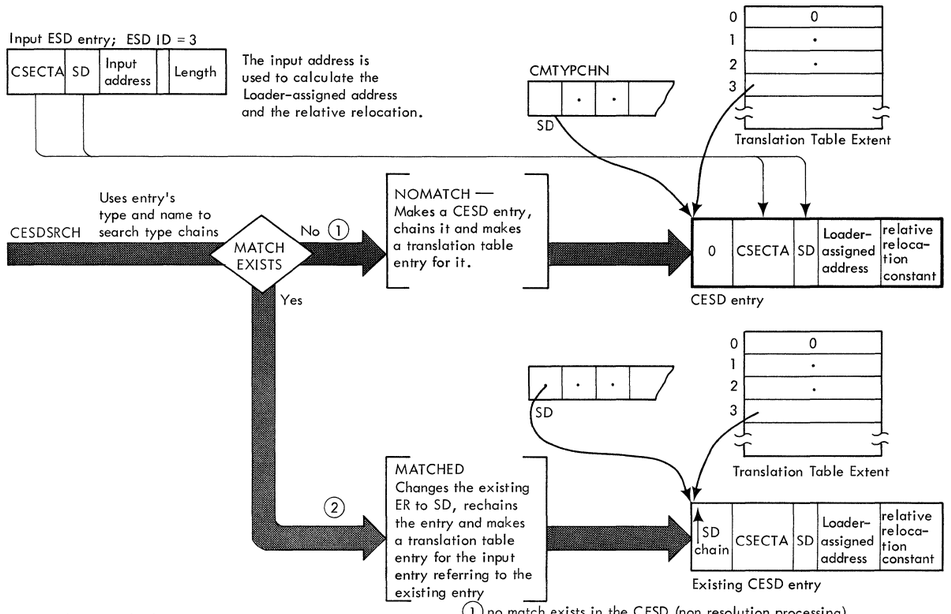


- Diagram D3. ESD Record Processing (Generalized)



Note: ESD processing differs according to entry type and whether resolution is possible. For detailed information, refer to "External Symbol Dictionary Processing". The following diagrams give some examples of processing for different conditions.

Diagram D4. Example of Input ESD Processing (IEWLESD)



This example shows processing for an input SD entry when
(1) no match exists in the CESD (non resolution processing)
(2) a match exists (resolution processing)

Diagram D5. Example of Input ESD Processing (IEWLESD)

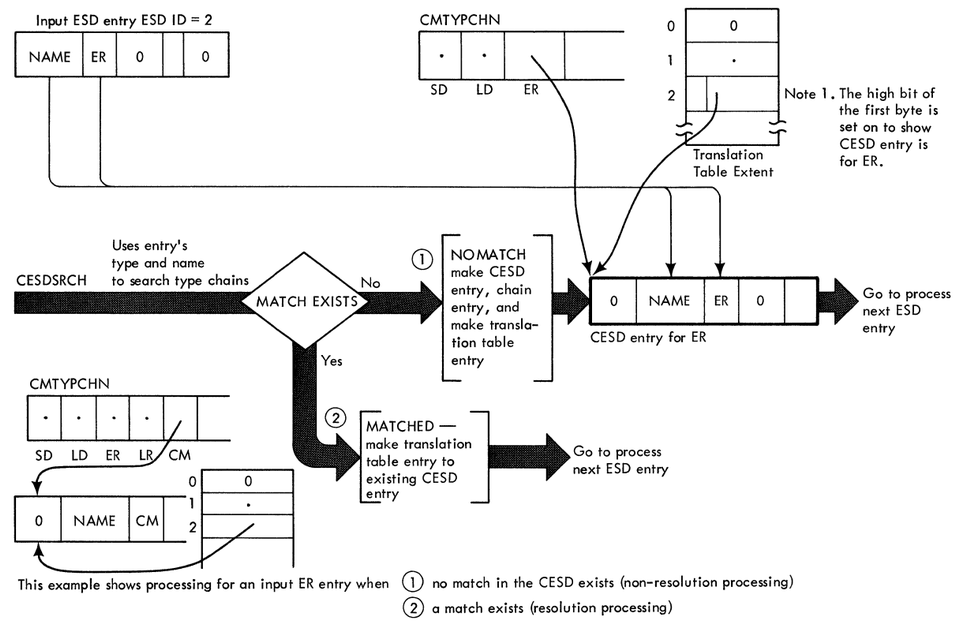
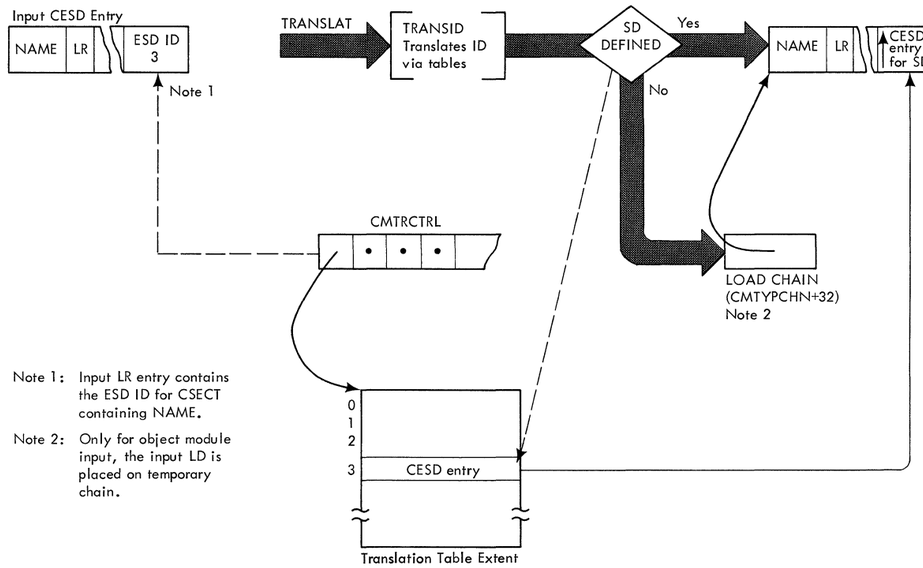


Diagram D6. Example of ESD ID Translation

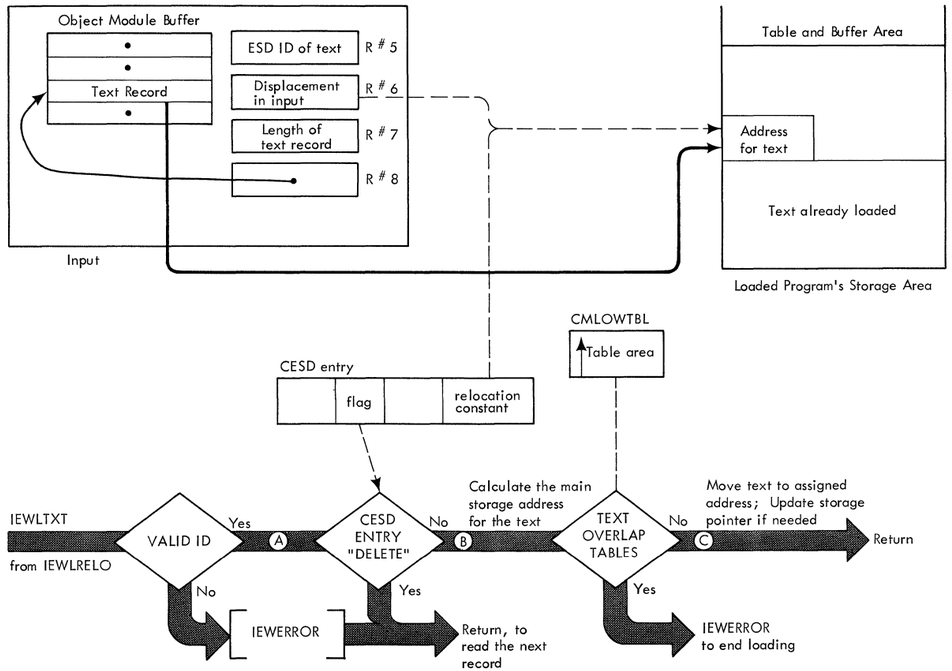


Note 1: Input LR entry contains the ESD ID for CSECT containing NAME.

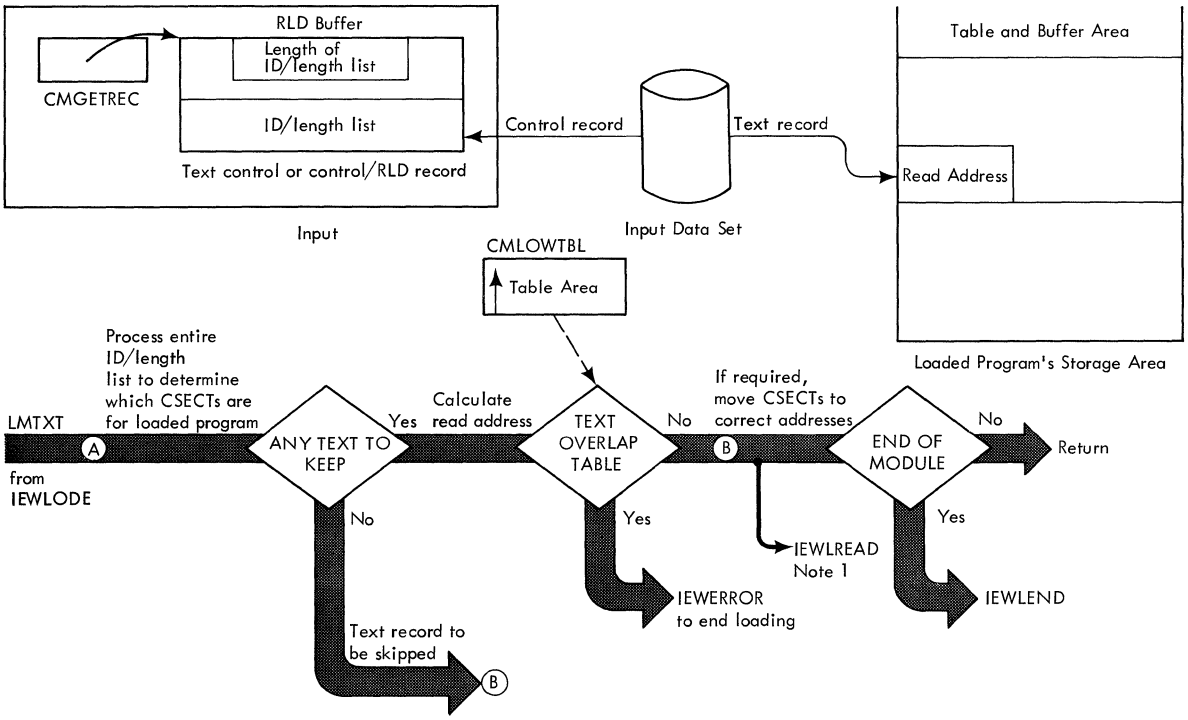
Note 2: Only for object module input, the input LD is placed on temporary chain.

This example shows preliminary processing of an input LR. Translation insures the input ID is valid and obtains the CESD address of the related SD.

Diagram D7. Object Module Text Processing



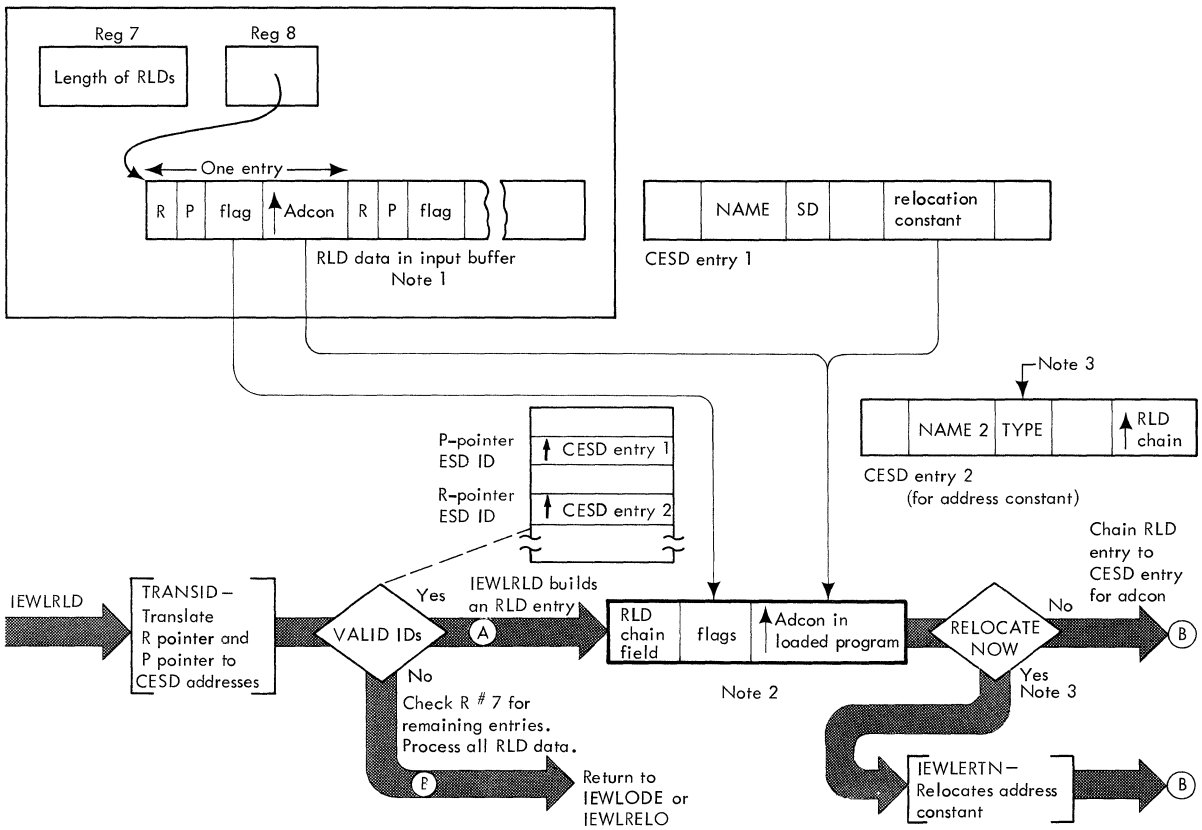
- Diagram D8. Load Module Text Processing



Note 1: Read text record, unless the record is to be skipped; read the following control record also, unless the text record is the last or CSECTs are to be deleted.

Note 2: See Figure 12.

- Diagram D9. RLD Record Processing

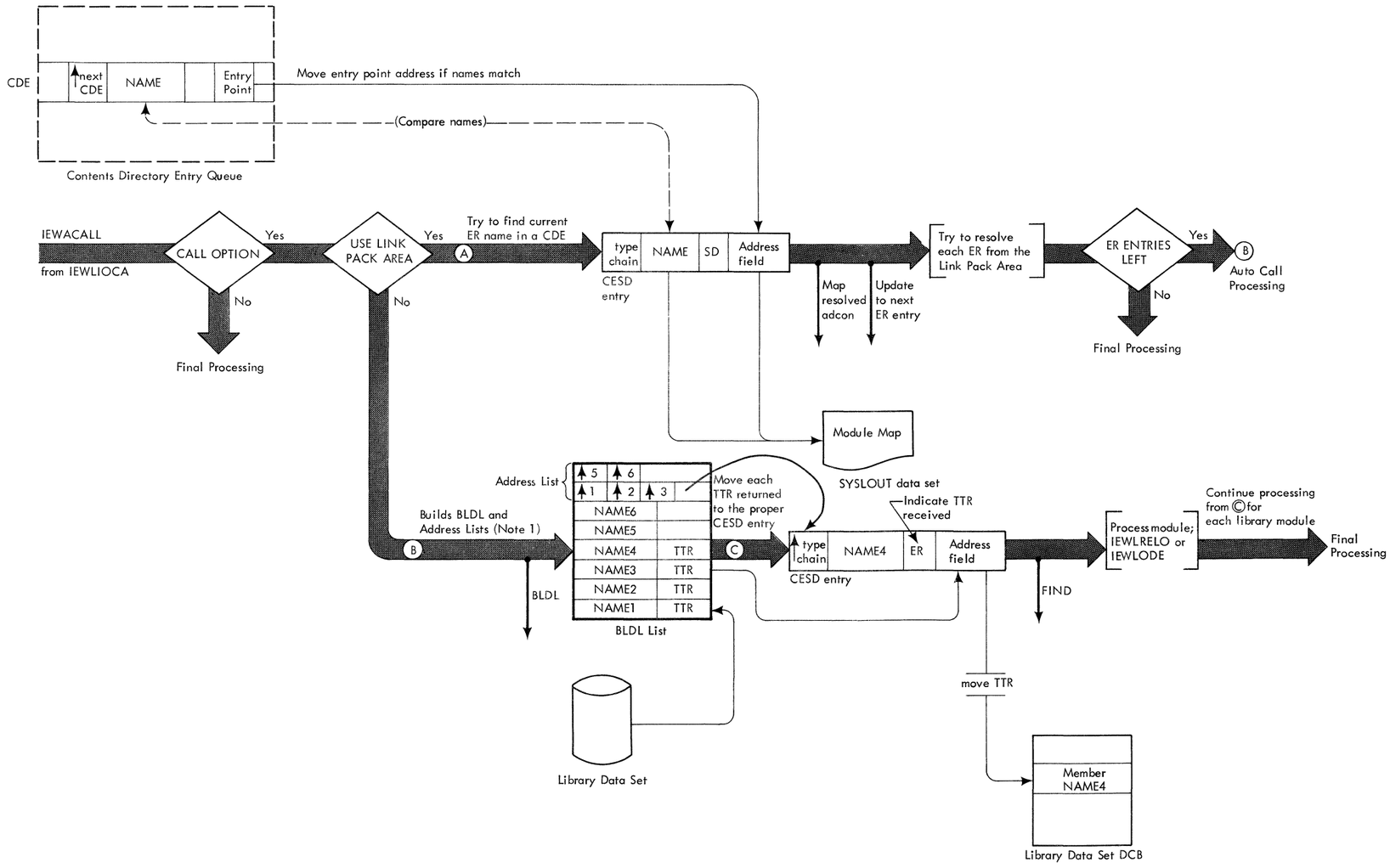


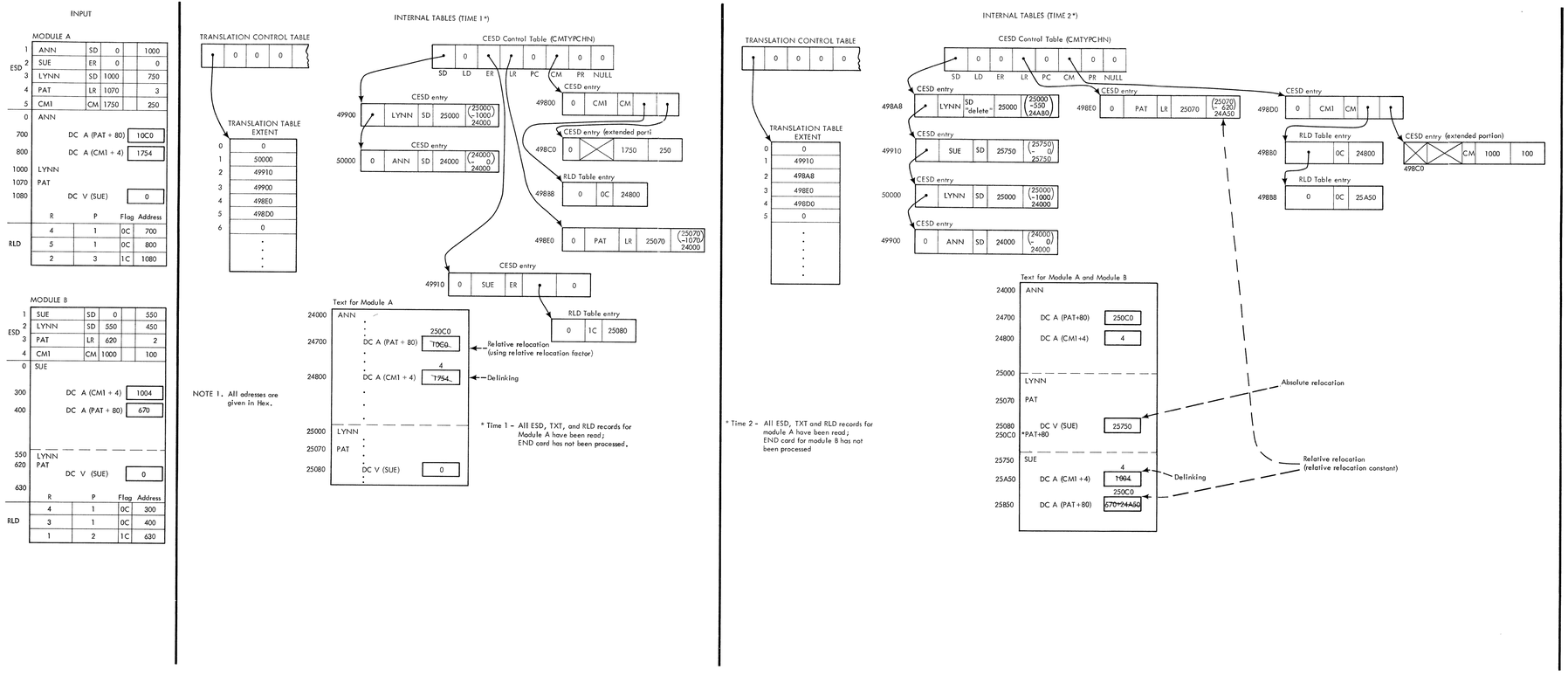
Note 1: The input buffer is the RLD buffer (load module) or an object module buffer.

Note 2: The Loader calculates the adcon address using the P-pointer CESD entry's relocation constant and the ↑ Adcon and flags from the input RLD entry. The flags are inserted in the new RLD entry unless the input RLD is for a CXD PR.

Note 3: If the type in the CESD entry for the address constant is PC, SD, or LR relocation is performed. If the type is CM, PR, or ER, the RLD entry is chained to the CESD entry.

• Diagram E1. Secondary Input Processing





• Figure 14. ESD and RLD Processing

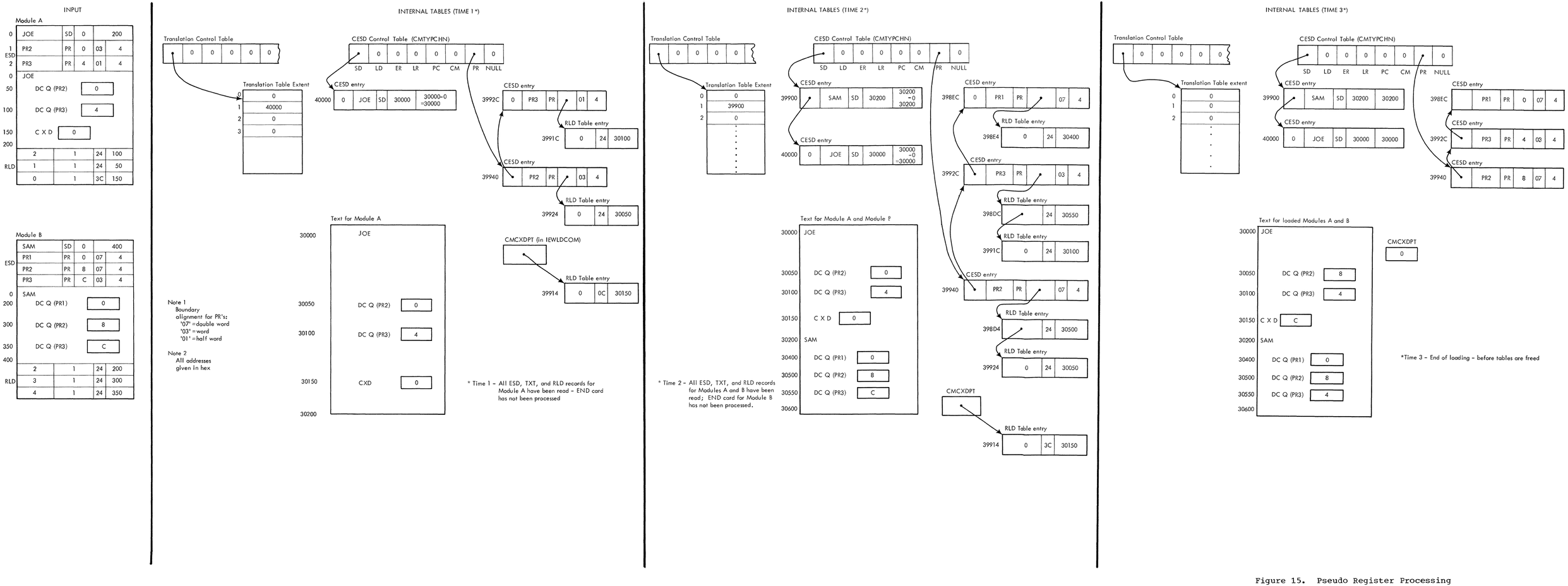


Figure 15. Pseudo Register Processing

The following text and the flowcharts in this section describe the Loader interface and the routines that accomplish the functions of the Loader. The organization of this section corresponds to the organization of the Loader; descriptions of all routines which constitute a phase of the Loader are grouped together. For each routine the symbolic name is given to facilitate use of program listings (See "Section 4: Microfiche directory") and the descriptive name is given to facilitate reference to the "Method of Operation" (Section 2).

Figure 16 shows the organization of the Loader. The flow of control through the first four levels of the processing portion of the Loader (module IEWLOADR) is listed in the control level tables included at the end of this section.

LOADER/SCHEDULER INTERFACE

Loader Control Portion - IEWLCTRL (Chart 100)

Entrance: IEWLCTRL is entered from the scheduler when the Loader is invoked.

Operation: IEWLCTRL loads the processing portion of the Loader (IEWLOADR) and passes control to it. After loading is complete, IEWLCTRL deletes IEWLOADR and passes control to the loaded program for its execution.

Routines Called: When the PCP or MFT option of the control program is being used, IEWLCTRL calls IEWLOADR at entry point IEWLIOCA; when the MVT option of the control program is being used, IEWLCTRL calls IEWLOADR at entry point IEWLOAD. IEWLCTRL invokes the loaded program as follows. For PCP or MFT, the program is called at the established entry point. For MVT, the program is attached by its established program name.

Exit: IEWLCTRL returns to the scheduler.

INITIAL, I/O, CONTROL, AND ALLOCATION PROCESSING

Loader Processing Control - IEWLIOCA (Charts 200-201)

Entrance: This routine can be entered at entry point IEWLOAD, for loading with identification (MVT only), or at entry point IEWLIOCA, for loading without identification. It is entered from IEWLCTRL or it can be called directly by the user.

Operation: IEWLIOCA analyzes the options passed by the calling program and prints a list of options. IEWLIOCA also obtains the Loader processing storage, initializes the communication area, opens data sets, allocates buffers, and handles I/O.

Routines Called: IEWLIOCA calls the following routines: the buffer allocation routine (IEWBUFFR), the buffer prime routine (IEWPRIME), the object module processor (IEWLRELO), the load module processor (IEWLODE), the automatic library call processor (IEWACALL), and the identification routine (IEWLIDEN).

Exit: When loading is completed, IEWLIOCA returns control to the Loader control module (IEWLCTRL).

Buffer Allocation Routine - IEWBUFFR (Chart 203)

Entrance: IEWBUFFR is entered from IEWLIOCA when a new input module is to be read.

Operation: For object and load modules, IEWBUFFR allocates and deallocates buffers and DECBs from the Loader's processing storage.

Routines Called: IEWBUFFR calls the routine to free areas from deallocated buffers and DECBs (FREECORE) and the routine to allocate Loader processing storage for buffers and DECBs (GETCORE).

Exit: When allocation is completed, IEWBUFFR returns control to the I/O, control, and allocation processor (IEWLIOCA).

Storage Allocation Routine - GETCORE

Entrance: GETCORE is entered from IEWBUFFR when storage is needed for a DECB-buffer allocation.

Operation: GETCORE allocates storage from a list of areas freed from previous allocations or from storage not previously used for allocations.

Routine Called: None

Exit: After making the allocation, GETCORE returns to the buffer management routine (IEWBUFFR).

Return Storage Routine - FREECORE

Entrance: FREECORE is entered from IEWBUFFR when storage is no longer needed for a DECB-buffer allocation.

Operation: FREECORE returns storage to a free list pointed to by 'CMFRECOR.' The freed area is blocked with other freed areas whenever possible.

Routine Called: None

Exit: After chaining the freed area, FREECORE returns to the buffer management routine (IEWBUFFR).

Object Module Buffer Prime Routine - IEWPRIME (Chart 204)

Entrance: IEWPRIME is entered from IEWLIOCA before an object module is loaded.

Operation: IEWPRIME reads records into all buffers but one to expedite record processing.

Routine Called: None

Exit: After priming the buffers, IEWPRIME returns control to the I/O, control, and allocation processor (IEWLIOCA).

Read Routine - IEWLREAD (Chart 205)

Entrance: IEWLREAD is entered from the object module processor (IEWLRELO)or the load module processor (IEWLODE) when a record is required for processing.

Operation: To read external object modules IEWLREAD uses the DCB information and the DECB to direct reading of fixed records into the buffers. IEWLREAD also deblocks the physical records and returns the address of the next record to be processed. Similarly, IEWLREAD deblocks an internal data area and returns the address of the next record to be processed. To read load modules, IEWLREAD uses the parameter information to direct reading of different type records.

Routines Called: IEWLREAD calls the generalized read and check routines (RDREAD and RDCHECK, respectively.)

Exit: After the required records are read, IEWLREAD returns to the caller.

Print Routine - IEWLPRNT (Chart 207)

Entrance: IEWLPRNT is entered whenever output to the SYSLOUT data set is to be processed.

Operation: IEWLPRNT inserts the ASA carriage control character before printing the output. The proper code is obtained from the 'PRTCNTRL' table via an index found in 'CMPRTCTL'. This index is reset to space 1 unless changed before the next print.

Routines Called: IEWLPRNT calls the generalized write and check routines (WTWRITE and WTCHECK, respectively).

Exit: After printing the output, IEWLPRNT returns to the caller.

SYSTERM Routine - IEWTERM (Chart 208)

Entrance: IEWTERM is entered whenever output to the SYSTERM data set is to be processed.

Operation: IEWTERM initializes the SYSTERM DCB, opens the SYSTERM data set, and prints the output.

Routines Called: IEWTERM calls the generalized write and check routines (WTWRITE and WTCHECK, respectively).

Exit: After printing the output, IEWTERM returns to the caller.

SYNAD Exit Routine - SYNAD

Entrance: SYNAD is entered from the supervisor when an I/O error occurs.

Operation: SYNAD determines the access method used at the time of the I/O error, and prints and accepts the error.

Routine Called: None

Exit: SYNAD returns to the supervisor.

INPUT MODULE PROCESSING

Object Module Processor - IEWLRELO (Chart 300)

Entrance: IEWLRELO is entered from the input/output - control processor (IEWLIOCA) or from the automatic library call processor (IEWACALL) when object module input is to be processed.

Operation: IEWLRELO requests records to be read, determines the record type, and passes control to the appropriate processor.

Routines Called: IEWLRELO calls the read and deblock routine (IEWLREAD) and the following processors:

- ESD Processor (IEWLESD)
- TXT Processor (IEWLTXT)
- RLD Processor (IEWLRLD)
- END Processor (IEWLEND)
- MOD Processor (IEWLMOD)

Exit: IEWLRELO returns to the input/output - control processor (IEWLIOCA) or to the automatic library call processor (IEWACALL) when end of module is recognized.

ESD Processor - IEWLESD (Charts 301-304)

Entrance: IEWLESD is entered from the object module processor (IEWLRELO) when an ESD record is recognized, or from the load module processor (IEWLODE) when a CESD record is recognized.

Operation: IEWLESD combines ESDs in the Loader input into a composite ESD. Matching input symbols are resolved. A translation table is produced to allow input ESD IDs to be translated into CESD entry addresses.

Routines Called: IEWLESD calls the following routines:

- The allocation routine (ALLOCATE) to allocate storage for a CESD entry.
- The translation routine (TRANSID) to build the translation table and to translate an ESD ID into a CESD entry address.
- The adcon routine (IEWLERTN).
- The map routine (IEWLMAP) to create a map printout.

Exit: IEWLESD returns to the object module or load module processor when the ESD/CESD record has been processed.

RLD Processor - IEWLRLD (Chart 305)

Entrance: IEWLRLD is entered from the object module processor (IEWLRELO) or from the load module processor (IEWLODE) when an RLD record is recognized.

Operation: IEWLRLD builds the RLD table using the input RLDs. If relocation is not possible, the RLDs are chained from the R pointer of the CESD entry.

Routines Called: IEWLRLD calls the following routines:

- The translation routine (TRANSID) to translate an ESD ID to the CESD entry address.
- The allocation routine (ALLOCATE) to allocate storage for an RLD entry.
- The relocation routine (IEWLERTN) to relocate an RLD or to delink an adcon.

Exit: IEWLRLD returns to the object module processor (IEWLRELO) or to the load module processor (IEWLODE).

End Processor - IEWLEND (Chart 307)

Entrance: IEWLEND is entered from the object module processor (IEWLRELO) when an END statement is recognized or from the load module processor (IEWLODE) when end-of-module is detected.

Operation: IEWLEND processes object module END cards for entry point and CSECT length. IEWLEND also processes entry point information for the loaded program if a symbolic entry point is indicated. At end-of-module, IEWLEND resets storage pointers and clears the translation table.

Routine Called: IEWLEND calls the translation routine (TRANSID) to translate an ID to a CESD entry address.

Exit: When end processing is complete, IEWLEND returns control to the object module or load module processor.

Translation Routine - TRANSID (Chart 309)

Entrance: TRANSID is entered from IEWLESD when a translation table entry is required or from IEWLRLD, IEWLTXT, or LMTXT when translation of an ID is required.

Operation: TRANSID translates the ESD ID to a corresponding entry address in the translation table through the translation control table.

Routine Called: TRANSID calls the ALLOCATE routine when a new extent is required for the translation table.

Exit: TRANSID returns to the caller after translation is terminated.

Table Allocation Routine - ALLOCATE

Entrance: ALLOCATE is entered, from IEWLESD when a CESD entry is required, from IEWLRLD when an RLD entry is required, from TRANSID when a translation table extent is required, or from IEWLMOD when storage is required for saving extent information.

Operation: ALLOCATE allocates the required amount of storage for the caller.

Routine Called: None

Exit: After the allocation, ALLOCATE returns to the caller.

MOD Processor - IEWLMOD (Chart 310)

Entrance: IEWLMOD is entered from the object module processor (IEWLRELO) when a MOD record is recognized.

Operation: IEWLMOD processes object module MOD cards for text origin, length, and extent information. If no entry point has been defined, IEWLMOD stores the first extent address for use as a default entry point.

Routine Called: The allocation routine (ALLOCATE) to allocate storage for saving extent information.

Exit: IEWLMOD returns to the object module processor (IEWLRELO).

Address Constant Relocation Routine - IEWLERTN (Chart 306)

Entrance: IEWLERTN is entered from IEWLESD, IEWACALL, or IEWLRLD when address constant (external reference) resolution is required.

Operation: IEWLERTN relocates all address constants pointed to by an RLD chain after determining the type of relocation required.

Routine Called: None

Exit: After the resolution, IEWLERTN returns to the caller.

Map Routine - IEWLMAP (Chart 308)

Entrance: IEWLMAP is entered from IEWLESD or from IEWACALL when a main storage address is to be mapped.

Operation: IEWLMAP formats the proper map entry and causes it to be printed.

Routine Called: IEWLMAP calls the print routine IEWLPRNT and the binary-hex conversion routine IEWLCNVT.

Exit: After printing the map entry, IEWLMAP returns to the caller.

Conversion Routine - IEWLCNVT

Entrance: IEWLCNVT is entered from IEWACALL or IEWLMAP when binary-hex conversion is required.

Operation: IEWLCNVT converts a binary quantity to print characters.

Routine Called: None

Exit: After converting the quantity received, IEWLCNVT returns to the caller.

Load Module Processor - IEWLODE (Charts 400-403)

Entrance: IEWLODE is entered from the input/output-control processor (IEWLIOCA) when load module input is indicated.

Operation: IEWLODE makes read requests for record types (control and/or text) as needed. IEWLODE then determines the particular record type (TXT, CESD, scatter/translation, SYM, text control, control/RLD) and goes to the appropriate processor or requests another record to be read (e.g., scatter/translation records are ignored).

Routines Called: IEWLODE calls the following routines:

- The read routine (IEWLREAD) to read records.
- The ESD processor (IEWLESD) to process CESD records.
- The end processor (IEWLEND) to process end-of-module.
- The translation routine (TRANSID) to translate text IDs to the proper CESD addresses.
- The map processor (IEWLMAP) to create a map printout.
- The RLD processor (IEWLRLD) to process an RLD record.
- The load module text processor (LMTXT) to read text.

Exit: When end-of-module has been processed, IEWLODE returns to the input/output - control processor (IEWLIOCA).

Load Module Text Processor - LMTXT (Charts 401-403)

Entrance: LMTXT is entered from the load module processor when a text control record is recognized.

Operation: LMTXT processes the ID/length list of the text control record to determine which CSECTs are to be retained. CSECTs to be retained are then read into the loaded program's area. If necessary, they are moved to their Loader-assigned addresses.

Routines Called: LMTXT calls the translation routine (TRANSID) to translate the input IDs to the proper CESD entry addresses. The read routine (IEWLREAD) is called to read the text into storage.

Exit: LMTXT returns to the load module processor after processing all IDs and reading all text to be kept.

SECONDARY INPUT AND FINAL PROCESSING

Automatic Library Call Processor - IEWACALL (Charts 500-504)

Entrance: IEWACALL is entered from the control processor (IEWLIOCA) after all SYSLIN input has been processed.

Operaton: IEWACALL first scans the CESD for unresolved ERs at the end of primary input. If CALL is specified, IEWACALL tries to resolve these ERs from SYSLIB and/or the link pack area. After attempting to resolve the ERs, IEWACALL assigns addresses to the common areas and relocates related address constants. Displacements are assigned to the loaded program's external DSECT. Finally, the entry point of the loaded program is determined.

Routines Called: IEWACALL calls the following routines:

- Library open routine (IEWOPNLB) to open the SYSLIB data set.
- RLD relocation routine (IEWLRLD).
- Object module buffer prime routine (IEWPRIME).
- Object module processor (IEWLRELO).
- Load module processor (IEWLODE).
- Map processor (IEWLMAP).

Exit: IEWACALL returns control to the control processor (IEWLIOCA).

ERROR PROCESSING

Error Log Routine - IEWERROR (Chart 505)

Entrance: IEWERROR is entered whenever an error occurs during loading.

Operation: IEWERROR sets bit-map indicators for errors encountered, and formats and prints error messages on the SYSLOUT and/or SYSTERM data set. It also

determines the severity of the error and terminates loading if a severity code 4 is recognized.

Routines Called: IEWERROR calls IEWLPRNT and/or IEWTERM to print a message.

Exits: IEWERROR returns to the highest level caller if a severity code 4 error occurs. Otherwise, it returns to the caller.

Diagnostic Dictionary Processing Routine - IEWBTMAP (Chart 506)

Entrance: IEWBTMAP is entered after final processing for the loaded program is completed.

Operation: IEWBTMAP selects the diagnostic messages to be printed by indexing into the message table.

Routines Called: IEWBTMAP calls IEWLPRNT and/or IEWTERM whenever the bit map indicates an error message.

Exit: After processing the bit map, IEWBTMAP returns to IEWLIOCA.

IDENTIFYING LOADED PROGRAM

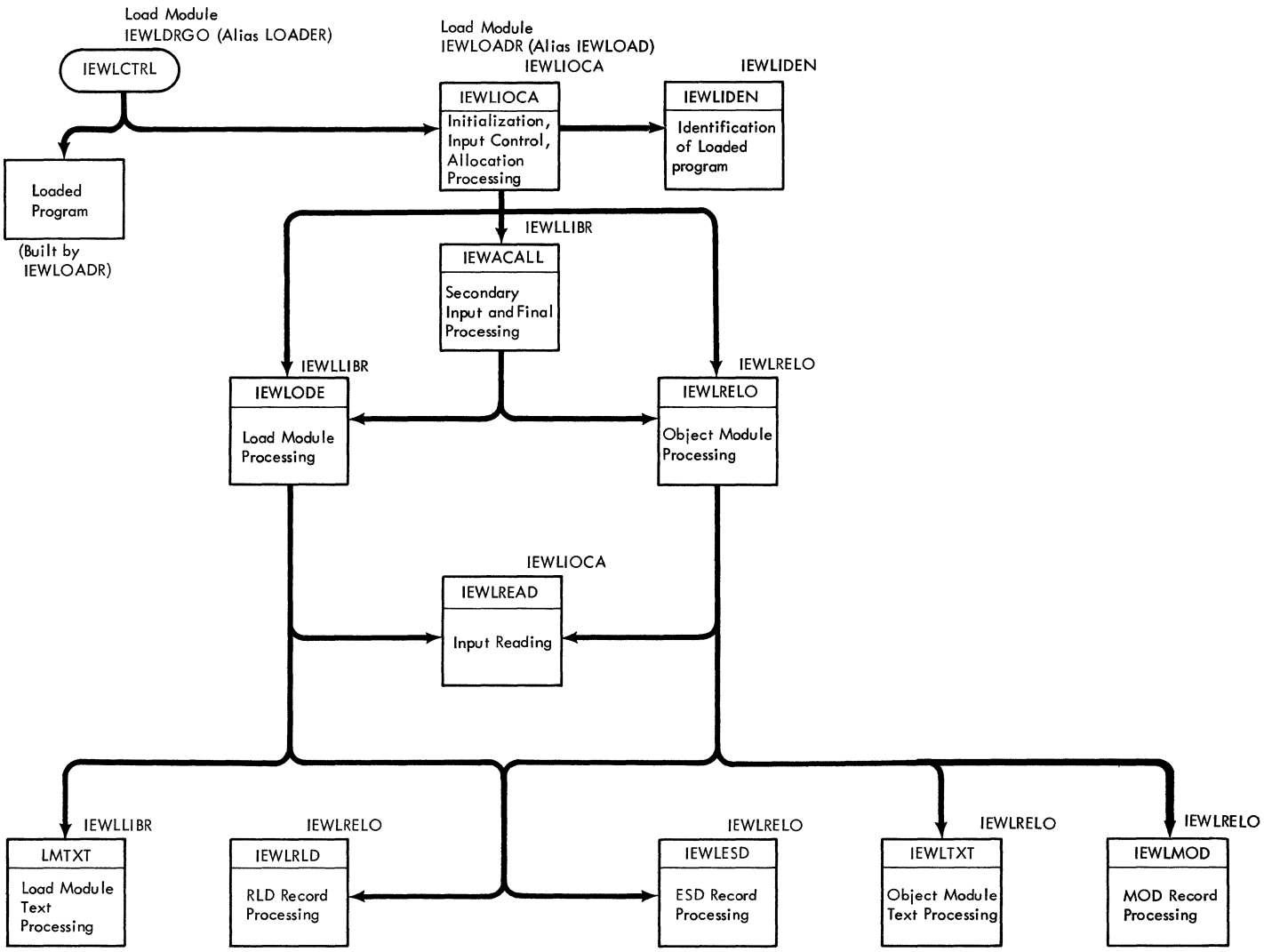
Identification Routine - IEWLIDEN (Charts 600-601)

Entrance: IEWLIDEN is entered after final processing for the loaded program is completed if the processing portion of the Loader (IEWLOADR) was invoked at the entry point IEWLOAD.

Operation: IEWLIDEN creates an extent list and invokes the IDENTIFY macro instruction to identify the loaded program to the control program. A condensed symbol table is also constructed if enough space is available.

Routines Called: IEWLIDEN calls the extent list entry routine (IDENTER) and the condensed symbol table routine (IDMINI).

Exit: After identifying the loaded program, IEWLIDEN returns to IEWLIOCA.



Note: The CSECT containing the code of a function is noted outside the functional block.

- Figure 16. Loader Organization

ROUTINE CONTROL - LEVEL TABLES

The following tables follow control flow in the Loader (processing portion) through four levels. The routine descriptions are listed alphabetically within a level.

Module: IEWLOADR-Level 1

Routine	Purpose	Called Routines	Calling Conditions
IEWLIOCA	Initialization, primary input control, and allocation processing	IEWLPRNT	Called if SYSLOUT data set is open
		IEWBUFFR	If more data exists on SYSLIN
		IEWPRIME	If SYSLIN input is an object module
		IEWLRELO	If SYSLIN input is an object module
		IEWLODE	If SYSLIN input is a load module
		IEWACALL	When all SYSLIN input is processed, unless SYSLIN did not open
		IEWLIDEN	If the loaded program is to be identified to the control program
		IEWBTMAP	Input processing completed

IEWLOADR-Level 2

Routine	Purpose	Called Routines	Calling Conditions
IEWACALL	Secondary input and final processing	IEWOPNLB	If ERs cannot be resolved from primary input or the LPA
		COMMON	Always
		IEWLMAP	If an ER is resolved
		IEWLERTN	If an ER is resolved
		IEWERROR	If an error occurs
		IEWPRIME	If SYSLIB input is object modules
		IEWLRELO	If SYSLIB input is object modules
		IEWLODE	If SYSLIB input is load modules
IEWBTMAP	Processing of error-bit map and printing of diagnostic dictionary	IEWLPRNT	If SYSLOUT is open and messages are required
		IEWTERM	If the TERM option is specified and messages are required
IEWBUFFR	Buffer Management	FREECORE	If previous or current (not the first) allocation is for object module
		GETCORE	If no previously allocated area is large enough for current request
IEWLIDEN	Identification of the loaded program to the control program	IDENTER	Always, unless extents will overlap Loader work space
		IDMINI	Always, unless extents will overlap Loader work space
		IEWERROR	If an error occurs
IEWLODE	Process a load module	IEWLREAD	Always
		IEWLEND	If end-of-module is indicated
		IEWLESD	If CESD record is received
		IEWLRLD	If RLD record is received
		LMTXT	If TXT record is read in
IEWLPRNT	Print output to SYSLOUT data set	RDCHECK	If DECB was previously written
		WTWRITE	Always
		WTCHECK	Always
IEWLRELO	Process an object module	IEWLREAD	Always
		IEWLEND	If END card received
		IEWLESD	If ESD card received
		IEWLRLD	If RLD card received
		IEWLTXT	If TXT card received
		IEWLMOD	If MOD card received
IEWPRIME	Read records into all but one buffer before IEWLRELO receives control	RDREAD	Always

IEWLOADR - Level 3 (Part 1 of 2)

Routine	Purpose	Called Routines	Calling Conditions
COMMON	Assign addresses to common areas	PSEUDOR	Always
		IEWLMAP	Always, unless no CM entries were received
		IEWLERTN	Always, unless no CM entries were received
FREECORE	Chain deallocated area to free list	none	
GETCORE	Allocate prime core for allocation request	IEWERROR	If table overflow occurs
IDENTER	Create entry in extent list	none	
IDMINI	Create a condensed symbol table	none	
IEWERROR	Handle error messages, severity code 4 errors	IEWLPRNT	If SYSLOUT data set is open
		IEWTERM	If the TERM option is specified
IEWLCNVT	Convert binary quantity to hex	none	
IEWLEND	Process END card, reinitialize for next module	TRANSID	If END card specifies entry point address
		IEWERROR	If error occurs in end card processing
IEWLERTN	Relocate all adcons indicated by RLD chain	IEWERROR	Invalid 2-byte adcon
IEWLESD	Create CESD from input ESD/CESD	LOADPROC	If input is a load module
		CESDSRCH	Input entry is not NULL or PC
		TRANSLAT	If NULL entry is made
		CESDENT	If PC or LR entry is required
		ENTER	If PC entry is required
		CHECKEP	If PC entry is required
		MATERSD2	If PC entry is required
		TRANSID	If LD/LR is received
IEWLMAP	Create Map entry for referenced location in loaded program	IEWLPRNT	Always
		IEWLCNVT	Always
IEWLMOD	Process MOD card, store text origin, length, and extent information	ALLOCATE	If extent information is passed on MOD card
IEWLODE	Process a load module	IEWLREAD	Always
		IEWLEND	If end-of-module is indicated
		IEWLESD	If ESD record is read in
		IEWLRLD	If RLD record is read in
		LMTXT	If TXT record is read in
IEWLPRNT	Print output to SYSLOUT data set	RDCHECK	If DECB was previously written
		WTWRITE	Always
		WTCHECK	Always
IEWLREAD	Handle request for data	RDREAD	Always
		RDCHECK	Always

IEWLOADR - Level 3 (Part 2 of 2)

Routine	Purpose	Called Routines	Calling Conditions
IEWLRELO	Process an object module	IEWLREAD	Always
		IEWLEND	If END card is received
		IEWLESD	If ESD card is received
		IEWLRLD	If RLD card is received
		IEWLTXT	If TXT card is received
IEWLRLD	Relocate adcons indicated by RLD entries received or chain RLDs off CESD entry for R pointer	TRANSID	Always
		ALLOCATE	If no free RLD entry is available
		IEWLERTN	If relocation is possible or if delinking required
IEWLTXT	Move object module text to correct space	TRANSID	Always
		RELOREAD	Always
		IEWERROR	If invalid ID received
IEWOPNLB	Open SYSLIB; close SYSLIN	IEWBUFFR	Unless SYSLIB was not opened
IEWPRIME	Read records into all but one buffer before IEWLRELO receives control	RDREAD	Always
IEWTERM	Print output to SYSTERM data set	WTWRITE	Always
		WTCHECK	Always
LMTXT	Read load module text into main storage	TRANSID	Always
		IEWLREAD	Unless record is to be skipped
		IEWERROR	If text record not received
		PROCEOM	Always
RDCHECK	Check DECB	none	
RDREAD	Read input using DECB information	none	
WTCHECK	Check DECB	none	
WTWRITE	Write output using DECB information	none	

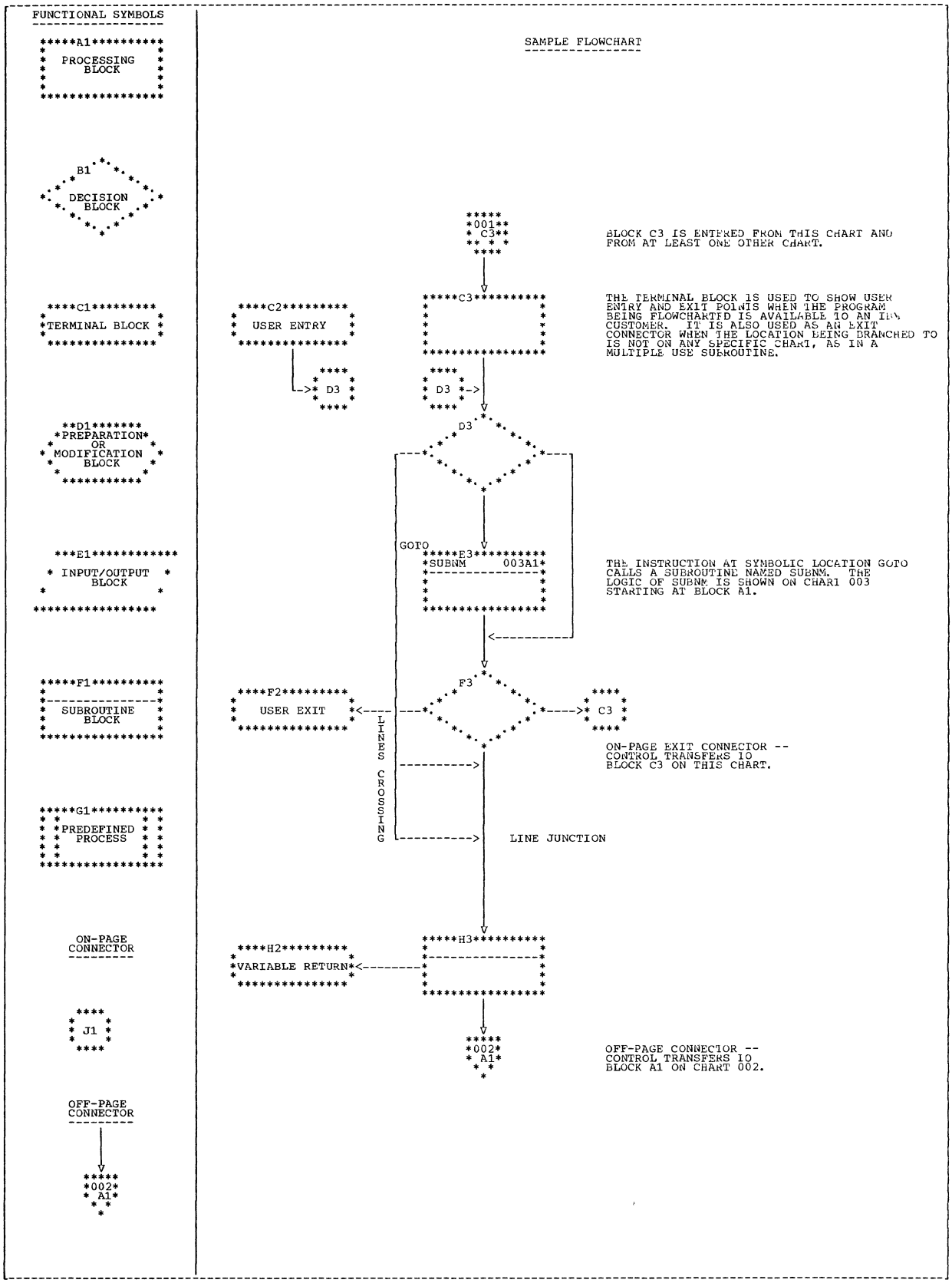
IEWLOADR - Level 4 (Part 1 of 2)

Routine	Purpose	Called Routines	Calling Conditions
ALLOCATE	Allocate table extent	IEWERROR	Table overflow
CESDENT	Get CESD entry from free entry list or prime storage	ALLOCATE	No free entries on list
CESDSRCH	Search CESD for input name	MATCHED NOMATCH	If name is found If name is not found
CHECKEP	Check CESD entry for specified entry point	none	
ENTER	Enter information in CESD entry for PC or SD	IEWERROR	If program is too large
IEWBUFFR	Buffer management	FREECORE GETCORE	If previous or current (not the first) allocation request is for object module If no previously allocated area is large enough for current request
IEWERROR	Handles error messages, severity code 4 errors	IEWLPRNT IEWTERM	If SYSLOUT data set is open If the TERM option is specified
IEWLCNVT	Convert binary quantity to hex	none	
IEWLEND	Process END card, reinitialize for next module	TRANSID IEWERROR	If END card specifies entry point address If error occurs in END card processing
IEWLERTN	Relocate all adcons indicated by RLD chain	IEWERROR	Invalid 2-byte adcon
IEWLESD	Create CESD from input ESD/CESD	LOADPROC CESDSRCH TRANSLAT CESDENT ENTER CHECKEP MATERSD2 TRANSID	If input is a load module Input entry is not NULL or PC If NULL entry is made If PC or LR entry is required If PC entry is required If PC entry is required If PC entry is required If LD/LR is received
IEWLMAP	Create Map entry for referenced location in loaded program	IEWLPRNT IEWLCVNT	Always Always
IEWLPRNT	Print output to SYSLOUT data set	RDCHECK WRWRITE WTCHECK	If DECB was previously written Always Always
IEWLREAD	Handle request for data	RDREAD RDCHECK	Always Always

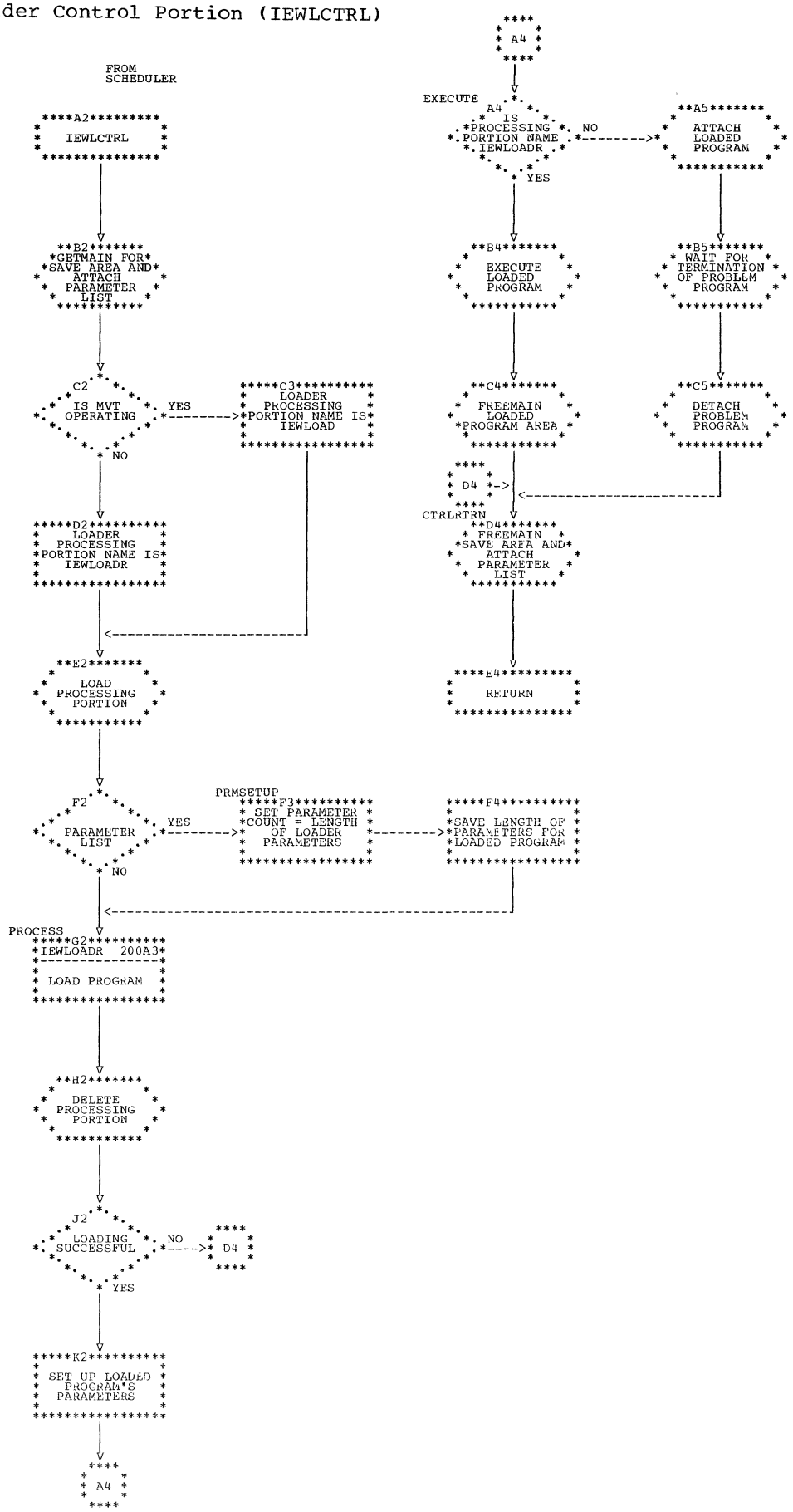
IEWLOADR - Level 4 (Part 2 of 2)

Routine	Purpose	Called Routines	Calling Conditions
IEWLRLD	Relocate adcons indicated by RLD entries received or chain RLDs off CESD entry for R pointer	TRANSID ALLOCATE IEWLERTN	Always If no free RLD entry is available If relocation is possible or if delinking is required
IEWLTXT	Move object module text to correct spaces	TRANSID RELOREAD IEWERROR	Always Always If invalid ID is received
IEWTERM	Print output to SYSTERM data set	WTWRITE WTCHECK	Always Always
LMTXT	Read load module text into main storage	TRANSID IEWLREAD IEWERROR PROCEOM	Always Unless record is to be skipped If text record not received Always
LOADPROC	Preliminary processing for load module CESD	CESDENT	If entry type is PC,SD,LR
MATERSD2	Test length and request map entry	CHAINING	Always
PROCEOM	Go to process end-of-module	IEWLEND	Always
PSEUDOR	Assign displacements to pseudo registers	IEWLPRNT FINISHUP IEWLMAP IEWLERTN	If displacement is assigned Always If displacement is assigned If displacement is assigned
RDCHECK	Check DECB	none	
RDREAD	Read input using DECB information	none	
RELOREAD	Go to IEWLREAD for more input	IEWLREAD	Always
TRANSID	Translate input ESD ID to CESD address	ALLOCATE IEWERROR	If new extent is required If table overflow or invalid ID occurs
TRANSLAT	Make a translation table entry	TRANSID	Unless LD entry
WTCHECK	Check DECB	none	
WTWRITE	Write output using DECB information	none	

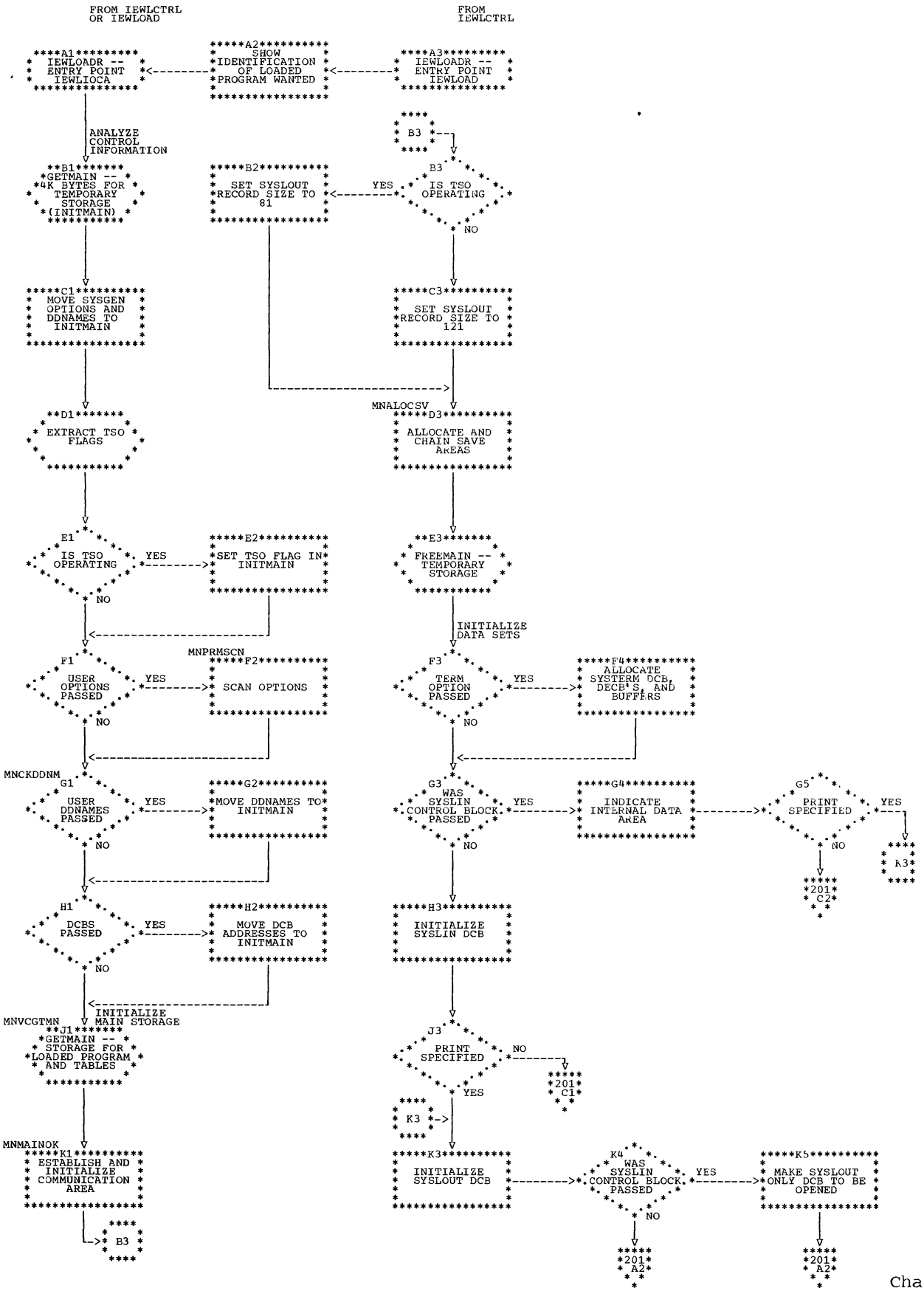
Chart 001. Sample Flowchart



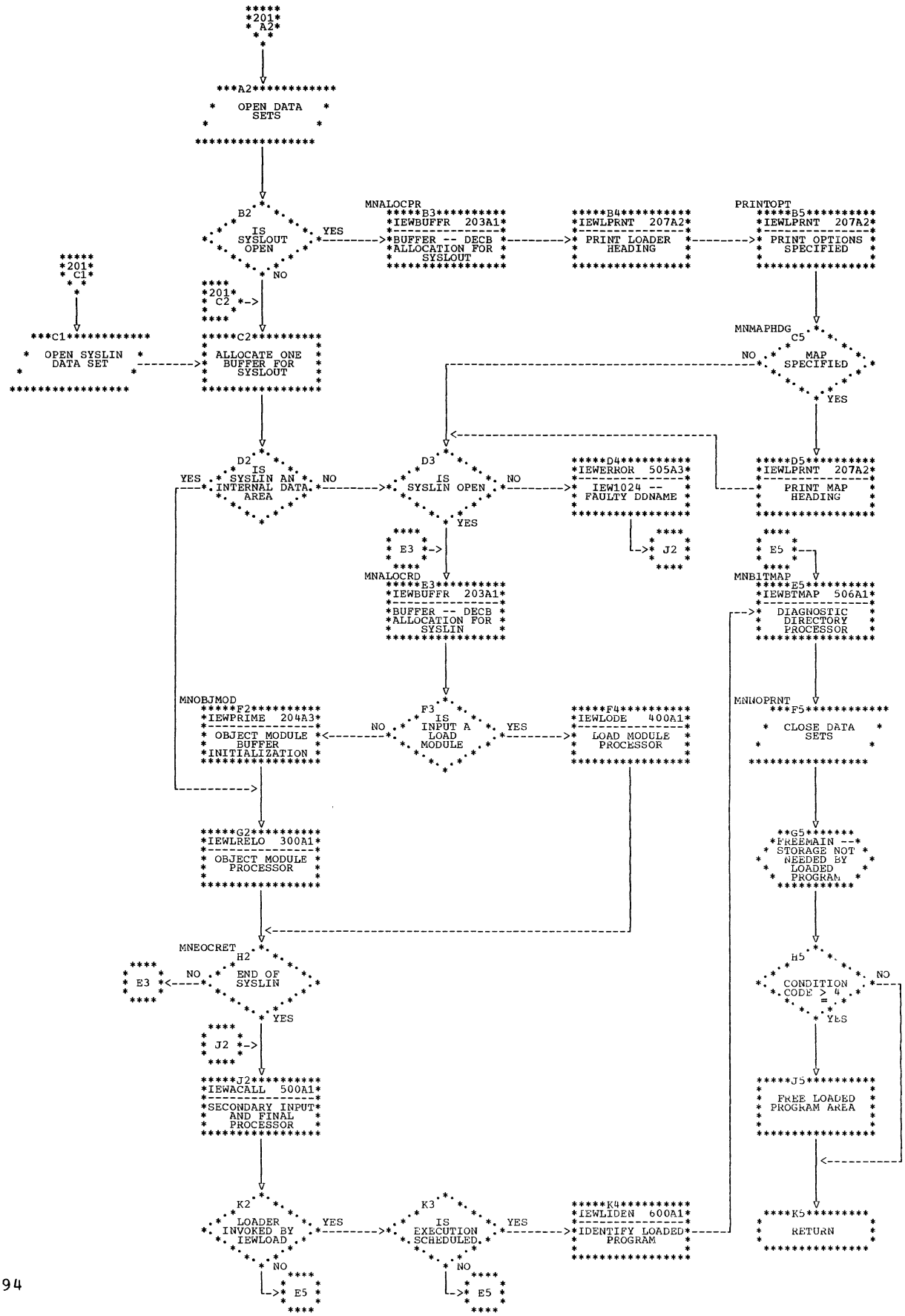
• Chart 100. Loader Control Portion (IEWLCTRL)



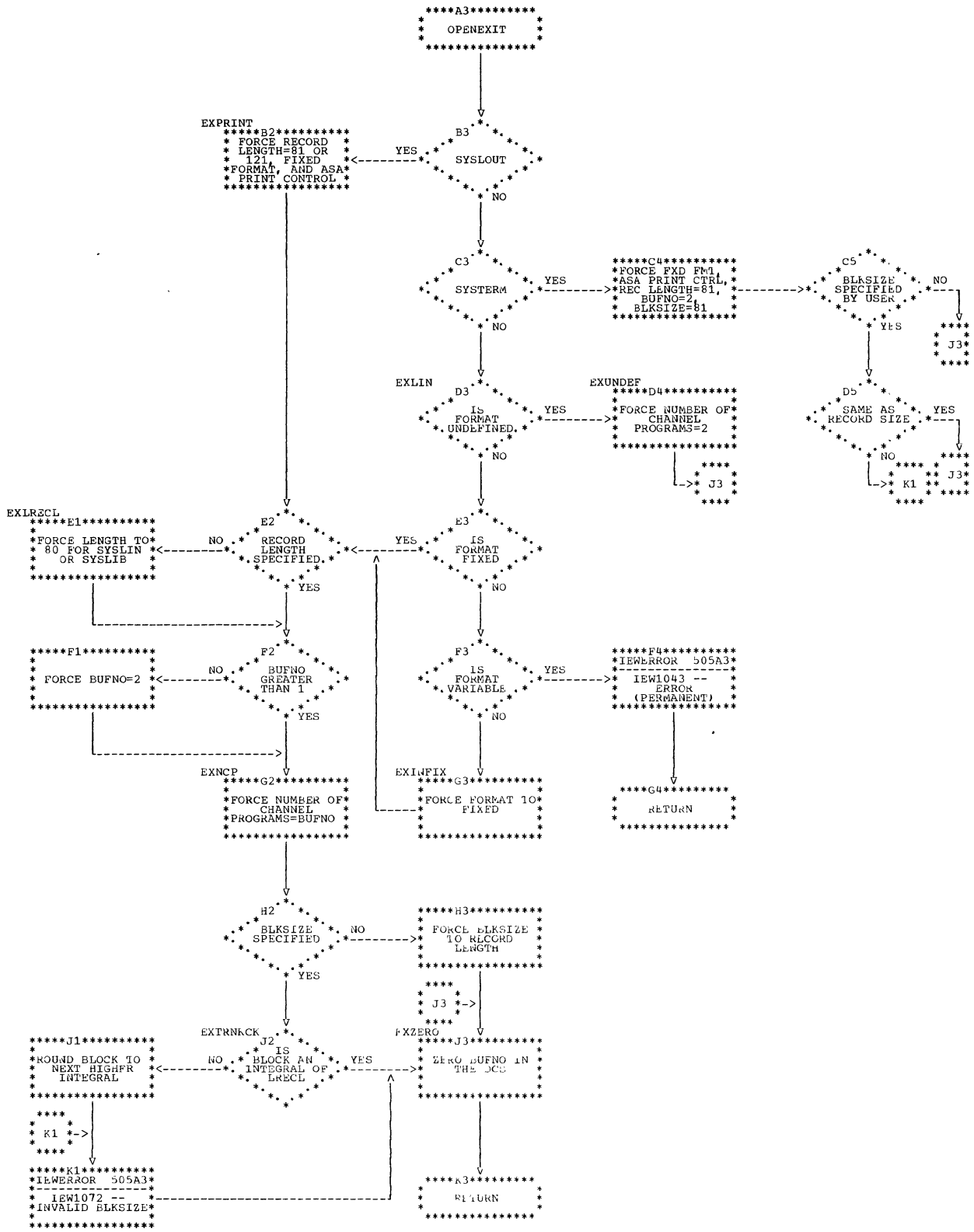
• Chart 200. Initialization, I/O, Control, Allocation Processing (IEWLIOCA) (Part 1 of 2)



• Chart 201. Initialization, I/O, Control, Allocation Processing (IEWLIOCA) (Part 2 of 2)



• Chart 202. DCB Exit Routine (OPENEXIT)



• Chart 203. Buffer Allocation Routine (IEWBUFFR)

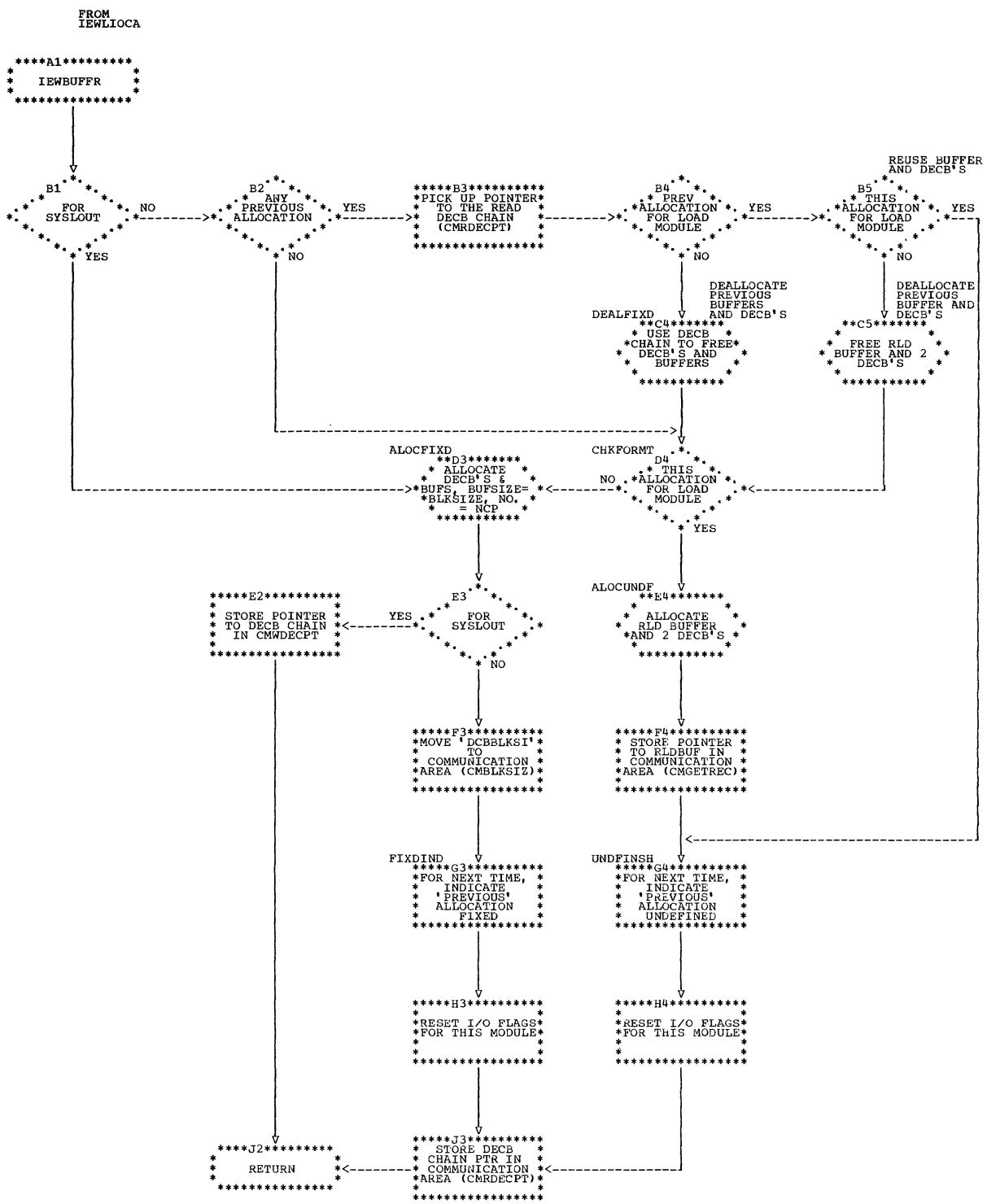
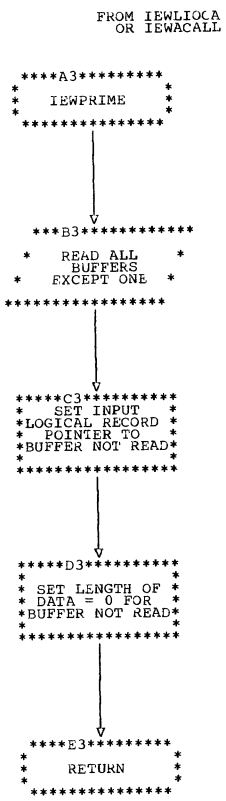
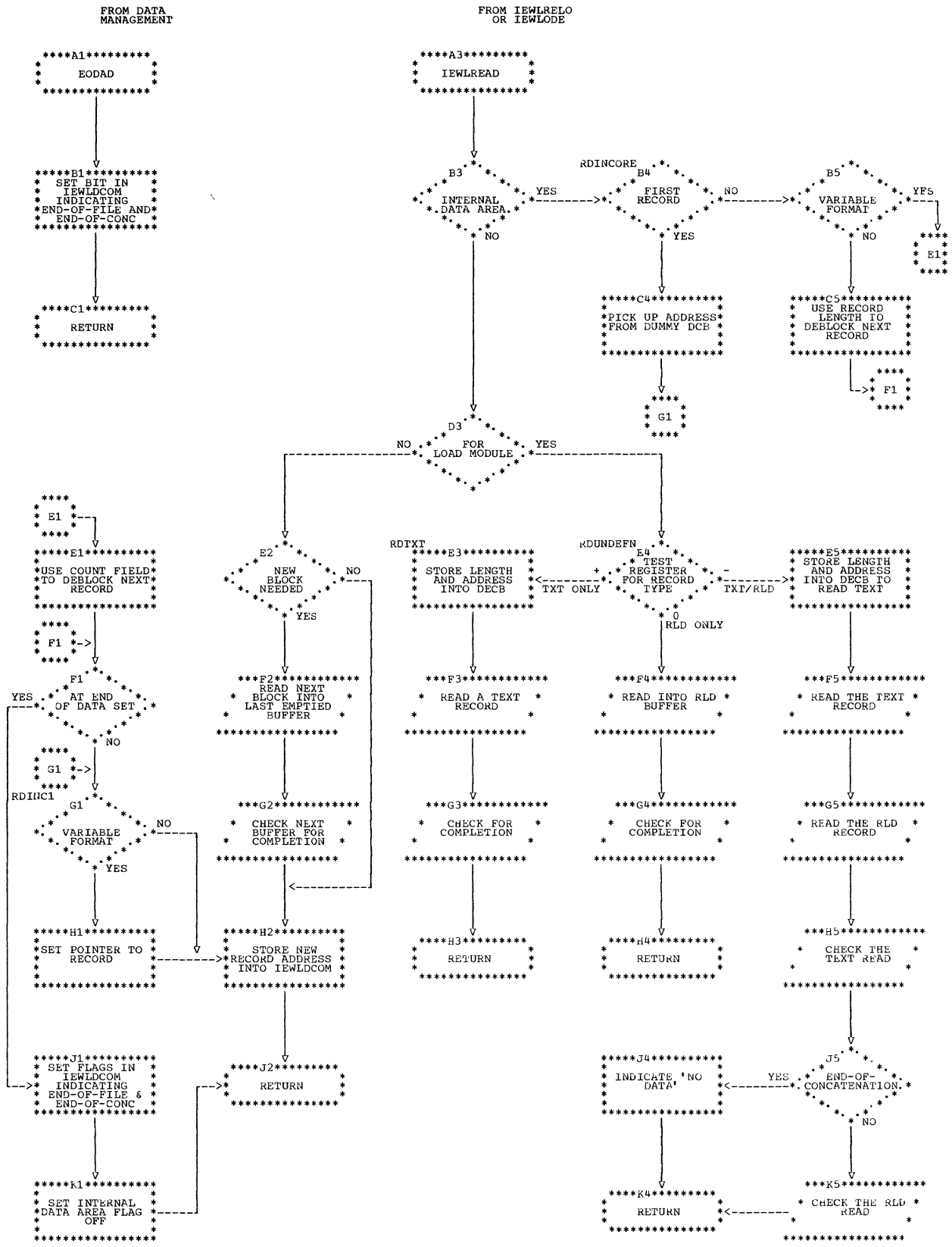


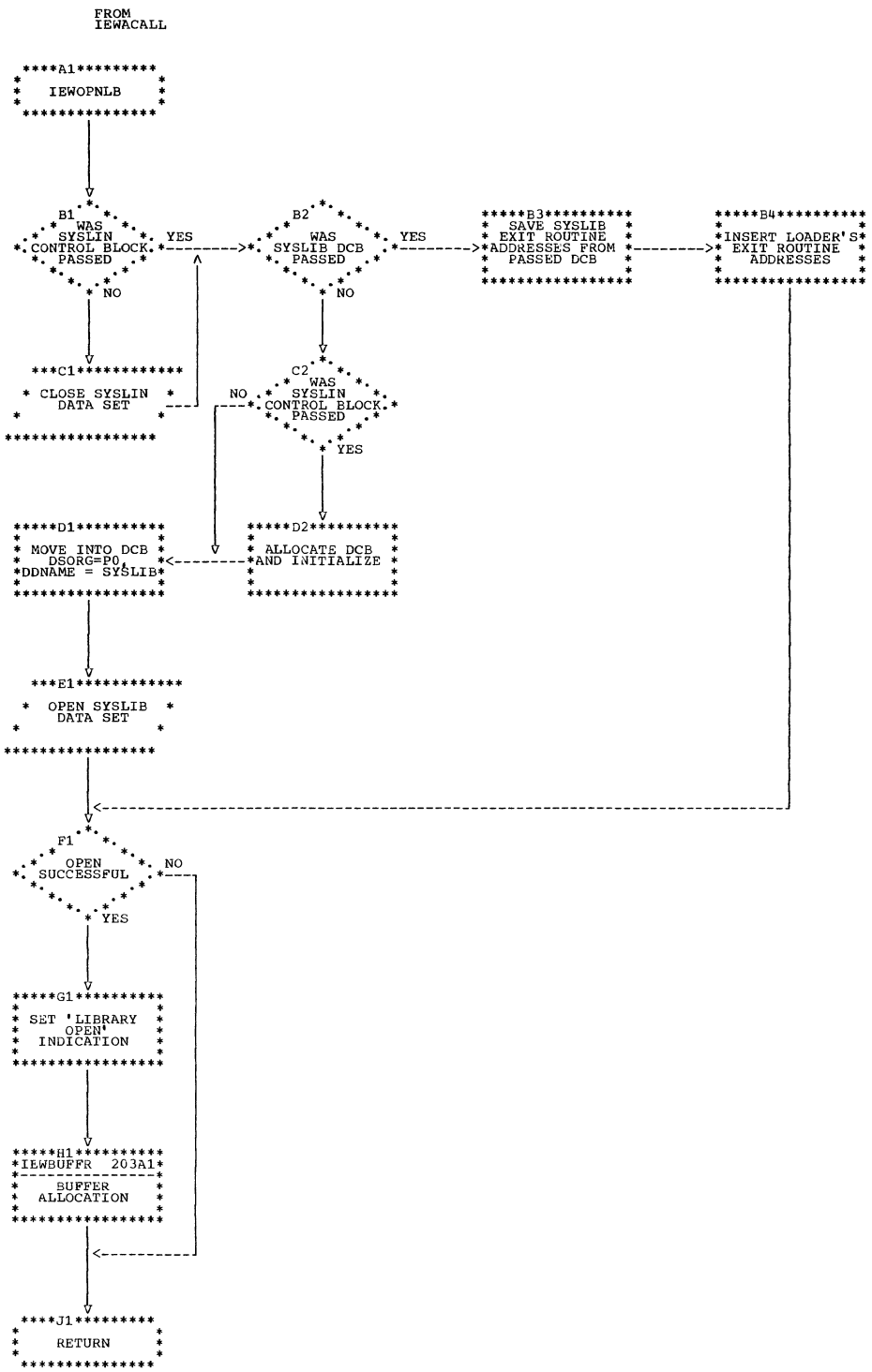
Chart 204. Object Module Buffer Prime Routine (IEWPRIME)



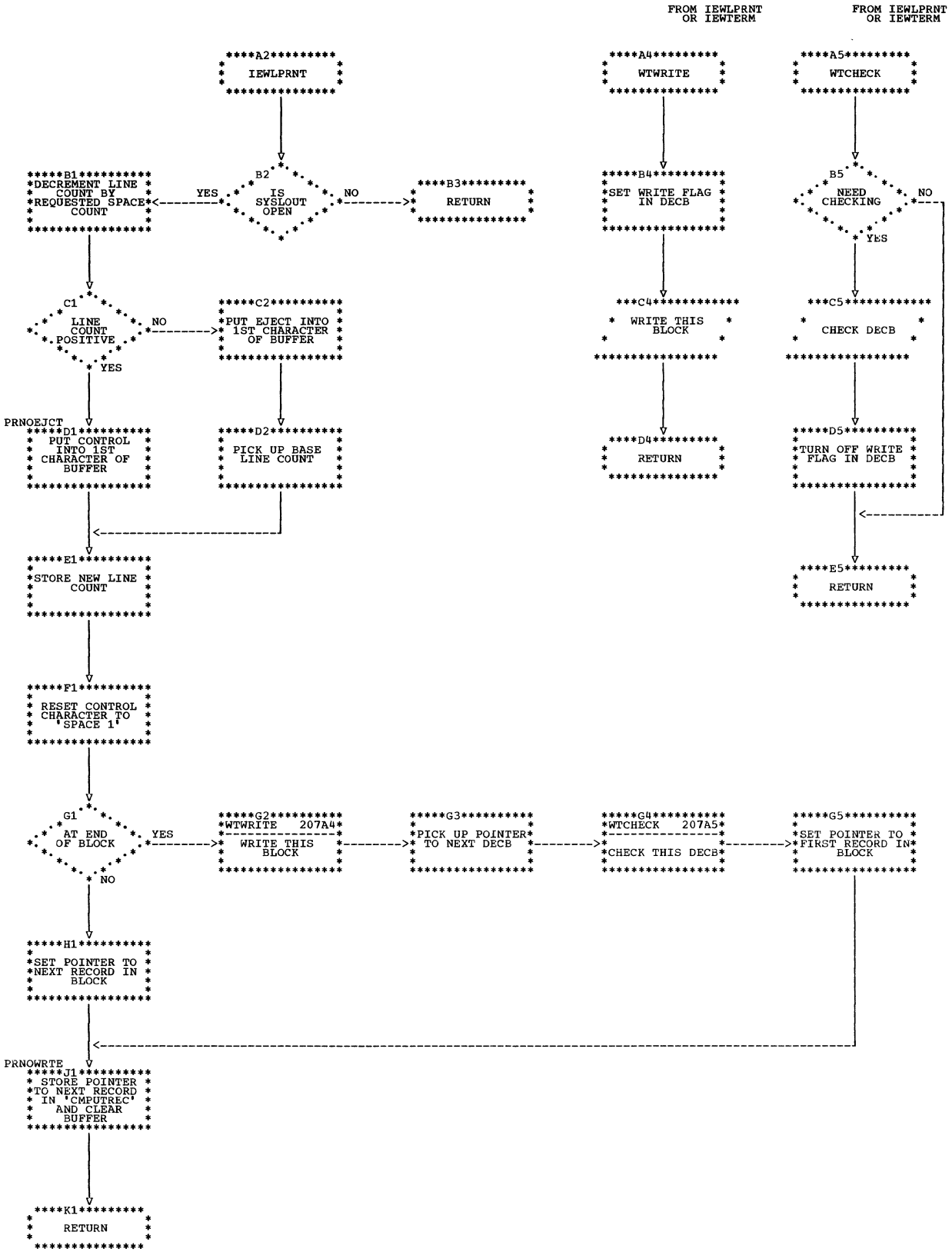
- Chart 205. Read Routine (IEWLREAD)



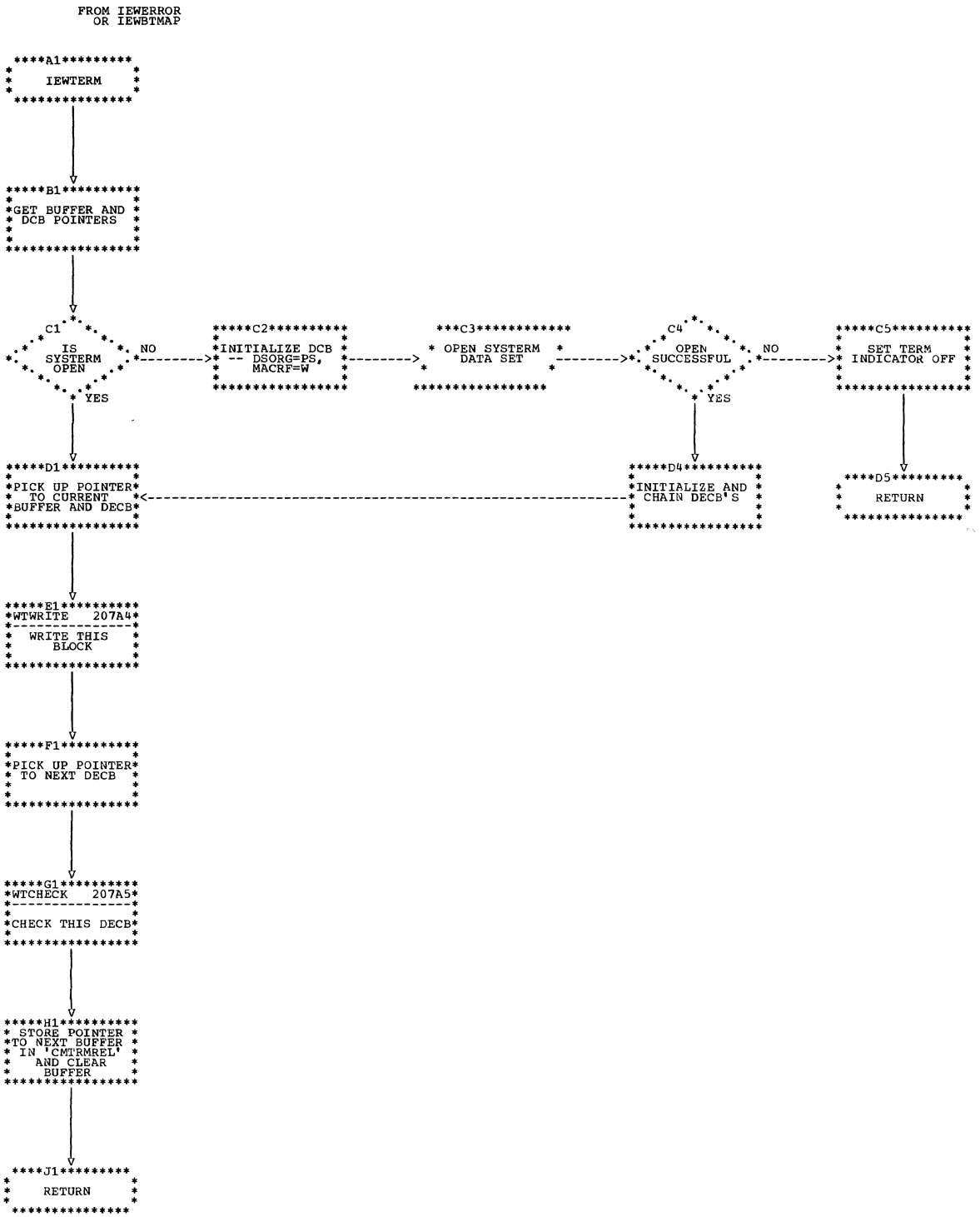
- Chart 206. Library Open Routine (IEWOPNLB)



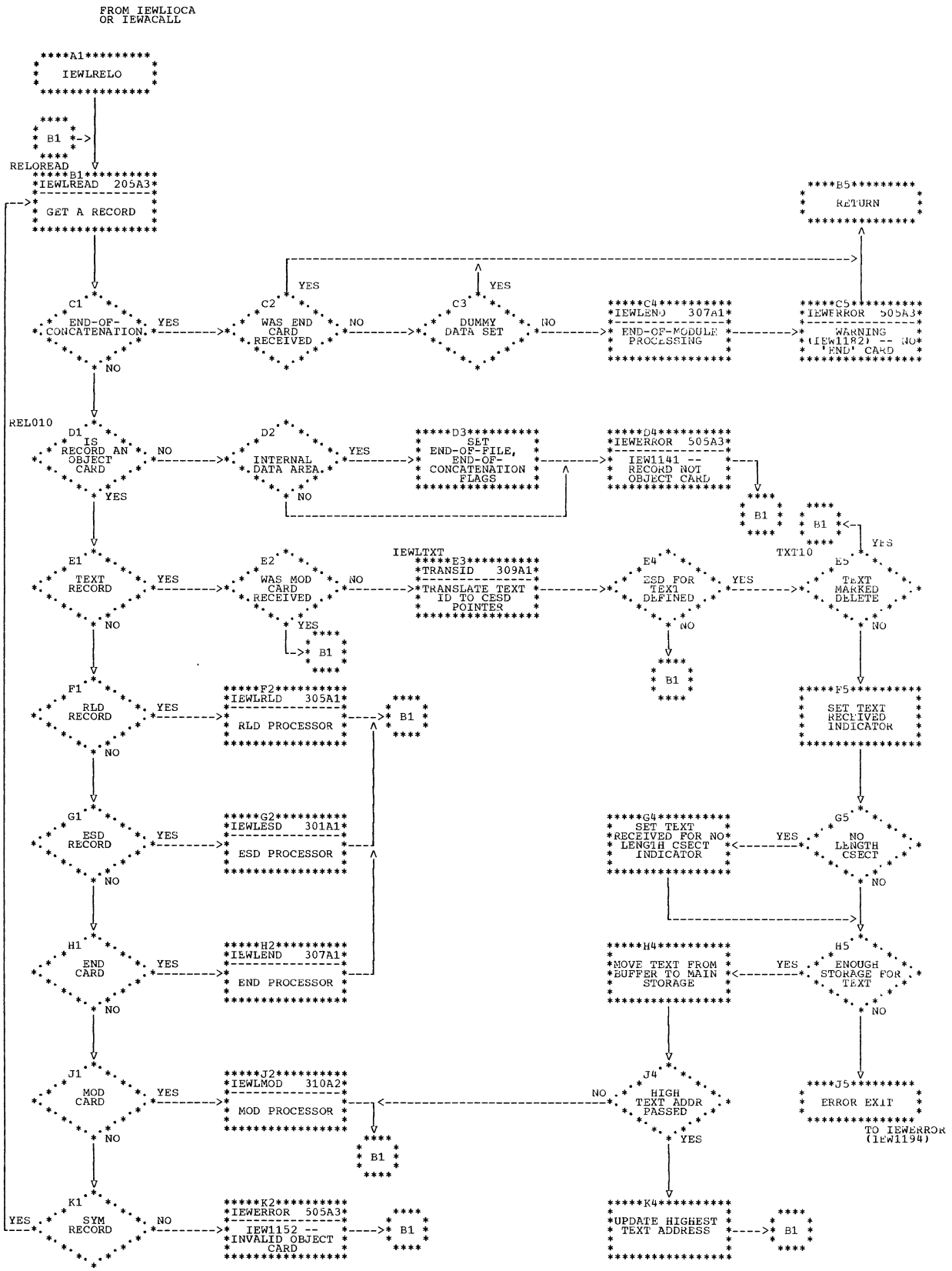
• Chart 207. Print Routine (IEWLPRNT), Write Routine (WTWRITE), Check Routine (WTCHECK)



- Chart 208. SYSTERM Routine (IEWTERM)



- **Chart 300. Object Module Processor (IEWLRELO)**



- Chart 301. ESD Processing Routine (IEWLESD) (Part 1 of 4)

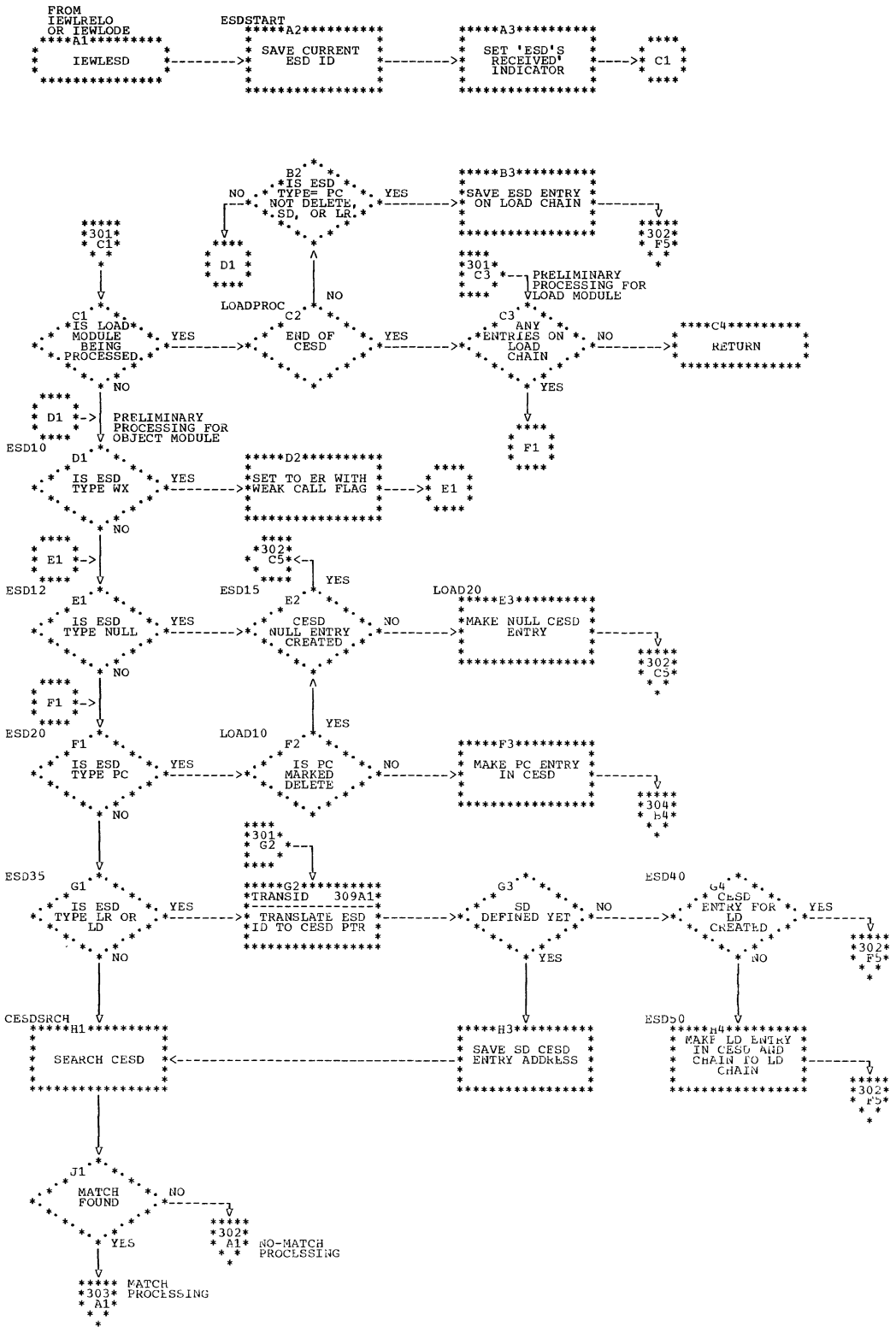
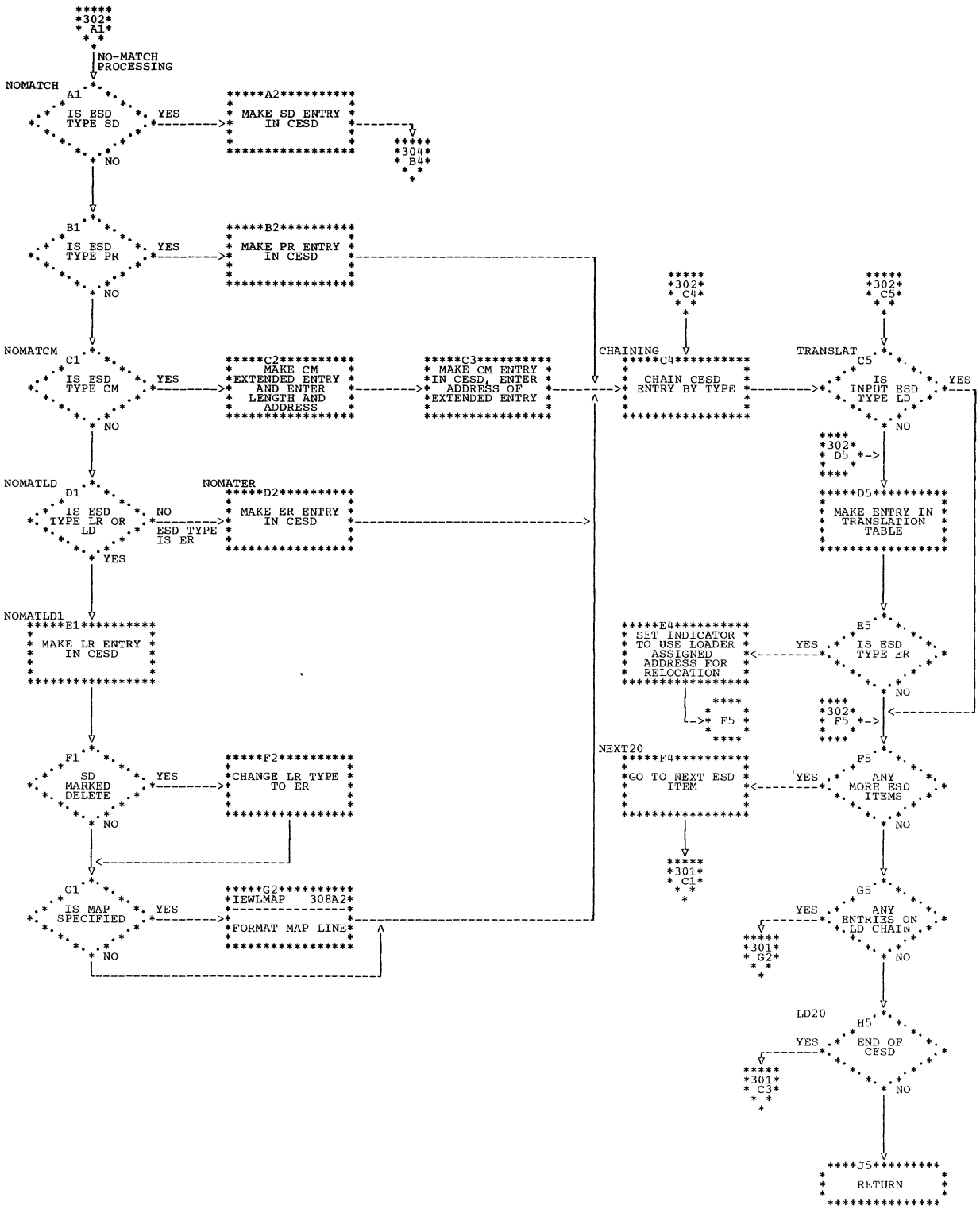


Chart 302. ESD Processing Routine (IEWLESD) (Part 2 of 4)



- Chart 303. ESD Processing Routine (IEWLESD) (Part 3 of 4)

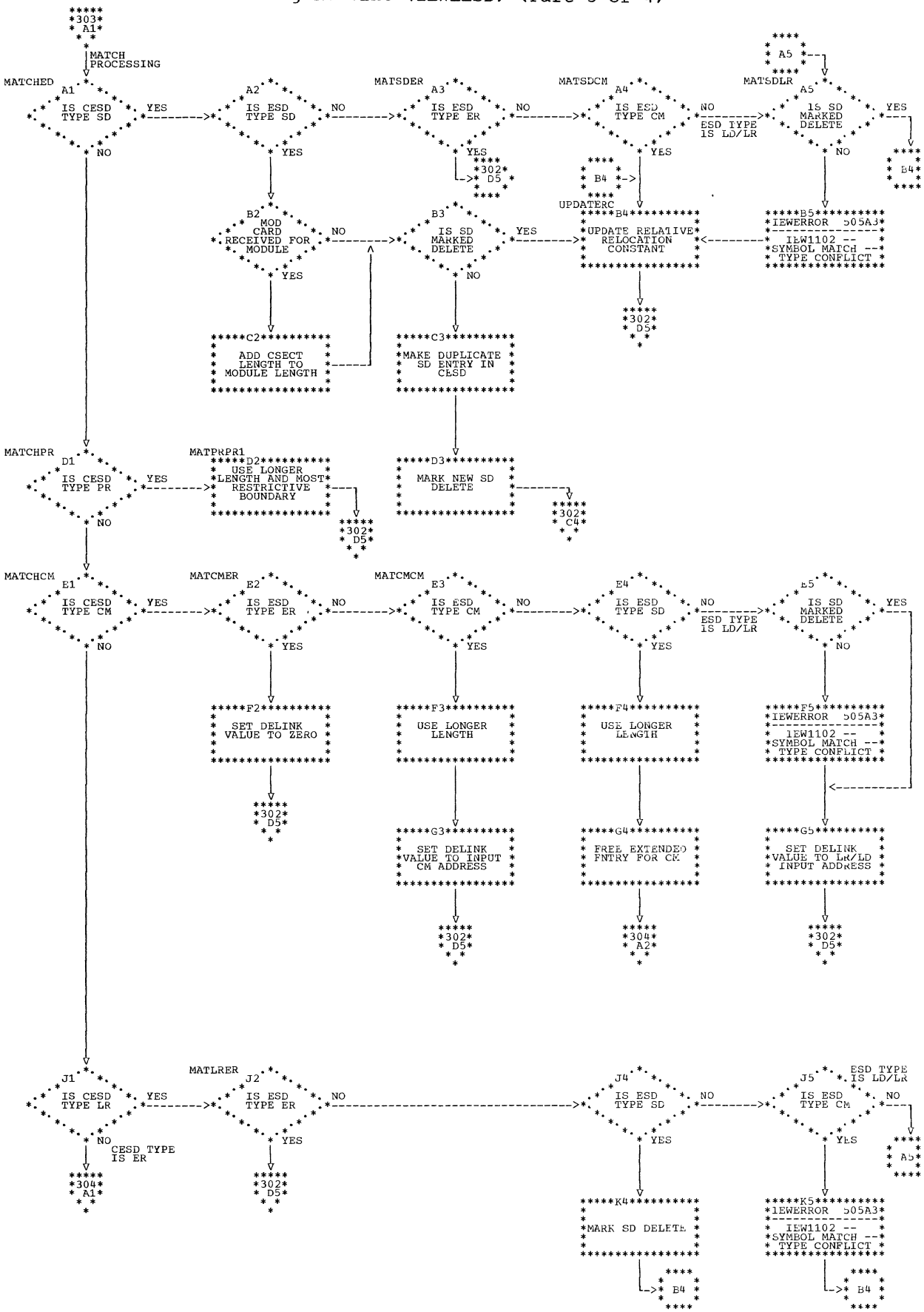
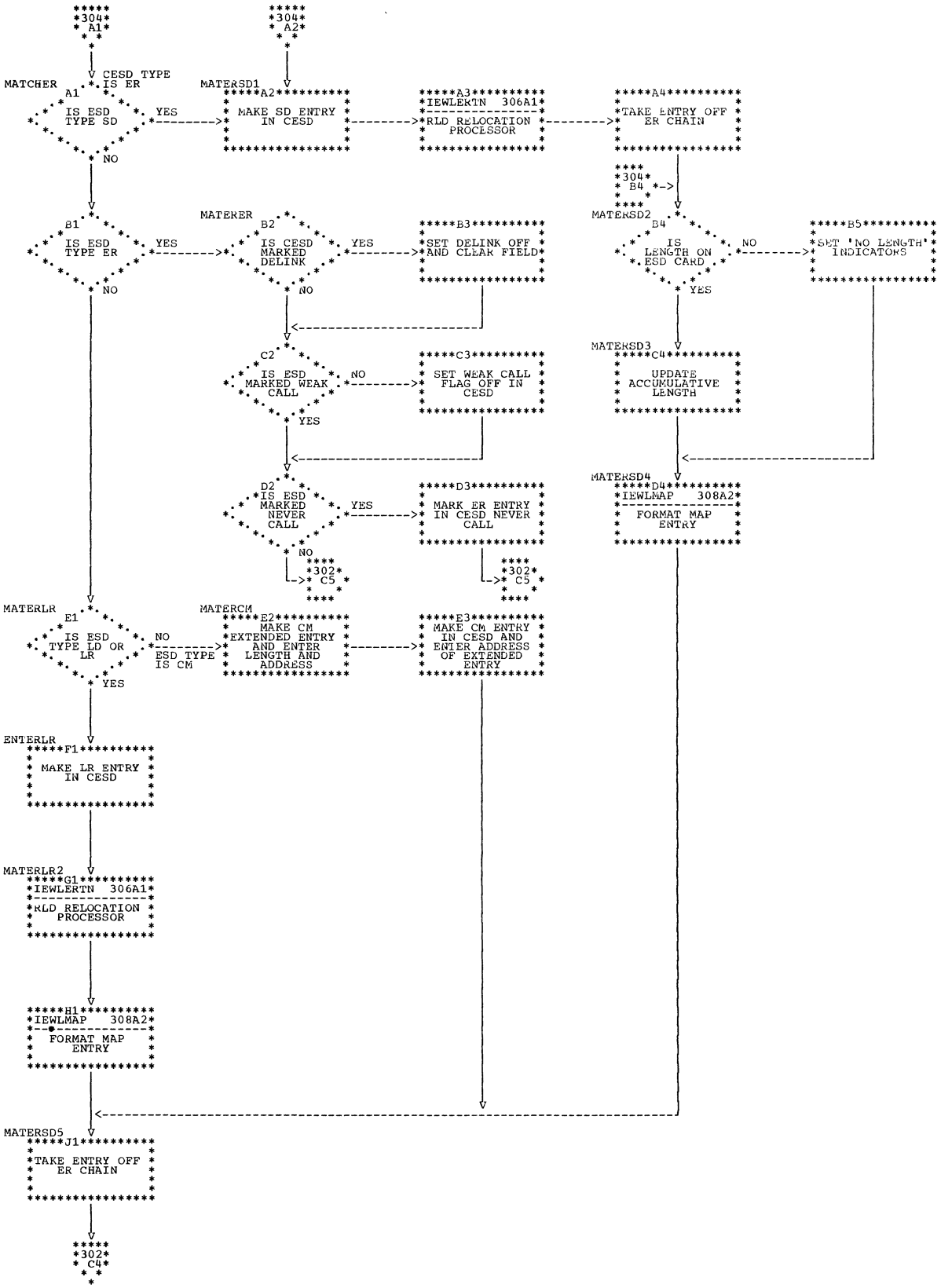
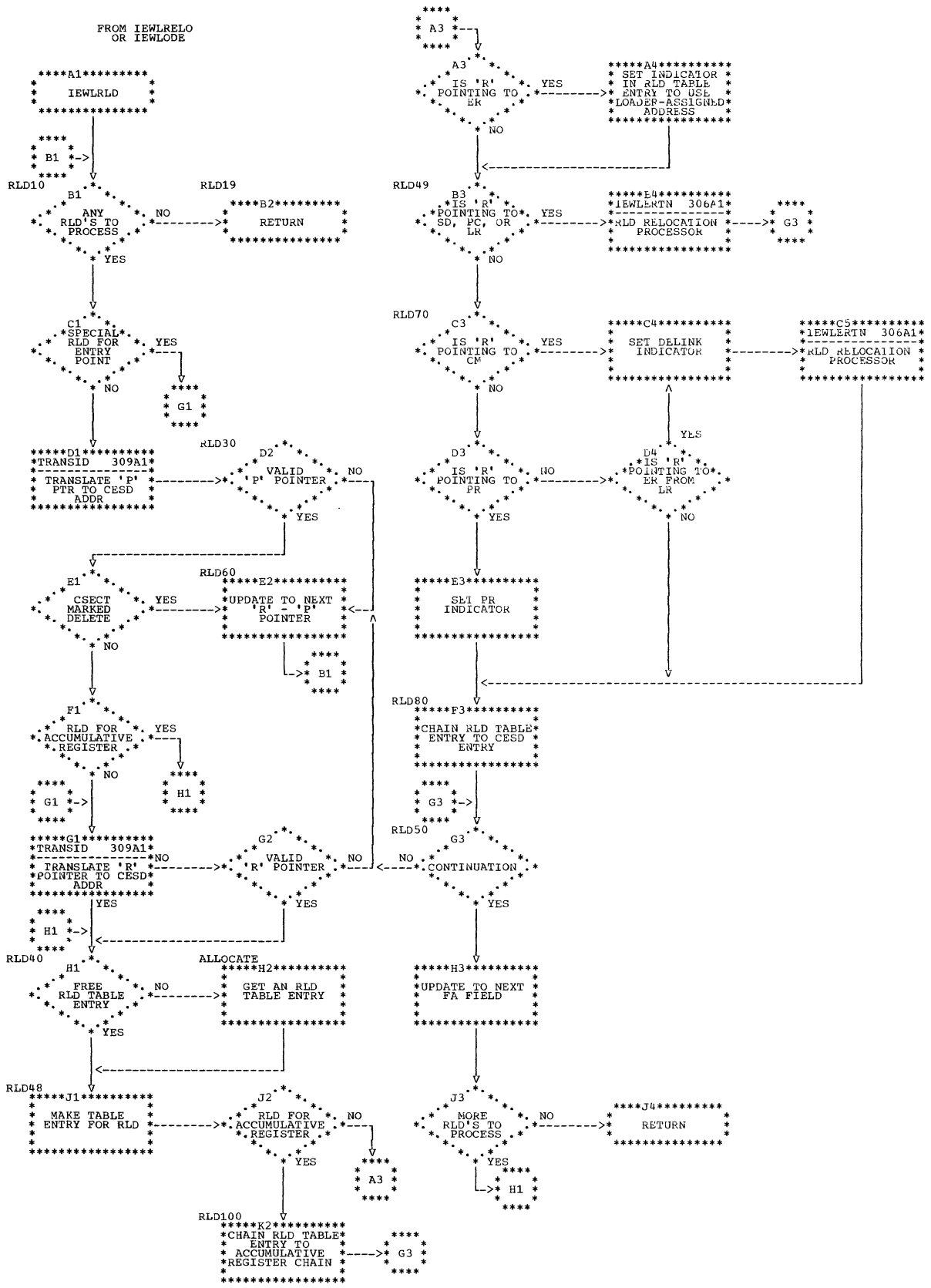


Chart 304. ESD Processing Routine (IEWLESD) (Part 4 of 4)



- Chart 305. RLD Processing Routine (IEWLRLD)



- Chart 306. Address Constant Relocation Routine (IEWLERTN)

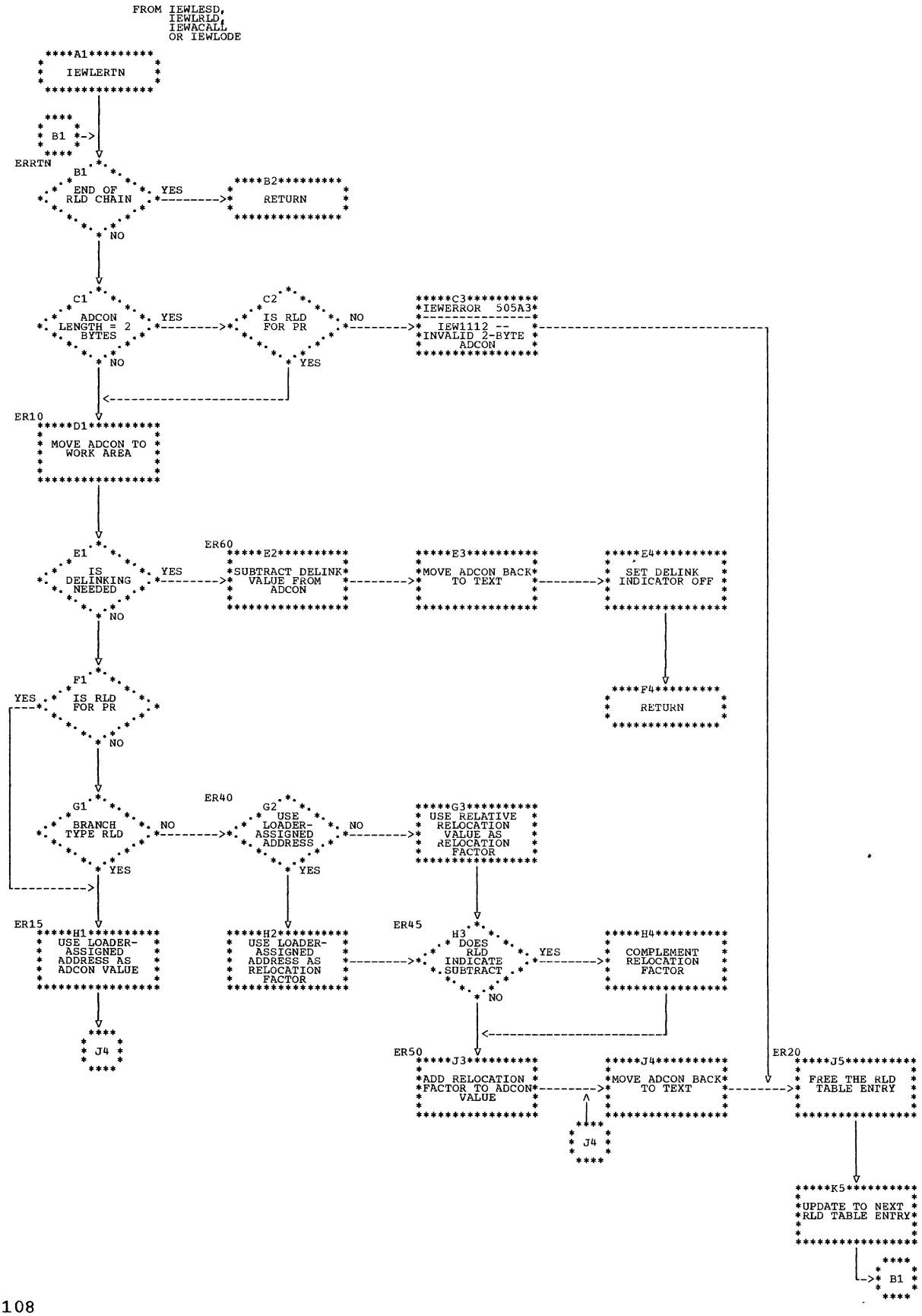
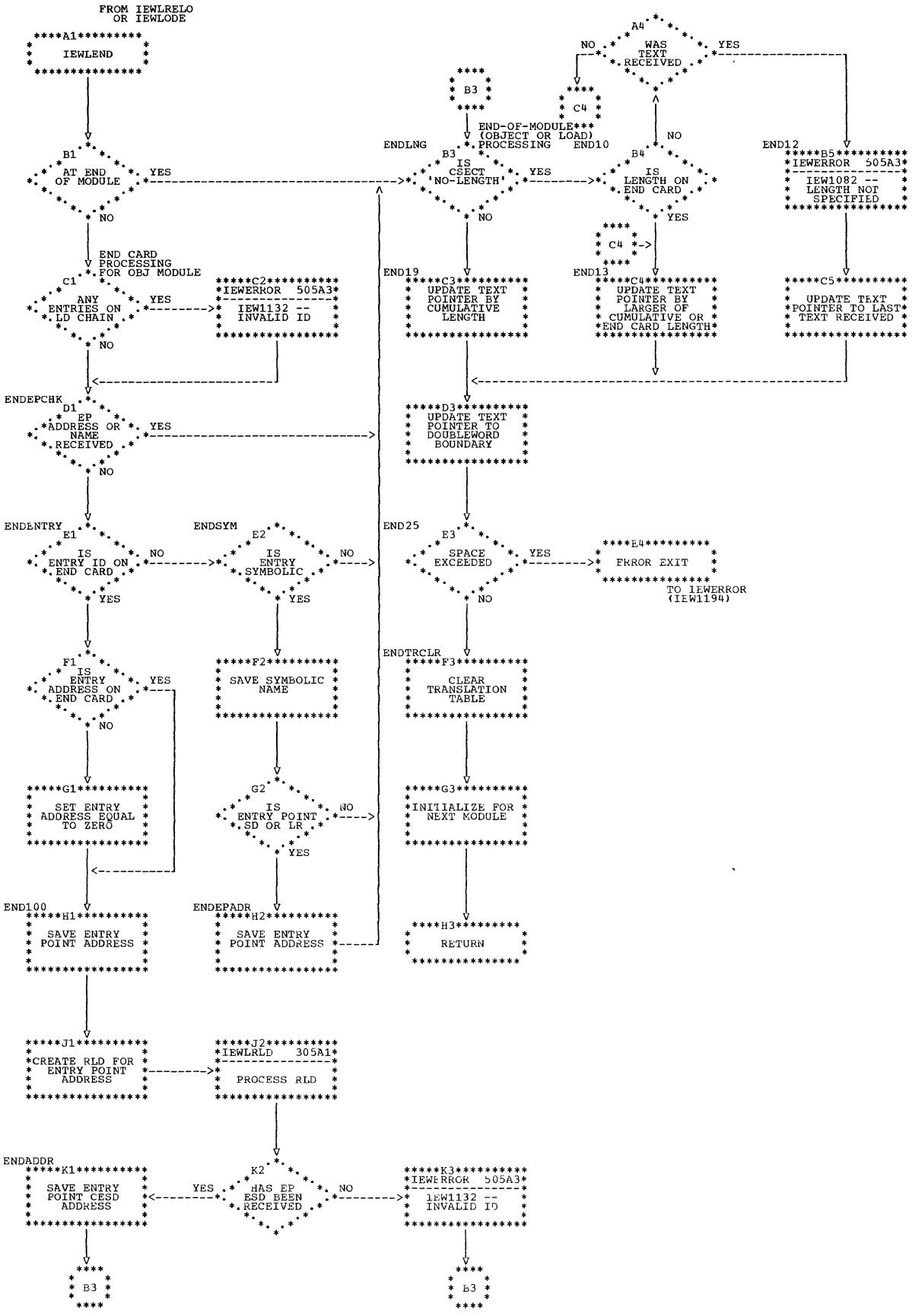


Chart 307. End Processing Routine (IEWLEND)



- **Chart 308. Map Routine (IEWLMAP)**

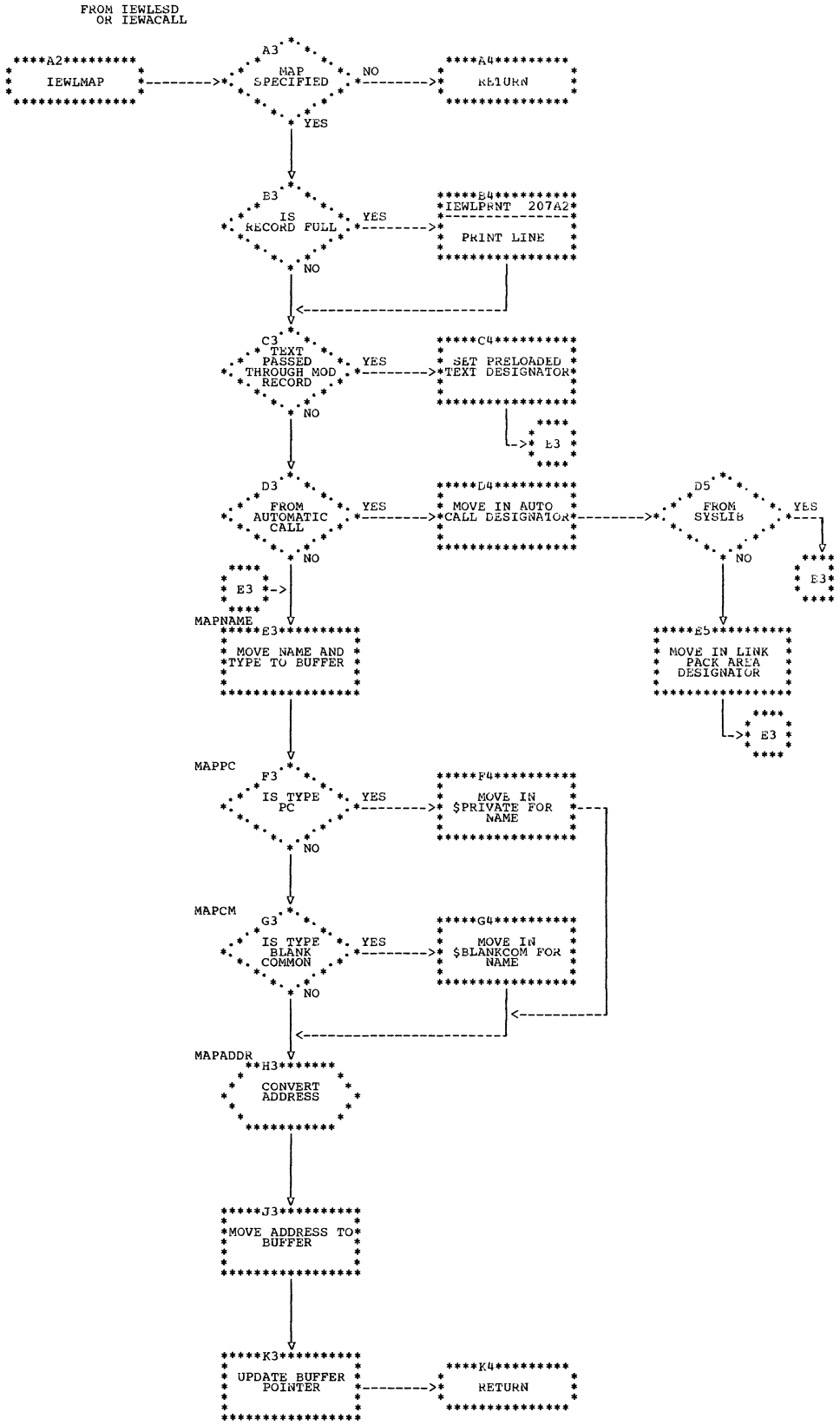
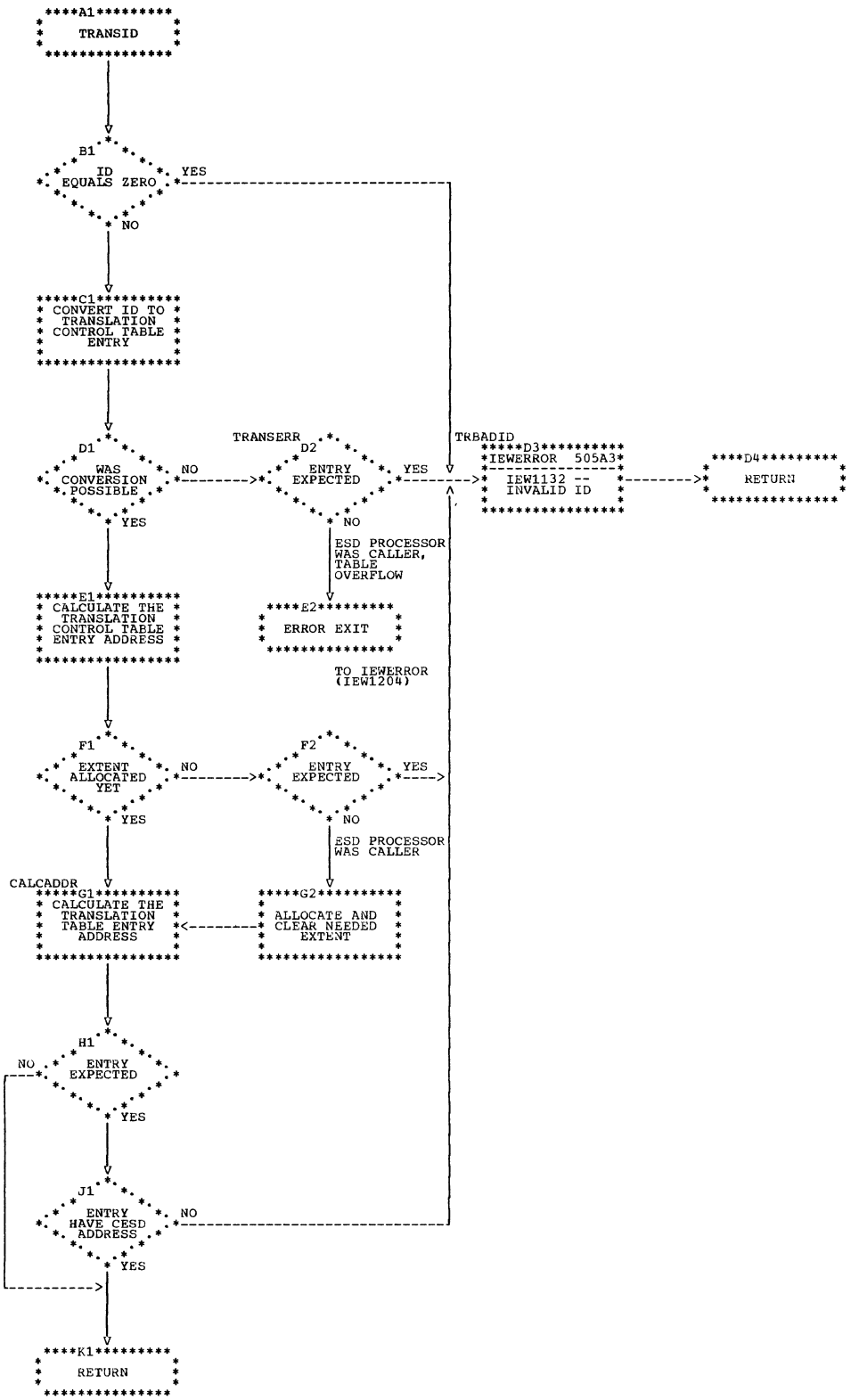
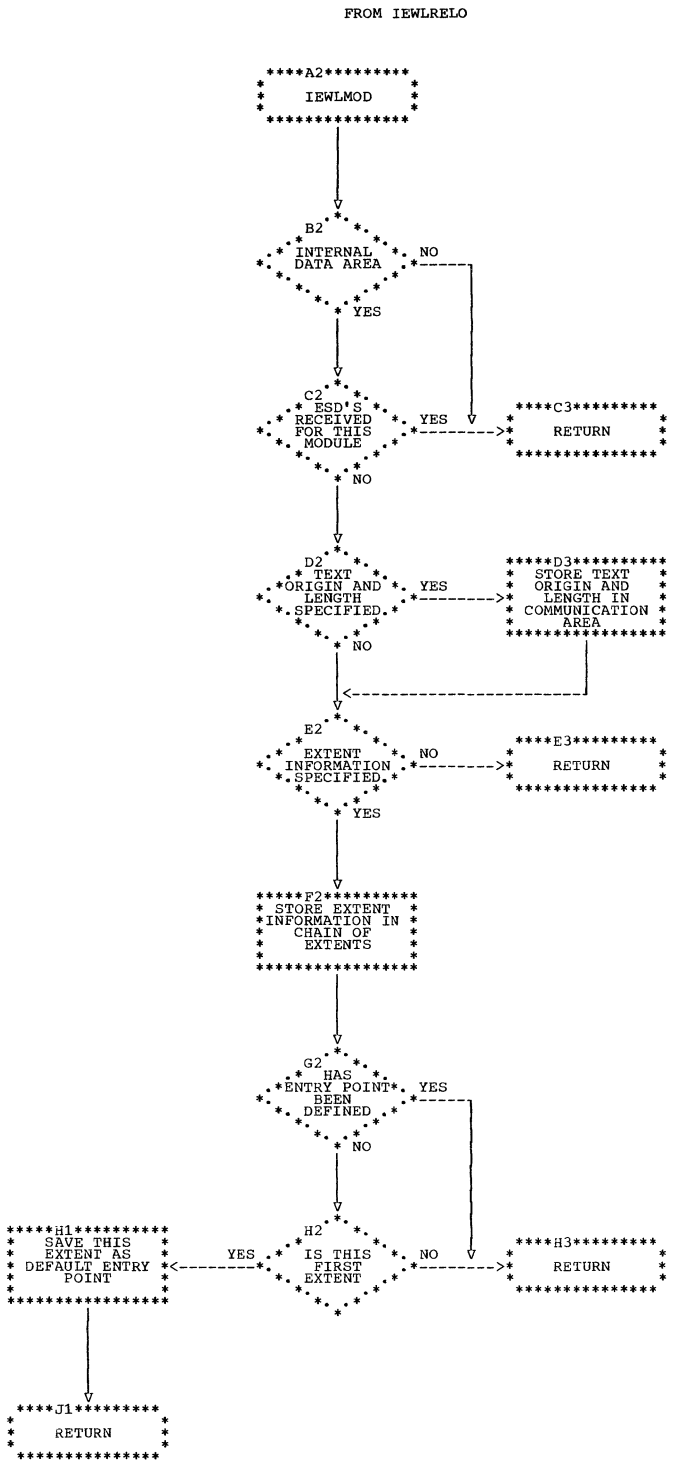


Chart 309. Translation Routine (TRANSID)



- Chart 310. MOD Processing Routine (IEWLMOD)



- Chart 400. Load Module Processing Routine (IEWLODE)

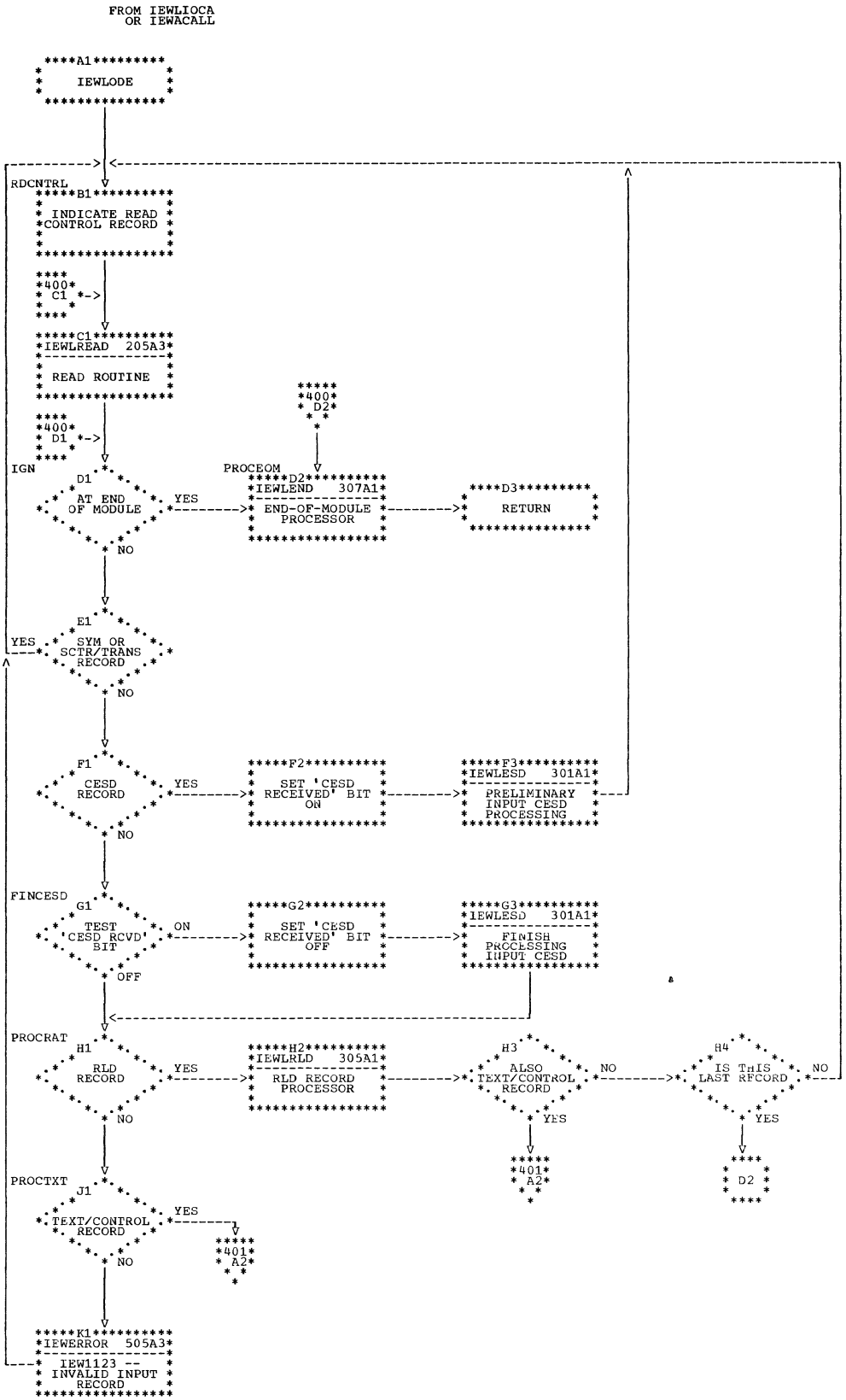
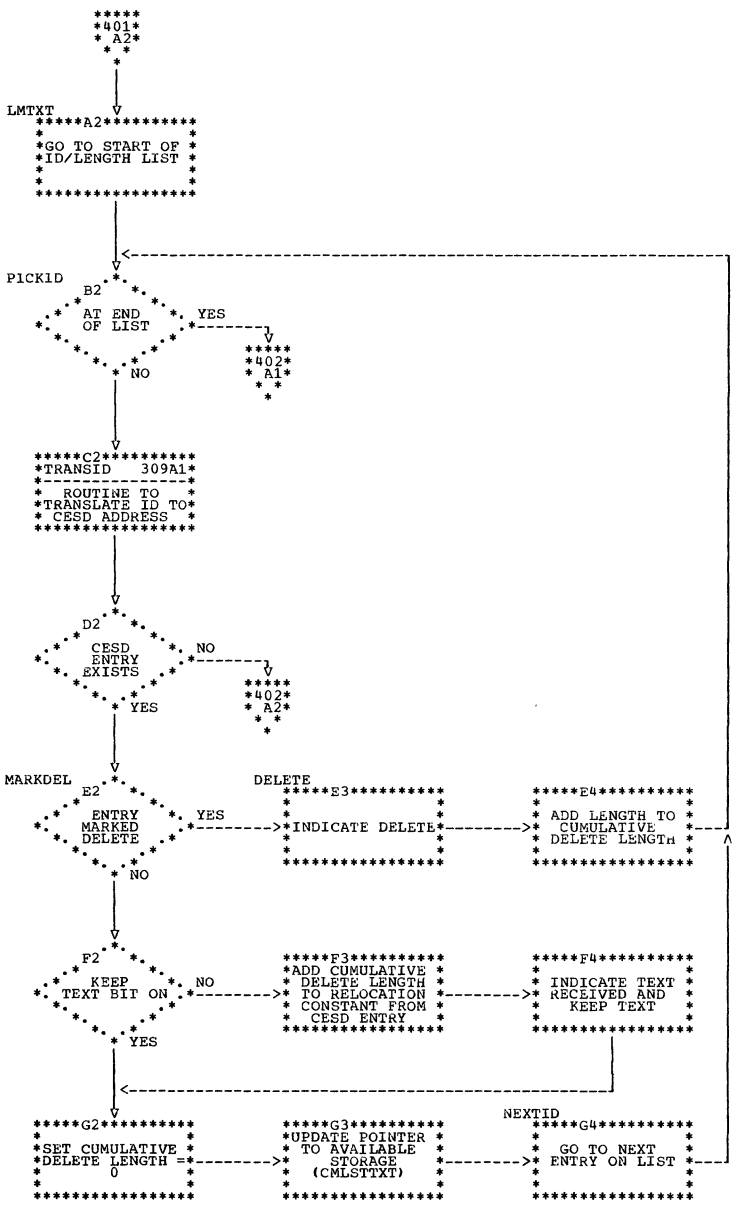


Chart 401. Load Module Text Processing Routine (LMTXT) (Part 1 of 3)



- Chart 402. Load Module Text Processing Routine (LMTXT) (Part 2 of 3)

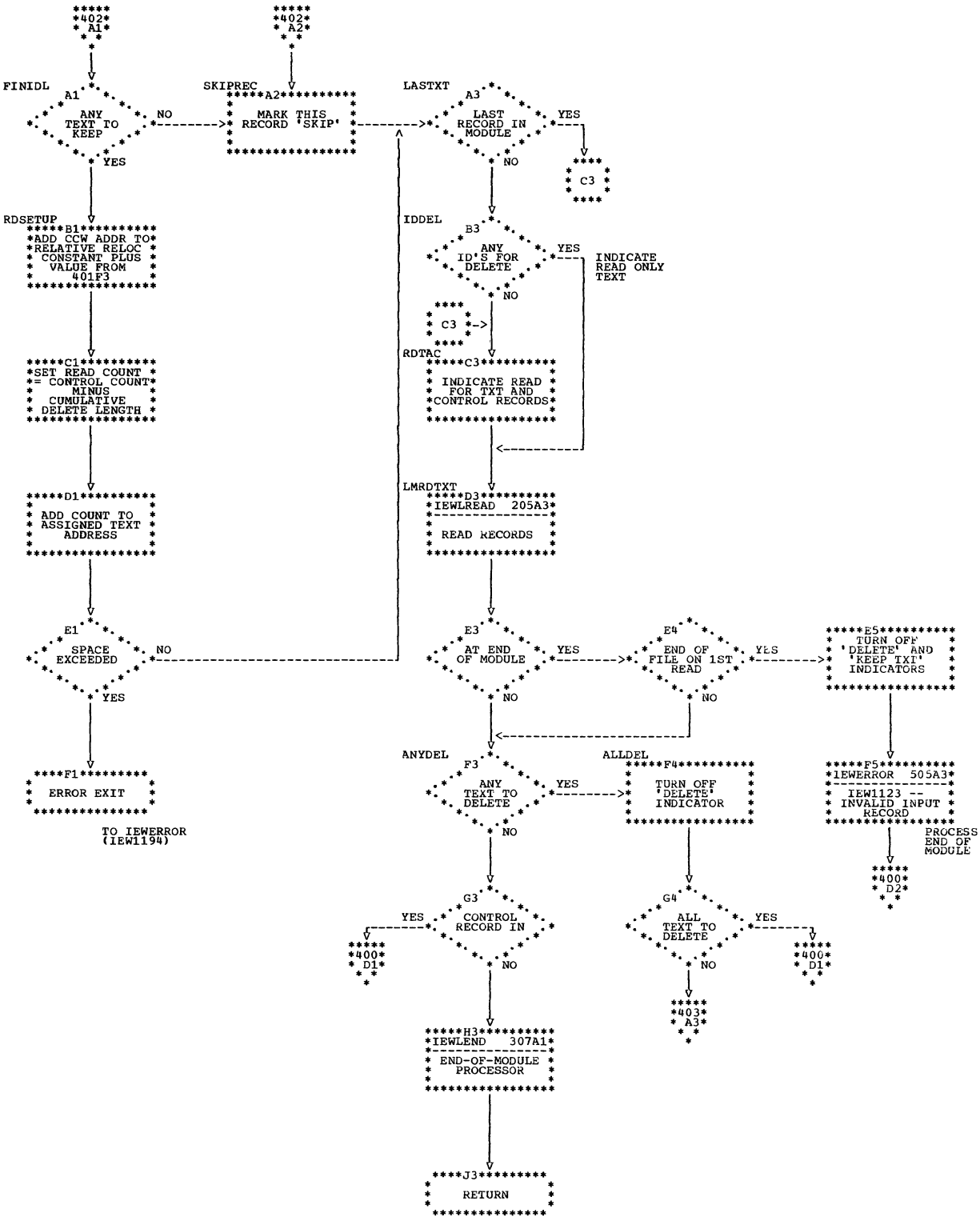
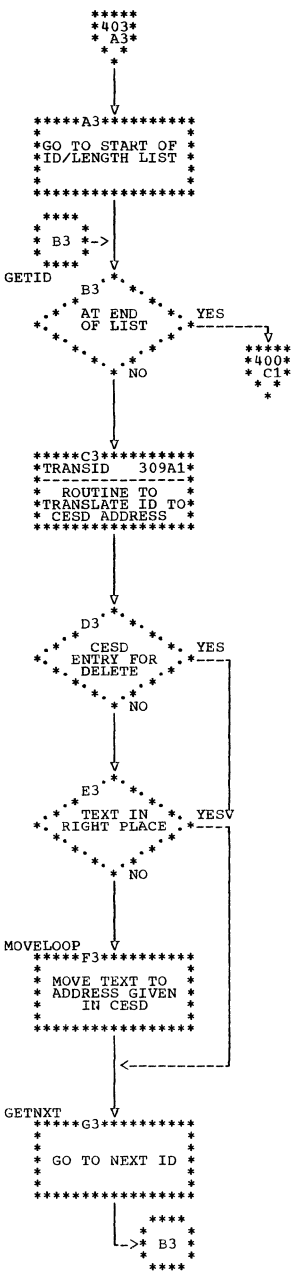


Chart 403. Load Module Text Processing Routine (LMTXT) (Part 3 of 3)



- Chart 500. Secondary Input Processing Routine (IEWACALL) (Part 1 of 5)

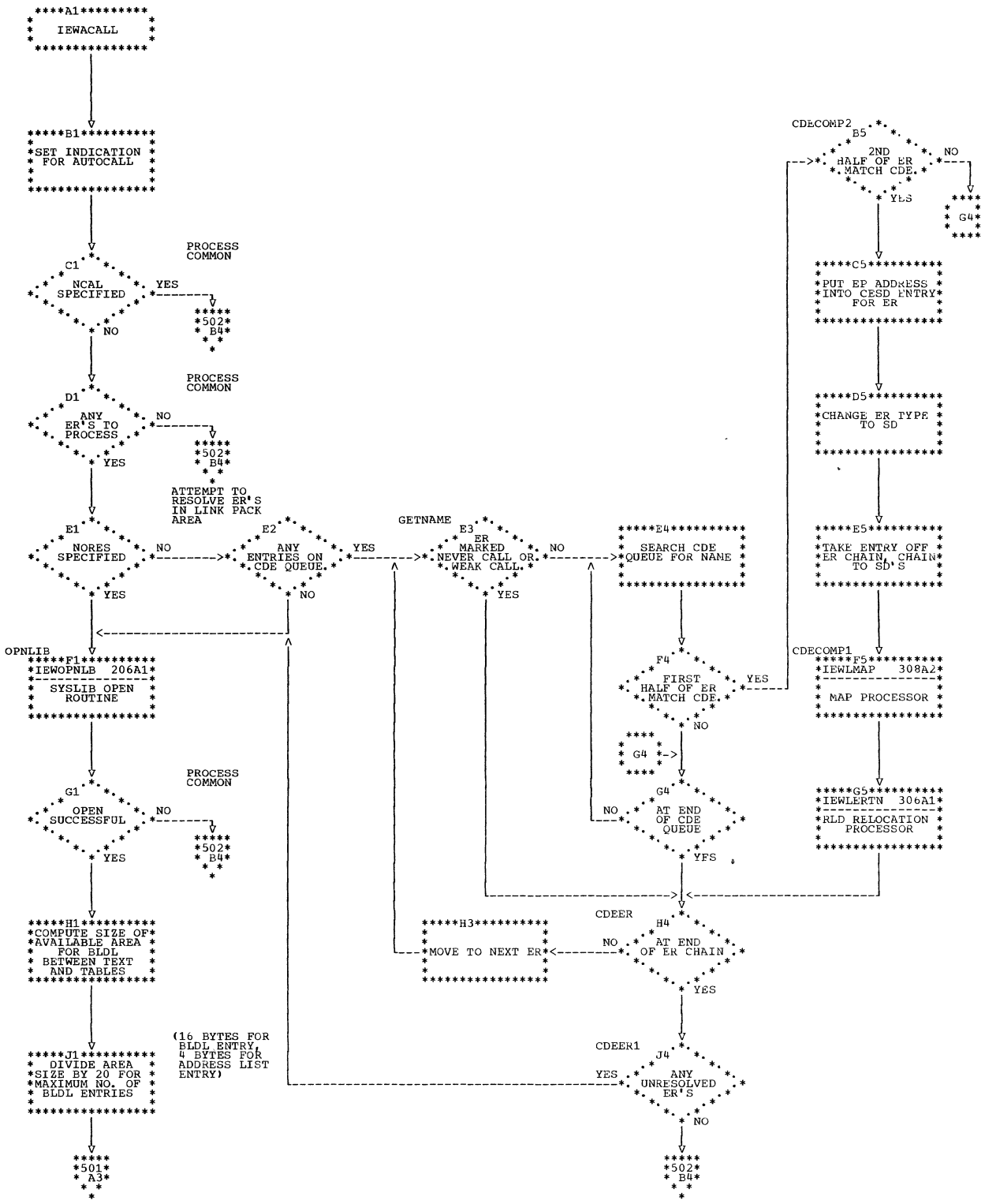
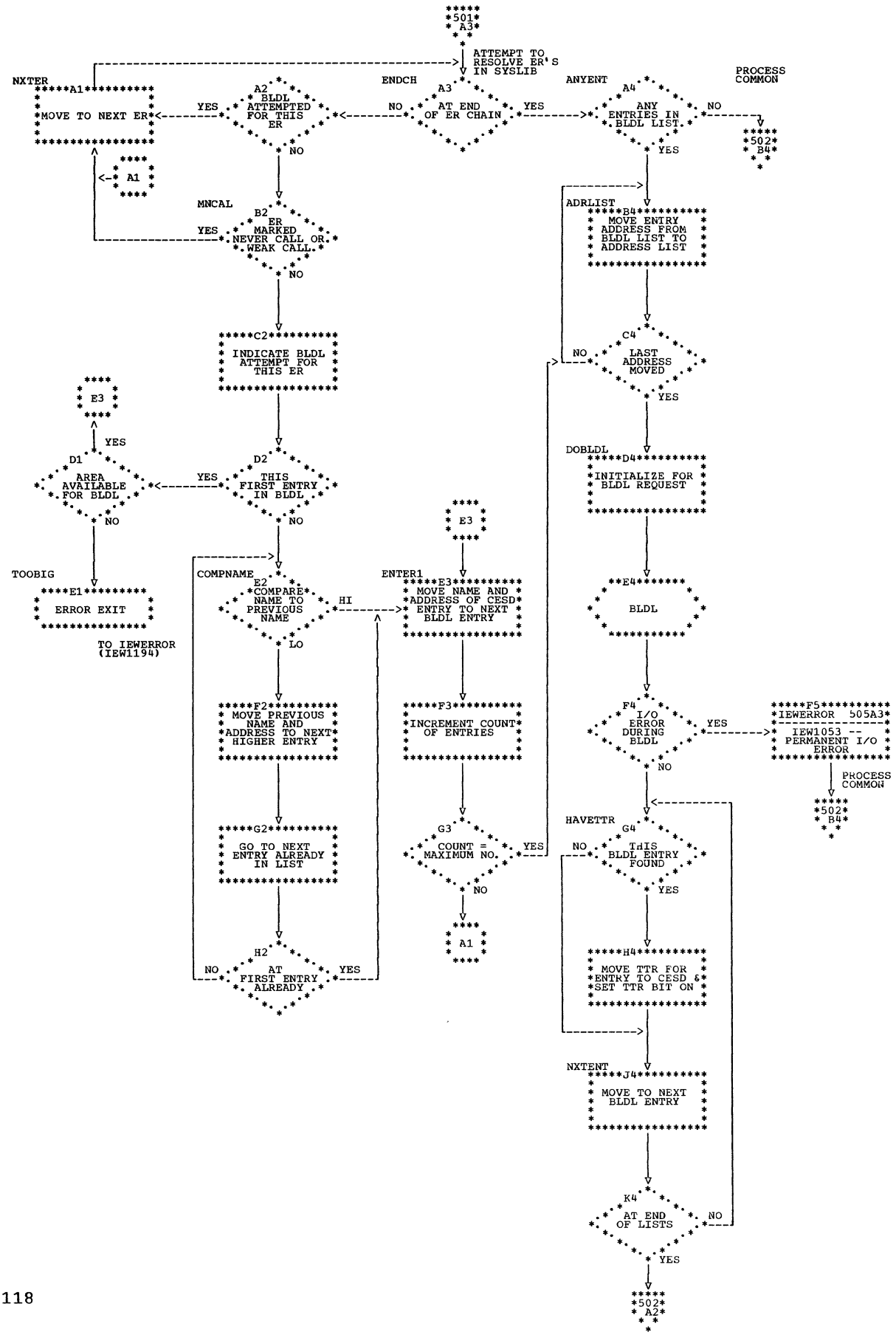


Chart 501. Secondary Input Processing Routine (IEWACALL) (Part 2 of 5)



- Chart 502. Secondary Input Processing Routine (IEWACALL) (Part 3 of 5)

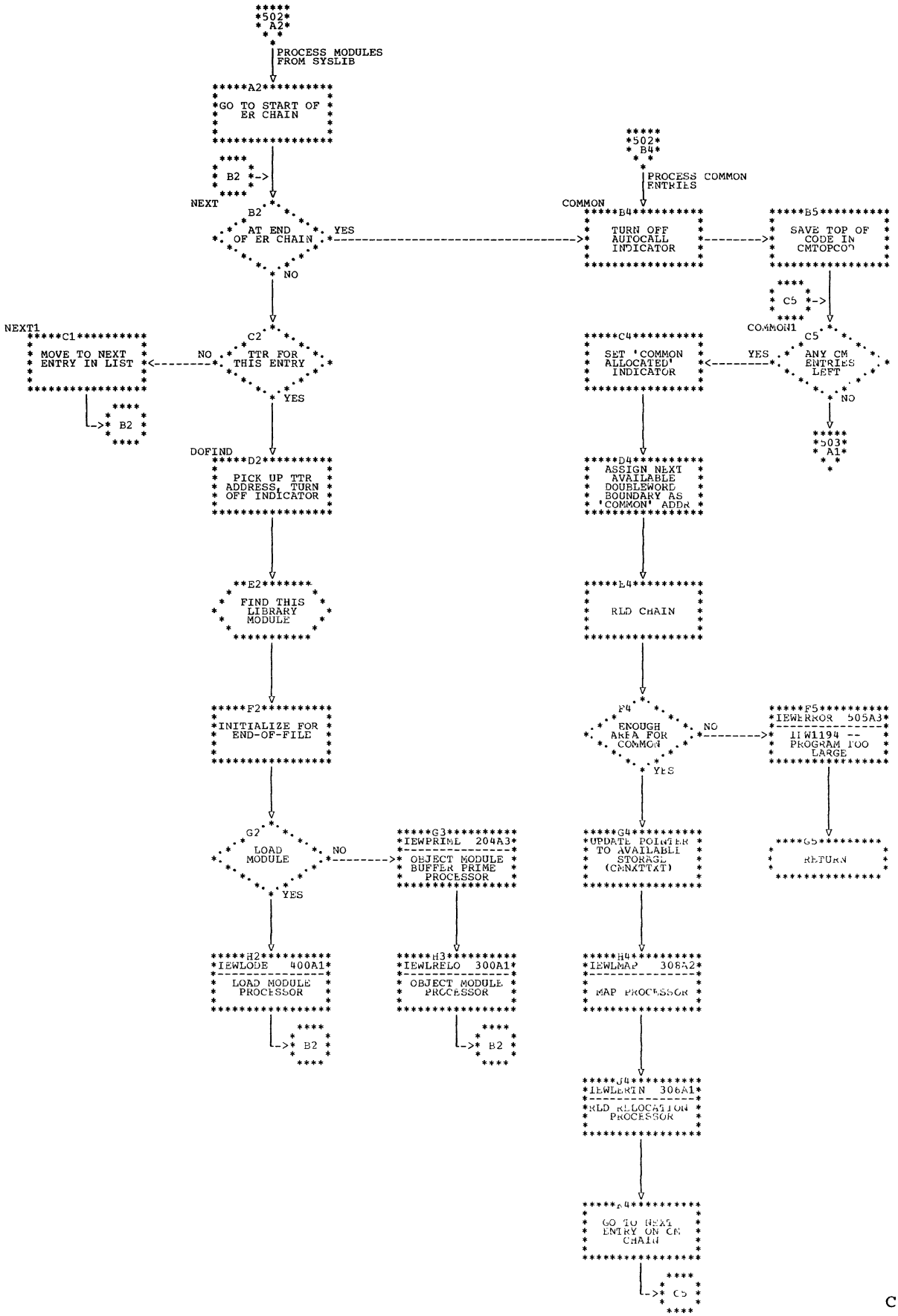
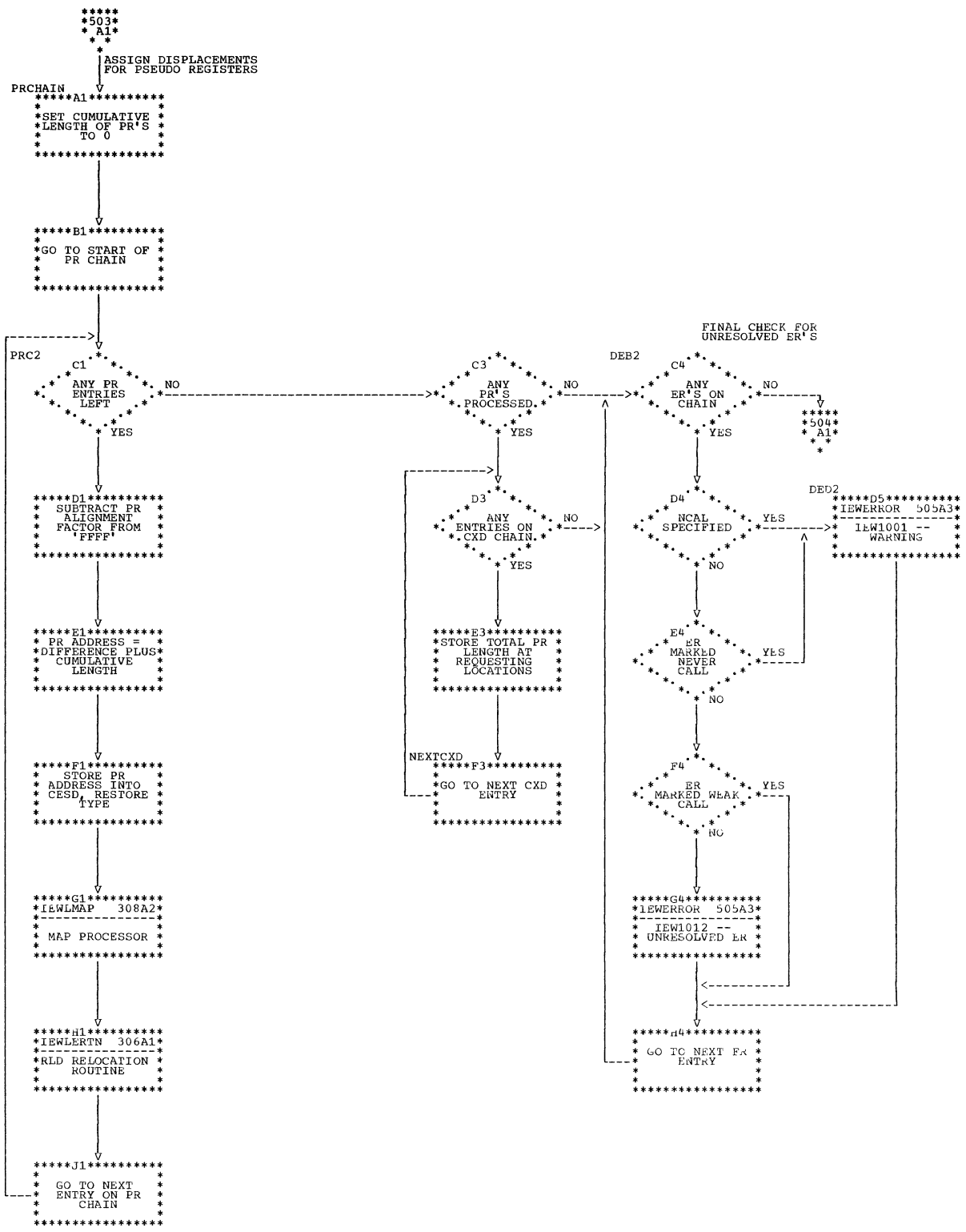
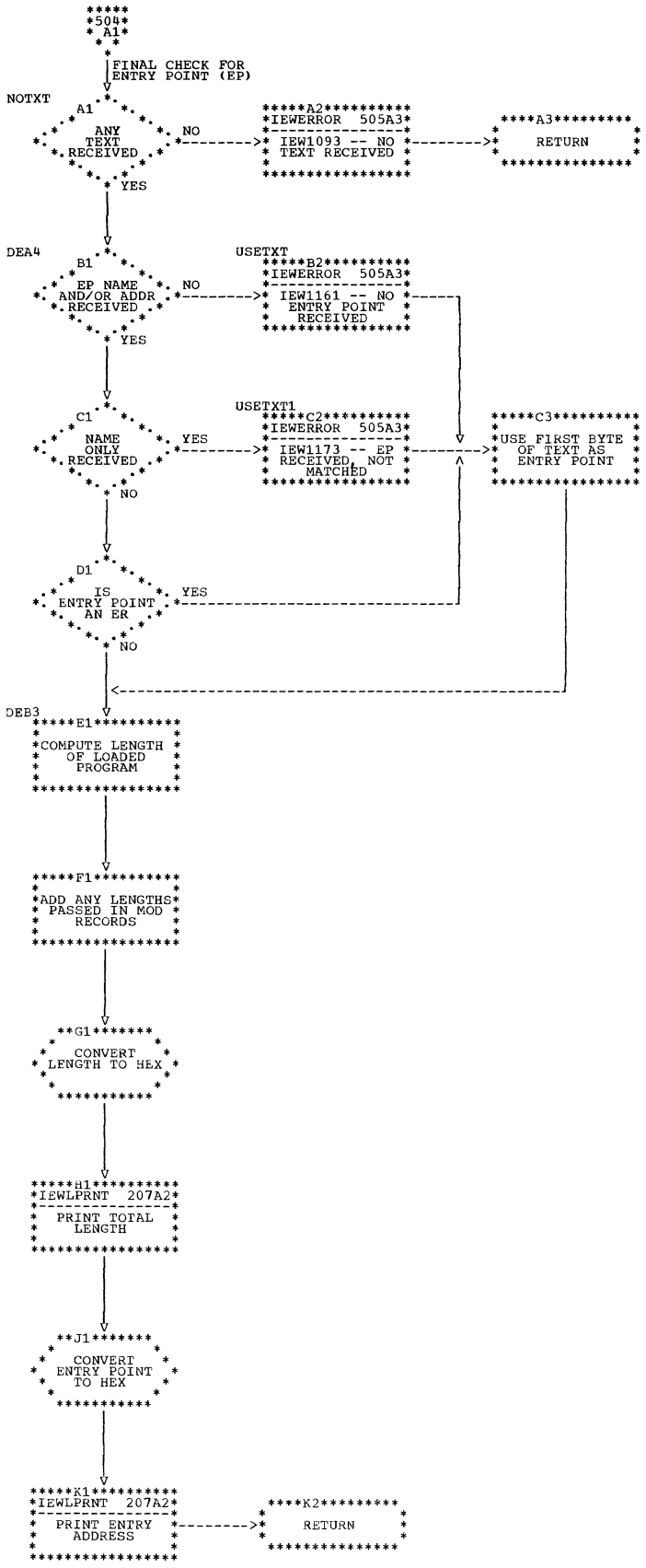


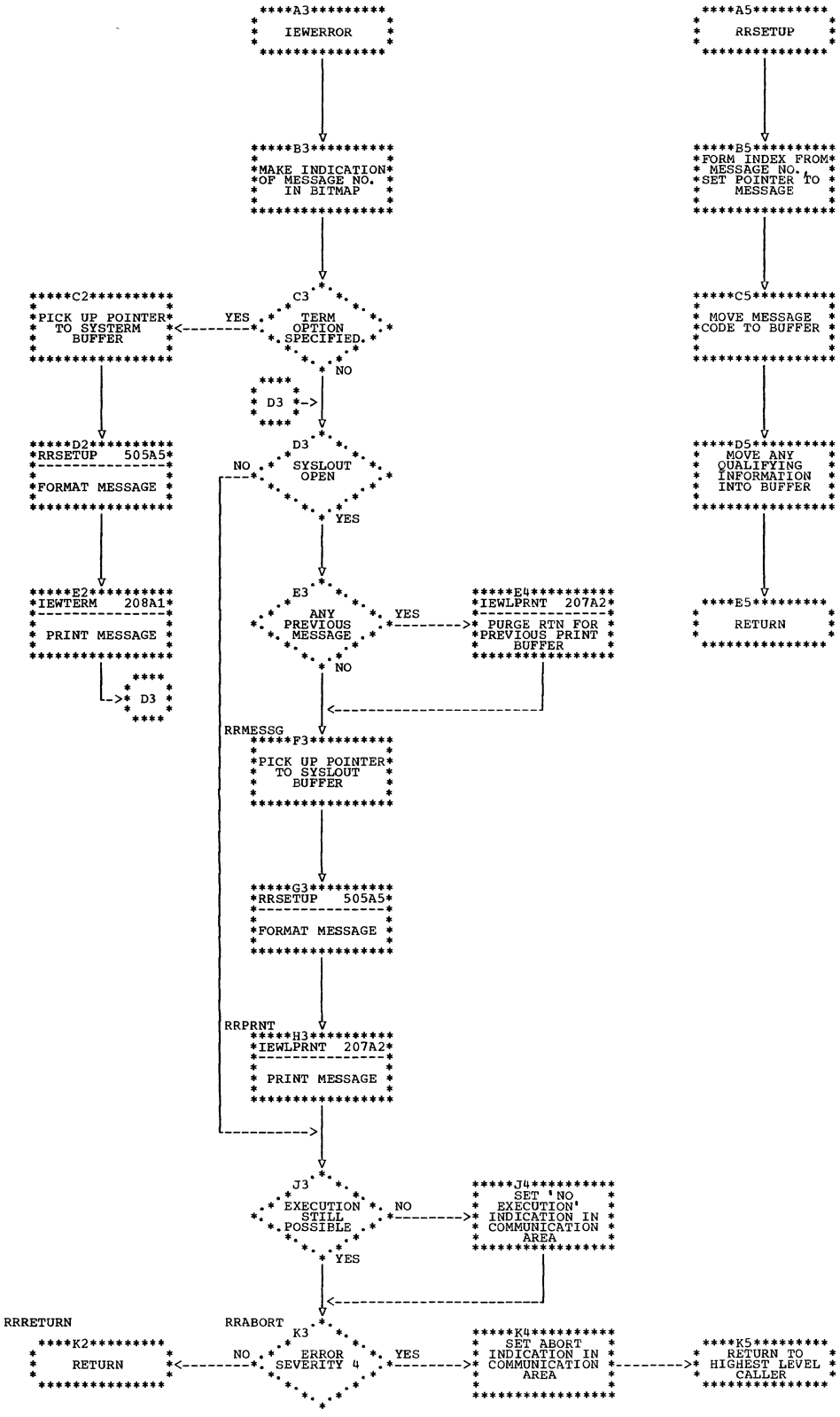
Chart 503. Secondary Input Processing Routine (IEWACALL) (Part 4 of 5)



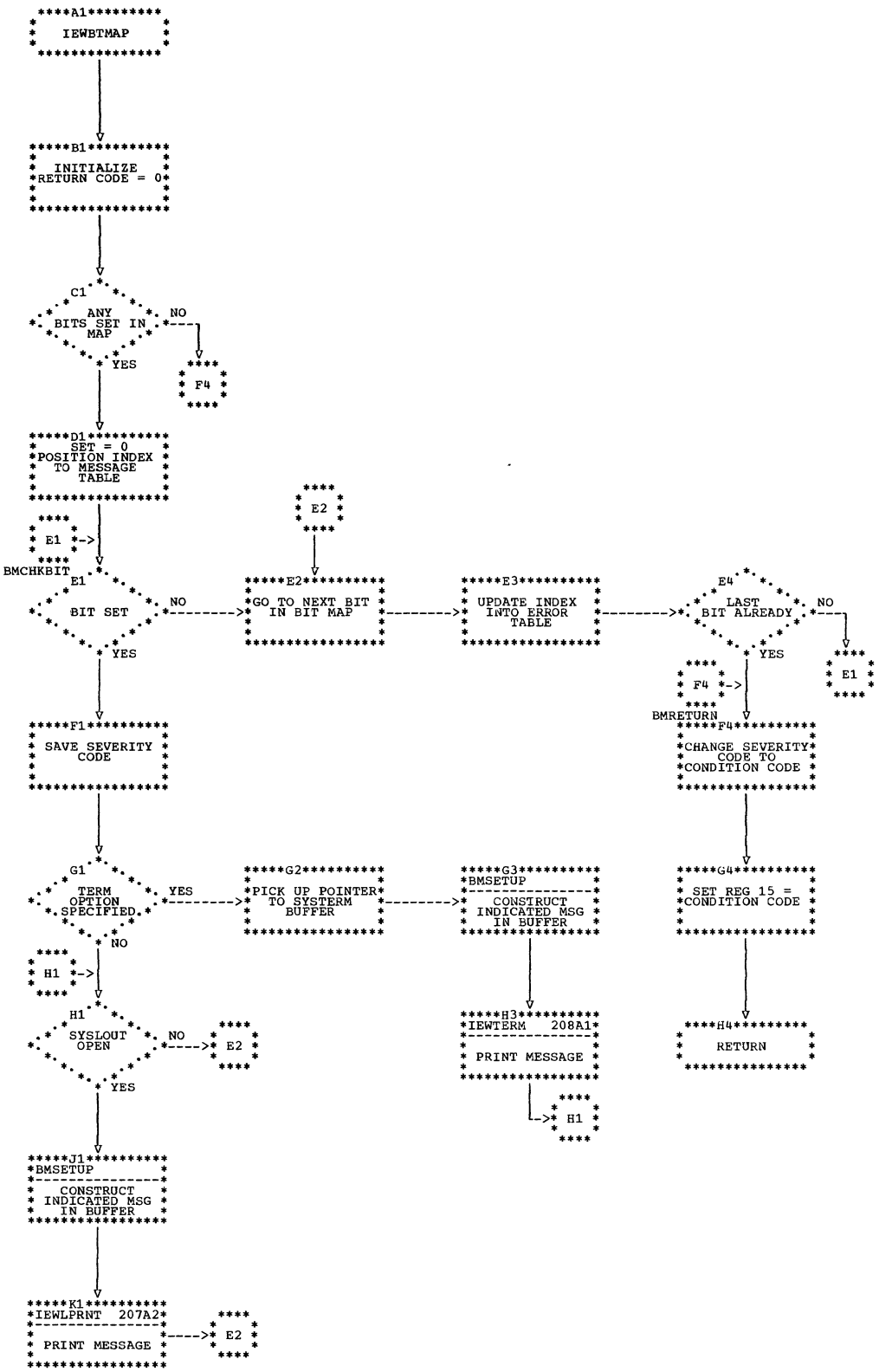
- Chart 504. Secondary Input Processing Routine (IEWACALL) (Part 5 of 5)



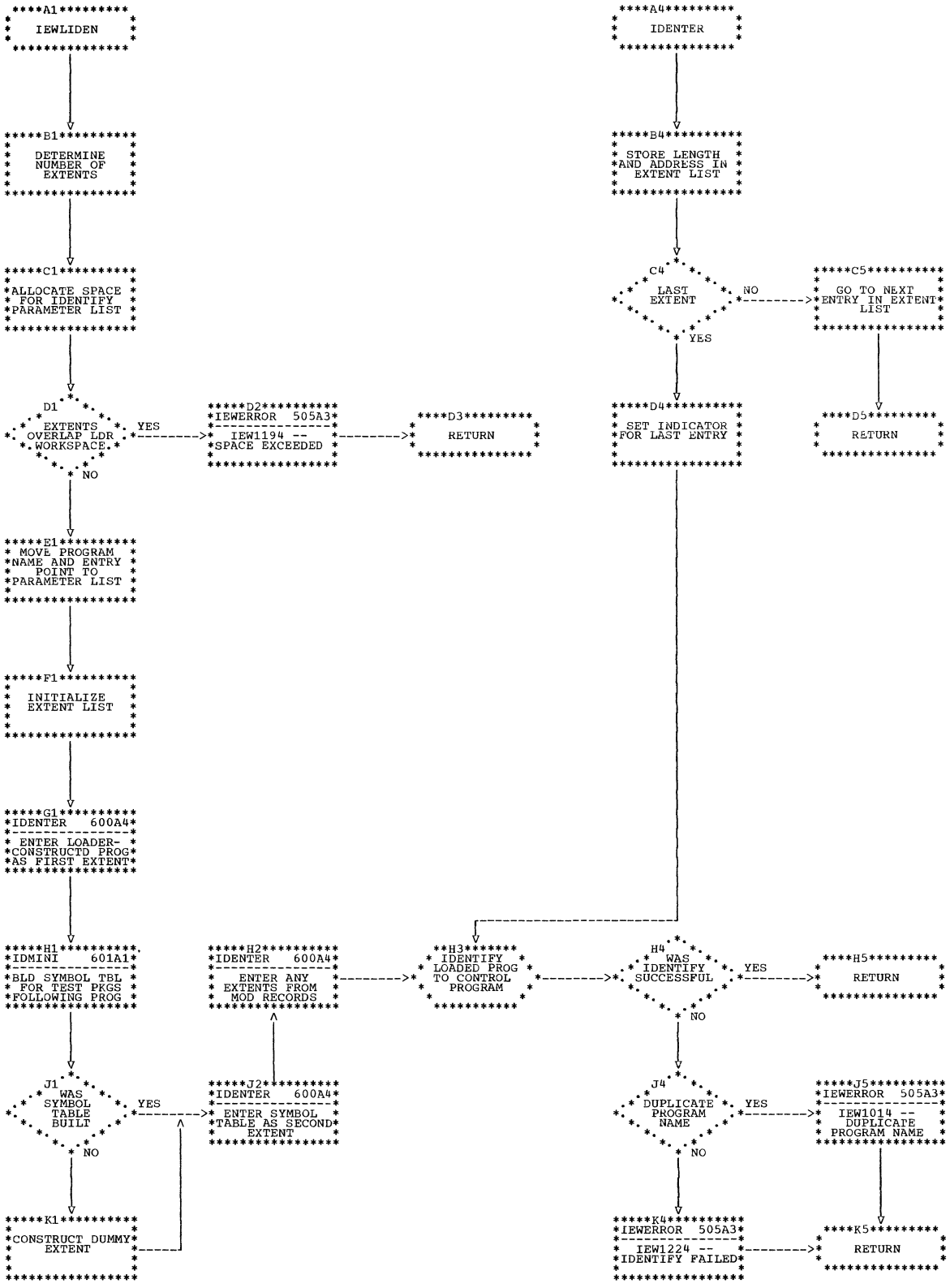
- Chart 505. Error Log Routine (IEWERROR), Format Routine (RRSETUP)



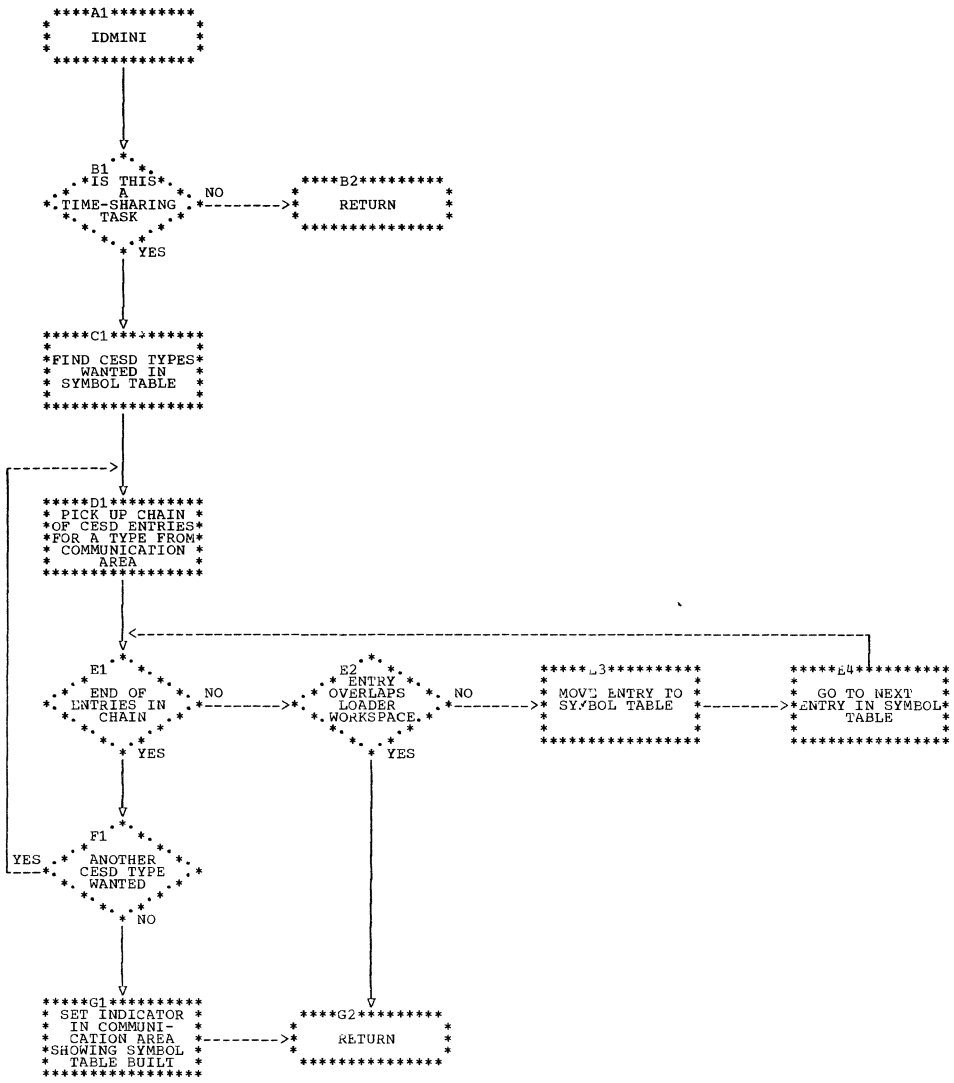
- Chart 506. Diagnostic Dictionary Processing Routine (IEWBTMAP)



- Chart 600. Identification Routine (IEWLIDEN), Extent List Entry Routine (IDENTER)



- Chart 601. Condensed Symbol Table Routine (IDMINI)



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 1 and note the associated object module name. You can then find the item on microfiche.

Name	Description	Object Module	CSECT	Chart ID	Synopsis
ALLOCATE	Allocation Routine	IEWLDREL	IEWLRELO	--	Allocates storage for table entries
CMTRCTRL	Table	IEWLDREL	IEWLRELO	--	Pointers to translation table extents
CMTYPCHN	Table	IEWLDREL	IEWLRELO	--	Pointers to CESD type chains
COMMON	Label	IEWLDLIB	IEWLLIBR	402	Assigns addresses to common
DECB	DSECT	IEWLDIOC	IEWLIOCA	--	Model DECB
ERCODES	DSECT	IEWLDIOC IEWLDREL IEWLDLIB	IEWLIOCA IEWLRELO IEWLLIBR	--	Error code definitions
FINISHUP	Label	IEWLDLIB	IEWLLIBR	--	Prints finishing messages
IEWACALL	Entry Point	IEWLDLIB	IEWLLIBR	500-504	Automatic library call processing
IEWBTMAP	Entry Point	IEWLDLIB	IEWLLIBR	506	Diagnostic dictionary processing
IEWBUFFR	Buffer allocation routine	IEWLDIOC	IEWLIOCA	203	Buffer and DECB allocation routine
IEWERROR	Entry Point	IEWLDLIB	IEWLLIBR	505	Error log routine
IEWLCNVT	Entry Point	IEWLDREL	IEWLRELO	--	Binary-Hex conversion routine
IEWLCTRL	Entry Point CSECT	IEWLDCTR	IEWLCTRL	100	Loader control module
IEWLDCOM	DSECT	IEWLDIOC IEWLDLIB IEWLDREL	IEWLIOCA IEWLLIBR IEWLRELO	--	Communication area
IEWLDDEF	CSECT	IEWLDDEF	IEWLDDEF	--	SYSGEN option defaults
IEWLEND	Entry Point	IEWLDREL	IEWLRELO	307	End processing
IEWLERTN	Entry Point	IEWLDREL	IEWLRELO	306	RLD relocation routine
IEWLESD	Entry Point	IEWLDREL	IEWLRELO	301-304	ESD record processing

(Continued)

(Continued)

Name	Description	Object Module	CSECT	Chart ID	Synopsis
IEWLIDEN	Entry Point CSECT	IEWLDIDY	IEWLIDEN	600	Identification routine
IEWLIOCA	Entry Point CSECT	IEWLDIOC	IEWLIOCA	200-201	Initialization, I/O, control, and allocation processing
IEWLLIBR	CSECT	IEWLDLIB	IEWLLIBR	--	Automatic library call and load module processing
IEWLMAP	Entry Point	IEWLDREL	IEWLRELO	308	Creates map printout
IEWLMOD	Entry Point	IEWLDREL	IEWLRELO	310	MOD record processing
IEWLOAD	Entry Point	IEWLDIOC	IEWLIOCA	200	Entry point for loading with identification
IEWLODE	Entry Point	IEWLDLIB	IEWLLIBR	400-403	Load module processing
IEWLPRNT	Entry Point	IEWLDIOC	IEWLIOCA	207	Print routine
IEWLREAD	Entry Point	IEWLDIOC	IEWLIOCA	205	Read routine
IEWLRELO	Entry Point	IEWLDREL	IEWLRELO	300	Object module processor
IEWLRELO	CSECT	IEWLDREL	IEWLRELO	--	Object module, ESD, RLD, and map processing
IEWLRLD	Entry Point	IEWLDREL	IEWLRELO	305	RLD record processing
IEWLTXT	Label	IEWLDREL	IEWLRELO	300	Object module text processing
IEWOPNLB	Entry Point	IEWLDIOC	IEWLIOCA	206	Opens SYSLIB data set
IEWPRIME	Entry Point	IEWLDIOC	IEWLIOCA	204	Object module buffer prime routine
IEWTERM	Entry Point	IEWLDIOC	IEWLIOCA	208	SYSTERM routine
INITMAIN	DSECT	IEWLDIOC	IEWLIOCA	--	Initial work area
LMTXT	Label	IEWLDLIB	IEWLLIBR	401-403	Load module text processing
MODELDCB	Label	IEWLDIOC	IEWLIOCA	--	Model DCB for SYSLIN, SYSLIB
OPENEXIT	Entry Point	IEWLDIOC	IEWLIOCA	202	DCB exit routine
PSEUDOR	Label	IEWLDLIB	IEWLLIBR	503	Processes pseudo registers
SYNAD	Entry Point	IEWLDIOC	IEWLIOCA	--	SYNAD routine
TRANSID	Entry Point	IEWLDREL	IEWLRELO	309	Translates ESD ID to CESD address

This section provides detailed layouts of internal data areas used during Loader processing. The tables are described in alphabetic order.

Also included in this section is a summary of table use and construction (Table 7).

Table 7. Table Construction and Usage

Table	Built by	Used and/or Modified by
Address list	IEWACALL	*
BLDL list	IEWACALL	*
CESD control table (CMTYPCHN)	IEWLESD	IEWACALL, IEWLESD
CESD table	IEWLESD	IEWACALL, IEWLERTN, IEWLESD, IEWLRLD, IEWLTXT, LMTXT
Condensed symbol table	IEWLIDEN	TSO test facilities
Extent chain	IEWLMOD	IEWLIDEN
IDENTIFY parameter list	IEWLIDEN	IDENTIFY macro instruction
IEWLDCOM	IEWLIOCA	**
INITMAIN	IEWLIOCA	*
RLD table	IEWLRLD	IEWACALL, IEWLERTN, IEWLRLD
Translation table	IEWLESD	IEWACALL, IEWLESD, IEWLRLD, IEWLTXT, LMTXT, TRANSID

*Built and processed entirely within one routine.
**Major communication area throughout Loader processing.

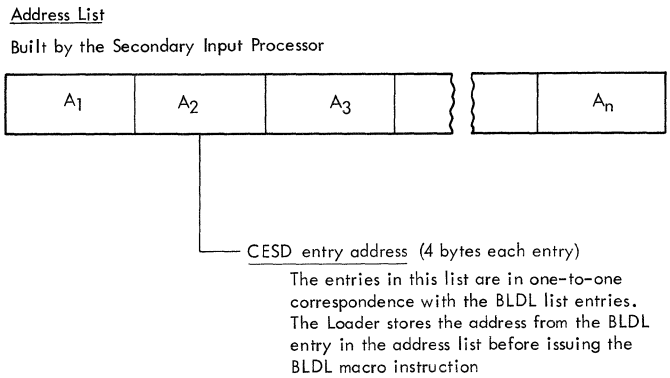


Figure 17. Address List

BLDL List

Built by Secondary Input Processor

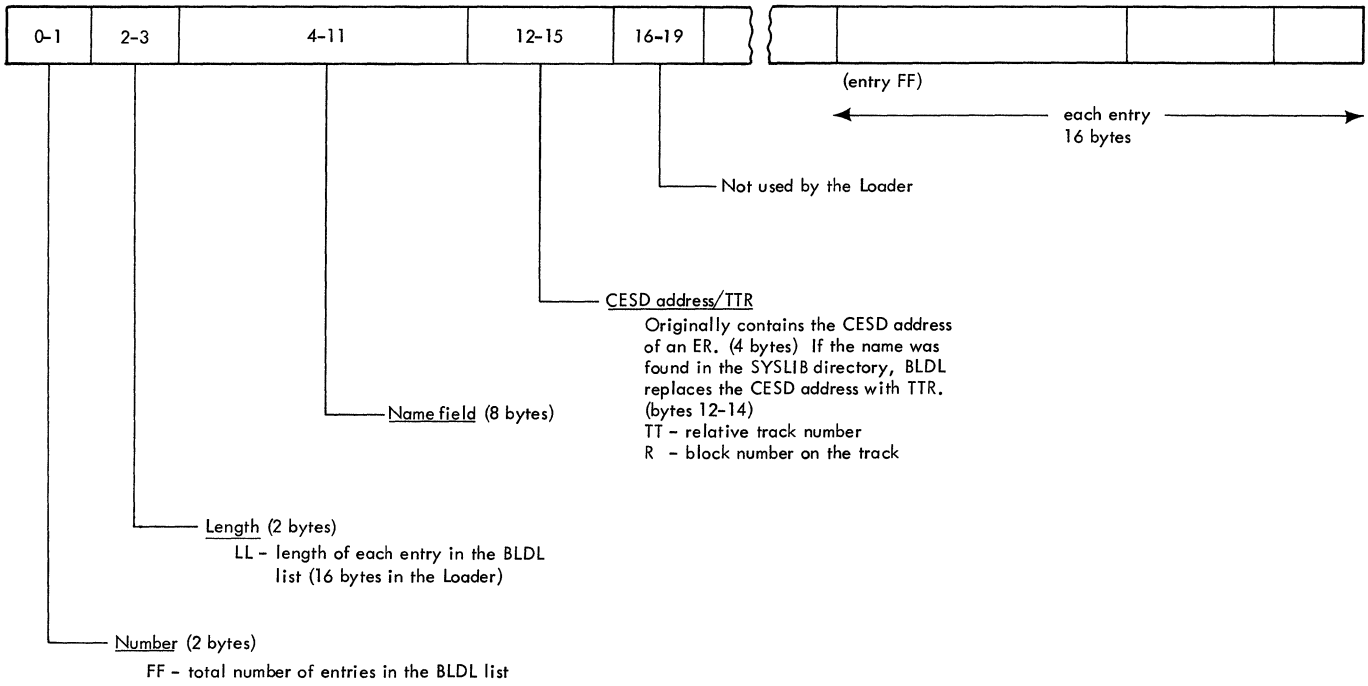
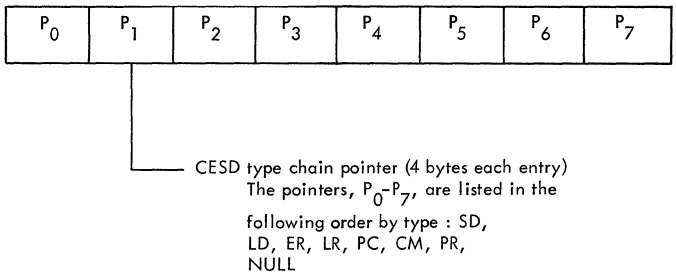


Figure 18. BLDL List

CESD Control Table (CMTYPCHN)

Built by the ESD Processor



Note : The CESD control table is defined in the communications area (IEWLDCOM).

Figure 19. CESD Control Table (CMTYPCHN)

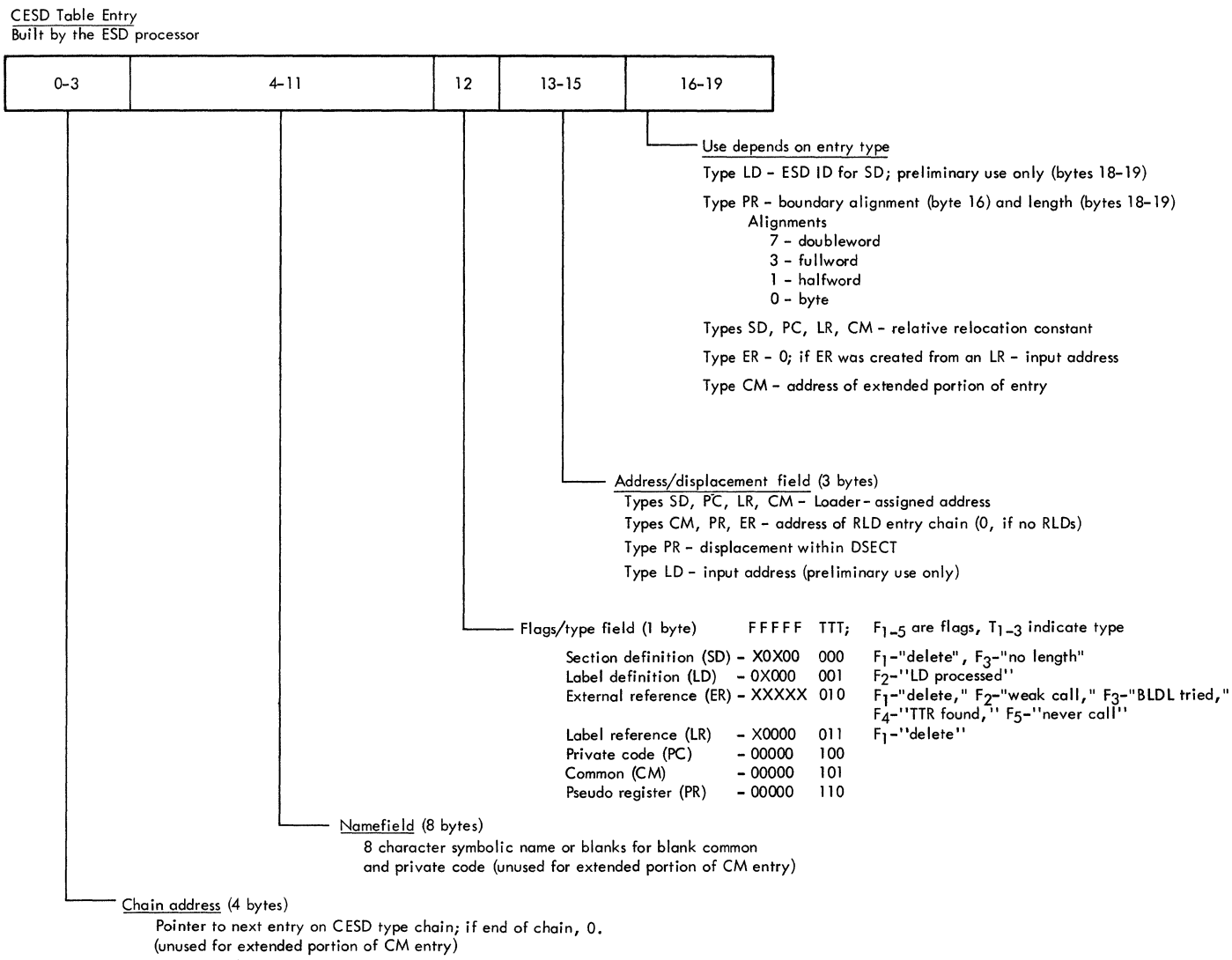
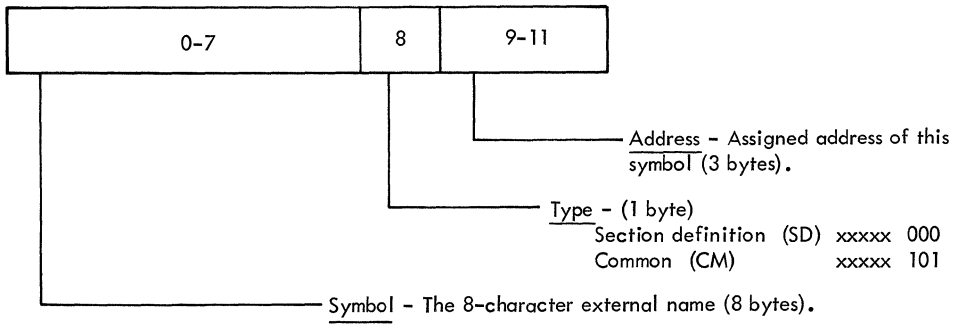


Figure 20. CESD Entry

Condensed Symbol Table Entry

Built by the Identification Processor



- Figure 21. Condensed Symbol Table Entry

Contents Directory Entry
(Only the fields used by the Loader are described.)

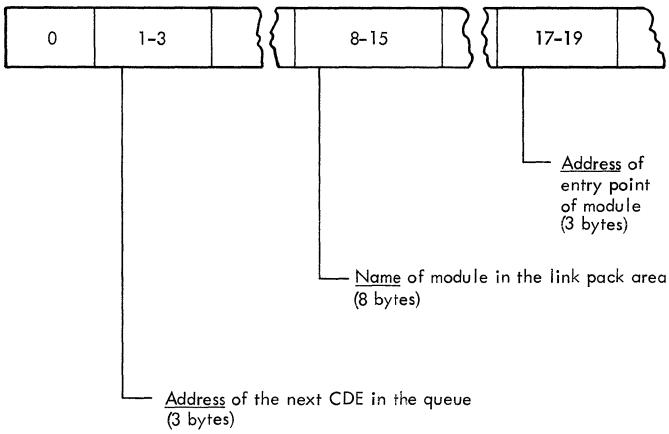


Figure 22. Contents Directory Entry (CDE)

Data Event Control Block
Built by I/O, Control, and Allocation Processor

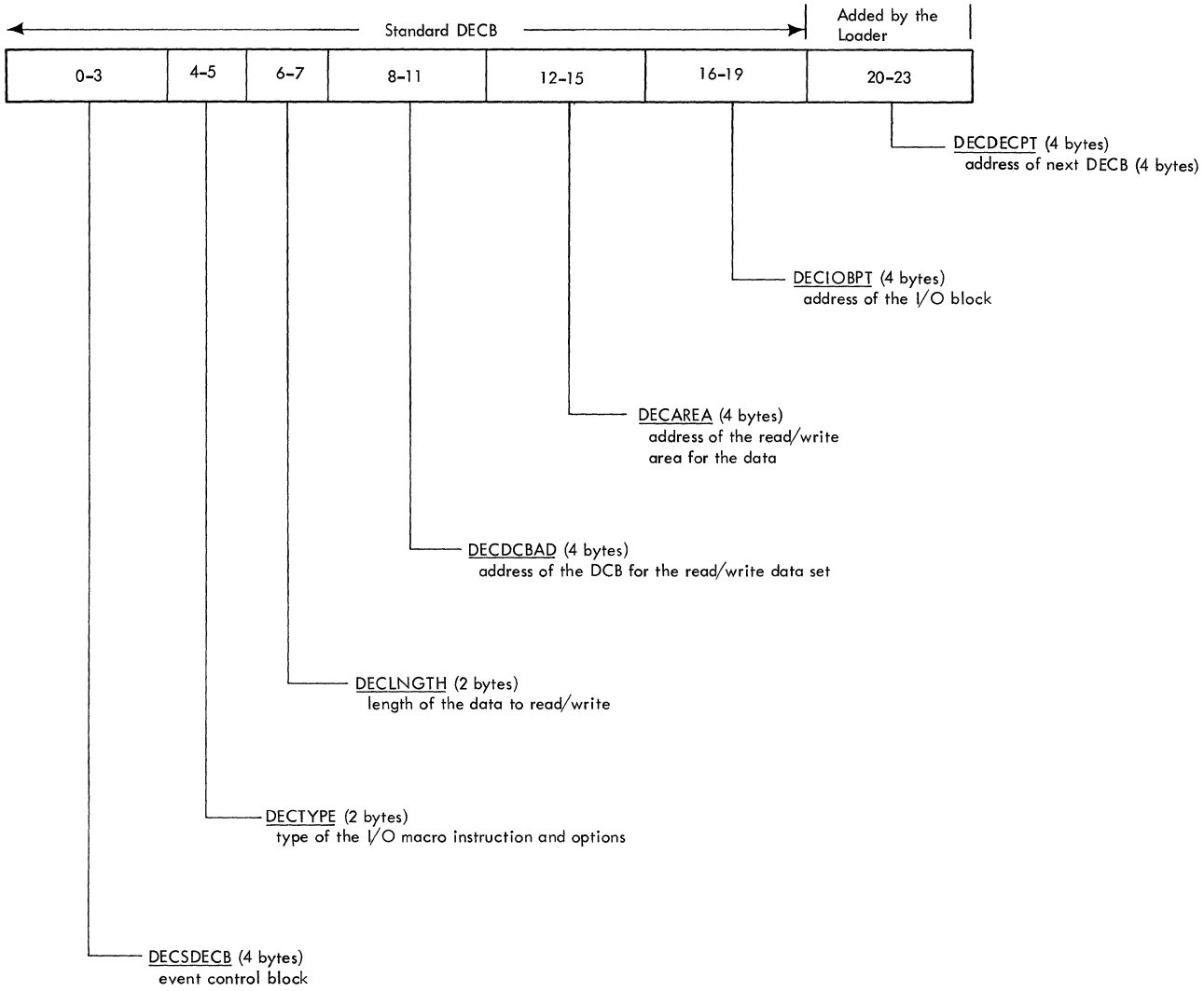
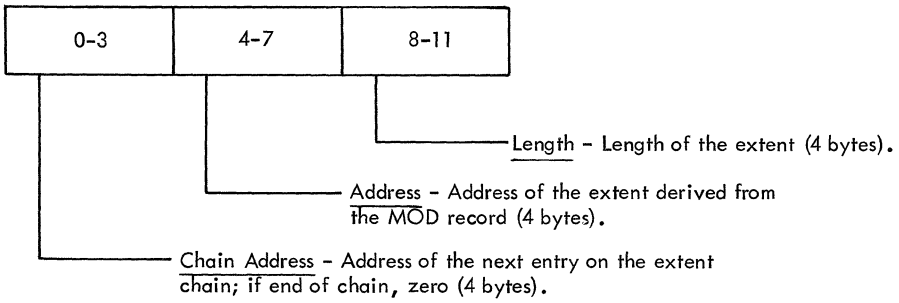


Figure 23. Data Event Control Block (DECB)

Extent Chain Entry

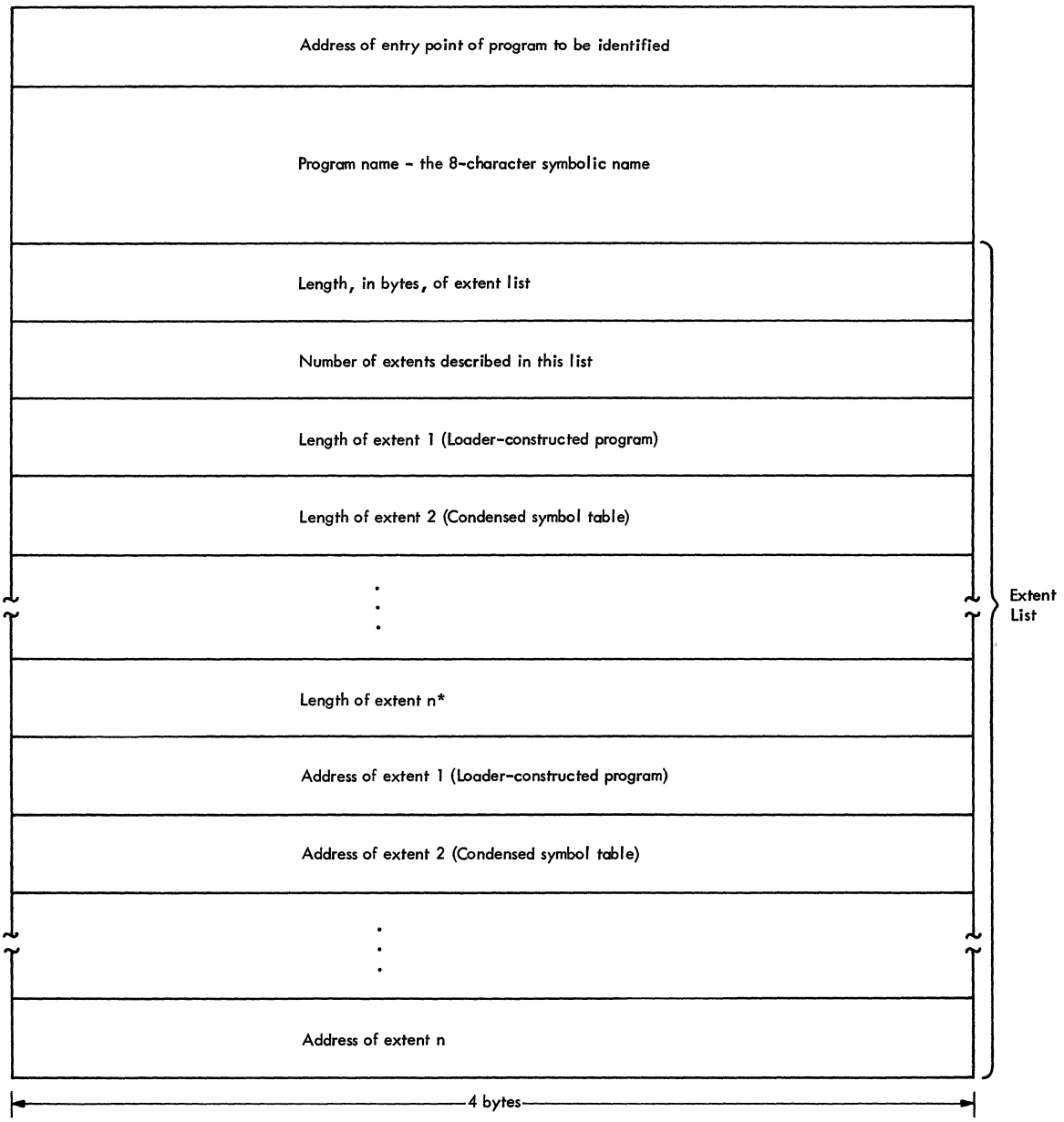
Built by the MOD Processor



• Figure 24. Extent Chain Entry

IDENTIFY Parameter List

Built by the Identification Processor



*A hex '80' in the high-order byte signifies the last length.

Figure 25. IDENTIFY Parameter List

IEWLDCOM DSECT - Communication Area

Hex	Dec				
0	0	CMXDBLWD			
8	8	CMFSTSAV		CMBEGADR	
10	16	CMRDCBPT		CMWDCBPT	
18	24	CMTDCBPT		CMRDECPT	
20	32	CMWDECPT		CMGETREC	
28	40	CMPUTREC		CMTRMREC	
30	48	CMNXTTXT		CMLSTTXT	
38	56	CMLOWTBL		CMHITBL	
40	64	CMIOLST1		CMIOLST2	
48	72	CMCORE1		CMCORE2	
50	80	CMTOPCOD		CMLIBEOD	
58	88	CMLIBSYN		CMLIBEXL	
60	96	CMBLKSIZ	CMMAXLNE	CMMAPLIN	CMWLRECL
68	104	CMMAXLST	unused		
70	112	CMMAINPT		CMMAINSZ	
78	120	CMPRNTDD			
80	128	CMLINDD			
88	136	CMLIBDD			
90	144	CMTERMDD			
98	152	CMEPNAME			
A0	160	CMPGMNM			
A8	168	CMLINDCB		CMLIBDCB	
B0	176	CMPRMFLG	CMIOFLGS	CMFLAG3	CMFLAG4	unused
B8	184	CMXLCHN		CMBITMAP	
C0	192	CMERLIST		CMRLDCHN	
C8	200	CMESDCHN		CMEPADDR	
D0	208	CMTRCTRL			
150	336	CMBLDLPT		CMCXDPT	
158	344	CMFRECOR		CMMODLNG	
160	352	CMTEMPCH		CMEPCESD	
168	360	CMPREVPT		CMLOADCH	
170	368	CMSDCHN		CMLDCHN	
178	376	CMERCHN		CMLRCHN	
180	384	CMPCCHN		CMCMCHN	
188	392	CMPRCHN		CMNULCHN	
190	400	CMCURRID	CMLNECNT	CMBLDLNO	CMWTBFCT
198	408	CMNUMXS	CMLIBFLG	CMRELFLG	CMSTATUS	CMPRTCTL	CMOPTECT	unused

CMINITCM (area common to INITMAIN)

CMTYPCHN (CESD type chain pointers)

- Figure 26. IEWLDCOM DSECT - Communication Area

Explanation of IEWLDCOM Entries

CMXDBLWD (CMADSON)	Temporary doubleword for relocation alignment area
CMFSTSAV	Pointer to first save area
CMBEGADR	Default entry point to module
CMRDCBPT	Input DCB pointer
CMWDCBPT	Output DCB pointer
CMTDCBPT	SYSTERM DCB pointer
CMRDECPT	Input DECB pointer
CMWDECPT	Output DECB pointer
CMGETREC	Input logical record pointer
CMPUTREC	Output logical record pointer
CMTRMREC	SYSTERM buffer pointer
CMNXTTXT	Next address to be assigned to a CSECT
CMLSTTXT	Highest text address assigned to current CSECT
CMLOWTBL	Lowest address assigned for Loader tables
CMHITBL	Highest storage address available to Loader
CMIOLIST1	Open List (DCB pointer)
CMIOLIST2	Open List (DCB pointer)
CMCORE1	Corresponds to CMNXTTXT for preloaded text
CMCORE2	Corresponds to CMLSTTXT for preloaded text
CMTOPCOD	Highest text address before common allocated
CMLIBEOD	EODAD error routine pointer for passed SYSLIB
CMLIBSYN	SYNAD error routine pointer for passed SYSLIB
CMLIBEXL	Exit list pointer for passed SYSLIB
CMBLKSIZ	Blocksize of current input object module
CMMAXLNE	Maximum line-count (SYSLOUT)
CMMAPLIN	Length of map line
CMWLRECL	SYSLOUT record size
CMMAXLST	Maximum length of invalid options list
CMMAINPT	Variable-conditional GETMAIN address
CMMAINSZ	Variable-conditional GETMAIN size
CMPRNTDD	Print ddname
CMLINDD	Primary input ddname
CMLIBDD	Library ddname
CMTERMDD	SYSTERM ddname
CMEPNAME	Entry point name
CMPGMNM	Program name
CMLINDCB	Passed SYSLIN control block pointer
CMLIBDCB	Passed SYSLIB DCB pointer
CMPRMFLG	Parameter flags passed from parameter list Bit 01 - CQRES - RES/MORES Bit 02 - CQMAP - MAP/NOMAP Bit 04 - CQPRINT - PRINT/NOPRINT Bit 08 - CQLET - LET/NOLET Bit 10 - CQCALL - CALL/NOCALL Bit 20 - CQEPNAME - Entry point name defined Bit 40 - CQEPADDR - Entry point address defined Bit 80 - CQTERM - TERM/NOTERM
CMIOFLGS	Input - Output flags Bit 01 - CQEOCB - End of concatenation Bit 02 - CQEOFB - End of file Bit 04 - CQEOFSB - End of file significance Bit 08 - CQRECFM - Input record format 0-fixed 1-undefined Bit 08 - CQUNDEF - CQRECFM separate name in allocation for undefined Bit 10 - CQFIXED - Fixed record format Bit 20 - CQIGNOR - Ignore control record on load module Bit 40 - CQIOERR - An I/O error has occurred
CMFLAG3	Assorted flags Bit 02 - CQTS - Time-sharing environment Bit 04 - CQPGMNM - Program name passed Bit 08 - CQPASLIN - SYSLIN control block passed Bit 10 - CQPASLIB - SYSLIB DCB passed Bit 20 - CQINCORE - Internal SYSLIN data area being processed Bit 40 - CQIDEN - Entered at IEWLOAD (identification wanted)

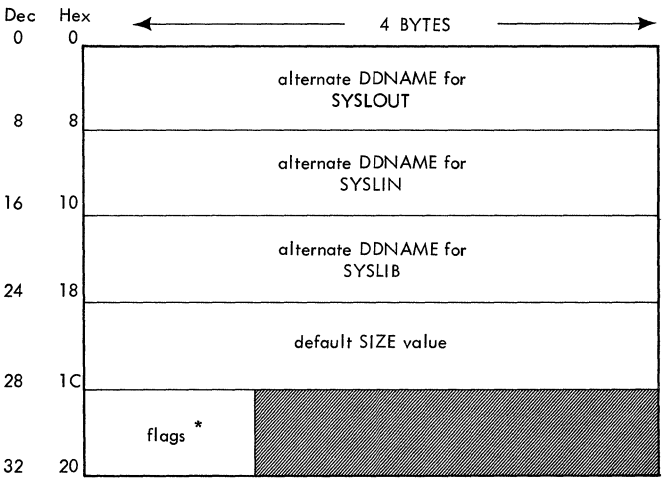
```
CMFLAG4        Assorted flags
               Bit 01 - CQESDS - ESDs have been encountered
               Bit 02 - CQMOD - MOD card has been encountered
               Bit 04 - CQNOEX - Execution not scheduled
               Bit 08 - CQMINI - Condensed Symbol Table built
               Bit 10 - CQMVT - MVT operating
               Bit 20 - CQCOMMON - Common received
               Bit 40 - CQTRMOPN - SYSTERM open
               Bit 80 - CQIDONE - Identification accomplished
(All of the following entries are initialized to zero)
CMXLCHN        Pointer to chain of extents
CMBITMAP       Error bit map
CMERLIST       Pointer to errors encountered during Open
CMRLDCHN       Free RLD entry chain (8 bytes/entry)
CMESDCHN       Free CESD chain entry chain
CMEPADDR       Entry point address to loaded program
CMTRCTRL       Translate Control Table
CMBLDLPT       BLDL pointer
CMCXDPT        Pointer to CXD addresses
CMFRECOR       Free storage chain
CMMODLNG       Length of module currently being processed
CMTEMPCHN      Pointer to load chain entry to be freed
CMEPCESD       CESD line address of the entry point name
CMPREVPT       Previous element in a chain for insert-delete
CMLOADCH       Temporary chain for ESDs in a load module
CMTYPCHN       CESD type chain pointers in order of type number
CMSDCHN        Section definition
CMLDCHN        Label definition
CMERCHN        External reference
CMLRCHN        Label reference
CMPCCHN        Private code
CMCMCHN        Common
CMPRCHN        Pseudo register
CMNULCHN       Null
CMCURRID       ESDID counter
CMLNECNT       Current line-count (SYSLOUT)
CMBLDLNO       Number of BLDL entries
CMWTBFCT       Horizontal byte count in print record
CMNUMXS        Number of extents
CQMAXEXT       32 - Maximum Translation Table extents
CQEXTSIZ       32 - Translation Table extent size (number of entries)
CMLIBFLG       Autocall and load module processor flags
               Bit 01 - CQKEEPS - Keep some text from this record
               Bit 02 - CQDELETE - Delete some text from this record
               Bit 04 - CQAUTOC - Autocall is in progress
               Bit 08 - CQCESDR - CESD has been received for load module
               Bit 10 - CQNOTXT - Text has been received
               Bit 20 - CQLPASRH - LPA resolution possible
               Bit 40 - CQFIRST - First record from load module was CESD
               Bit 80 - CQMFTLPA - MFT Link Pack Area
CMRELFLG       Relocation and object module processor flags
               Bit 01 - CQESD - ESD routine called ID translation routine.
               Bit 02 - CQNOLNG - Length not yet received for current module
               Bit 04 - CQDELINK - Delinking is required
               Bit 08 - CQLIB - Resolution from SYSLIB in process
               Bit 10 - CQNOEND - End card has been received
               Bit 20 - CQINPUT - Input has been received
               Bit 40 - CQENTRY - RLD is for entry point
               Bit 80 - CQNOLNTX - Text received for "no length" CSECT
CMSTATUS       Loader status flag
               Bit 01 - CQPRTOPN - Print DCB allocated
               Bit 02 - CQLIBOPN - Library DCB open
               Bit 04 - CQABORT - Abort loading
               Bit 08 - CQREJOPT - Invalid options are to be printed
               Bit 10 - CQOPNERR - Errors were encountered during Open
               Bit 20 - CQRETURN - Caller to error routine must regain control
               Bit 40 - CQMSGSAV - Request Open-exit to save error messages
               Bit 80 - CQPRTDCB - Print DCB is open
```

CMPRTCTL	Index for printer carriage control
CMOPTECT	Count of invalid options to be printed
CQINTSIZ	Size of zero initialization area
CQCMSIZE	Size of communication area

IEWLDDEF

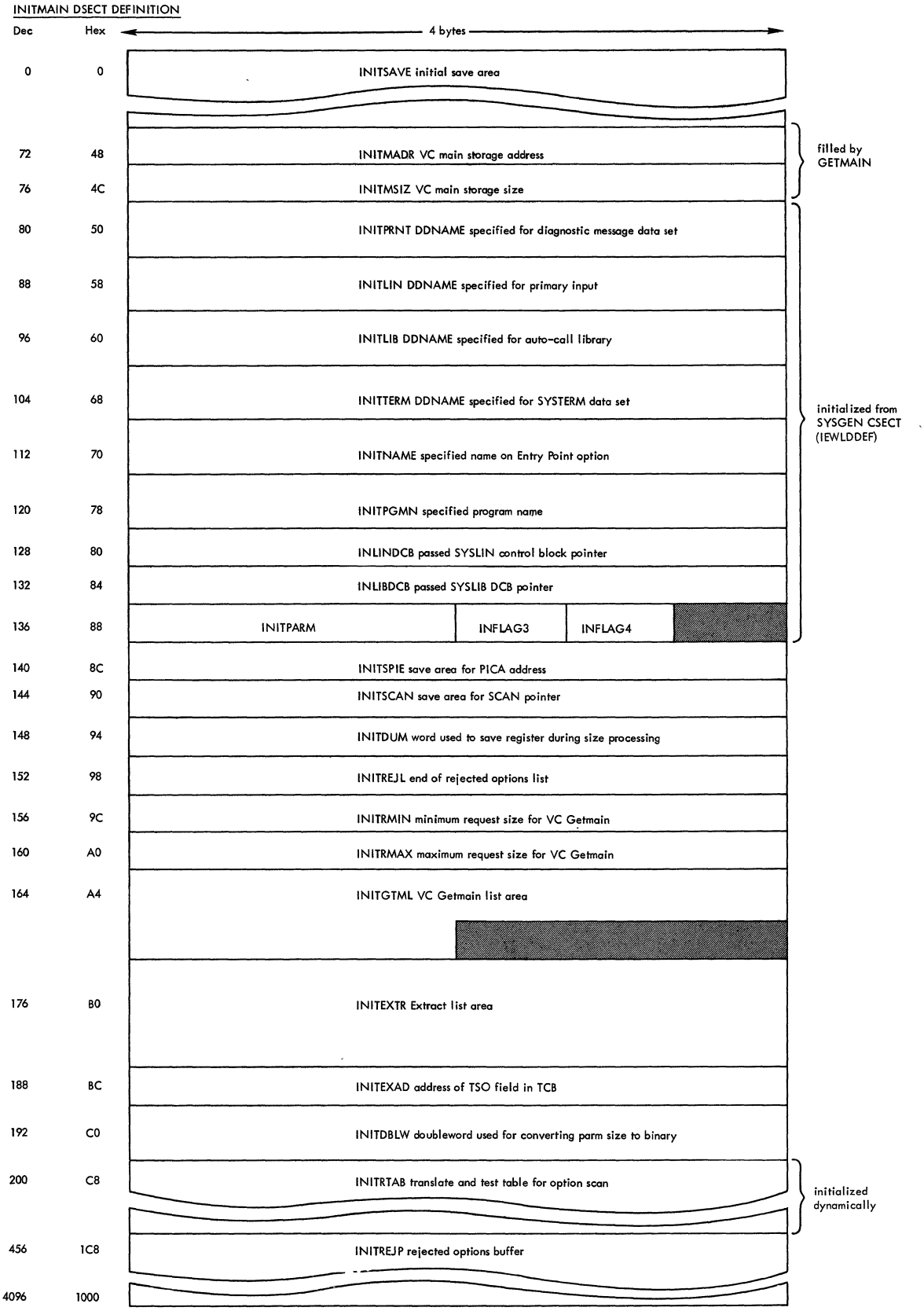
IEWLDDEF is a static CSECT that defines default options and DDNAMES to be used by the Loader. It is assembled at SYSGEN using values supplied by the LOADER macro instruction.

During Loader execution, the default values are moved to dynamic storage (INITMAIN) where they are modified by the parameter list values passed internally. The IEWLDDEF CSECT is described in Figure 27.



* Bits 01-10 correspond to CMPRMFLG flags. See Figure 26.

Figure 27. IEWLDDEF CSECT



• Figure 28. INITMAIN DSECT Definition

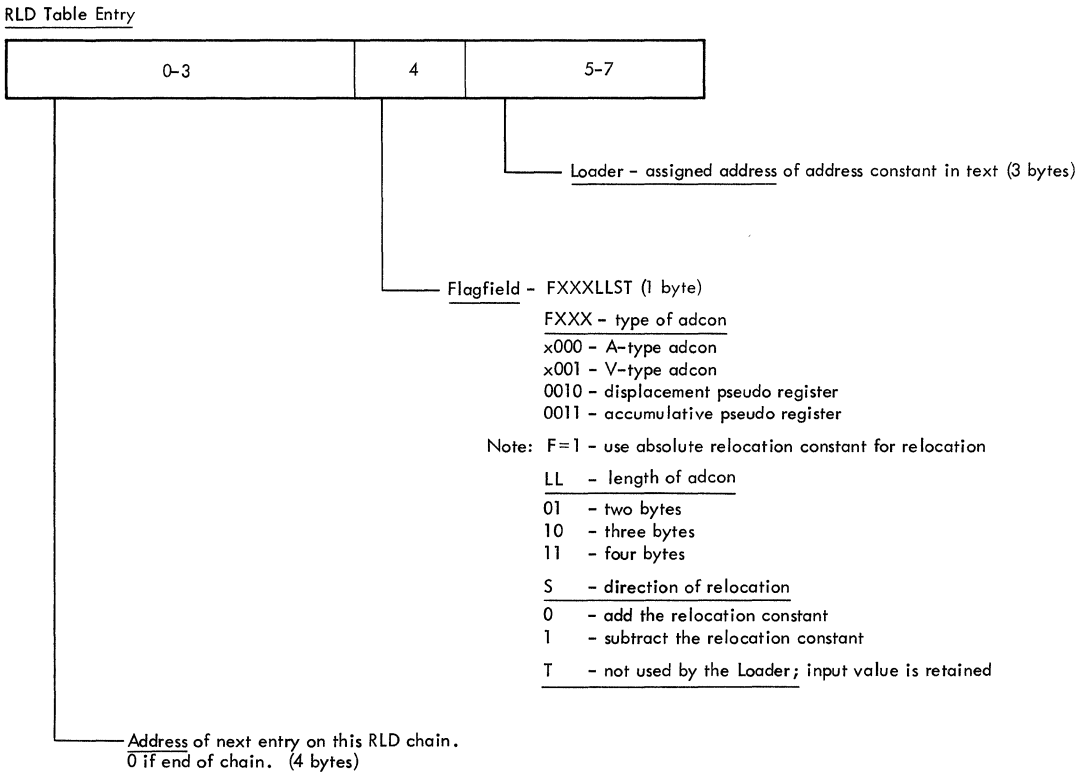
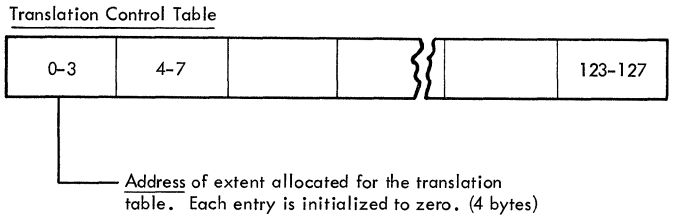
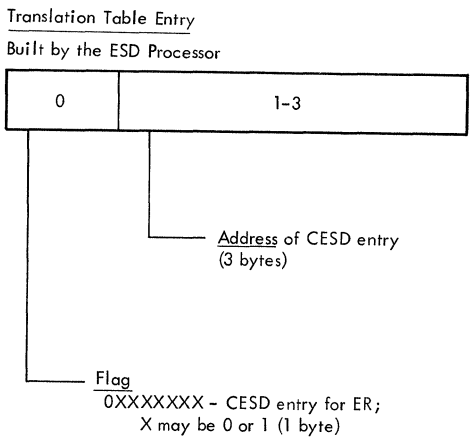


Figure 29. RLD Table Entry



Note: This table is defined in the communications area (IEWLDCOM) at location CMTRCTRL.

Figure 30. Translation Control Table



Note: A translation table extent contains 32 of these entries. The Loader can allocate a maximum of 32 extents. When allocated, an extent is initialized to zero.

Figure 31. Translation Table

This section contains information that may be useful in diagnosing difficulties with the Loader program. Included are: register contents at entry to routines (Table 8), and error code definitions (Table 9), an example of a module map (Table 10), and a list of serviceability aids available with the Loader.

- Table 8. Register Contents at Entry to Routines (Part 1 of 2)

Module Entry Point	Register Contents*
IEWLCTRL	#1 - address of parameter list
IEWLRELO IEWLRELO	#11 - address of communication area
IEWLESD	#5 - ID of first ESD item other than LD #7 - length of ESD information #8 - address of ESD information #11 - address of communication area
IEWLTXT	#5 - Text ID #6 - displacement address of text #7 - length of text #8 - address of text in object module buffer #11 - address of communication area
IEWLMOD	#7 - length of MOD information #8 - address of MOD information #11 - address of communication area
IEWLRLD	#7 - length of RLD information #8 - address of RLD information #11 - address of communication area
IEWLEND	#5 - ID of entry point (if present) #6 - address of entry point (if present) #8 - address of symbolic entry point name (if present) #11 - address of communication area
TRANSID	#5 - ESD ID to be translated #11 - address of communication area
IEWLERTN	#1 - starting address of RLD chain #9 - CESD entry address to be used for relocation #11 - address of communication area
IEWLMAP	#9 - address of CESD entry to be mapped #11 - address of communication area
IEWLCNVT	#1 - binary quantity to be converted #11 - address of communication area
*All entry points expect address of save area in #13 and a return address in #14.	

• Table 8. Register Contents at Entry to Routines (Part 2 of 2)

Module Entry Point	Register Contents*
IEWLLIBR	
IEWLODE	#11 - address of communication area #15 - address of IEWLODE
IEWERROR	#0 - error message code #1 - pointer to qualifying information (if it exists) #11 - address of communication area #15 - address of IEWERROR
IEWACALL	#11 - address of communication area #15 - address of IEWACALL
IEWBTMAP	#11 - address of communication area #15 - address of IEWBTMAP
IEWLIOCA	
IEWLIOCA	#1 - address of parameter list #15 - entry point address
IEWLOAD	#1 - address of parameter list #15 - entry point address
OPENEXIT	#1 - address of DCB #11 - address of communication area #12 - base address of IEWLIOCA
IEWBUFFR	#10 - address of DCB #11 - address of communication area #15 - entry point address
IEWLREAD	For Object and Load Modules #11 - address of communication area #15 - entry point address For Load Modules a. read control/RLD record #0 - zero b. read text records #0 - length of text record #1 - address of text c. read text and control/RLD #0 - compliment of length of text #1 - address of text
IEWOPNLB	#11 - address of communication area #15 - entry point address
IEWLPRNT	#11 - address of communication area #15 - entry point address
IEWTERM	#11 - address of communication area #15 - entry point address
IEWPRIME	#10 - address of DCB #11 - address of communication area #15 - entry point address
IEWLIDEN	
IEWLIDEN	#11 - address of communication area #15 - entry point address
*All entry points expect address of save area in #13 and a return address in #14.	

ERROR CODE DEFINITIONS

Table 9 contains the Loader error codes listed in the order of their bit positions in the error-bit map. (The codes are also listed in DSECT ERCODES in CSECTs IEWLIOCA, IEWLRELO, IEWLLIBR, and IEWLIDEN.)

- Table 9. Internal Error-Code Definitions

Code	Definition	Error Severity
ERREL01	Unresolved ER warning (NCAL specified)	1
ERENTR1	No entry point received	1
ERINPT8	Card received not an object card	1
ERINPT10	END card missing	2
ERINPT2	Length not specified	2
ERREL02	Unresolved ER error	2
ERINPT4	Doubly defined ESD	2
ERINPT5	Invalid 2-byte adcon	2
ERINPT7	Invalid ID received	2
ERINPT9	Invalid object card received	2
ERINPT1	Input block size is invalid	2
ERINPT3	No text received	3
ERENTR2	Entry-point name received but not matched	3
ERIOUT4	Error on BLDL	3
ERINPT6	Invalid record from load module	3
ERIOUT3	Unacceptable record format (variable on input)	4
ERIOUT1	Ddname cannot be opened	4
ERIOUT2	Synchronous error	4
ERSIZE2	Program too large	4
ERSIZE3	Input ESD ID too large	4
ERIDEN1	Identification failed because of duplicate program name	4
ERIDEN2	Identification failed for any other reason	4

- Table 10. Module Map Format Example

Module Map Format

Map heading	Name	Type	Addr	Name	Type	Addr	Name	Type	Addr	Name	Type	Addr	Name	Type	Addr
CSECTs, entry points	Main	SD	9000	ENTRY	LR	9050	ENTRY2	LR	9100	SUB1*	SD	A000	SUB2*	SD	A100
Common entry	$ BLANKCOM	CM	A200												
Pseudo Register information	PSEUDO REGISTERS														
	Name	Origin	Length	Name	Origin	Length	Name	Origin	Length	Name	Origin	Length	Name	Origin	Length
	IHEQINV	00	4	IHEQERR	4	4	IHEQTIC	8	4	IHEQLWF	C	4	IHEQLWO	10	4
	IHEQSLA	14	4												
	TOTAL LENGTH OF PSEUDO REGISTERS 18														
Length of loaded program	TOTAL LENGTH 2000														
Entry of loaded program	ENTRY ADDRESS 9050														

Notes:

- Name * denotes a module included from the SYSLIB data set.
- Name ** denotes a module included from the link pack area.
- Name *** denotes a module pointed to by a MOD record.
- The map entries are made as addresses are assigned, so the map reflects the order of ESD entries in the ESD.

SERVICEABILITY AIDS

Following are serviceability aids provided in the Loader.

- The control section IEWLDDEF contains the SYSGEN default values. It is always resident during loading in IEWLDCOM (load module IEWLOADR).
- A storage dump will typically produce information on the nature of the error. Register 11 will contain a pointer to IEWLDCOM and register 12 will contain the base register associated with the CSECT in control.
- All nine save areas are forward and backward chained. Lower level save areas will be printed. A hex "FF" in word 4 of the save area indicates that the routine represented by the save area has returned control.
- Input/output control information is contained in the Loader communication area. This information consists of the DECB address, the buffer locations, the block size, the logical record length, the blocking factor, the number of records left in the buffer, the address of the current record, and the associated switches. See Section 5 for the IEWLDCOM layout.
- Appropriate diagnostic messages are produced when an error has been detected. The message has a specific number and, where appropriate, lists the data in error. The message number and text are listed by IEWLLIBR at the end of loading. (Section 7 contains a list of these messages.)
- A module map (MAP) is provided to furnish information concerning the structure and contents of the program. This section contains an example of a map listing.
- The Loader uses the SYNADAF to obtain information regarding permanent I/O errors and lists the information on the SYSLOUT data set.

This section contains: a list of error messages and the routines and CSECTs in which they originate, a list of Loader input conventions and restrictions, and detailed descriptions of input record formats. (The input record formats are the same as for the Linkage Editor programs.) In addition, the compiler/Loader interface is described for the processing of data sets passed to the Loader.

Table 11 lists the Loader diagnostic messages. Each message contains a severity code in the final position of the message code. These severity codes are defined as follows:

0 - indicates a condition that will not cause an error during execution of the loaded program.

1 - indicates a condition that may cause an error during execution of the loaded program.

2 - indicates an error that can make execution of the loaded program impossible.

3 - indicates an error that will make execution of the loaded program impossible.

4 - indicates an unrecoverable error. Such an error causes termination of loading.

• Table 11. Error Message/Issuer Cross-Reference Table (Part 1 of 2)

Error Message Number	Error Message Text	Issuer	
		Routine	Cont.Sect.
IEW1001	Warning - Unresolved external reference because of user specification.	IEWACALL	IEWLLIBR
IEW1012	Error - Unresolved external reference	IEWACALL	IEWLLIBR
IEW1024	Error - Ddname cannot be opened	IEWLIOCA	IEWLIOCA
IEW1034	Error - Ddname had synchronous error	SYNAD	IEWLIOCA
IEW1043	Error - Unacceptable record format (variable input)	OPENEXIT	IEWLIOCA
IEW1053	Error - I/O error while searching library directory	IEWACALL	IEWLLIBR
IEW1072	Error - Input BLKSIZE is invalid	OPENEXIT	IEWLIOCA
IEW1082	Error - Length invalid	IEWLEND	IEWLRELO
IEW1093	Error - No text received	IEWACALL	IEWLLIBR
IEW1102	Error - Doubly defined ESDs have conflicting types	IEWLESD	IEWLRELO
IEW1112	Error - Invalid 2-byte adcon	IEWLRLD	IEWLRELO
IEW1123	Error - Invalid record from load module	IEWLODE	IEWLLIBR

• Table 11. Error Message/Issuer Cross-Reference Table (Part 2 of 2)

Error Message Number	Error Message Text	Issuer	
		Routine	Cont.Sect.
IEW1132	Error - Invalid ID received	IEWLRLD IEWLTXT IEWLEND TRANSID IEWLODE	IEWLRELO IEWLRELO IEWLRELO IEWLRELO IEWLLIBR
IEW1141	Warning - Card received not an object record	IEWLRELO	IEWLRELO
IEW1152	Error - Invalid record from object module	IEWLRELO	IEWLRELO
IEW1161	Warning - No entry point received	IEWACALL	IEWLLIBR
IEW1173	Error - Entry point name received but not matched	IEWACALL	IEWLLIBR
IEW1182	Error - No END card received	IEWLRELO	IEWLRELO
IEW1194	Error - Available core exceeded	IEWBUFFR IEWLESD IEWLEND IEWLTXT IEWACALL IEWLODE IEWLIDEN	IEWLIOCA IEWLRELO IEWLRELO IEWLRELO IEWLLIBR IEWLLIBR IEWLIDEN
IEW1204	Error - Too many external names in input module	TRANSID	IEWLRELO
IEW1214	Error - Identification failed - duplicate program name found	IEWLIDEN	IEWLIDEN
IEW1224	Error - Identification failed	IEWLIDEN	IEWLIDEN

INPUT CONVENTIONS

Input modules (object or load) to be processed by the Loader must conform with a number of input conventions.

- 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, EOM record in load modules.)
- Any RLD item must be read after the ESD items to which it refers and after the TXT item in which it is positioned.
- (Applicable only to FORTRAN IV language processing.) Once a BLOCK DATA subprogram has been received, any following named common referencing it must not specify a longer length.
- Since each control section is assigned an address as it is encountered in the input stream, any control section appearing between the ESD for a 'no-length' CSECT and the END card for that 'no-length' CSECT will have an erroneous address assigned. (A 'no-length' CSECT is a control section whose length is defined on the END card.)
- Each record of text and each LD or LR type ESD record must refer to an SD or PC entry in the ESD.
- The position pointers of every RLD record must point to an SD or PC entry in the ESD.
- No LD or LR may have the same name as an SD or CM.
- The Loader accepts TXT records that are out of order within a control section. TXT records are accepted even though they may overwrite previous text in the same control section. The Loader does not eliminate any RLD records that correspond to overwritten text.
- During a single execution of the Loader, if two or more control sections having the same name are read in, the first control section is accepted; the subsequent control sections are deleted.
- The Loader interprets common (CM) ESD items (blank or with the same name) as reference 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.
- (Applicable only to Assembler Language Programming.) When control sections that were or are part of a separately assembled module are to be replaced, A-type address constants that refer to a deleted symbol will be incorrectly resolved unless the entry name is in the same position relative to the origin of the replaced control section. If all control sections of a separately assembled module are replaced, no restrictions apply.
- The MOD record must physically precede all ESD records for an internal object module and logically replace all text records. If a MOD record appears as the first record of an internal object module, all succeeding text records are ignored until an END statement has been processed. A MOD record is ignored if it appears outside an internal object module, if it appears after other records have been encountered for a module, or if its byte count is zero.

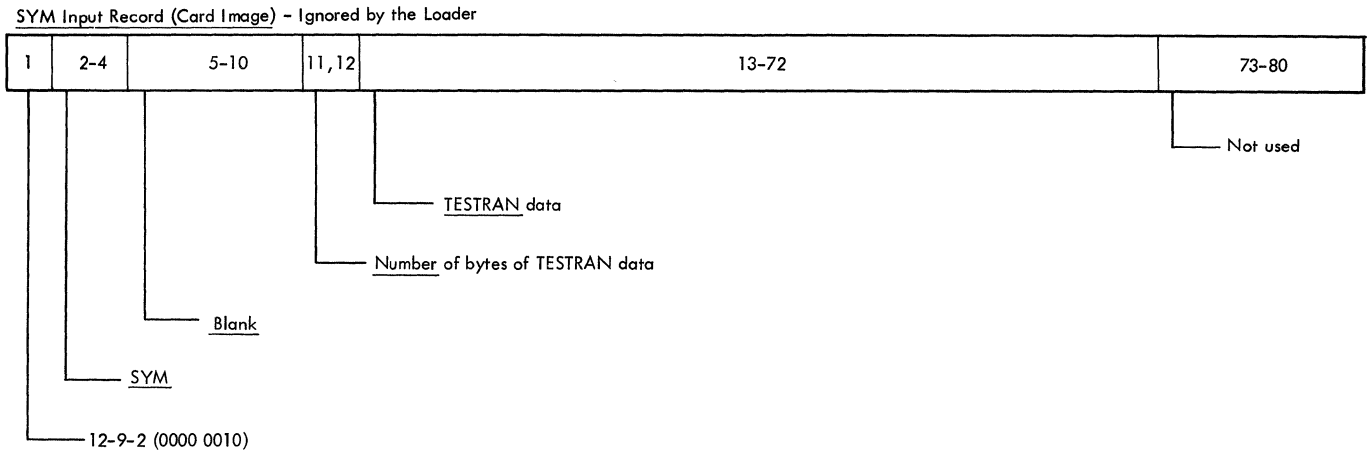


Figure 32. SYM Input Record (Card Image) - Ignored by the Loader

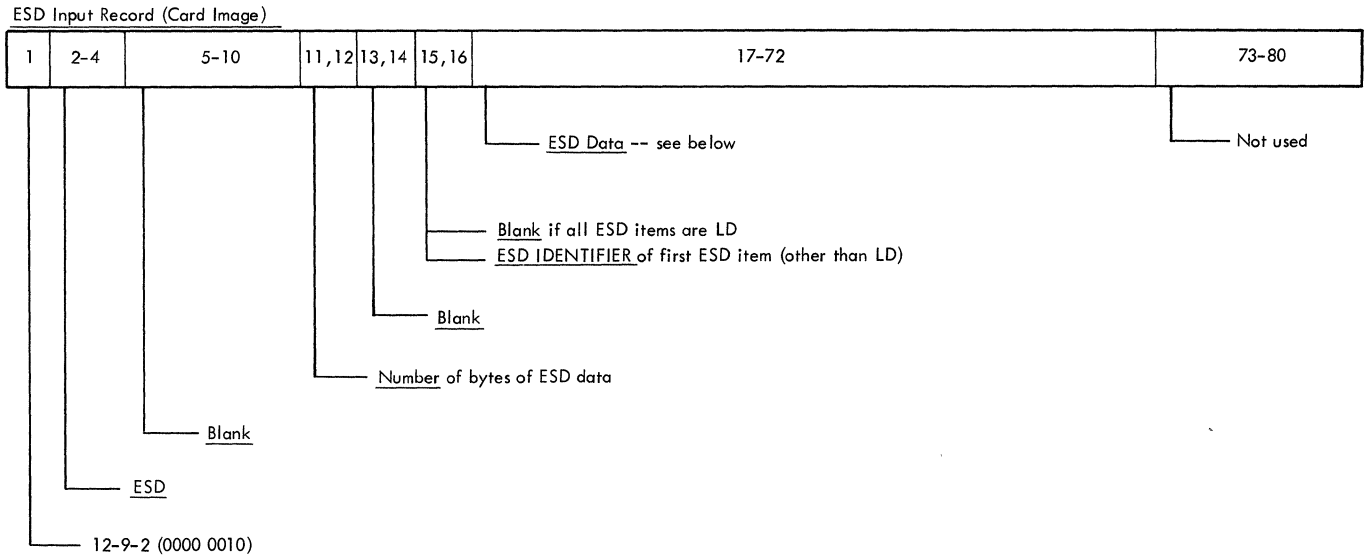


Figure 33. ESD Input Record (Card Image)

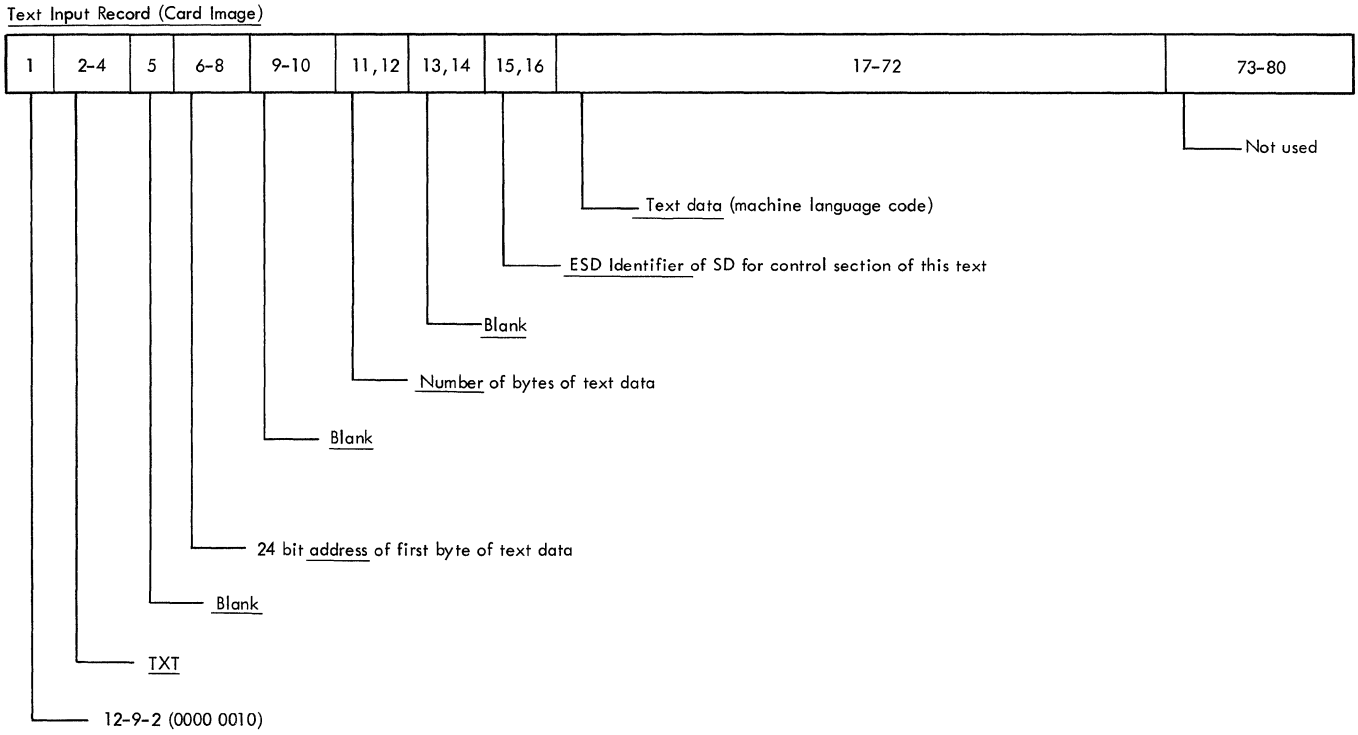


Figure 34. Text Input Record (Card Image)

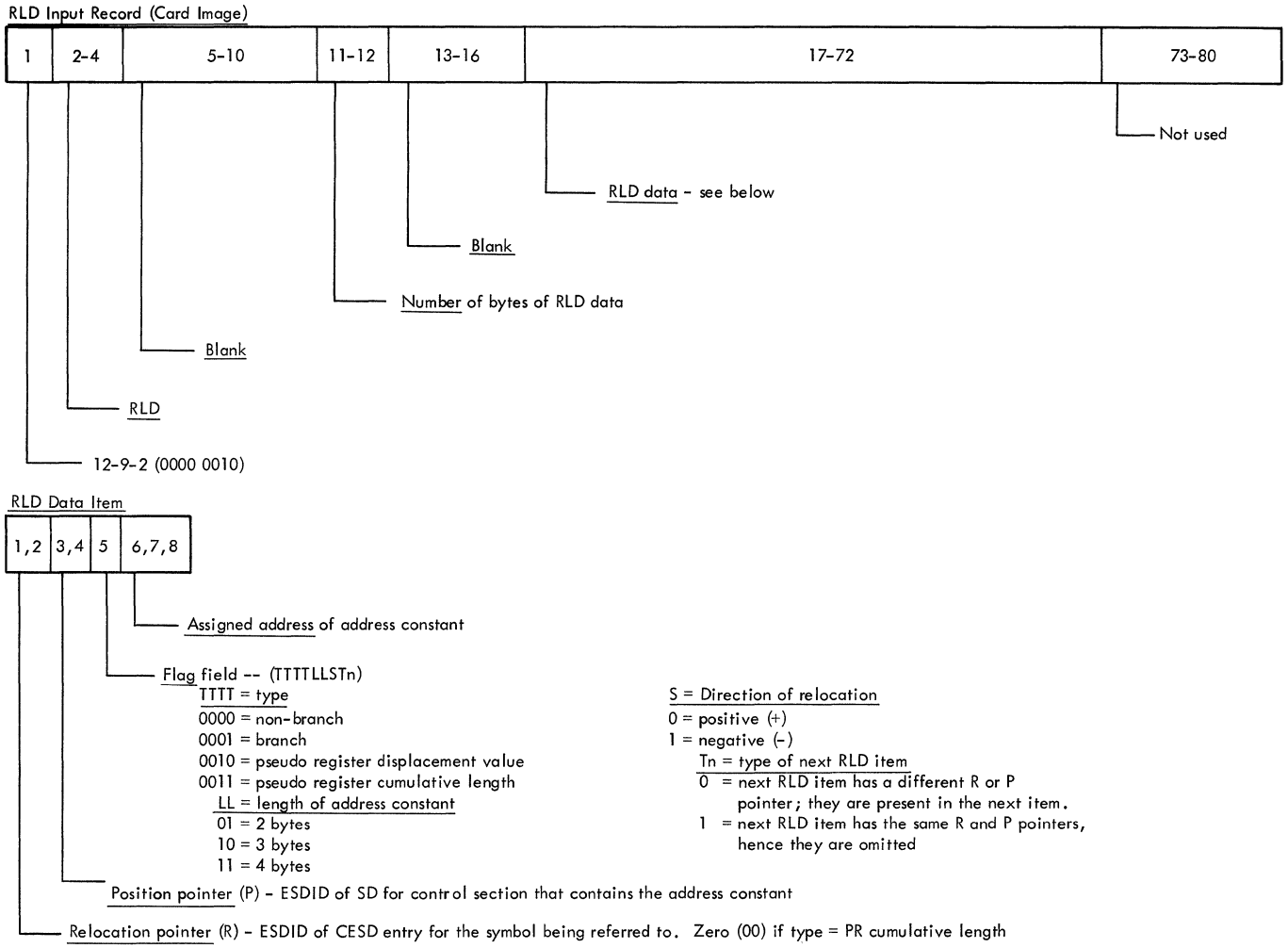


Figure 35. RLD Input Record (Card Image)

END Input Record - Type 1 (Card Image)

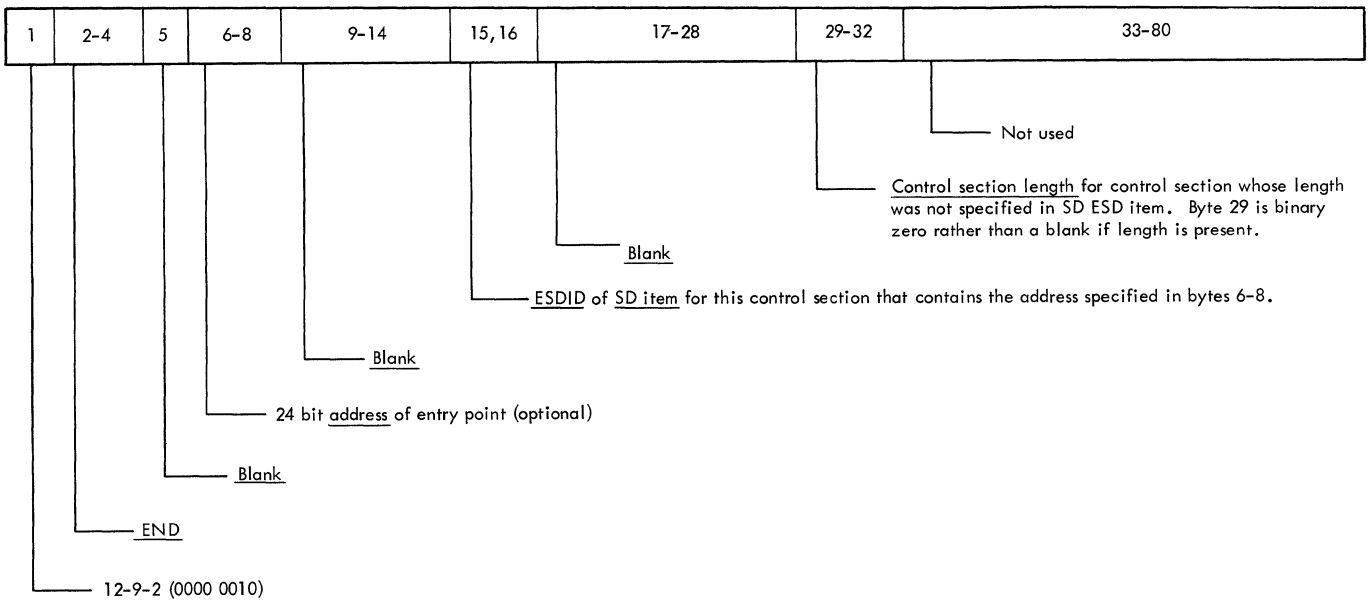


Figure 36. END Input Record - Type 1 (Card Image)

END Input Record - Type 2 (Card Image)

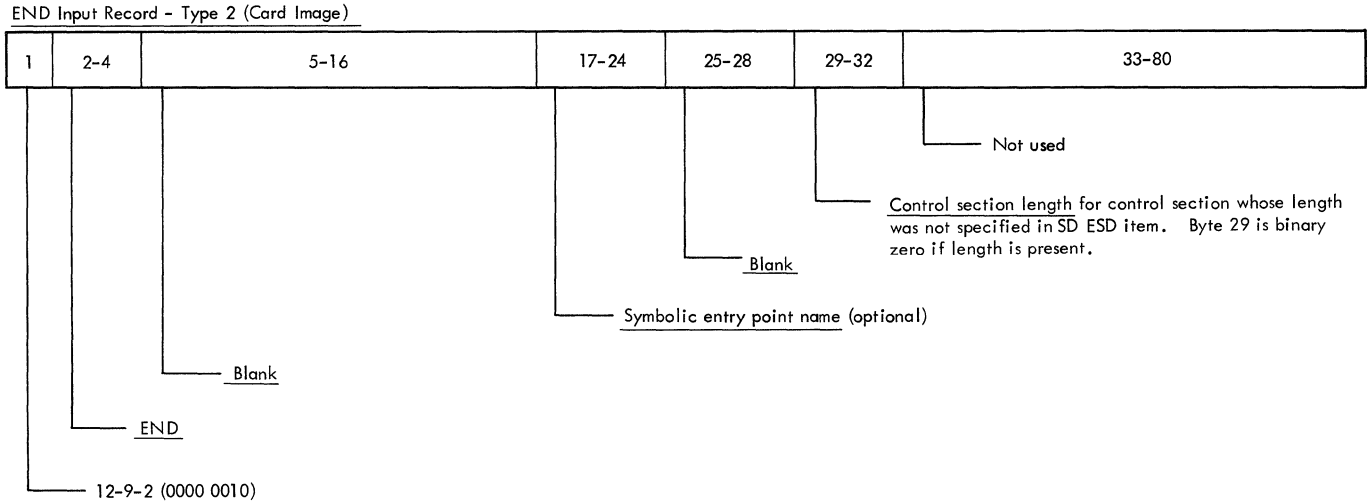


Figure 37. END Input Record - Type 2 (Card Image)

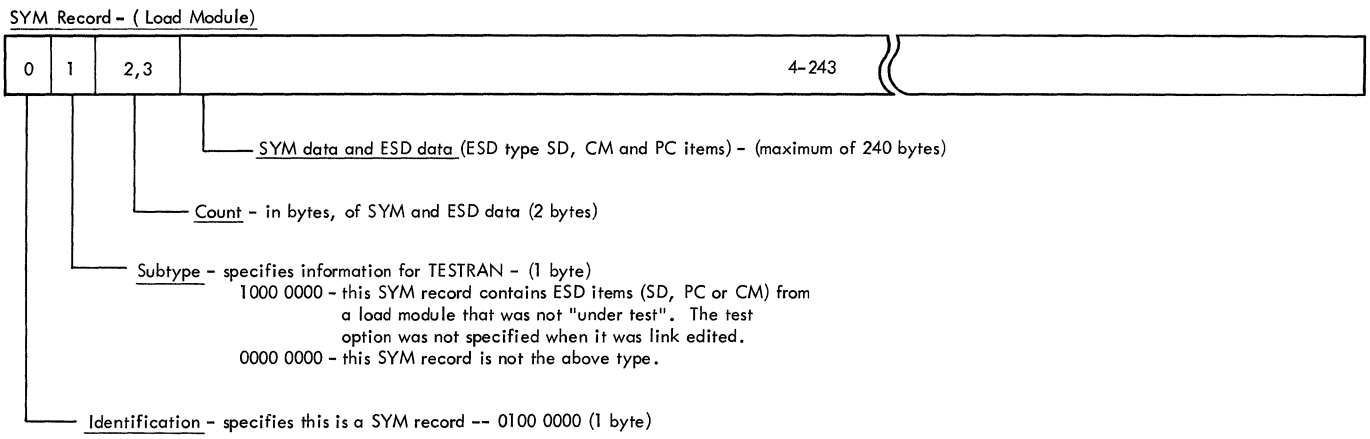
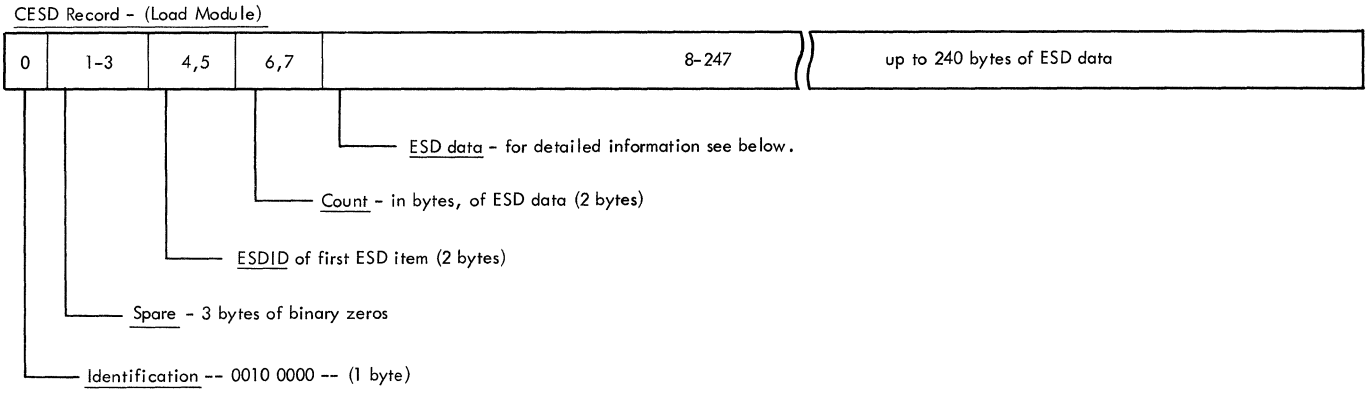


Figure 38. SYM Record (Load Module) - Ignored by the Loader



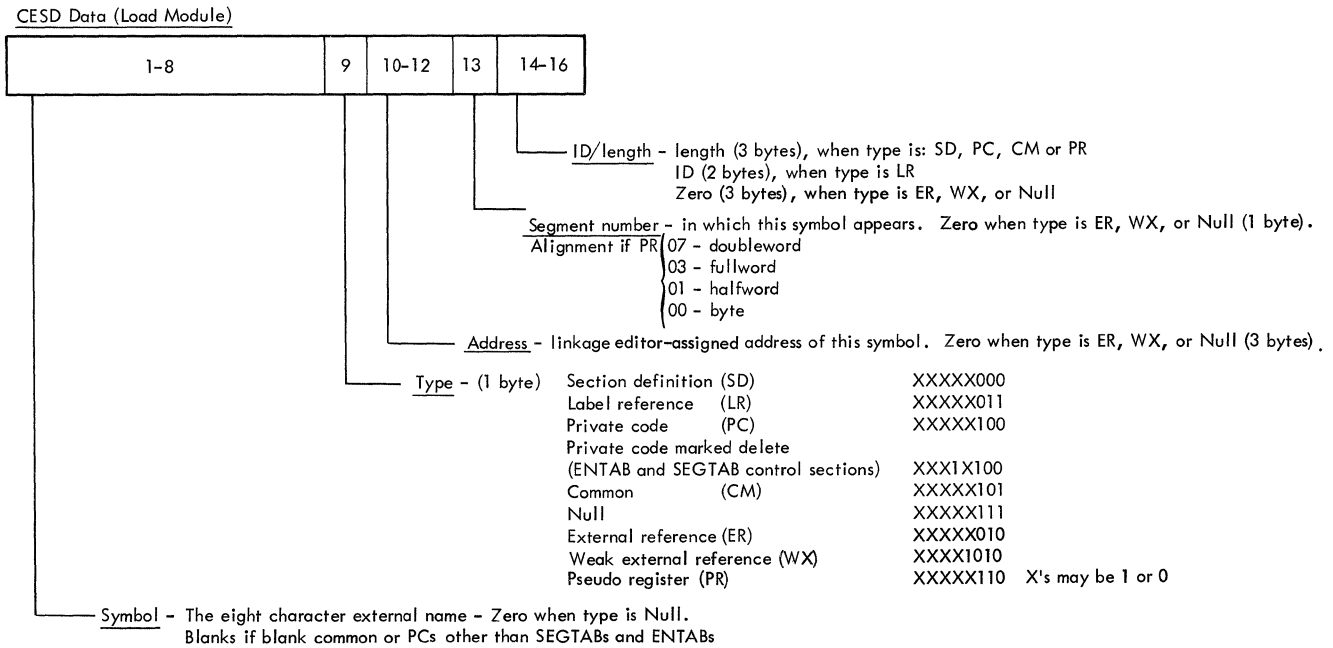


Figure 39. CESD Record - (Load Module)

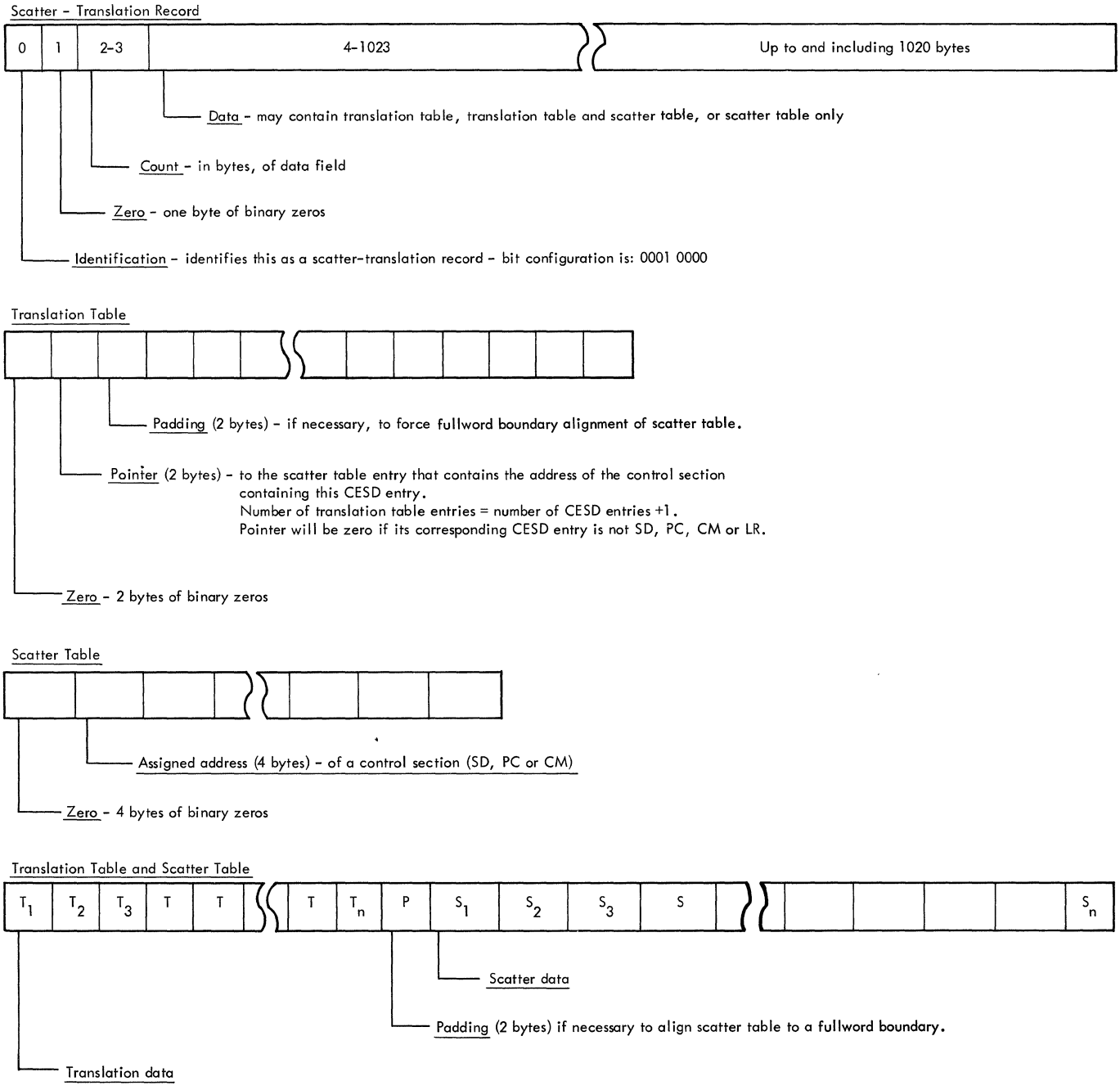


Figure 40. Scatter/Translation Record - Ignored by the Loader

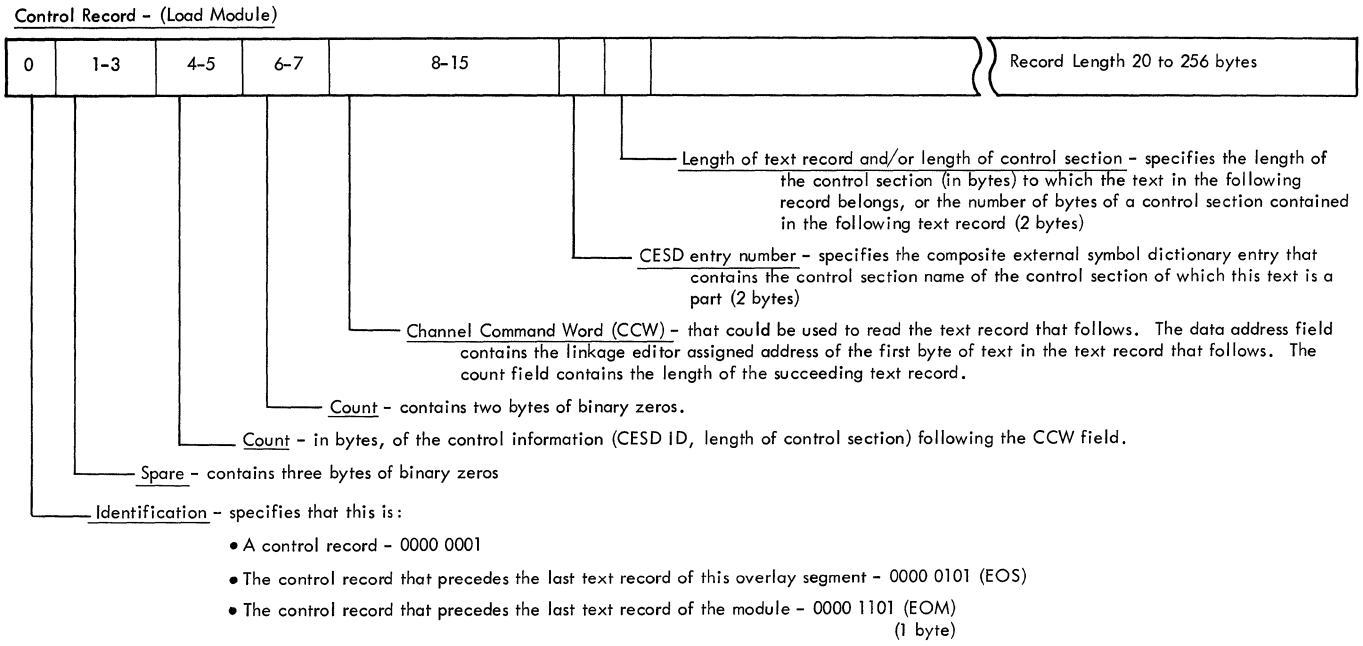


Figure 41. Control Record - (Load Module)

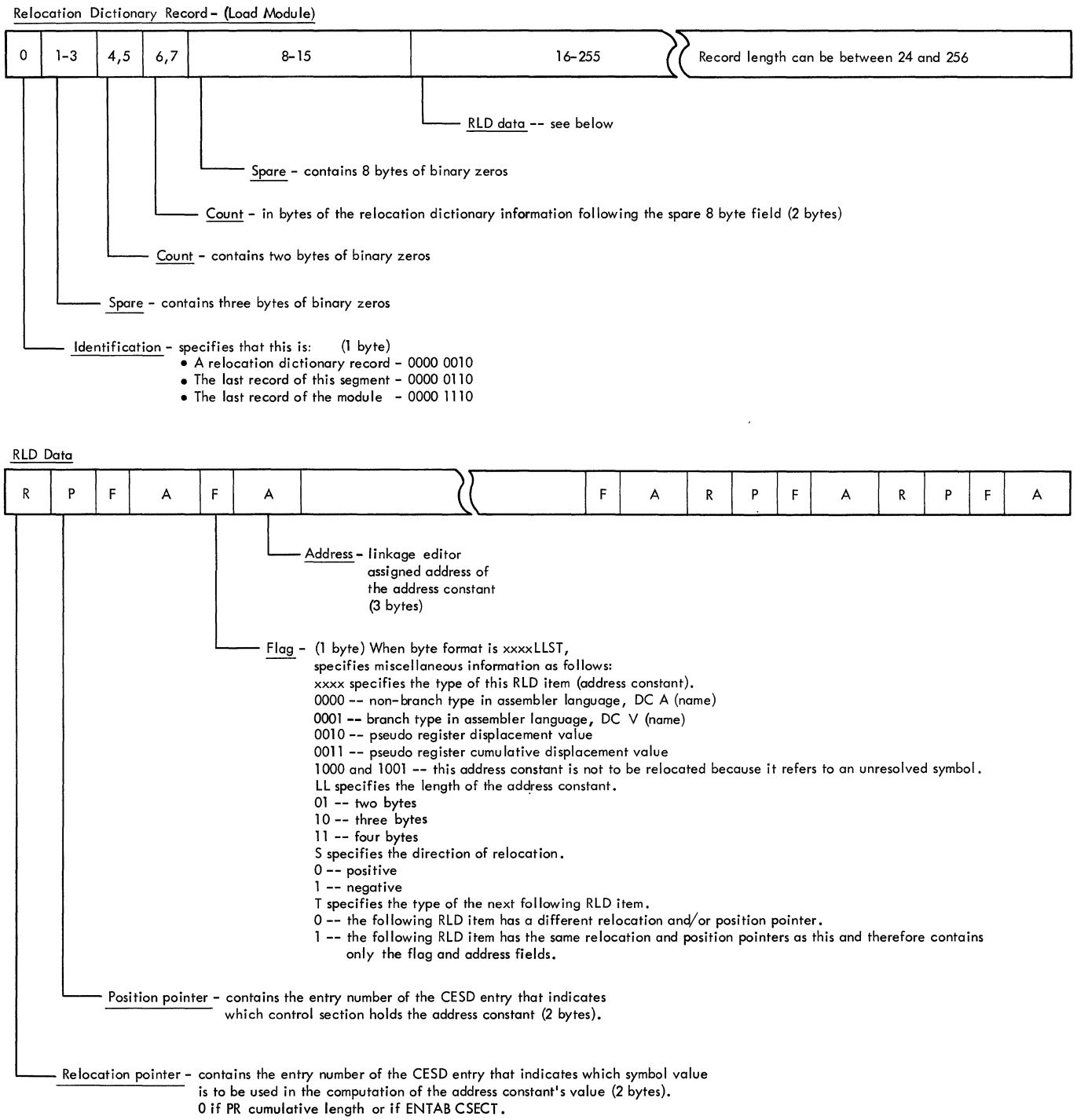
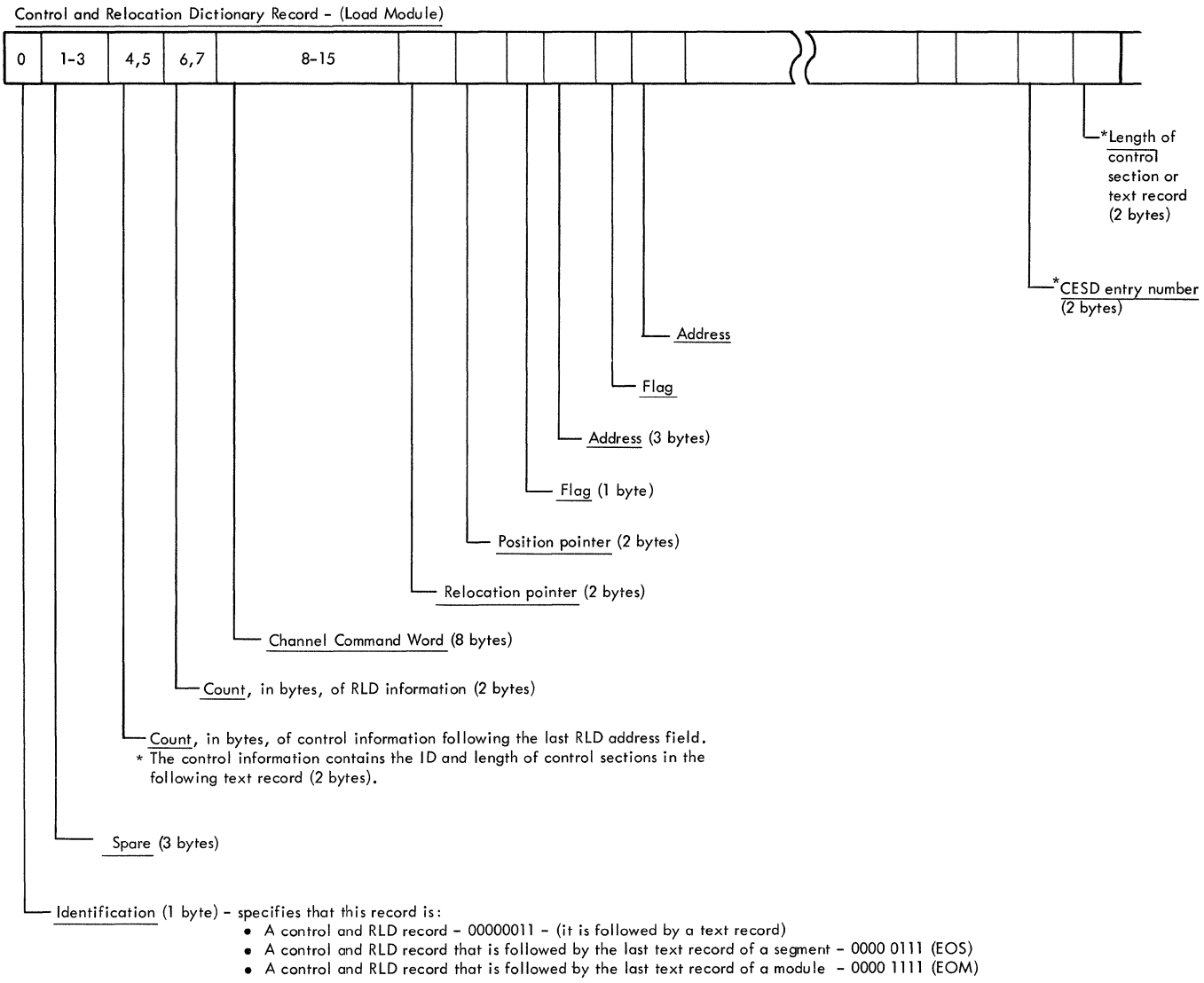


Figure 42. Relocation Dictionary Record - (Load Module)



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 Loader.

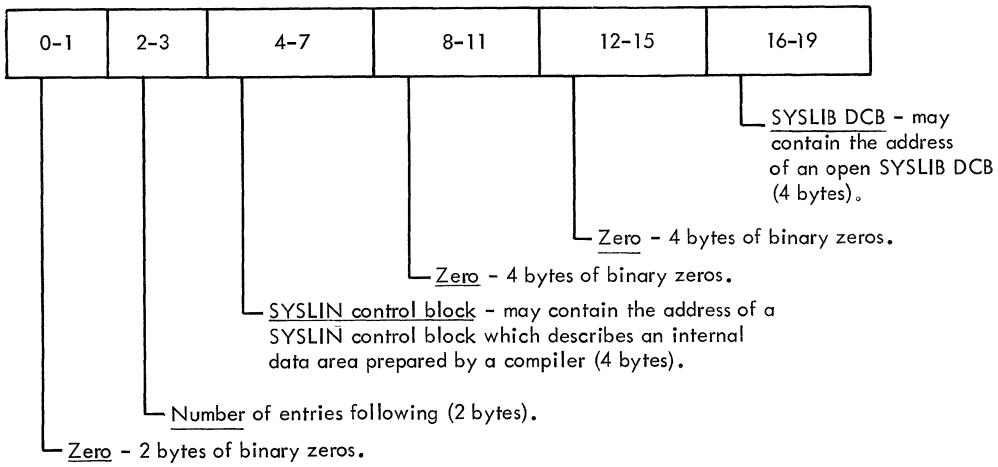
Figure 43. Control and Relocation Dictionary Record - (Load Module)

COMPILER/LOADER INTERFACE FOR PASSED DATA SETS

If the Loader is to process an internal SYSLIN data area (that is, a data area residing in main storage and consisting of contiguous object module records prepared by a compiler) and/or an open SYSLIB data set, the compiler/Loader interface described here is used. The description includes the format of the DCB list, the control block or DCB parameters which must be specified for the data area or data set, the format of an internal data area consisting of either fixed- or variable-length records, and the format of the MOD record.

DCB List

Pointed to by the fourth entry in the parameter list passed to the Loader



- Figure 44. DCB List

Internal SYSLIN Control Block

The SYSLIN control block[1] used to describe an internal input data area should have the following fields initialized:

DCBDEVT = 0, to describe an internal data area and to indicate that an internal SYSLIN control block was passed.

DCBRELAD = starting address of the internal object module records.

DCBBLKSI = length of the entire internal data area.

DCBRECFM = FB, if the internal object module records are in fixed-length format.

VB, if the internal object module records are in variable-length format.

DCBLRECL = length of a logical record if the data set records are in fixed-length format.

Open SYSLIB DCB

The open SYSLIB DCB[2] passed to the Loader should have the following DCB fields initialized:

DCBDSORG = PO

DCBMACRF = R

DCBNCP = 2

DCBRECFM = U, if the SYSLIB data set contains load modules.

F or FB, if the SYSLIB data set contains object modules. (In this case, values for the fields DCBLRECL and DCBBLKSI should also be specified.)

DCBBUFNO = 0

Exit routine addresses may be specified. Before reading SYSLIB, the Loader overlays these addresses with the addresses of its own routines. The Loader also restores these addresses before returning to the caller.

If an open SYSLIB DCB is passed to the Loader, SYSLIB is not closed by the Loader.

---

[1]The control block, has the format and content of a SYSLIN data control block, but is not to be considered a data control block because there is no data management activity in connection with this control block.

[2]A complete description of the content of a DCB is contained in the publication IBM System/360 Operating System: System Control Blocks, Order No. GC28-6628.

Internal Data Area in Fixed-length Record Format

(Logical record length = 72)

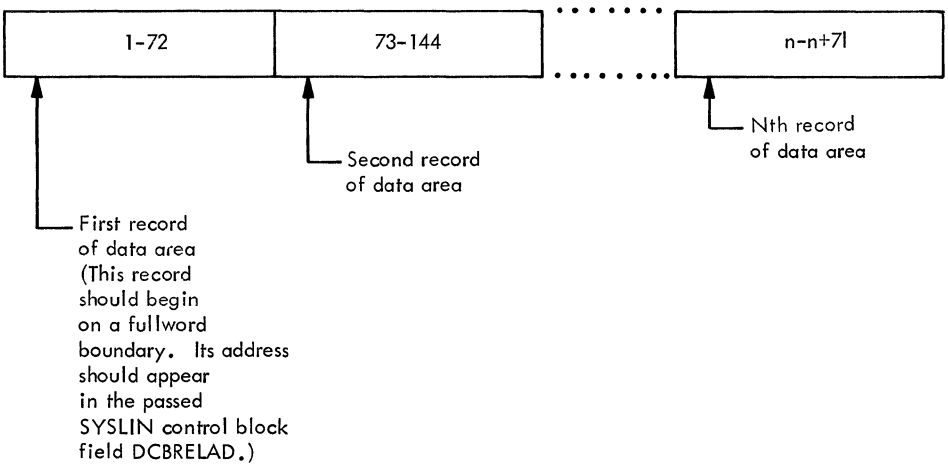
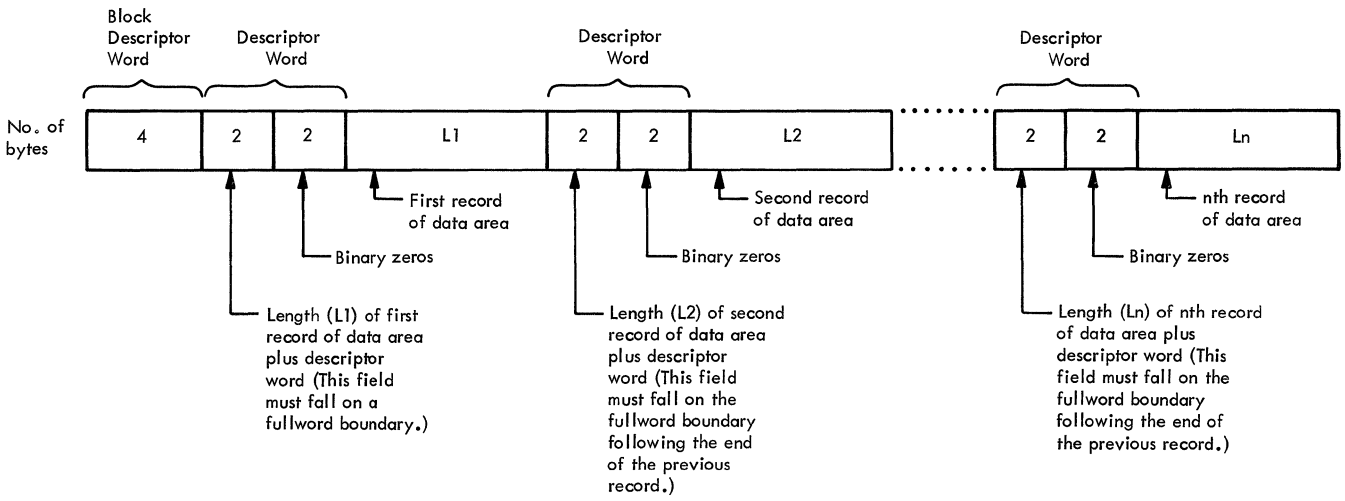
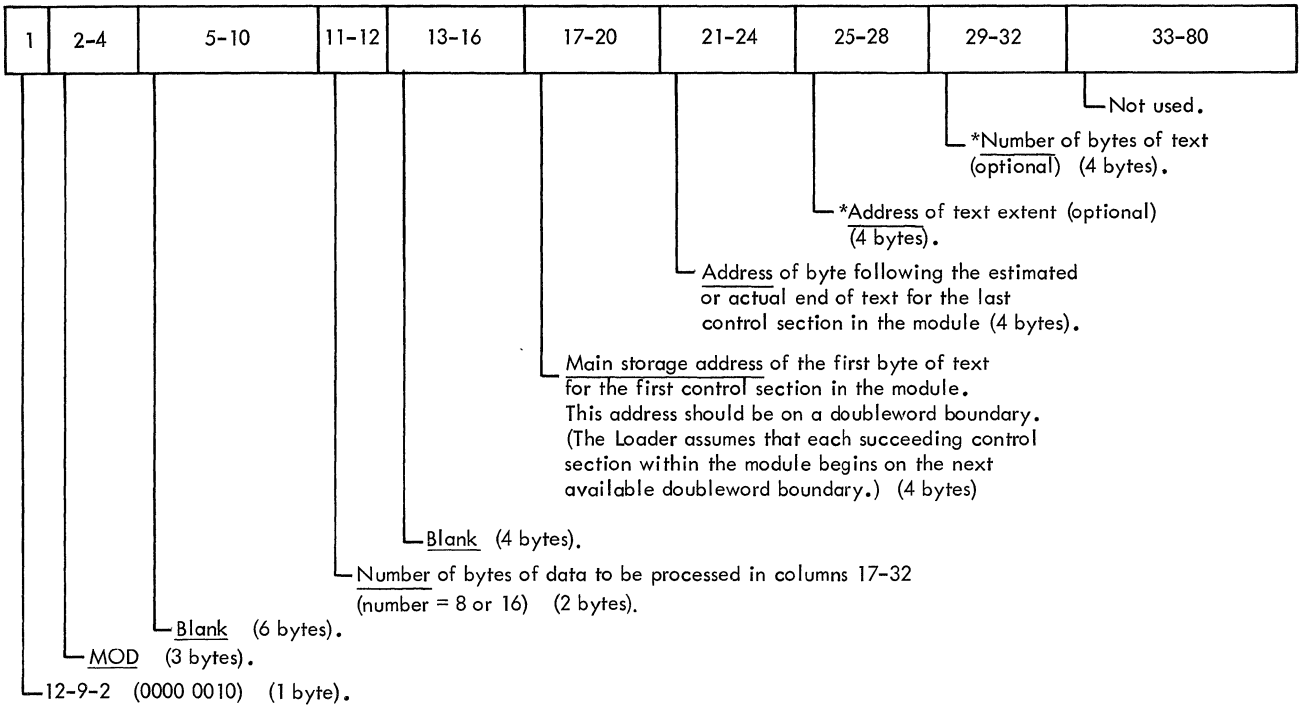


Figure 45. Internal Data Area in Fixed-length Record Format



• Figure 46. Internal Data Area in Variable-length Record Format

MOD Record (Card Image)



*Note: These two fields define storage that is to be identified as part of the loaded program. They are optional, but must occur on at least one of the MOD records in the internal data area if the Loader is invoked via the entry points LOADER, IEWLDRGO, or IEWLOAD. Each occurrence of these two fields defines a new extent of the program. The values must conform to the rules for FREEMAIN parameters, that is, the address must begin on a doubleword boundary and the length must be a multiple of 8.

Figure 47. MOD Record (Card Image)

IDENTIFY MACRO INSTRUCTION - IDENTIFYING LOADED PROGRAM (MVT ONLY)

The IDENTIFY macro instruction, when invoked as described below, permits the Loader to describe a program constructed in subpool 0 so that the program may later be invoked by a macro instruction, such as LINK, XCTL, or ATTACH. The IDENTIFY macro instruction creates a contents directory entry (CDE) and an extent list for the program constructed. These system control blocks allow the MVT Supervisor to identify the program.

Under MVT, the addresses and lengths of the program's extents, the entry point address, and the program name must be passed to the IDENTIFY macro instruction. (The format of the parameter list passed by the Loader to the IDENTIFY macro instruction is shown in Figure 25.) The IDENTIFY macro instruction flags the CDE that it creates to indicate that the program can be invoked by other macro instructions as well as the LOAD macro instruction. Residence of the program in subpool 0 and the absence of the program as a load module on an external device is also indicated in the CDE. The IDENTIFY macro instruction places the CDE on the user's Job Pack Area Control Queue; it also derives the extent list from the parameter list passed to it and stores the extent list within the system queue area.

When the form of the IDENTIFY macro instruction described below is specified, all other operands are ignored. The format is:

Name	Operation	Operand
[symbol]	IDENTIFY	MF=(E, {address of parameter list / (1)})

MF=
indicates the execute form of the macro instruction using a remote parameter list. (The format of the parameter list passed by the Loader is shown in Figure 25.) The address of the parameter list can be loaded into register 1, in which case MF=(E,(1)) should be coded. If the address is not loaded into register 1, it can be coded as an address that is valid in an RX-type instruction, or as one of the registers 2-12 which were previously loaded with the address. A register can be designated symbolically or with an absolute expression, and is always coded within parentheses.

Programming Notes: Failure to meet any of the following requirements will cause an exit with a return code to indicate the reason for unsuccessful completion. The requirements are:

1. The extent list size must be a positive multiple of 8.
2. The addresses in the parameter list must be in subpool 0.
3. The program name should not duplicate a name already on the Link Pack Area Control Queue or the user's Job Pack Area Control Queue.
4. The entry point must be within one of the extents.
5. The caller must be a non-supervisory routine.
6. The program must be run under MVT.
7. The extents must be in the user's region in subpool 0 and they must begin on doubleword boundaries.

When the IDENTIFY macro instruction returns control, register 15 contains one of the following hexadecimal codes:

Code	Meaning
00	Successful completion.
04	Program name and address already exist.
08	Program name duplicates the name of a load module currently in main storage; CDE was not created.
0C	Entry point address is not within an eligible program or system is not MVT; CDE was not created.
14	An IDENTIFY macro instruction was previously issued using the same program name, but a different address; this request was ignored.
18	Parameter list address is not on a doubleword boundary, or the program name specified is already on the Link Pack Area Control Queue or the user's Job Pack Area Control Queue; CDE was not created.
1C	Extent list length is negative, not a multiple of 8, or the extent addresses are not on doubleword boundaries; CDE was not created.
20	Extents are not in subpool 0; CDE was not created.

SECTION 8: DICTIONARY OF ABBREVIATIONS AND ACRONYMS

adcon	address constant
CESD	composite external symbol dictionary
CSECT	control section
DECB	data event control block
DSECT	dummy section
EOM	end of module
ESD ID	external symbol dictionary identification
LD	label definition
LR	label reference
PC	private code
PR	pseudo register
P pointer	position pointer
RLD	relocation dictionary
R pointer	relocation pointer
SD	section definition
TTR	relative track and record address on a direct access device
WX	weak external reference

(Where more than one page reference is given, the major reference appears first.)

READER'S COMMENTS

TITLE *IBM System/360 Operating System*
Loader
Program Logic Manual

ORDER NO. *GY28-6714-1*

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Corrections or clarifications needed:

Page	*Comment*

Please include your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

GY28-6714-1

fold fold

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION
1271 Avenue of the Americas
New York, New York 10020

Attention: PUBLICATIONS

fold fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

GY28-6714-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)