IBM

MVS/Extended Architecture Loader Logic

Licensed Program

Order Number LY26-3901-1
File Number S370-31

Program Number 5665-284
Version 1, Release 1.2

IBM

MVS/Extended Architecture Loader Logic

Licensed Program

Order Number LY26-3901-1
File Number S370-31

Program Number 5665-284
Version 1, Release 1.2

Second Edition (January 1987)

This edition replaces and makes obsolete the previous edition, LY26-3901-0, and its technical newsletters, LN26-8114 and LN26-8155.

This edition applies to Version 1 Release 1.2 of MVS/Extended Architecture Data Facility Product, Licensed Program 5665-284, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Changes" following the preface. Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent publication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370, 30xx, and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

This publication applies to Version 1 and Version 2 of MVS/Extended Architecture Data Facility Product (MVS/XA DFP).

ORGANIZATION

This publication contains the following:

- "Introduction" describes the loader as a whole, including its relationship to the operating system. This section also describes the major divisions of the program and how they work together.
- "Method of Operation" provides an overview of, and an introduction to, the logic of the loader. This section also contains detailed descriptions of specific operations.
- "Organization of the Loader" describes the organization of the loader and the control flow within it.
- "Microfiche Directory" directs the reader to named areas of code in the program listing, which is contained on microfiche cards.
- "Data Areas" illustrates the layout of tables and control blocks used by the loader. These layouts may not be essential for an understanding of the program's logic, but they are essential for analysis of storage dumps.
- "Diagnostic Aids" includes the general contents of the register at entry points to program components, definitions of the internal error codes, and a list of service aids available with the loader.
- "Appendix. Error Messages, Etc." on page 102 contains a list of error messages and the routines and CSECTs in which they originate. This section also contains a list of loader input conventions and restrictions, and detailed descriptions of input record formats.
- "List of Terms and Abbreviations" lists the terms and abbreviations used in this book, and what they mean.

An index is also included.

PREREQUISITE KNOWLEDGE

To use this book effectively, you should be familiar with the following topics:

- Assembler language functions and specifications under Assembler H
- How to analyze a main storage dump from MVS/XA
- General concepts of the linkage editor and loader.

REQUIRED PUBLICATIONS

- Assembler H Version 2 Application Programming: Language Reference, GC26-4037, for a description of basic assembler language functions.
- MVS/Extended Architecture Debugging Handbook, LC28-1164 through LC28-1168, for details on how to analyze a main storage dump.
- MVS/Extended Architecture Linkage Editor and Loader User's Guide (GC26-4011 for Version 1; GC26-4143 for Version 2) for a description of the linkage editor and loader.

RELATED PUBLICATIONS

References are made within the text to various related publications. Separate tables of related publications for Version 1 and Version 2 are provided below.

VERSION 1

Short Title	Publication Title	Order Number
Assembler H V2 Application Programming: Language Reference	Assembler H Version 2 Application Programming: Language Reference	GC26-4037
Debugging Handbook	MVS/Extended Architecture Debugging Handbook, Volumes 1 through 5	LC28-1164[1] LC28-1165 LC28-1166 LC28-1167 LC28-1168
JCL	MVS/Extended Architecture JCL	GC28-1148
Linkage Editor and Loader User's Guide	MVS/Extended Architecture Linkage Editor and Loader	GC26-4011
Supervisor Services and Macro Instructions	MVS/Extended Architecture System Programming Library: Supervisor Services and Macro Instructions	GC28-1154

Note:

[1] All five volumes may be ordered under one order number, LBOF-1015.

Short Title	Publication Title	Order Number
Assembler H V2 Application Programming: Language Reference	Assembler H Version 2 Application Programming: Language Reference	GC26-4037
Debugging Handbook	MVS/Extended Architecture Debugging Handbook, Volumes 1 through 5	LC28-1164[1] LC28-1165 LC28-1166 LC28-1167 LC28-1168
JCL	MVS/Extended Architecture JCL	GC28-1148
Linkage Editor and Loader User's Guide	MVS/Extended Architecture Linkage Editor and Loader	GC26-4143
Supervisor Services and Macro Instructions	MVS/Extended Architecture System Programming Library: Supervisor Services and Macro Instructions	GC28-1154

Note:

[1] All five volumes may be ordered under one order number, LBOF-1015.

SUMMARY OF CHANGES

RELEASE 1.2 LIBRARY UPDATE, JANUARY 1987

SERVICE CHANGES

Information has been added, corrected, or deleted to reflect technical service changes.

CONTENTS

FIGURES

INTRODUCTION

This section provides a general description of the loader. It includes the purpose and functions of the program, its physical and environmental characteristics, and operational considerations necessary for its use. The generalized theory of loading is also discussed in this section.

PURPOSE

The purpose of the loader is to combine input object and load modules into an executable program in virtual storage. In this regard, 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.)

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

- Elimination of scheduler overhead, since loading and execution occur in a single job step
- Elimination of linkage editor I/O for intermediate and final output
- Elimination of certain linkage editor functions, such as control statement processing and overlay structuring
- Reduction of time required for reading input, through improved buffering techniques
- Reduction of time required for library search, through use of link pack resident modules
- 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:

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

FUNCTIONS

The loader performs the basic logical functions of the linkage editor and of the program fetch module. 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 regard, the loader functions as the program fetch module does.

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.

VIRTUAL STORAGE REQUIREMENTS

Loader operation requires about 25K bytes of virtual 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 16K bytes), for the data management access methods (about 6K bytes), and for loader buffers and tables (about 3K bytes). If the access methods are resident, and if the loader code is resident in the link pack area, part of the loader storage may be allocated from system storage.

Figure 1 shows the loader virtual storage layout when loading below the 16-megabyte virtual storage line. Figure 2 on page 3 shows the storage layout when loading above the 16-megabyte virtual storage line.

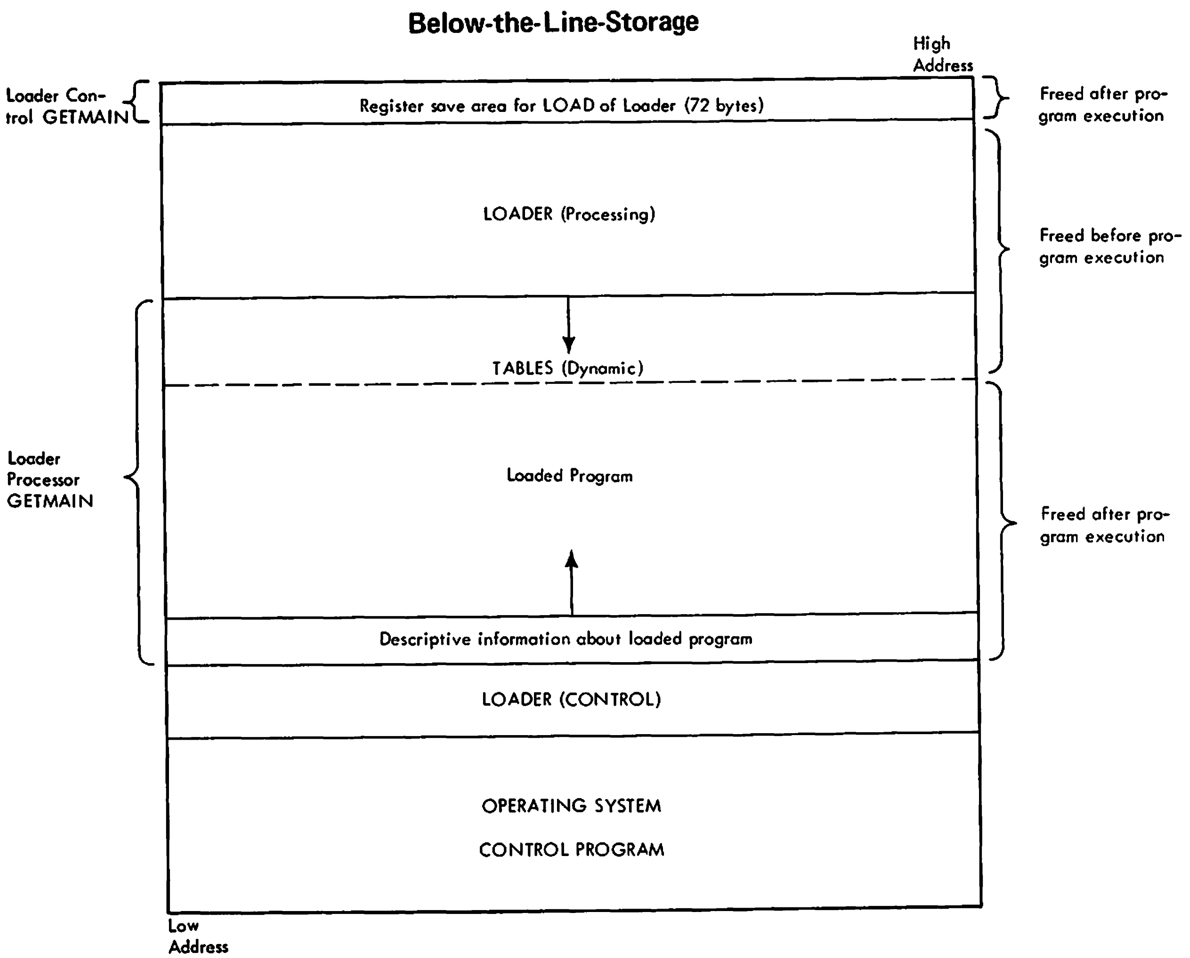


Figure 1. Loader Storage Layout (Below-the-Line Loading)

---

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

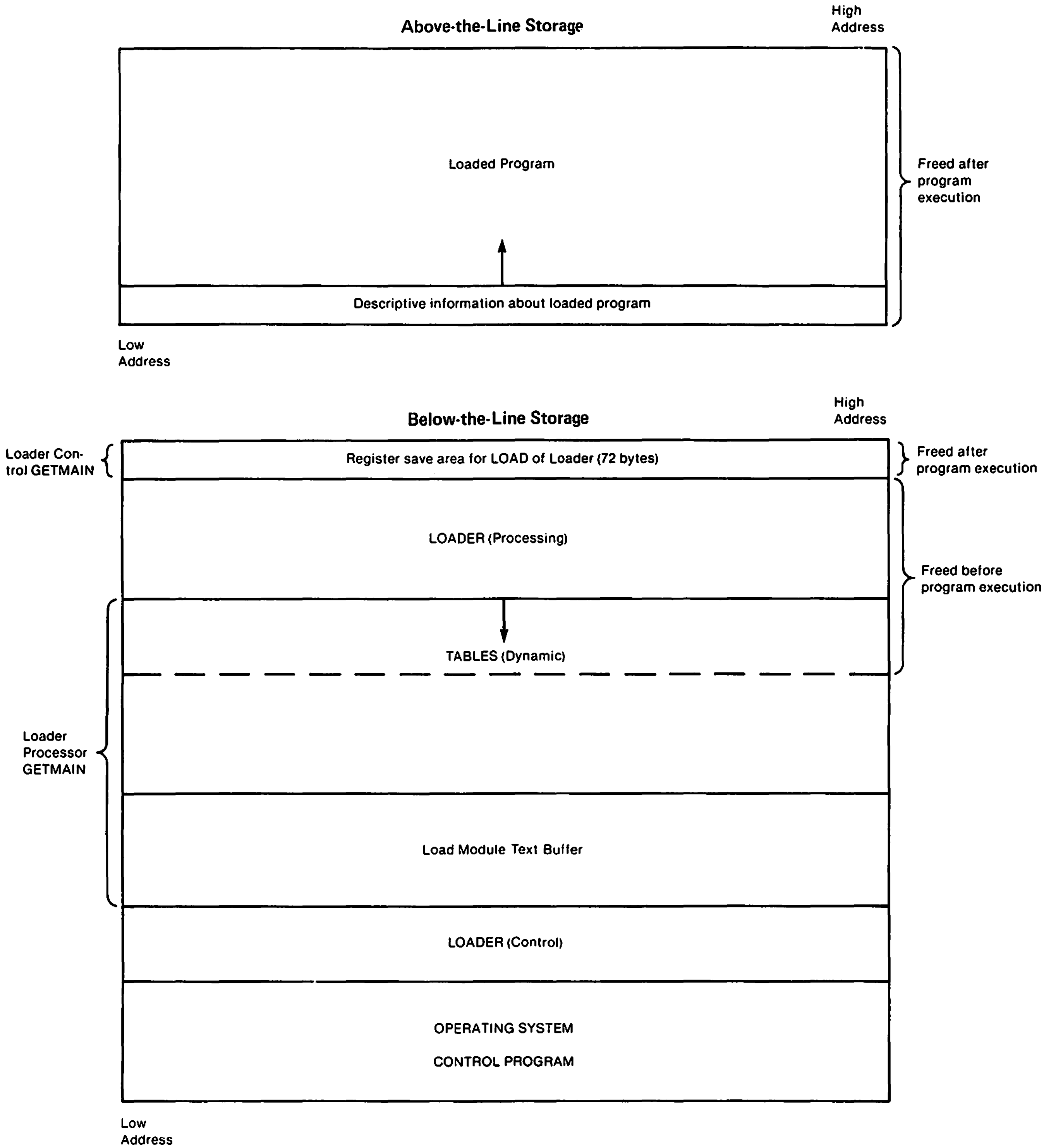


Figure 2. Loader Storage Layout (Above-the-Line Loading)

ENVIRONMENT

The loader can be used either in batch mode, or under the time sharing option (TSO).

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 SYSLIB data set. Input may be from a card reader, magnetic tape, or a direct access device. Input may be a concatenation of data sets from different types of devices. Input may also be in the form of an internal input data area prepared by a compiler.

An automatic search of a system library can occur to complete the input. The automatic search requires use of the SYSLIB data set. The SYSLIB data set 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 resident modules listed in the contents directory entry queue in the loaded program.

The loader uses the SYSLOUT data set for both diagnostic messages and module maps, and uses the SYSTERM data set only for diagnostic messages. These data sets may be used in conjunction with each other 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. Figure 3 on page 5 illustrates the relationship between the portions of the loader.

The loader consists of two loads, the first contains module HEWLCTRL, the control portion. The other load contains control sections HEWLDDEF, HEWLIOCA, HEWLRELO, HEWLIDEN, and HEWLLIBR, which together perform program loading. Because of the interrelationships among module functions, the loader is not a candidate for overlay structuring.

The control portion of the loader executes in 24-bit addressing mode; the processing portion executes in 31-bit addressing mode. Both portions of the loader reside below the 16-megabyte virtual storage line.

OPERATIONAL CONSIDERATIONS

Loader operation depends on the types of input received and on the types of user options specified.

Input to the loader may be in the form of load modules produced by the linkage editor, and/or as object modules produced by the following language processors: ALGOL, COBOL, FORTRAN, PL/I, RPG,

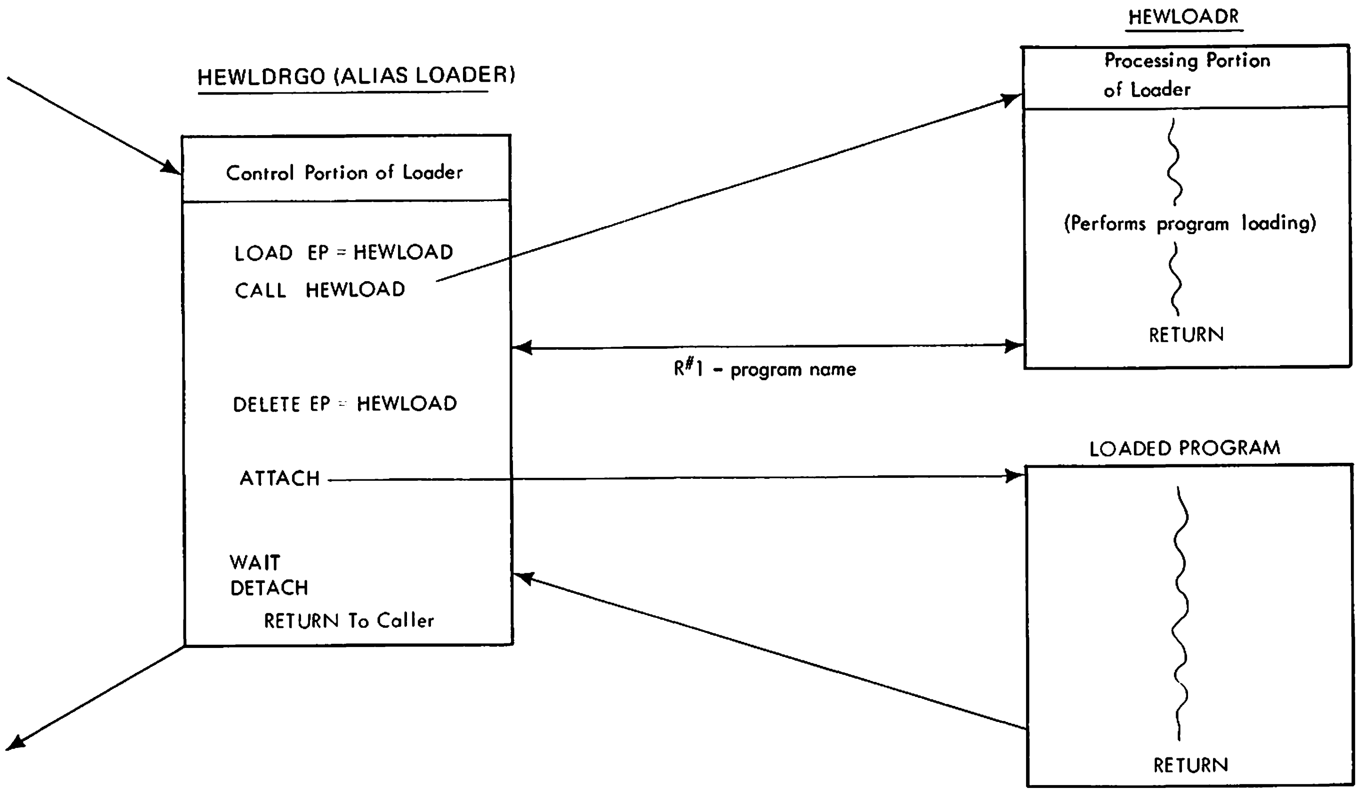


Figure 3. Loader Control Logic Flow

and Assembler.[2] Input may be from an external device, or it may be as one or more internal object modules (that is, a data area that resides in virtual storage and consists of contiguous object module records). If inputting 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, you cannot concatenate input from an external device to it.

INPUT MODULE STRUCTURE

Object modules and load modules have basically the same logical structure (see Figure 4 on page 6). 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.

---

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

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

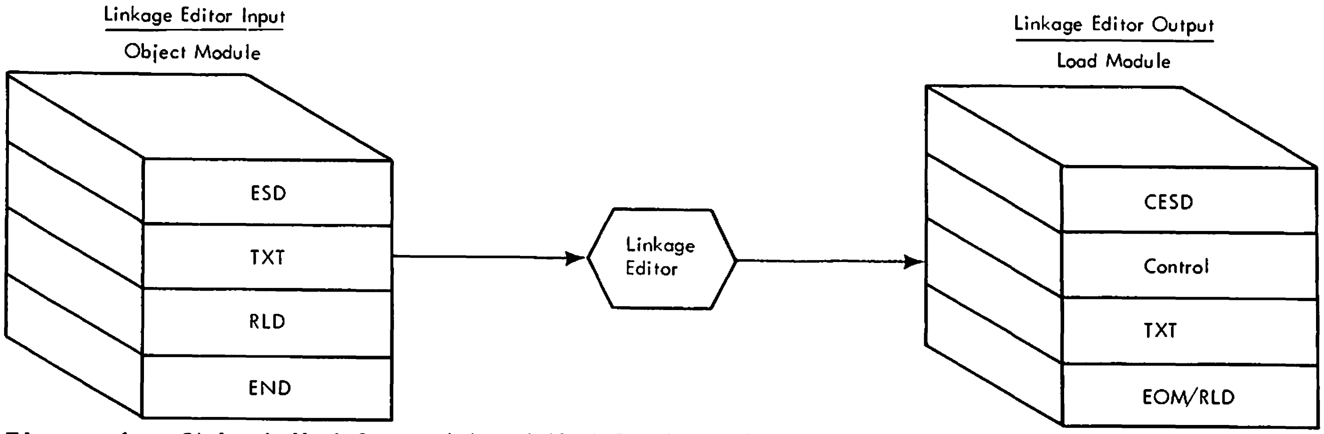


Figure 4. Object Module and Load Module Structure

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 referenced in another. Thus, symbolic references are either internal or external with respect to the module in which they occur. A symbol that 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). If the module contains no relocatable address constants, an RLD is not present.

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 the Appendix.

External Symbol Dictionary (ESD)

The external symbol dictionary contains entries for all external symbols defined or referenced 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 (RLD) contains at least one entry for every relocatable address constant (thus, one 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 (created by a compiler or an assembler) have the same sequential order, so the ESD ID gives the dictionary entry number of an external symbol.[3] (The linkage editor renumbers the ESD IDs to maintain the ordered relationship when combining modules into a load module.)

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

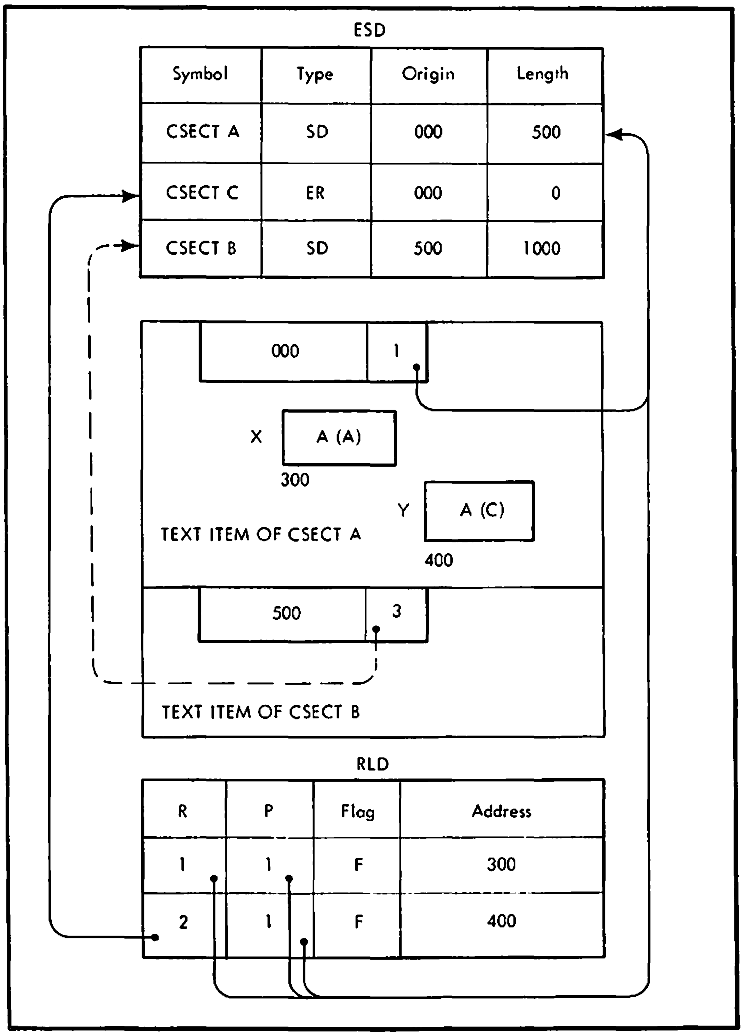
In RLD entries the ESD IDs are used to show two relationships between the RLD and ESD entries:

- The RLD relocation pointer (R pointer) gives the ESD ID for the symbol to which the address constant refers.
- The RLD position pointer (P pointer) gives the ESD ID for the CSECT in which the address constant occurs.

Figure 5 on page 8 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 X's 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.

---

[3] In an object module, an ESD item with type=LD cannot 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 5. Example of an Input Module

LOADER OPTIONS

User options may be specified by parameters listed on the EXEC job control statement[4], or may be passed internally by a program requesting the loader via LINK, LOAD, ATTACH, or XCTL macro instruction.[5] If the options are not user specified, the defaults provided by the loader are used.

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

---

[4] See JCL manual.

[5] See Supervisor Services and Macros.

Figure 6 describes the loader options. Parameters are listed with their associated options. Some options use 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 was retained for compatibility with the linkage editor option NCAL.) Figure 6 also indicates defaults for the options.

Parameters	Options	Defaults
RES NORES	The loader searches the link pack area queue for resident modules after primary input completes, 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=300K
EP=	Specifies an external name to be used as the entry point of the loaded program.	No default[1]
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 to the SYSLOUT data set.	NOTERM
NAME=	Specifies the name to use as the name of the loaded program.	GO[1]
AMODE=	Specifies the addressing mode to be in effect when entering the module at its entry point.	24
RMODE=	Specifies the residence mode that applies to the module.	24

Figure 6. Loader Options

Note to Figure 6:

The loader assigns an entry point to the loaded program if no name was specified.

GENERAL THEORY OF OPERATION

In processing input modules, the loader assigns virtual-storage addresses to the control sections to be included in the loaded program. The loader also resolves external references in the CSECTs.

Because the origin of each input module 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.

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

METHOD OF OPERATION

This section describes the logic of the loader. It contains an introduction that 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 to the corresponding steps of a function through lettered references. For example, (A) refers to the portion of a diagram that shows the GETMAIN function in "Diagram B1. Loader/Scheduler Interface and Initialization" on page 58. (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 (Figure 14 on page 28 and Figure 15 on page 29). These illustrations stress the changes to data; the diagrams 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. The system interface is shown in "Diagram A1. Overall Loader Operation (Above-the-Line Loading)" on page 55. See also "Diagram A0. Overall Loader Operation (Below-the-Line Loading)" on page 54.

Initialization

When the loader begins processing, it performs initialization to prepare for all subsequent processing. The operations included in initial processing are:

- Analyzing control information
- Initializing virtual storage
- Initializing DCBs
- Opening data sets.

"Diagram B1. Loader/Scheduler Interface and Initialization" on page 58 shows initialization processing.

Input Control and Buffer Allocation

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) is read into the RLD buffer. in below-the-line loading, Text from load modules is read directly into the loaded program's storage area. In above-the-line loading, text from load modules is read into the load module text buffer and then moved into the loaded program's storage area. "Diagram C1. Primary Input Control and Buffer Allocation" on page 59 shows input control and buffer allocation.

Primary Input Processing

The loader performs the processing for all SYSLIN modules. (All overlay and scatter control statements from load modules and SYM records are ignored.) "Diagram D1. Object Module Processing" on page 60, and "Diagram D2. Load Module Processing" on page 61, both show the flow of primary input processing.

External Symbol Dictionary Processing

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 that allow translation of the input ESD IDs to CESD addresses. The loader calculates storage addresses and stores them in the CESD entries. See "Diagram D3. ESD Record Processing (Generalized)" on page 62 through "Diagram D6. Example of ESD ID Translation" on page 65, for depictions of external symbol dictionary processing.

Text Record Processing

For an object module, 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 virtual-storage addresses. In below-the-line loading, the loader reads the record directly into the loaded program's storage area. CSECTs at the end of the record that are to be deleted are not read. CSECTs within the record that are to be deleted are overlaid when the CSECTs that are to be kept are compressed. In above-the-line loading, the loader reads the record into the load module text buffer, located in below-the-line storage. If all CSECTs in the record are not to be kept, the entire record is moved into the loaded program's storage area, above the line. If all CSECTs in the record are not to be kept, only the CSECTs to be kept are moved into the loaded program's storage area. See "Diagram D7. Object Module Text Processing" on page 66, through "Diagram D9. Load Module Text Processing (Above-the Line Loading)" on page 68, for depictions of text record processing.

Relocation Dictionary Processing

The loader builds its RLD table from information contained in the RLD records. It 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 address constants (adcons), and uses the flag field to determine the method of address constant relocation required. "Diagram D10. RLD Record Processing" on page 69 shows relocation dictionary processing.

Address Constant Relocation Processing

When resolving external references in the CESD, 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

If some unresolved external references remain 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 address constants 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. "Diagram E1. Secondary Input Processing" on page 70 shows secondary input processing.

Final Processing

After processing all input for the loaded program, the loader:

- Assigns addresses for the common areas and for displacements in the external dummy section
- Issues messages for unresolved ERs
- Determines the address of the loaded program's entry point.

Identifying Loaded Program

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.[6] At this time, a condensed symbol table may also be constructed for use by test facilities available under the Time Sharing Option while the program is executing.

End of Loading

Before ending loader processing, the loader:

- writes out the diagnostic message dictionary and any remaining diagnostic messages
- closes data set DCBs
- sets up return information
- frees storage not required for the loaded program.

INITIALIZATION (HEWLIOCA)

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. Loader/Scheduler Interface and Initialization" on page 58.

---

[6] This processing is performed only when the processing portion of the loader is invoked, either directly or by the control portion of the loader, by the name HEWLOAD.

ANALYZING CONTROL INFORMATION

Loader operation depends on the control information. Control information consists 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. (The defaults are contained in the control section HEWLDDEF.)

(A) To analyze the control information, the loader obtains a temporary work area, INITMAIN. (See "Data Areas" on page 83 for the contents of INITMAIN.) The loader saves the default ddnames and option indicators in the temporary work area. An EXTRACT macro instruction is then issued to determine whether the loader is currently operating under Time Sharing Option, and an indicator is set in INITMAIN. If the processing portion of the loader was invoked through the entry point HEWLOAD, 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 was specified incorrectly, the default value is used.

Specifying the EP option saves the associated entry point name in INITMAIN.

Specifying the NAME option saves the associated program name in INITMAIN. Otherwise, the default name **GO is used.

If the user specified the AMODE option, the loader verifies that the value is either 24, 31, or ANY. If so, the value is saved in INITMAIN; if not, the loader ignores the AMODE option.

If the user specified the RMODE option, the loader verifies that the value is saved in INITMAIN; if not, the loader ignores the RMODE option.

After all the loader options have been processed, the loader examines the AMODE and RMODE values. If only one was provided in the options, the loader supplies the implied companion value. If the user specified both values in the options, the loader verifies that the combination is valid. If not, the loader ignores both specified values.

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, the loader checks for the addresses of alternates for the data control blocks. Both addresses, if specified, must be 24-bit-only addresses; otherwise, they are ignored. The loader will accept a SYSLIN control block if it describes an internal data area. It saves the address of the SYSLIN control block and sets an indicator for an internal SYSLIN data area 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 the Appendix.)

The loader will accept an alternate SYSLIB DCB if the alternate SYSLIB DCB describes a data set that has been opened. The loader also saves the address of this DCB and sets an indicator for an open library data set in INITMAIN.

INITIALIZING VIRTUAL STORAGE

(B) Using the GETMAIN macro instruction, the loader obtains the required below-the-line storage from the supervisor. The request is conditional and variable. The maximum amount requested is for 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 amount. If at least 2K bytes of storage is still unavailable, an 804 or 80A system abend occurs.

If the supervisor returns virtual storage space, the loader establishes its permanent communication area. (The communication area is described in "Data Areas" on page 83.) The loader then moves the information stored in INITMAIN to the communication area.

If a user option specified an RMODE value of ANY, the loader obtains the required above-the-line storage from the supervisor using the GETMAIN macro instruction. The request is conditional and variable. The minimum and maximum values are the same as those used in obtaining below-the-line storage. If the supervisor does not return storage, loading takes place in the below-the-line storage already obtained. If the supervisor returns virtual storage space, the loader initializes values in the communication area required for above-the-line loading.

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 required for use of its data sets. If the TERM option was specified, space is reserved for a SYSTERM DCB, two DECBs, and two buffers. Unless an internal SYSLIN data set was passed to the loader, a SYSLIN DCB must be prepared and opened. Similarly, unless the NOPRINT option was 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.

Because 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 usable. 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 virtual 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 opening 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 to OFF and 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 terminates. If SYSLIN opened properly, the loader sets the "unlike attributes" indicator in the DCB to signify that SYSLIN may consist of a concatenation of data sets with unlike record formats. The buffers for the first input module are then allocated as described under "Buffer Allocation" on page 17.

Redrive

If the loader encounters a control section having an RMODE of 24 while loading a program above the line (because the first control section encounted had an RMODE of ANY), the loader will abandon the above-the-line loading. The loader then releases the above-the-line storage obtained, and closes and reopens the SYSLIN data set. Finally, the loader reinitializes the communication area for below-the-line loading and restarts the loading process. An error message is issued indicating that this second attempt at loading was made.

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 virtual storage. This is indicated by indicators (CMFLAG3) in the communication area as well as by 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.[7] (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 it reaches the end of the current input, the loader checks for additional SYSLIN input.[8]

Another data set in SYSLIN is indicated unless both the end-of-file and end-of-concatenation switches are set to ON. When the loader opens a new data set in SYSLIN input, the loader determines the new attributes by using the same procedures as those used during loader initialization for the first input data set.

---

[7] The end-of-buffer condition signifies both end-of-file and end-of-concatenation conditions for an internal data area.

[8] 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 to ON. They are tested at the end of each module.

BUFFER MANAGEMENT (HEWBUFFR)

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 use by subsequent modules.

Buffer Deallocation

If both the current and previous inputs consist of load modules, the loader uses the same buffer and DECBs. This is possible because the buffer-DECB requirement for load modules is constant. Figure 7 on page 18 illustrates the buffers 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 8 on page 19.) The first four (4) bytes of each freed area are used to store a pointer to the next freed area in the chain. The second four (4) bytes give the size of the current area. (The size is always rounded to doubleword value.) See Figure 8 for an illustration of freed area chaining.

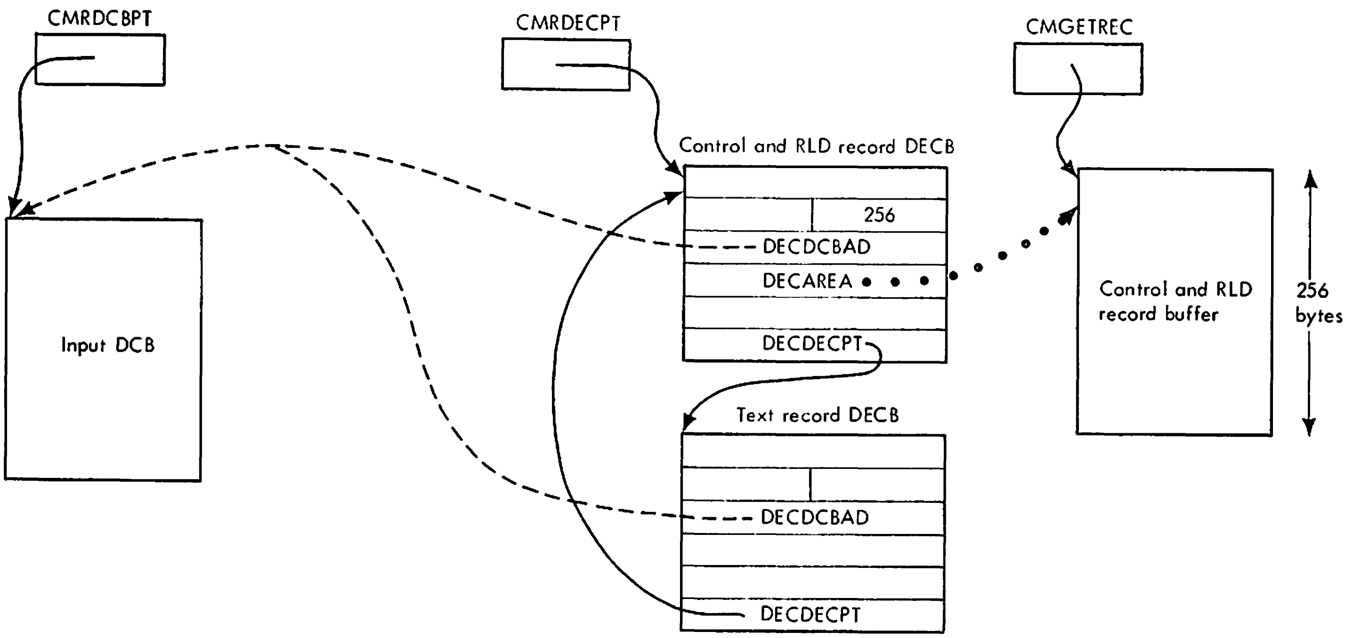
Before chaining an area deallocated from a DECB or a buffer, the loader checks the area's location against the pointers of the other areas in the chain for contiguity. Contiguous freed areas are combined under a single pointer. For example, in Figure 8, Freed Area 1 could consist of areas from three separate deallocations: One from 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 and 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 and the number from DCBNCP, where the value from DCBBUFNO was stored.) The allocation procedure repeats 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 9 on page 20 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 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). In below-the-line loading, the loader reads load module text directly into the loaded program's storage area. In above-the-line loading, the loader reads load module text into the load module text buffer located in below-the-line storage, and moves the text into the loaded program's storage area above the line. 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



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

Figure 7. Load Module Storage Allocation for Buffer and DECBs

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 the DECB. If the size of the freed area is greater than that of 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 exhaust the table and buffer area, the loading process terminates and sends a printed message 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 primed buffer.

At the beginning of processing for a module, the DECB pointer (CMRDECPT) specifies the DECB associated with the first primed buffer (see Figure 9.) 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

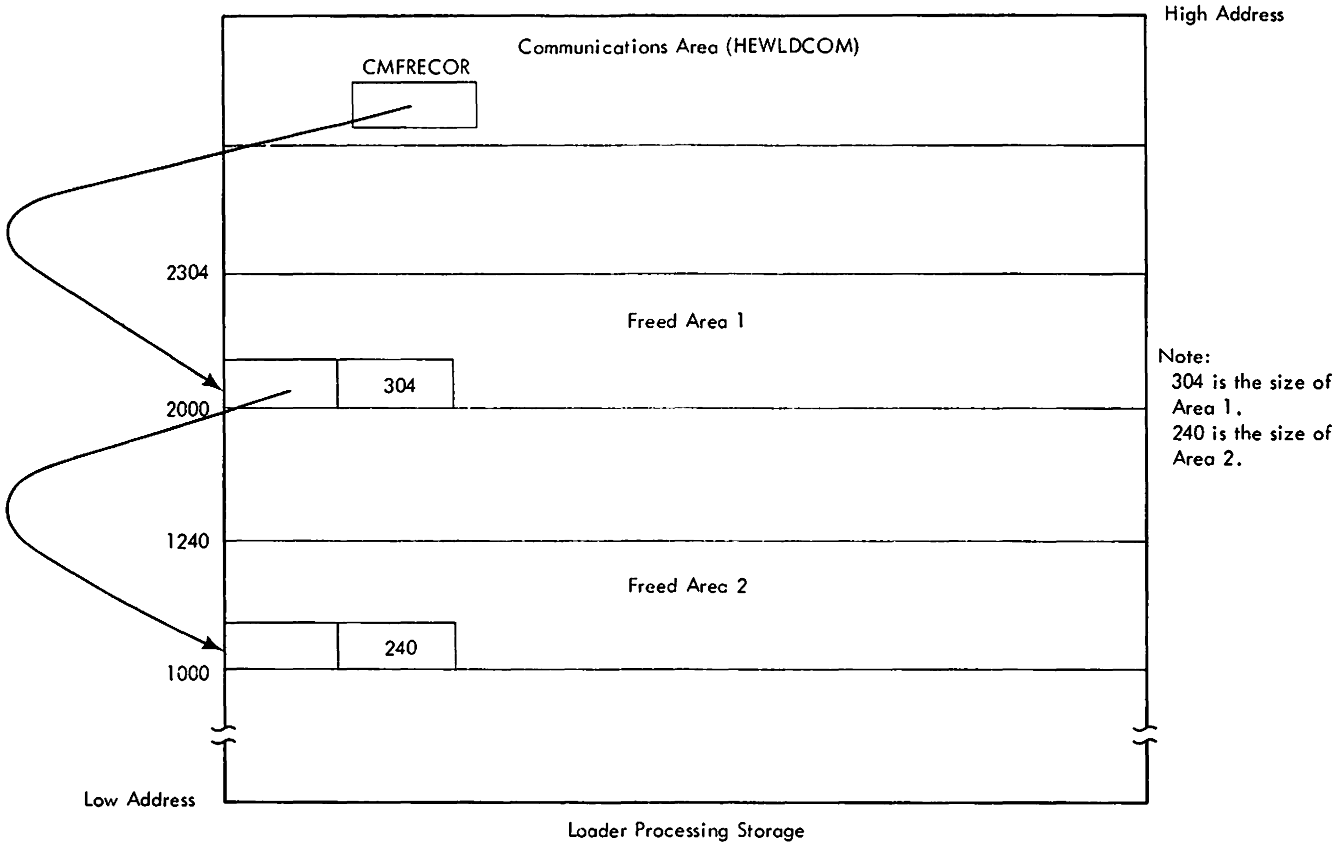
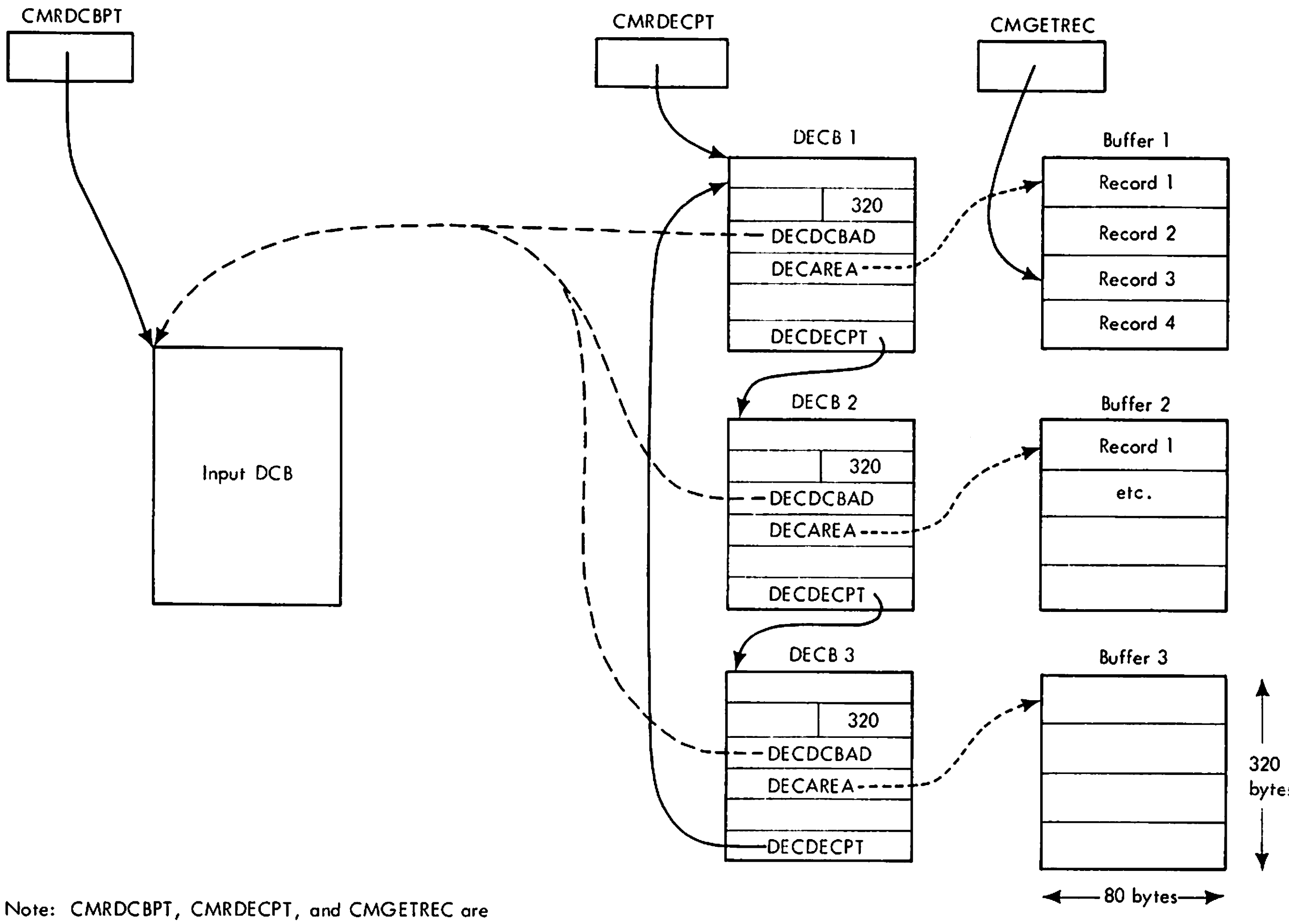


Figure 8. Freed Areas from Buffer-DECB Allocation

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 repeats until the end of the module is recognized.

READING INTERNAL OBJECT MODULE INPUT

Record formats for internal object modules prepared by a compiler may be of fixed or variable type. 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 to ON.



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

Figure 9. Storage Allocation of Buffers and DECBs for Object Module Input

READING LOAD MODULE INPUT

For load modules, the record format type 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 nontext records of load modules read into the same buffer. This buffer, the RLD buffer, has the same length as the maximum length of nontext records processed by the loader (256 bytes).

In below-the-line loading the loader allocates a DECB for reading load module text, but does not allocate a buffer because the text is read directly into the loaded program's assigned area. In above-the-line loading the loader allocates both a DECB for reading load module text, and a load module text buffer, into which all the text is read before being moved to the loaded program's assigned area. The loader determines the address that receives the text during module processing. At the time a text record is read, the following record is also read because it is always nontext.

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 to ON so that no further input will process.

Figure 10 shows processing differences for object and load modules. Input module processing for object and load modules is shown in "Diagram D1. Object Module Processing" on page 60, and in "Diagram D2. Load Module Processing" on page 61.

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.	After processing the entire ID/length list, the loader reads load module text directly into the loaded program's storage area. (below-the-line loading), or into the load module text buffer (above-the-line loading).
RLD	No difference.	No difference.
Relocation	No difference.	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.

Figure 10. Object and Load Module Processing Differences

Load module record types consist of composite ESD, control, RLD, control/RLD, text, SYM, IDR and scatter/translation. When the loader recognizes a SYM, IDR, or scatter/translation record, it simply ignores that record and requests another control record. Descriptions follow for those load module records that the loader does process. (For detailed descriptions, see the record formats in "Appendix. Error Messages, Etc." on page 102.)

- CESD: Each record contains no more than 15 ESD entries.[9] The first 8 bytes give the following control information for the entries in that record: (1) the ESD ID of the first entry, (2) the number of bytes occupied by the entries, and (3) an indication of whether the CESD entries contain overlay segment numbers, or AMODE and RMODE data.

- Control: These records give control information about the module text on the following text record. They contain the related ESD IDs and the lengths of each control section in the following text record, and an indication of EOM, when pertinent. Control records also contain a channel command word (CCW), the linkage editor-assigned relative address, and the total length of the text record. The loader uses this information to read the text.

- Text: These records contain the control sections with the module instructions and data. 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 above. 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. The control information contains the virtual 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 following text records are ignored until an END statement processes.

EXTERNAL SYMBOL DICTIONARY (ESD) PROCESSING (HEWLESD)

The loader processes records from the input record External Symbol Dictionary (ESD) to resolve symbols used in internal and external addressing. Resolution ensures that each named location in the text for the loaded program has a unique symbol.[10]

To resolve symbols the loader builds a composite ESD (CESD) from individual ESDs and CESDs in the input. The loader creates CESD entries as required during processing of input entries. See "Data Areas" on page 83 for detailed descriptions of CESD entries.

Because of the outcome of ESD processing, the loader 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.[11] For a single module, the loader records multiple ESD entries for a symbol in the translation

---

[9] The loader can accept a maximum of 1024 ESD entries per input module.

[10] Names for areas of private code or for external dummy section displacements need not be unique, because they are treated in a special way. These are defined by PC and PR entries, respectively.

[11] The only exception involves control sections with identical

table.[12] 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 occupies 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. Figure 11 gives the function specified by each type. 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 for types related to the current input entry's type.) In certain cases, special preliminary processing is performed to delay or to bypass the CESD search.

CESD processing is shown in "Diagram D3. ESD Record Processing (Generalized)" on page 62 through "Diagram D6. Example of ESD ID Translation" on page 65.

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 because the LD entry has no ESD ID. The loader changes the type to LR in the CESD entry.

Figure 11 (Part 1 of 2). ESD Entry Types and Functions

names. In this case, two entries, one of which is flagged "delete," are kept in the CESD.

[12] The loader clears the translation table after processing each module.

Type	Function	Occurrence	Comments
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 virtual 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.

Figure 11 (Part 2 of 2). ESD Entry Types and Functions

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, it processes no entries for symbols that define module text locations 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 its ID and segment number in the CESD entry. The entries are chained in the order of their linkage editor-assigned addresses.

Besides performing 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, because 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 "Data Areas" on page 83.) 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" on page 32.) For all following NULL entries, processing consists only of making a translation table entry that refers to the CESD entry pointed to by CMNULCHN.

PC
The loader does not perform a CESD search for PC entries because 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 the input contains an object module, 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 was 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 to OFF. If an ER entry with a "never-call" flag matches an ER entry in the CESD, the flag is left set to 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 to search 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 12 on page 26 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.[13] 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" on page 26.) If a matching symbol is found, symbol resolution occurs. (See "Match Processing" on page 34.)

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.
For example, 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. For example, 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 12. Tables Used in the CESD Search

No-Match Processing

When it receives a symbol for the first time, the loader performs processing that depends on the type of the input entry for the symbol. This always includes construction of the CESD entry, which differs by entry type. No-match processing also includes construction of a translation table entry. No-match processing does not trigger construction of a translation table entry for LD entries.

If the user specified the MAP option the loader formats a map entry for each symbol (except ERs). See Figure 48 on page 107 for an example of map output. The loader prints the map entries on the SYSLOUT data set.

Figure 13 summarizes the processing performed for each input entry type.

Input Entry Type	CESD Entry	Translation Table Entry	Map Entry
SD	X	X	X

Figure 13 (Part 1 of 2). No-Match Processing Required for Input Entry Types

[13] 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.

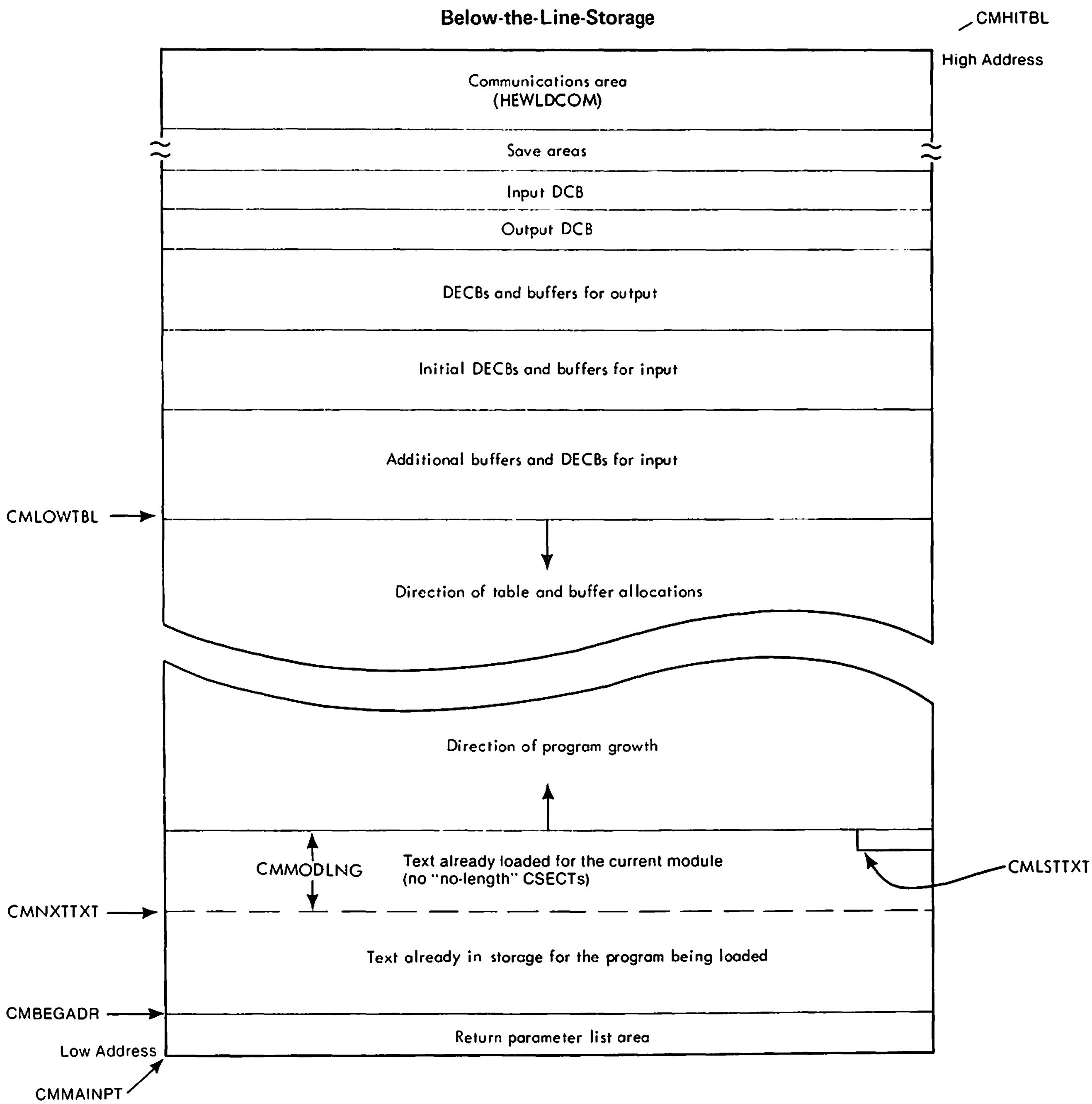
Input Entry Type	CESD Entry	Translation Table Entry	Map Entry
LD	X		X
LR	X	X	X
ER	X	X	
CM	X	X	X
PR	X	X	X

Figure 13 (Part 2 of 2). No-Match Processing Required for Input Entry Types

Note: Because CM and PR entries are assigned addresses during final processing, they are also mapped at that time.

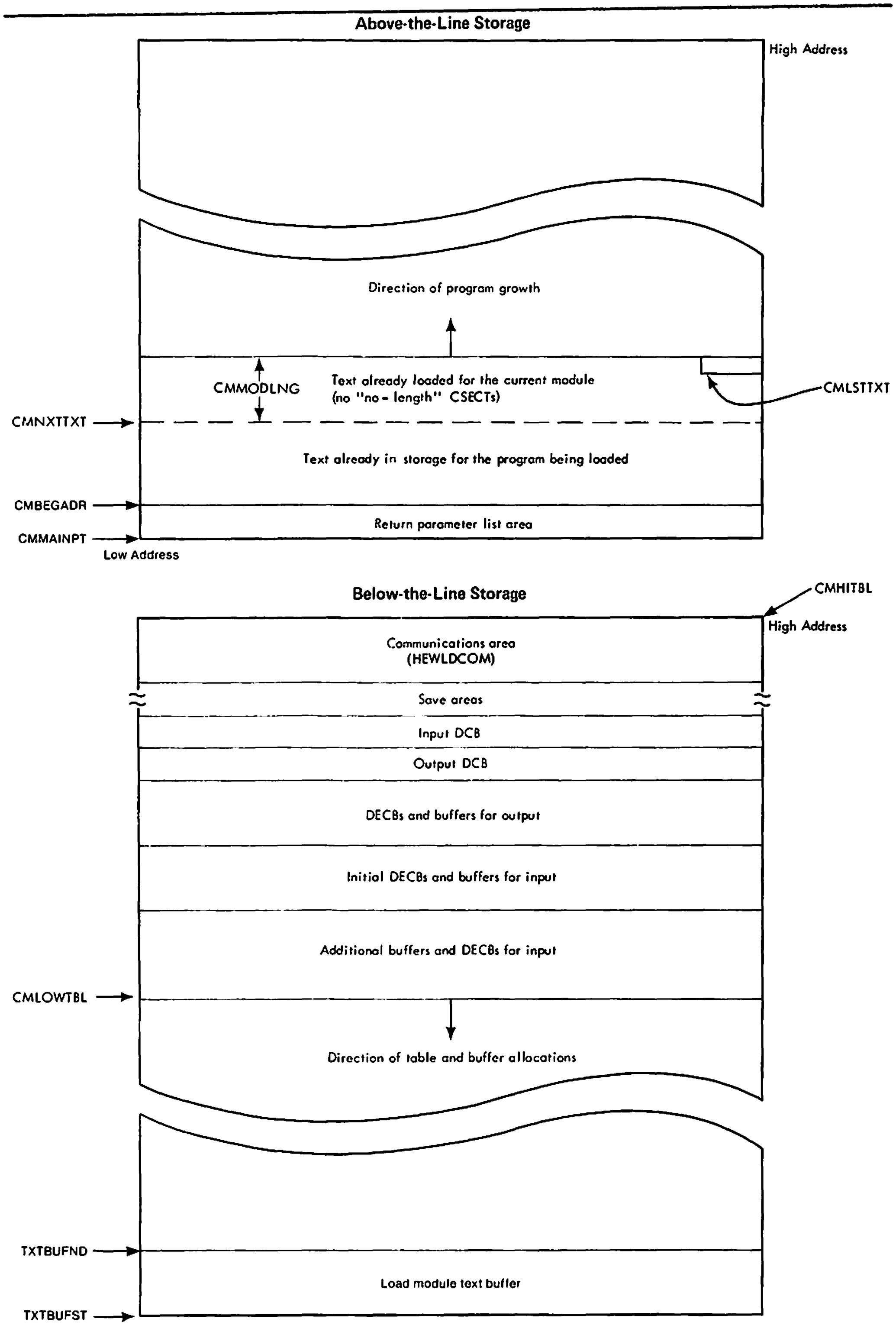
MAKING A CESD ENTRY: For each input entry type, the loader makes a CESD entry. A WX entry type is treated as an ER input entry type with a "weak-call" flag. The loader first obtains the storage required for the entry (22 bytes). Whenever possible, the loader uses storage previously allocated for CESD entries that were later freed. (A CESD entry can be freed as a result of preliminary ESD processing 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 14 on page 28, and Figure 15 on page 29.) If the space required for the entry exceeds available storage, the loading process terminates with an error message. In below-the-line loading, the loader determines this by comparing the pointer for the beginning of the loader's tables (CMLOWTBL) with the overflow pointer for the highest address used for the loaded program's text (CMLSTTXT). In above-the-line loading, the loader compares the pointer for the beginning of the loader's tables with the end address of the load module's text buffer (TXTBUFND).



Notes: CMBEGADR = Beginning address of loaded program
CMHITBL = End address of Loader processing storage below the line
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
CMMAINPT = Beginning address of loaded program space

Figure 14. Storage Allocation (Below-the-Line-Loading)



Notes: TXTBUFST = Beginning address of load module text buffer
TXTBUFND = End address of load module text buffer

Figure 15. Storage Allocation (Above-the-Line Loading)

After obtaining storage for the CESD entry, the loader stores descriptive information in the entry. The specific kind of 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 determines whether an ESD item from a load module contains a segment number or AMODE/RMODE data. Segment numbers are ignored; AMODE/RMODE data is verified and copied to the CESD entry. The loader treats ESD items from an object module as having AMODE/RMODE data; that data is verified and copied to the CESD entry.

The loader next determines whether the RMODE for the loaded program was specified by a user option, or is to be taken from the first CSECT loaded. If the RMODE was specified by user option, then the obtaining of storage for the loaded program (either above or below the line) and the initialization of the communication area was already appropriately done; allocation of following storage is bypassed. However, if the RMODE is to be taken from the first CSECT loaded, and if the current ESD item represents the first CSECT loaded, and if the RMODE for that CSECT is ANY, then the storage for the loaded program has not been obtained and the communication area was not properly initialized.

If the RMODE for the first CSECT loaded is 24, loading occurs below the line in the storage already obtained and according to the initialization of the communication area already done; allocation of following storage is bypassed.

The loader obtains required above-the-line storage from the supervisor module via the GETMAIN macro instruction. The request is conditional and variable. The minimum and maximum values are the same as those used in obtaining below-the-line storage. If the supervisor module does not return storage, loading occurs below the line in the storage already obtained and according to the initialization of the communication area already done. If the supervisor module returns virtual storage space, the loader initializes values in the communication area required for above-the-line loading.

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. If the CSECTS are passed through text records, the loader assigned address of the first CSECT is found at CMNXTTXT. If the CSECTS are pointed to by MOD records, the loader-assigned address of the first CSECT is found at location CMCORE1.) For CSECTs pointed to by MOD records, the resulting address is stored in the CESD entry for the SD, assigned by the loader as the address of the CSECT. For CSECTs passed through text records, however, the resulting address is compared to the overflow pointer—the beginning address of the loader tables (CMLOWTBL) in below-the-line loading, or the highest address of the loaded program area (ATLHIADR) in above-the-line loading. If there is no more unused storage, the loading process terminates and sends an error message. Otherwise, the resulting address is stored in the 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 loading is taking place above the line, the loader verifies that each CSECT loaded (that is, added to the CESD as a CSECT to be kept) has an RMODE of ANY. If a CSECT having an RMODE of 24 is encountered, the loader indicates that the redrive condition exists (see "Redrive" on page 16).

If the option to specify the entry point name for the loaded program was used, the loader determines whether the SD with that name was already received. If not, the loader compares the specified entry point 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 a specification of an SD entry, the loader determines whether the CSECT length specified in the input entry equals zero. 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 set to ON, the 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 it processes 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 terminates and sends 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 is also 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 address constants referring to it should be adjusted.

CM

To make a CM entry, the loader uses two separately obtained 22-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 22-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 3 bytes in the entry to be used as a pointer to related ERs, and sets a pointer 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 in the order they were 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" indicator in the CESD entry. If the input ER entry is marked "weak call," the loader similarly sets the "weak-call" indication. The loader then 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.[14] 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, and so forth. (Because the initial 4 bytes are used for indexing purposes, the first extent contains only 31 translation table entries.) Thus, the position designated for creation of a particular translation table entry depends on the ESD ID of the associated input entry.

Figure 16 shows 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. (See "Diagram D6. Example of ESD ID Translation" on page 65, 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 one [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 thirty-two 4-byte entries, and 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 in the loader's table and buffer space. If not enough loader processing storage remains to make the allocation, loading terminates and sends an error message.

After the extent allocation completes, the loader clears the extent. The loader then 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 of the translation table entry to one (1). (This setting indicates absolute relocation.)

Figure 17 on page 34 shows the overall relationship of tables used in ESD processing.

---

[14] For each input module, the loader reinitializes the translation table.

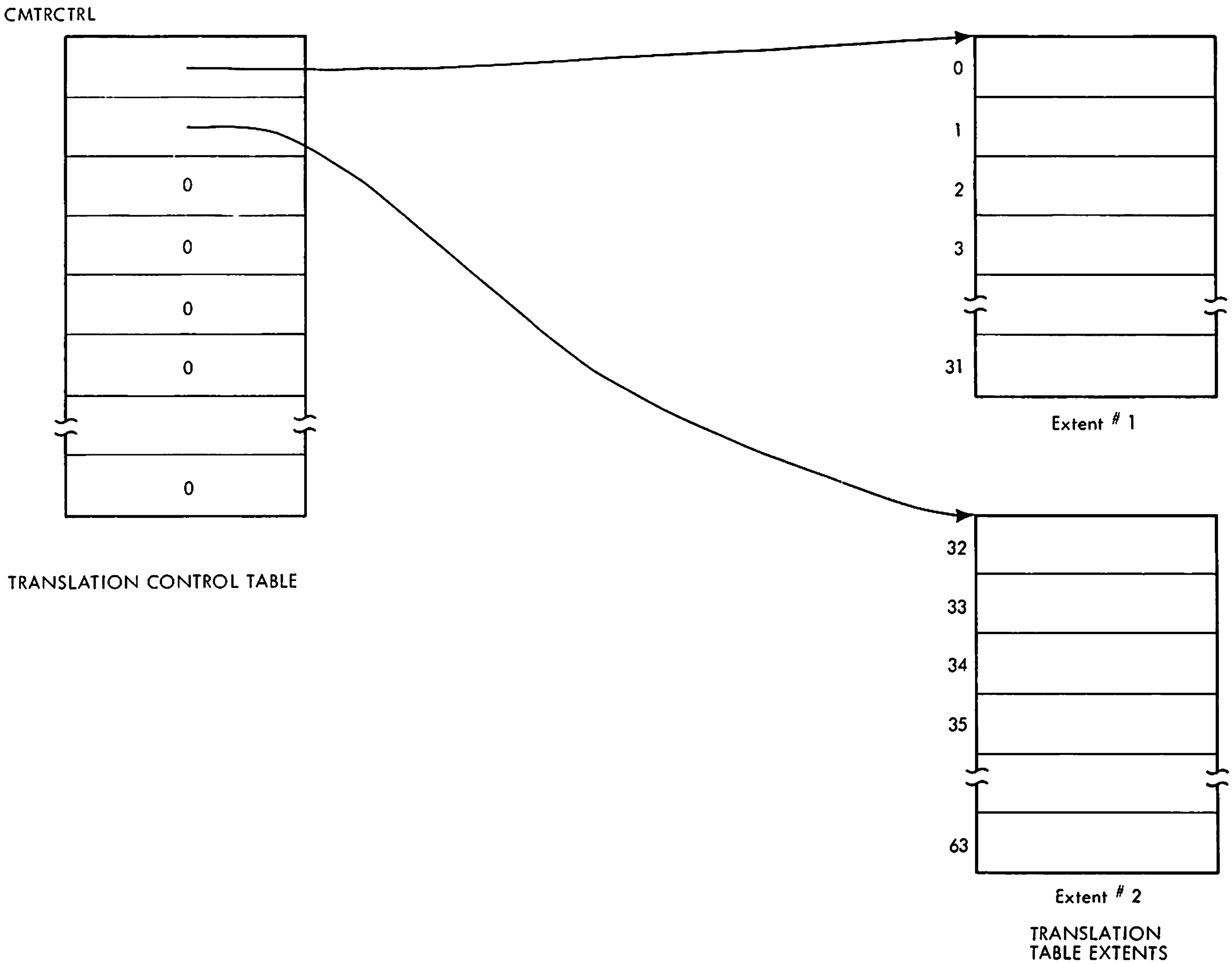


Figure 16. Translation Control Table and Translation Table

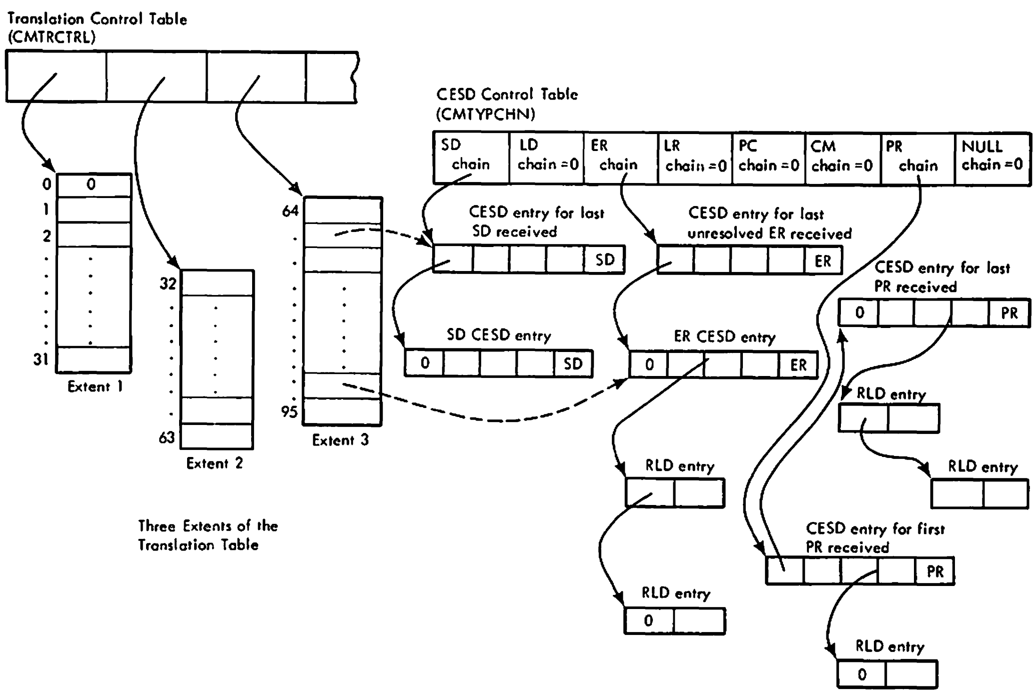


Figure 17. Overall Relationship of Tables

Match Processing

If the loader finds a match for an input symbol during the CESD search, it then performs symbol resolution. Through resolution, the loader ensures that each named location within the text of the loaded program has a unique symbol.[15] Also, all references to a named location are set to the correct loader-assigned virtual 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 to retain 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.

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

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

Figure 18 gives a summary of symbol resolution.

Input Type	CESD Type	Result
SD	ER SD CM LR	SD SD SD LR
CM	CM ER SD LR	CM CM SD LR[1]
LD/LR	ER LR SD CM	LR LR SD[2] CM[2]
ER	SD ER LR CM	SD ER LR CM

Figure 18. Symbol Resolution

Notes to Figure 18:

[1] Match results in an error.

[2] 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 consists of:

- Chaining the entry to other SDs
- Updating the cumulative length of the loaded program
- Determining whether the ER entry is the loaded program's entry point name
- Mapping the entry
- 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 SD 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 original 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 subtracting 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 was 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 entry. Because 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 the length of the common area from the extended portion. For the SD entry, the loader retains the one with the greater length between the first length and the length specified in the input SD. To change the CM entry to an SD entry, 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 length between the extended portion of the CESD entry and the length specified in the input CM. This greater length is retained in the CESD entry. The loader stores the new input address in the extended portion of the CM entry. A translation table entry is also made.

CESD type is ER
To change an ER entry to a CM, the loader obtains a 22-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 indicator 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 for both entries.

INPUT ENTRY TYPE IS LD OR LR: With one exception, LD and LR entries are processed in the same way. The difference is that, because 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 entry. The loader assigns a virtual storage address for the symbol by adding the relocation constant from the related SD entry to the relative address in the input LR entry. 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 entry, 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 entry 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 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 also sets the "never-call" indicator in the CESD entry. If the "delink" indicator is set to ON, the loader sets the indicator set to 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 to OFF. If both ERs are marked "weak call," the flag remains set to 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 virtual-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 (below-the-line loading) or into the load module text buffer (above-the-line loading).

Processing Object Module Text (HEWLTXT)

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 ensuring 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 as the one used prior to allocating a translation table extent. (See "Making a Translation Table Entry" on page 32.)

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 one (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. (Object module text processing is shown in "Diagram D7. Object Module Text Processing" on page 66.)

(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 was received for this module. If the "no length" indicator in the CESD entry was set to ON, an indicator is set in the communication area to show that text was received for a "no length" CSECT. The loader then calculates the address for this text in the loaded program's virtual-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 virtual-storage address. The resulting end address for the text is compared to the overflow pointer (the beginning address of the loader tables [CMLOWTBL] in below-the-line loading) or the highest address of the loaded program area (ATLHIADR) in above-the-line loading. If the text would overlap, loading terminates abnormally.

If there exists 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).

Processing Preloaded Text (HEWLMOD)

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 (HEWLIDEN) 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.

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

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. Load Module Text Processing (Below-the-Line Loading)" on page 67, and in "Diagram D9. Load Module Text Processing (Above-the Line Loading)" on page 68.

PROCESSING THE ID/LENGTH LIST: The loader obtains each ID in turn from the length 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 invalid 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 process). 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 which are 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 was saved. (Only one relocation constant per control record is used, because 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: After processing all IDs in the ID/length list, the loader prepares to read the text into storage, either directly into the load program's storage area in below-the-line loading, or into the load module text buffer in above-the-line loading. 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 19 on page 41.)
- Obtains the actual read count by subtracting the sum of the lengths of consecutive, deleted CSECTs at the end of the text record from the text length in the control record.
- 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 terminates.
- Determines by examining the control record's type whether the text record is the last record in the module.

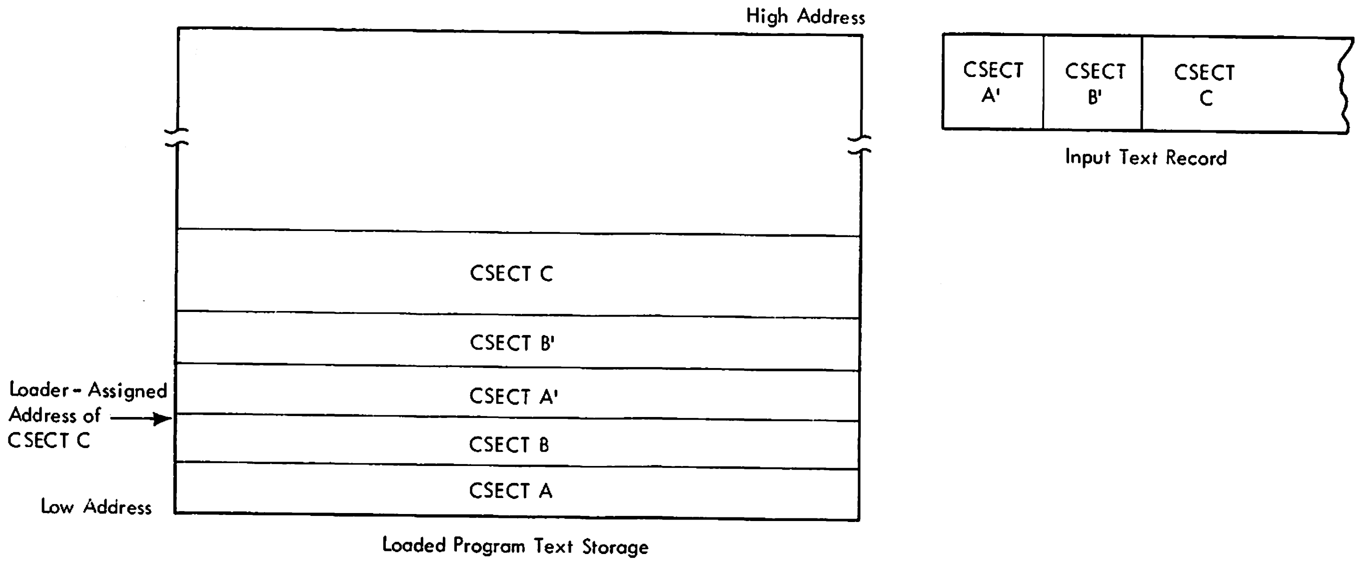
If the record is not the last one, the loader determines whether any CSECTs from the record are to be deleted. If not, the text record and the following control record are read. (The control record is read into the RLD buffer.)

If the text record is the last one 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 end-of-module processing.

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 either into the loaded program's storage area in below-the-line loading, or into the load module text buffer in above-the-line loading.

The loader ensures 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 it finds one that is not marked "delete." The loader determines whether the related CSECT is in the correct place by comparing its current address to the loader-assigned address found in the CESD entry. If the text is correctly placed, the loader continues to translate IDs.



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 19. Loading the Text from a Load Module Record

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 loader updates the address of the current CSECT with the length of the current CSECT in order to get the current address of the next CSECT. When all CSECTs are in the correct location, the loader continues processing the module with the next record.

In above-the-line loading, the loader determines whether any CSECTs that were read into the load module text buffer are to be deleted. If not, the entire text record is moved into the loaded program's storage area above the line.

Next, the loader determines whether a control record was read at the same time as was 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 (HEWLRLD)

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 address constants is performed as the RLD is encountered, unless the referenced CSECT is not in virtual storage.)

RLD record processing is shown in "Diagram D10. RLD Record Processing" on page 69.

To build the RLD table, the loader tests the R and P pointers of the entries in an RLD record for validity.[16] 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.

Because the pointers act as IDs, they are valid if translation yields the address for the ID to a CESD entry. If an invalid ID is received, the loader issues an error message and continues RLD record processing by going to 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 ensuring 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 the R pointer refers to an ER entry, 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 it reaches the end of the record. 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

---

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

directly. If the pointers for the next entry are different from the current entry, the loader performs the validity check.

RELOCATING ADDRESS CONSTANTS (HEWLERTN)

Address constant relocation is the replacement of an address constant in the text of the loaded program with the actual virtual-storage address. Whenever possible, the loader relocates address constants as it encounters their RLD entries.

The loader processes three types of relocatable address constants:

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

In general, the virtual 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.[17] The relative relocation constant is the subtracted value between the loader-assigned address and the input address of the referenced location. The absolute relocation constant is simply the loader-assigned virtual-storage address of the referenced location. Figure 20 on page 44 relates the types of relocation constants and address constants, to the types of relocation.

---

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

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.
Relative Relocation	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.

Figure 20. Relocation of Address Constants

Note to Figure 20:

Absolute relocation constant = loader-assigned address
Relative relocation constant = loader-assigned address minus the input address

When the loader resolves a CESD entry (for example, 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 ensures that the address constant is not an invalid 2-byte address constant. (Two-byte address constants can only be used to define external DSECT displacements.) If the address constant is invalid, the loader issues an error message and continues loading the program. Otherwise, the loader moves the address constant 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 address constant 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 address constant 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 address constant. The result is a reference to a displacement in the common area or input module. When these entries are resolved (that is, 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.

The loader, during the relocation of an address constant, checks for an attempt to provide a 31-bit address (that is, an address in above-the-line storage) in a three-byte (24-bit) address constant. If found, the address constant is not relocated, and an error message is issued.

Also, during relocation of four-byte V-type address constants, the loader preserves the high-order bit from the unresolved address constant in the resolved address constant.

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 was 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.[18] If the END card of a "no-length" CSECT does not specify a length, and text was received for the CSECT, the loader issues an error mess age. (In this case, the length of the text is used.)

The loader determines whether the loaded program's entry point name or address was already received. If so, the loader does not process the END card for entry point. If not, the loader searches the END card for an ID to use 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 zero (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 with that name in the CESD, the loader uses the specified address as the program entry point address.

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

End-of-Module Processing

At the end of module point for a load or object module, the loader initializes the next input module for processing. If text was 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 was exceeded. If so, an error message is issued, and loading terminates. If not, the loader clears the translation table and the module length counter (CMMODLNG). All flags except the END and LIB flags are set to OFF. The loader either begins processing another module from SYSLIN or, if end of file on SYSLIN is recognized, processes any secondary input.

SECONDARY INPUT PROCESSING (HEWACALL)

After the loader processes 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 attempting to resolve them from SYSLIB.

Secondary input processing is shown in "Diagram E1. Secondary Input Processing" on page 70.

RESOLVING ERS FROM THE LINK PACK AREA

The loader obtains the address of the link pack area directory search routine from the communication vector table (CVT). It then searches the ER chain for an ER that is not marked "never call" or "weak call." (A) When one is found, the name in the ER is passed to the LPA directory search routine. If the directory search routine does not find a match for the name, the loader searches for the next ER that is not marked "never call" or "weak call."

If the directory search routine finds a match for the name, the loader puts the entry point in the CESD entry and changes the entry's type to SD. The loader then takes the entry off of 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 that are chained to the CESD entry.

The loader then searches for the next ER that is not marked "never call" or "weak call."

This search repeats until the entire ER chain has been processed.

If there remain unresolved ERs after resolution of the link pack area, the loader performs library call processing. Otherwise, the loader performs final processing for the loaded program. (See "Final Processing for the Loaded Program" on page 48.)

RESOLVING ERS FROM THE SYSLIB DATA SET

(A) Before resolving ERs from the SYSLIB data set, the loader checks whether an open SYSLIB data set was 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 was 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 was not passed, a SYSLIB DCB is initialized and opened.[19]

(B) If the loader determines that an open SYSLIB data set was passed, it constructs two lists used for BLDL information in the available storage. In below-the-line loading, the available storage is defined by CMLOWTBL (the lowest address used by the loader tables and buffers) and CMLSTTXT (the highest address used by the loaded program's text). In above-the-line loading, the available storage is defined by CMLOWTBL and TXTBUFND (the highest address used by the load module text buffer). 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 to the ER entries. Figure 21 on page 48 shows this relationship.

Before constructing the lists, the loader determines the maximum possible number of entries 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 that 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.

Because SYSLIB contains only load or object modules, processing for each located module 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. It then performs final processing for the loaded program. (If during processing of one of these modules a program size error occurs, the loading procedure terminates and produces an error message.)

---

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

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 displacement 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
- Identifies the loaded program to the system, unless the processing portion of the loader was directly invoked by the name HEWLOADR
- Writes out the diagnostic message dictionary.

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 that may be overlapped with common areas in below-the-line loading.) The address contained in CMNXTTXT rounded to doubleword value is the address used. The loader ensures 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

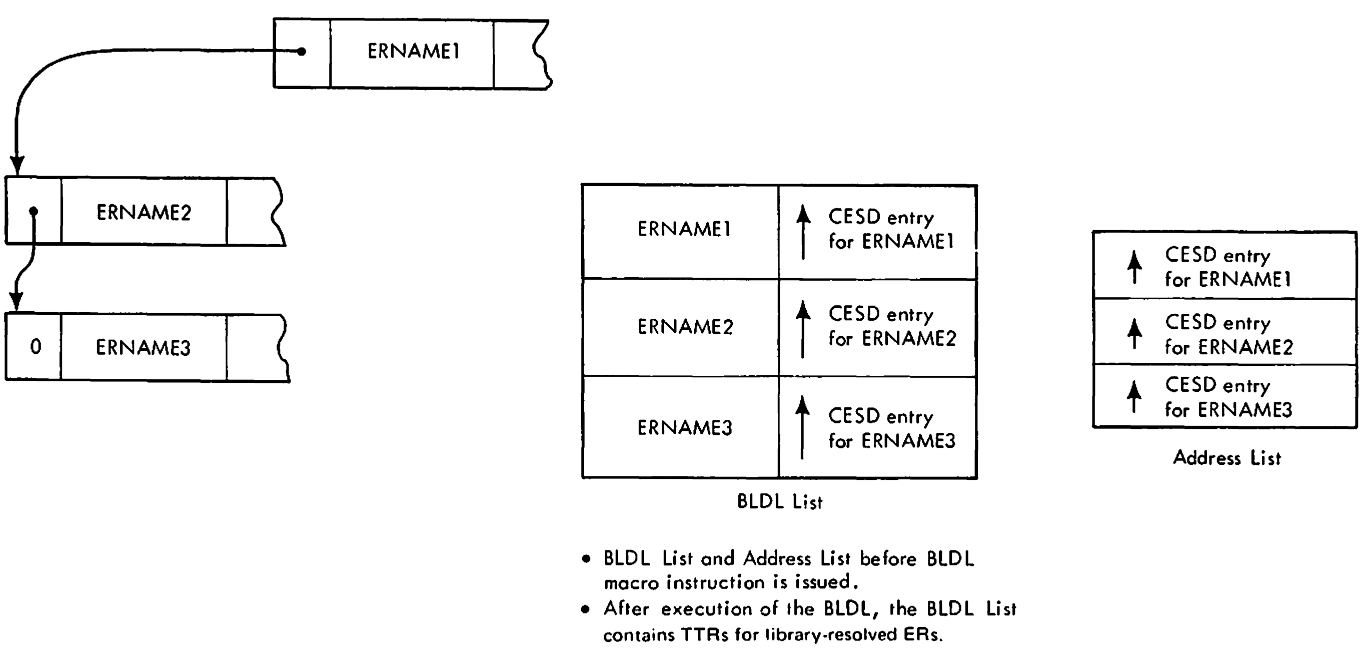


Figure 21. BLDL List and Address List

CMNXTTXT value. (If there is not enough storage, an error message is issued and loading terminates.) Next, if the MAP option was chosen, the common area is mapped. Finally, the loader relocates the address constants that refer to the current "common" definition. (The address constants are relocated by processing the RLDs chained from the current CESD CM entry.)

After processing all the CM entries in the CESD, the loader assigns addresses to 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. The individual DSECTs are displacements within the area.)

For each entry on the chain, the loader subtracts the alignment factor from hexadecimal "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 is zero [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 address constants 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.[20] (If NCAL was specified, there is no CXD chain pointer in CMCXDPT.)

ISSUING UNRESOLVED ER MESSAGES

For all ERs remaining in the CESD that 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 processing the loaded program, the loader checks to determine whether the entry point name and address were received. This is determined by testing the program flag field (CMPRMFLG). Processing for possible conditions is as follows:

- Entry point name and address both received. No further entry point processing is required.
- Only entry point name received. If the entry point name was specified by the EP= parameter but no address for the name was 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.

---

[20] See <u>Assembler Language</u> for the use of external DSECTs and the CXD statement.

- Only entry point address 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 THE LOADED PROGRAM

If program loading is successful, the loader prepares to identify the program to the control system.[21] A parameter list is constructed to pass the program name, addressing mode, 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 terminates.

The loader initializes the parameter list with the program name, addressing mode, 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 that 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 terminates. Otherwise, a flag is set in the communication area, indicating that the program was identified.

END OF LOADING

After all processing for the loaded program completes, the loader processing portion of the loaded program performs termination processing and then passes control to the loader control portion. The control portion then attempts to execute the loaded program.

LOADER PROCESSING TERMINATION

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

Next the loader ensures that all diagnostic data was written to SYSLOUT, and then closes both the output and the current input data sets.[22]

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

---

[21] This processing is performed only when invoking the processing portion (either directly or by the control portion of the loader) by the name HEWLOAD.

[22] 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.

If the loaded program is loaded below the line and if it is to be executed, the loader calls the page services processor, via the PGSER macro instruction, to reset the PGTBELOW flags in the page table entries reflecting the loaded program storage. The PGTBELOW flags were possibly set via the reading of load module text directly into the loaded program's storage. Resetting the flags allows the pages of the loaded program's storage to be backed above the 16-megabyte real storage line on subsequent page-ins.

Finally, the loader issues a FREEMAIN macro instruction for all processing storage not assigned to either the loaded program or to the condensed symbol table. (If the completion code for loading is greater than four (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 execution of the LOAD and DELETE macro instructions.) The loader then returns control to the control portion.

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, through the execution of an ATTACH macro instruction, passes the user's parameter list to the loaded program for its execution.

After the program's execution, the loader control portion issues a DELETE macro instruction for the loaded program, frees its processing storage, and returns to the scheduler.

OPERATION DIAGRAMS

The following diagrams show the flow of data through the loader. Use them with the descriptions given previously in this section to give an integrated picture of the loader logic. Each diagram has an alphameric identification (for example, A1). Within each diagram, specific points of reference have alphabetic labels. When the description at the beginning of this section discusses a function, it refers to the operation diagram as a whole, and to the specific labeled references where appropriate. For example, the description of initialization refers to Diagram B1. Within the discussion, reference (B) refers to point (B) in Diagram B1.

The symbols used in the diagrams are shown in the following chart.

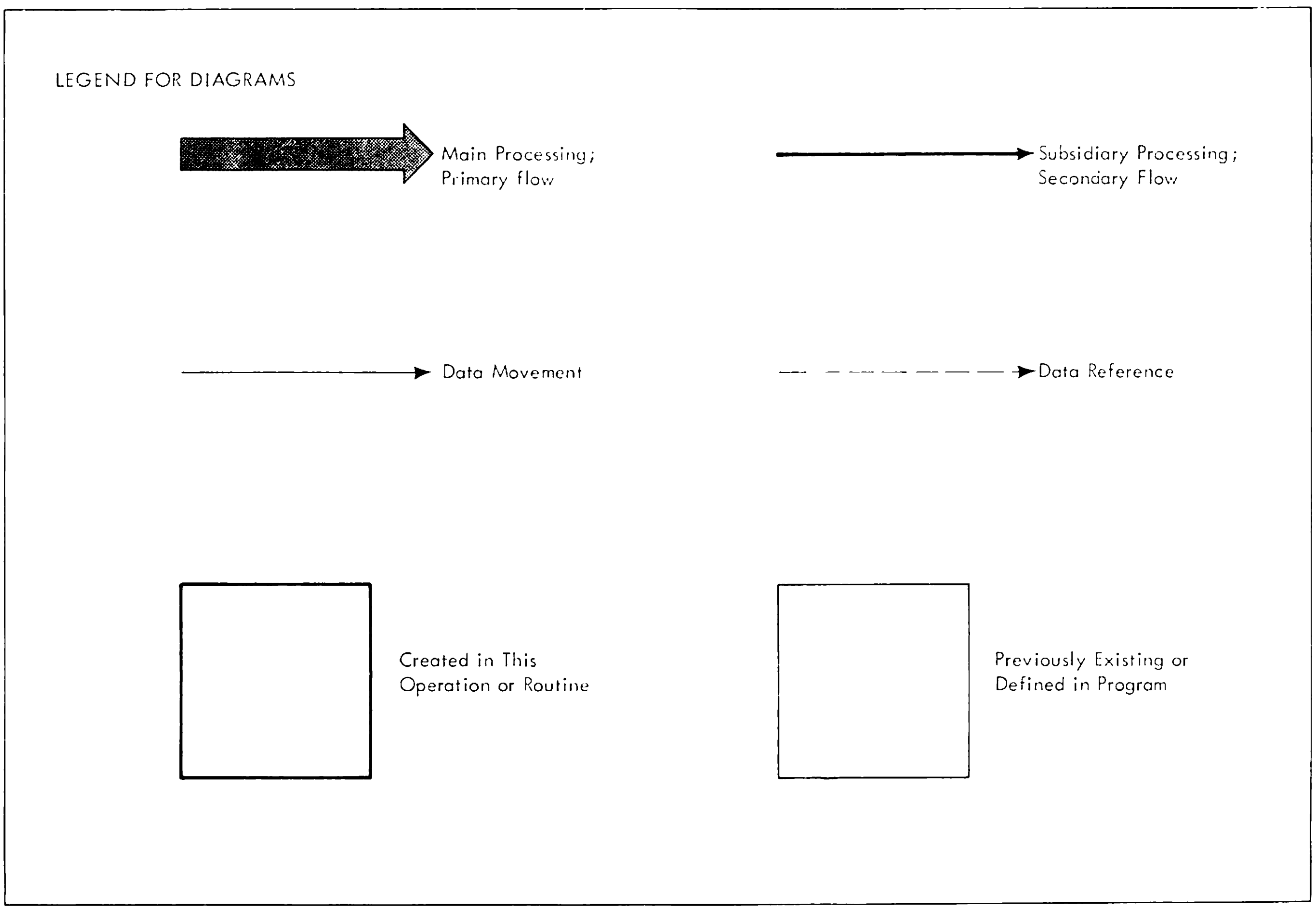


DIAGRAM A0. OVERALL LOADER OPERATION (BELOW-THE-LINE LOADING)

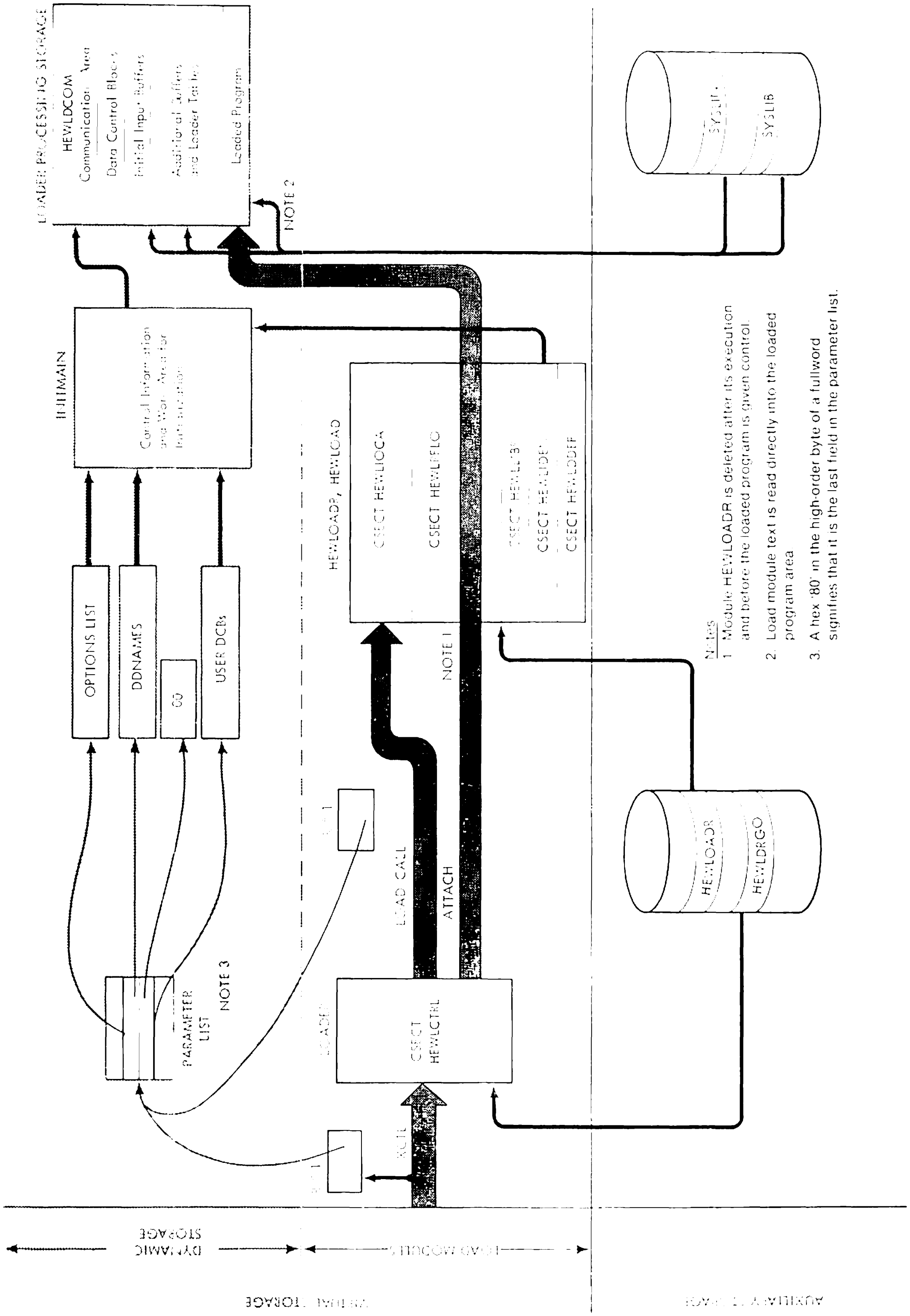


DIAGRAM A1. OVERALL LOADER OPERATION (ABOVE-THE-LINE LOADING)

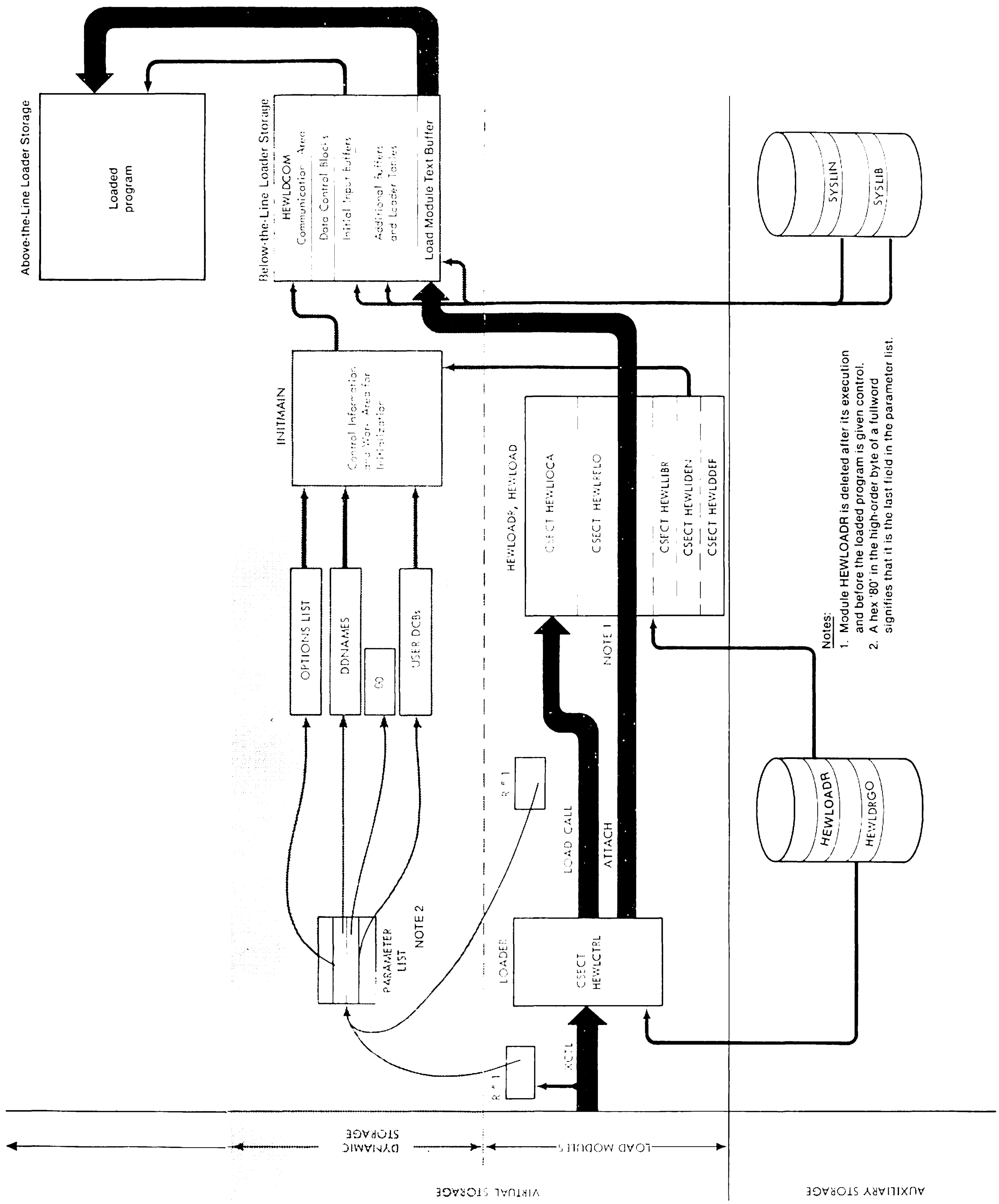
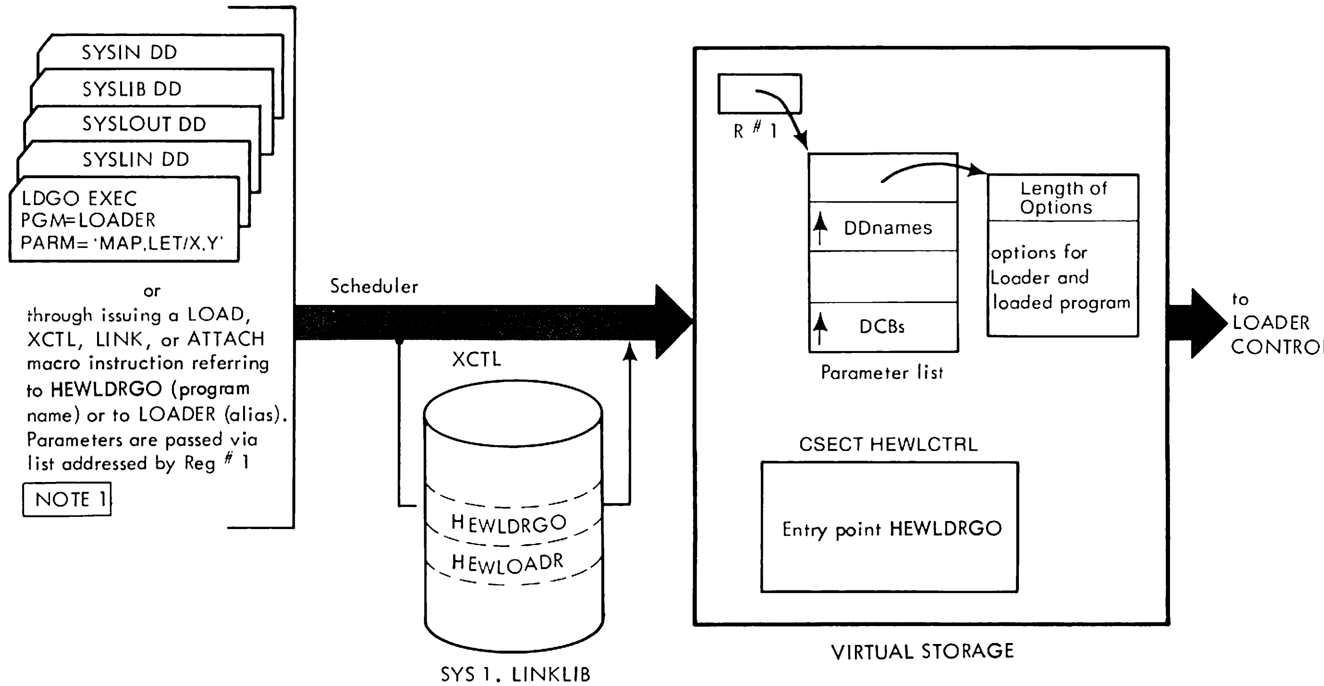


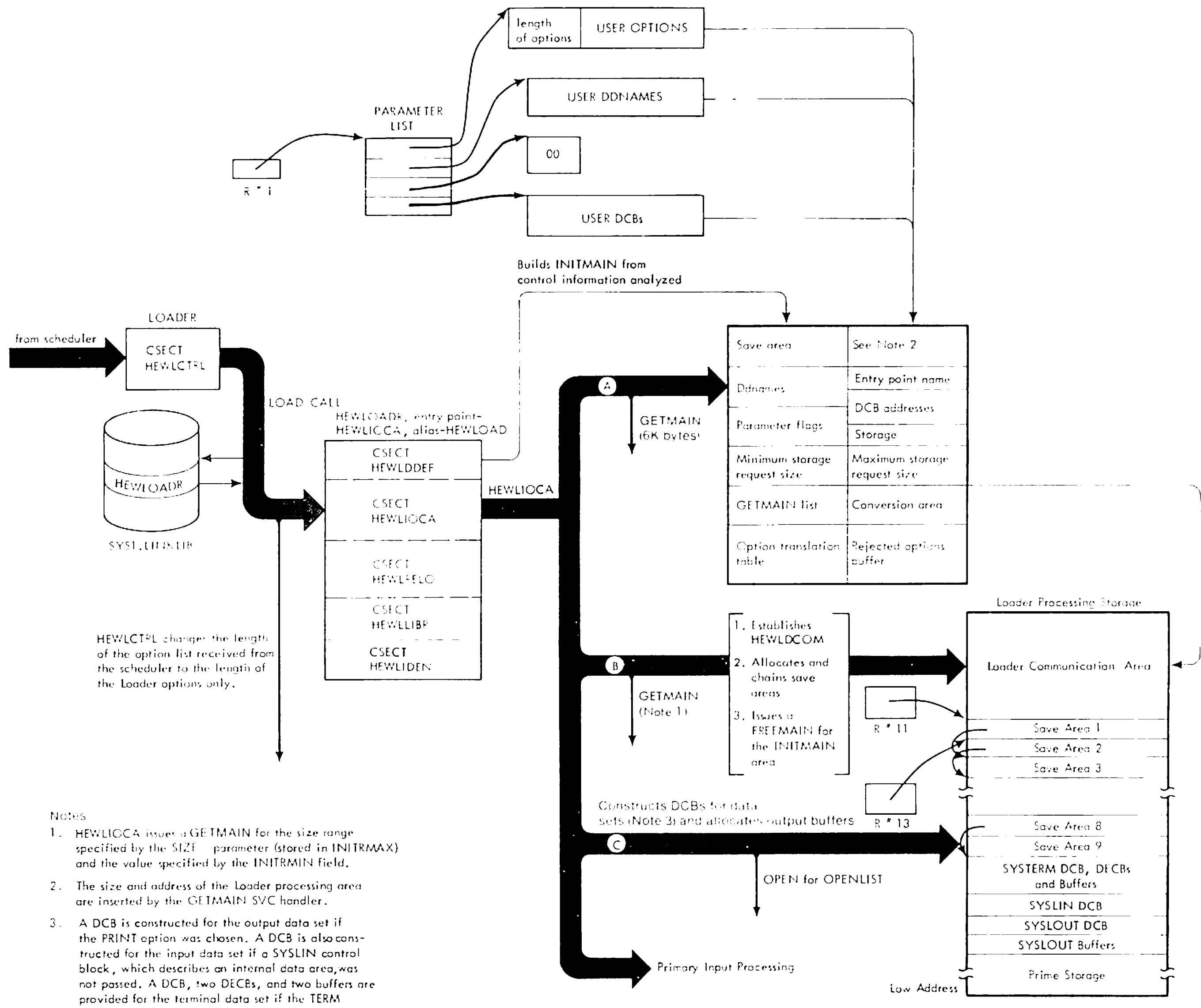
DIAGRAM A2. LOADER INVOCATION



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 HEWLOADR (for loading without identification) or to HEWLOAD (for loading with identification).

DIAGRAM B1. LOADER/SCHEDULER INTERFACE AND INITIALIZATION



Notes

1. HEWLIOCA issues a GETMAIN for the size range specified by the SIZE parameter (stored in INITRMAX) and the value specified by the INITRMIN field.
2. The size and address of the Loader processing area are inserted by the GETMAIN SVC handler.
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

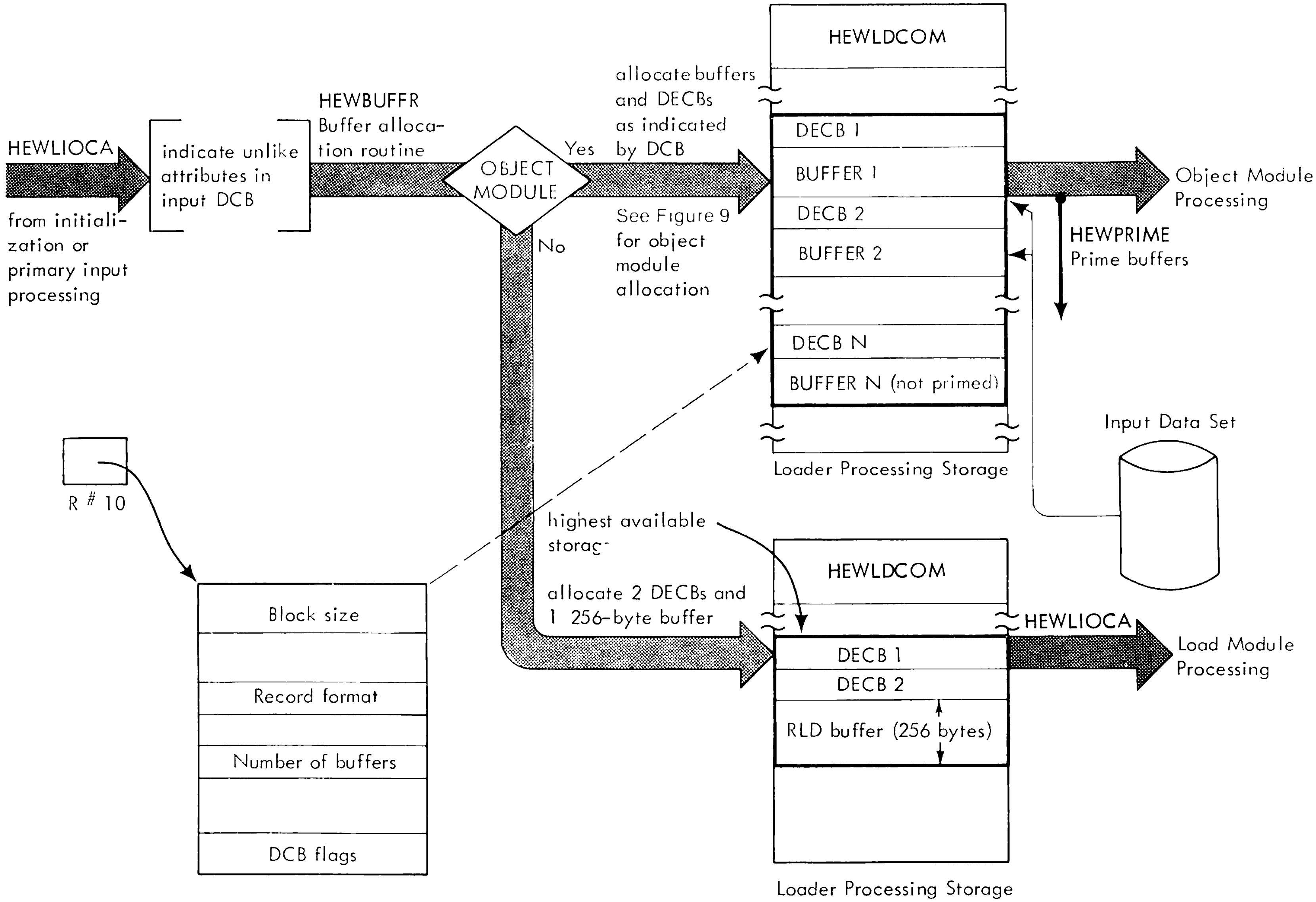


DIAGRAM D1. OBJECT MODULE PROCESSING

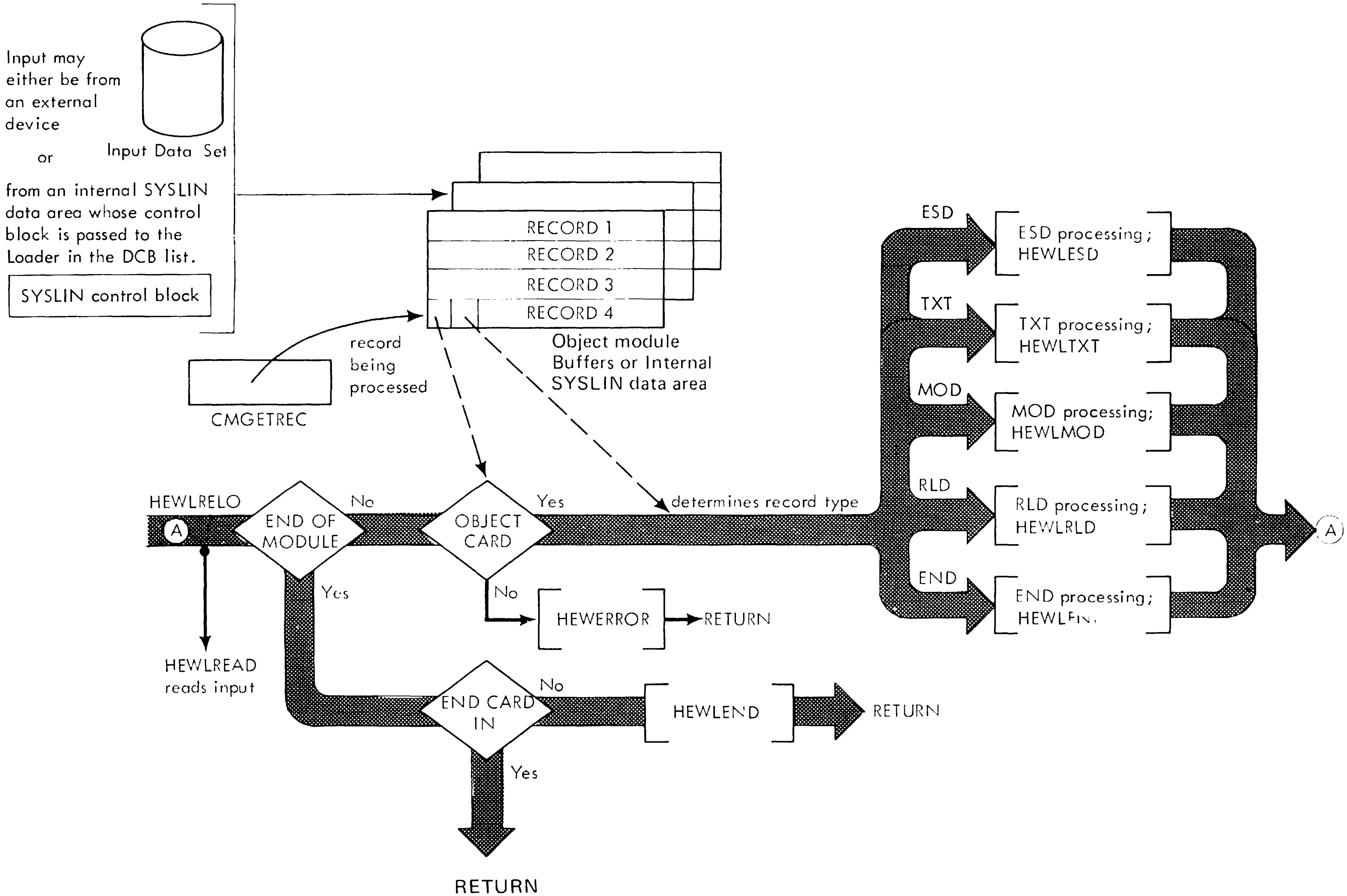


DIAGRAM D2. LOAD MODULE PROCESSING

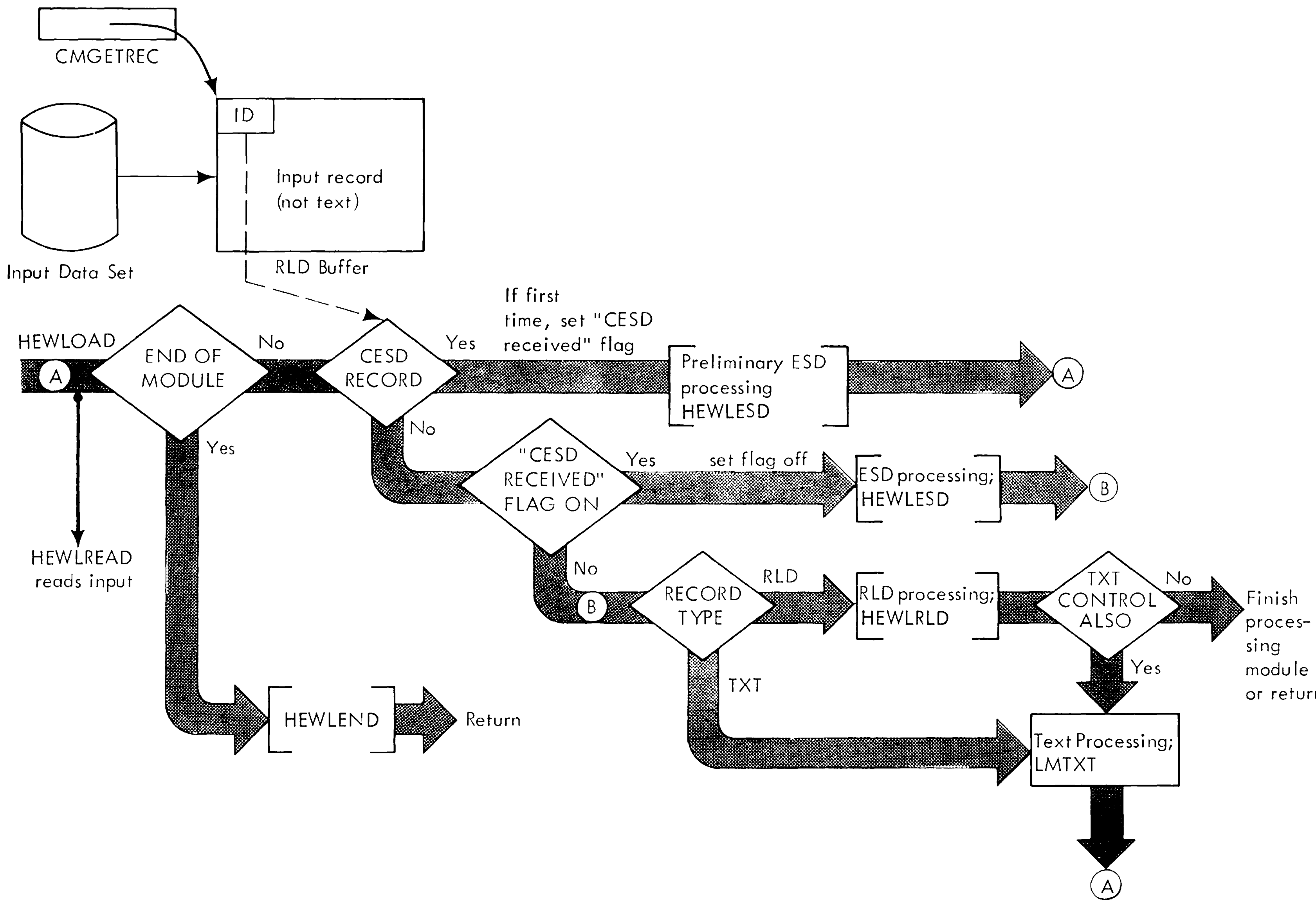
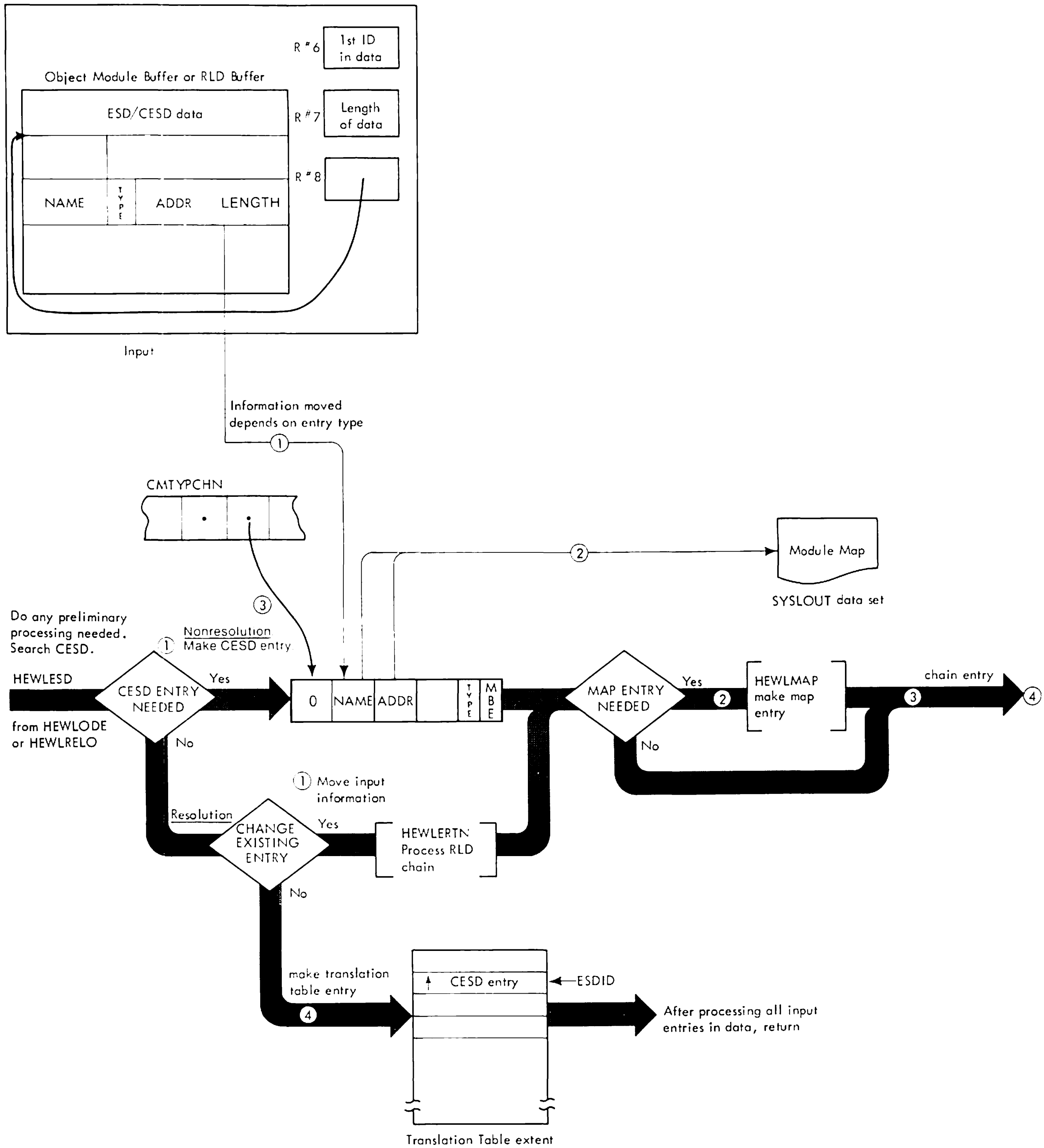


DIAGRAM D3. ESD RECORD PROCESSING (GENERALIZED)



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 OF SD-SECTION DEFINITION (HEWLESD)

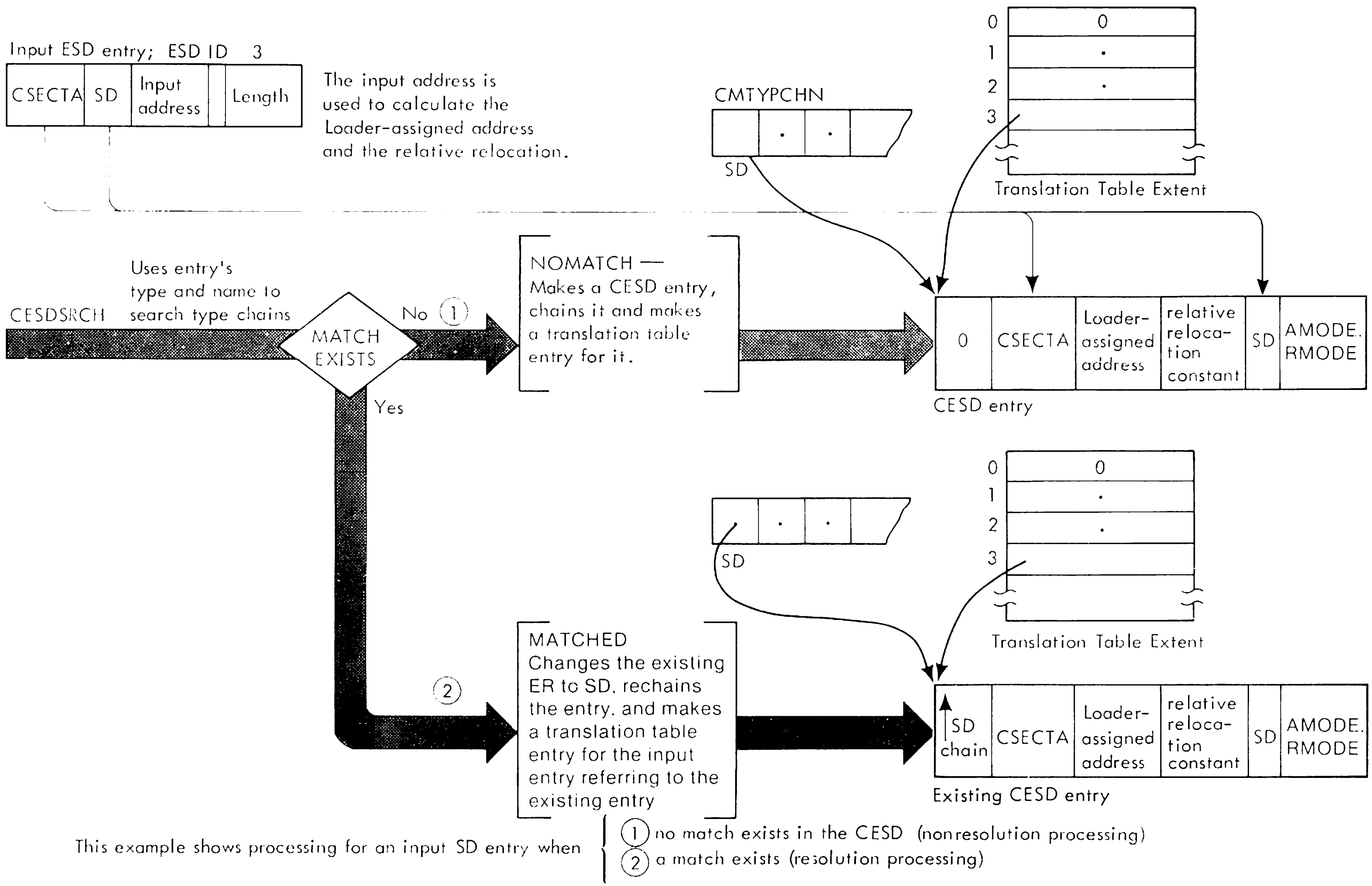
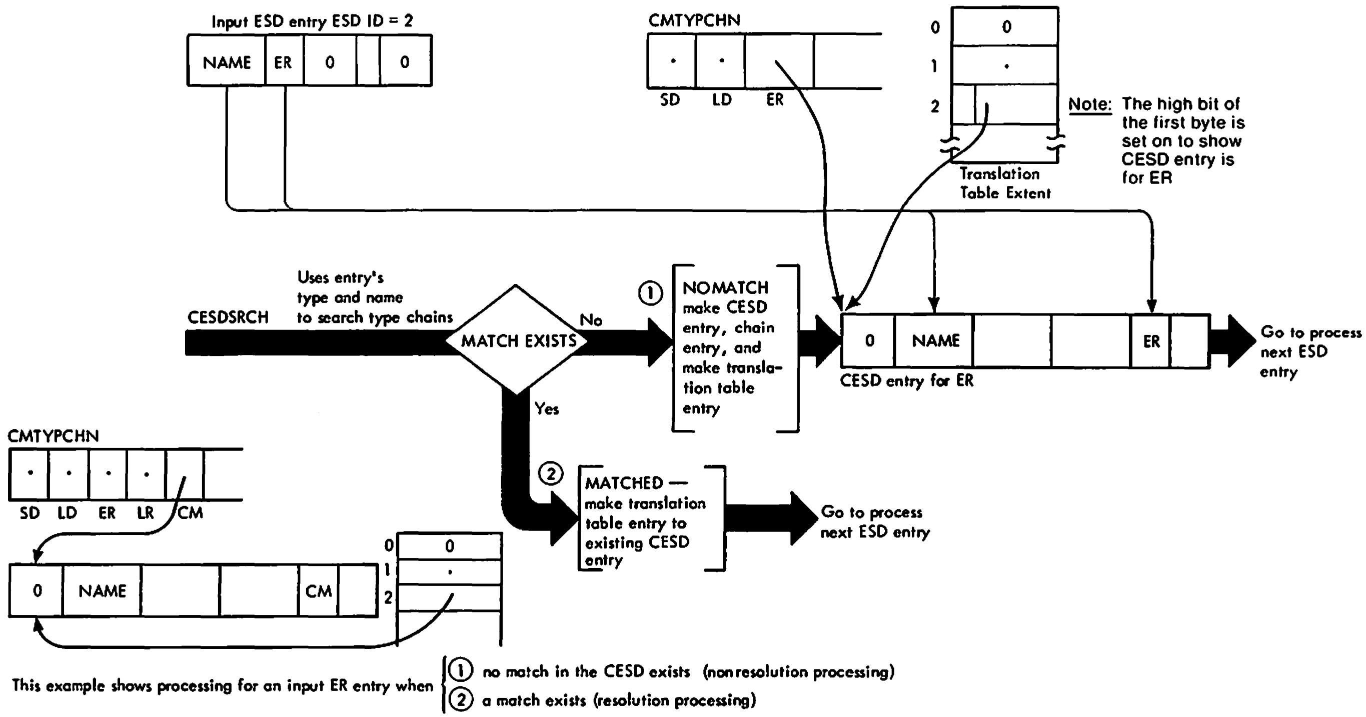
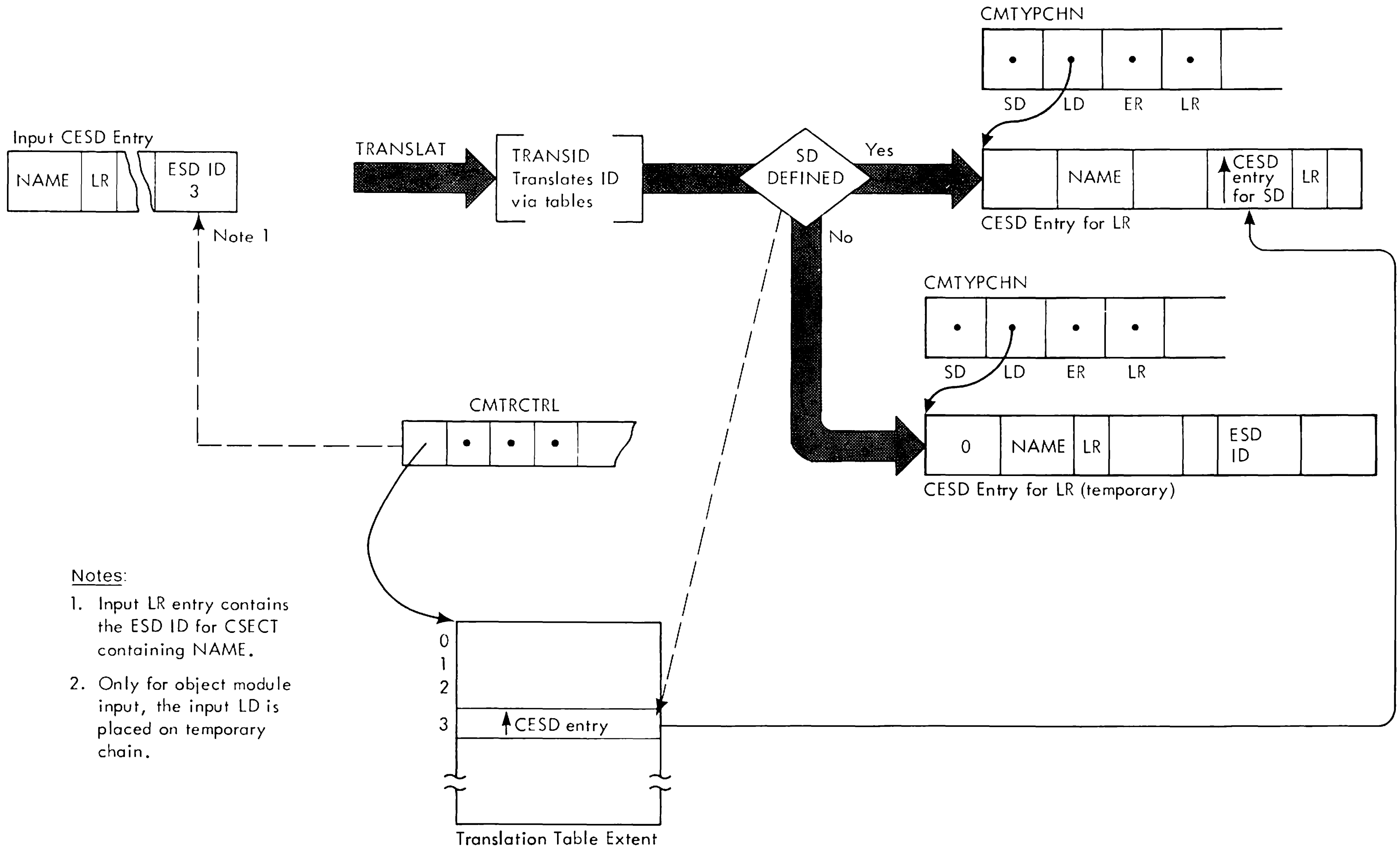


DIAGRAM D5. EXAMPLE OF INPUT ESD OF ER-EXTERNAL REFERENCE PROCESSING (HEWLESD)





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

DIAGRAM D7. OBJECT MODULE TEXT PROCESSING

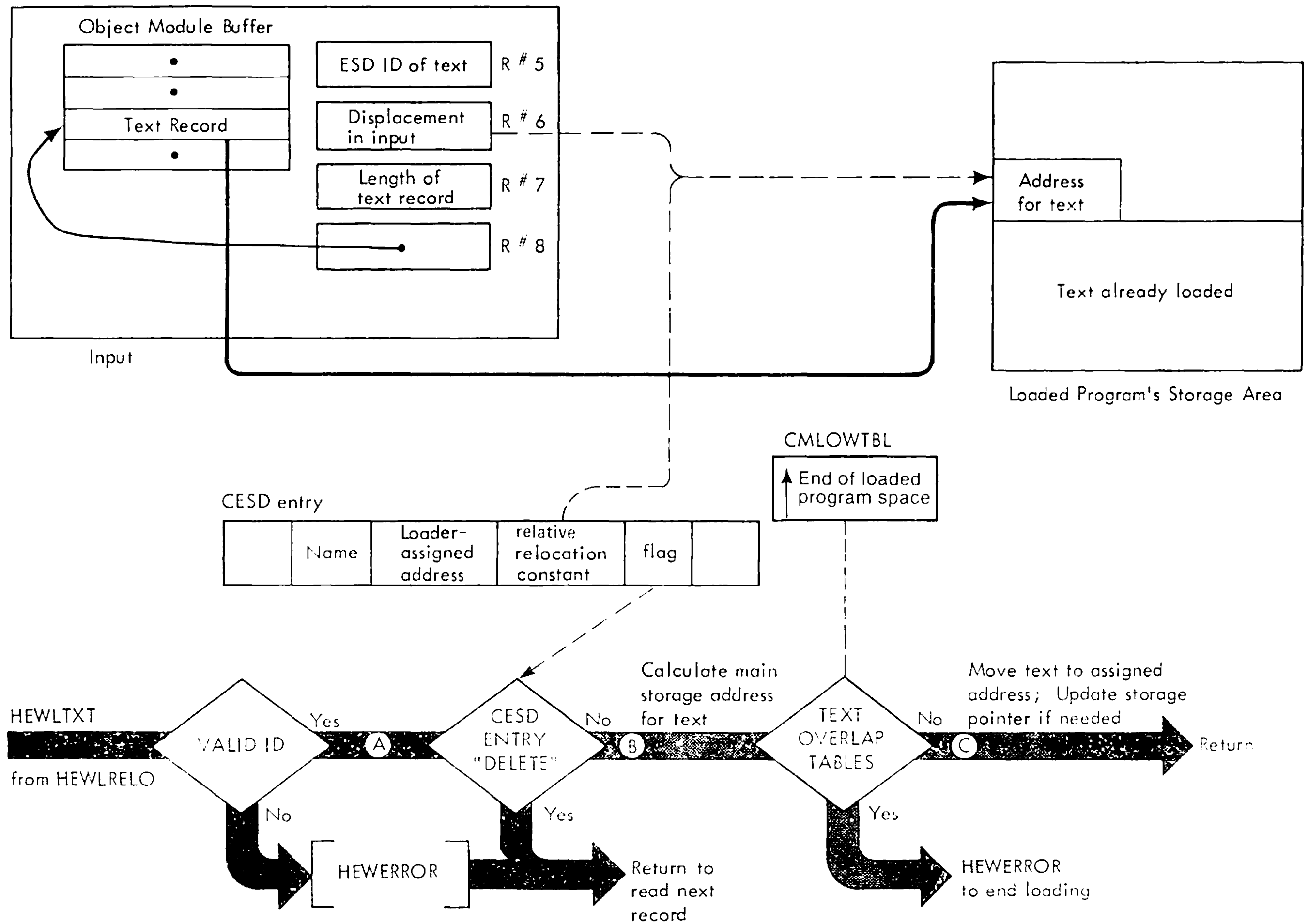
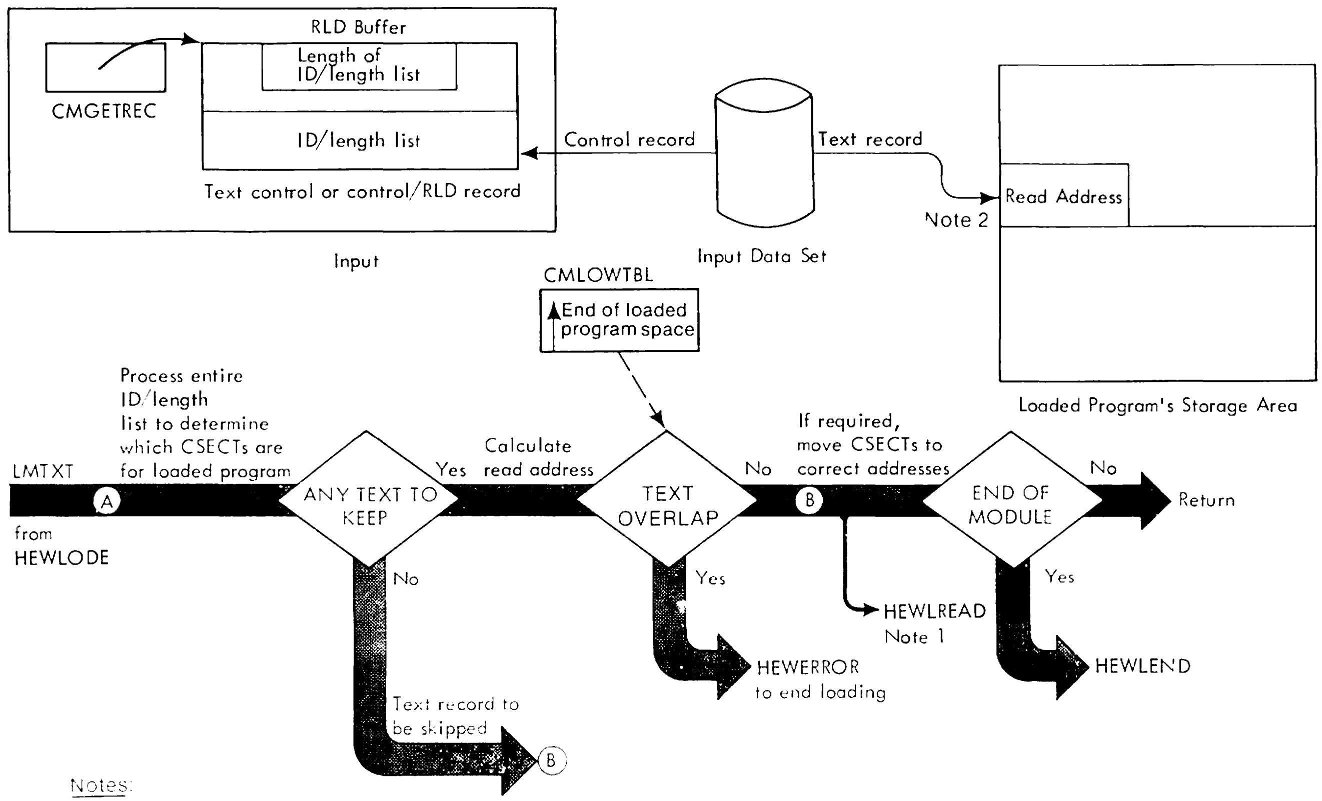


DIAGRAM D8. LOAD MODULE TEXT PROCESSING (BELOW-THE-LINE LOADING)

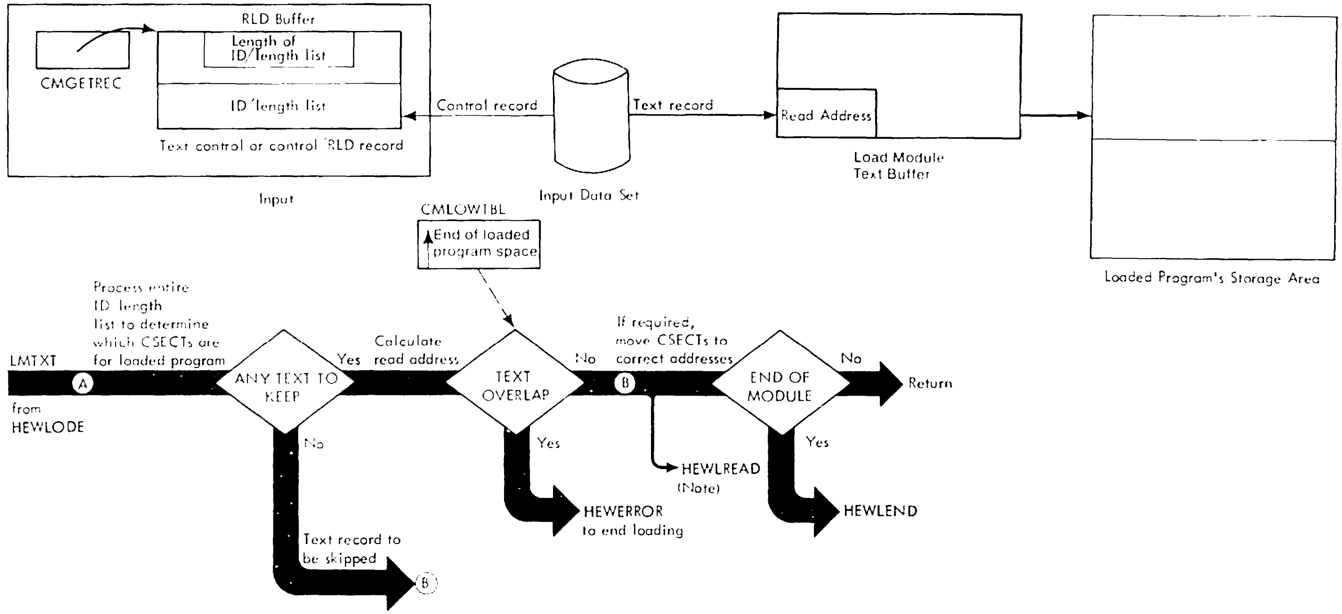


Notes:

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.

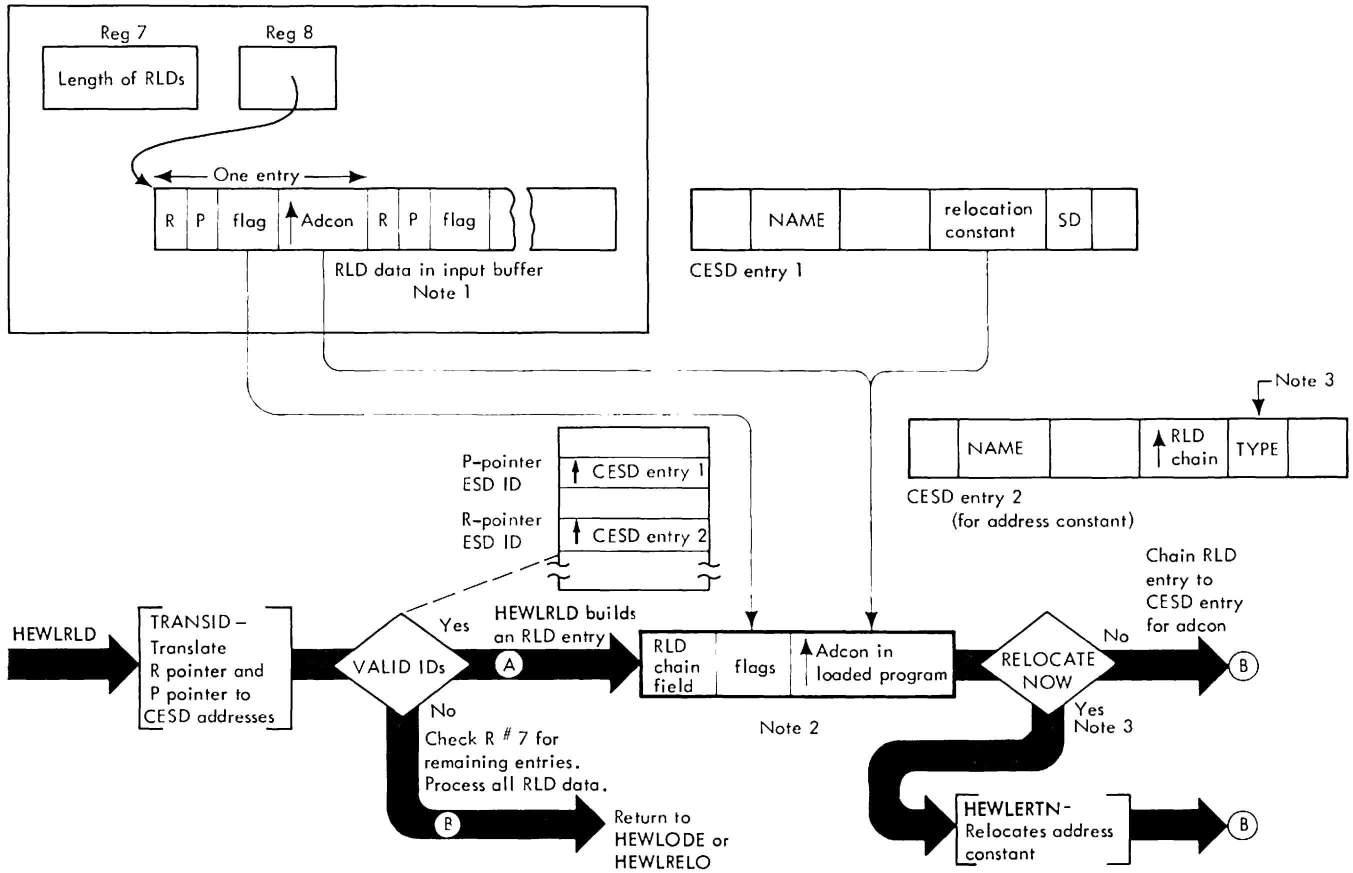
2. See Figure 19.

DIAGRAM D9. LOAD MODULE TEXT PROCESSING (ABOVE-THE LINE LOADING)



Note : 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.

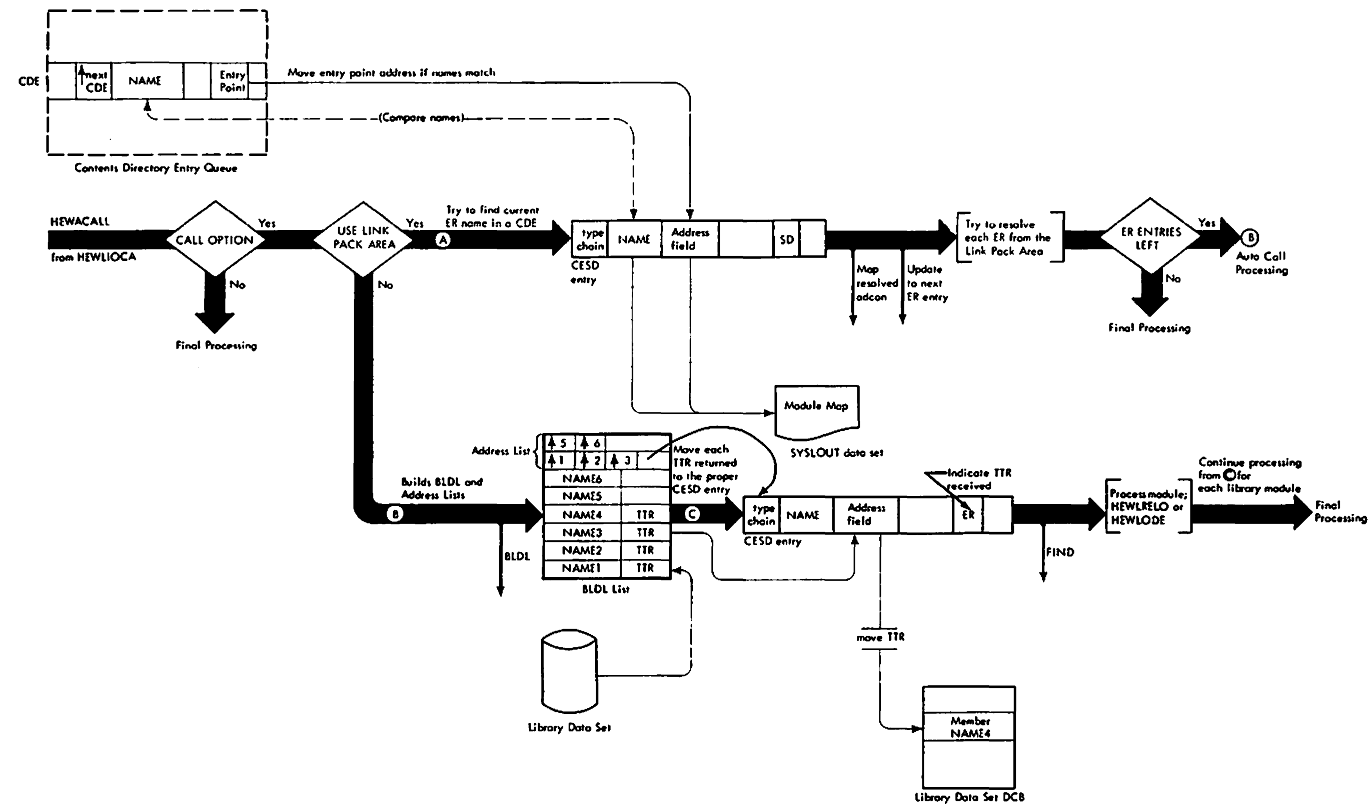
DIAGRAM D10. RLD RECORD PROCESSING



Notes:

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



ORGANIZATION OF THE LOADER

Figure 22 shows the organization of the loader. The flow of control through the first four levels of the processing portion of the loader (module HEWLOADR) is listed in the control level tables below.

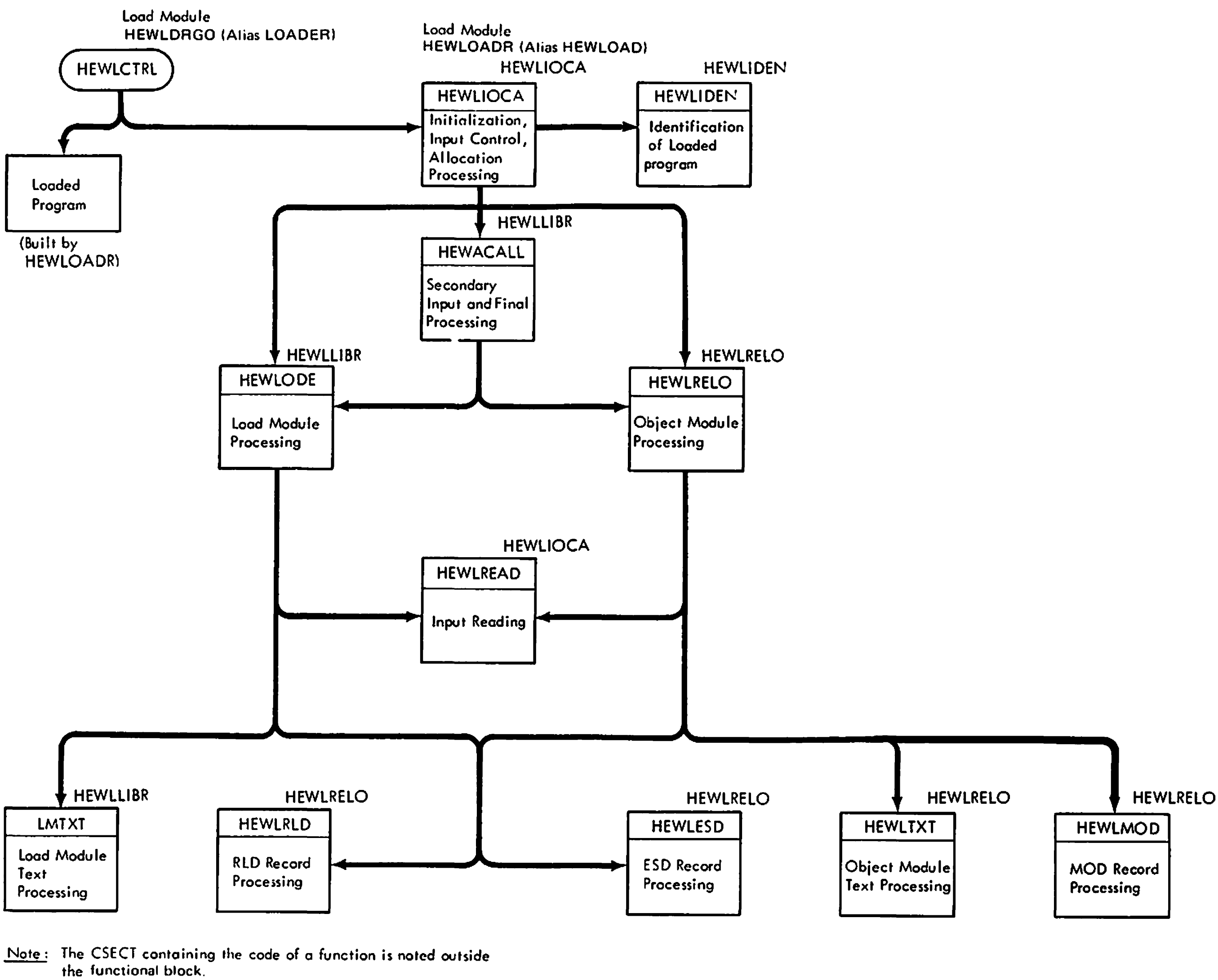


Figure 22. Loader Organization

ROUTINE CONTROL-LEVEL TABLES

The routine descriptions within a level are listed alphabetically in Figure 23 through Figure 26.

Routine	Purpose	Called Routines	Calling Conditions
HEWLIOCA	Initialization, primary input control, and allocation processing	HEWLPRNT	Called if SYSLOUT data set is open
		HEWBUFFR	If more data exists on SYSLIN
		HEWPRIME	If SYSLIN input is an object module
		HEWLRELO	If SYSLIN input is an object module
		HEWLODE	If SYSLIN input is a load module
		HEWACALL	When all SYSLIN input is processed, unless SYSLIN did not open
		HEWLIDEN	If the loaded program is to be identified to the control program
		HEWBTMAP	Input processing completed
		HEWERROR	If AMODE/RMODE parameter error detected; if redrive required

Figure 23. HEWLOADR—Level 1

Routine	Purpose	Called Routines	Calling Conditions
HEWACALL	Secondary input and final processing	HEWOPNLB	If ERs cannot be resolved from primary input or the LPA
		COMMON	Always
		HEWLMAP	If an ER is resolved
		HEWLERTN	If an ER is resolved
		HEWERROR	If an error occurs
		HEWPRIME	If SYSLIB input is object modules
		HEWLRELO	If SYSLIB input is object modules
		HEWLODE	If SYSLIB input is load modules
HEWBTMAP	Processing of error-bit map and printing of diagnostic dictionary	HEWLPRNT	If SYSLOUT is open and messages are required

Figure 24 (Part 1 of 2). HEWLOADR—Level 2

Routine	Purpose	Called Routines	Calling Conditions
		HEWTERM	If the TERM option is specified and messages are required
HEWBUFFR	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
HEWLIDEN	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
		HEWERROR	If an error occurs
HEWLODE	Process a load module	HEWLREAD	Always
		HEWLEND	If end-of-module is indicated
		HEWLESD	If CESD record is received
		HEWLRLD	If RLD record is received
		LMTXT	If TXT record is read in
HEWLPRNT	Print output to SYSLOUT data set	RDCHECK	If DECB was previously written
		WTWRITE	Always
		WTCHECK	Always
HEWLRELO	Process an object module	HEWLREAD	Always
		HEWLEND	If END card received
		HEWLESD	If ESD card received
		HEWLRLD	If RLD card received
		HEWLTXT	If TXT card received
		HEWLMOD	If MOD card received
HEWPRIME	Read records into all but one buffer before HEWLRELO receives control	RDREAD	Always

Figure 24 (Part 2 of 2). HEWLOADR—Level 2

Routine	Purpose	Called Routines	Calling Conditions
COMMON	Assign addresses to common areas	PSEUDOR	Always
		HEWLMAP	Always, unless no CM entries were received
		HEWLERTN	Always, unless no CM entries were received
FREECORE	Chain deallocated area to free list	none	
GETCORE	Allocated storage for allocation request	HEWERROR	If table overflow occurs
IDENTER	Create entry in extent list	none	
IDMINI	Create a condensed symbol table	none	
HEWERROR	Handle error messages, severity code 4 errors	HEWLPRNT	If SYSLOUT data set is open
		HEWTERM	If the TERM option is specified
HEWLCNVT	Convert binary quantity to hexadecimal	none	
HEWLEND	Process END card, reinitialize for next module	TRANSID	If END card specifies entry point address
		HEWERROR	If error occurs in end card processing
HEWLERTN	Relocate all address constants indicated by RLD chain	HEWERROR	Invalid 2-byte address constant
HEWLESD	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
		CKECKEP	If PC entry is required
		MATERSD2	If PC entry is required
		TRANSID	If LD/LR is received
HEWLMAP	Create map entry for referenced location in loaded program	HEWLPRNT IEWLCNVT	Always Always

Figure 25 (Part 1 of 3). HEWLOADR—Level 3

Routine	Purpose	Called Routines	Calling Conditions
HEWLMOD	Process MOD card, store text origin, length, and extent information	ALLOCATE	If extent information is passed on MOD card
HEWLODE	Process a load module	HEWLREAD	Always
		HEWLEND	If end-of-module is indicated
		HEWLESD	If ESD record is read in
		HEWLRLD	If RLD record is read in
		LMTXT	If TXT record is read in
HEWLPRNT	Print output to SYSLOUT data set	RDCHECK	If DECB was previously written
		WTWRITE	Always
		WTCHECK	Always
HEWLREAD	Handle request for data	RDREAD	Always
		RDCHECK	Always
HEWLRELO	Process an object module	HEWLREAD	Always
		HEWLEND	If END card is received
		HEWLESD	IF ESD card is received
		HEWLRLD	If RLD card is received
		HEWLTXT	If TXT card is received
HEWLRLD	Relocate address constants indicated by RLD entries received, or chain RLDs off CESD entry for R pointer	TRANSID	Always
		ALLOCATE	If no free RLD entry is available
		HEWLERTN	If relocation is possible, or if delinking required
HEWLTXT	Move object module text to correct space	TRANSID	Always
		RELOREAD	Always
		HEWERROR	If invalid ID received
HEWOPNLD	Open SYSLIB; close SYSLIN	HEWBUFFR	Unless SYSLIB was not opened
HEWPRIME	Read records into all but one buffer before HEWLRELO receives control	RDREAD	Always
HEWTERM	Print output to SYSTERM data set	WTWRITE	Always
		WTCHECK	Always

Figure 25 (Part 2 of 3). HEWLOADR—Level 3

Routine	Purpose	Called Routines	Calling Conditions
LMTXT	Read load module text into main storage	TRANSID	Always
		HEWLREAD	Unless record is to be skipped
		HEWERROR	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	

Figure 25 (Part 3 of 3). HEWLOADR—Level 3

Routine	Purpose	Called Routines	Calling Conditions
ALLOCATE	Allocate table extent	HEWERROR	Table overflow
CESDENT	Get CESD entry form free entry list or, call ALLOCATE to obtain an entry	ALLOCATE	No free entries on list
CESDSRCH	Search CESD for input name	MATCHED	If name is found
		NOMATCH	If name is not found
CHECKEP	Check CESD entry for specified entry point	none	
ENTER	Enter information in CESD entry for PC or SD	HEWERROR	If program is too large; if AMODE/RMODE data error detected
HEWBUFFR	Buffer management	FREECORE	If previous or current (not the first) allocation request is for object module
		GETCORE	If no previously allocated area is large enough for current request
HEWERROR	Handles error messages, severity code 4 errors	HEWLPRNT	If SYSLOUT data set is open
		HEWTERM	If the TERM option is specified
HEWLCNVT	Convert binary quantity to hexadecimal	none	
HEWLEND	Process END card, reinitialize for next module	TRANSID	If END card specifies entry point address
		HEWERROR	If error occurs in END card processing
HEWLERTN	Relocate all address constants indicated by RLD chain	HEWERROR	Invalid 2-byte address constant; invalid 3-byte address constant
HEWLESD	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

Figure 26 (Part 1 of 3). HEWLOADR—Level 4

Routine	Purpose	Called Routines	Calling Conditions
HEWLMAP	Create map entry for referenced location in loaded program	HEWLPRNT	Always
		HEWLCVNT	Always
HEWLPRNT	Print output to SYSLOUT data set	RDCHECK	If DECB was previously written
		WRWRITE	Always
		WTCHECK	Always
HEWLREAD	Handle request for data	RDREAD	Always
		RDCHECK	Always
HEWLRLD	Relocate address constants indicated by RLD entries received, or chain RLDs off CESD entry for R pointer	TRANSID	Always
		ALLOCATE	If no free RLD entry is available
		HEWLERTN	If relocation is possible, or if delinking is required
HEWLTXT	Move object module text to correct spaces	TRANSID	Always
		RELOREAD	Always
		HEWERROR	If invalid ID is received
HEWTERM	Print output to SYSTERM data set	WTWRITE	Always
		WTCHECK	Always
LMTXT	Read load module text into virtual	TRANSID	Always
		HEWLREAD	Unless record is to be skipped
		HEWERROR	If text record not received
		PROCEOM	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	HEWLEND	Always
PSEUDOR	Assign displacements to pseudo registers	HEWLPRNT	If displacement is assigned
		FINISHUP	Always
		HEWLMAP	If displacement is assigned
		HEWLERTN	If displacement is assigned

Figure 26 (Part 2 of 3). HEWLOADR—Level 4

Routine	Purpose	Called Routines	Calling Conditions
RDCHECK	Check DECB	none	
RDREAD	Read input using DECB information	none	
RELOREAD	Go to HEWLREAD for more input	HEWLREAD	Always
TRANSID	Translate input ESD ID to CESD address	ALLOCATE	If new extent is required
		HEWERROR	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	

Figure 26 (Part 3 of 3). HEWLOADR—Level 4

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

Name	Description	Object Module	CSECT	Synopsis
ALLOCATE	Allocation Routine	HEWLDREL	HEWLRELO	Allocates storage for table entries
CMTRCTRL	Table	HEWLDREL	HEWLRELO	Pointers to translation table extents
CMTYPCHN	Table	HEWLDREL	HEWLRELO	Pointers to CESD type chains
COMMON	Label	HEWLDLIB	HEWLLIBR	Assigns addresses to common
DECB	DSECT	HEWLDIOC	HEWLIOCA	Model DECB
ERCODES	DSECT	HEWLDIOC HEWLDREL HEWLDLIB	HEWLIOCA HEWLRELO HEWLLIBR	Error code definitions
FINISHUP	Label	HEWLDLIB	HEWLLIBR	Prints finishing messages
HEWACALL	Entry point	HEWLDLIB	HEWLLIBR	Automatic library call processing
HEWBTMAP	Entry point	HEWLDLIB	HEWLLIBR	Diagnostic dictionary processing
HEWBUFFR	Buffer allocation routine	HEWLDIOC	HEWLIOCA	Buffer and DECB allocation routine
HEWERROR	Entry Point	HEWLDLIB	HEWLLIBR	Error log routine
HEWLCNVT	Entry Point	HEWLDREL	HEWLRELO	Binary-Hex conversion routine
HEWLCTRL	Entry Point and CSECT	HEWLDCTR	HEWLCTRL	Loader control module
HEWLDCOM	DSECT	HEWLDIOC HEWLDLIB HEWLDREL	HEWLIOCA HEWLLIBR HEWLRELO	Communication area
HEWLDDEF	CSECT	HEWLDDEF	HEWLDDEF	SYSGEN option defaults
HEWLEND	Entry Point	HEWLDREL	HEWLRELO	End processing
HEWLERTN	Entry Point	HEWLDREL	HEWLRELO	RLD relocation routine
HEWLESD	Entry Point	HEWLDREL	HEWLRELO	ESD record processing

Name	Description	Object Module	CSECT	Synopsis
HEWLIDEN	Entry Point	HEWLDIDY	HEWLIDEN	Builds extent list for IDENTIFY and issues IDENTIFY
HEWLIDEN	Entry Point and CSECT	HEWLDIDY	HEWLIDEN	Identification routine
HEWLIOCA	Entry Point and CSECT	HEWLDIOC	HEWLIOCA	Initialization, I/O, control, and allocation processing
HEWLLIBR	CSECT	HEWLDLIB	HEWLLIBR	Automatic library call and load module processing
HEWLMAP	Entry Point	HEWLDREL	HEWLRELO	Creates map printout
HEWLMOD	Entry Point	HEWLDREL	HEWLRELO	MOD record processing
HEWLOAD	Entry Point	HEWLDIOC	HEWLIOCA	Entry point for loading with identification
HEWLODE	Entry Point	HEWLDLIB	HEWLLIBR	Load module processing
HEWLPRNT	Entry Point	HEWLDIOC	HEWLIOCA	Print routine
HEWLREAD	Entry Point	HEWLDIOC	HEWLIOCA	Read routine
HEWLRELO	Entry Point	HEWLDREL	HEWLRELO	Object module processor
HEWLRELO	CSECT	HEWLDREL	HEWLRELO	Object module, ESD, RLD, and map processing
HEWLRLD	Entry Point	HEWLDREL	HEWLRELO	RLD record processing
HEWLTXT	Label	HEWLDREL	HEWLRELO	Object module text processing
HEWOPNLB	Entry Point	HEWLDIOC	HEWLIOCA	Opens SYSLIB data set
HEWPRIME	Entry Point	HEWLDIOC	HEWLIOCA	Object module buffer prime routine
HEWTERM	Entry Point	HEWLDIOC	HEWLIOCA	SYSTERM routine
IDMINI	Label	HEWLDIDY	HEWLIDEN	Constructs MINI-CESD for test package if TSO is operating
INITMAIN	DSECT	HEWLDIOC	HEWLIOCA	Initial work area
LMTXT	Label	HEWLDLIB	HEWLLIBR	Load module text processing
MODELDCB	Label	HEWLDIOC	HEWLIOCA	Model DCB for SYSLIN, SYSLIB
OPENEXIT	Entry Point	HEWLDIOC	HEWLIOCA	DCB exit routine
PSEUDOR	Label	HEWLDLIB	HEWLLIBR	Processes pseudo registers
SYNAD	Entry Point	HEWLDIOC	HEWLIOCA	SYNAD routine

Name	Description	Object Module	CSECT	Synopsis
TRANSID	Entry Point	HEWLDREL	HEWLRELO	Translates ESD ID to CESD address

This section provides a detailed description of internal data areas used during loader processing. The data areas are described in alphabetic order.

Also included in this section is a summary of data area usage and construction (Figure 27).

Data Area	Built By	Used and/or Modified By
Address list	HEWACALL	[1]
BLDL list	HEWACALL	[1]
CESD control table (CMTYPCHN)	HEWLESD	HEWACALL, HEWLESD
CESD table	HEWLESD	HEWACALL, HEWLERTN, HEWLESD, HEWLRLD, HEWLTXT, LMTXT
Condensed symbol table	HEWLIDEN	TSO test facilities
Extent chain	HEWLMOD	HEWLIDEN
IDENTIFY parameter list	HEWLIDEN	IDENTIFY macro instruction
HEWLDCOM	HEWLIOCA	[2]
INITMAIN	HEWLIOCA	[1]
RLD table[1]	HEWLRLD	HEWACALL, HEWLERTN, HEWLRLD
Translation table	HEWLESD	HEWACALL, HEWLESD, HEWLRLD, HEWLTXT, LMTXT, TRANSID

Figure 27. Data Area Construction and Usage

Notes to Figure 27:

[1] Built and processed entirely within one routine.

[2] Major communication area throughout loader processing.

Address List

Built by the Secondary Input Processor

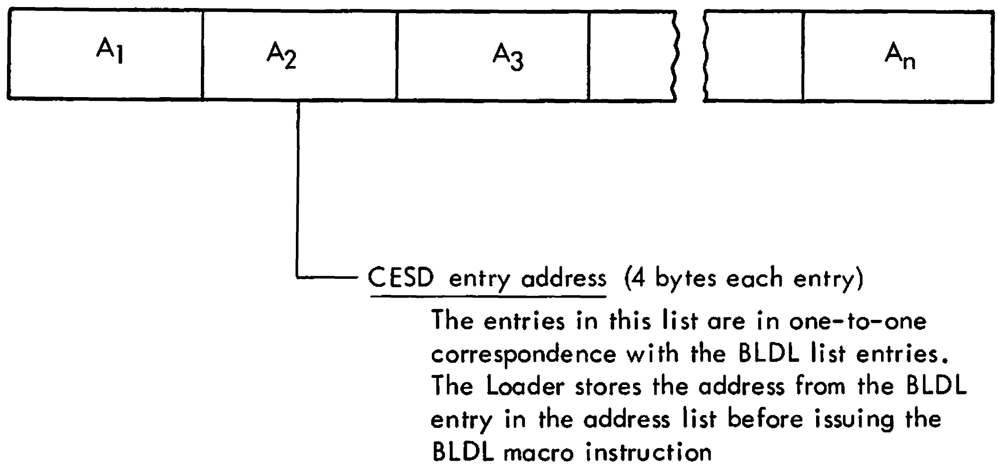


Figure 28. Address List

BLDL List

Built by Secondary Input Processor

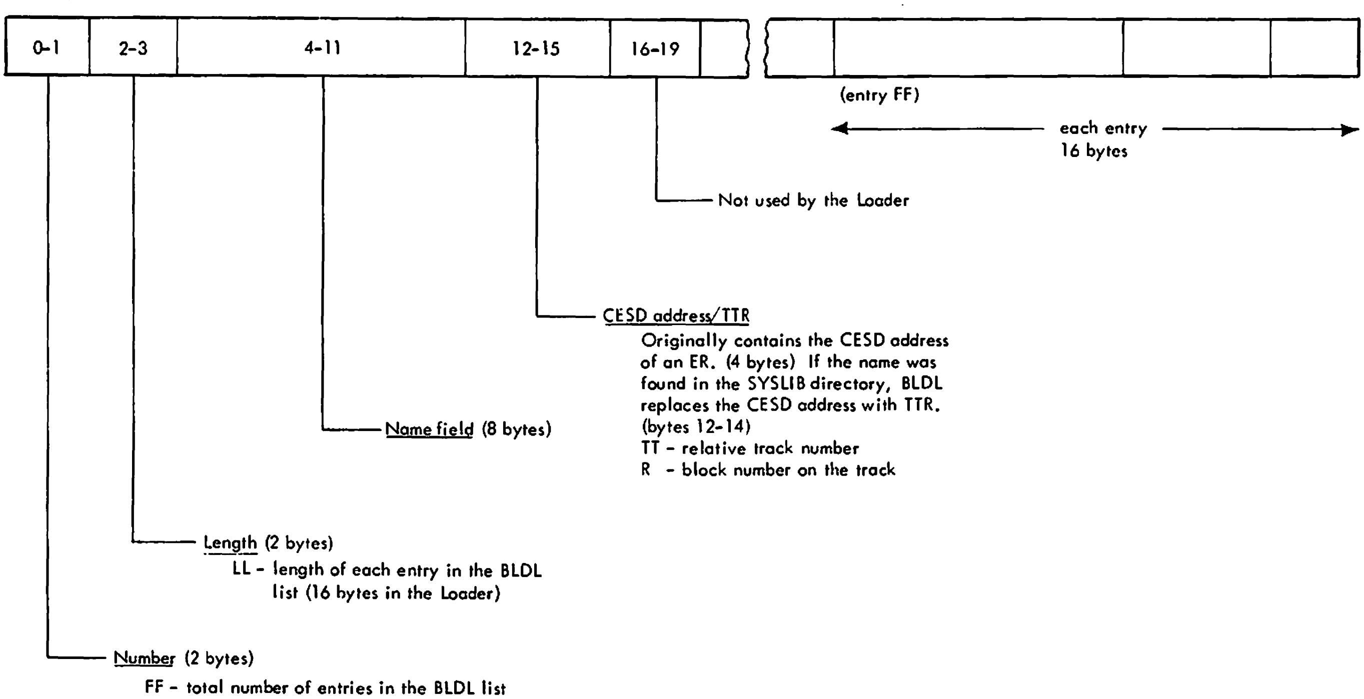
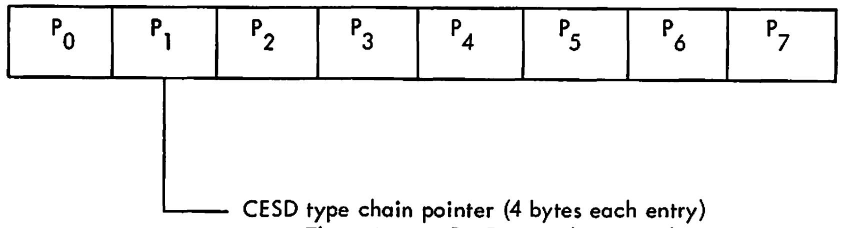


Figure 29. BLDL List

CESD Control Table (CMTYPCHN)
Built by the ESD Processor



CESD type chain pointer (4 bytes each entry)
The pointers, P_0-P_7, are listed in the following order by type : SD, LD, ER, LR, PC, CM, PR, NULL

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

Figure 30. CESD Control Table (CMTYPCHN)

CESD Table Entry
Built by the ESD processor

0-3	4-11	12-15	16-19	20	21

21 — AMODE/RMODE flags
- 0-4 not used
- 5 RMODE
  - 0 = 24
  - 1 = ANY
- 6-7 AMODE
  - 00 = 24
  - 01 = 24
  - 10 = 31
  - 11 = ANY

20 — Flags/type field (1 byte)

	F F F F F	T T T;	F_{1-5} are flags, T_{1-3} indicate type
Section definition (SD)	- X 0 X 0 0	0 0 0	F_1 - "delete," F_3 - "no length"
Label definition (LD)	- 0 X 0 0 0	0 0 1	F_2 - "LD processed"
External reference (ER)	- X X X X X	0 1 0	F_1 - "delete," F_2 - "weak call," F_3 - "BLDL tried," F_4 - "TTR found," F_5 - "never call"
Label reference (LR)	- X 0 0 0 0	0 1 1	F_1 - "delete"
Private code (PC)	- 0 0 0 0 0	1 0 0	
Common (CM)	- 0 0 0 0 0	1 0 1	
Pseudo register (PR)	- 0 0 0 0 0	1 1 0	

16-19 — Use depends on entry type

Type LD - ESD ID for SD; preliminary use only (bytes 18, 19)

Type PR - boundary alignment (byte 16) and length (bytes 17-19)
- Alignments
  - 7 - doubleword
  - 3 - fullword
  - 1 - halfword
  - 0 - byte

Types SD, PC, LR, CM - relative relocation constant

Type ER - 0; if ER was created from an LR - input address

Type CM - address of extended portion of entry

12-15 — Address/displacement field (4 bytes)
- Types SD, PC, LR, CM - Loader-assigned address
- Types CM, PR, ER - address of RLD entry chain (0, if no RLDs)
- Type PR - displacement within DSECT
- Type LD - input address (preliminary use only)

4-11 — Namefield (8 bytes)
- 8-character symbolic name or blanks for blank common and private code (unused for extended portion of CM entry)

0-3 — Chain address (4 bytes)
- Pointer to next entry on CESD type chain; if end of chain, 0 (unused for extended portion of CM entry)

Figure 31. CESD Entry

Condensed Symbol Table Entry

Built by the Identification Processor

0-7	8-11	12

Type - (1 byte)

Section definition	(SD)	xxxxx 000
Common	(CM)	xxxxx 101

Address - Assigned address of this symbol (4 bytes).

Symbol - The 8-character external name (8 bytes).

Figure 32. Condensed Symbol Table Entry

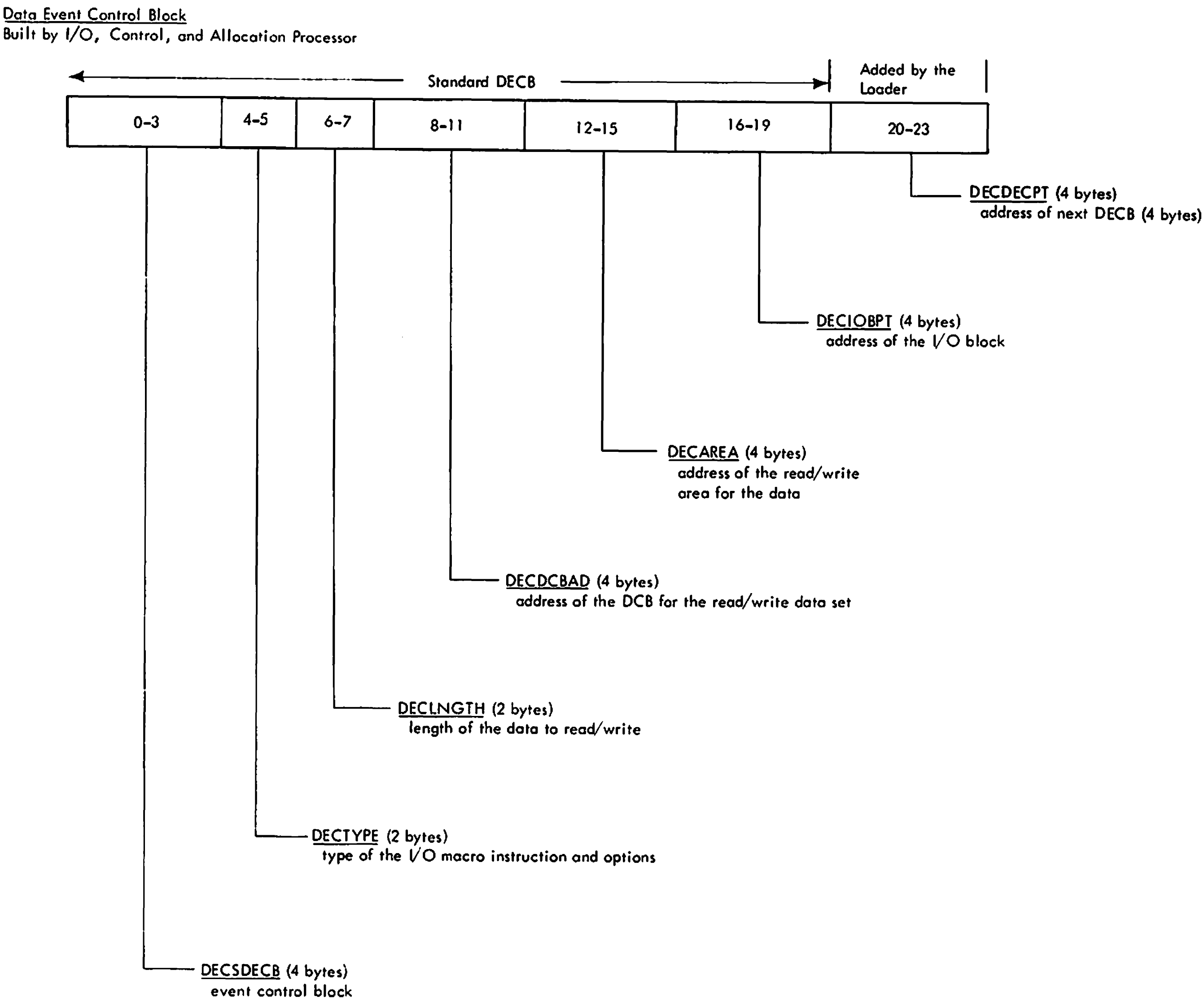


Figure 33. Data Event Control Block (DECB)

Extent Chain Entry

Built by the MOD Processor

0-3	4-7	8-11

Length - Length of the extent (4 bytes).

Address - Address of the extent derived from the MOD record (4 bytes).

Chain Address - Address of the next entry on the extent chain; if end of chain, zero (4 bytes).

Figure 34. Extent Chain Entry

IDENTIFY Parameter List

Built by the Identification Processor

Address of entry point of program to be identified		
Program name - the 8-character symbolic name		
Addressing mode	Not used	
Length, in bytes, of extent list		Extent List
Number of extents described in this list		
Length of extent 1 (Loader-constructed program)*		
Length of extent 2 (Condensed symbol table), if any*		
Length of extent 3 (first block of preloaded text); if any*		
. . .		
Length of extent n*		
Address of extent 1 (Loader-constructed program)		
Address of extent 2 (Condensed symbol table)		
Address of extent 3 (first block of preloaded text), if any		
. . .		
Address of extent n		

4 bytes

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

Figure 35. IDENTIFY Parameter List

Offset Decimal	Offset Hex	Length	Symbol	Description
0	0	8	CMRDRSET	Multipurpose doubleword
8	8	8	CMXDBLWD	Temporary doubleword
			(CMADCON)	Relocation alignment area
16	10	8		Multipurpose doubleword
24	18	4	CMFSTSAV	Pointer to first save area
28	1C	4	CMBEGADR	Default entry point to module
32	20	4	CMRDCBPT	Input DCB pointer
36	24	4	CMWDCBPT	Output DCB pointer
40	28	4	CMTDCBPT	System DCB pointer
44	2C	4	CMRDECPT	Input DECB pointer
48	30	4	CMWDECPT	Output DECB pointer
52	34	4	CMGETREC	Input logical record pointer
56	38	4	CMPUTREC	Output logical record pointer
60	3C	4	CMTRMREC	System buffer pointer
64	40	4	CMNXTTXT	Next address to be assigned to a CSECT
68	44	4	CMLSTTXT	Highest text address assigned to current CSECT
72	48	4	CMLOWTBL	Lowest address assigned for loader tables
76	4C	4	CMHITBL	Highest storage address available to loader
			(BTLHIADR)	Highest address in below-the-line loading
80	50	4	CMIOLST1	Open list, DCB pointer #1
84	54	4	CMIOLST2	Open list, DCB pointer #2
88	58	4	CMIOLST3	Open list, DCB pointer #3
92	5C	4	CMCORE1	Corresponds to CMNXTTXT for pre-loaded text
96	60	4	CMCORE2	Corresponds to CMLSTTXT for pre-loaded text
100	64	4	CMTOPCOD	Highest text address before common allocated
104	68	4	CMLIBEOD	EODAD error routine pointer for passed SYSLIB
108	6C	4	CMLIBSYN	SYNAD error routine pointer for passed SYSLIB
112	70	4	CMLIBEXL	Exit list pointer for passed SYSLIB
116	74	4	REDRLIST	CLOSE/OPEN list for SYSLIN (DCB pointer) used by redrive
120	78	4	ATLHIADR	Highest address in above-the-line loading
124	7C	4	TXTBUFND	Lowest table address in above-the-line loading
128	80	4	ATLMADR	Lowest address of above-the-line GETMAIN
			(ATLLOADR)	Lowest address in above-the-line loading
132	84	4	ATLMLNG	Length of above-the-line GETMAIN
136	88	4	TBLLO	Vector to table overflow address
140	8C	4	TXTHI	Vector to text overflow address
144	90	4	CMMAINPT	Address of loaded program
148	94	4	CMGETPRM	Minimum for variable conditional GETMAIN
152	98	4		Maximum for variable conditional GETMAIN
156	9C	10	CMGETLST	Parameter list for variable conditional GETMAIN
166	A6	2	CMBLKSIZ	Block size of current input object module
168	A8	2	CMMAXLNE	Maximum line count (SYSPRINT)
170	AA	2	CMMAPLIN	Length of map line
172	AC	2	CMWLRECL	SYSPRINT record size
174	AE	2	CMMAXLST	Maximum length of invalid options list
176	B0	4	BTLMADR	Variable conditional GETMAIN address
			(BTLLOADR)	Lowest address of below-the-line GETMAIN
			(TXTBUFST)	Text buffer address for above-the-line loading
180	B4	4	BTLMLNG	Length of below-the-line GETMAIN
			(CMMAINSZ)	Variable conditional GETMAIN size
184	B8	8	CMPRNTDD	Print ddname
192	C0	8	CMLINDD	Primary input ddname
200	C8	8	CMLIBDD	Library ddname
208	D0	8	CMTERMDD	SYSTERM ddname
216	D8	8	CMEPNAME	Entry point name
224	E0	8	CMPGMNM	Program name
232	E8	4	CMLINDCB	Passed SYSLIN control block pointer
236	EC	4	CMLIBDCB	Passed SYSLIB DCB pointer
240	F0	1	CMPRMFLG	Parameter flags:
			CQRES	X'01' RES/NORES
			CQMAP	X'02' MAP/NOMAP

Figure 36 (Part 1 of 3). HEWLDCOM DSECT—Communication Area

Offset Decimal	Offset Hex	Length	Symbol	Description
			CQPRINT	X'04' PRINT/NOPRINT
			CQLET	X'08' LET/NOLET
			CQCALL	X'10' CALL/NOCALL
			CQEPNAME	X'20' Entry point name defined
			CQEPADDR	X'40' Entry point address defined
			CQTERM	X'80' TERM/NOTERM
241	F1	1	CMIOFLGS	I/O flags:
			CQEOCB	X'01' End of concatenation
			CQEOFB	X'02' End of file
			CQEOFSB	X'04' End of file significance
			CQRECFM	X'08' Input record format (0 is Fixed)
			(CQUNDEF)	Separate name in allocation for undefined
			CQFIXED	X'10' Fixed record format
			CQIGNCR	X'20' Ignore control record on load module
			CQIOERR	X'40' An I/O error has occurred
242	F2	1	CMFLAG3	Assorted flags:
			CQTS	X'02' Time-sharing environment
			CQPGMNM	X'04' Program name passed
			CQPASLIN	X'08' SYSLIN DCB passed
			CQPASLIB	X'10' SYSLIB DCB passed
			CQINCORE	X'20' Processing incore SYSLIN
			CQIDEN	X'40' Entered at IEWLOAD. Identification wanted
243	F3	1	CMFLAG4	Assorted flags:
			CQESDS	X'01' ESDs have been encountered
			CQMOD	X'02' MOD card has been encountered
			CQNOEX	X'04' Execution not scheduled
			CQMINI	X'08' Mini-CESD built
			COMVT	X'10' MVS operating
			CQCOMMON	X'20' Common received
			CQTRMOPN	X'40' SYSTERM open
			CQIDONE	X'80' Identification accomplished
244	F4	1	CMFLAG5	Assorted flags:
			CMAMDREQ	X'01' AMODE req'd from entry point definition
			HDGLINE2	X'02' Print line 2 of heading
			MODEDATA	X'04' AMODE/RMODE data in ESD (not seg. no.)
			MODERREQ	X'08' Parm field mode specif error detected
			FIRSTESD	X'10' RMODE from first ESD processed
			REDRIVE	X'20' Restart of loading required
			MODEIMPL	X'40' AMODE or RMODE parm implied
			ATLLOAD	X'80' Loading above the line
245	F5	1	CMCHRMOD	RMODE value move length
246	F6	3		RMODE value character string
249	F9	1	CMCHAMOD	AMODE value move length
250	FA	3	CMCH	AMODE value character string
253	FD	1	CMRMODE	RMODE from parm field
254	FE	1	CMAMODE	AMODE from parm field
256	100	4	CMSYSTYP	System type saved by HEWLDLIB
260	104	36	CMRSAVE	Register save area used by HEWLDLIB
296	128	4	CMBITMAP	Error bit map
300	12C	2	CMLNECNT	Current line count (SYSPRINT)
302	12E	2	CMWTBFCT	Horizontal byte count in print record
304	130	4	CMXLCHN	Pointer to chain of extents
308	134	4	CMERLIST	Pointer to errors encountered during open
312	138	4	CMRLDCHN	Free RLD entry chain (8 bytes/entry)
316	13C	4	CMESDCHN	Free CESD entry chain (22 bytes/entry)
320	140	4	CMEPADDR	Entry point address to loaded program
324	144	128	CMTRCTRL	Translate control table
452	1C4	4	CMBLDLPT	BLDL pointer
456	1C8	4	CMCXDPT	Pointer to CXD addresses
460	1CC	4	CMFRECOR	Free storage chain
464	1D0	4	CMMODLNG	Length of module currently being processed
468	1D4	4	CMOBJST	Starting point for object module
472	1D8	4	CMTEMPCH	Pointer to load chain entry to be freed
476	1DC	4	CMEPCESD	CESD line address of the entry point name
480	1E0	4	CMPREVPT	Previous element in a chain for insert-delete

Figure 36 (Part 2 of 3). HEWLDCOM DSECT—Communication Area

Offset				
Decimal	Hex	Length	Symbol	Description
484	1E4	4	CMLOADCH	Temporary chain for ESDs in a load module
488	1E8	4	CMESDSAV	CESD register save area for HEWLDREL
492	1EC	4	CMSDCHN	Type 0 - Section definition - chain pointer
			(CMTYPCHN)	Index point for the vector table
496	1F0	4	CMLDCHN	Type 1 - Label definition - chain pointer
500	1F4	4	CMERCHN	Type 2 - External reference - chain pointer
504	1F8	4	CMLRCHN	Type 3 - Label reference - chain pointer
508	1FC	4	CMPCCHN	Type 4 - Private code - chain pointer
512	200	4	CMCMCHN	Type 5 - Common - chain pointer
516	204	4	CMPRCHN	Type 6 - Pseudo register - chain pointer
520	208	4	CMNULCHN	Type 7 - Null entry - chain pointer
524	20C	2	CMCURRID	ESDID counter
526	20E	2	CMBLDLNO	Number of BLDL entries
528	210	2	CMNUMXS	Number of extents
530	212	1	CMLIBFLG	Autocall and load module processor flags:
			CQKEEPS	X'01' Keep some text from this record
			CQDELETE	X'02' Delete some text from this record
			CQAUTOC	X'04' Autocall is in process
			CQCESDR	X'08' CESD has been received for load module
			CQNOTXT	X'10' Text has been received
			CQLPASRH	X'20' LPA resolution possible
			CQFIRST	X'40' First record from load module was CESD
531	213	1	CMRELFLG	Relocation and object module processor flags:
			CQESD	X'01' ESD routine is caller to ID translate rtn
			CQNOLNG	X'02' Length not yet received from current CSECT
			CQDELINK	X'04' Delinking if required for common
			CQLIB	X'08' Resolution from SYSLIB in process
			CQNOEND	X'10' End card has been received
			CQINPUT	X'20' Input has been received
			CQENTRY	X'40' RLD is for entry point
			CQNOLNTX	X'80' Text received for no-length CSECT
532	214	1	CMSTATUS	Loader status flag:
			CQPRTOPN	X'01' Print DCB allocated for
			CQLIBOPN	X'02' Library DCB open
			CQABORT	X'04' Abort loading
			CQREJOPT	X'08' Invalid options are to be printed
			CQOPNERR	X'10' Errors were encountered during open
			CQRETURN	X'20' Caller to error routine must regain control
			CQMSGSAV	X'40' Request open exit to [illegible] error messages
			CQPRTDCB	X'80' Print DCB is open
533	215	1	CMPRTCTL	Index for printer carriage control
534	216	1	CMOPTECT	Count of invalid options to be printed
535	217	1	CMEPMODE	AMODE for entry point

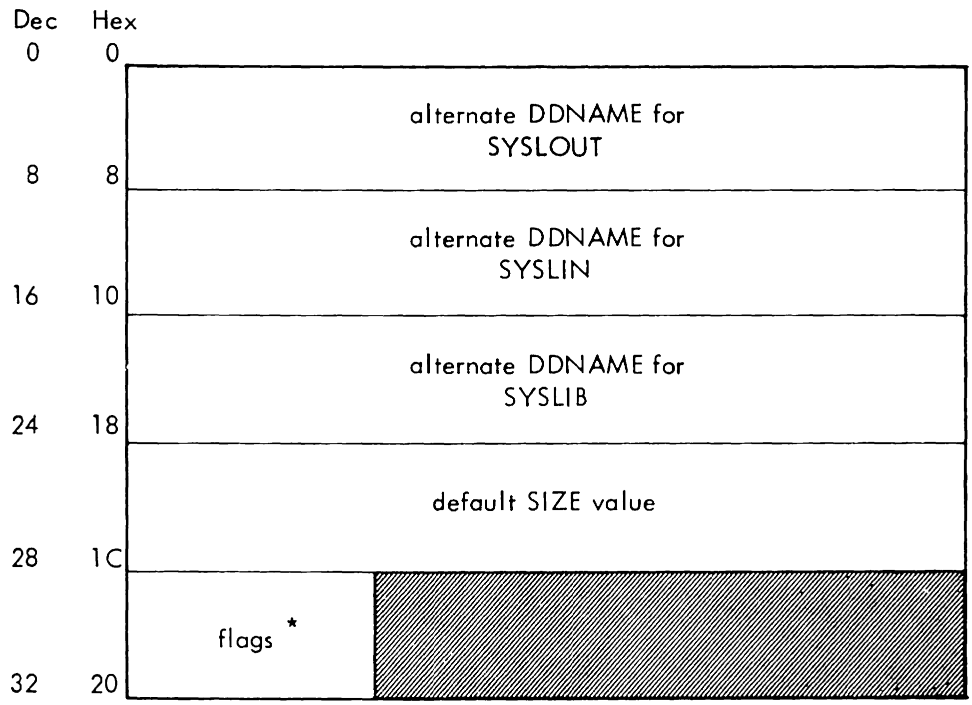
Figure 36 (Part 3 of 3). HEWLDCOM DSECT—Communication Area

Notes to Figure 36:

1. Symbols in parentheses are equated to preceding symbol.
2. Locations BTLMADR through CMAMODE are initialized from locations INITMADR through PARMAMOD in INITMAIN (Figure 38 on page 95) by HEWLDIOC.
3. Locations CMBITMAP through CMEPMODE are initialized to zero by HEWLDIOC.

HEWLDDEF is a static CSECT that defines default options and ddnames to be used by the loader.

During loader execution, the default values are moved to dynamic storage (INITMAIN), where they are modified by the parameter list values passed internally. The HEWLDDEF CSECT is described in Figure 37.



*Correspond to CMPRMFLG flags. See Figure 36.

Figure 37. HEWLDDEF CSECT

Offset Decimal	Hex	Length	Symbol	Description
0	0	72	INITSAVE	Initial save area
72	48	4	INITMADR	Variable conditional GETMAIN storage address
76	4C	4	INITMSIZ	Variable conditional GETMAIN storage size
80	50	8	INITPRNT	ddname for diagnostic message data set
88	58	8	INITLIN	ddname for primary input data set
96	60	8	INITLIB	ddname for autocall library data set
104	68	8	INITTERM	ddname for SYSTERM data set
112	70	8	INITNAME	Parameter list entry point name
120	78	8	INITPGMN	Program name
128	80	4	INLINDCB	Address of passed SYSLIN DCB
132	84	4	INLIBDCB	Address of passed SYSLIB DCB
136	88	2	INITPARM	Parameter flags and error flags
138	8A	1	INFLAG3	Assorted flags
139	8B	1	INFLAG4	Assorted flags
140	8C	1	INFLAG5	Assorted flags
141	8D	1	CHARRMOD	RMODE value move length
142	8E	3		RMODE value character string
145	91	1	CHARAMOD	AMODE value move length
146	92	3		AMODE value character string
149	95	1	PARMRMOD	RMODE from parm field
150	96	1	PARMAMOD	AMODE from parm field
152	98	4	INITSPIE	Pointer to previous SPIE for 'SIZE=' SCAN
156	9C	4	INITSCAN	Scan pointer save area for 'SIZE=' SPIE
160	A0	4	INITDUM	Save word for register during size processing
164	A4	4	INITREJL	End of rejected options list
168	A8	4	INITRMIN	Minimum size request for variable conditional GETMAIN
172	AC	4	INITRMAX	Maximum size request for variable conditional GETMAIN
176	B0	12	INITGTML	Parameter list area for variable conditional GETMAIN
188	BC	12	INITEXTR	Parameter list area for Extract
200	C8	4	INITEXAD	Address of TCB TSO field from Extract
208	D0	8	INITDBLW	Doubleword for parm 'SIZE' conversion
216	D8	256	INITRTAB	Translate and test table for option scan
472	1D8	VL	INITREJP	Rejected options buffer

Figure 38. INITMAIN DSECT Definition

Note to Figure 38:

Locations BTLMADR through CMAMODE in HEWLDCOM (the communication area Figure 36 on page 91) are initialized from locations INITMADR through PARMAMOD in INITMAIN.

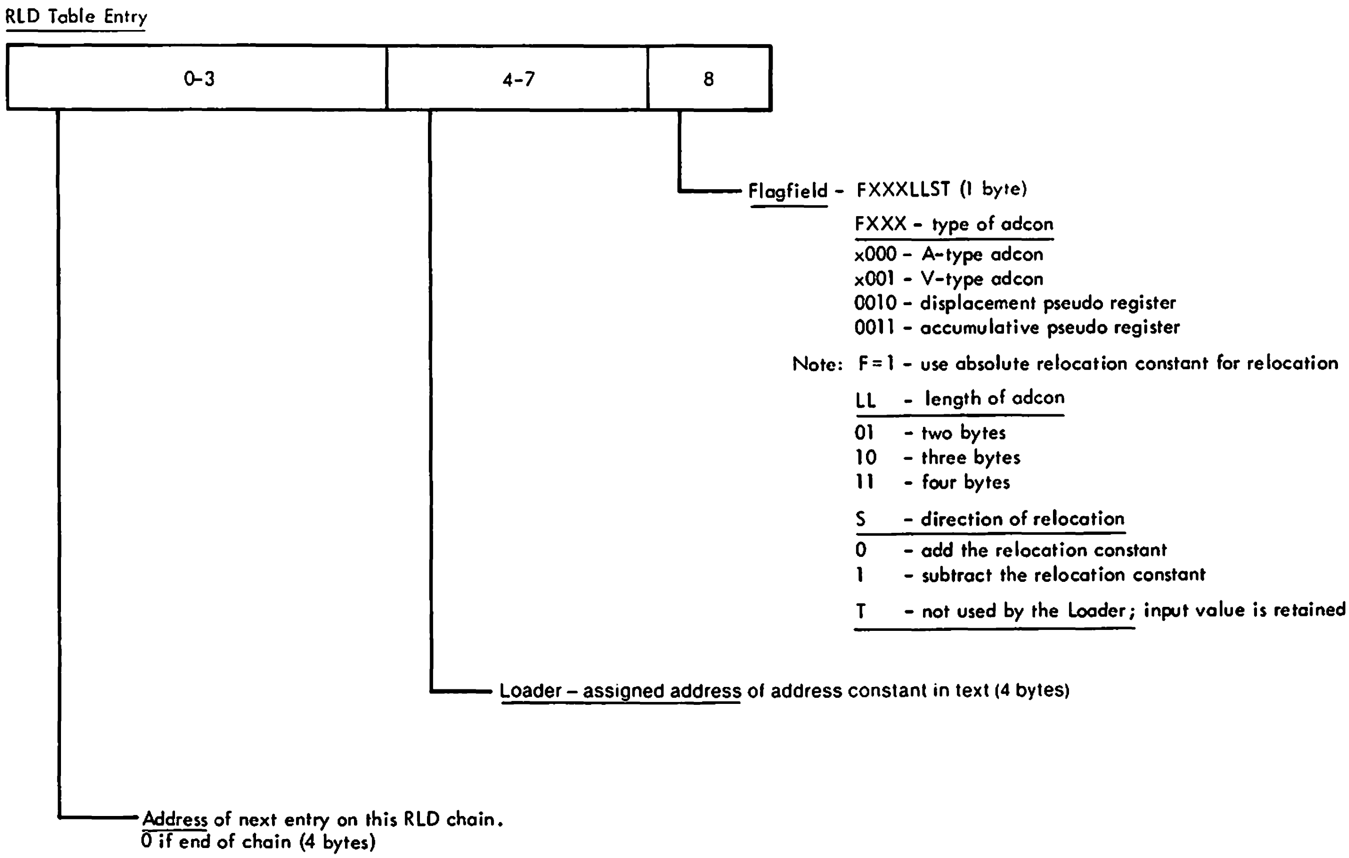
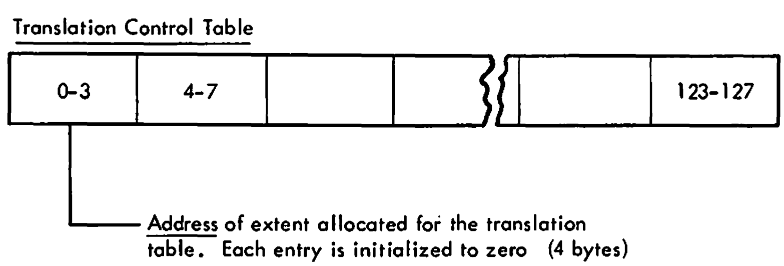
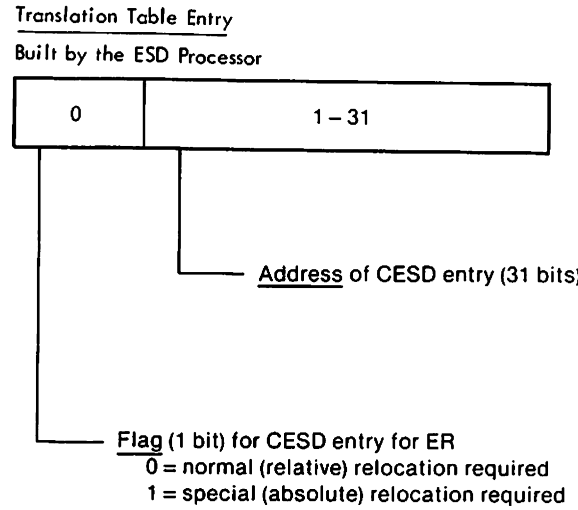


Figure 39. RLD Table Entry



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

Figure 40. 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 41. Translation Table

DIAGNOSTIC AIDS

This section contains information that is useful in diagnosing difficulties with the loader program. Included are: register contents at entry to routines (Figure 42), error code definitions (Figure 43 on page 100), an example of a module map (Figure 44 on page 101), and a list of serviceability aids available with the loader. To use this section, refer to Figure 22 on page 71 through Figure 26 on page 77 which show the logic flow, and Figure 27 on page 83 which shows data area usage.

Note: At the entry point to each module, register 13 contains the save area address and register 14 contains the return address.

Module	Entry Point	Register Contents
HEWLCTRL		1 - address of parameter list
HEWRELO	HEWLRELO	11 - address of communication area
	HEWLESD	5 - ID of first ESD item other than LD 7 - length of ESD information 8 - address of ESD information 11 - address of communication area
	HEWLTXT	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
	HEWLMOD	7 - length of MOD information 8 - address of MOD information 11 - address of communication area
	HEWLRLD	7 - length of RLD information 8 - address of RLD information 11 - address of communication area
	HEWLEND	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
	HEWLERTN	1 - starting address of RLD chain 9 - CESD entry address to be used for relocation 11 - address of communication area
	HEWLMAP	9 - address of CESD entry to be mapped 11 - address of communication area
	HEWLCNVT	1 - binary quantity to be converted 11 - address of communication area

Figure 42 (Part 1 of 2). Register Contents at Entry to Routines

Module	Entry Point	Register Contents
HEWLLIBR	HEWLODE	11 - address of communication area 15 - entry point address
	HEWERROR	0 - error message code 1 - pointer to qualifying information (if it exists) 11 - address of communication area 15 - entry point address
	HEWACALL	11 - address of communication area 15 - entry point address
	HEWBTMAP	11 - address of communication area 15 - entry point address
HEWLIOCA	HEWLIOCA	1 - address of parameter list 15 - entry point address
	HEWLOAD	1 - address of parameter list 15 - entry point address
	OPENEXIT	1 - address of DCB 11 - address of communication area 12 - base address of HEWLIOCA
	HEWBUFFR	10 - address of DCB 11 - address of communication area 15 - entry point address
	HEWLREAD	<u>For Object and Load Modules</u> 11 - address of communication area 15 - entry point address <u>For Load Modules</u> 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 - complement of length of text 1 - address of text
	HEWOPNLB	11 - address of communication area 15 - entry point address
	HEWLPRNT	11 - address of communication area 15 - entry point address
	HEWTERM	11 - address of communication area 15 - entry point address
	HEWPRIME	11 - address of communication area 15 - entry point address
HEWLIDEN	HEWLIDEN	11 - address of communication area
	IDMINI	5 - starting address for mini-CESD 10 - upper limit of storage available

Figure 42 (Part 2 of 2). Register Contents at Entry to Routines

ERROR CODE DEFINITIONS

Figure 43 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 HEWLIOCA, HEWLRELO, HEWLLIBR, and HEWLIDEN.)

Error Code	Definition	Sev	Message
ERRELO1	Unresolved external reference (NOCALL specified)	1	IEW1001
ERENTR1	No entry point received	1	IEW1161
ERINPT8	Card received not an object record	1	IEW1141
ERMODE1	Invalid AMODE/RMODE combination in PARM field	1	IEW1241
ERMODE2	Invalid AMODE/RMODE combination in ESD data	1	IEW1251
ERMODE3	Inconsistent RMODE data - RMODE=24 forced	1	IEW1271
ERINPT10	No END card received	2	IEW1182
ERINPT2	Invalid length specified	2	IEW1082
ERRELO2	Unresolved external reference	2	IEW1012
ERINPT4	Doubly defined ESD	2	IEW1102
ERINPT5	Invalid 2-byte address constant	2	IEW1112
ERINPT7	Invalid ID received	2	IEW1132
ERINPT9	Invalid record from object module	2	IEW1152
ERINPT1	Block size is invalid	2	IEW1072
ERINPT11	Common exceeds size of CSECT with same name	2	IEW1232
ERINPT12	Invalid 3-byte address constant	2	IEW1262
ERINPT3	No text received	3	IEW1093
ERENTR2	Entry point received but not matched	3	IEW1173
ERIOUT4	I/O error while searching library directory	3	IEW1053
ERINPT6	Invalid record from load module	3	IEW1123
ERIOUT3	Unacceptable record format (variable on input)	4	IEW1044
ERIOUT1	ddname cannot be opened	4	IEW1024
ERIOUT2	ddname had synchronous error	4	IEW1034
ERSIZE2	Available storage exceeded	4	IEW1194
ERSIZE3	Too many external names in input module	4	IEW1204
ERIDEN1	Identification failed; duplicate program name	4	IEW1214
ERIDEN2	Identification failed	4	IEW1224

Figure 43. Internal Error Code Definitions

Module Map Format

Map heading	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	IHEQINV	PR	00	IHEQERR	PR	04	IHEQTIC	PR	08	IHEQLWF	PR	0C	IHEQLWO	PR	10
	IHEQSLA	PR	14												
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 CESD.

Figure 44. Module Map Format Example

SERVICEABILITY AIDS

The loader provides the following serviceability aids:

- The control section, HEWLDDEF, contains the default values for loader options and is resident in load module HEWLOADR.
- A storage dump will typically produce information on the nature of the error. Register 11 will contain a pointer to HEWLDCOM, 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 hexadecimal "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 Figure 38 on page 95 for the layout of HEWLDCOM.
- 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 HEWLLIBR at the end of loading. (Figure 49 on page 108 is a list of these messages.)
- A module map (MAP) is provided to furnish information concerning the structure and contents of the program. Figure 48 on page 107 is 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.

APPENDIX. ERROR MESSAGES, ETC.

This appendix 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 the data sets passed to the loader.

Figure 45 lists the loader diagnostic messages. Each message contains a severity code in the final position of the message number. 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 executing the loaded program impossible.

3 Indicates an error that will make executing the loaded program impossible.

4 Indicates an unrecoverable error. Such an error causes termination of loading.

Message Number	Message Text	Issuer Routine	Issuer CSECT
IEW1001	Warning - Unresolved external reference (NOCALL specified)	HEWACALL	HEWLLIBR
IEW1012	Error - Unresolved external reference	HEWACALL	HEWLLIBR
IEW1024	Error - Ddname cannot be opened	HEWLIOCA	HEWLIOCA
IEW1034	Error - Ddname had synchronous error	SYNAD	HEWLIOCA
IEW1044	Error - Unacceptable record format (variable on input)	OPENEXIT	HEWLIOCA
IEW1053	Error - I/O error while searching library directory	HEWACALL	HEWLLIBR
IEW1072	Error - BLKSIZE is invalid	OPENEXIT	HEWLIOCA
IEW1082	Error - Invalid length specified	HEWLEND	HEWLRELO
IEW1093	Error - No text received	HEWACALL	HEWLLIBR
IEW1102	Error - Doubly defined ESD	HEWLESD	HEWLRELO
IEW1112	Error - Invalid 2-byte address constant	HEWLRLD	HEWLRELO
IEW1123	Error - Invalid record from load module	HEWLODE	HEWLLIBR
IEW1132	Error - Invalid ID received	HEWLRLD HEWLTXT HEWLEND TRANSID	HEWLRELO HEWLRELO HEWLRELO HEWLRELO

Figure 45 (Part 1 of 2). Error Message/Issuer Cross-Reference Table

Message Number	Message Text	Issuer Routine	Issuer CSECT
IEW1141	Warning - Card received not an object record	HEWLRELO	HEWLRELO
IEW1152	Error - Invalid record from object module	HEWLRELO	HEWLRELO
IEW1161	Warning - No entry point received	HEWACALL	HEWLLIBR
IEW1173	Error - Entry point received but not matched	HEWACALL	HEWLLIBR
IEW1182	Error - No END card received	HEWLRELO	HEWLRELO
IEW1194	Error - Available storage exceeded	HEWBUFFR HEWLESD HEWLEND HEWLTXT HEWACALL HEWLODE HEWLIDEN	HEWLIOCA HEWLRELO HEWLRELO HEWLRELO HEWLLIBR HEWLLIBR HEWLIDEN
IEW1204	Error - Too many external names in input module	TRANSID	HEWLRELO
IEW1214	Error - Identification failed - duplicate program name found	HEWLIDEN	HEWLIDEN
IEW1224	Error - Identification failed	HEWLIDEN	HEWLIDEN
IEW1232	Error - Common exceeds size of CSECT with same name	MATCHCM	HEWLRELO
IEW1241	Warning - Invalid AMODE/RMODE combination found in PARM field - ignored	NOLINE2	HEWLIOCA
IEW1251	Warning - Invalid AMODE/RMODE combination in ESD data for the named CSECT - ignored	ENTER	HEWLRELO
IEW1262	Error - Invalid 3-byte address constant	HEWLERTN	HEWLRELO
IEW1271	Warning - Inconsistent RMODE data - RMODE=24 forced	MNREDRIV	HEWLIOCA
IEW1991	Error - User program has abnormally terminated	HEWLCTRL	HEWLCTRL

Figure 45 (Part 2 of 2). Error Message/Issuer Cross-Reference Table

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 indication (END record in an object module, EOM flag in a load module.)
- 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.
- Because 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 references to a single control section whose length is the maximum length specified in the CM items of that name (or blank length). 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.

INPUT RECORD FORMATS

Figure 46 on page 105 through Figure 58 on page 116 show input record formats.

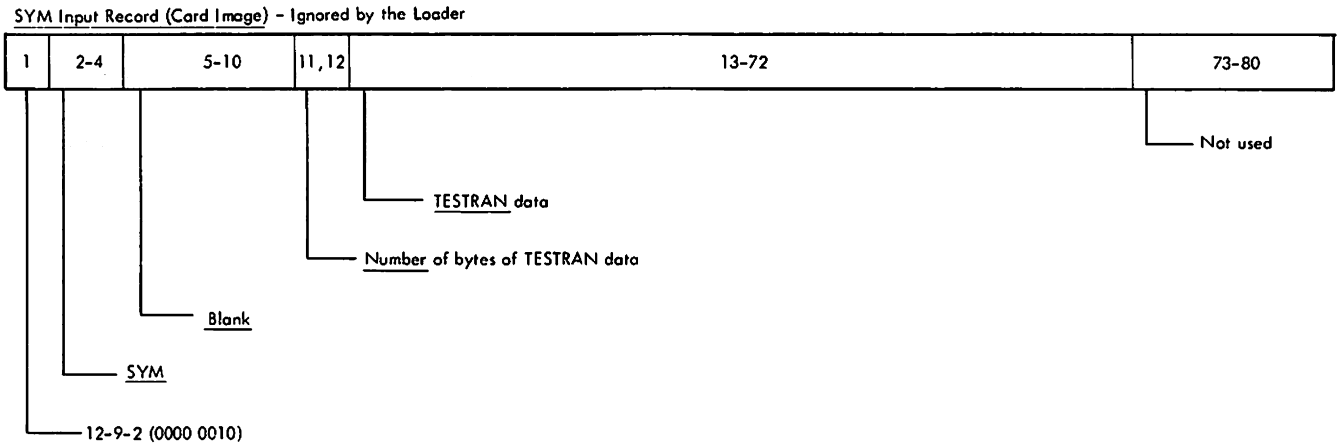


Figure 46. SYM Input Record (Card Image)—Ignored by the Loader

ESD Input Record (Card Image)

1	2-4	5-10	11,12	13,14	15,16	17-72	73-80

- 73-80: Not used
- 17-72: ESD Data -- see below
- 15,16: Blank if all ESD items are LD; ESD IDENTIFIER of first ESD item (other than LD)
- 13,14: Blank
- 11,12: Number of bytes of ESD data
- 5-10: Blank
- 2-4: ESD
- 1: 12-9-2 (0000 0010)

ESD Data Item

1-8	9	10-12	13	14-16

- 14-16:
  - Zero - if length is on END card.
  - Length of control section (if type is: SD, PC, CM)
  - Identifier of SD entry for LD or LR
  - Blank if type is ER, WX, or 06 for 'never-call' from PL/I compiler
  - Length of pseudo-register (PR)
- 13:
  - Alignment Factor (PR)
    - 07 – doubleword alignment
    - 03 – word alignment
    - 01 – halfword alignment
    - 00 – byte alignment
  - AMODE/RMODE/RSECT data (SD, PC)

X X X X	not used
. . . . R . . .	RSECT information (ignored)
. . . . . R . .	RMODE data 0 = 24 1 = ANY
. . . . . . A A	AMODE data 00, 01 = 24 10 = 31 11 = ANY

  - Blank (CD, ER, CM, NULL, WX)
- 10-12: 24-bit address (SD, PC, LD)
- 9: Type - Hex (00 SD, 01 LD, 02 ER, 04 PC, 05 CM, 06 PR, 07 = NULL, 0A = WX)
- 1-8:
  - Name -- when type is : SD, LD, LR, ER, CM, PR, WX
  - Blank -- when type is : PC or blank CM.

Figure 47. ESD Input Record (Card Image)

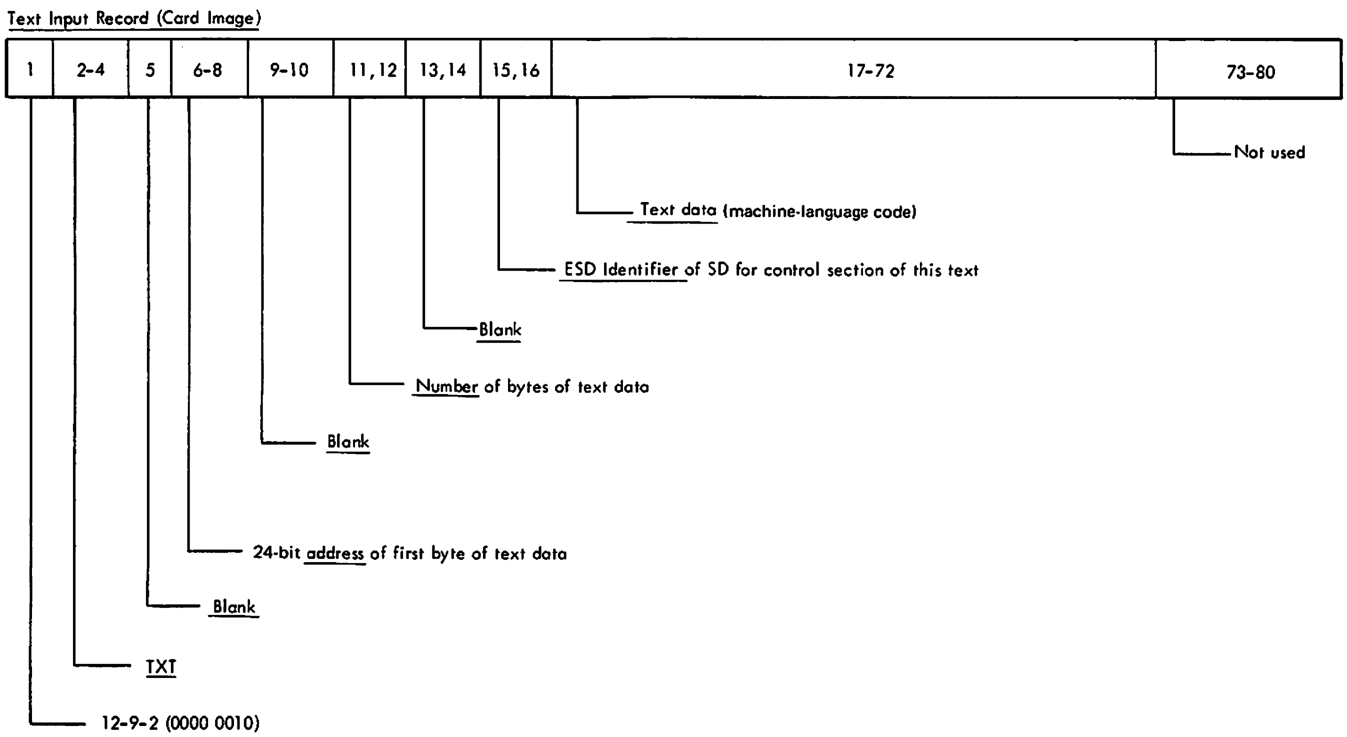


Figure 48. Text Input Record (Card Image)

RLD Input Record (Card Image)

1	2-4	5-10	11-12	13-16	17-72	73-80
12-9-2 (0000 0010)	RLD	Blank	Number of bytes of RLD data	Blank	RLD data - see below	Not used

RLD Data Item

1,2	3,4	5	6,7,8

- 6,7,8: Assigned address of address constant
- 5: Flag field -- (TTTTLLSTn)
  - TTTT = type
    - 0000 = non-branch
    - 0001 = branch
    - 0010 = pseudo register displacement value
    - 0011 = pseudo register cumulative length
  - LL = length of address constant
    - 01 = 2 bytes
    - 10 = 3 bytes
    - 11 = 4 bytes
  - S = Direction of relocation
    - 0 = positive (+)
    - 1 = negative (-)
  - Tn = type of next RLD item
    - 0 = next RLD item has a different R or P pointer; they are present in the next item.
    - 1 = next RLD item has the same R and P pointers, hence they are omitted
- 3,4: Position pointer (P) - ESDID of SD for control section that contains the address constant
- 1,2: Relocation pointer (R) - ESDID of CESD entry for the symbol being referred to. Zero (00) if type = PR cumulative length

Figure 49. RLD Input Record (Card Image)

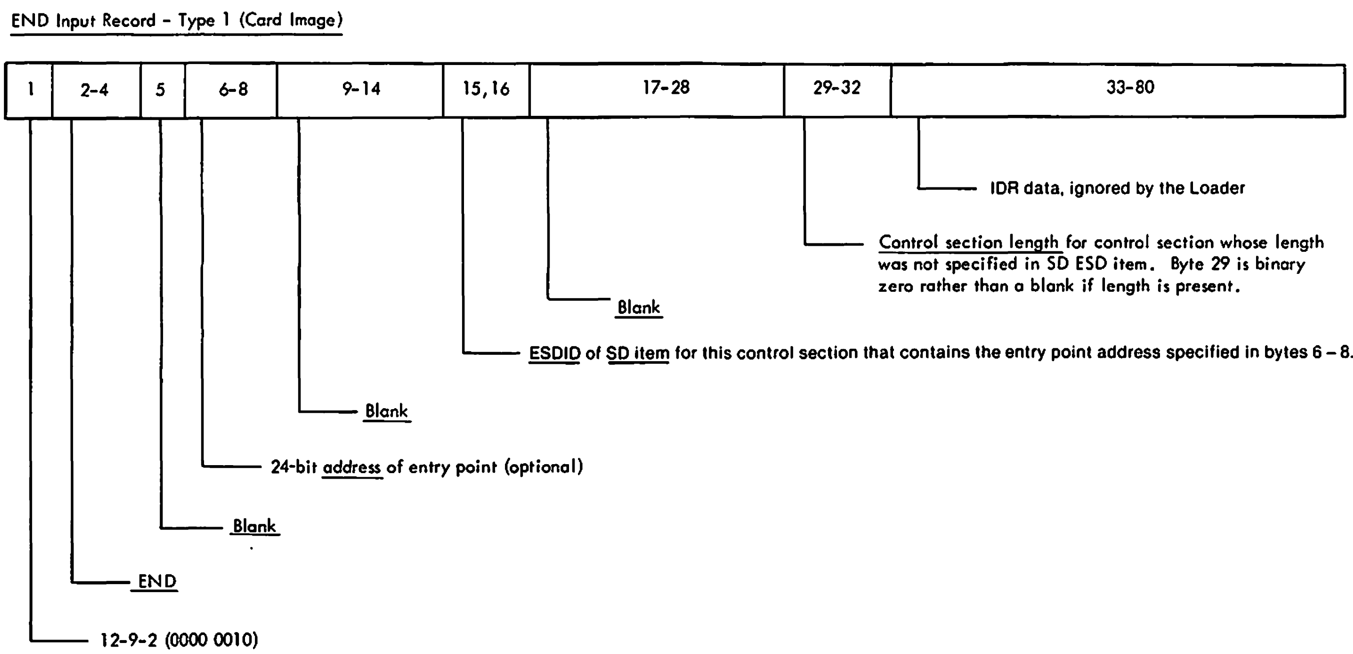


Figure 50. END Input Record—Type 1 (Card Image)

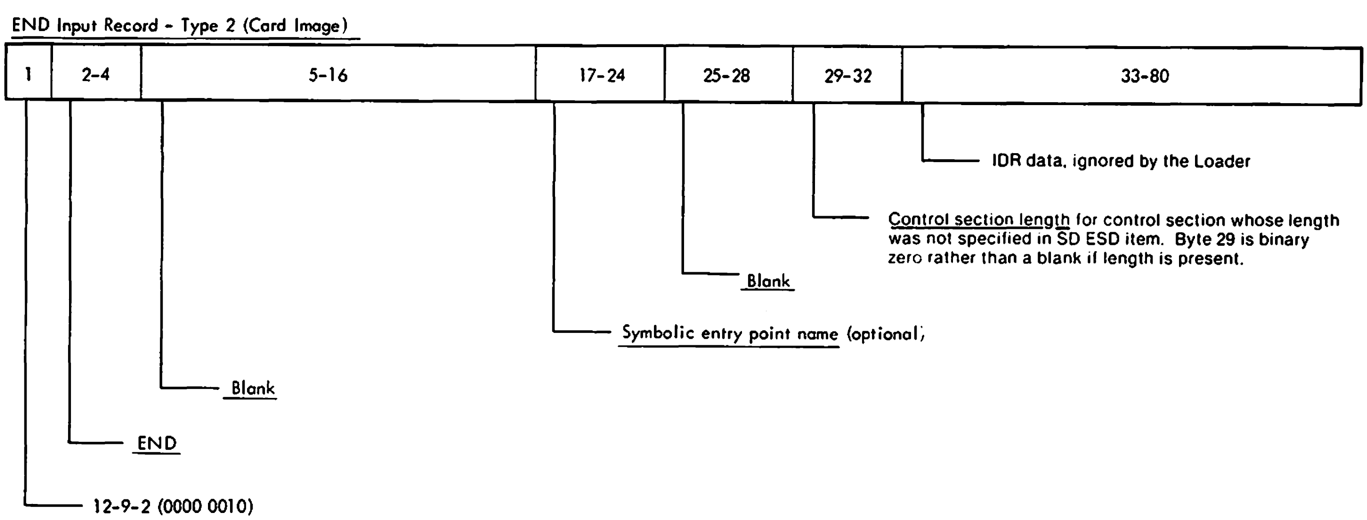


Figure 51. END Input Record—Type 2 (Card Image)

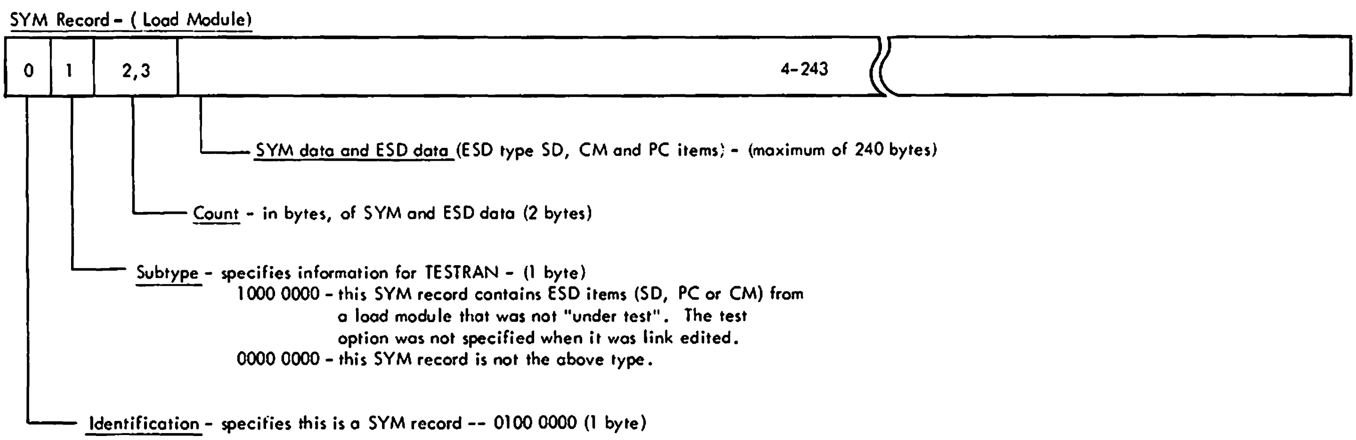


Figure 52. SYM Record (Load Module)—Ignored by the Loader

CESD Record - (Load Module)

0	1	2,3	4,5	6,7	8-247	up to 240 bytes of ESD data

ESD data - for detailed information see below.

Count - in bytes, of ESD data (2 bytes)

ESDID of first ESD item (2 bytes)

Spare - 2 bytes of binary zeros

Flag (1 byte)
0XXX XXXX — byte 12 of CESD data items contains segment numbers
1XXX XXXX — byte 12 of CESD data items contains AMODE/RMODE data

Identification -- 0010 0000 -- (1 byte)

CESD Data (Load Module)

1-8	9	10-12	13	14-16

ID/length - length (3 bytes), when type is: SD, PC, CM or PR
ID (2 bytes), when type is LR
Zero (3 bytes), when type is ER, WX, or Null

Alignment factor (PR)
07 – doubleword
03 – fullword
01 – halfword
00 – byte

Zero (ER, WX, Null)
If flag byte (byte 1) indicates CESD data items contain segment numbers - segment number (SD, PC, CM, LR)
If flag byte (byte 1) indicates CESD data items contain AMODE/RMODE data —

X X X X	not used
. . . . R . . .	RSECT information (ignored)
. . . . . R . .	RMODE data 0 = 24 1 = ANY
. . . . . . A A	AMODE data 00, 01 = 24 10 = 31 11 = ANY

(SD, PC)

Address - linkage editor-assigned address of this symbol. Zero when type is ER, WX, or Null (3 bytes).

Type - (1 byte)

Section definition (SD)	XXXXX000
Label reference (LR)	XXXXX011
Private code (PC)	XXXXX100
Private code marked delete (ENTAB and SEGTAB control sections)	XXX1X100
Common (CM)	XXXXX101
Null	XXXXX111
External reference (ER)	XXXXX010
Weak external reference (WX)	XXXX1010
Pseudo register (PR)	XXXXX110 X's may be 1 or 0

Symbol - The 8-character external name — Zero when type is Null.
Blanks if blank common or PCs other than SEGTABs and ENTABs

Figure 53. CESD Record (Load Module)

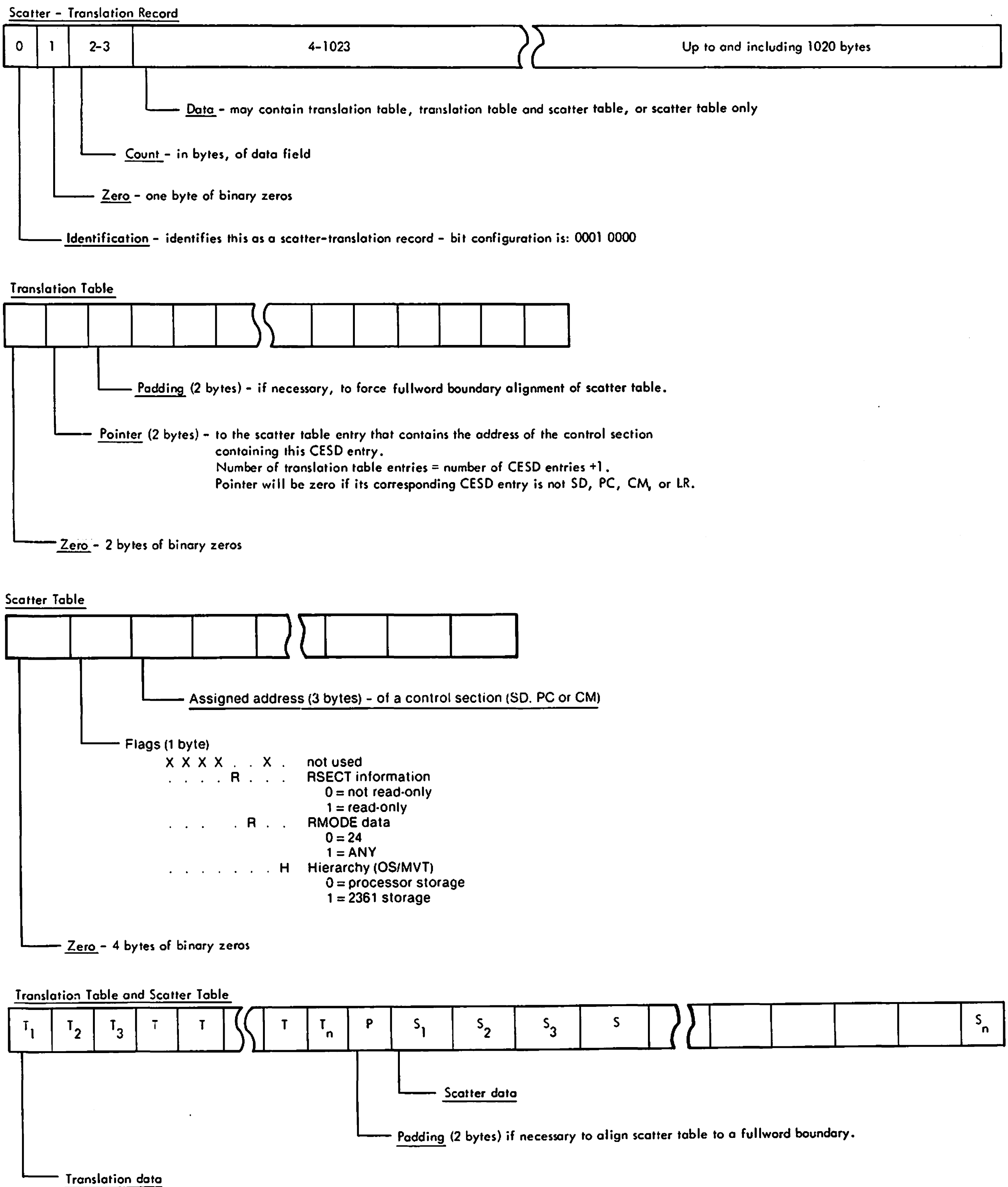


Figure 54. Scatter/Translation Record—Ignored by the Loader

Control Record - (Load Module)

0	1-2	3	4-5	6-7	8-15				Record Length 20 to 256 bytes

Length of text record and/or length of control section - specifies the length of the control section (in bytes) to which the text in the following record belongs, or the number of bytes of a control section contained in the following text record (2 bytes)

CESD entry number - specifies the composite external symbol dictionary entry that contains the control section name of the control section of which this text is a part (2 bytes)

Channel Command Word (CCW) - that could be used to read the text record that follows. The data address field contains the linkage editor assigned address of the first byte of text in the text record that follows. The count field contains the length of the succeeding text record.

Count – contains 2 bytes of binary zeros.

Count - in bytes, of the control information (CESD ID, length of control section) following the CCW field.

Count – (1 byte) of RLD and/or CTL/RLD records following next text record.

Spare – contains 2 bytes of binary zeros.

Identification - specifies that this is:

- A control record - 0000 0001
- The control record that precedes the last text record of this overlay segment - 0000 0101 (EOS)
- The control record that precedes the last text record of the module - 0000 1101 (EOM)
(1 byte)

Figure 55. Control Record (Load Module)

Relocation Dictionary Record – (Load Module)

0	1-3	4,5	6,7	8-15	16-255	Record length can be between 24 and 256

RLD data -- see below

Spare - contains 8 bytes of binary zeros

Count - in bytes of the relocation dictionary information following the spare 8 byte field (2 bytes)

Count - contains two bytes of binary zeros

Spare - contains three bytes of binary zeros

Identification - specifies that this is: (1 byte)
- A relocation dictionary record - 0000 0010
- The last record of this segment - 0000 0110
- The last record of the module - 0000 1110

RLD Data

R	P	F	A	F	A			F	A	R	P	F	A	R	P	F	A

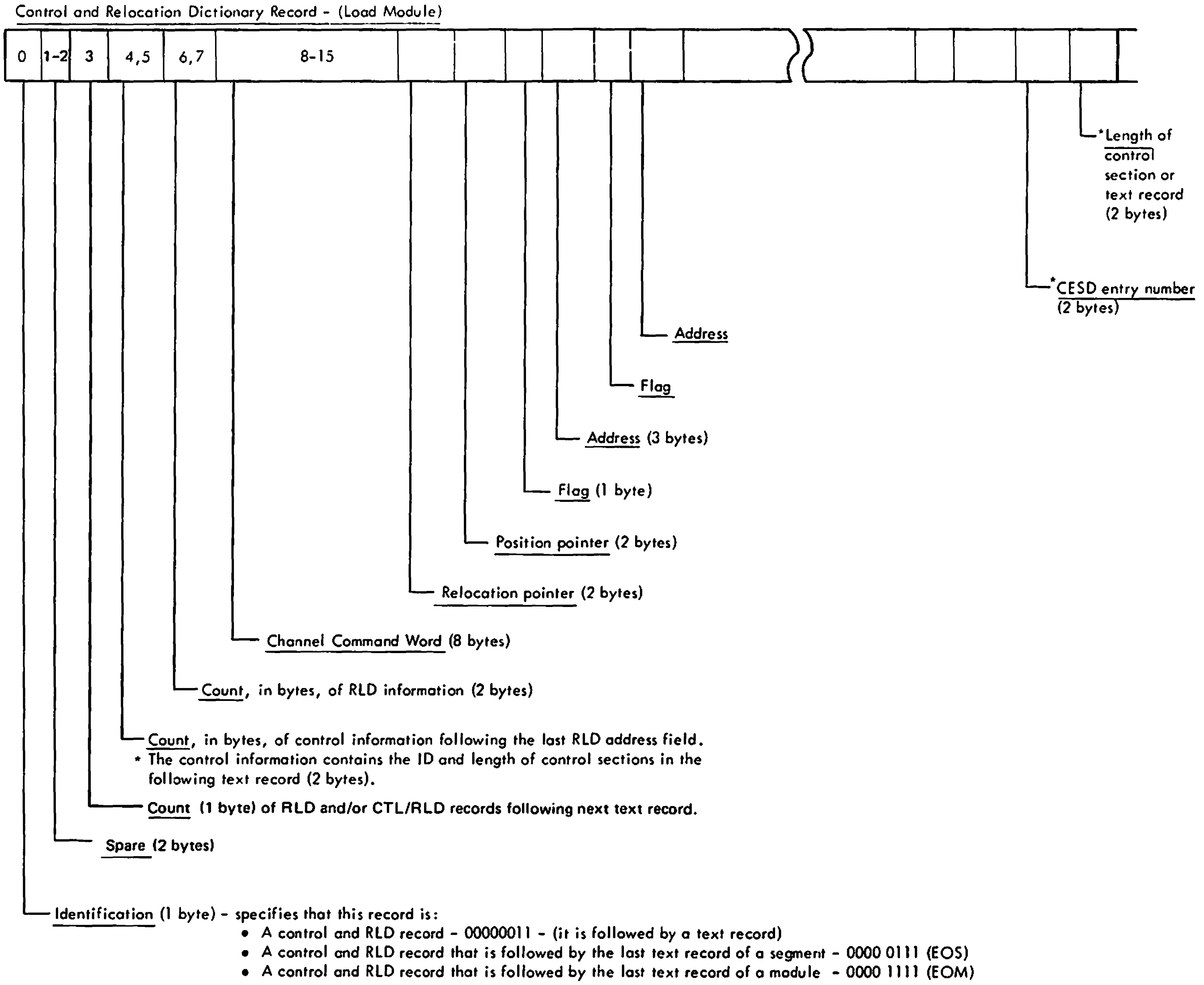
Address - linkage editor assigned address of the address constant (3 bytes)

Flag - (1 byte) When byte format is xxxxLLST, specifies miscellaneous information as follows:
xxxx specifies the type of this RLD item (address constant).
0000 -- non-branch type in assembler language, DC A (name)
0001 -- branch type in assembler language, DC V (name)
0010 -- pseudo register displacement value
0011 -- pseudo register cumulative displacement value
1000 and 1001 -- this address constant is not to be relocated because it refers to an unresolved symbol.
LL specifies the length of the address constant.
01 -- two bytes
10 -- three bytes
11 -- four bytes
S specifies the direction of relocation.
0 -- positive
1 -- negative
T specifies the type of the next following RLD item.
0 -- the following RLD item has a different relocation and/or position pointer.
1 -- the following RLD item has the same relocation and position pointers as this and therefore contains only the flag and address fields.

Position pointer - contains the entry number of the CESD entry that indicates which control section holds the address constant (2 bytes).

Relocation pointer - contains the entry number of the CESD entry that indicates which symbol value is to be used in the computation of the address constant's value (2 bytes).
0 if PR cumulative length or if ENTAB CSECT.

Figure 56. 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 256 bytes.

Figure 57. Control and Relocation Dictionary Record (Load Module)

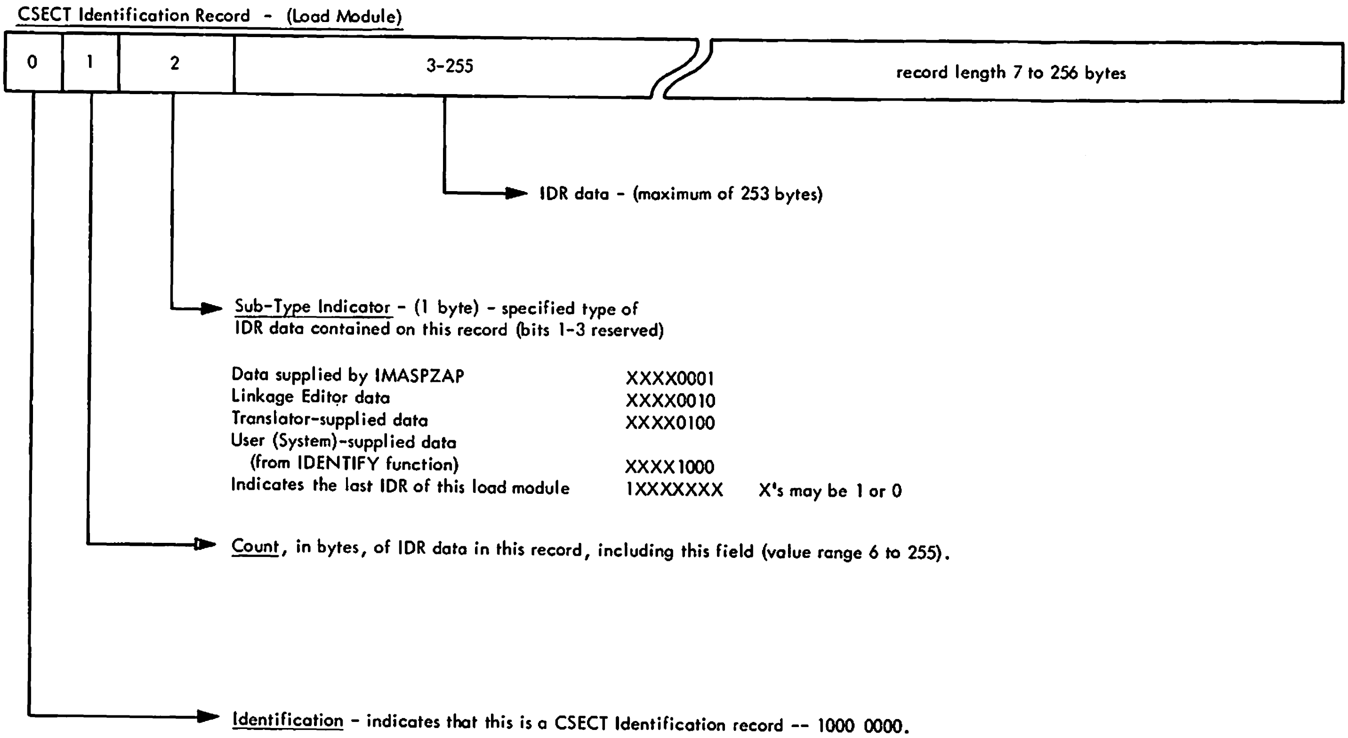


Figure 58. Record Format of IDRs (Load Module)—Ignored by the Loader

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 virtual storage and consisting of contiguous object module records prepared by a compiler) and/or an open SYSLIB data set, use the compiler/loader interface described here. The description includes the format of the DCB list, the control block or DCB parameters that 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

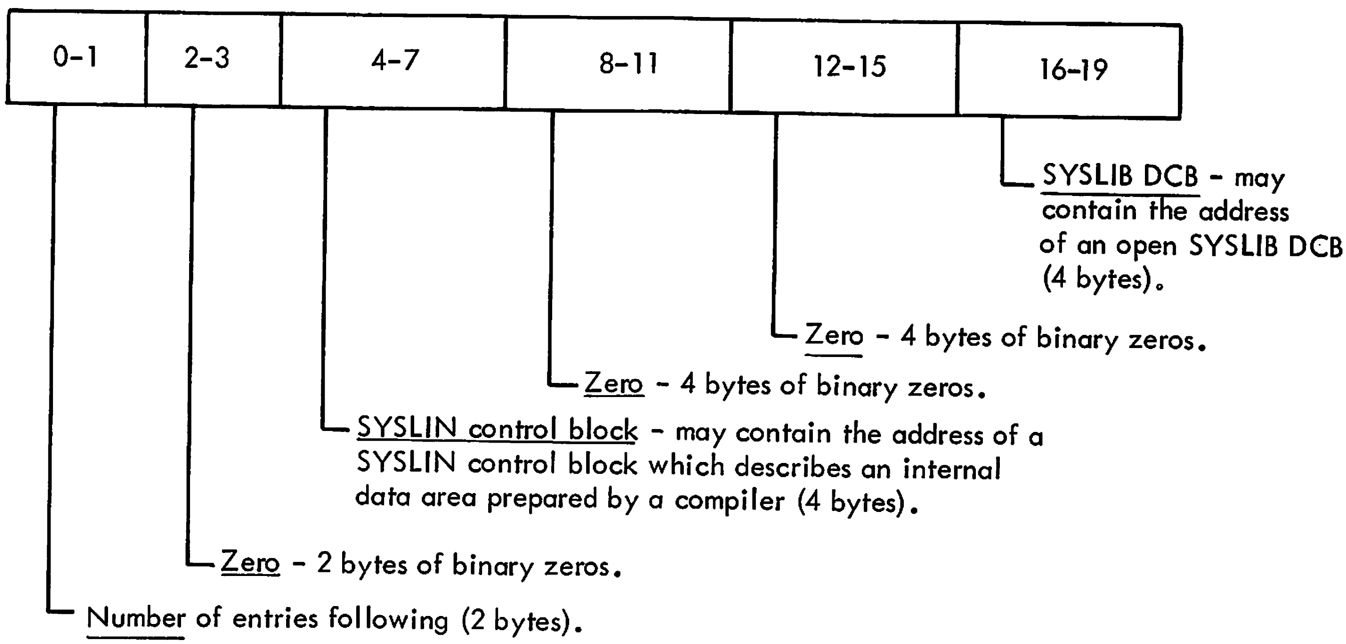


Figure 59. DCB List

Internal SYSLIN Control Block

The SYSLIN control block[23] 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.

[23] 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.

Open SYSLIB DCB

The open SYSLIB DCB 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.

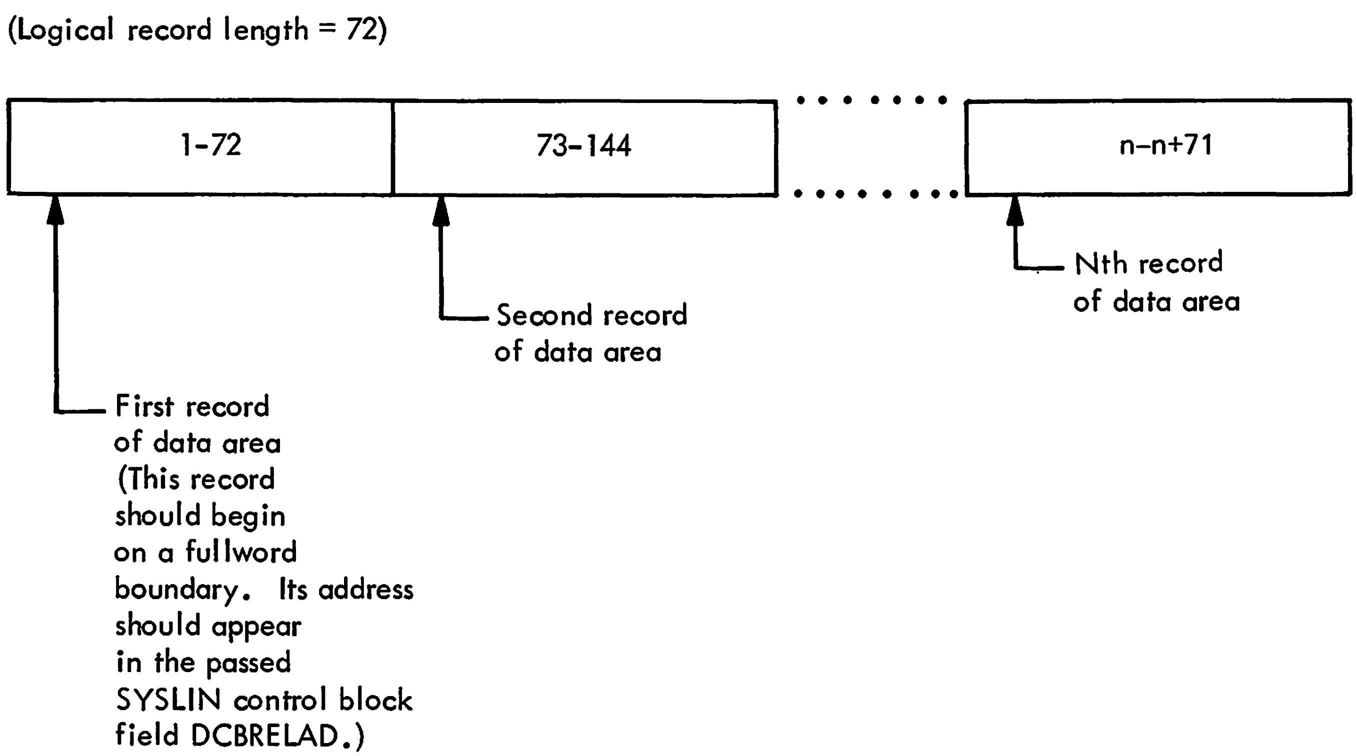


Figure 60. Internal Data Area in Fixed-Length Record Format

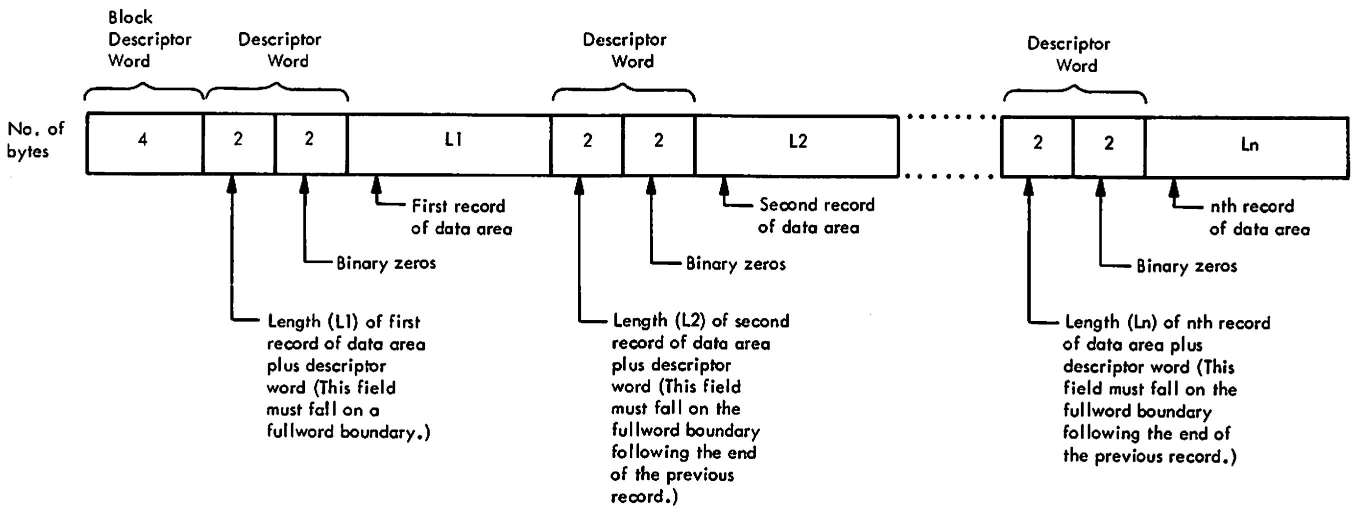
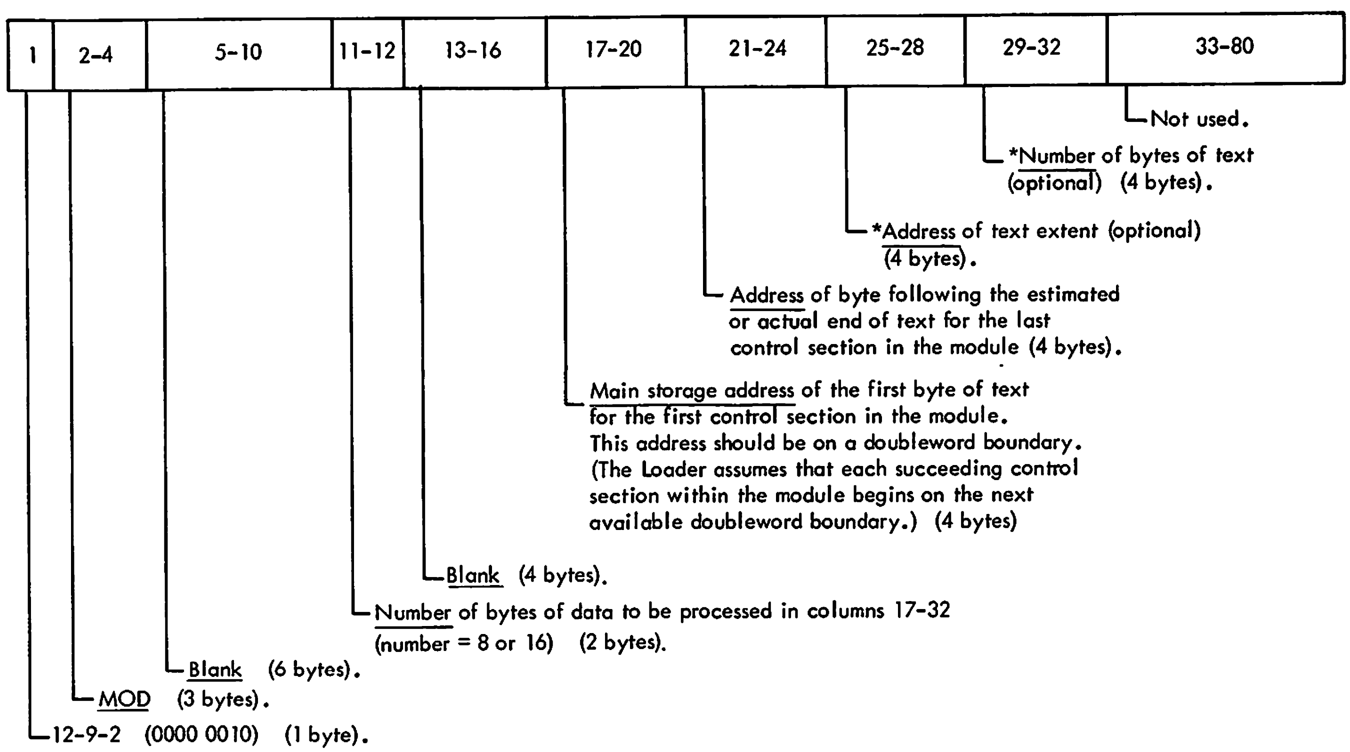


Figure 61. Internal Data Area in Variable-Length Record Format



*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, HEWLDRGO, or HEWLOAD. 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 62. MOD Record (Card Image)

IDENTIFY MACRO INSTRUCTION—IDENTIFYING LOADED PROGRAM

The IDENTIFY macro instruction, when invoked as described below, permits the loader to describe a program constructed in subpool zero (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 supervisor to identify the program.

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 35 on page 90.) 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 by the LOAD macro instruction. Residence of the program in subpool zero (0) and the absence of the program as a load module on an external device are 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))

where:

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 35 on page 90.) The address of the parameter list can be loaded into register one (1), in which case MF=(E,(1)) should be coded. If the address is not loaded into register one (1), it can be coded as an address that is valid in an RX-type instruction, or as one of the registers two (2) through 12 that 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 eight (8).

2. The addresses in the parameter list must be in subpool zero (0).

3. The program name should not duplicate a name already on either the link pack area control queue, or the user's job pack area control queue.

4. The entry point must lie within one of the extents.

5. The caller must be a nonsupervisory routine.

6. The extents must be found in the user's region in subpool zero (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 virtual storage; CDE was not created.
0C	Entry point address is not within an eligible program; CDE was not created.
14	An IDENTIFY macro instruction was previously issued using the same program name, but with 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 eight (8), or the extent addresses are not on doubleword boundaries; CDE was not created.
20	Extents are not in subpool zero (0); CDE was not created.

LIST OF TERMS AND ABBREVIATIONS

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
K	1024
LD	label definition
LR	label reference
P pointer	position pointer
PC	private code
PR	pseudo register
R pointer	relocation pointer
RLD	relocation dictionary
SD	section definition
TTR	relative track and record address on a direct-access device
WX	weak external reference

INDEX

F

G

H

I

L

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

LY26-3901-1

Reader's Comment Form

MVS/XA Loader Logic

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: ______________________

Chapter/Section ____

______________________________________ Page No. ______________

Comments:

If you want a reply, please complete the following information.

Name ______________________________ Phone No. (____)______________

Company ____

Address ____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

LY26-3901-1

Reader's Comment Form

Fold and tape

Please do not staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape

Please do not staple

Fold and tape

IBM®

MVS/Extended Architecture
Loader Logic

File Number S370-31

LY26-3901-01