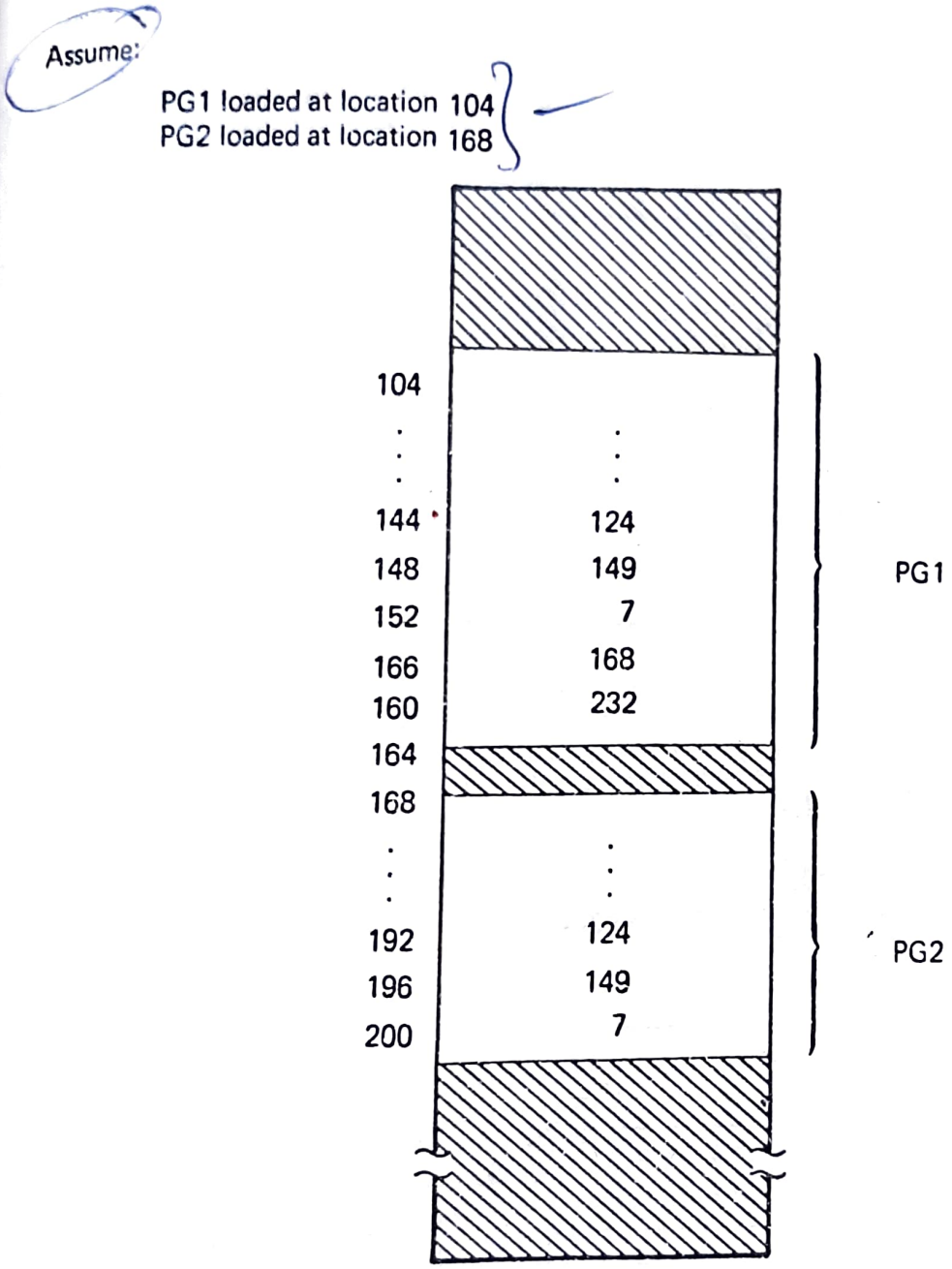


FIGURE 5.15 Main storage after loading programs PG1 and PG2

Pass 2 data bases:

1. Copy of object programs inputted to pass 1
2. The Initial Program Load Address parameter (IPLA)
3. The Program Load Address counter (PLA)
4. The Global External Symbol Table (GEST), prepared by pass 1, containing each external symbol and its corresponding absolute address value
5. An array, the Local External Symbol Array (LESA), which is used to establish a correspondence between the ESD ID numbers, used on ESD and RLD cards, and the corresponding external symbol's absolute address value

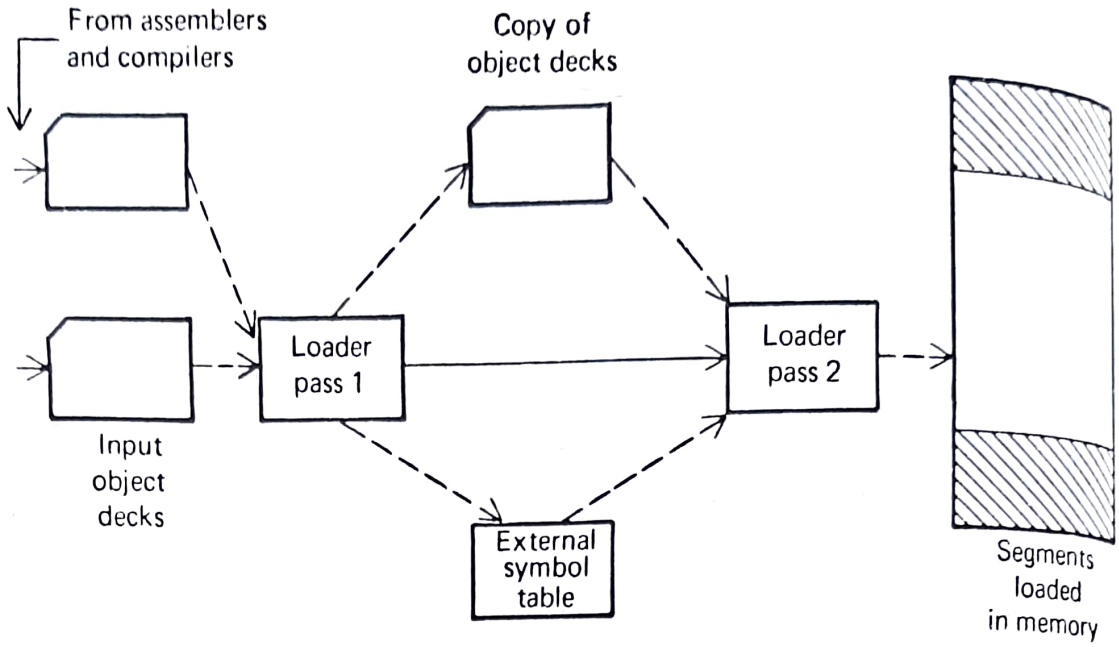


FIGURE 5.16 Two pass direct-linking loader scheme

5.3.3 Format of Data Bases

The third step in our design procedure is to specify the format and content of each of the data bases. The major data bases are depicted in Figure 5.17.

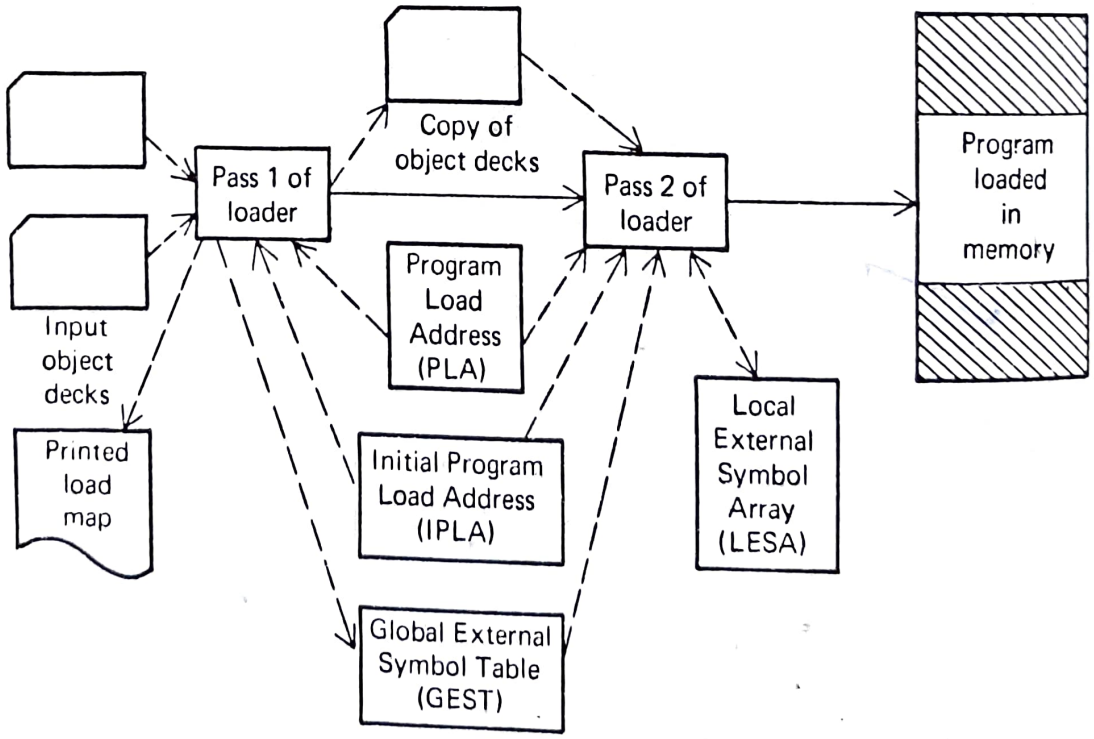


FIGURE 5.17 Use of data bases by loader passes

OBJECT DECK

The object deck has been discussed several times; Figures 5.18, 5.19, 5.20 and 5.21, depict in detail the actual card deck format that is used by various IBM 370 or 360 direct-linking loaders. The specific card format is not crucial, but these figures present a good example of the various techniques used to encode information.

GLOBAL EXTERNAL SYMBOL TABLE

The Global External Symbol Table (GEST) is used to store the external symbols defined by means of a Segment Definition (SD) or Local Definition (LD) entry on an External Symbol Dictionary (ESD) card. When these symbols are encountered during pass 1, they are assigned an absolute core address; this address is stored, along with the symbol, in the GEST as illustrated in Figure 5.22.

The reader may wish to review the discussion on symbol tables and searching/sorting techniques presented in Chapter 3 in conjunction with the design of an

Columns	*Contents*
1	Hexadecimal byte X'02' (card punch 12-2-9)
2-4	Characters ESD
5-14	Blank
15-16	ESD identifier (ID) for program name (SD) or external symbol (ER) (IDs unique within program) or blank for entry (LD), see box below
17-24	Name, padded with blanks
25	ESD type code (TYPE)
26-28	Relative address or blank (ADDR), see box below
29	Blank
30-32	Length of program otherwise blank (LENGTH)
33-72	Blank
73-80	Card sequence number

TYPE	*Hexadecimal code*
SD	01
LD	02
ER	03

ESD forms and conventions

	Type	*ID*	*ADDR*	*Length*
Program name (segment definition)	SD	01	Zero	Length of program
Entry (local definition)	LD	—	Relative address	—
External reference	ER	Unique number	—	—

(The unique ID numbers are usually assigned sequentially.)

FIGURE 5.18 ESD card format

Columns	*Contents*
1	Hexadecimal byte X'02'
2-4	Characters TXT
5	Blank
6-8	Relative address of first data byte (ADDR)
9-10	Blank
11-12	Byte Count (BC) = number of bytes of information in cc. 17-72
13-16	Blank
17-72	From 1 to 56 data bytes (instructions and "data" look the same)
73-80	Card sequence number

FIGURE 5.19 TXT card format

Columns	*Contents*
1	Hexadecimal byte X'02'
2-4	Characters RLD
5-18	Blank
19-20	ID corresponding to a number assigned to SD or ER on ESD card
21	Flag byte (see box below)
22-24	Relative address of first byte of address constant (ADDR)
25-72	Blank
73-80	Card sequence number

Flag byte conventions

bits	
0-3	Not used
4-5	Length (in bytes) of address constant 00 = one byte 01 = two bytes 10 = three bytes 11 = four bytes
6	0 means add ESD address to address constant 1 means subtract ESD address from address constant

FIGURE 5.20 RLD card format

Columns	*Contents*
1	Hexadecimal byte X'02'
2-4	Characters END
5	Blank
6-8	Start of execution entry (ADDR), if other than beginning of program (specified on assembly END card)
9-72	Blank
73-80	Card sequence number

FIGURE 5.21 END card format

← 12 bytes per entry →

External symbol (8-bytes) (characters)	Assigned core address (4-bytes) (decimal)
"PG1*bbbbb*"	104
"PG1ENT1*b*"	124
"PG1ENT2*b*"	134
"PG2*bbbbb*"	168
"PG2ENT1*b*"	184

Notation: This sample GEST content is based upon the example in Figures 5.12 and 5.15.

FIGURE 5.22 Global External Symbol Table (GEST) format

assembler. The GEST has the same general use and characteristics as the assembler's Symbol Table.

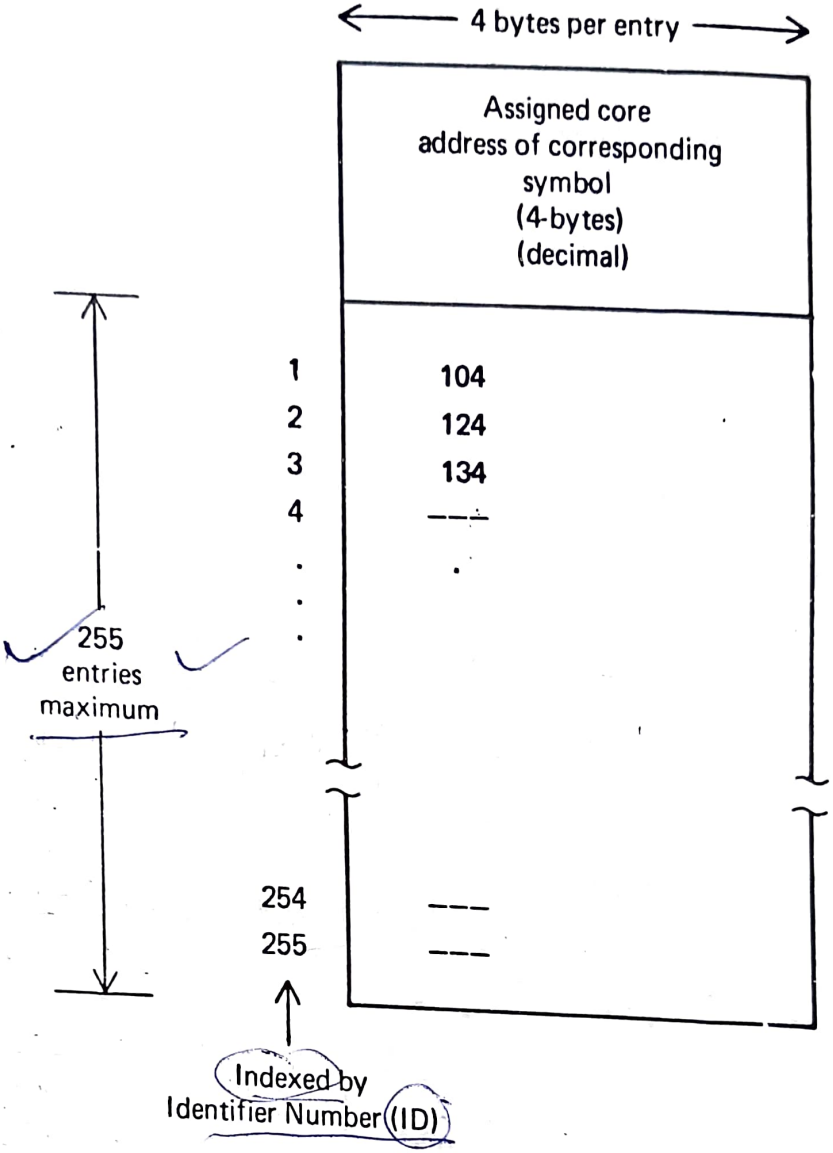
LOCAL EXTERNAL SYMBOL ARRAY

As mentioned earlier, the external symbol to be used for relocation or linking is identified on the RLD cards by means of an ID number rather than the symbol's name. The ID number must match an SD or ER entry on the ESD cards. This technique both saves space on the RLD cards and speeds processing by eliminating many searches of the Global External Symbol Table.

It is necessary to establish a correspondence between an ID number on an RLD card and the absolute core address value. The ESD cards contain the ID numbers and the symbols they correspond to, while the information relating these symbols to absolute core address values may be found in the GEST. In pass 2 of the loader the GEST and ESD information for each individual object deck is merged to produce the local external symbol array that directly relates ID number and value. In principle it is necessary to create a separate LESA for each segment, but since the LESAs are only produced one at a time, the same array can be reused for each segment. Figure 5.23 depicts the format of the Local External Symbol Array (LESA). Note that unlike the case with the GEST, it is not necessary to *search* the LESA; given an ID number, the corresponding value is written as LESA(ID) and can be immediately obtained.

5.3.4 Algorithm

The following two flowcharts (Figs. 5.24 and 5.25) describe an algorithm for a direct-linking loader for an IBM System/360-type computer While they illustrate



Notation: This sample LESA contents is based upon the PG1 segment as presented in Figures 5.12, 5.13, and 5.15.

FIGURE 5.23 Local External Symbol Array (LESA) format

most of the logical processes involved, these flowcharts are still a simplification of the operations performed in a complex loader. In particular, many "special" features, such as COMMON segments, library processing, dynamic loading and dynamic linking, are not explicitly included (many of them are discussed in the problems section and in later chapters).

Pass 1 – allocate segments and define symbols The purpose of the first pass is to assign a location to each segment, and thus to define the values of all external symbols. Since we wish to minimize the amount of core storage required for the total program, we will assign each segment the next available location after the

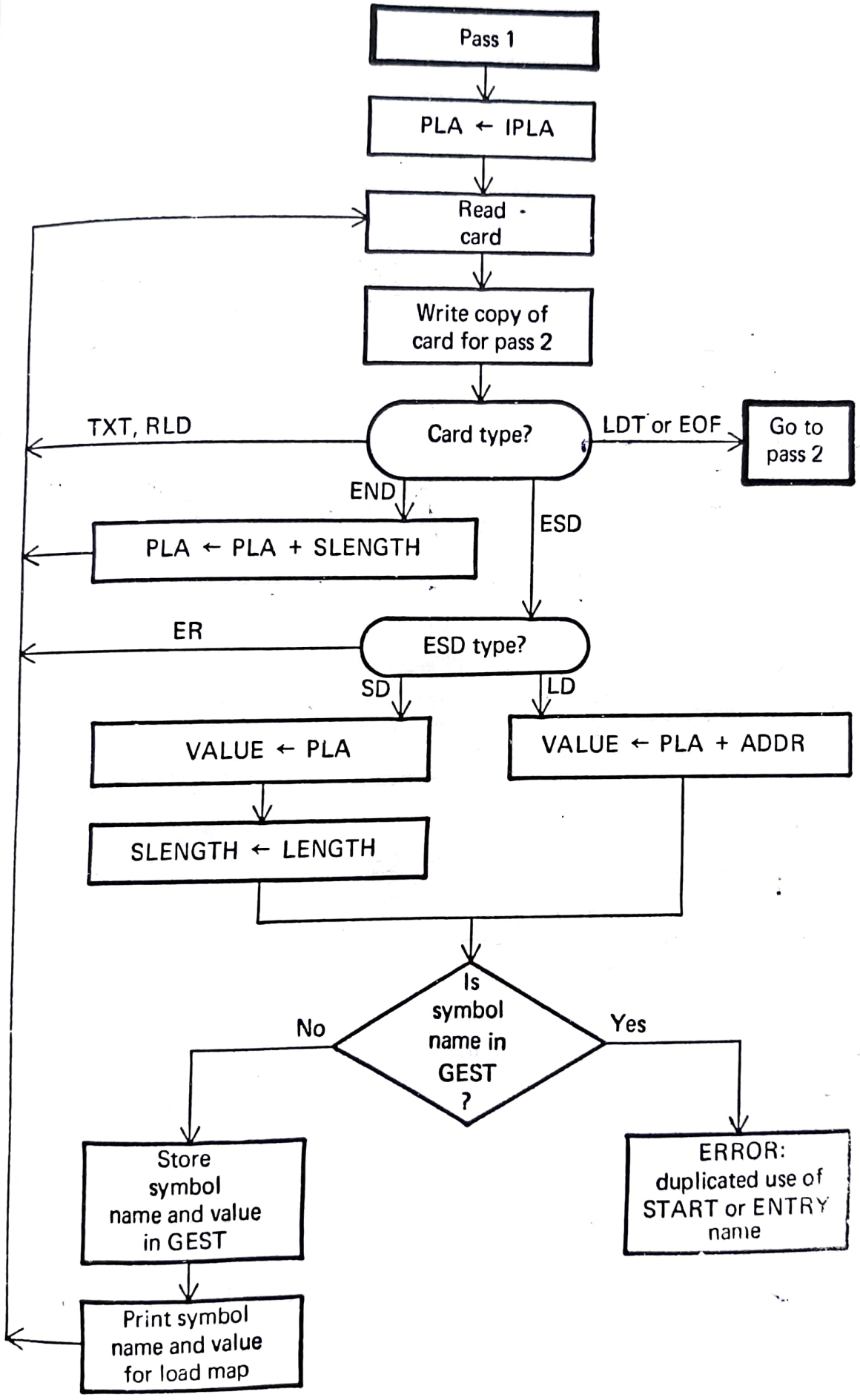


FIGURE 5.24 Detailed pass 1 flowchart

preceding segment. It is necessary for the loader to know where it can load the first segment. This address, the Initial Program Load Address (IPLA), is normally determined by the operating system. In some systems the programmer may specify the IPLA; in either case we will assume that the IPLA is a parameter supplied to the loader.

Initially, the Program Load Address (PLA) is set to the Initial Program Load Address (IPLA). An object card is then read and a copy written for use by pass 2. The card can be one of five types, ESD, TXT, RLD, END, or LDT/EOF. If it is a TXT or RLD card, there is no processing required during pass 1 so the next card is read. An ESD card is processed in different ways depending upon the type of external symbol, SD, LD or ER. If a segment definition ESD card is read, the length field, LENGTH, from the card is temporarily saved in the variable, SLENGTH. The value, VALUE, to be assigned to this symbol is set to the current value of the PLA. The symbol and its assigned value are then stored in the GEST; if the symbol already existed in the GEST, there must have been a previous SD or LD ESD with the same name – this is an error. The symbol and its value are printed as part of the load map. A similar process is used for LD symbols; the value to be assigned is set to the current PLA plus the relative address, ADDR, indicated on the ESD card. The ER symbols do not require any processing during pass 1. When an END card is encountered, the program load address is incremented by the length of the segment and saved in SLENGTH, becoming the PLA for the next segment. When the LDT or EOF card is finally read, pass 1 is completed and control transfers to pass 2.

Pass 2 – load text and relocate/link address constants After all the segments have been assigned locations and the external symbols have been defined by pass 1, it is possible to complete the loading by loading the text and adjusting (relocation or linking) address constants. At the end of pass 2, the loader will transfer control to the loaded program. The following simple rule is often used to determine where to commence execution:

1. If an address is specified on the END card, that address is used as the execution start address.
2. Otherwise, execution will commence at the beginning of the first segment.

At the beginning of pass 2 the program load address is initialized as in pass 1, and the execution start address (EXADDR) is set to IPLA. The cards are read one by one from the object deck file left by pass 1. Each of the five types of cards is processed differently, as follows:

ESD CARD

Each of the ESD card types is processed differently.

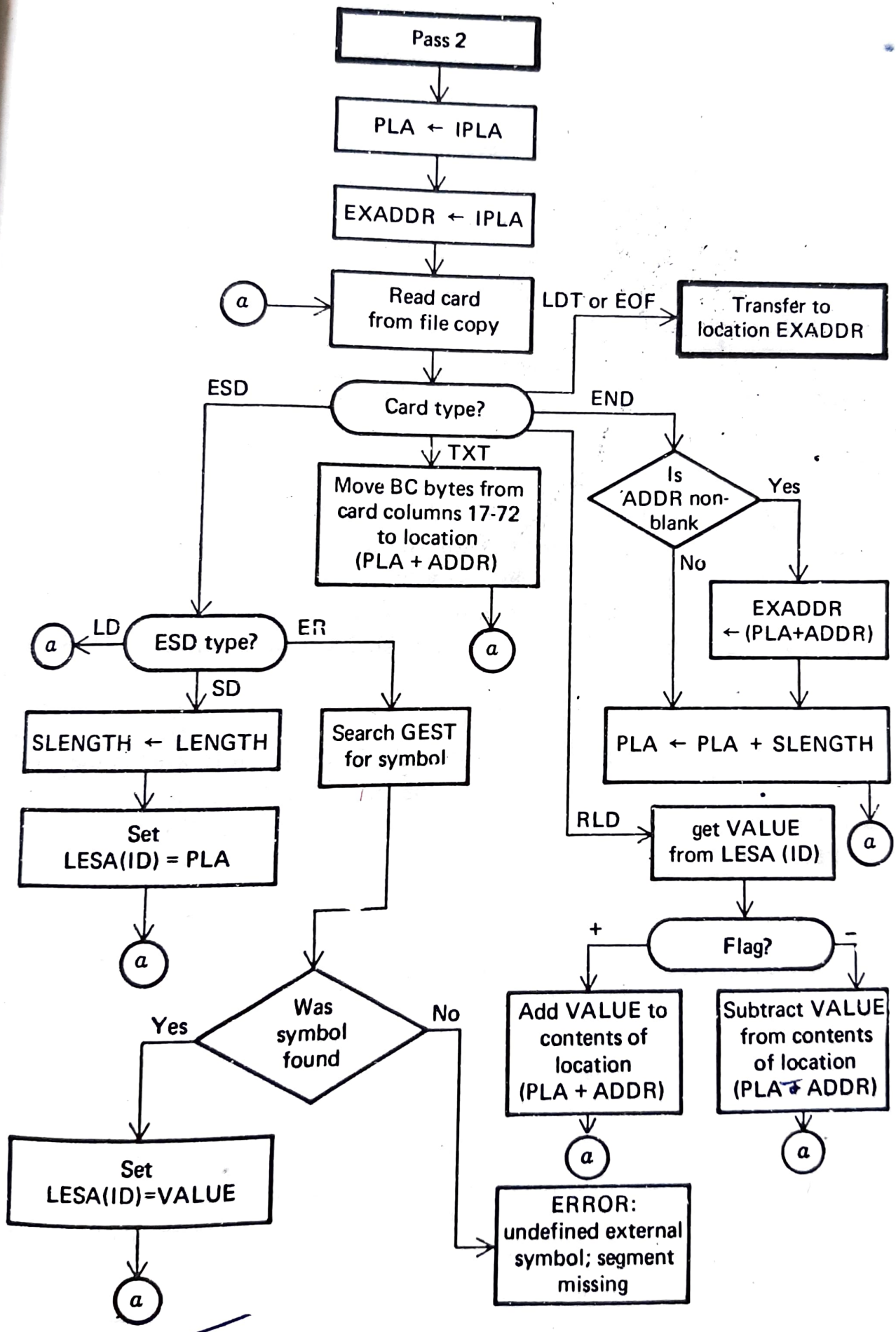


FIGURE 5.25 Detailed pass 2 flowchart

SD-type ESD The LENGTH of the segment is temporarily saved in the variable SLENGTH. The appropriate entry in the local external symbol array, LESA(ID), is set to the current value of the Program Load Address.

LD-type ESD The LD-type ESD does not require any processing during pass 2.

ER-type ESD The Global External Symbol Table (GEST) is searched for a match with the ER symbol. If it is not found, the corresponding segment or entry must be missing – this is an error. If the symbol is found in the GEST, its value is extracted and the corresponding Local External Symbol Array entry, LESA(ID), is set equal to it.

TXT CARD

When a TXT card is read, the text is copied from the card to the appropriate relocated core location (PLA + ADDR).

RLD CARD

The value to be used for relocation and linking is extracted from the local external symbol array as specified by the ID field, i.e., LESA(ID). Depending upon the flag setting (plus or minus) the value is either added to or subtracted from the address constant. The actual relocated address of the address constant is computed as the sum of the PLA and the ADDR field specified on the RLD card.

END CARD

If an execution start address is specified on the END card, it is saved in the variable EXADDR after being relocated by the PLA. The Program Load Address is incremented by the length of the segment and saved in SLENGTH, becoming the PLA for the next segment.

LDT/EOF CARD

The loader transfers control to the loaded program at the address specified by current contents of the execution address variable (EXADDR).

5.4 SUMMARY

The four basic functions of a loader are allocation, linking, relocation, and loading. The various types of loaders (e.g., "compile-and-go," absolute, relocating, direct-linking, dynamic loading, and dynamic linking) differ primarily in the manner in which the four basic functions are accomplished.

A typical direct-linking loader requires two passes. The first pass allocates space for the segments and defines the values of the external symbols. The

second pass actually loads the text and uses the global external symbol table produced by pass 1, to relocate and link the address constants.

Although their purposes are quite different, the design of the direct-linking loader has many similarities to the design of an assembler. In particular, the use of a symbol table is important in both cases.