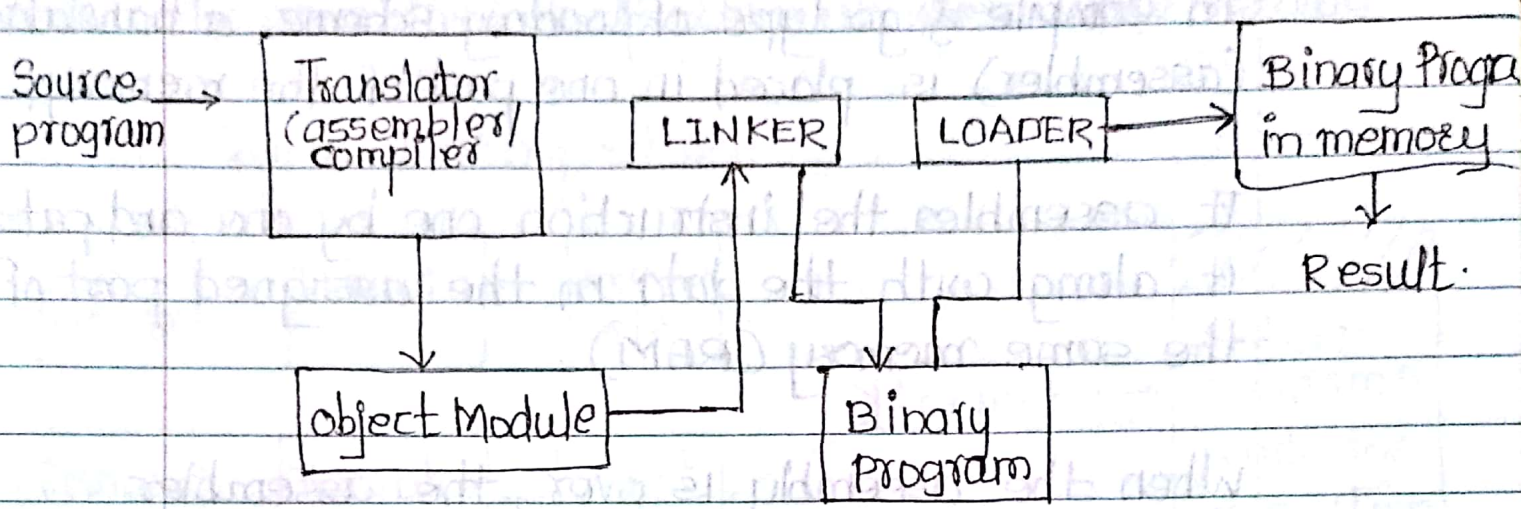
chapter - 04.

System Programming by John J. Donovan

Loaders and Linkers

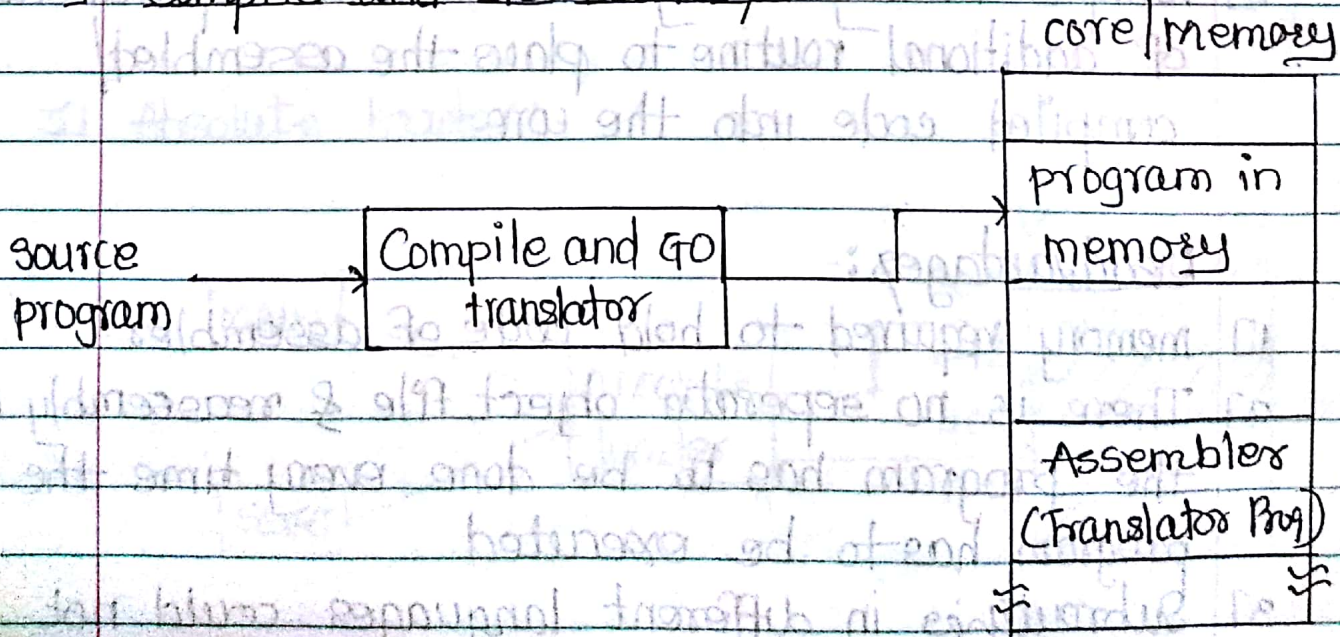


Functions of Loader :-

1) Program loading @ physically place the machine instructions & data into memory
2) Program allocation @ Allocate space in memory for the program
3) Program linking @ Resolve symbolic references between object decks
4) Relocation @ Adjust all address dependent locations, such as address constants, to correspond to the allocated space.

Loader Schemes:

1] Compile and Go Loaders:-



In Compile & go type of loading scheme, a translator (assembler) is placed in one part of the memory.

It assembles the instruction one by one and puts it along with the data in the assigned part of the same memory (RAM).

When the assembly is over, the assembler transfers control to the first instruction.

Assembler places the object code into the main memory and the loader loads the code as the first instruction of the program.

Advantages:-

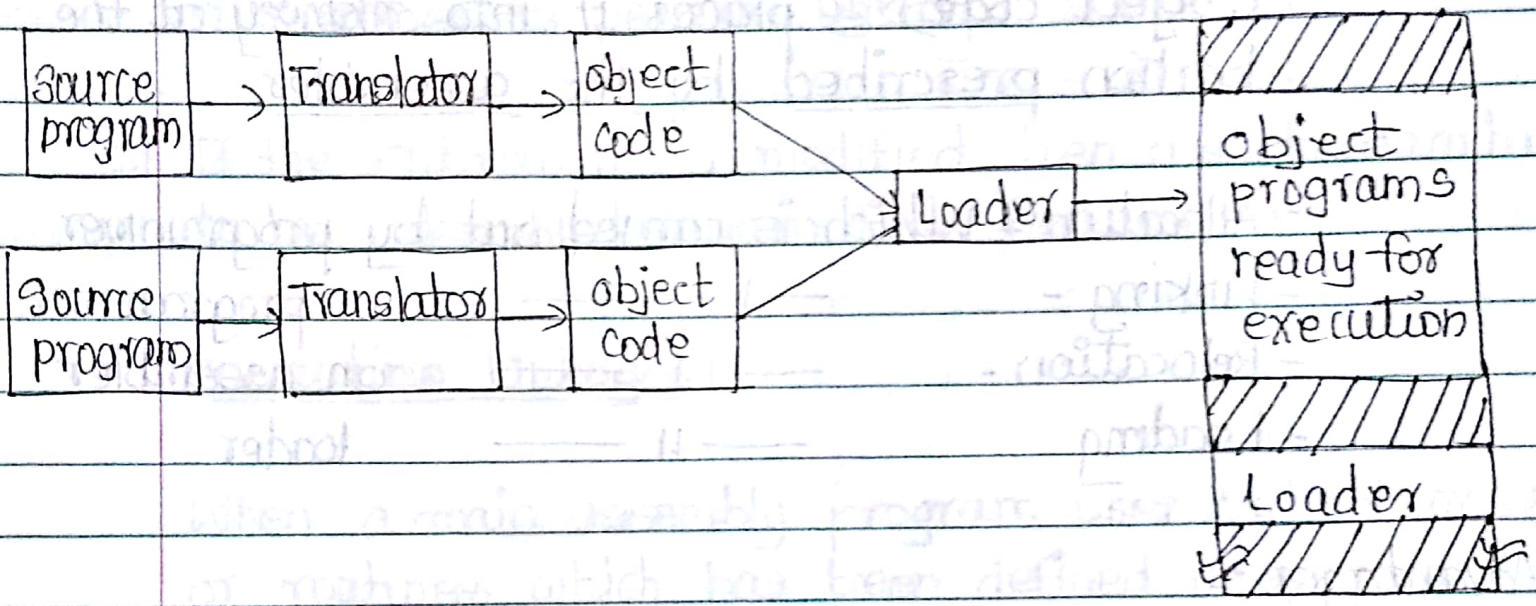
1] Simplest loading scheme
2] simple memory management with no requirement of additional routine to place the assembled/ compiled code into the core.

Disadvantages:-

1] memory required to hold code of assembler
2] There is no seperate object file & reassembly of the program has to be done every time the program has to be executed.
3] Subroutines in different languages could not

be combined to produce an executable module.

2] General Loading Scheme:-

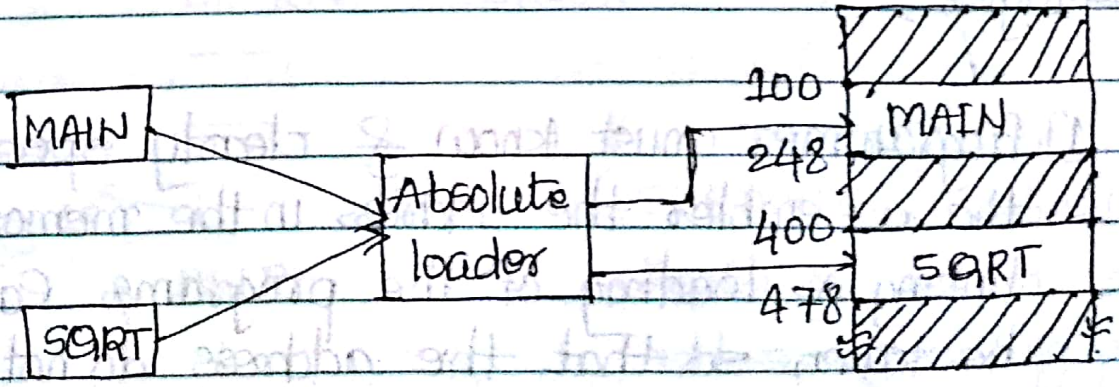


Advantages

1] Loader is assumed to be smaller than the assembler, so more memory is available to the user.

2] Re-assembly is no longer necessary to run the program at a later date.

3] Absolute Loader:-



- It follow the general loading scheme.

- The loader accept the machine language text (object code) & places it into memory at the location prescribed by the assembler.

- Allocation - which is carried out by programmer
- Linking - —11— programmer.
- Relocation - —11— an assembler
- Loading —11— loader.

Advantages :-

1] Simple to implement & efficient in execution
2] Saves the memory because the size of loader is smaller than the assembler.
3] Loader functioning is not complex in nature & it follow the instruction to place the desired object code in the main memory

Disadvantages :-

1] Programmer must know & clearly specify to the assembler the address in the memory for inner linking & loading of the programs. Care should be taken so that the address do not overlap.

2] For programs with multiple subroutines, the programm must remember the absolute address of each subroutine & use them explicitly in their other subroutines to perform linking.

3] If the subroutine is modified, then the program has to be assembled again from first to last.

Subroutine Linkages

When a main assembly program uses variable or sym or routines which has been defined in another assembly language program.

At this instant, the assembler for main progr generate an error (undefined symbol or routine), unless a special mechanism has been provided.

ex

```
MAIN   START
       EXTRN   SUBROUT.
       ---
       ---
       L       15, = A(SUBROUT)   } call subrout
       BALR    14,15              - Register 14 contain return
       :                            address of calling program.
       :
       END
```

↗ Instruction declared SUBRO as an external variable, i.e variable referenced but not defi in this program.

Loads address of SUBROUT in reg. 15 ↖

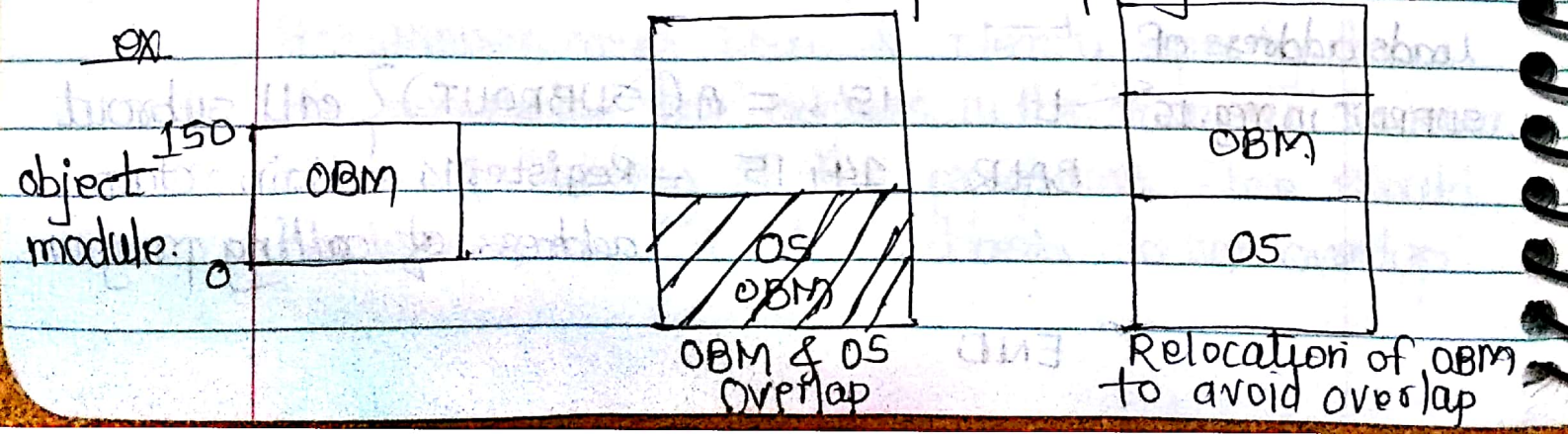
This mechanism is typically implemented with a relocating or a direct-linking loader. The assembler pseudo-op EXTRN followed by a list of symbols indicates that these symbols are defined in other programs but referenced in the present program.

- If a symbol is defined in one program & referenced in others, we insert it into a symbol list following the pseudo-op ENTRY.

4] Relocating Loader Scheme

In case of absolute loader when a single subroutine is modified the assembling, linking etc. are to be done again for the entire program. To avoid this relocating loader scheme has been introduce, In this scheme if specified memory for object module is free then that module will be loaded in memory. otherwise relocating loader will find next free location & place program in it.

Ex



example of relocating loader is Binary Symbolic Subroutine (BSS) loader.

In this scheme of loading, the assembler translate & assembles each subroutine & passes them onto loader together with the.

1] Translated text of each subroutine.
2] Info. about the relocation of each subroutine.
3] Intersegment references for each subroutine

The o/p of the BSS loader are.

1] The object program.
2] Information about all other program & its referenc
3] Information about locations that need to be changed, If the program is loaded at arbitrary location.

Design of Direct-Linking Loaders :-

- It is a general relocatable loader. It is most widely used scheme.

- It ~~allow~~ provide the programmers the flexibility of defining multiple subroutines & multiple segments in their program.

- The programmers also have full freedom of referencing data or instructions across routines & segments.

- The translator provides much information to the loader about each procedure or data segment but does not allocate space to segments nor defines the values of external symbols. Both these tasks are done by the loader.

- An assembler would generally provide information to the loader, including the length of segment, list of public Definitions & their relative location within the segment, list of external references, information about address constant & the translated machine code version of the source program & their relative addresses.

- Direct linking provides flexible Intersegment referenci for doing all this, DLL required following card

1] ESD (External Symbol Directory)

ESD ID	Length	Flag	Relative Address

ESD card contains information about all symbols that are define in a program that may be referred somewhere else & vice versa.

There are 3 types of symbols.

1] Segment Definition (SD) - It is name of the prog Which return prior to sta keyword.

2] Local Definition :-(LD) - This are the symbols whic are define in the program

3] External Reference (ER) - This are symbols which are referred in the program but are defining somewhere else.

ID - Giving unique no. to all segment definition & external reference.

Relative address:- It is address at which those symbols are define.

Length :- Size of the symbol.

2] TXT card

Count	Relative address	content

- It contains actual object code which is translated version of source program.

3] RLD (Relocation & Linkage Directory card)

Symbol	Type	ID	relative Address	Length

It contains information about those location of the program whose contains depend on address at which the program is placed.

4] End card.

- It specifies that start of ~~execution~~ execute of the object program.

In a single pass we cannot find the runtime address of all the symbols therefore DLL is perform in 2 pass.

1] Pass - I Database:

1] Object card - This card contain object program in four cards.
i) ESD ii) TXT iii) RLD iv) END.

2] IPLA (Initial Program Load Address) - It is the address obtained by Loader from operating syste

3] PLA (Program Load Address) - used to keep track of each segment's assigned location.

4] GEST (Global External Symbol Table) - that is used to store each external symbol & its corresponding assigned core address.

5] A copy of the input to be used later by pass 2.

6] Load Map → It is printed listing of GEST table.

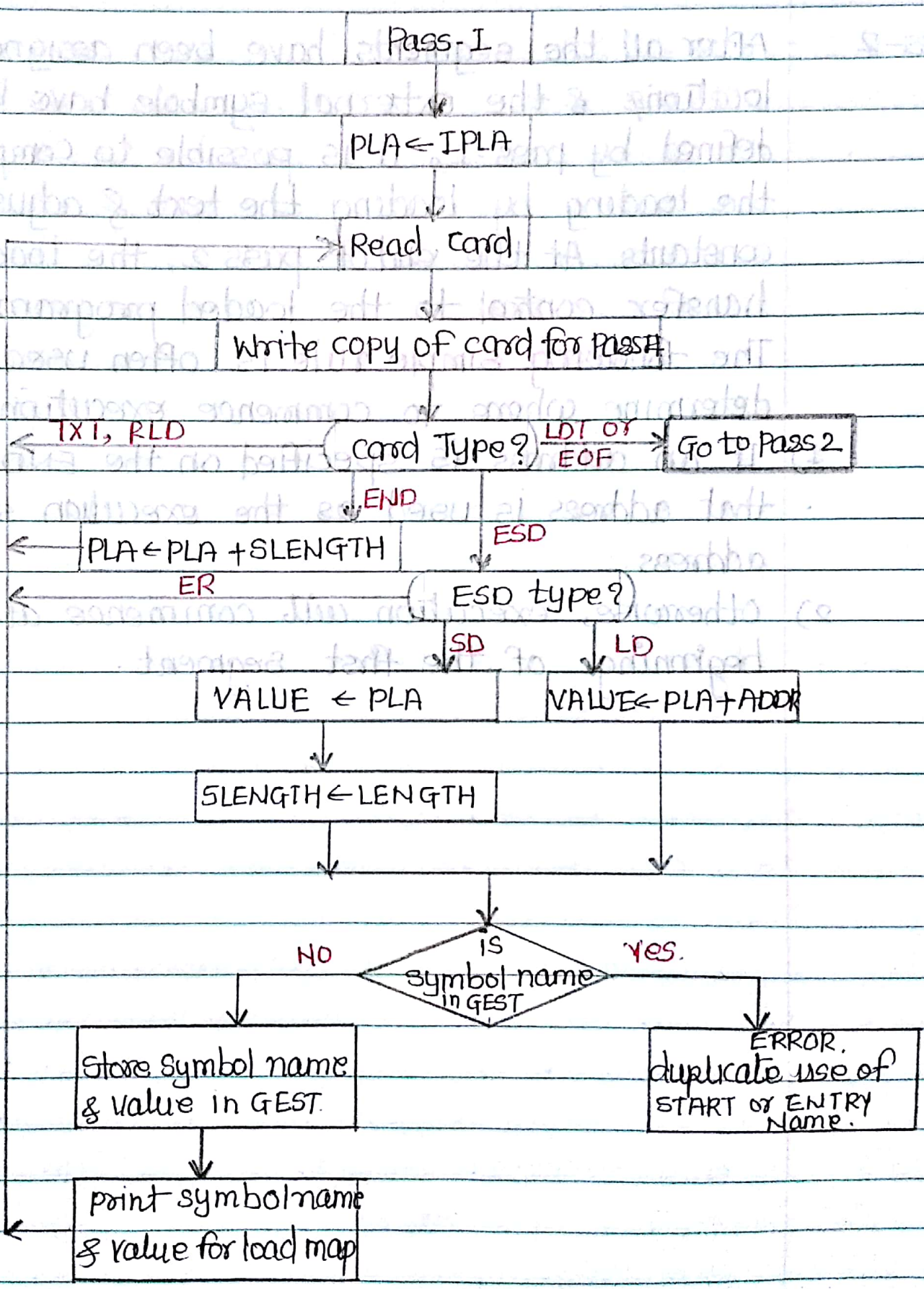
2] Pass-2 Databases :-

1) Copy of object programs inputted to pass 1
2] IPLA
3] PLA
4] Execution Address (EXADDR) - It indicates the locality from where the execution of object program should begin.
5] GEST (Global External Symbol Table)
6] Local External Symbol Array (LESA) - It is prepared with the help of ESD & GEST.

Algorithm :- (for an IBM system 360).

Pass-I :- The purpose of the first pass is to assign a location to each segment, & thus to define the values of all external symbols.

- It is necessary for the loader to know where it can load the first segment. This address, the IPLA is normally determined by OS. 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.



Pass-2 - After all the segments have been assigned locations & the external symbols have been defined by pass-1, it is possible to complete the loading by loading the text & adjusting addr 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.