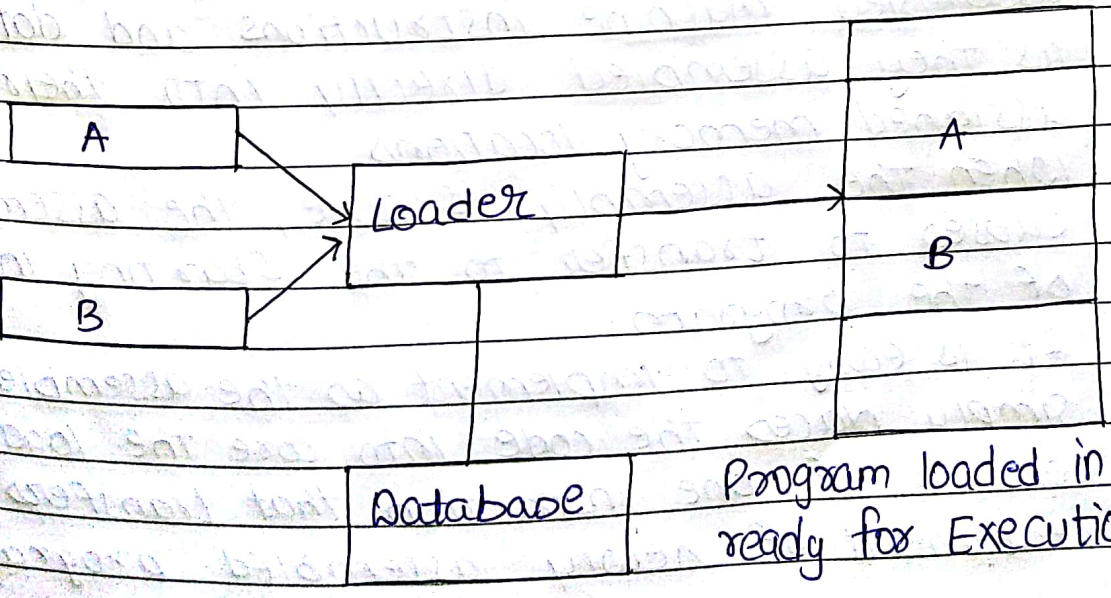
CHAPTER 3 : LOADERS AND LINKERS

- A loader is a program/software which accepts the object/target code by assembler and compiler and intiated the execution

Four basic function of a Loader

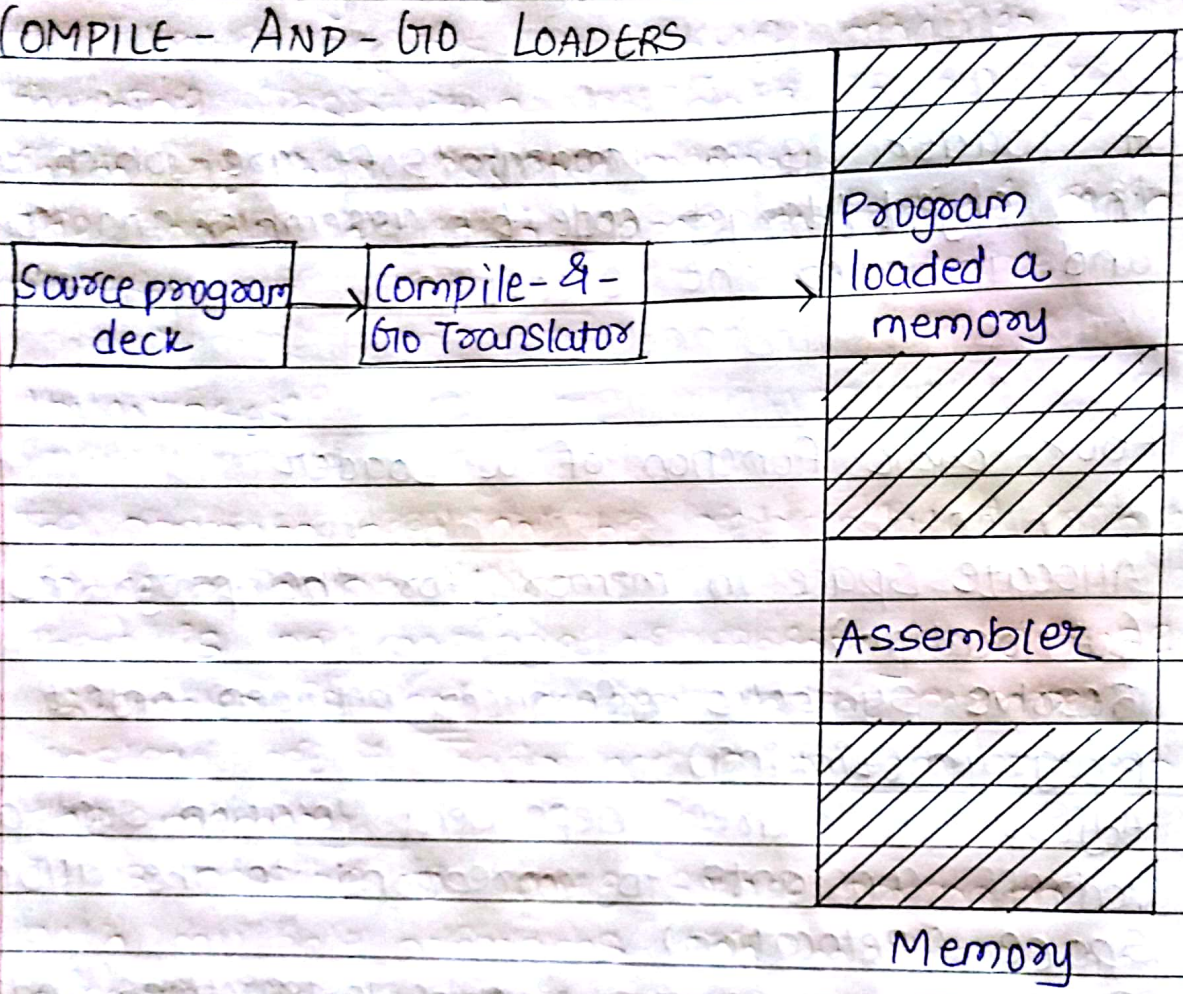
- Allocate space in memory for the programs (Allocation)
- Resolve symbolic references between object program (Linking)
- Adjust all address dependent location such as address constants to correspond to the allocated space (Relocation)
- Physically place the machine instructions and data into the memory (Loading)



Program loaded in memory ready for Execution.

LOADER SCHEMES

1] COMPILE - AND - GO LOADERS



- The assembler is allowed to run in one part of the memory and place the assembled machine instructions and data as they assembled directly into their assigned memory locations
- When the assembly is done, the assembler caused to transfer to the starting instruction of the program.
- It is easy to implement as the assembler simply places the code into core the loader consists of one instruction that transfers to the statement of newly assembled program.

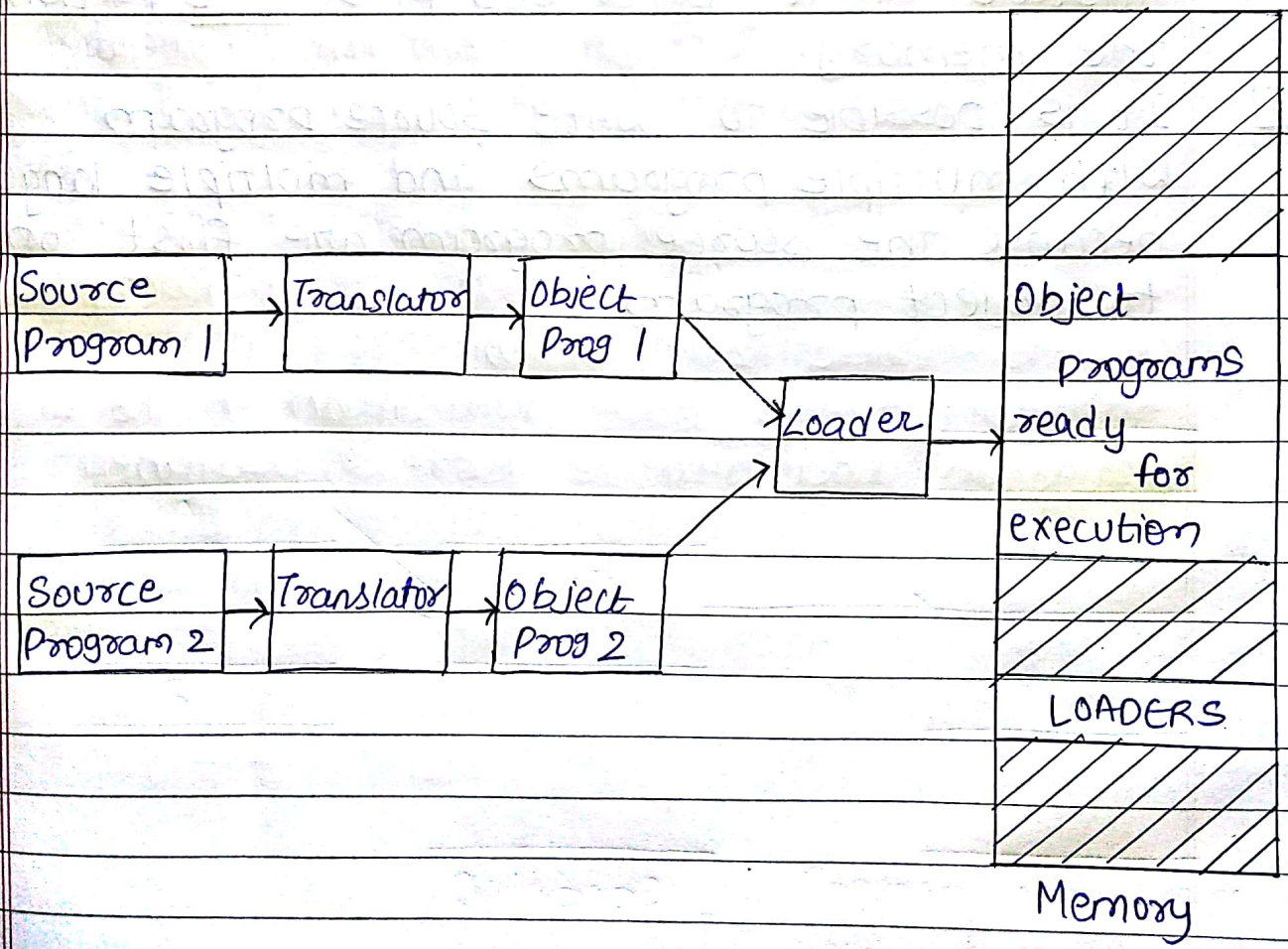
DISADVANTAGES

i) A portion of memory is always occupied by assembler. That portion is unavailable to any other object program

ii) User's program have to reassemble every time it runs.

iii) If source program contains subroutine written in different languages, then different assemblers have to be stored in memory (i.e wastage of memory).

2.] GENERAL LOADER SCHEMES



- The source program is converted to object program by some translator (assembler)
- The loader accepts the object modules and puts machine instruction and data in an executable form at their assigned memory
- The loader occupies some portion of main memory.

ADVANTAGES

1] The program need not be retranslated each time while running it.
2] There is no wastage of memory, because assembler is not placed in the memory, instead of it, loader occupies some portion of the memory.
3] It is possible to write source program with multiple programs and multiple languages, because the source program are first converted to object program.

SUBROUTINE LINKAGE

- The problem : A MAIN program A wished to transfer to sub-program B. However the assembler does not know this symbol and declare it is an undefined symbol unless a special mechanism has been provided.
- This mechanism has been implemented with a direct-linking or relocating loader.
- The assembler pseudo-op EXTRN followed by a list of symbol indicates that these symbols are defined in other programs.
- Corresponding if a symbol is defined in one program & referenced in others, we insert it into symbol list following the ENTRY
- Assembler will also inform the loader that these symbols may be referenced by other programs.

RELOCATING LOADERS

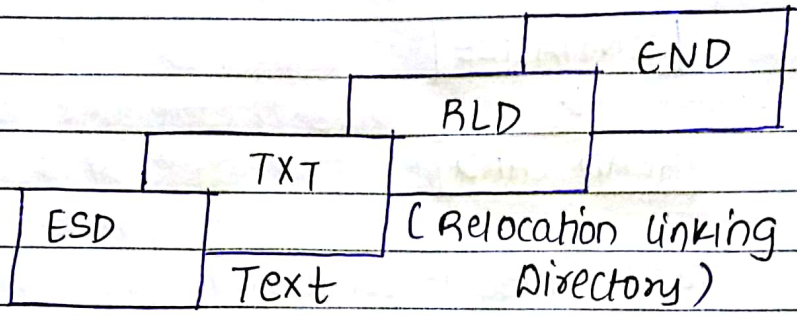
- To avoid possible re-assembling of all subroutines when a single subroutine is changed and to perform the task of allocation and linking for the programmer, this class of loader was introduced
- The assembler assembles each procedure segment independently and passes on to the loader the text information as to relocation and their segment references.

DIRECT LINKING LOADER

- It is a general relocatable loader it has has an advantage of allowing the programmer multiple procedure segments & multiple data segments and giving him complete freedom
- In referencing data or instructions content in other segments this provides flexible intersegment referencing and accessing ability, while at the same time allowing independent translation of programs
- The assembler must give the loader the following information with each procedure or data segment.
  i) The length of the segment
  ii) The list of all the symbols in the segment that may be referenced by other segments & their relative location within the segment.
  iii) A list of all symbol not defined in segment but referenced in the segment.
  iv) Information as to where address constants are located in the segment and a description of how to revise their values
  v) The machine code translation of the source program and relative address assigned.

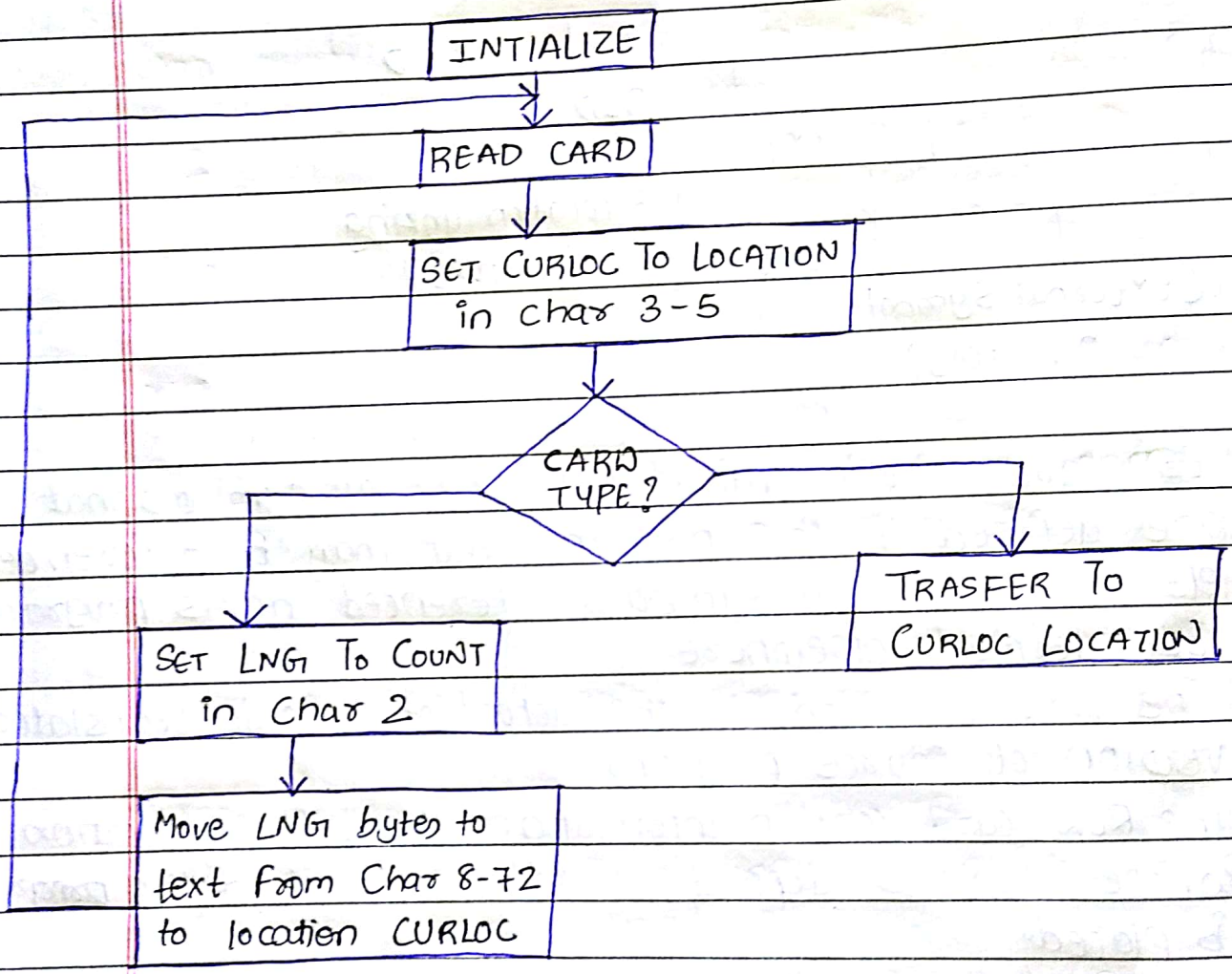
DESIGN OF DIRECT LINKING LOADER

Object deck for DLL



- ESD card contain information about all symbols that are defined in this program that may be referenced elsewhere, and all symbols referenced in this program but defined elsewhere.
- The text card contain the actual object code translated version of source program.
- The RLD card contain information about those location whose contents depend on address at which the program is placed.
- The END card indicates the end of object deck & specifies the starting address for execution if the assembled routine is the "Main Program"
- RLD card also contains location of each constant that needs to be changed due to relocation
- By what it has to be changed & operation to be performed.

ABSOLUTE LOADER



DESIGN OF ABSOLUTE LOADER

- With an absolute loading scheme the programmer & assembler perform the task of allocation, relocation & linking
- The loader has to read cards of the object deck & move the text on the cards into the absolute locations specified by the assembler
- There are 2 types of information that the object deck must communicate from the assembler to the loader
  - It must convey the machine instruction along with assigned core locations.
  - It must convey the entry point of the program which is where the loader is to transfer control when all instructions are loaded
- Assuming that this information is transmitted on cards
- Instruction are stored on cards as one core byte per column. 8-bit means 356 possible contents
- Information is stored in core as 80 contiguous bytes.

CARD FORMATS FOR AN ABSOLUTE LOADER

TEXT CARDS (for instruction and data)

CARD COLUMN	CONTENTS
1	card type=0 (for text identifier)
2	count of number of byted (1 byte per column)
3-5	address at which data on card is to be put
6-7	empty (could be used for validating checking)
8-72	instruction & data to be loaded
73-80	card sequence number

TRANSFER CARDS (to hold entry point to program)

CARD COLUMN	CONTENTS
1	card type=1 (transfer card identifier
2	count=0
3-5	address of entry point
6-72	empty
73-80	card sequence number.