Date / /

UNIT-3

Design of a direct linking loader →

The design of direct linking loader follow the steps of general design procedure scheme of any system software

(1) Specification of problem (2) Specify the data structure
(3) Specifications are (4) Define format of data structure
(5) Specification of Data structure (6) Specify algo
(7) formatic of databases (8) Look for modularity
(9) Repeat step 5.

(1) Specification of problem ⇒ The organization of the IBM 360 facilitates the task to be performed by its relocating loader. On the IBM 7094, a direct access machine, it was necessary to relocate the address portion on almost all instruct In the 360, instruction relocation is accomplished by the use of the base register, which is set by neither the assembler nor the loader. Therefore, the 360 relocating loaders can treat instructions exactly like nonrelocatable data (full word constants, characters etc.). However address constants must still be relocated.

The 360 direct-linking loaders processes programs generated by the assemblers FORTRAN compiler, or PL/I compiler. Recall that neither the original source program nor the assembler symbol table is available to the loader. Therefore the object deck must contain

Pg No.
Date / /

For such location the assembler must supply information enabling the loader to correct these contents.

The end card indicates the end of the object deck and specify the starting address for execution if the assembled routine is the main program.

8 Specification of Data Structure => (databases)

The next step in our design procedure is to identify the databases required by each pass of the loader.

Pass1 databases:

(i) Input object deck.

(2) A parameter, the initial Program Load Address (IPLA) supplied by the programmer or the operating system, that specific specifies the address to load the first segment.

(3) A program load address (PLA) counter, used to keep track of each segment's assigned location.

(4) A table, the Global External symbol table (GEST), that is used to store each external symbol and its corresponding assigned core address.

(5) A copy of the input to be used later by pass2. This may be store on an auxiliary storage device, such as magnetic tape, disk, or drum, or the original object decks may be reread by the loader a second time for pass2.

(6) A printed ~~list~~ listing, the load map, that specifies each external symbol and its assign value.

Pass2 databases:-

(1) Copy of object programs inputted to pass1.

(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.

(7) ~~Format of database~~ :-

3) Formate of databases:-

The third step in our design procedure is to specify the formate and content of each of the databases.

4) Global External Symbol table:-

The GEST is used to store the external symbol defined by means of a segment definition (SD) or local definition (LD) entry or an External Symbol Dictionary (ESD) card when these symbol 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 fig.

12 bytes / entry

External Symbol (8 bytes) (characters)	Assigned code address (4-bytes) (decimal)
"PG1bbbbb"	104
"PG1ENT1b"	124
"PG1ENT2b"	134
"PG2bbbbb"	168
"PG2ENT1b"	184

Notation: This simple GEST content is based upon the example

Fig: Global External Symbol table (GEST) Format

Q Formate of Cards =>

Simple program (Source deck)

Source card reference	Relative address			
1	0	PG1	START	
2			ENTRY	PG1ENT1, PG1ENT2
3			EXTRN	PG2ENT1, PG2
4	20	PG1ENT1	=	
5	30	PG1ENT2	=	
6	40		DC	A(PG1ENT1)
7	44		DC	A(PG1ENT2+16)
8	48		DC	A(PG1ENT2-PG1ENT1-8)
9	52		DC	A(PG2)
10	56		DC	A(PG2ENT1+PG2-PG1ENT1+4)
11			END	
12	0	PG2	START	
13			ENTRY	PG2ENT1
14			EXTRN	PG1ENT1, PG1ENT2
15	16	PG2ENT1	=	
16	24		DC	A(PG1ENT1)
17	28		DC	A(PG1ENT2+16)
18	32		DC	A(PG1ENT2-PG1ENT1-3)
19			END	

(a) procedure PG1

(b) procedure PG2

fig: Sample procedures PG1 and PG2

P.T.O.

Algorithm =>

Pass 1 - Allocate segments and define symbols =>

The purpose of the first pass is to assign allocation to each segment, and thus to define the value of all external symbols. Since we wish to minimise the amount of core storage required form the total available location after the preceeding segment. It is neccesary 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 system, the programmer may specify the IPLA, in either case we will assume that the IPLA is a parameter to the loader. Initially the PLA is set to the IPLA. An object card is then read and a copy written for use by pass 2. The card can be one of the five types; ESD, TXT RLD, ELD or EOF. If it is 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 based depending upon the type of external symbol. When the ELD or EOF card is encountered the program load address is increamented by the length of segment and pass 1 is completed respectively and control transfer 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 address constants. At the end of pass 2 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 is set to IPLA. The card are read one by one from the object deck file left by pass 1 then each of the five types of card is processed differently.

Pg. No.
Date / /

Format of card:-

ESD card

Source card reference	Name	Type	ID	Relative address	length
1	PGM1	SD	01	0	60
2	PGM1ENT1	LD	--	20	
2	PGM1ENT2	LD	--	30	
3	PGM2	ER	02	--	
3	PGM2ENT1	ER	03	--	

TXT cards

Only the interesting ones, i.e those involving address constants)

Source card reference	Relative address	Contents	comments
6	40-43	20	
7	44-47	45	= 30+15
8	48-51	7	= 30-20-3
9	52-55	0	Unknown to PGM1
10	56-59	-16	= -20+4

RLD cards

Source card reference	ESDID	Length (bytes)	Flag +or-	Relative address
6	01	4	+	40
7	01	4	+	44
9	02	4	+	52
10	03	4	+	56
10	02	4	+	56
10	01	4	-	56

Figure: Object deck program PGM1

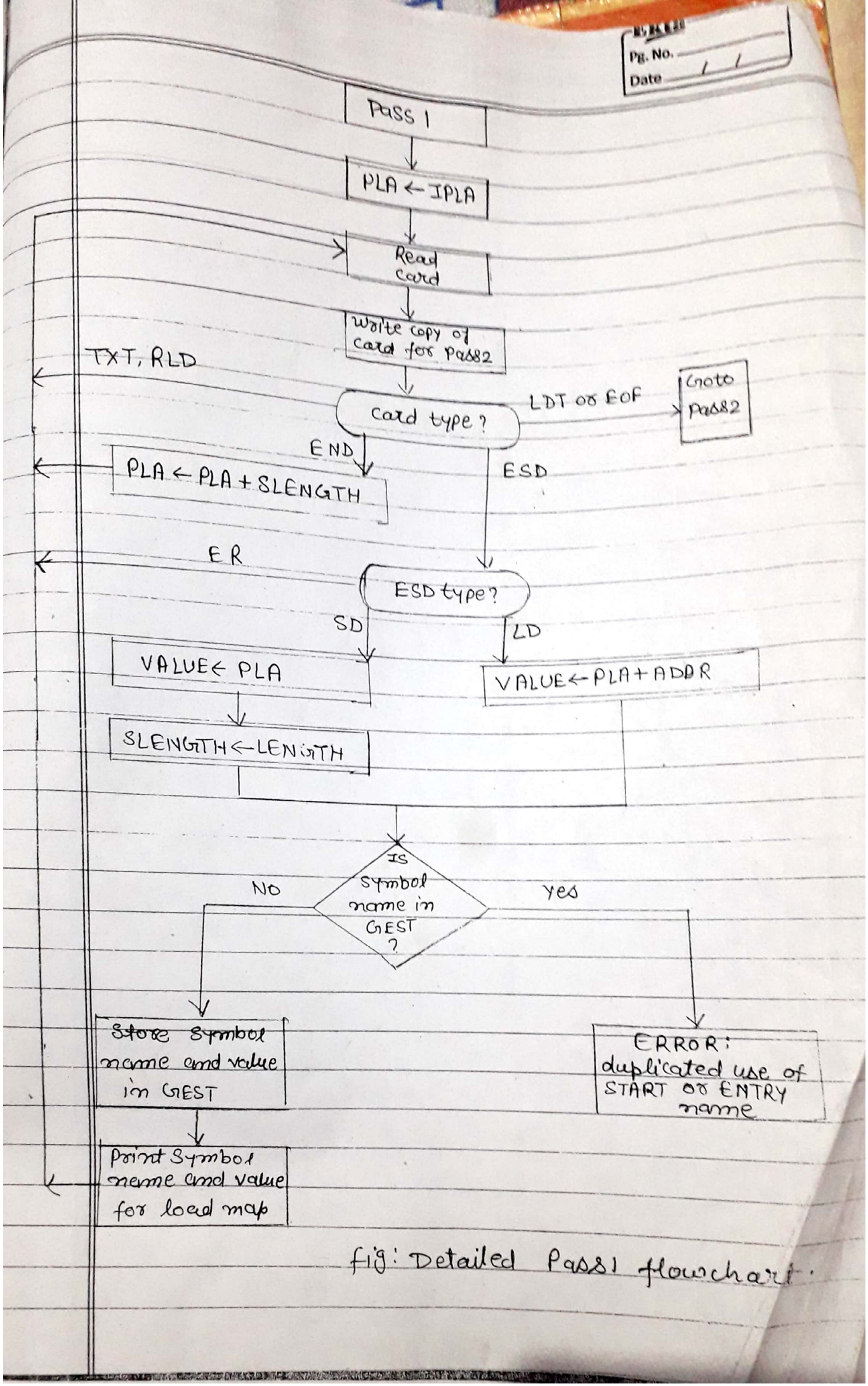

fig: Detailed Pass1 flowchart.

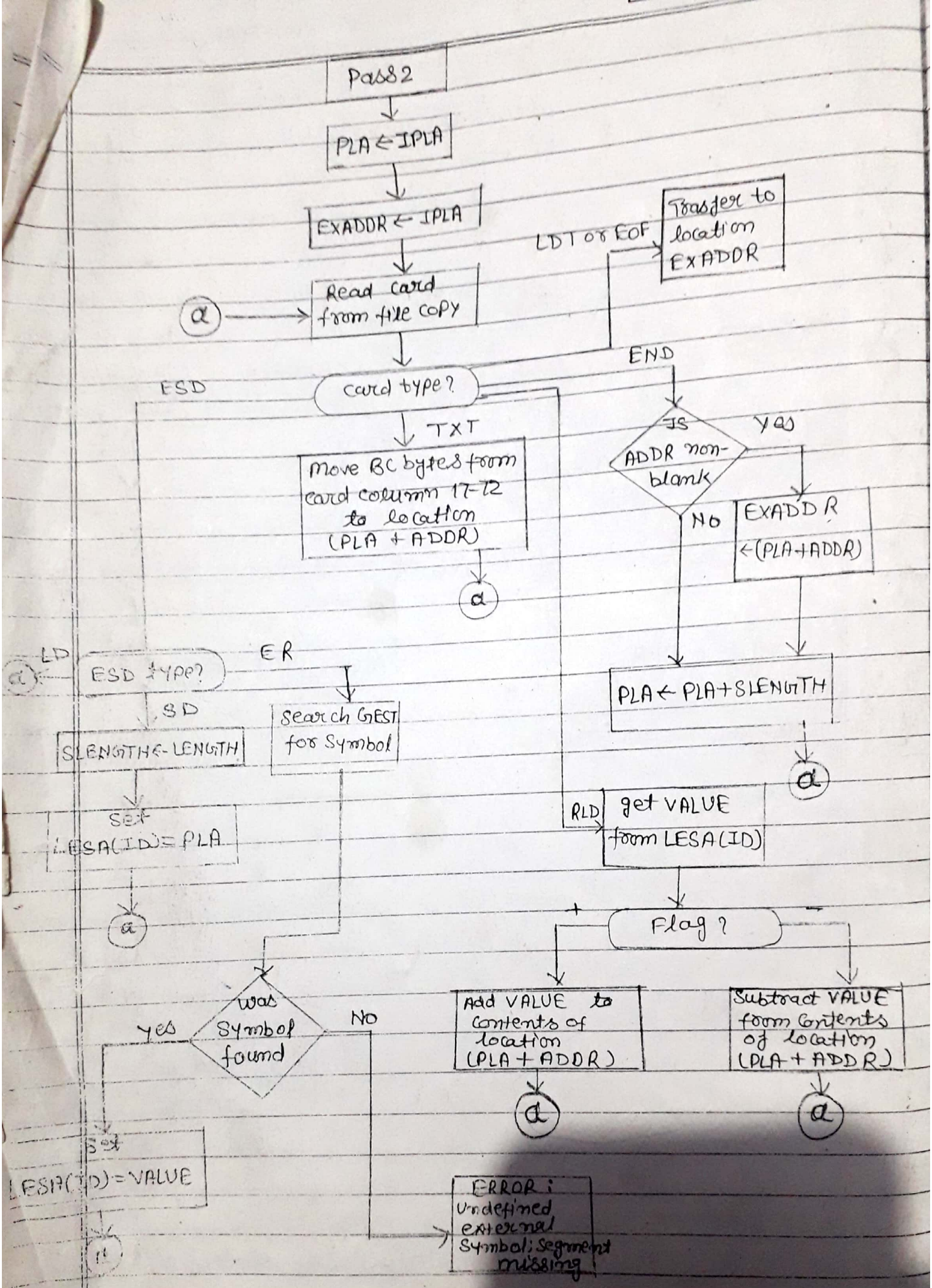

fig: Detailed Pass2 flowchart