List the databases of 2 pass assembler

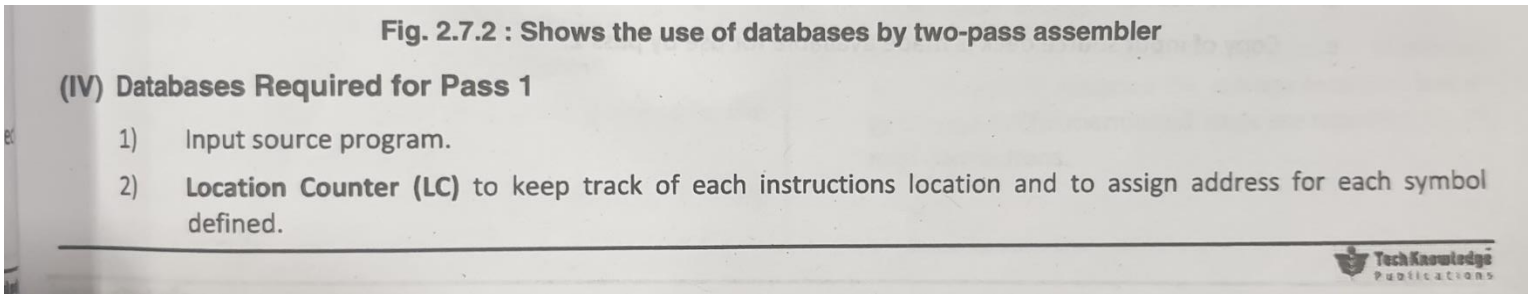


Fig. 2.7.2 : Shows the use of databases by two-pass assembler

(IV) Databases Required for Pass 1

1) Input source program.
2) **Location Counter (LC)** to keep track of each instructions location and to assign address for each symbol defined.

3) **Machine Operation Table (MOT)** that indicates the symbolic mnemonic for each instruction and its lengt
4) **Pseudo Operation Table (POT)** that indicates the pseudo-opcode and the action to be taken in pass 1.
5) **Symbol Table (ST)** that is used to store each label and its corresponding value.
6) **Literal Table (LT)** that is used to store each literal encountered and its corresponding assigned location.
7) A **copy** of the input to be used later by pass 2.

(V) Databases Required for Pass 2

1) **Copy** of source program input to pass 1.
2) Location Counter (LC)
3) **MOT** that indicates for each instruction
   (a) Symbolic mnemonic
   (b) Length
   (c) Binary machine op-code
   (d) Format (RR, RX, ...)
4) **POT** that indicates for each pseudo-opcode the symbolic mnemonic and the action to be taken in pass 2.
5) **Symbol Table (ST)** prepared by pass 1, containing each label and its corresponding value.
6) **Literal Table (LT)** prepared by pass 1, containing each literal encountered and its assigned location.
7) **Base Table (BT)** that indicated which registers are specified as base registers by USING pseudo-opcodes a what the specified contents of these registers are.
8) **INSTRUCTION workspace** which is used to hold each instruction as its various parts is being assemb together.
9) **PRINT LINE workspace** used to produce printed listing.
10) **PUNCH CARD workspace** used to punch the assembled instructions in the format needed by the loader.
11) An **output deck** of assembled instructions in the format needed by the loader.

What is forward reference

When a symbol is referred before it is defined, it is called a ***forward reference***.

The function of the assembler is to replace each symbol by its machine address and if we refer to that symbol before it is defined its address is not known by the assembler such a problem is called **forward reference problem**. Forward reference problem is solved by making different passes (i.e. One pass, Two pass, Multi pass) over the assembly code. E.g.:

```
class C {
public :
void muta for (int x) {myValue = x;}
int accessor () {return myValue;}
private :
int myValue;
};
```

In this there are two reference to myValue before it is declared.

List the databases of single pass assembler

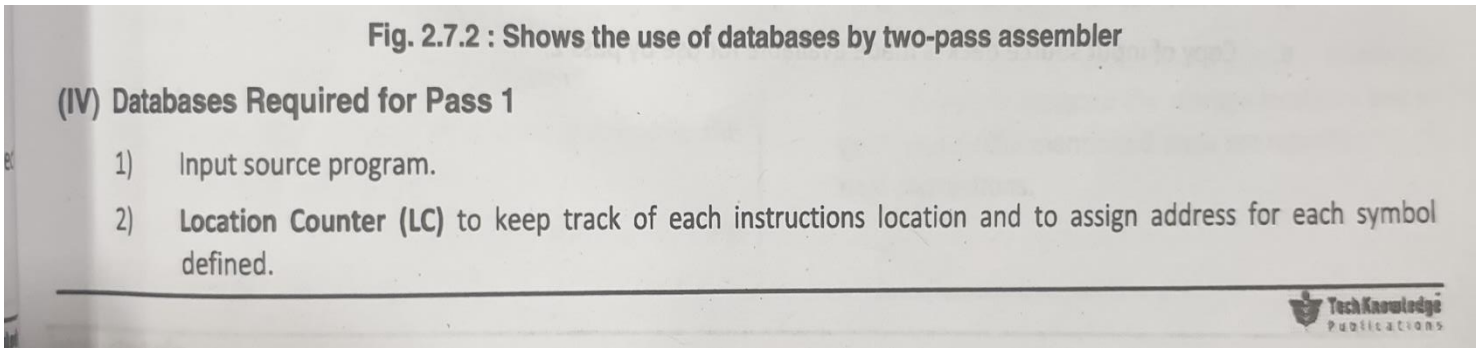


Fig. 2.7.2 : Shows the use of databases by two-pass assembler

(IV) Databases Required for Pass 1

1) Input source program.
2) **Location Counter (LC)** to keep track of each instructions location and to assign address for each symbol defined.

3) **Machine Operation Table (MOT)** that indicates the symbolic mnemonic for each instruction and its length.
4) **Pseudo Operation Table (POT)** that indicates the pseudo-opcode and the action to be taken in pass 1.
5) **Symbol Table (ST)** that is used to store each label and its corresponding value.
6) **Literal Table (LT)** that is used to store each literal encountered and its corresponding assigned location.
7) A **copy** of the input to be used later by pass 2.

What is general format of Macros

3.2.2 Example of Macro Instruction

Q. Explain macro calls within macro giving example.
Q. Explain the concept of macro definition within a macro.

A **macro** instruction is an abbreviation for a sequence of operations. The macro processor replaces each m instruction with a group of source language statements. This is called **expansion of macros.**

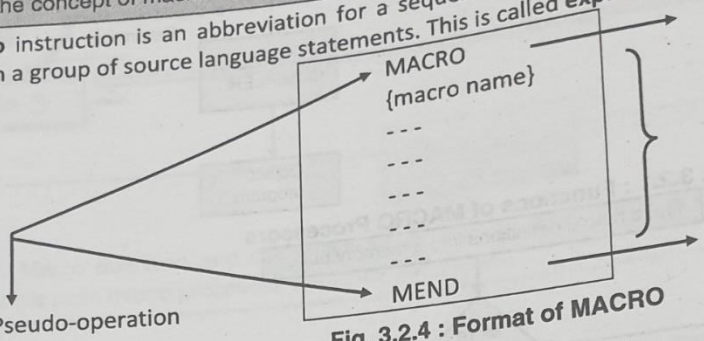


Fig. 3.2.4 : Format of MACRO

1. Copy Code : Example

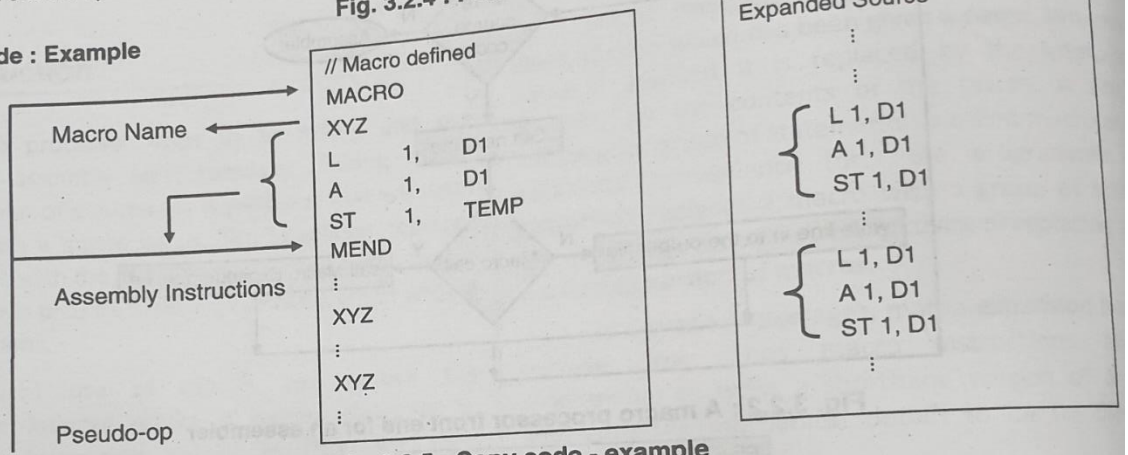


Fig. 3.2.5 : Copy code - example

The statements of expansion are generated each time the macro is invoked.

2. Example for Parameter Substitution

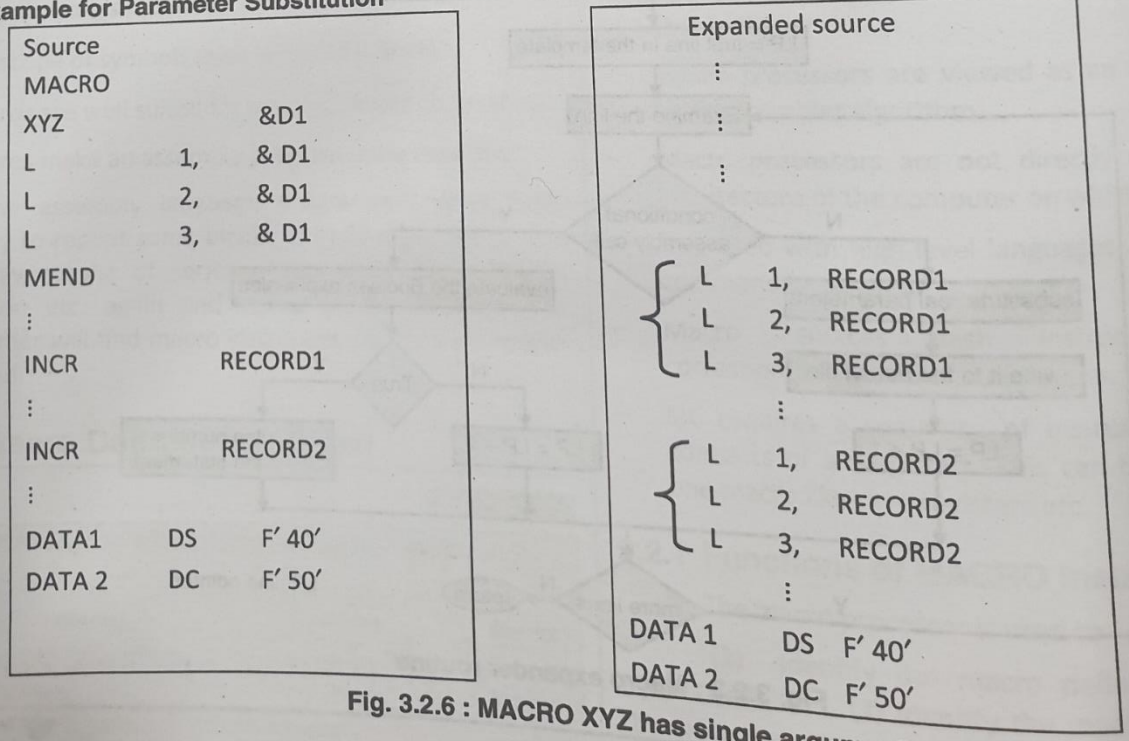


Fig. 3.2.6 : MACRO XYZ has single argument

List the difference between MACRO and Function

Fig. 3.5.2 : Flow chart for single pass macro processor

3.6 Difference between MACROS and FUNCTIONS

1.	A macro just replaces the code assigned to it for e.g. if a macro 'P1' is defined with a value 3.14. Function does not replace code but it performs what it is supposed to do.
2.	Macro are faster because it avoids function call overhead.
3.	No type restriction is placed in arguments so that one macro may serve for several data types.
4.	Macros check the number of arguments; they do not check argument types. Functions checks both number of arguments and arguments type.
5.	The macro replaces the shorthand text, with the longer definition. If a macro is used twenty time in code in all instances, the longer definition will be substituted in and thus code grow. A function definition occurs only once.
6.	Macros do not have return type but a function has a return type.
7.	Debugging in macro is difficult as compared to function as size of the code grows.

Tech Knowledge Publications

8.	Macro should be used for small tasks and functions for the complex tasks.
9.	Macro does not alters the flow of execution but the function alters the flow of execution.
10.	Macro calls a process at the translation time function calls a process at the execution time.

What are parameterized macros

2. Parameterised Macro / Positional Macro (Macro definition with one, two or many arguments)

MU - Dec. 14

Q. Explain positional parameter in macro. **(Dec. 14, 5 Marks)**

Q. Explain the different ways of parameter passing in macros. **(5 Marks)**

A parameterized macro is a type of macro that is able to insert given objects into its expansion.

Parameterized macro is not able to modify the instruction that replaces the call. To solve this problem we use macro calls within arguments, parameters, corresponding to these dummy arguments will appear in macro definition. Parameterized macros are useful mechanism for performing in line expansion.

In mentioned example we have specified four arguments including **STRG** argument. In parameterised macros parameters and arguments are associated with each other according to their positions in the macro prototype and macro invocation statements.

A certain macro instruction GENER has 10 possible parameters.

E.g. GENER MACRO &1, &2, & type ------- & 10

GENER , , DIRECT, 3

There are generally three ways of specifying arguments to a macro call.

List the databases of two pass microprocessor

In pass2 we perform recognize macro call and perform macro expansion

1.COPY FILE

It is a file it contains the out put given from PASS1

2.MNT

It is used for recognizing macro name

3.MDT

It is used to perform macro EXPANSION

4.MDTP

It is used to point to the index of MDT .

The starting index is given by MNT

5.ALA

It is used to replace the index notation by it actual value

6.ESC

What are conditional macros

4. Conditional MACRO Expansion

MU - Dec. 15

Q. Explain different pseudo-ops used for conditional macro expansion along with an example. **(Dec. 15, 10 Marks)**

In our previous example of macro instruction, each invocation of a particular macro was expanded into the same sequence of statements. These statements could be varied by the substitution of parameters, but the form of the statements, and the order in which they appeared was unchanged.

Most macro processors can modify the sequence of statements generated from a macro expansion, depending on the arguments supplied in the macro-expansion.

AIF and AGO are two conditional macro expansion pseudo opcodes which permit conditional selection of the sequence of the machine instruction that appear in expansion of macro call. In conditional macro expansion only part of the macro is copied out into the code. Which part is copied out will be under the control of the parameters in the macro call.

Example

```
:
MACRO
START 1   D  1,    VALUE 1
          D  2,    VALUE 2
          :
START 2   D  1,    VALUE 1
             2,    VALUE 2
          :
VALUE 1   DC       F' 0'
VALUE 2   DC       F' 1'
          :
MEND
```

In this example the labels, the operands and the number of instruction generated change in each sequence. This example can be written as :

Give example of nested macro

```
Example :
MACRO
MAC_X     &par1, &par2
MOVER     REG1,  &par1
  – MACRO
  – MAC_Y   &par2, REG=REG3
  – ADD     REG, &par2
  – MOVEM   REG, &par2
  – MEND
PRINT     &par1
MEND
```

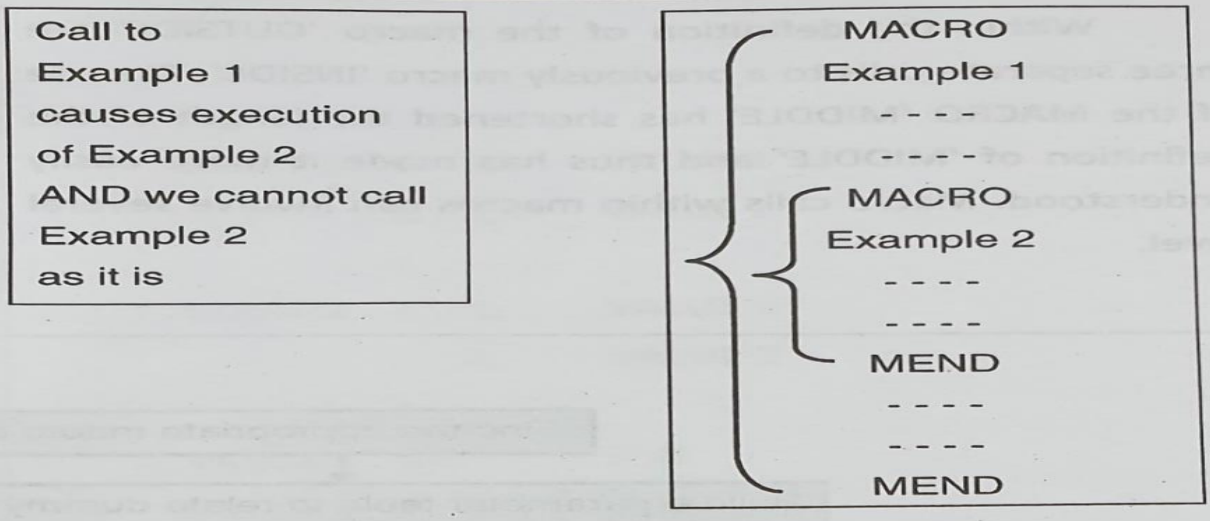


Fig. 3.3.4 : Example of NESTED MACROS

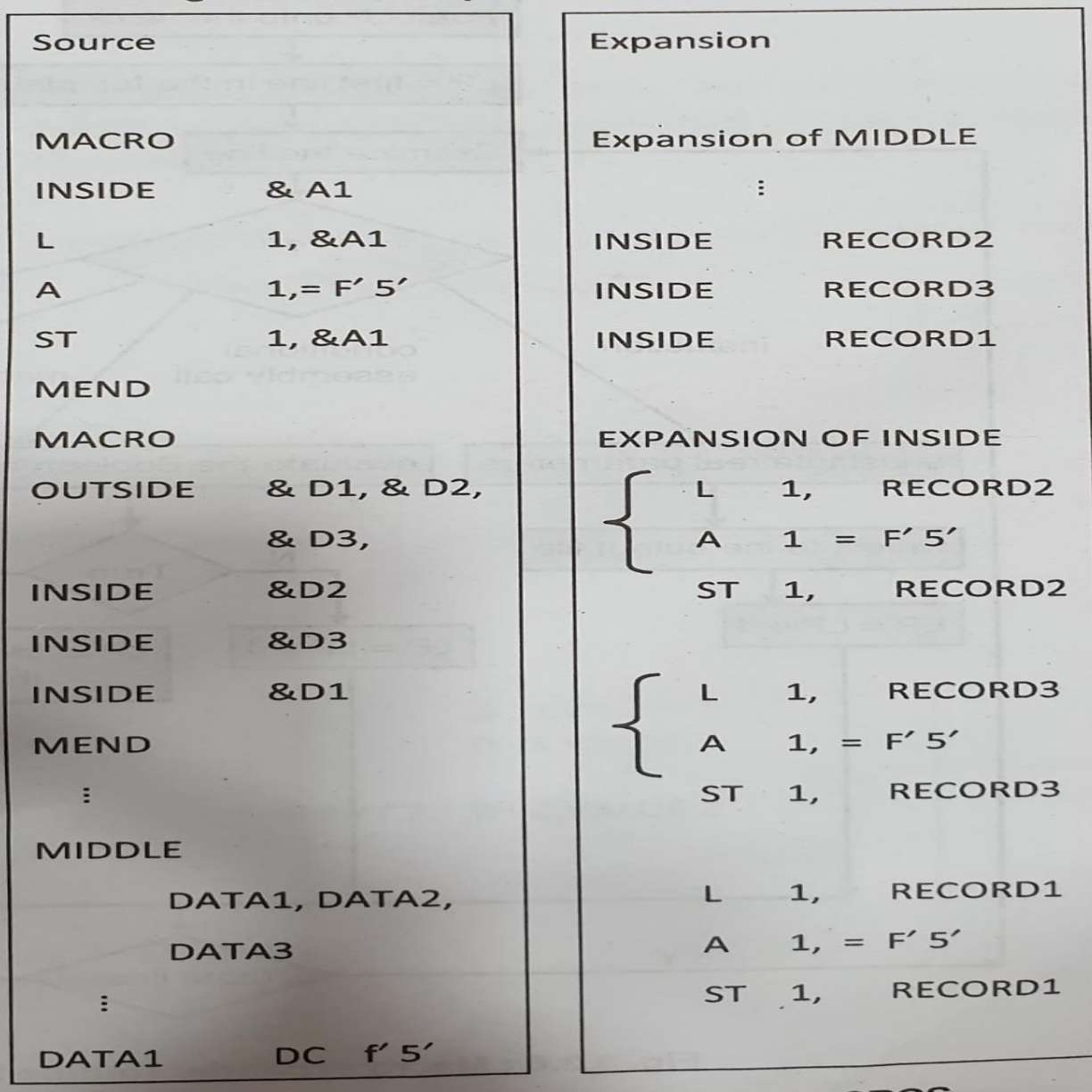


Fig. 3.3.5 : Example of NESTED MACROS

What are the functions of Loader

4.3 Functions of Loader

MU - Dec. 14, May 15

Q. What are the different functions of loader explain in brief ? **(Dec. 14, 5 Marks)**

Q. Explain Functions of loader. **(May 15, 5 Marks)**

Q. Explain relocation. **(2 Marks)**

- The fundamental task of a loader is to
  - bringing an object program into memory and
  - starting its execution
- An object program contains translated instructions and data from the source program. It also specifies addresses in memory where these items are to be loaded.

To execute a program a loader performs the following four functions

1. **Allocation :** It is used to allocate space in memory for the object programs. Translators cannot allocate space since overlap may occur or large wastage of memory takes place.
2. **Linking :** It combines two or more separate object programs and resolve symbolic references between object decks. It also supplies the information needed to allow to reference between them.
3. **Relocation :** It modifies the object program so that it can be loaded at an address different from the location originally specified and adjusts all address dependent location.
4. **Loading :** Physically it places the machine instructions and data into the memory for the execution.

What is Compile and Go loader

A compile and go loader is one in which the assembler itself does the processes of compiling then place the assembled instruction in the designated memory locations. The assembly process is first executed and then the assembler causes a transfer to the first instruction of the program.

Advantages

- Simple to implement.
- No extra procedures are involved.

Disadvantages

- Wastage of memory.
- It retranslates the user's program code every time it is executed.
- When the source code is in different program is in different languages it is hard to handle multiple segments.

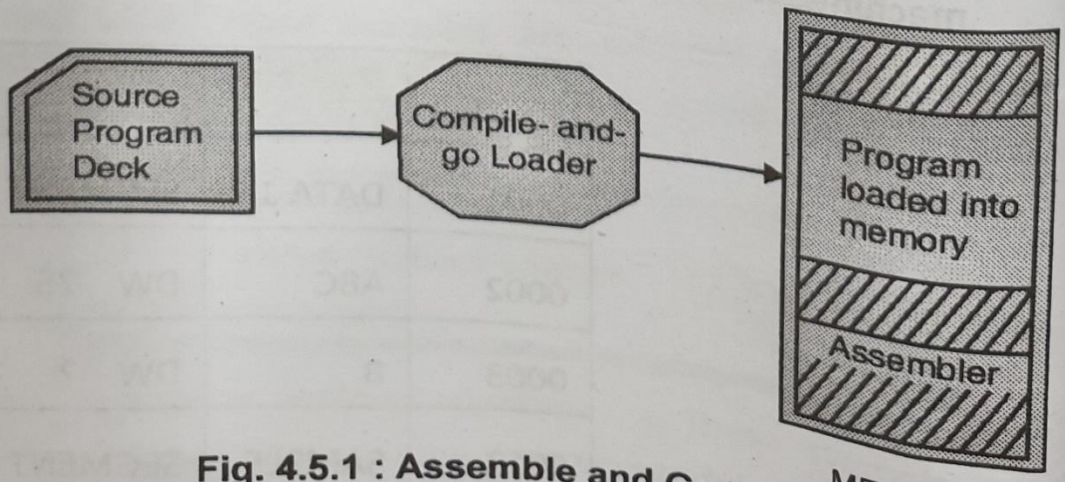


Fig. 4.5.1 : Assemble and Go Loader

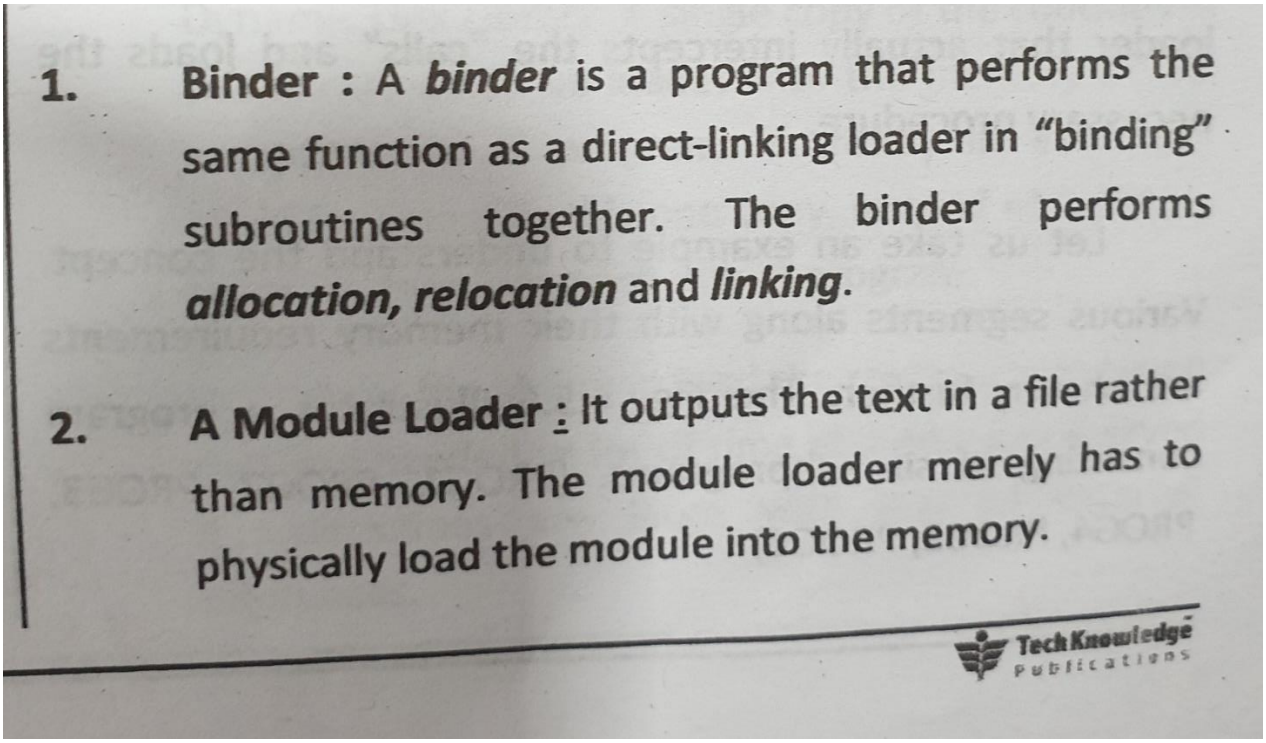
4.5.1(A) General Loading Sch...

Differentiate between absolute loader and direct linking loader

BASIS FOR COMPARISON	LINKER	LOADER
Basic	It generates the executable module of a source program.	It loads the executable module to the main memory.
Input	It takes as input, the object code generated by an assembler.	It takes executable module generated by a linker.
Function	It combines all the object modules of a source code to generate an executable module.	It allocates the addresses to an executable module in main memory for execution.
Type/Approach	Linkage Editor, Dynamic linker.	Absolute loading, Relocatable loading and Dynamic Run-time loading

What is binder

What module loader



1. **Binder :** A ***binder*** is a program that performs the same function as a direct-linking loader in "binding" subroutines together. The binder performs ***allocation, relocation*** and ***linking***.

2. **A Module Loader :** It outputs the text in a file rather than memory. The module loader merely has to physically load the module into the memory.

List the databases of DLL

Direct Linking Loader 2 Pass Loader
- do address correction.
- RLO : meaning or working
* To content of 40 add absolute ~~add~~ ~~of ESD Absolute~~ address of PG1 ie 1000
* Assembler given RLD table to OLL.
- GEST - Global external symbol Table
- IPLA - Initial Program load address (which add prog start)

Explain design of Pass1 of two pass assembler

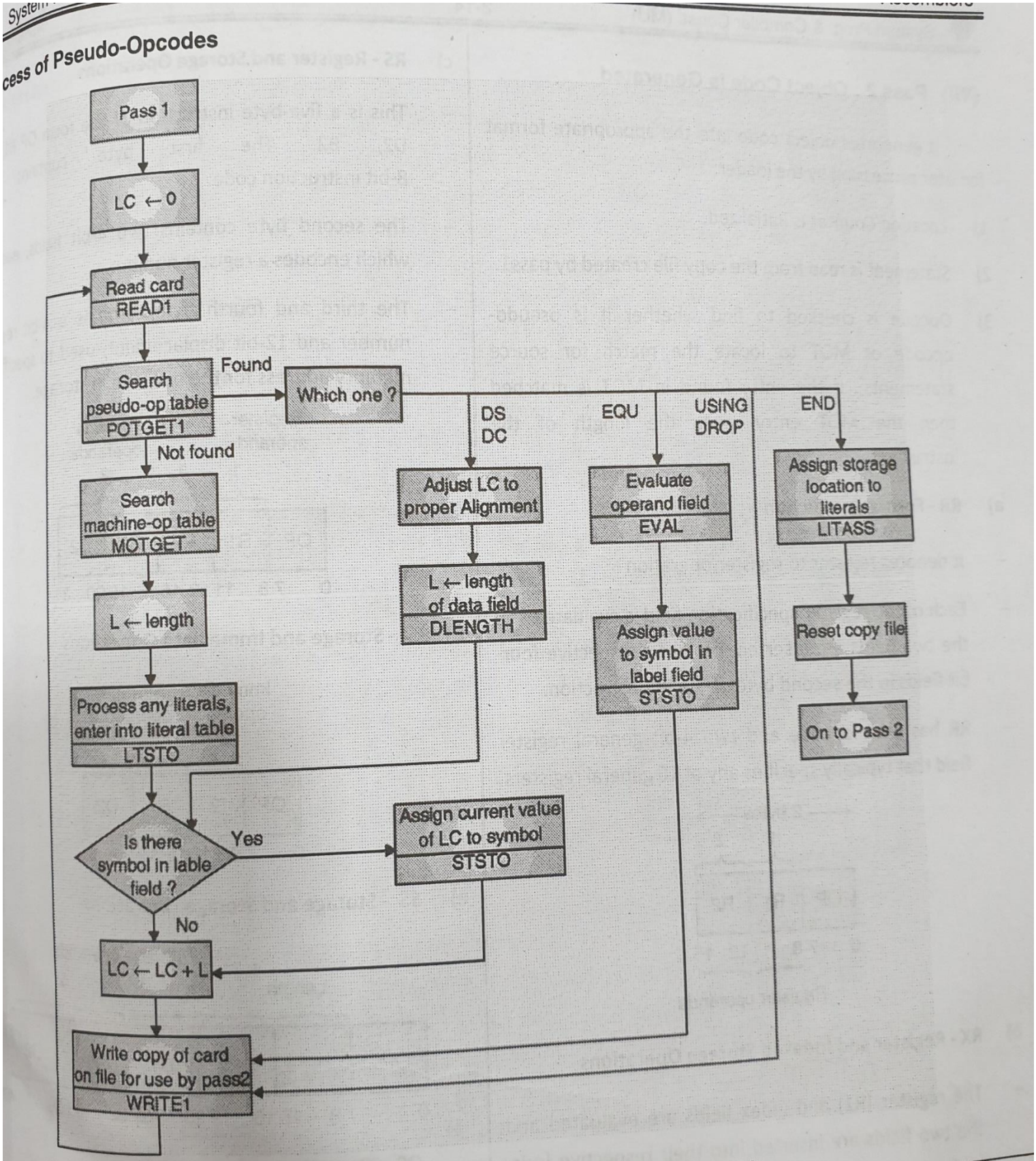
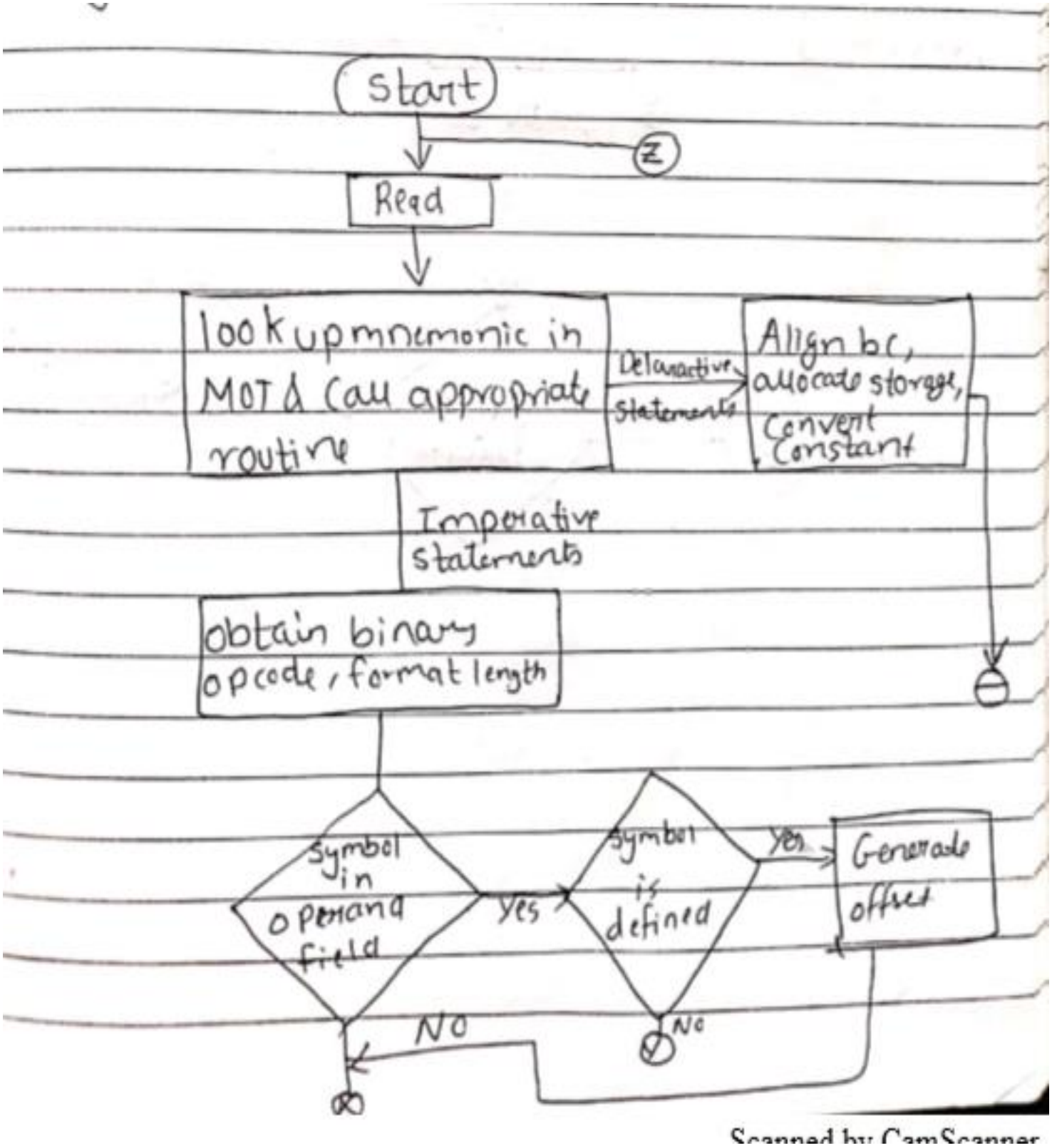
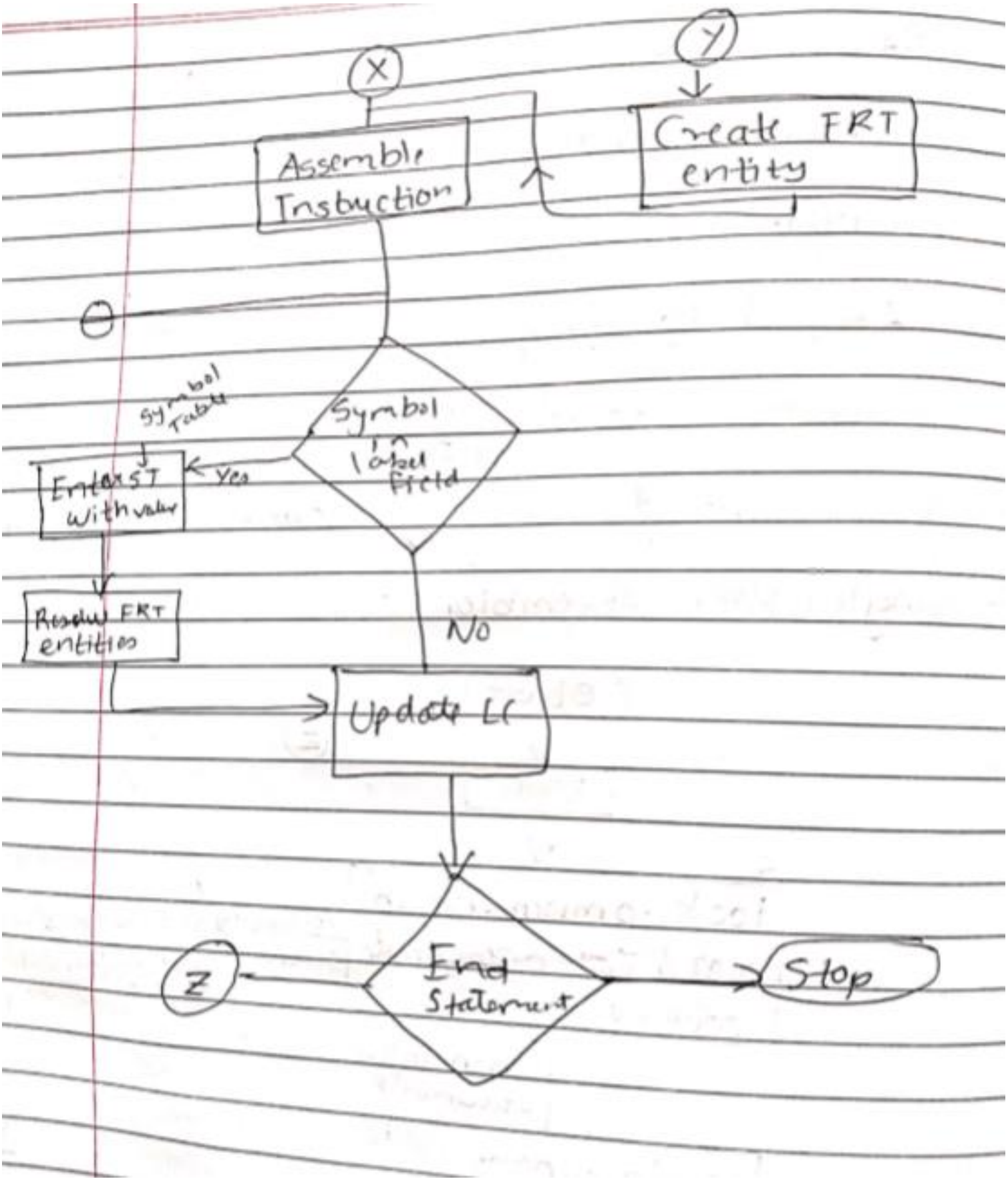


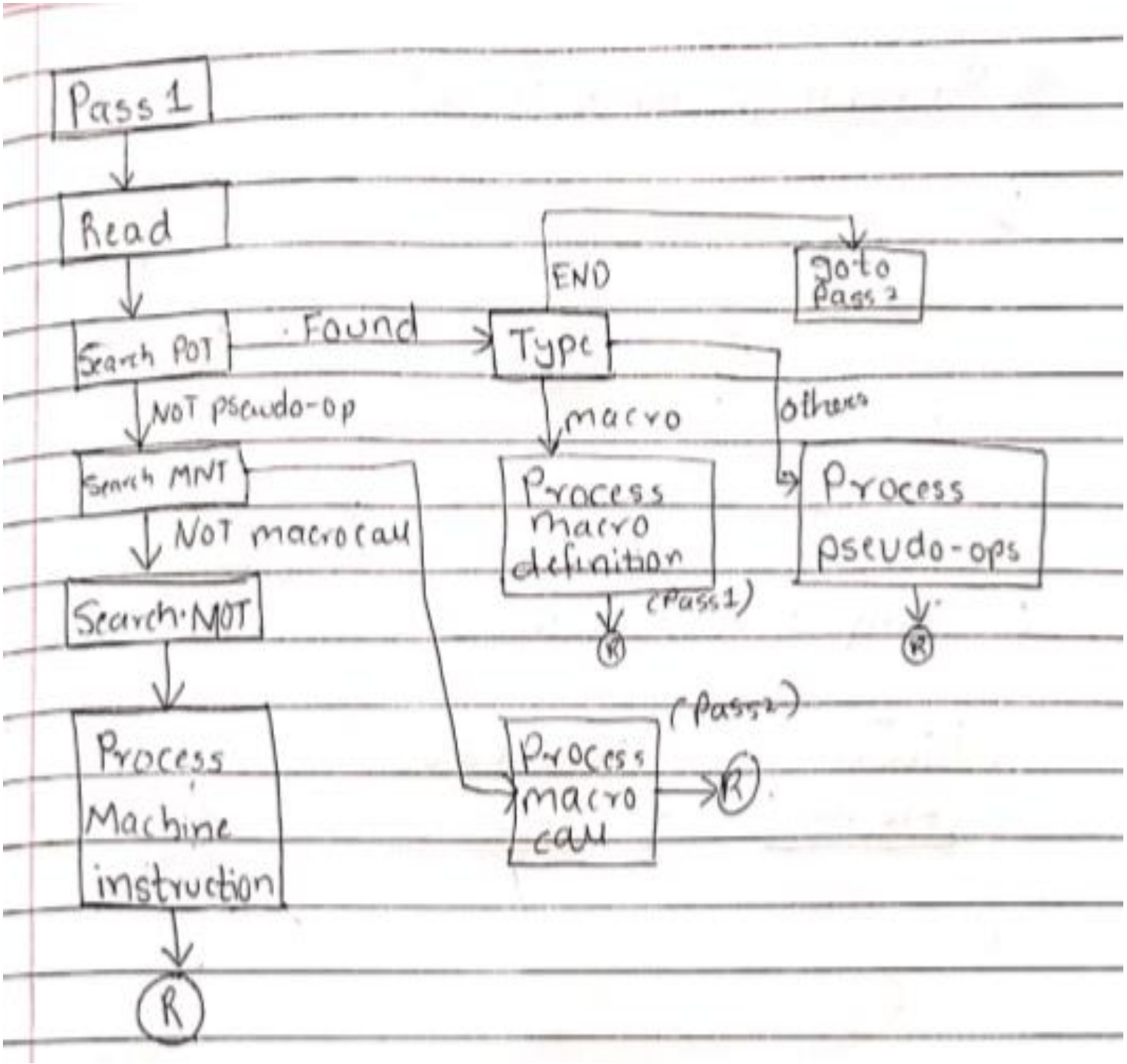
Fig. 2.7.3 : A flowchart for pass 1 of Two Pass Assembler

Explain design of single pass assemble

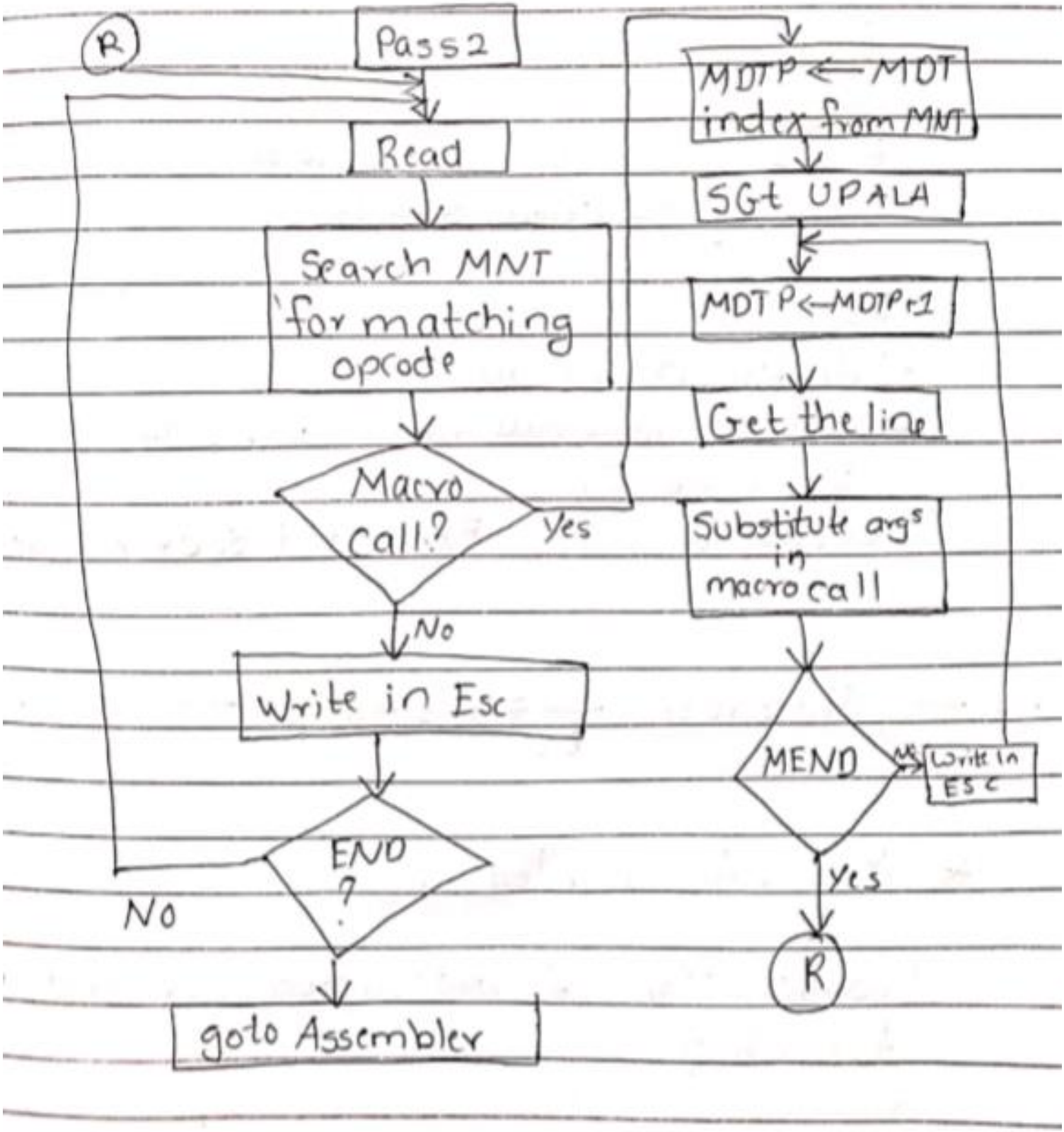



X
Y
Assemble Instruction
Create FRT entity
Symbol Table
Symbol in Label Field
Yes
Enter ST with value
Resolve FRT entities
No
Update LC
Z
End Statement
Stop

Explain design of pass1 of Macro processor



Explain design of pass2 of Macro processor

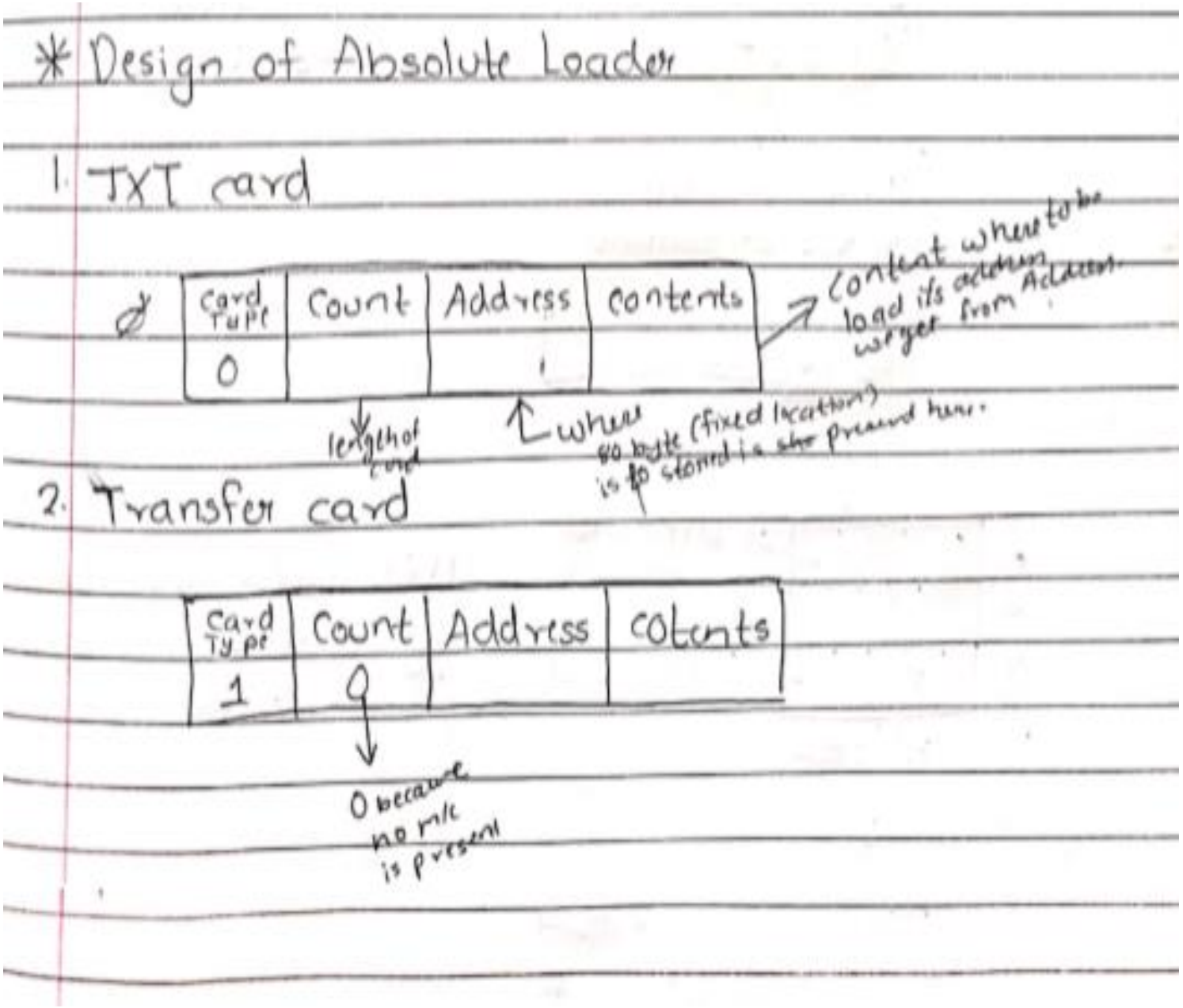


What are the features provided by Macro? Explain with Example.

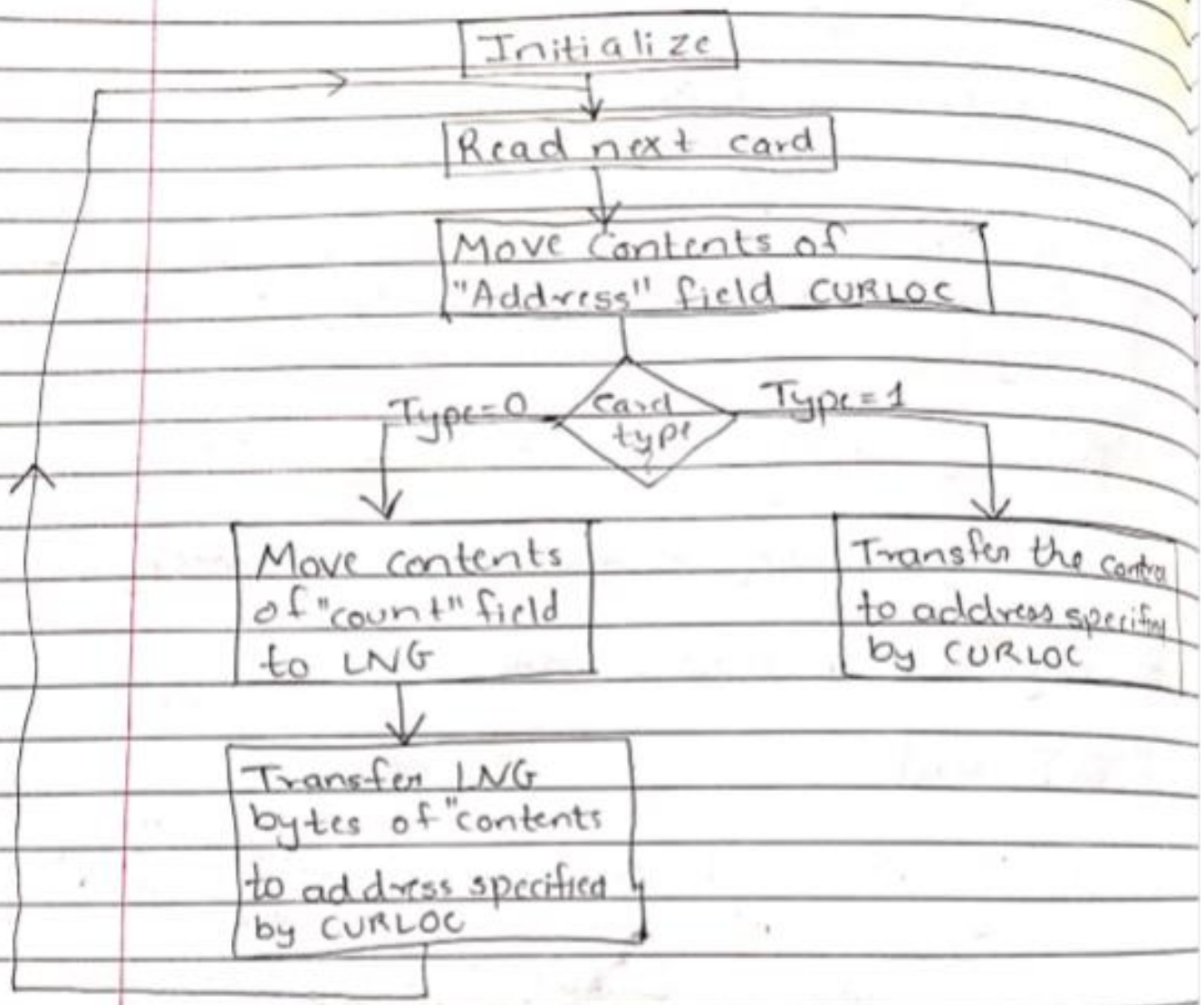
- **Macro** represents a group of commonly used statements in the source programming language.
- Macro Processor replace each macro instruction with the corresponding group of source language statements. This is known as expansion of macros.
- Using Macro instructions programmer can leave the mechanical details to be handled by the macro processor.
- Macro Processor designs are not directly related to the computer architecture on which it runs.
- Macro Processor involves definition, invocation and expansion.

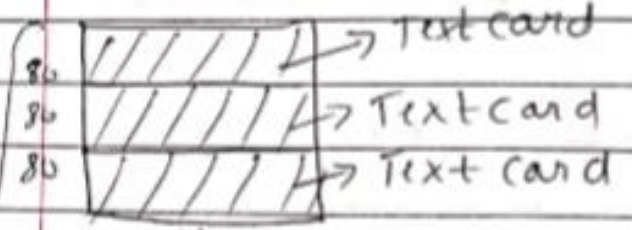
.

Explain design of absolute Loader



* Flowchart of Absolute Loader

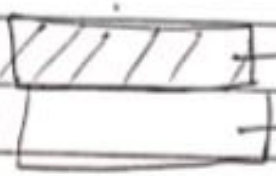




this is o/p generated by Assembler as i/p for loader (~~Abs~~ Absolute loader)

Absolute code

500 bytes



Transfer card

(First point of execution) information is present execution start from.

Given an assembly language code, write the output of pass1 and pass2 of assembler

Given an assembly language code, write the output of pass1 and pass2 of microprocessor

LC	Source program
0	PG1 Start 0
0	using *,15
0	L1, five
4	A 1, four
8	ST 1, temp
12	SR 2,2
14	AR 1,2
16	five DC F'5'
20	four DC F'4'
24	temp DS TF
28	END

o/p of Pass1

Symbol	LC value	length	Relocation
PG1	0	1	R
five	16	4	R
four	20	4	R
temp	24	4	R

Page No.:
Date: 7/2/20

Pass 2 o/p

LC	MIC
0	~~M~~ —
0	—
0	2A 1, 16
4	3A 1, 20
8	4A 1, 24
12	5A 2, 2
14	6A 1, 2
16	5
20	4
24	—
28	—

registers

space is reserved

Base Table

15	00