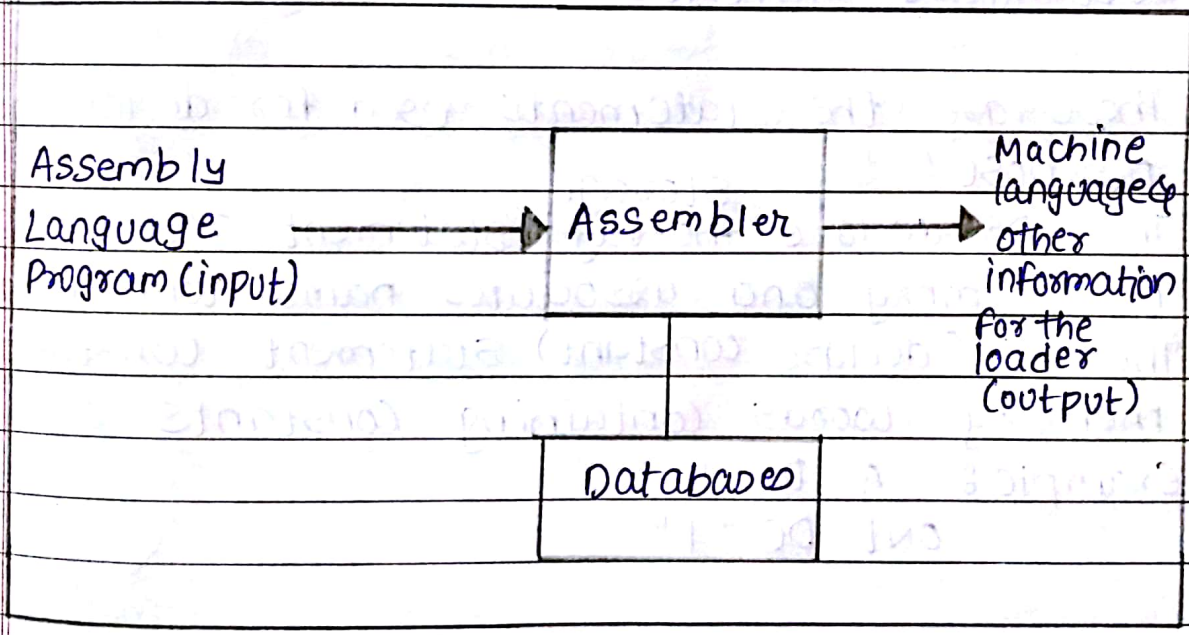
CHAPTER 2 : ASSEMBLERS

ASSEMBLER :

DEFINTION

An assembler is a program that accepts input as assembly language program and produces its machine language equivalent along with information for the loader.

DIAGRAM REPRESENTATION



Features of Assembly Language Programming [ALP]

I Language specification
II Statement in ALP
III Structure of ALP

* TYPES OF ASSEMBLY LANGUAGE STATEMENTS

1.] IMPERATIVE STATEMENTS

- An imperative statement indicates an actions to be performed during the execution of the assembled program. Each imperative statement typically translates into one machine instruction
- In simple terms, these are the statements understood by the machine & executed by the machine.

Example: MOV A,B

2.] DECLARATIVE STATEMENTS

- These are the statements used for declaration purpose
- The DS (declare ~~shortage~~ storage) statement reserves areas for memory and associates names with them
- The DC (declare constant) statement constants memory words containing constants.

Example: A DS 1
ONE DC "1"

3.] ASSEMBLER DIRECTIVES

- These are the statements used to instruct the assembler to perform certain actions regarding how assembly of input program is to be performed.

Example: START < constant >
END < operand spec. >

* GENERAL DESIGN PROCEDURE OF ASSEMBLER

- Specify the problem
- Specify data structures
- Define format of data structures
- Specify Algorithm
- Look for modularity [capability of one program to be subdivided into independent program units]
- Repeat 1 through 5 on modules

* SYNTHESIS PHASE

The synthesis phase is concerned with the construction of target language statements which have the same meaning as a source statement. Synthesis phase used the table to obtain the machine address with a name associated and machine opcode corresponding to the memonic

It performs two main activities

- memory Allocation
- Code Generation

* Types of Assembler

- Single Pass Assembler:

A single pass assembler scans the program only once and created the equivalent binary program. The assembler substitute all of the symbolic instruction with machine code in one pass.

2.] Multi Pass Assembler :

Does the work in two pass - Resolves the forward references

- First pass - Scans the code
  Validates the tokens
  Creates a symbol table
- Second pass - Solves forward references
  Converts the code to the machine code

Statement of Problem

- The assembler must do the following
  1.] Generate Instructions
    - Evaluate the mnemonic in the operation field
    - Evaluate sub fields
  2.] Process Pseudo ops

ASSEMBLER PASS I

Step 1: Specify the problem

Pass 1: Define symbols & literals

1) Determine length of machine instruction [MOT]
2) Keep track of Location Counter [LC]
3) Remember values of Symbols [ST]
4) Process some pseudo ops [EQU, DS etc] [POT]
5) Remember literals [LT]

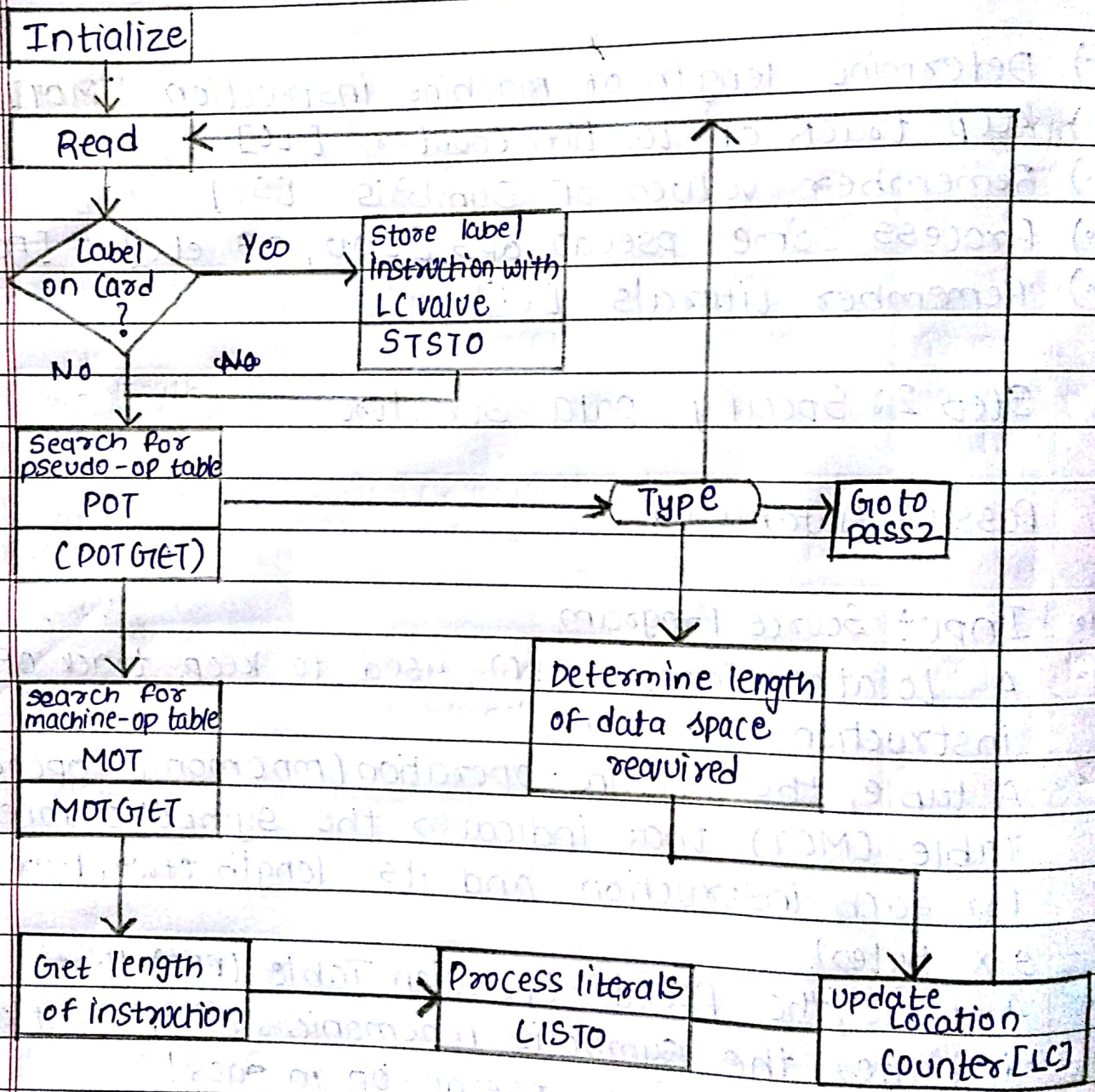
Step 2: Specify Data Structure

Pass 1: Databases

1. Input Source Program
2. A Location Counter (LC), used to keep track of each instruction's location
3. A table, the Machine-operation (mnemonic opcode) Table (MOT) that indicates the symbolic mnemonic, for each instruction and its length (two, four or six bytes)
4. A table, the Pseudo-Operation Table (POT) that indicates the symbolic mnemonic and the action to be taken for each pseudo-op in pass 1
5. A table, Symbol Table (ST) that is used to store each label and its corresponding value
6. A table, the literal table (LT) that is used to store each literal encountered and its corresponding assignment locations

OVERVIEW OF ASSEMBLER PASS I

FLOW CHART

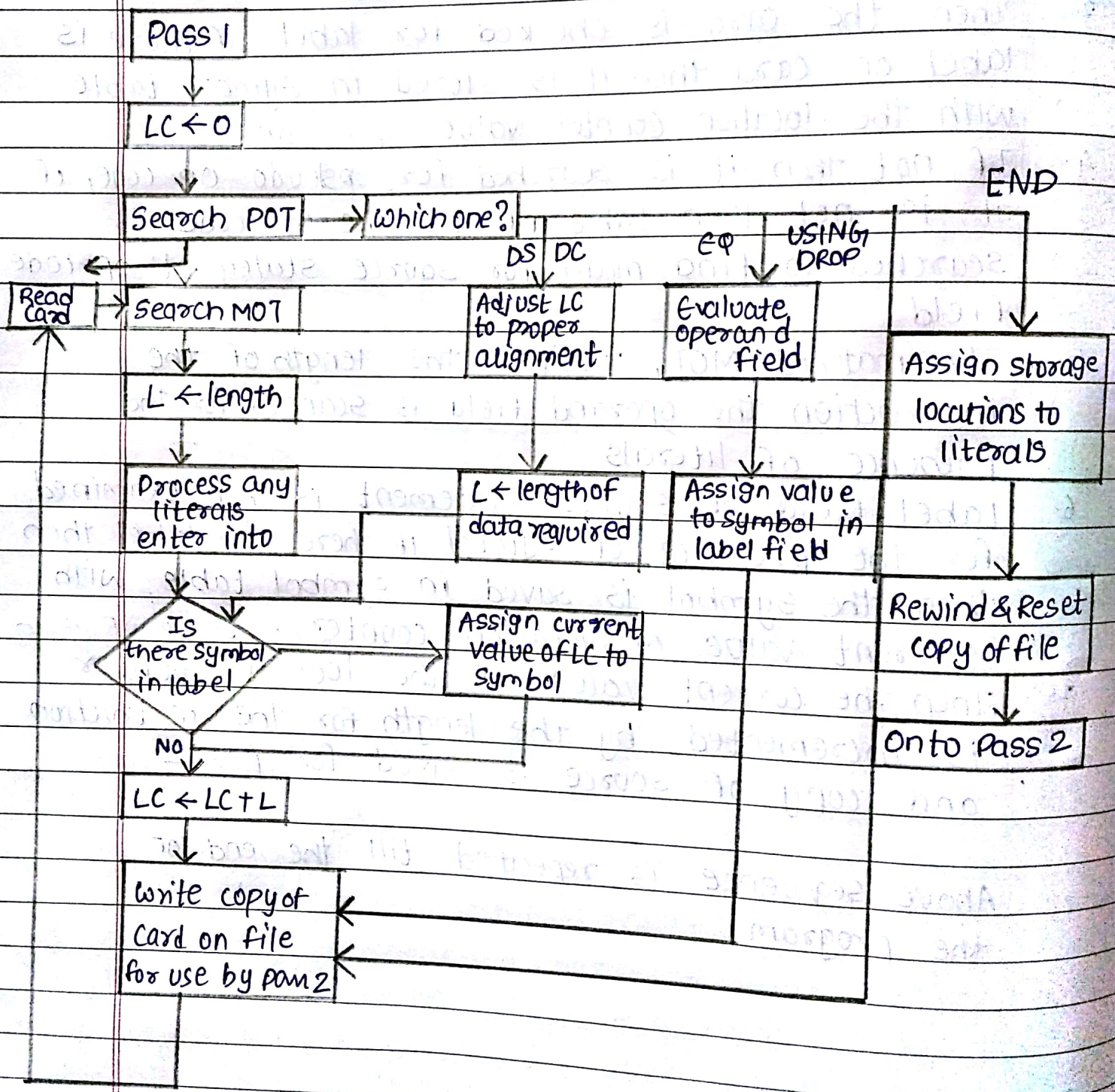


PROCESS

1. The Location Counter is set to zero first location where relative address is zero
2. Then the source statement is read
3. Then the card is checked for label, if there is label on card then it is stored in symbol table with the location counter value
4. If not then it is searched for pseudo-op code, if it is not then table of machine-op code is searched to find match for source statement op-code field
5. The matched MOT, specifies the length of the instruction the operand field is scanned for the presence of literals
6. Label field of source statement is then examined for the presence of symbol if there is label then then the symbol is saved in symbol table with current value of location counter
7. Then the current value of the location counter is incremented by the length for the instruction and copy of source is saved for pass 2

Above sequence is repeated till the end of the program.

DETAILED EXPLANATION OF ASSEMBLER PASS I



PROCESS

1. The location counter is set to first location where relative address is zero
2. Then the source statement is read
3. Then it is searched for pseudo-op table if it is found then it is checked for it's type
4. If the POT is DS/DC then LC is adjusted to proper alignment or if it is EQ then using DROP evaluate operand and assign value to symbol in label field
5. If there is no pseudo-op then it is searched for machine-op code is searched to find match for source statement op-code field and it's length is determined
6. The matched MOT is then scanned for the presence of literals and if label is found then the symbol is saved in the symbol table with current value of location counter

And the above seqvence is repeated till the end of program.