System Programming

Introduction

- What is System?

  System is the collection of various components

  Ex:- College is a system

- What is Programming?

Art of designing and implementing the programs.

What is Software ?

- Software is collection of many programs
- Two types of software
  - System software

    These programs assist general use application programs

  Ex:- Operation System , Assembler etc.
  - Application software

    These are the software developed for the specific goal

- System Program:-

  "These are programs which are required for the effective execution of general user programs on computer system."

- System Programming:-

  " It is an art of designing and implementing system programs."

Components of System Programming

- Interpreter
- Assembler
- Compiler
- Macros and Microprocessors
- Formal systems
- Debugger
- Linkers
- Operating system

Need Of System Software

The basic need of system software is to achieve the following goals :-

- To achieve efficient performance of the system
- To make effective execution of general user program
- To make effective utilization of human resources
- To make available new, better facilities

Operating System

- It is the collection of system programs which acts as an interface between user and the computer and computer hardware.
- The purpose of an operating system is to provide an environment in which A user can execute programs in a convenient manner

Functions of Operating System

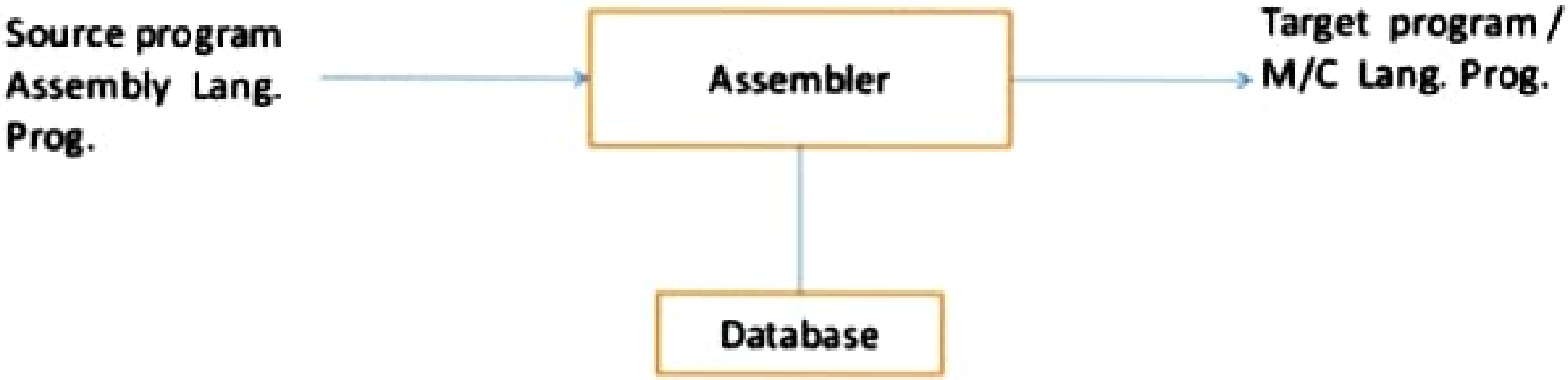
- File handling and management.
- Storage management (Memory management).
- Device scheduling and management.
- CPU scheduling.
- Information management.
- Process control (management).
- Error handling
- Protecting itself from user & protecting user from other users.

Translators

- These are the system programs that converts the source program into computer understandable fashion
- Types of translators
  - Single Pass translator
  - Multi Pass translator

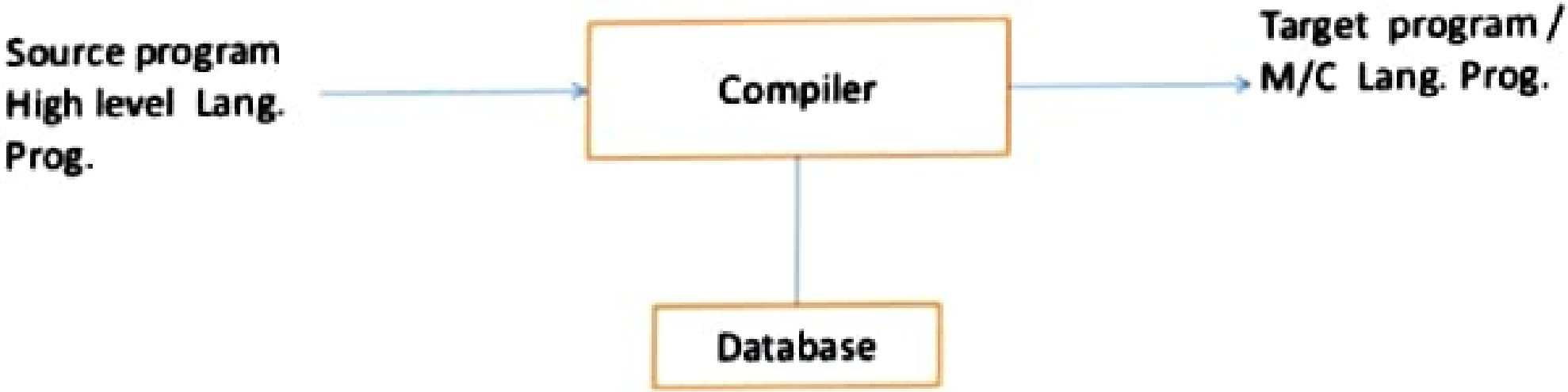
Translators

- Assembler:-
- These are the system programs which will automatically translate the assembly language program in to the machine language program



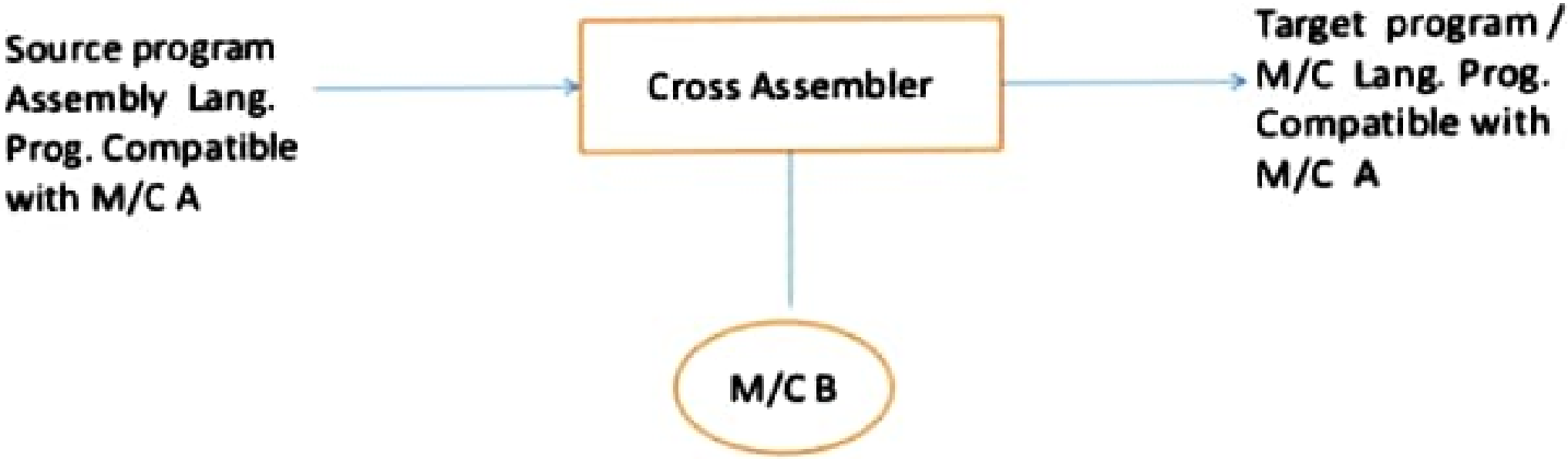
Translators

- Complier:-
- These are the system programs which will automatically translate the High level language program in to the machine language program



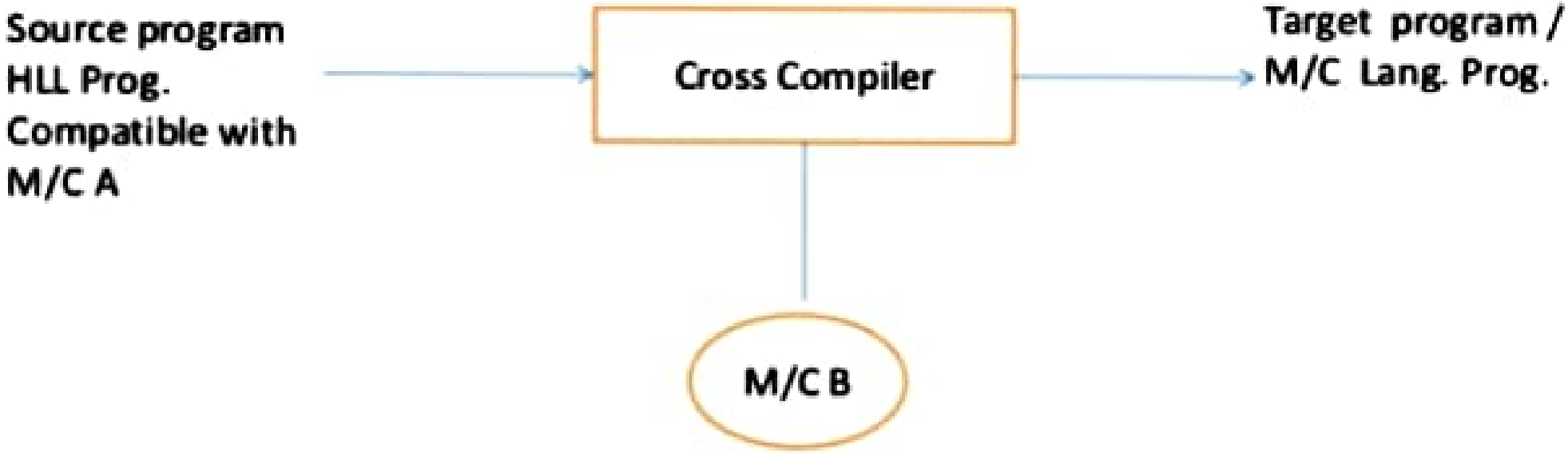
Translators

- Cross Assembler:-
- These are the system programs which will automatically translate the Assembly Language program compatible with M/C A, in to the machine language program compatible with M/C A, but the underlying M/C is M/C B



Translators

- Cross Compiler:-
- These are the system programs which will automatically translate the HLL program compatible with M/C A, in to the machine language program compatible with M/C A , but the underlying M/C is M/C B



Translators

- Interpreter
  - It is the language translator which execute source program line by line with out translating them into machine language.

  Types of Interpreter

  - Pure Interpreter
    - In this case no preprocessing is required on source program before an interpretation starts.
    - Some preprocessing is required on source program before an interpretation starts.

Loader

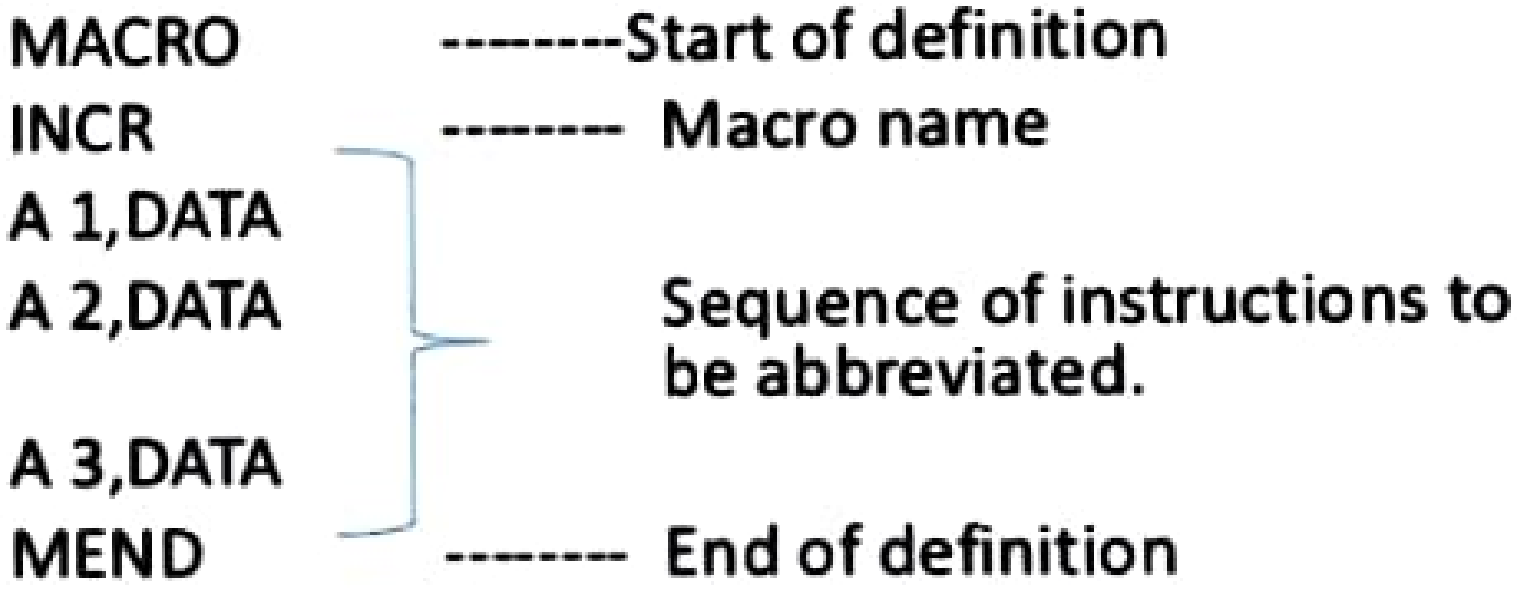
- A Loader is system program that place the object program into main memory and prepares it for execution.
- Basic functions of loader
  - Allocation
  - Linking
  - Relocation
  - Loading

Types of Loader

- Compile-and-go Loader
- Relocating Loader
- Direct Linking Loader
- Absolute Loader
- General Loader
- Dynamic Loader

Macro & Macro processor

- Macro
  - Macro is a single line abbreviation for a group of instruction.



Linking and Linker

- Linking
  - The Process of merging many object modules to form a single object program is called as linking.
  - Linker
    - The Linker is the software program which binds many object modules to make a single object program.

Formal System

- A formal system is an un interpreted calculus. It consists of
  - Alphabets
  - A set of words called Axioms.
  - Finite set of relations called rules of inference or production rules.
  - Ex Boolean algebra.

Types of Assembly Language statements

- Imperative statements
  - An imperative statement in assembly language indicates the action to be performed during execution of assembly statement

Ex:- A 1,FOUR

- Declarative Statement:-
  - These statements declares the storage area or declares the constant in program.

  - EX A DS 1
    ONE DC "1"

- Assembler Directives
  - These are the statements used to indicate certain thing regarding how assembly of input program is to be performed.

  - Ex START 100
    USING *, 15

Types of Assembler

- Single pass Assembler

- Multi pass Assembler

Problem of Forward Reference

- When the variables are used before their definition at that time problem of forward reference accurse.

Problem of Forward Reference

```
JOHN   START    0
       USING *, 15
        L  1, FIVE
        A  1,FOUR
        ST  1, TEMP
FOUR   DC    F'4'
FIVE   DC    F'5'
TEMP   DS    1F
       END
```

General Design Procedure of Two Pass Assembler

1. Specify the problem
2. Specify data structures
3. Define format of data structures
4. Specify algorithm
5. Look for modularity [capability of one program to be subdivided into independent programming units.]
6. Repeat 1 through 5 on modules.

SOURCE PROGRAM	FIRST PASS	SECOND PASS
RAM START 0		
USING *, 15		
L 1, FIVE	0 L 1,_(0,15)	0 L 1, 16(0,15)
A 1,FOUR	4 A 1,_(0,15)	4 A 1, 12(0,15)
ST 1, TEMP	8 ST 1,_(0,15)	8 ST 1, 20(0,15)
FOUR DC F'4'	12 4	12 4
FIVE DC F'5'	16 5	16 5
TEMP DS 1F	20 --	20 --
END		

Specify the problem

Pass1: Define symbols & literals.

1) Determine length of m/c instruction [MOTGET1]
2) Keep track of Location Counter [LC]
3) Remember values of symbols [STSTO]
4) Process some pseudo ops[EQU,DS etc] [POTGET1]
5) Remember Literals [LITSTO]

Pass2: Generate object program

1) Look up value of symbols [STGET]

2) Generate instruction [MOTGET2]

3) Generate data (for DS, DC & literals)

4) Process pseudo ops[POTGET2]

Step 2. Data structure:-

Pass1: Databases

- Input source program
- "LC" location counter used to keep track of each instructions addr.
- M/c operation table (MOT) [Symbolic mnemonic & length]
- Pseudo operation table [POT], [Symbolic mnemonic & action]
- Symbol Table (ST) to store each lable & it's value.
- Literal Table (LT), to store each literal (variable) & it's location.
- Copy of input to used later by PASS-2.

Step 2. Data structure:-

- **Pass2: Databases**
- Copy of source program input to Pass1.
- Location Counter (LC)
- MOT [Mnemonic, length, binary m/c op code, etc.]
- POT [Mnemonic & action to be taken in Pass2
- ST [prepared by Pass1, label & value]
- Base Table [or register table] indicates which registers are currently specified using 'USING' pseudo op & what are contents.
- Literal table prepared by Pass1. [Lit name & value].

Format of Data Structures

- Machine Operation Table
  - The op-code is the key and it's value is the binary op code equivalent, which is used for use in generating machine code.
  - The instruction length is stored for updating the location counter.
  - Instruction format is use in forming the m/c language equivalent

6 bytes entry

Mnemonic Op-code (4 byte) character	Binary Op-code (1 byte) Hex	Instruction length (2 bits) (binary)	Instruction format (3 bits) (binary)	Not used in this design (3 bits)
"Abbb"	5A	10	001	-
"LOAD"	4A	10	001	-
"MOVb"	5E	01	000	-
"MVIb"	1E	11	100	-

b→ represents blank

Codes used-

Instruction length

01 = 1 half words

10 = 2 half words

11 = 3 half words

Instruction

000 - RR

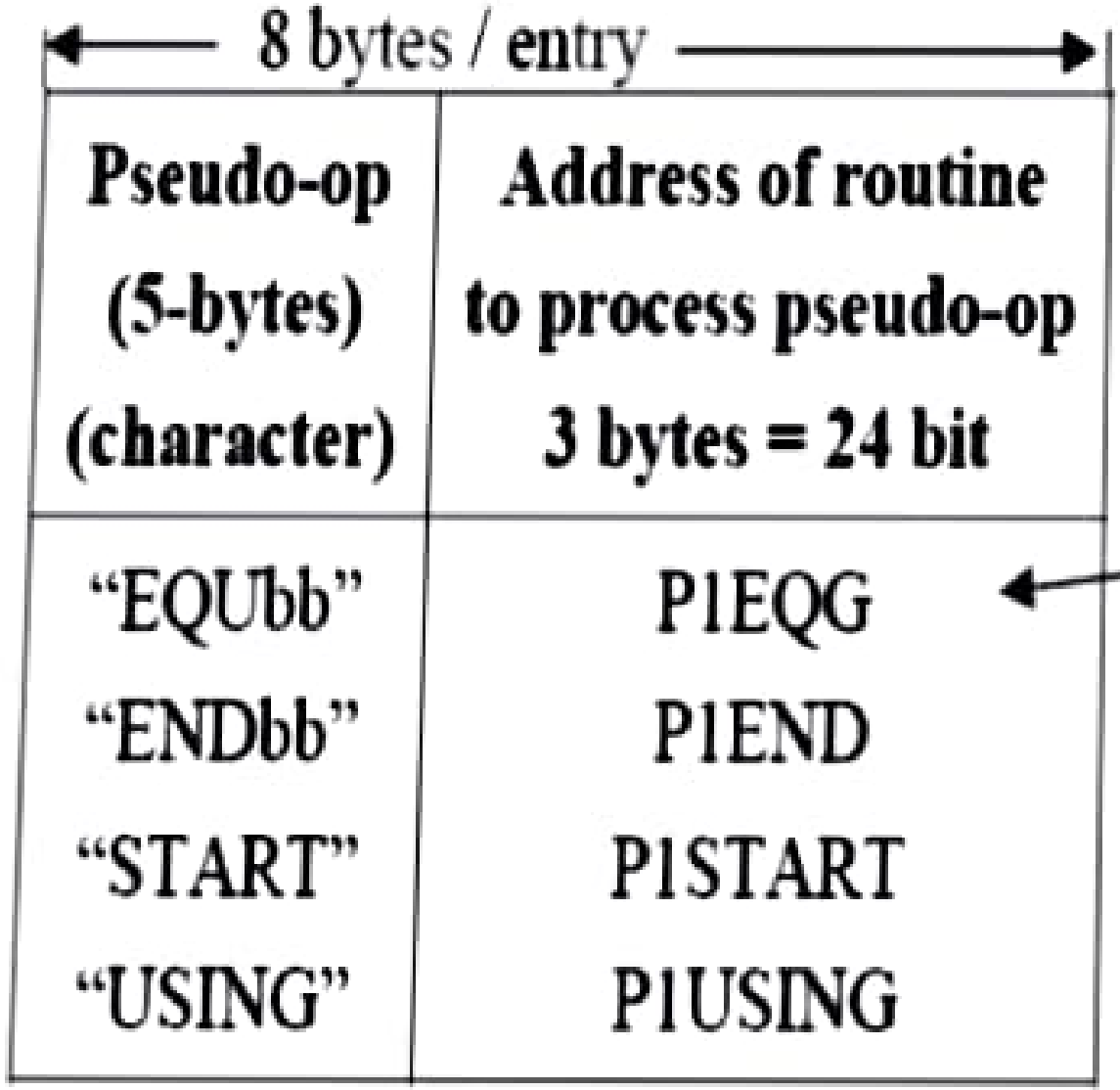
001 – RX

010 – RS

011 – SI

100 – SS

Pseudo Operation Table



Symbol table & Literal table:-

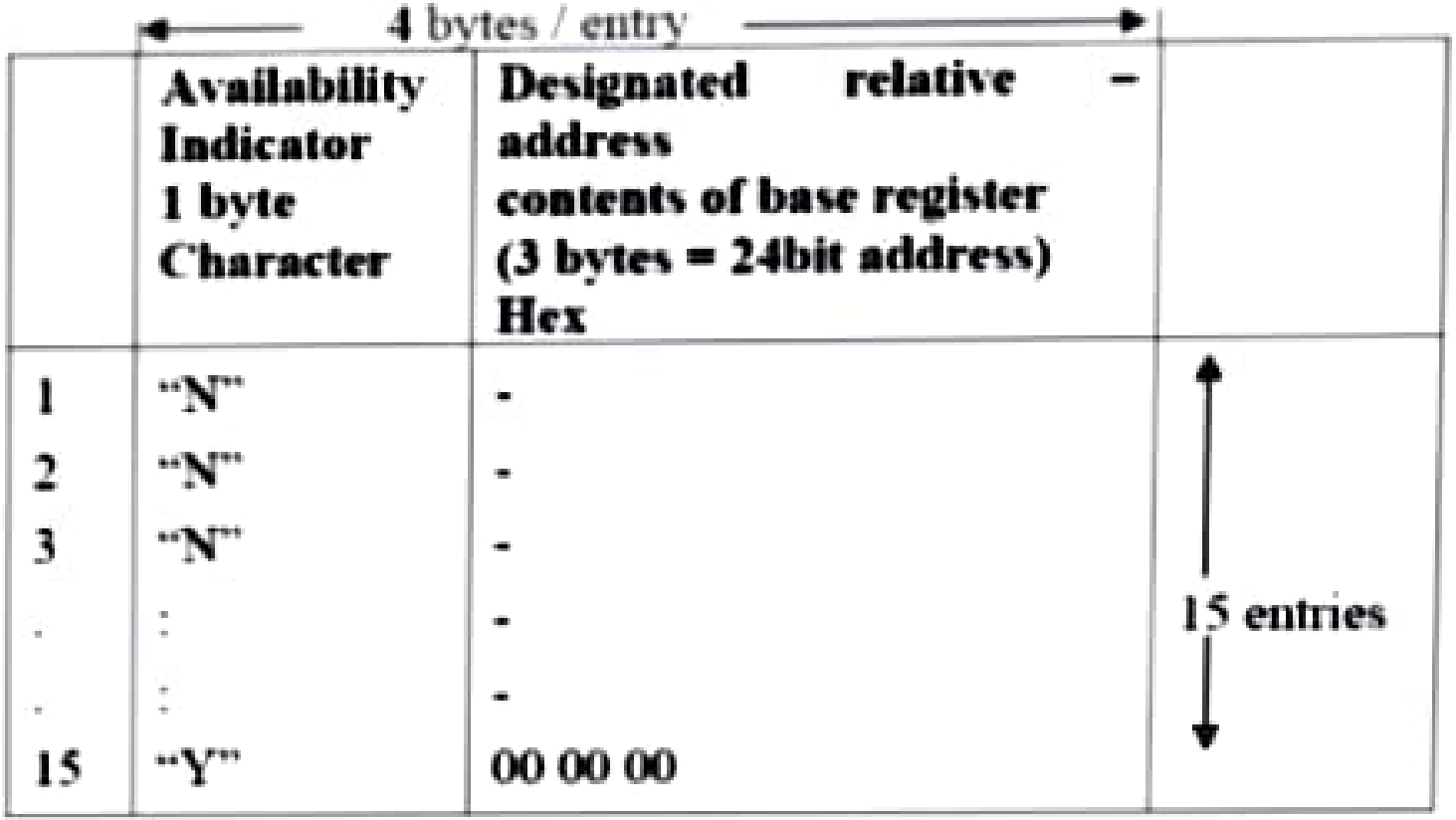
14 bytes / entry

Symbol 8 byte character	Value (4 byte) Hex	Length 1 byte Hex	Relocation 1 byte character
"LOOPbbbb"	0000	01	"R"
"JOHNbbbb"	0010	04	"R"

Relation – R- Relative

A- Absolute (i.e. does not change)

Base table



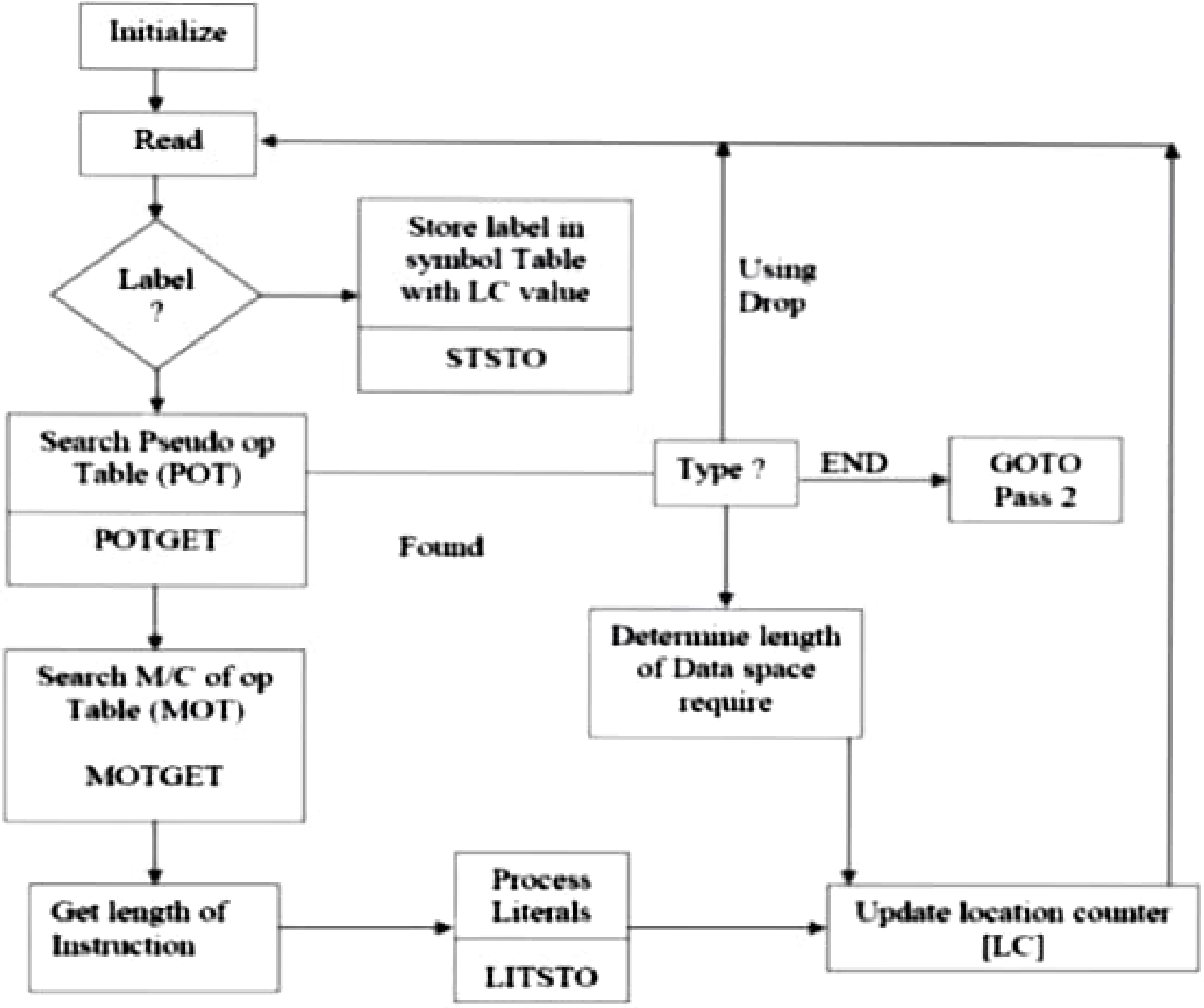
Codes:-

Availability:-

Y = register specified in USING pseudo-op

N = register never specified in USING pseudo-op or subsequently made unavailable by Drop pseudo-op.

Pass – I of ASSEMBLER


Initialize
Read
Label ?
Store label in symbol Table with LC value
STSTO
Using Drop
Search Pseudo op Table (POT)
POTGET
Found
Type ?
END
GOTO Pass 2
Search M/C of op Table (MOT)
MOTGET
Determine length of Data space require
Get length of Instruction
Process Literals
LITSTO
Update location counter [LC]

Pass-II
Assembler

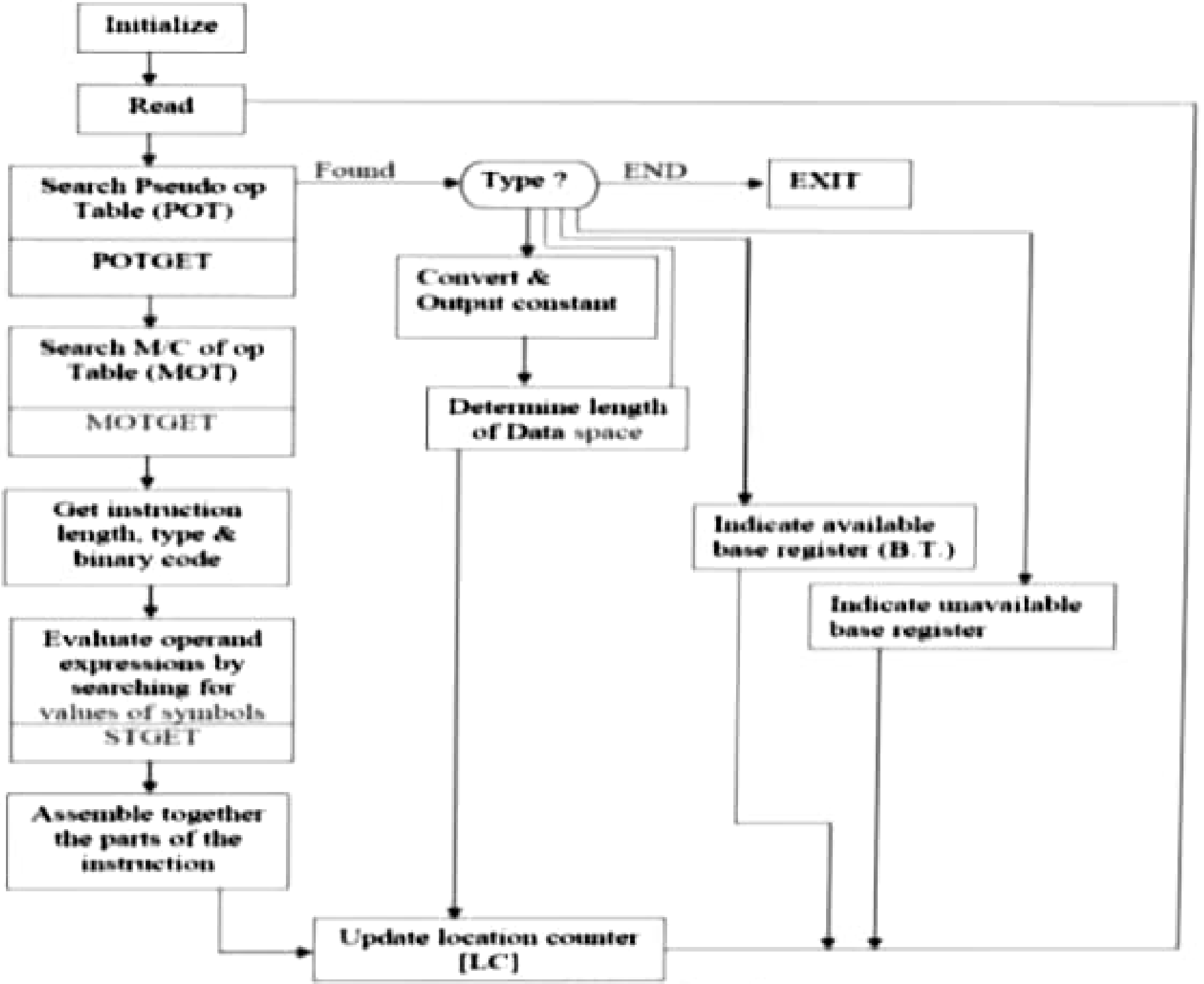


Fig. 1.5 Flow chart of Pass -2

Machine Dependent and Machine Independent features of Assembler

- M/C Dependent Features
  - **A] Instruction format & addr. mode:-**
  - **B] Program Relocation**
- Machine Independent Assembler Features
  - **1) Literals**
  - **2) Symbol defining statements**
  - **3) Expressions**