Title.....

Internal Commands :-

DOS

1. CLS (clear the screen) :- This command is used to clear the screen or wipe out everything written on the screen.

Syntax - C:\> CLS and press enter.

2. DIR (Directory) :- Dir command is used for listing files and directories present in the current disk.

Syntax - C:\> DIR [/switches]

E.g. - C:\> DIR /P

3. COPY :- Copy command is used for copy and any file to another location or to copy the files to another directory. This command may also be used for copying any file to another disk with different file name.

Syntax - C:\> COPY <Source filename> <Target file name>

C:\> COPY ROSE.TXT ROSE.MSG

1 file (s) copied

4. REN (Rename) :- This command is used to change the name of any file or directory.

Syntax - C:\> REN <Source filename> <Target filename>

5. DEL :- This command is used for erasing any file from the disk.

Syntax - C:\> DEL <Filename>

6. CD (Change Directory) :- We can enter or exit from any directory using this command.

Syntax - To access any directory
C:\> CD <Directory name>

7. MD (Make Directory) :- This command allows to create a new directory.

Syntax - C:\> MD <Dirname>

8. RD (Remove Directory) :- This command is used when we want to remove any unusable directory ~~from~~ from our disk.

Syntax - C:\> RD <Directory name>

9. TYPE :- This command is used ~~for~~ copy ~~any file to another~~ to display the contents or text of any file to the display device.

Syntax :- C:\> TYPE <Filename>

UNIX Internal Commands -

1. cp : Copy files

Syntax : cp [OPTION] Source destination.

E.g. Copies the contents from file 1 to file 2 and the contents of file 1 are retained.

- $ cp file1 file 2

2. rm : Remove files and directories.

Syntax : rm [OPTION]... [FILE]

Title.....

3. mv : MOVE files or rename files
   Syntax : mv [OPTION] source destination.

4. cd : Change directory.
   Syntax : cd [OPTION] directory.

5. mkdir : Make a directory
   Syntax : mkdir [OPTION] directory
   E.g. Create a directory called dir1
   $ mkdir dir1

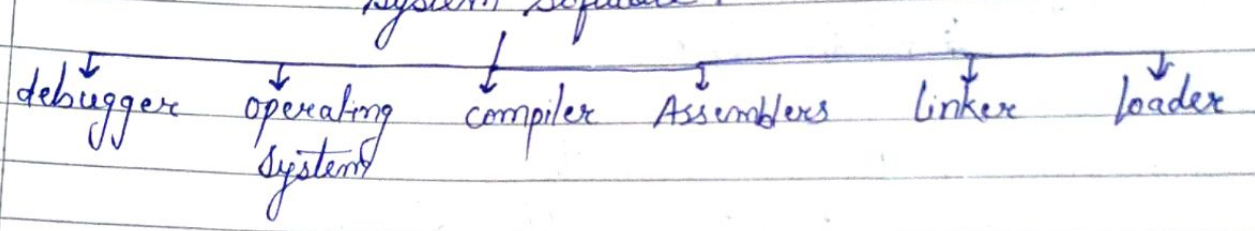
6. rmdir : Remove a directory.
   Syntax rmdir [OPTION] directory.

7. cat : Concatenate files and print to stdout.
   Syntax : cat [OPTION]... [FILE]

Title.

1. What do you mean by System Software. Describe briefly.

Ans System Software is a software that provides platform to other softwares. System software are programs used for functioning of computer system its maintenance and for providing the general user with very user friendly development environment.




Debugger :- A debugger is a software tool that can help the software development process by identifying coding errors at various stages of the operating system or application development.

Operating System :- An operating system is a type of system software that manage computers hardware and software resources. It controls and keeps a record of the execution of ~~the~~ all other programs that are present in the computer, including applications programs and other system software.

Compiler :- A compiler is a software that translates the code written in one language to some other language without changing the meaning of the program. The compiler is also said to make the target code efficient and optimized in terms of

time and space.
Examples of Compiler may include gcc (C compiler), g++ (C++ compiler), javac (Java Compiler) etc.

Assembler :- An assembler is a program that converts assembly language into machine code. It takes the basic commands and operations and converts them into binary code specific to a type of processor. Assemblers are more simplistic since they only convert low-level code to machine code.

Linker :- A linker is a useful utility tool that combines object files and other code created by the assembler & compiler into a single executable file.

Loader :- It is responsible for loading applications and libraries primary goal is to load executable files into the main memory.

Features Features of System Software –

1. It is very difficult to design.
2. It is smaller in size.
3. It is usually written in low-level language.
4. Fast speed.
5. Difficult to manipulate.
6. It must be as efficient as possible for the smooth functioning of the computer system.

Title.

2. What do you mean by language processor and describe their types with example?

Ans A special translator system software is used to translate the program written in a high-level language into machine code is called language processor.

The language processor can be any of the following three types :-

Compiler :- The language processor that reads the complete source program written in high-level language as a whole in one go and translates it into an equivalent program in machine language is called compiler.

Example - C, C++, Java

Source code (High-level language) → [Compiler] → Object code (Machine language)

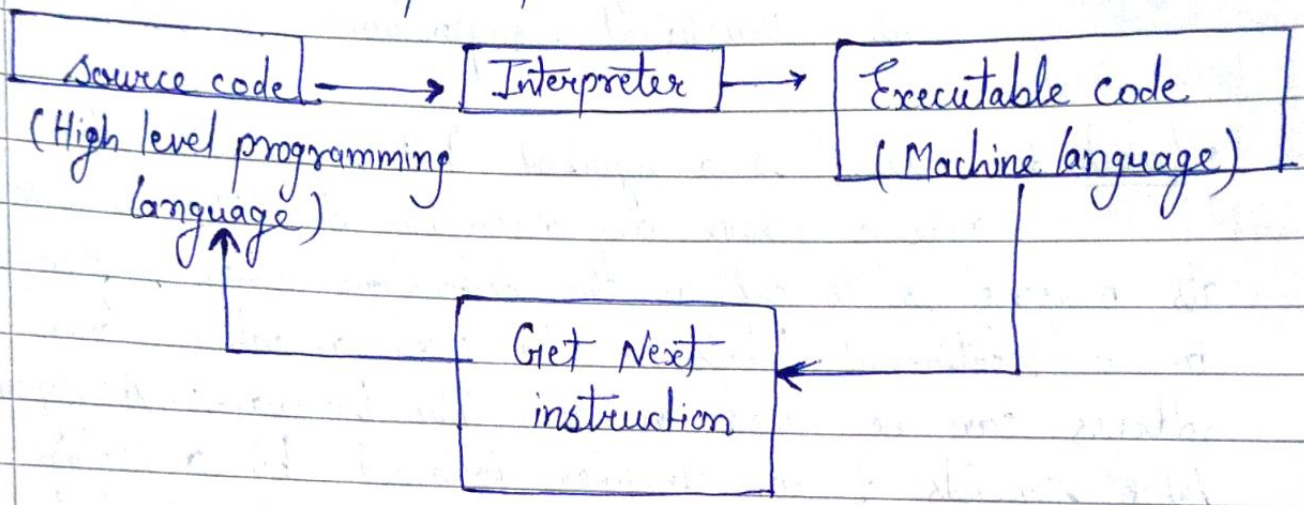
Assembler :- The assembler is used to translate the program written in assembly language into machine code. Assembler is the first interface that is able to communicate humans with machine.

Source code (Assembly language) → [Assembler] → Object code (Machine language)

Title.

Interpreter :- The translations of a single statement of the source program into machine code is done by a language processor and executes immediately before moving on to the next line is called an interpreter.

Example - Ruby, Python., PHP etc.




3. What are the elements of Assembly language. Describe them briefly.

Ans An Assembly language is a programming language that communicates with the hardware of a computer directly.

It is a low-level programming language helps in understanding the programming language to machine code.

Elements of Assembly language are -

- Syntax
- Label

Title.

- Command
- Mnemonic Opcode
- Operands
- Directive
- Macro

- Syntax :- The syntax of an assembly language is its structural composition.

- Label :- A label is a symbol that represents the address where an instruction or data is stored. Its purpose is to act as the destination when referenced in a statement. Labels can be used anywhere an address can be used in assembly languages. A Symbolic label consists of an identifier followed by a colon, while numeric labels consists of a single digital followed by a colon.

example - count

```
count  DWORD  100
↓
label
```

- Command :- A command is a ~~cod~~ order in a piece of assembly code that tells the assemble what action to perform.

- Mnemonic Opcode :- A Mnemonic is an abbreviation for an operation. A mnemonic is entered into the operation code for each assemble program instruction to specify a shortened

"opcode" that represents a larger, complete set of codes.

Examples : MOV, ADD, SUB, MUL, CALL
MOV Move one value to another
ADD ADD two values

- Operands :- An Operand is component the assembler can manipulate such as a variable ore piece of data by including a operands in instruction, you can tell the assembly language to which data to apply any command on that line. Operands depends on the instruction.

- Directive :- A directive is a type of statement that provides instruction to the assembler to change a setting or perform an action.

4. What are the type of statements in Assembly language. Describe them with example.

Ans There are basically three types of statements in an assembly language.

- Declarative statements
- Impereative statements
- Assembler directive

Title.....

Page No.....

• Declarative statements :- These statements are used for reserves areas of memory and associates names with them. The declarative statement constructs memory word containing constants. The Syntax of declarative statements is as follows:

[label] DS <Constant specifying the size of memory to be reserved>

e.g. A DS 1

[label] DC <value specifying the initial value of variable>

e.g. A DC '5'

• Imperative statements :- These are the executable statements where each statement indicates an action to be taken during the execution of the program. Also each statement translates into one machine instruction.

Example – MOVER X
READ A
PRINT Y
ADD B.

• Assembler directive :- These instruct the assembler to perform certain actions during the assembly of a program we can say that directives instructions act as direction for the assembler.

Example - START <constant>
END <operand specification>

Here START indicates that the first word of the machine code should be placed in memory word with address in <constant>. Here END indicates the end of the source program. The optional <operand spec> indicates the address of the instruction where the execution of the program should begin.

5. What do you mean by Literals. Describe it.

Ans A literal is an operand with the syntax = '<value>'. It differs from a constant because its location cannot be specified in the assembly program.

→ This helps to specify ensure that its value is not changed during execution of a program.

→ When the assembler encounters the use of a literals in the operand field of a statement, it handles the literals using an arrangement similar to FIVE DC '5' - it allocates the memory word to contain the value of the literal, and replaces the use of the literals in a statements by an operands expression ~~referencing~~ referring to this word. The value of the literals is protected by the fact that the name and address of this word is not know to the assembly language programmer.

Title..... Page No.....12.....

6. Write difference between Single pass Assembler and Two pass Assembler with their diagram.

| Single Pass Assembler | Two Pass Assembler |
|---|---|
| 1. It perform translation in one pass. | 1. It perform translation in two pass. |
| 2. Intermediate code not generated. | 2. Generation of intermediate code. |
| 3. Forward referencing is handled by back patching. | 3. After pass one, all Symbols and literals are getting address. |
| 4. Back patching is handled by TII (Table of incomplete instruction | 4. No need of back patching. |
| 5. Default addresses are zero for symbols and literals later on updated to actual addresses. | 5. After pass one, all Symbols and literals are getting addresses. |
| 6. More memory required compared to two pass assembler. | 6. Less memory required compare to single pass assembler. |
| 7. Data structures used: Symbol table, literal table, Pool Table, TII. | 7. Data structure used: Symbol Table, literal Table, Pool Table. |
| 8. It is faster than two pass assemble. | 8. It is slower than Single pass assembler. |

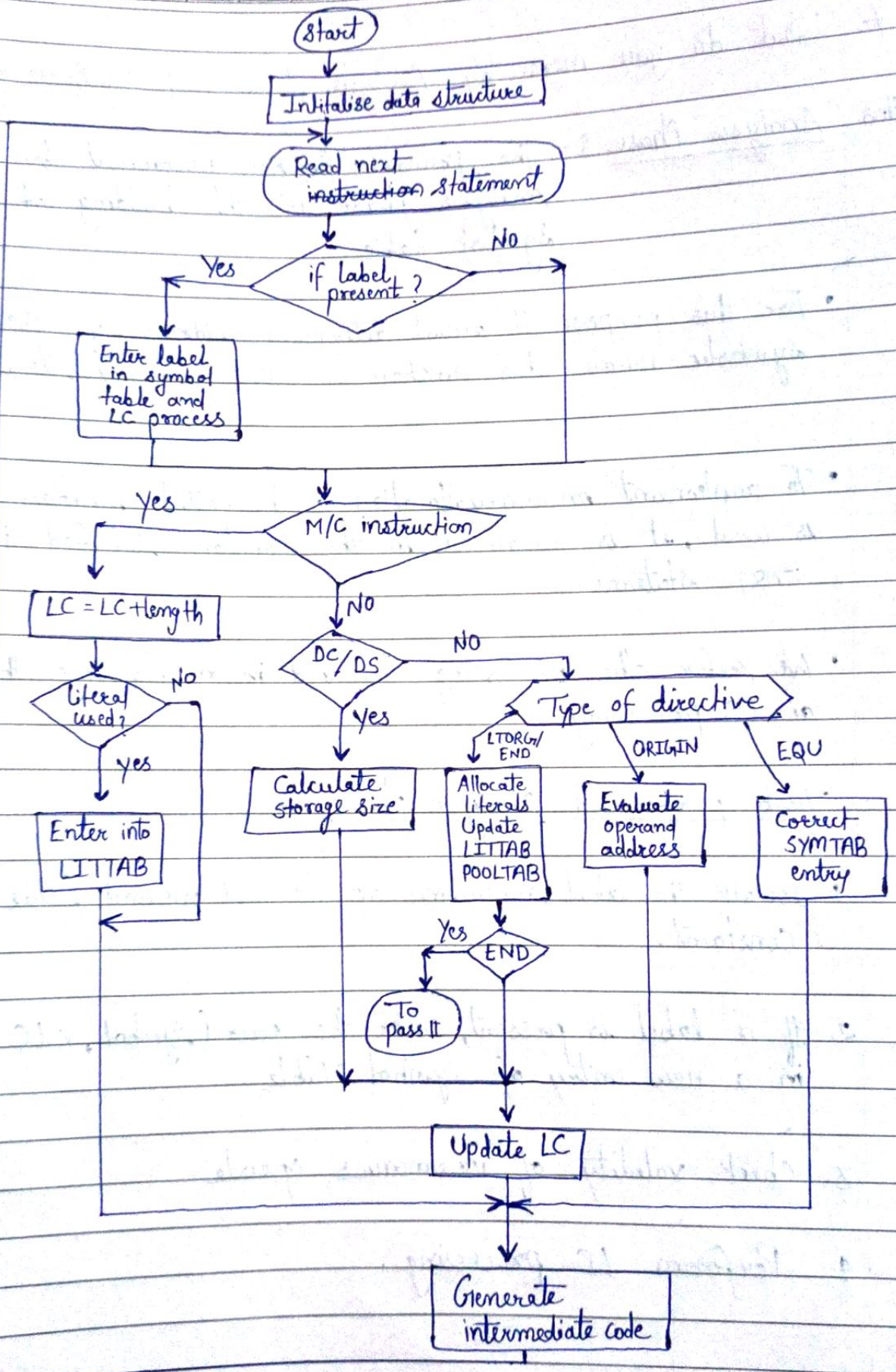



fig. 6.1 Pass I of the Assembler

7. What do you mean by Analysis phase and Synthesis phase?

Ans **Analysis Phase** :- The primary function performed by the Analysis phase is the building of the symbol table.

- For this purpose it must determine address of the symbolic name. This function is called memory allocation.

- To implement memory allocation a ds called location counter is used, it is initialized to the constant specified in the START statement.

- We refer the processing involved in maintaining the LC as LC processing.

**Task performed Analysis phase**

- 1. Isolate the label; mnemonic opcode and operand fields of a Constant.

2. If a label is present, enter the pair (symbol, <LC content>) in a new entry of symbol Table.

3. Check validity of mnemonics opcode.

4. Perform LC processing.

Synthesis Phase :- The Synthesis phase creates an equivalent target program from the intermediate representation.

Task of Synthesis Phase

- It refers to the mnemonic Table to obtain the machine opcode corresponding to the mnemonics.
- It refers to the symbol Table which is constructed during Analysis phase. It obtains the address of the memory operand from the symbol Table.

8. What are MACRO, define it and also define the macro ~~prepocessor~~ preprocessor.

Ans A macro is a sequence of instructions, assigned by a name and could be used anywhere in the program.

A macro is a unit of specification for program generation through expansion. A macro consists of a name, a set of formal parameters and body of code.

The use of a macro name with a set of actual parameters is replaced by some code generated from its body. This is called macro expansion.

Example –

```
MACRO       ------ Start of definition
INCR        ------ Macro name

A1, DATA
A2, DATA    ..... Sequence of instructions to be
A3, DATA          abbreviated.

MEND        ------ End of definition.
```

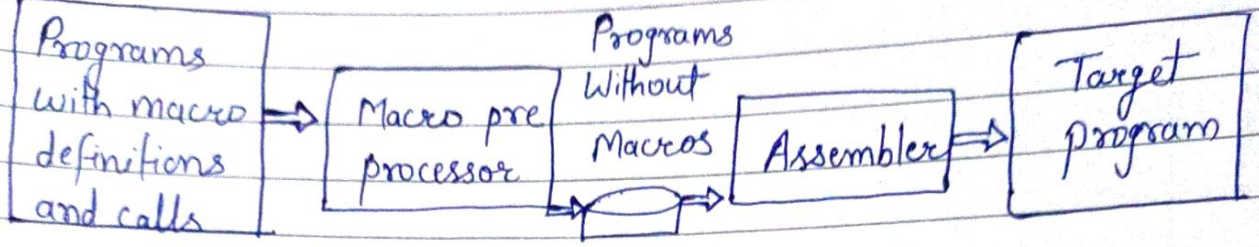
Features of Macros –

- Macro can make tasks less repetitive.
- Libraries of useful macros can be created.
- It is used to build up complex operations out of simpler operations.

Macro Preprocessor –

The macro preprocessor accepts an assembly program containing definitions and calls and translates it into an assembly program which does not contain any macro definitions and calls.

The program form output by the macro pre-processor can be handed over to an assembler to obtain the target program.




Functions of macro preprocessor —

- Identifies macro definitions and calls in the program.
- Determines the values of formal parameters.
- Maintain the values of expansion time variables declared in a macro.
- Organize expansion time control flow.
- Determine the values of sequencing symbols.
- Perform expansion of a model statement.