Course Contents		
Unit I	**Introduction**	**08 Hours**

Introduction to Systems Programming, Need of Systems Programming, Software Hierarchy, Types of software: system software and application software, Machine structure.
Evolution of components of Systems Programming: Text Editors, Assembler, Macros, Compiler, Interpreter, Loader, Linker, Debugger, Device Drivers, Operating System. **Elements of Assembly Language Programming**: Assembly Language statements, Benefits of Assembly Language, A simple Assembly scheme, Pass Structure of Assembler.
Design of two pass Assembler: Processing of declaration statements, Assembler Directives and imperative statements, Advanced Assembler Directives, Intermediate code forms, Pass I and Pass II of two pass Assembler.

#Exemplar/Case Studies	Study of Debugging tools like GDB
***Mapping of Course Outcomes for Unit I**	CO1, CO2, CO3

Course Outcome

To **analyze** & **synthesize** various system software & understand the design of two pass assemblers.

Introduction

- System is Collection of Component. e.g.College
- Programming is way to instruct the computer to perform various task.
- system programming is an art of designing and implementing system Programs

Outline

Introduction

Assemblers

Software

SNJB

System introduction

SPOS

Unit- I

system programming is an art of designing and implementing system Programs.

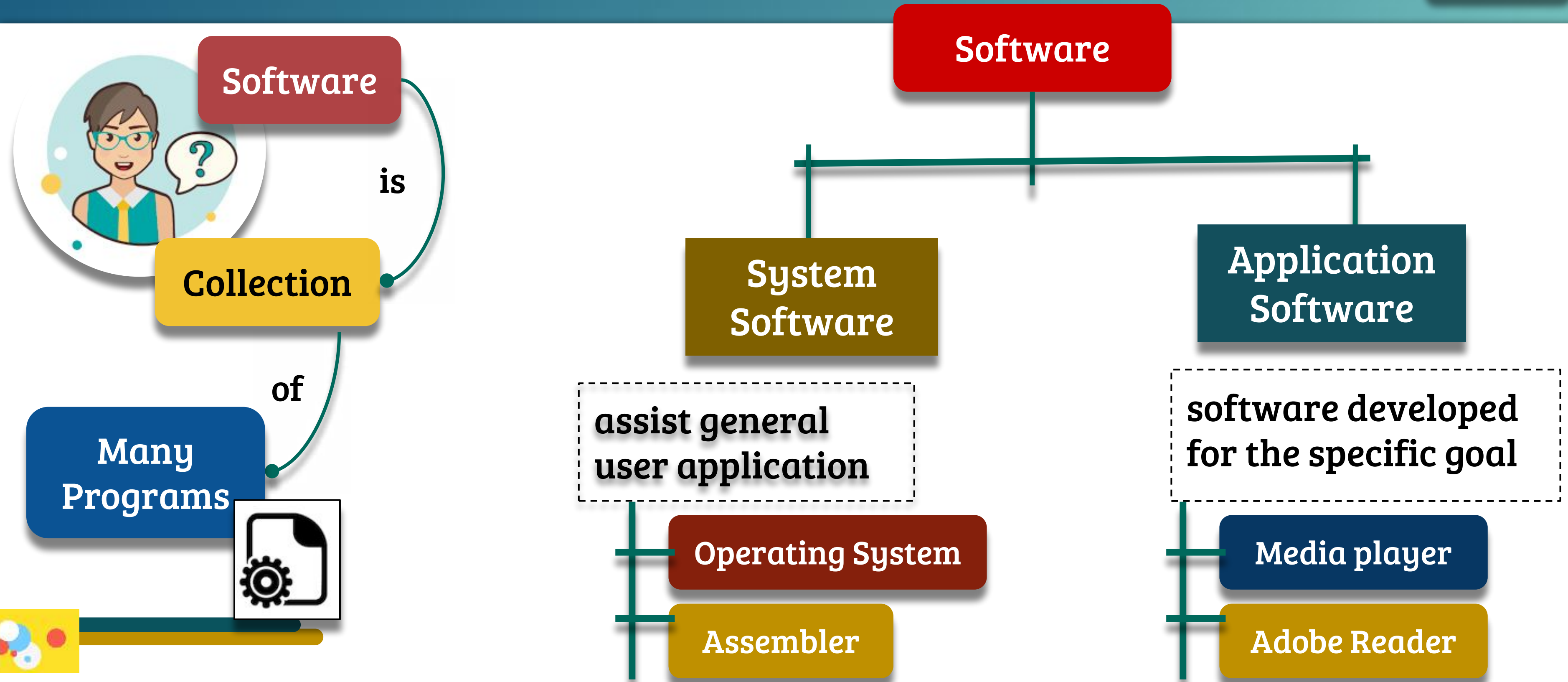
System is Collection of Component

eg. College

Programming is Way to instruct the compute To perform Various Task

Task

Software Hierarchy



Foundation of system Programming

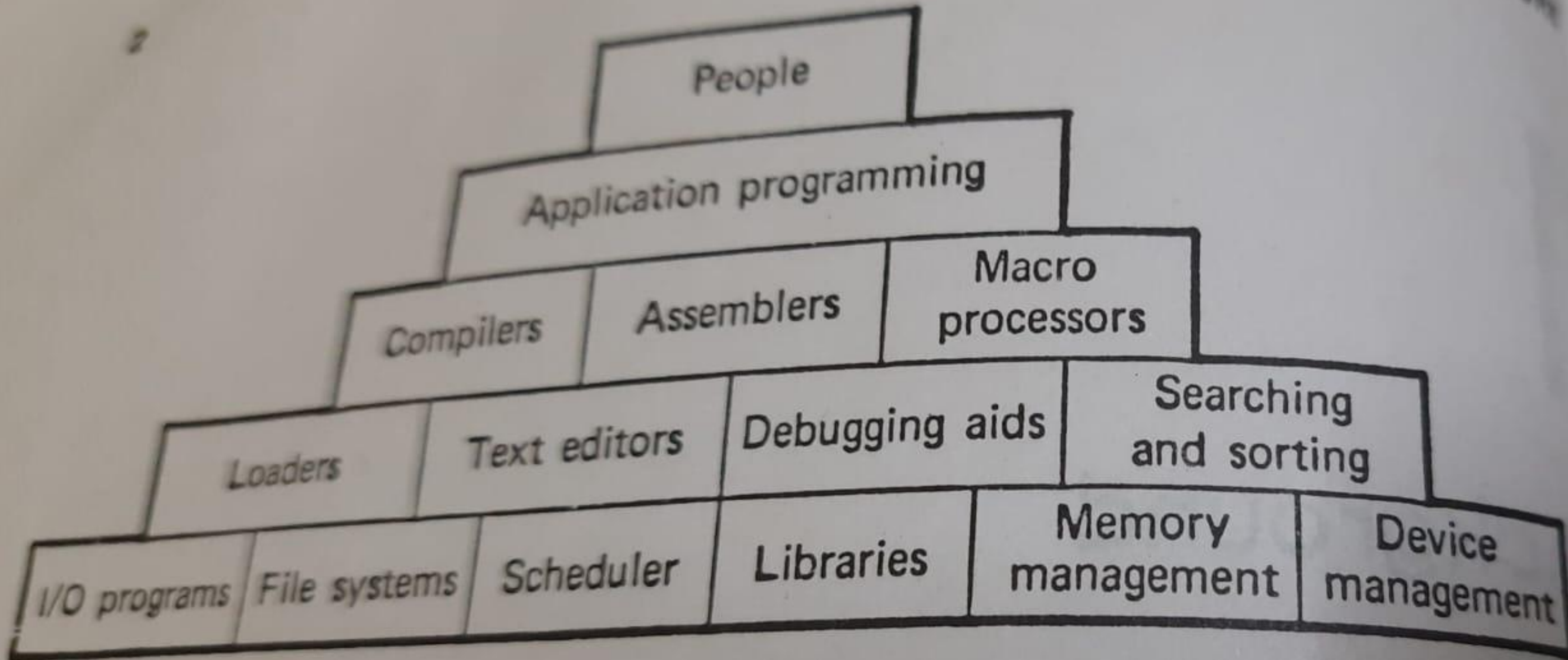


FIGURE 1.1 Foundations of systems programming

General Machine Structure

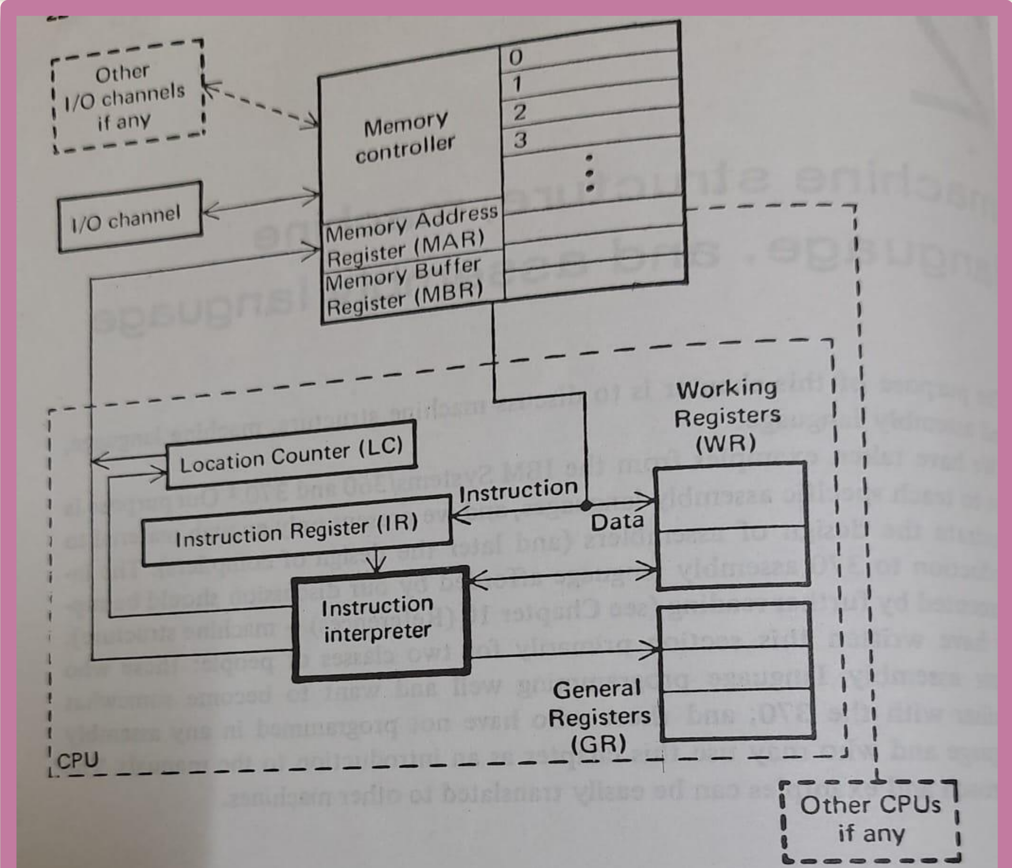


FIGURE 2.1 General machine structure

Need Of System Software

Hardware Management:

- System software, particularly the operating system (OS), acts as an intermediary between the hardware and the user. It manages hardware resources such as the CPU, memory, storage devices, and input/output devices.
- It ensures efficient and fair allocation of resources among various applications.

Application Support:

- Provides a platform for running application software.
- Offers necessary services and libraries required by applications to function correctly.

Network Management:

- Facilitates networking capabilities, enabling communication between computers and other devices.
- Manages network connections, data transmission, and network security.

Need Of System Software

Device Drivers:

- Includes device drivers that facilitate communication between the OS and hardware devices.
- Ensures proper functioning and compatibility of peripheral devices like printers, scanners, and network adapters.

Software Updates and Maintenance:

- Handles software updates, patches, and maintenance tasks to keep the system secure and up-to-date.
- Ensures that the system software remains compatible with new hardware and applications.

File Management:

- Organizes and manages files on storage devices.
- Provides functionalities like file creation, deletion, reading, writing, and permissions management.

Sr. No.	Key	System Software.	Application Software.
1	Definition	System Software is the type of software which is the interface between application software and system.	On other hand Application Software is the type of software which runs as per user request. It runs on the platform which is provide by system software.
2	Development Language	In general System software are developed in low level language which is more compatible with the system hardware in order to interact with.	While in case of Application software high level language is used for their development as they are developed as some specific purpose software.
3	Usage	System software is used for operating computer hardware.	On other hand Application software is used by user to perform specific task.
4	Installation	System software are installed on the computer when operating system is installed.	On other hand Application software are installed according to user's requirements.
5	User interaction	As mentioned in above points system software are specific to system hardware so less or no user interaction available in case of system software.	On other hand in application software user can interacts with it as user interface is available in this case.
6	Dependency	System software can run independently. It provides platform for running application software.	On other hand in application software can't run independently. They can't run without the presence of system software..
7	Examples	Some examples of system software's are compiler, assembler, debugger, driver, etc.	On other hand some examples of application software's are word processor, web browser, media player, etc.

System introduction

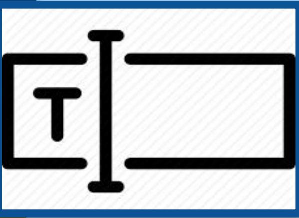
System Program are Required for Effective Execution of General user Programs on Computer System

System Programming Is an art of designing and implementing system programs

Components of Systems Programming:

- Text Editors,
- Loader and Linker
- Assembler,
- Compiler,
- Macros,
- Debugger,
- Interpreter,
- Device Drivers,
- Operating System.

Text Editor is **program** Used for **Editing plain text files**



With the help of **Text Editor** You can **Write Your Program**

Example: **C | Java Prog.**

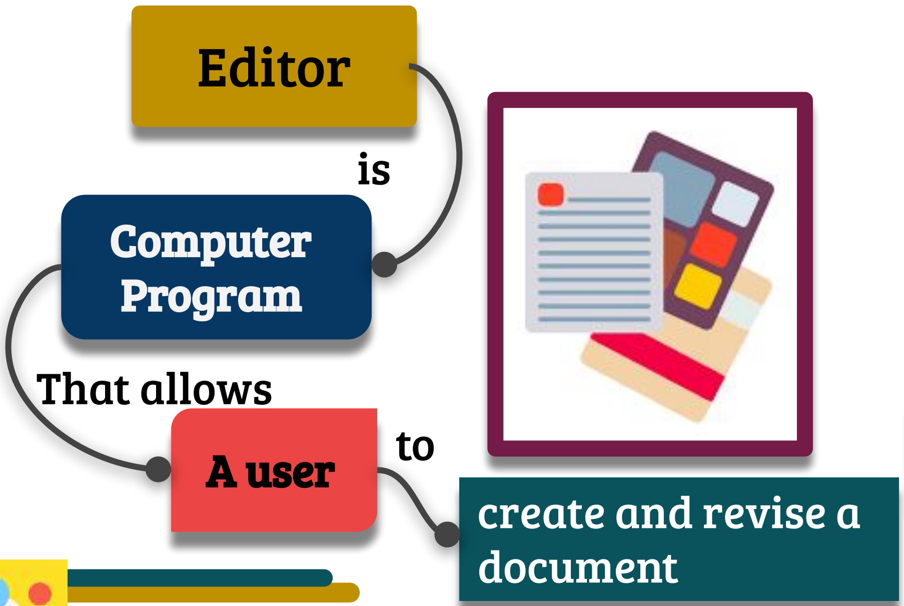
System Software

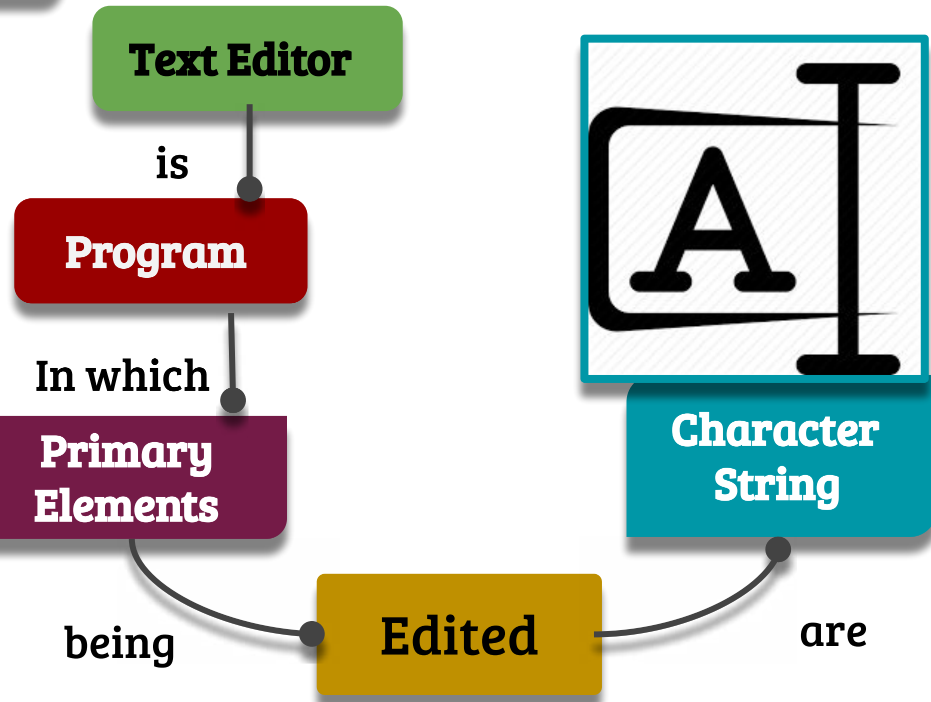
SPOS

Unit- I

Text Editor

Notepad





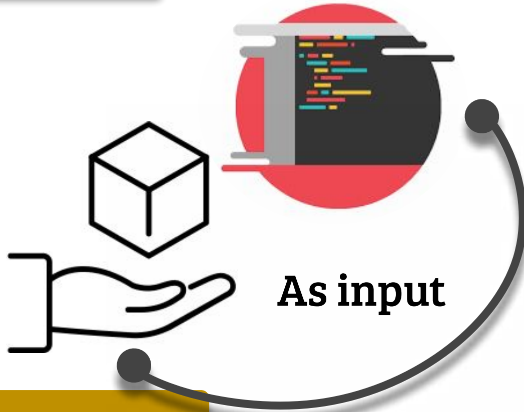
System Software

Loders

is

Program

That takes



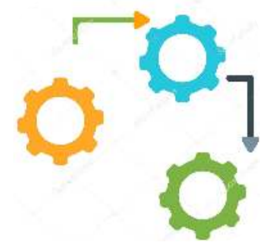
As input

Object Code

and

Prepares

Them for



Execution

Initiates

it

Execution

SNJB

SPOS

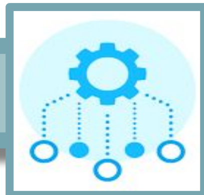
System Software

Unit- I

Loders

Functions

Allocation



Linking

Relocation

Loading

Loders

Functions

Allocation

Loader allocates space for programs in main memory.

Loders

Functions

Relocation

- Adjusting all address dependent location.
- E.g. If we have two Programs Program A and Program B.
- Program A is saved at location 100.
- And user wants to save Program B on same location. That is physically not possible.
- So loader relocates program B to some another free location

Functions

Linking

- If we have different modules of our program.
- Loader links object modules with each other.

Loders

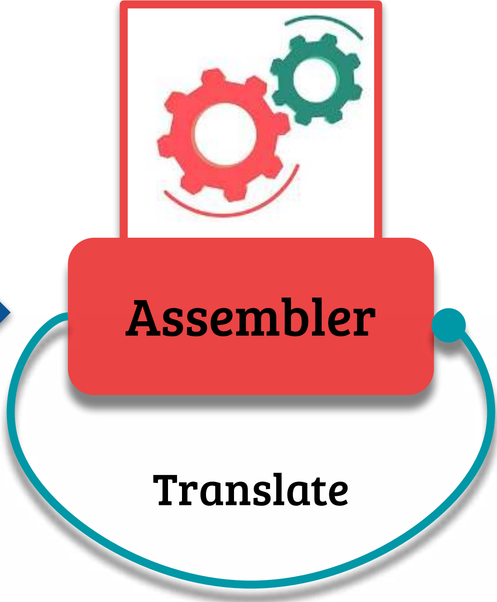
Functions

Loading

Physically loading the machine instructions and data into main memory.

Assembler

Assembly Lang. Program



Machine Lang.

Macro Processor

Macro

Allows

Sequence

of

Source Lang. Code

To be

Defined @ once

&

Referred many times

Macro Processor

Syntax

Macro Macro name [set of parameters]

// macro body

Mend

★ A macro processor takes a source with macro definition and macro calls and replaces each macro call with its body

5. Macro Processor

- It allows the programmer to write shorthand version of a program .

- Macro allows a sequence of source language code to be defined once and then referred to by name each time it is to be referred.
-
- Each time this name Occurs in a program, the sequence of codes is substituted at that point.

Macro code -- Example

Source

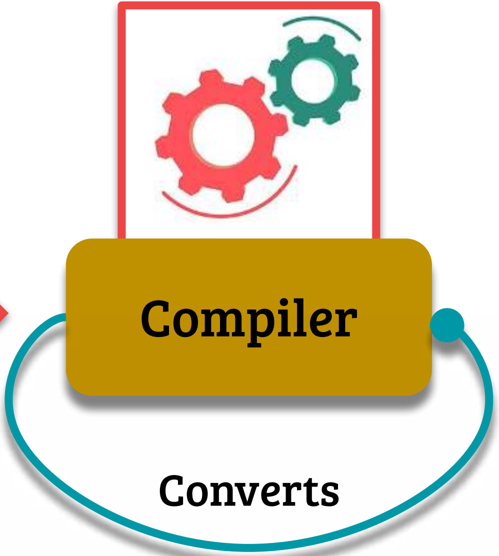
```
   MACRO   STRG
    STA DATA1
    STB DATA2
    STX     DATA3
    MEND
   .
STRG
   .
STRG
   .
   .
```

Expanded source

```
   .
   .
{  STA DATA1
   STB DATA2
   STX DATA3
   .
{  STA DATA1
   STB DATA2
   STX DATA3
   .
```

Compiler

High Level Lang.



Low Level Lang.

Compiler

Benefits of writing a program in a high level language

Increases productivity

It is very easy to write a program in a high level language

Machine Independence

A program written in a high level language is machine independent.

Debugger

Debugging tool helps programmer for testing and debugging programs

It provides some facilities:

- Setting breakpoints.
- Displaying values of variables.

7. Interpreter

- A **Interpreter** reads the source code **one** instruction or line at a this line into machine code or some intermediate form and **executes it.**

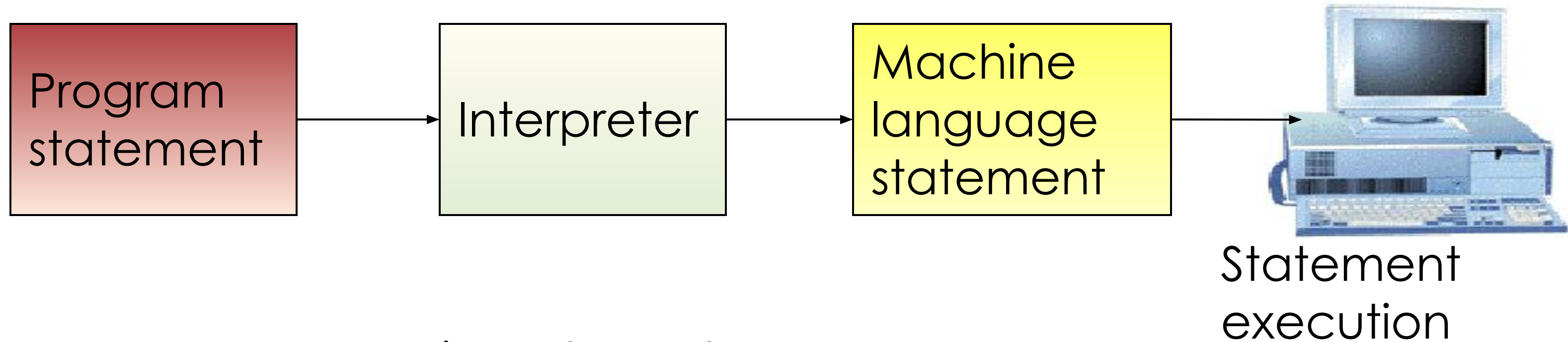


Fig.: Interpreter

8. Operating system (OS)

- An **operating system** (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.

9. Device driver (OS)

- **Device driver is a computer program that operates or controls a particular type of device that is attached to a computer .**

Assembly Language

Assembly Language

- Assembly language is middle level language.
- An assembly language is machine dependent.
- It differs from computer to computer.
- Writing programs in assembly language is very easy as compared to machine(binary) language

- Assembly lang. a **symbolic representation** of machine language.
- uses a **mnemonic** to represent each low-level machine instruction or operation.
- Assemblers with different syntax for a particular CPU or instruction set architecture.
- **Example**:- An instruction to add memory data to a register

  x86-family processor: **add eax,[ebx],**

  whereas this would be written **addl (%ebx),%eax**

  in the *AT&T syntax* used by the GNU Assembler.

Assembly Language

Assembly language programming Terms

(LC)

Location Counter points to the next instruction

Literals Constant Values

Assembly Language

Assembly language programming Terms

Symbols Name of variables and labels

Procedures Methods | Function

Elements of assembly language programs:

A. Basic features
B. Statement format
C. Operation code

A.Basic features

- **Assembly lang. Provides 3 basic features:**

1. Mnemonic Operation Codes(Opcodes)

   Ex: MOVER or MOVEM

2. Symbolic Operand:

   Ex: DS – Declare as storage

   DC – Declare as Constant

3. Data Declaration:

   Ex: X DC '-10.5'

B.Statement Format

- Statement Format:

[Label] <opcode> <operand1> [<operand2>..]

Label-Optional

Opcode- it contain **symbolic** operation code

Operand- Operand can also be a CPU register: AREG, BREG,CREG.

Example-

LOOP : MOVER AREG, '=5'

Machine supports 11 Different Operations

Symbolic opcode	Remark
STOP	Stop Excecution
ADD	Operand □ Oper1+Oper2
SUB	Operand □ Oper1- Oper2
MULT	Operand □Oper1*Oper2
MOVER	CPU Register□Memory move
MOVEM	Memory operand□CPU register
COMP	Set condition code Comparison instruction sets a condition code The condition code can be tested by BC
BC	Branch on condition **Format for BC** : **BC <condition code spec.>, <Memory address>** 1. **LT**-Less than 2. **LE**-Less or equal 3. **EQ**-Equal 4. **GT**-Greater than 5. **GE**-Greater or equal 6. **ANY**-Implies unconditional transfer of control

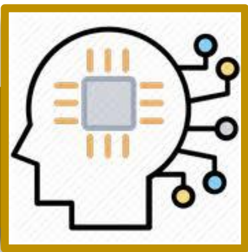
Machine supports 11 Different Operations

Symbolic opcode	Remark
DIV	Operand Oper1/Oper2
READ	Operand2 input value
PRINT	Output operand2

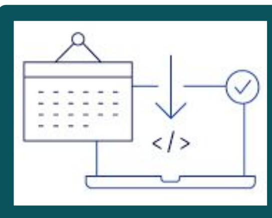
First operand is always a **CPU register**
Second operand is always a **memory operand**

Assembly language Statements:

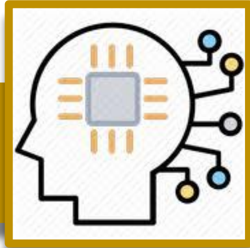
Imperative Statements



Declarative/Declaration Statements



Assembler Directive



Imperative Statements

- Imperative means mnemonics
- These are executable statements.
- Each imperative statement indicates an action to be taken during execution of the program

```
E.g.  MOVER BREG, X
      STOP
      READ X
      ADD AREG, Z
```

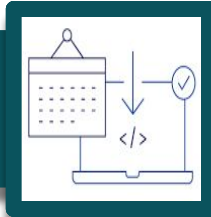
SNJB

Assembly Language

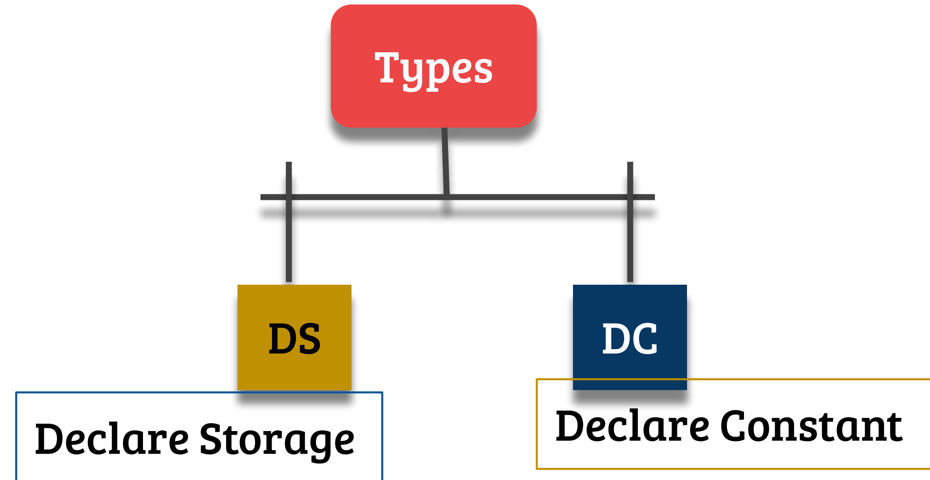
SPOS

Unit- I

Declarative/Declaration Statements



- Declaration statements are for reserving memory for variables.
- We can specify the initial value of a variable.



Declare Storage

Syntax

[Label] DS < Constraint Specifying size >

Example

X DS 1

Declare Constant

Syntax

[Label] DC < Constraint Specifying values >

Example

X DC ‘ 5 ’

3.Assembler Directive

- Instructs the assembler **to perform** certain actions during assembly of a program.
  - A **directive** is a direction for the assembler
  - A **directive** is also known as pseudo instruction
  - **machine code** is **not generated** for AD.

3.Assembler Directive…

- **START <Constant>**
- It indicates that the **first word** of the m/c code should be placed in the memory word with the address **<CONSTANT>**

3.Assembler Directive…

- END [<OPERAND SPECIFICATION>]

- Optional, indicates address of the instruction where the address of program should begin.
- By default, execution begins from the first instruction.
- It indicates the end of the source program.
- Class:AD

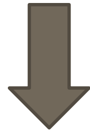
Advanced Assembler Directives

- These directive include:

1. **ORIGIN**
2. **EQU**
3. **LTORG**

ORIGIN

- Useful when m/c code is not stored in consecutive memory location.
- **ORIGIN <address specification>**



Operand or **constant** or **expression** containing an operand and a constant.

- Sets LC to the address given by **<address specification>**
- LC **processing in a relative** rather than absolute manner

ORGIN.....Example

Sr.No	Assembly Program	LC	Remark
1	START 100		ORGIN LOOP+5, Set LC to the value 106 (101+5=106) Here, LOOP associated with address 101
2	MOVER BREG,'=2'	100	
3	LOOP MOVER AREG,N	101	
4	ADD BREG,'=1'	102	
5	ORGIN LOOP+5		
6	NEXT BC ANY,LOOP	106	ORGIN NEXT+2 Sets LC to the value 108 (106+2=108) Here, NEXT associated with address 106
7	ORGIN NEXT+2		
8	LAST STOP	108	
9	N DC '5'	109	
10	END		

EQU

Syntax:

<symbol> **EQU** <address specification>

Where,

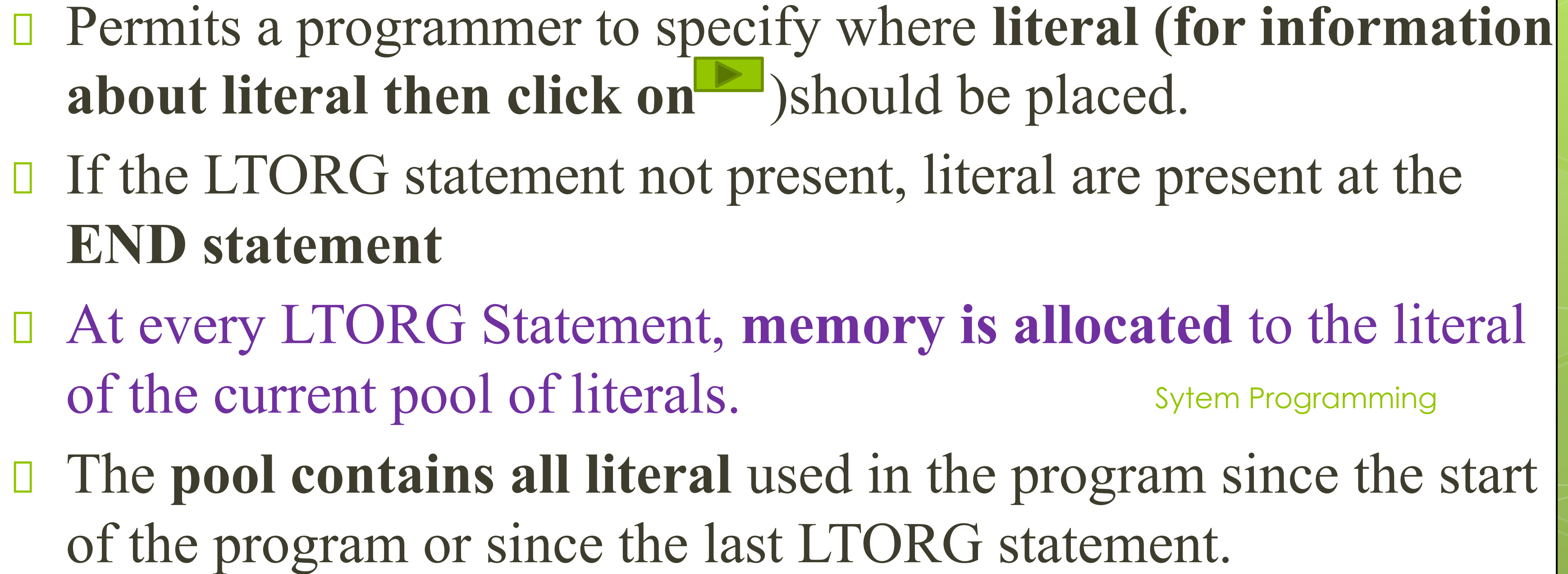
<address specification> :can be **operand specification** or a **constant**

<symbol>**: EQU** Associates **symbol** with the <address specification>

Ex. BACK EQU LOOP

The symbol **BACK** is set to the address of **LOOP**

LTORG



Literal and Constant

- A literal is an immediate operand
- A literal is an operand with constant value.

In the c-statement

```
int z=5;
x=y+5;
```

The constant value is ‘5’ known as literal.

- Literal can not be change during program execution
- They are specified using immediate addressing.

Literal and Constant...

- Literal in assembly language:
- Assembly instruction for 8086 with immediate operand

- **MOV AX 15 (8086 instruction)**

- But Hypothetical machine does **not support** immediate operand.

Literal and Constant...

- Handling a literal by our machine is as follows:

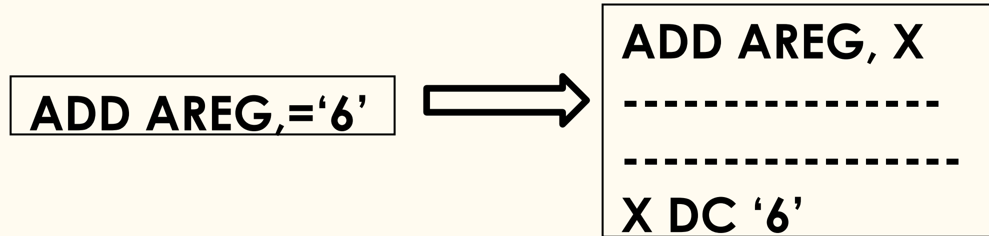


Fig.: Handling of literal

Sample Assembly language Code

```
1.  START 100
2.  MOVER A REG, X
3.  MOVER B REG, Y
4.  ADD A REG, Y
5.  MOVEM A REG, X
6.  X DC '10'
7.  Y DS 1
8.  END
```

SNJB

Assembly Language

SPOS

Code

Unit- I

Identify types of statement

1. START 100
2. MOVER B REG, Y
3. MOVER B REG, Y
4. ADD A REG, Y

Sr. No	IS	DS	AD
1.			
2.			
3.			
4.			

SNJB

Assembly Language

SPOS

Code

Unit- I

Identify types of statement

5. MOVEM A REG, X
6. X DC ‘10’
7. Y DS 1
8. End

Sr. No	IS	DS	AD
5.			
6.			
7.			
8.			

Types of Assembler

- 3 Types of Assemblers

1. Load and Go-Assembler
2. One-pass Assembler
3. Two-pass Assembler

1.Load and Go-Assembler

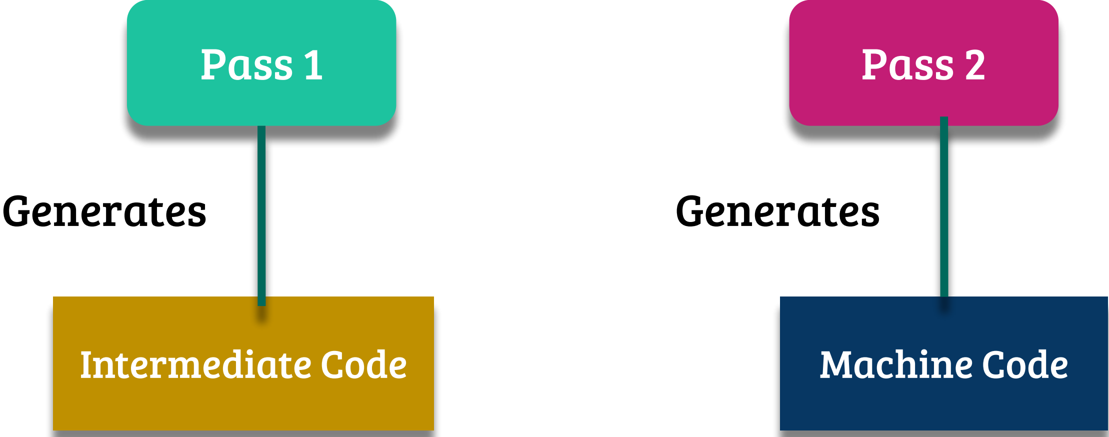
- Simplest form of assembler
- It produces machine language as output which are **loaded directly** in main memory and executed
- The ability to **design code** and test the different program components in parallel

2.One Pass Assembler

- Normally , it does **not allow** forward referencing.
- An assembler **cannot generate m/c code** for an assembly instruction with FR.
- Machine code is generated ,after the **address of variable** used in the instruction is known.
- Symbol table is used to record the address of the variables.

Sr. No	Pass 1	Pass 2
01	It requires only one scan to generate machine code	It requires two scan to generate machine code.
02	It has forward reference problem.	It don't have forward reference problem.
03	It performs analysis of source program and synthesis of the intermediate code.	It process the IC to synthesize the target program.
04	It is faster than pass 2.	It is slow as compared to pass 1.

Output of Pass 1 | Pass 2 Assembler



Two Pass Assembler

Pass 1 Assembler

- Separate
  - Labels
  - Mnemonic Opcode
  - Operand Fields
- Determine Storage Requirement
- Build the Symbol Table

Pass 2 Assembler

- Generate the machine code

Working of pass I

- Data structures required:
- MOT-use to search the opcode
- Symbol table-use to search the symbol
- Literal table
- Pool table : starting literal number of each pool.

Mnemonic Opcode Table(MOT)

Mnemonic opcode	m/c code for opcode	Class	Size of instructions
STOP	00	IS	1
ADD	01	IS	1
SUB	02	IS	1
MULT	03	IS	1
MOVER	04	IS	1
MOVEM	05	IS	1
COMP	06	IS	1
BC	07	IS	1
DIV	08	IS	1
READ	09	IS	1
PRINT	10	IS	1

Mnemonic Opcode Table(MOT)…

Mnemonic opcode	m/c code for opcode	Class	Size of instructions
START	01	AD	-----
END	02	AD	-------
ORIGIN	03	AD	-----
EQU	04	AD	-------
LTROG	05	AD	-----
DS	01	DL	-------
DC	02	DL	1
AREG	01	RG	-----
BREG	02	RG	-------
CREG	03	RG	-----

Enhanced Machine Opcode Table

Mnemonic opcode	Class	Opcode	Length
LT	CC	02	–
GT	CC	03	–
LE	CC	04	–
GE	CC	05	–
NE	CC	06	–
ANY	CC	07	–

Symbol Table

- It contains:

1. Name of variable or a label or symbol
2. Its address
3. Its size in number of words
4. Example-

Index	Symbol	address
	SYMBOL TABLE	
0	X	214
1	L1	202
2	NXT	207
3	BACK	202

rogramming

Literal Table

- It contains:

1. Value of the literal
2. Address of the memory location associated with the literal.

Example-

Index	LITERAL TABLE	
	Literal	address
0	5	205
1	2	206
2	1	210
3	2	211
4	4	215

ramming

POOL Table

- This table contains the literal number of the **starting literal** of each literal pool.

Index	POOL TABLE
0	0
1	2
2	4

Intermediate Code

- Is equivalent representation of source program.
- Pass-I of the assembler involve **scanning of the source file.**
- Every opcode is searched in MOT
- Every operand is searched in symbol table.

- It helps in avoiding:

1. Scanning of source file in PASS-II
2. Searching MOT and ST in PASS-II

Format of Intermediate Code

- Each Mnemonic opcode field is represented as:

(**Statement class** , **Machine code**)

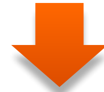
IS
AD
DL

MOT entry of opcode

Ex.: MOVER (IS,04)
LTORG (AD, 05)
START(AD,01)

Operand

- (**Operand Class,** **reference**)



C: constant
S: symbol
L: literal(variable)
RG: register
CC:condition code

for a symbol or literal , reference field contains the index of the operand's entry in the symbol table or literal table

Steps for Two Pass Assembler

Two Pass assembler: Pass1 and Pass2

Steps for Pass1:

1.Read source program

2.Add Location Count

3.Prepared Symbol table,Literal Table,Pool Table

4.Prepared Intermediate code using MOT table,Symbol table and Literal Table

Steps for Pass2:

1.It generate Machine code from Intermediate code

SNJB

Example 1

Pass 1 Assembler

```
        START 200
        MOVER AREG, ='5'
        MOVEM AREG, X
L1      MOVER BREG, ='2'
        ORIGIN L1+3
        LTORG

NEXT    ADD AREG, ='1'
        SUB BREG, ='2'
        BC LT, BACK
        LTORG

        BACK EQU L1
        ORIGIN NEXT+5
        MULT CREG, ='4'
        STOP
        X DS 1
        END
```

Observe code

Apply LC

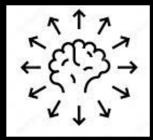
```
START 200
        MOVER AREG, ='5'        200
        MOVEM AREG, X           201
L1      MOVER BREG, ='2'        202
        ORIGIN L1+3
        LTORG
                  ='5'          205
                  ='2'          206

NEXT    ADD AREG, ='1'          207
        SUB BREG, ='2'          208
        BC LT, BACK             209
        LTORG
                  ='1'          210
                  ='2'          211

BACK    EQU   L1
        ORIGIN NEXT+5
        MULT CREG, ='4'         212
        STOP                    213
        X DS 1                  214
        END
                  ='4'          215
```

Construct

Symbol Table

index	Symbol Name	Address
0	X	214
1	L1	202
2	NEXT	207
3	BACK	202

Construct



Literal Table

index	Literal	Address
0	5	205
1	2	206
2	1	210
3	2	211
4	4	215

Pool Table

Pool table contains starting literal(index) of each pool.

Literal number
0
2
4

Intermediate code

Label	Statement	Address
START 200		
	MOVER AREG, ='5'	200
	MOVEM AREG, X	201
L1	MOVER BREG, ='2'	202
	ORIGIN L1+3	
	LTORG	
	='5'	205
	='2'	206
NEXT	ADD AREG, ='1'	207
	SUB BREG, ='2'	208
	BC LT, BACK	209
	LTORG	
	='1'	210
	='2'	211
BACK	EQU L1	
	ORIGIN NEXT+5	
	MULT CREG, ='4'	212
	STOP	213
	X DS 1	214
	END	
	='4'	215

(AD, 01) (C, 200)
200 (IS, 04) (RG,01) (L, 0)
201 (IS, 05) (RG,01) (S,0)
202 (IS, 04) (RG,02) (L,1)
203 (AD, 03) (C, 205)
205 (DL, 02) (C,5)
206 (DL, 02) (C, 2)
207 (IS,01) (RG, 01) (L, 2)
208 (IS, 02) (RG, 02) (L,3)
209 (IS, 07) (CC, 02) (S, 3)
210 (DL,02) (C,1)
211 (DL,02) (C,2)
212 (AD, 04) (C, 202)
212 (AD, 03) (C, 212)
212 (IS, 03) (RG, 03)(L, 4)
213 (IS, 00)
214 (DL, 01, C, 1)
215 (AD, 02)
215 (DL, 02) (C,4)

SNJB

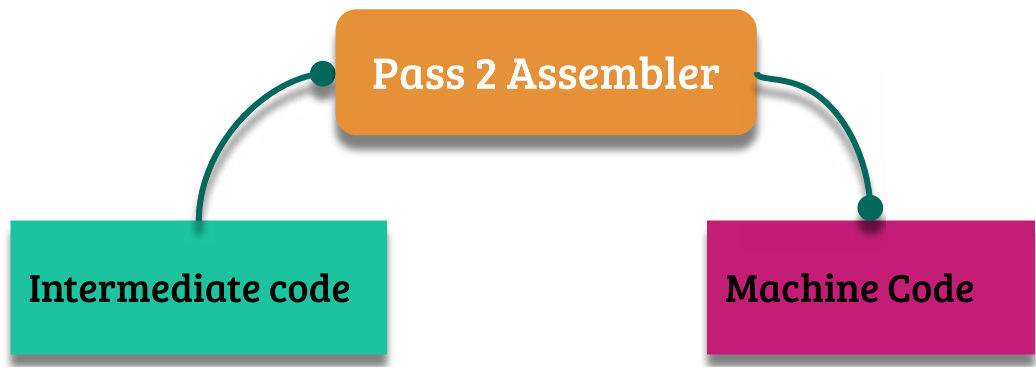
Example 2

Assignment

```
        START 205
        MOVER AREG, ='6'
        MOVEM AREG, A
LOOP    MOVER AREG, A
        MOVER CREG, B
        ADD CREG, ='2'
        BC ANY , NEXT
        LTORG
        ADD BREG, B
NEXT    SUB AREG, ='1'
        BC LT, BACK
LAST    STOP
        ORIGIN  LOOP+2
        MULT CREG, B
        ORIGIN LAST+1
A       DS      1
BACK    EQU     LOOP
B       DS      1
END
```

Pass 2 Assembler

- Processes the intermediate representation (IR) to synthesize the target program.



Intermediate code

```
START 200
            MOVER AREG, ='5'        200
            MOVEM AREG, X           201
    L1      MOVER BREG, ='2'        202
            ORIGIN L1+3
            LTORG
                    ='5'            205
                    ='2'            206

  NEXT      ADD AREG, ='1'          207
            SUB BREG, ='2'          208
            BC LT, BACK             209
            LTORG
                    ='1'            210
                    ='2'            211

BACK        EQU     L1
            ORIGIN NEXT+5
            MULT CREG, ='4'         212
            STOP                    213
            X DS 1                  214
            END
                    ='4'            215
```

```
(AD, 01) (C, 200)
200 (IS, 04) (RG,01) (L, 0)
201 (IS, 05) (RG,01) (S,0)
202 (IS, 04) (RG,02) (L,1)
203 (AD, 03) (C, 205)
205 (DL, 02) (C,5)
206 (DL, 02) (C, 2)
207 (IS,01) (RG, 01) (L, 2)
208 (IS, 02) (RG, 02) (L,3)
209 (IS, 07) (CC, 02) (S, 3)
210 (DL,02) (C,1)
211 (DL,02) (C,2)
212 (AD, 04) (C, 202)
212 (AD, 03) (C, 212)
212 (IS, 03) (RG, 03)(L, 4)
213 (IS, 00)
214 (DL, 01, C, 1)
215 (AD, 02)
215 (DL, 02) (C,4)
```

Dr. Mahesh R. Sanghavi

Example 2

Assignment


```
        START 205
        MOVER AREG, ='6'
        MOVEM AREG, A
LOOP    MOVER AREG, A
        MOVER CREG, B
        ADD CREG, ='2'
        BC ANY , NEXT
        LTORG
        ADD BREG, B
NEXT    SUB AREG, ='1'
        BC LT, BACK
LAST    STOP
        ORIGIN  LOOP+2
        MULT CREG, B
        ORIGIN LAST+1
A       DS      1
BACK    EQU     LOOP
B       DS      1
END
```

```
      START    200
      MOVER AREG, '=5'............    200
      MOVEM AREG, X    ............    201
L1 MOVER BREG, '=2'............       202
      ORIGIN L1+3          -------->  [202+3=205]
       LTORG                 ............    205,206  for literal
NEXT      ADD AREG, '= 1'     ............    207
      SUB BREG,'=2'          ............    208
      BC LT, BACK            .............   209
      LTORG                        210, 211.... for literal
BACK  EQU  L1                ............ 202
         ORIGIN NEXT+5       -------->  [207+5=212]
      MULT AREG,'=4'          ............212
      STOP...............................213
X   DS  '5'.............................. 214
      END                 ...........219 for literal     (214+5=219)
```

Index	SYMBOL TABLE	
	Symbol	address
0	X	214
1	L1	202
2	NEXT	207
3	BACK	202

Index	POOL TABLE
0	0
1	2
2	4

Index	LITERAL TABLE	
	Literal	address
0	5	205
1	2	206
2	1	210
3	2	211
4	4	219

Assembly Program	LC	Intermediate Code

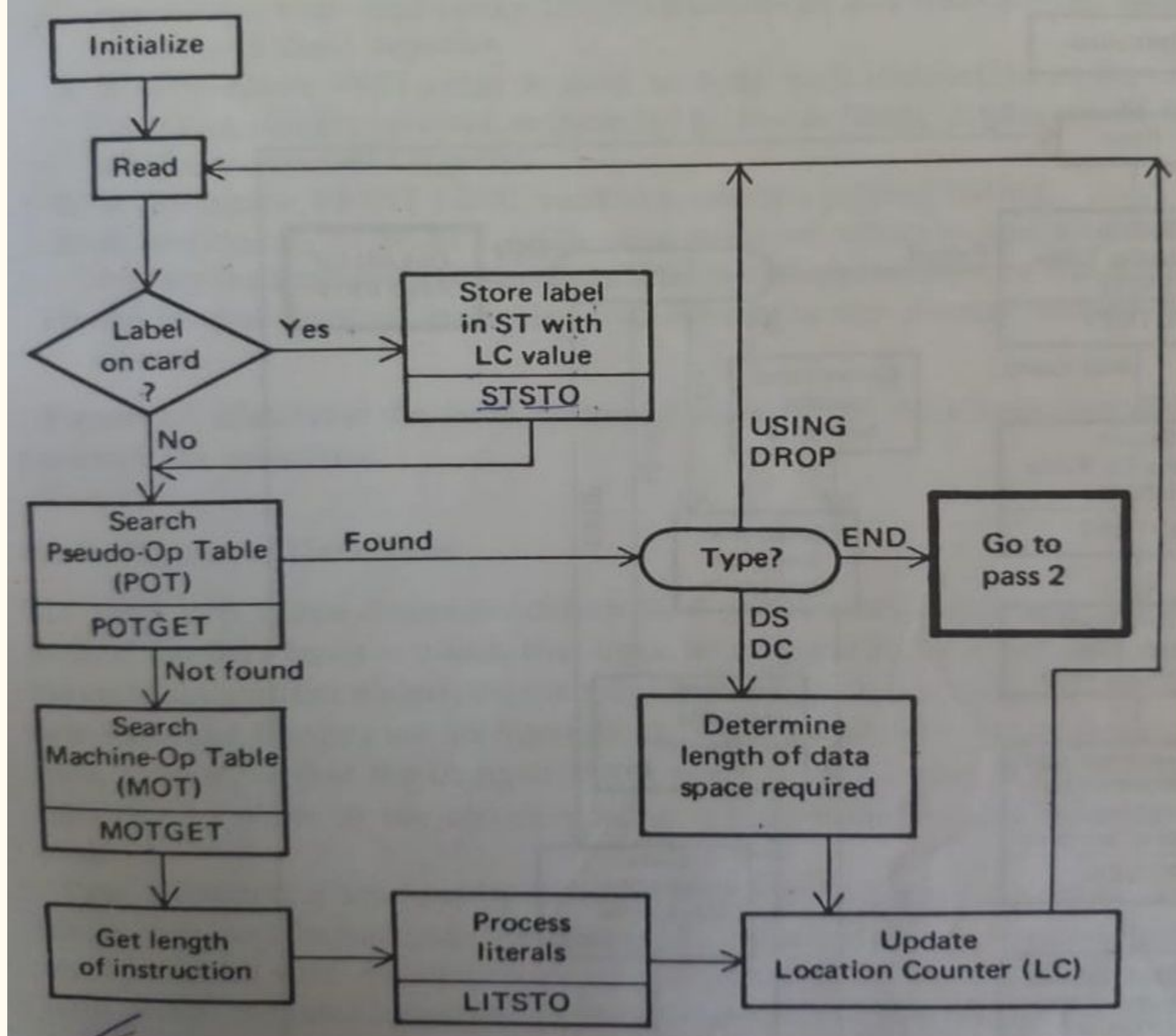
```
        START    200             ...........(AD,01)    (C,200)
        MOVER AREG, '=5'.....    200 ....(IS,04)    (RG,01)    (L,0)
        MOVEM AREG, X......      201....(IS,05)     (RG,01)    (S,0)
  L1    MOVER BREG, '=2'.....    202 ....(IS,04)    (RG,02)    (L,1)
        ORIGIN L1+3                 (AD,03)    (C,205)
         LTORG           205......(DL,02)            (C,5)
                         206 ......(DL,02)    (C,2)
NEXT       ADD AREG, '= 1'.......    207 ....(IS,01)    (RG,01)    (L,2)
        SUB BREG,'=2'  .........  208 ....(IS,02)    (RG,02)    (L,3)
        BC LT, BACK.............  209 ....(IS,07)    (CC,02)    (S,3)
        LTORG     ...........           210, ......(DL,02) (C,1)
                         211 ......(DL,02)           (C,2)
        BACK EQU L1 ...........   202 ...........(AD,01)    (C,202)
        ORIGIN NEXT+5             ...........(AD,01)    (C,212)
        MULT AREG,'=4'........    212 ....(IS,03)    (RG,03)    (L,4)
        STOP......................  213 ....(IS,00)
X       DS   '5'................ 214 ......(DL,01)            (C,1)
        END                       -------(AD,02)
                         219 .........(DL,02) (C,4)
```

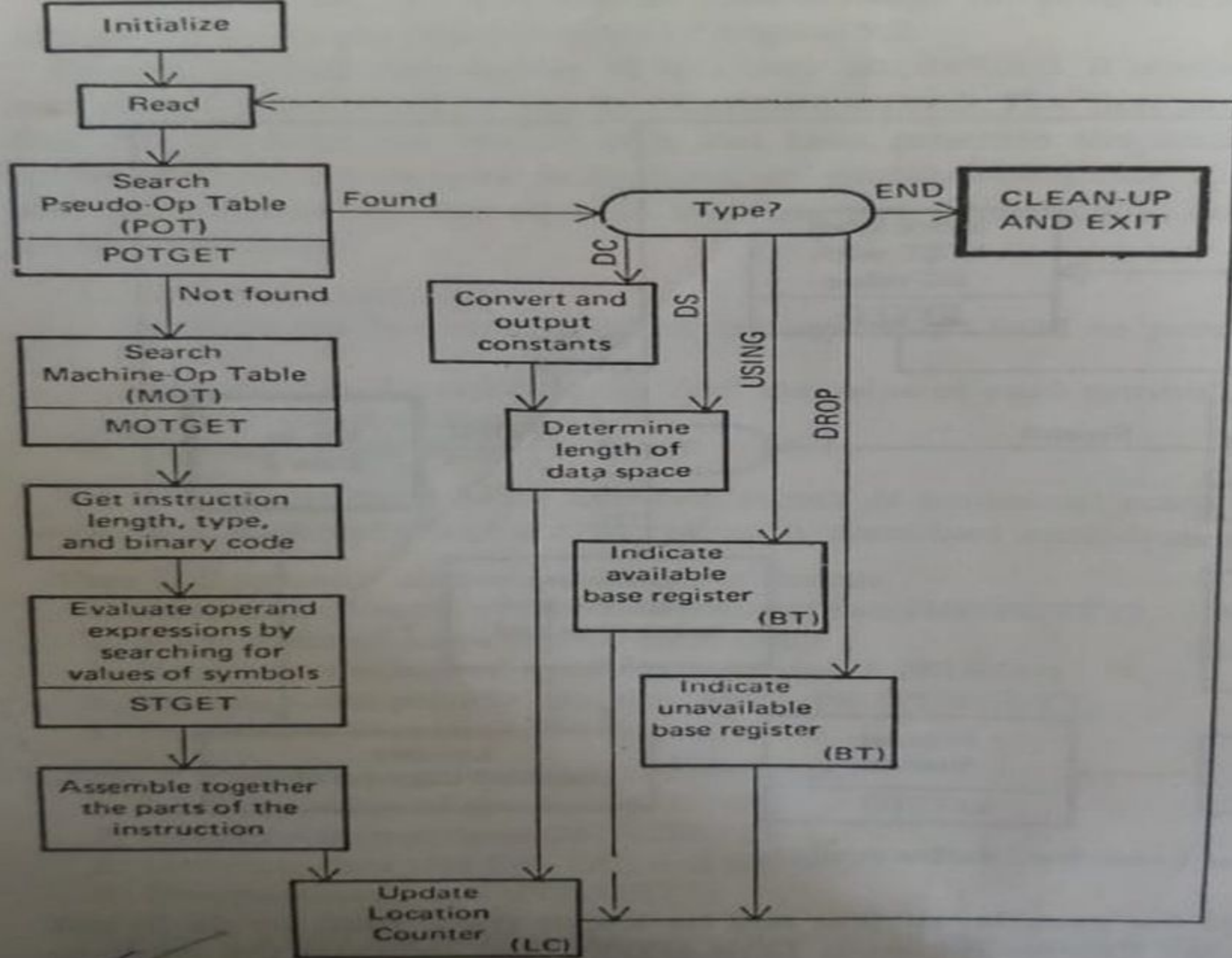
Assembly Program **LC** **Intermediate Code**

```
       START    200                      machine code
       MOVER AREG, '=5'…..          200 …   04    01    205
       MOVEM AREG, X……201….        05    01   214
L1 MOVER BREG, '=2'….. 202 ….04     02   206
       ORIGIN L1+3
       LTORG               205……00  00 05
                           206 ……00  00  02
NEXT      ADD AREG, '= 1'……. 207 ….01     01     210
       SUB BREG,'=2'  ………         208 ….(IS,02)    (RG,02)   (L,3)
       BC LT, BACK………….     209 ….(IS,07)    (CC,02)   (S,3)
       LTORG       ………..                 210, ……(DL,02) (C,1)
                           211 ……(DL,02)      (C,2)
       BACK EQU L1    ………..       202 ………..(AD,01)    (C,202)
       ORIGIN NEXT+5          ………..(AD,01)    (C,212)
       MULT AREG,'=4'……..        212 ….(IS,03)    (RG,03)   (L,4)
       STOP……………………         213 ….(IS,00)
X      DS    '5'…………….      214 ……(DL,01)       (C,1)
       END                         -------(AD,02)
                           219………(DL,02) (C,4)
```

Solved Example of Assembler

View


Initialize
Read
Label on card ?
Yes
Store label in ST with LC value
STSTO
No
Search Pseudo-Op Table (POT)
POTGET
Found
Type?
USING
DROP
END
Go to pass 2
DS
DC
Not found
Search Machine-Op Table (MOT)
MOTGET
Determine length of data space required
Get length of instruction
Process literals
LITSTO
Update Location Counter (LC)



Initialize
Read
Search
Pseudo-Op Table
(POT)
POTGET
Found
Type?
END
CLEAN-UP
AND EXIT
Not found
DC
Convert and
output
constants
DS
USING
DROP
Search
Machine-Op Table
(MOT)
MOTGET
Determine
length of
data space
Get instruction
length, type,
and binary code
Indicate
available
base register
(BT)
Evaluate operand
expressions by
searching for
values of symbols
STGET
Indicate
unavailable
base register
(BT)
Assemble together
the parts of the
instruction
Update
Location
Counter
(LC)

Thank You