LECTURE – 1

SYSTEM PROGRAMMING & SYSTEM ADMINISTRATION

SECTION -A

INTRODUCTION

- What is System Programming?
- Components of a System Programming
- Translational Hierarchy

WHAT IS SYSTEM PROGRAMMING ?

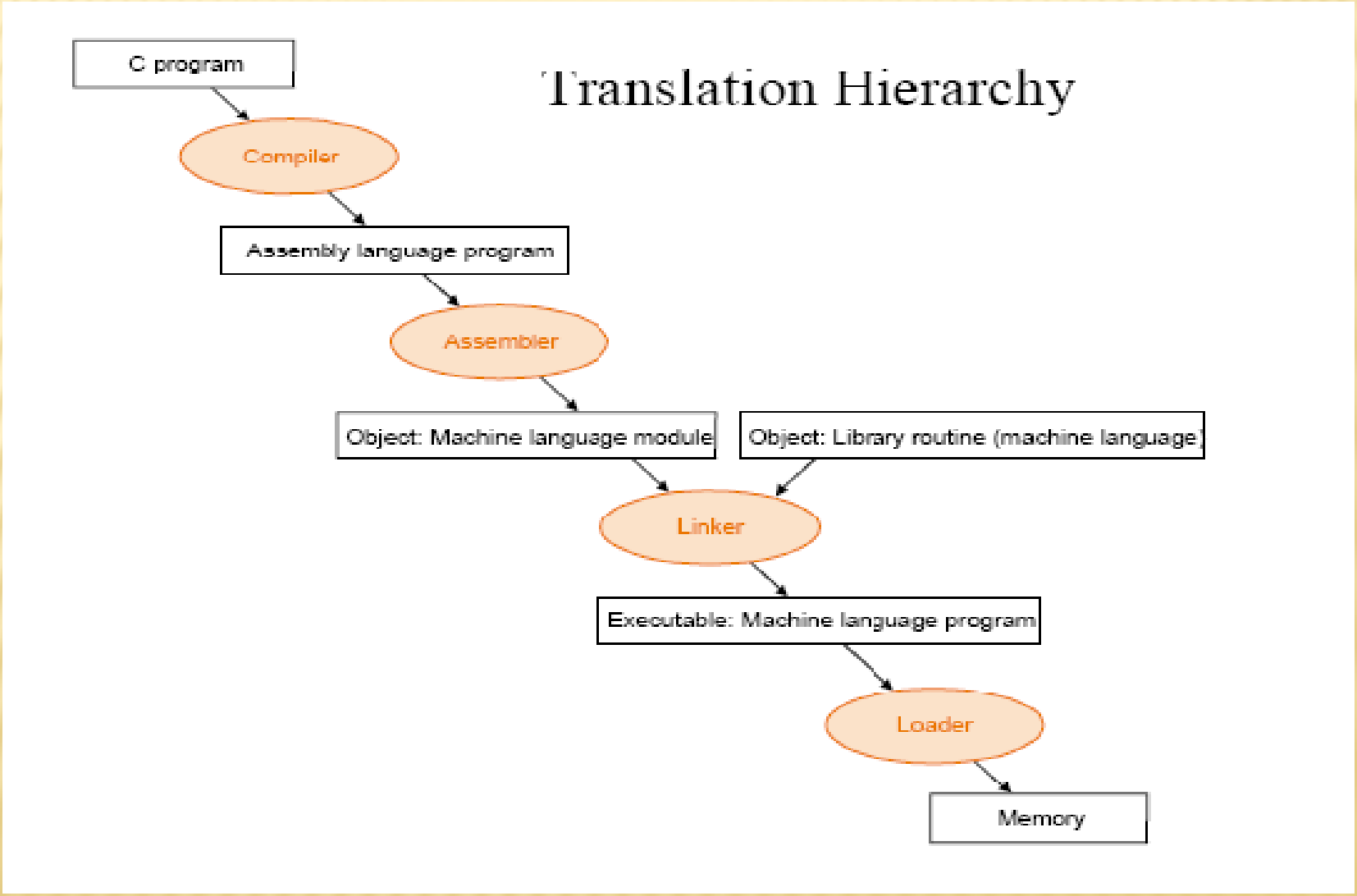
- **System programming** is the activity of programming system software.
- **Difference b/t System Programming & Application Programming :**
- Application programming aims to produce software which provides services to the user (e.g. word processor),
- whereas Systems programming aims to produce software which provides services to the computer hardware.

EVOLUTION OF THE COMPONENTS OF A PROGRAMMING SYSTEM

Components of Programming system are:-

- Assemblers
- Loaders
- Macros
- Compilers
- Linkers

Translation Hierarchy followed by Compiler, Assembler, Linker & Loader



LOADERS

- Once the assembler produces an object program, that program must be placed into memory and executed.
- It is the purpose of the loader to assure that object programs are placed in memory in an executable form.
- The assembler could place the object program directly in memory and transfer control to it, thereby causing the machine language program to be executed.
- However this would waste memory by leaving the assembler in memory while the user's program was being executed.
- Also the programmer would have to retranslate his program with each execution, thus wasting translation time.
- To overcome the problem of wasted translation time and wasted memory, system programmers developed another component, called the Loader.

Basic definition (Loaders)

- Loader is a program that places programs into memory and prepares them for execution.
- In a simple loading scheme, the assembler outputs the machine language translation of a program on a secondary storage device and a loader is placed in memory
- The loader places into memory the machine language version of the user's program & transfers control to it.
- Since the loader program is much smaller then the assembler, this makes more memory available to the user's program.

Machine Languages

- **Machine languages (first-generation languages) are the most basic type of computer languages, consisting of strings of numbers the computer's hardware can use.**

- **Different types of hardware use different machine code. For example, IBM computers use different machine language than Apple computers.**

Assembly Languages

- **Assembly languages (second-generation languages) are only somewhat easier to work with than machine languages.**
- **To create programs in assembly language, developers use cryptic English-like phrases to represent strings of numbers.**
- **The code is then translated into object code, using a translator called an assembler.**

```
;CLEAR SCREEN USING BIOS
CLR: MOV AX,0600H        ;SCROLL SCREEN
     MOV BH,30           ;COLOUR
     MOV CX,0000         ;FROM
     MOV DX,184FH        ;TO 24,79
     INT 10H             ;CALL BIOS;
;INPUTTING OF A STRING
KEY: MOV AH,0AH          ;INPUT REQUEST
     LEA DX,BUFFER       ;POINT TO BUFFER WHERE STRING STORED
     INT 21H             ;CALL DOS
     RET                 ;RETURN FROM SUBROUTINE TO MAIN PROGRAM;
; DISPLAY STRING TO SCREEN
SCR: MOV AH,09           ;DISPLAY REQUEST
     LEA DX,STRING       ;POINT TO STRING
     INT 21H             ;CALL DOS
     RET                 ;RETURN FROM THIS SUBROUTINE;
```

Assembly code

Assembler

```
0001010010110101010101010101010100010
1110110101010101010101110010100010110
0010100101010010111101011101011101010
1001010010110101010101010101010110110
0110100100110010111101011101010100010
0001000101011101010101000101010111010
1010100101010010101101011101011101011
0001010010110101010101010101010100010
```

Object code

ASSEMBLERS

- At one time, the computer programmer had a basic machine that interpret through hardware certain fundamental instructions.
- He would program this computer by writing a series of ones and zeros (machine language), place them into the memory of the machine, and press a button, whereupon the computer would start to interpret them as instructions.
- Programmers find it difficult to write or read programs in machine language, In their hunt for a more convenient language they began to use a mnemonic (symbol) for each machine instruction, which they could subsequently translate into machine language.
- Such a mnemonic machine language is now called an **assembly language**.
- Programs known as **assemblers** were written to automate/computerize the translation of assembly language into machine language. The input to an assembler program is called the **source program**; the output is a machine language translation object **program**).

TYPICAL APPLICATIONS

- Assembly language is typically used in a system's boot code, (BIOS on IBM-compatible PC systems and CP/M), the low-level code that initializes and tests the system hardware prior to booting the OS, and is often stored in ROM.
- Some compilers translate high-level languages into assembly first before fully compiling, allowing the assembly code to be viewed for debugging and optimization purposes.
- Relatively low-level languages, such as C, allow the programmer to embed assembly language directly in the source code. Programs using such facilities, such as the Linux kernel, can then construct abstractions using different assembly language on each hardware platform. The system's portable code can then use these processor-specific components through a uniform interface.

SCOPE OF RESEARCH

Demand

- The realization that many users were writing the same programs led to the development of “ready-made” programs (packages).
- These packages were written by computer manufacturers or users.
- As the programmer become more sophisticated, he wanted to mix and combine ready-made programs with his own.

In response to this demand:

A facility need to be provided where by the user could write a main program that used several other programs or subroutines.