LECTURE-17

Linkers and Loaders

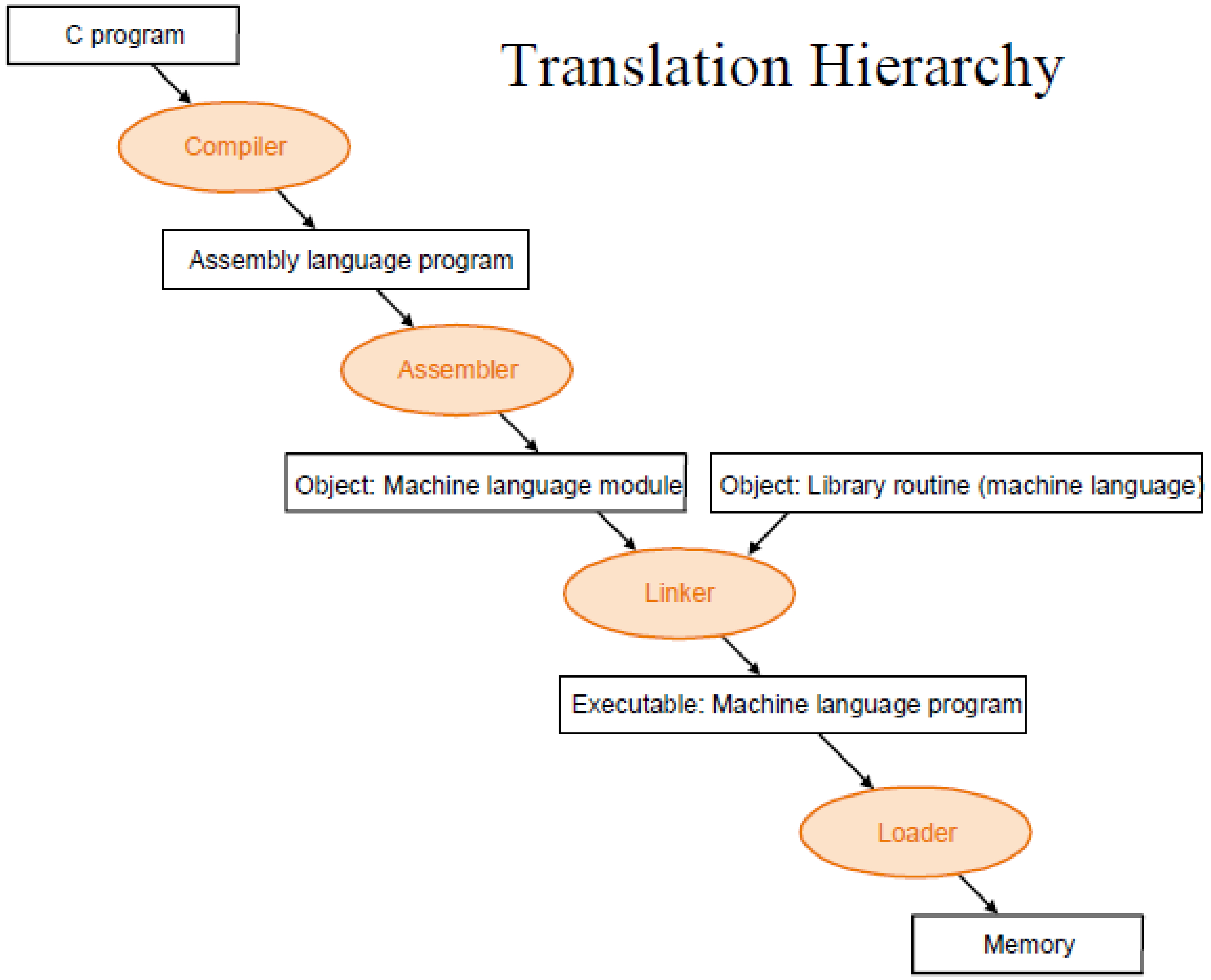
Translation Hierarchy

Compiler:
Translates high-level language program into assembly language.

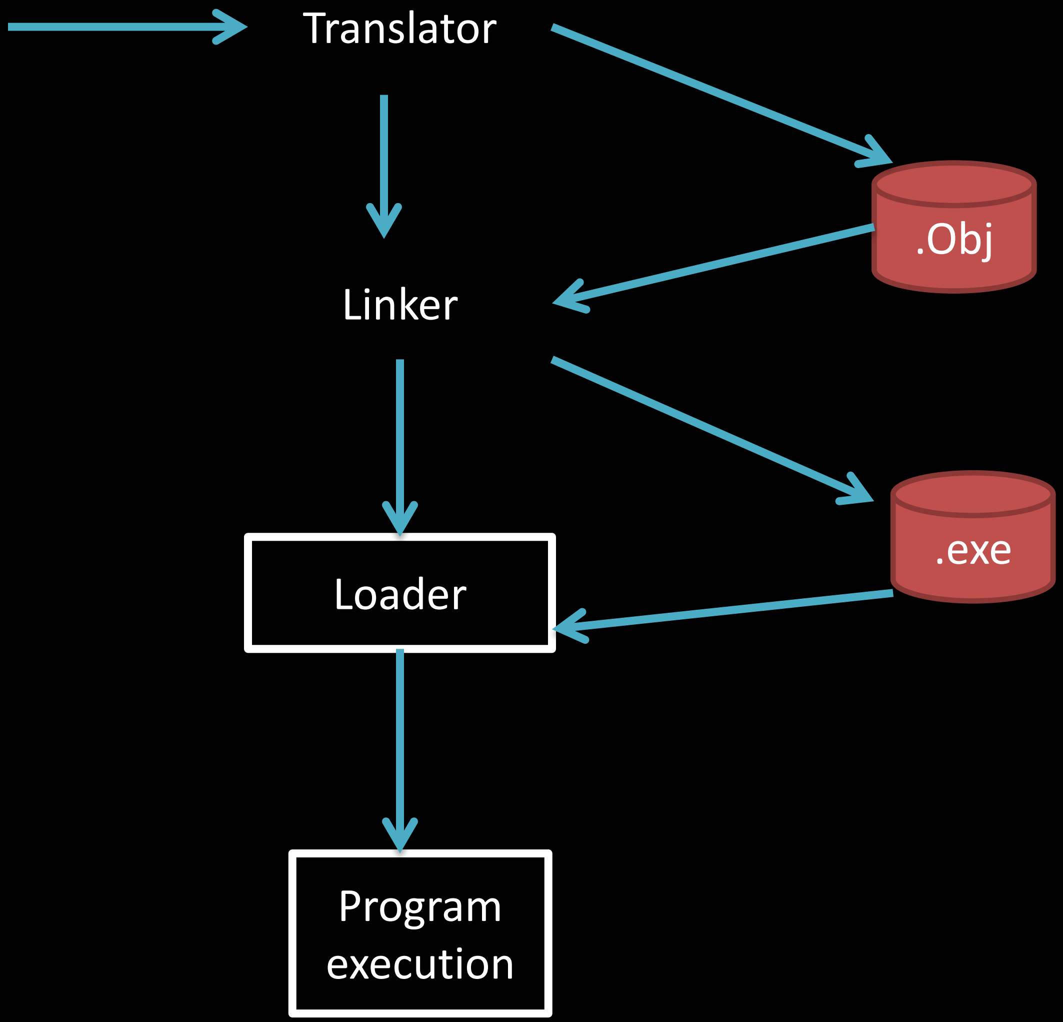
Assembler
Converts assembly language programs into *object* files.

Object files contain a combination of machine instructions, data, and information needed to place instructions properly in memory.

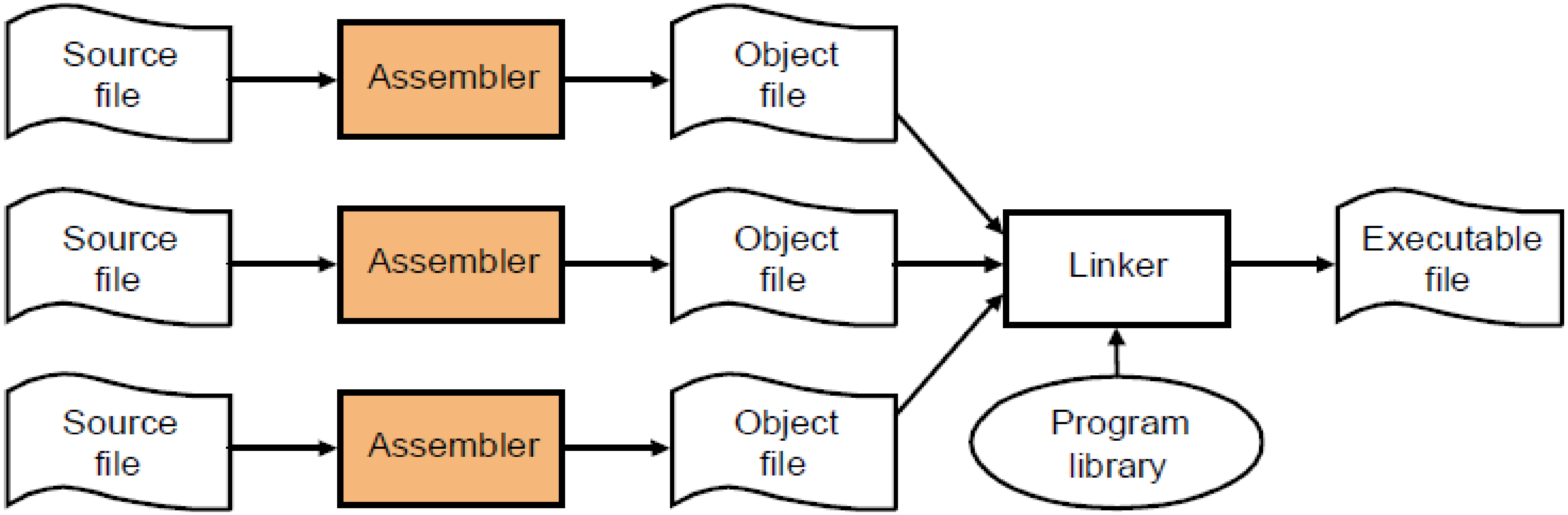
Translation Hierarchy


C program
Compiler
Assembly language program
Assembler
Object: Machine language module
Object: Library routine (machine language)
Linker
Executable: Machine language program
Loader
Memory

Flow of Execution



Process for producing an executable file



Linker

Tool that merges the object files produced by *separate compilation* or assembly and creates an executable file.

- Three tasks:-

❑ Searches the program to find library routines used by program, e.g. printf(),sqrt(),strcat() and various other.

❑ Determines the memory locations that code from each module will occupy and relocates its instructions by adjusting absolute references.

Relocation, which modifies the object program so that it can be loaded at an address different from the location originally specified.

❑ It combines two or more separate object programs and supplies the information needed to allow references between them .

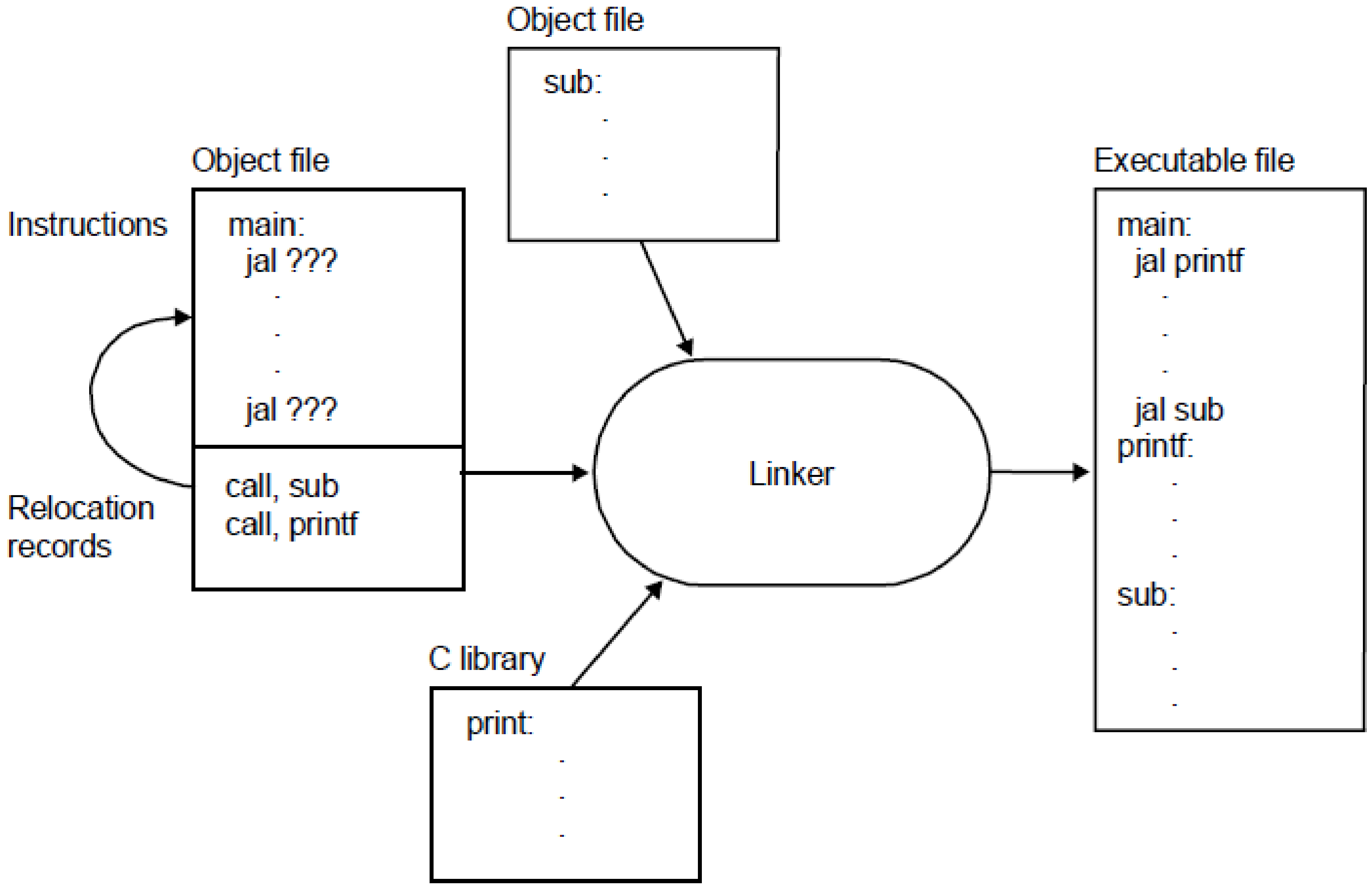
Linking Concepts

- Computer programs typically comprise several parts or modules; all these parts/modules need not be contained within a single **object file**, and in such case refer to each other by means of **symbols**. Typically, an object file can contain three kinds of symbols:

- ✓ ***Publicly defined symbols***, which allow it to be called by other modules , also called as **public definition** .

- ✓ ***Externally defined symbols(***undefined symbols), which calls the other modules where these symbols are defined, also called as **external reference.**

- ✓ ***Local symbols***, used internally within the object file to facilitate relocation.


Object file
sub:
.
.
.
Object file
Instructions
main:
jal ???
.
.
.
jal ???
Relocation records
call, sub
call, printf
Linker
C library
print:
.
.
.
Executable file
main:
jal printf
.
.
.
jal sub
printf:
.
.
.
sub:
.
.
.

Static Linking

- *Static linking* occurs when a calling program is linked to a called program in a single executable module. When the program is loaded, the operating system places into memory a single file that contains the executable code and data.

- The result of statically linking programs is an .EXE file or dynamic link library (DLL) subprogram that contains the executable code for multiple programs. This file includes both the calling program and the called program.

- The advantage of static linking is that you can create self-contained, independent programs. In other words, the executable program consists of one part (the .EXE file) that you need to keep track of.
- Disadvantages:

.You cannot change the behavior of executable files without relinking them.

- External called programs cannot be shared, requiring that duplicate copies of programs be loaded in memory if more than one calling program needs to access them.

Dynamic linking

- Many operating system environments allow dynamic linking, that is the postponing of the resolving of some undefined symbols until a program is run.
- ***That means that the executable code still contains undefined symbols, plus a list of objects or libraries that will provide definitions for these. Loading the program will load these objects/libraries as well, and perform a final linking.***

Advantages and Disadvantages

- ***Advantages:***
- Often-used libraries (for example the standard system libraries) need to be stored in only one location, not duplicated in every single binary.

- If an error in a library function is corrected by replacing the library, all programs using it dynamically will benefit from the correction after restarting them. Programs that included this function by static linking would have to be re-linked first.
- ***Disadvantages:***
- Known on the Windows platform as "DLL Hell", an incompatible updated DLL will break executables that depended on the behavior of the previous DLL.

- A program, together with the libraries it uses, might be certified (e.g. as to correctness, documentation requirements, or performance) as a package, but not if components can be replaced.

Loader

It is a ***SYSTEM PROGRAM*** that brings an executable file residing on disk into memory and starts it running.

- Steps:-

– Read executable file's header to determine the size of text and data segments.

– Create a new address space for the program.

– Copies instructions and data into address space.

– Copies arguments passed to the program on the stack.

– Initializes the machine registers including the stack pointer.

– Jumps to a startup routine that copies the program's arguments from the stack to registers and calls the program's main routine.

Types of Loaders

- ❑ Compile/Assemble and Go loader
- ❑ Absolute Loader
- ❑ Relocating Loader
- ❑ Direct Linking loader

Assemble-and-go Loader

- Compilation, assembly, and link steps are not separated from program execution all in single pass.
- The intermediate forms of the program are generally kept in RAM, and not saved to the file system.
- Compile and go systems differ from interpreters, which either directly execute source code or execute an intermediate representation.

Advantages

- The user need not be concerned with the separate steps of compilation, assembling, linking, loading, and executing.
- Execution speed is generally much superior to interpreted systems.
- They are simple and easier to implement.

Disadvantages

- There is wastage in memory space due to the presence of the assembler.
- The code must be reprocessed every time it is run.
- Systems with multiple modules, possibly in different languages, cannot be handled naturally within this framework.
- Compile-and-go systems were popular in academic environments, where student programs were small, compiled many times, usually executed quickly and, once debugged, seldom needed to be re-executed.

Absolute Loader

Absolute Program

- Advantage:
- Simple and efficient
- No linking or relocation

- Disadvantage:
- Difficult to use subroutine libraries.
- The need of programmer to state the actual address.

Algorithm for Absolute Loader

```
1.  begin
2.  read Header record
3.  verify program name and length
4.  read first Text record
5.  while record type <> ’E’ do
6.  begin
7.  {if object code is in character form, convert into
8.  internal representation}
9.  move object code to specified location in memory
10. read next object program record
11. end
12. jump to address specified in End record
13. end
```

Bootstrap Loader

- Special Type of Absolute Loader.
- When a computer is first tuned on or restarted bootstrap loader is executed.
- This bootstrap loads the first program to be run by computer that is the OS.
- It loads the first address 0x80.

Relocation

- Execution of the object program using any part of the available and sufficient memory.
- The object program is loaded into memory wherever there is room for it.
- The actual starting address of the object program is not known until load time.

Relocating Loader

- Efficient sharing of the machine with larger memory and when several independent programs are to be run together.

- Support the use of subroutine libraries efficiently.

Direct Linking Loader

- *This type of loader is a relocating loader.*
- The loader cannot have the direct access to the source code.
- To place the object code 2 types of addresses can be used:-

1. ABSOLUTE : In this the absolute path of object code is known and the code is directly loaded in memory.

2. RELATIVE :In this the relative path is known and this relative path is given by assembler.

Work Of Assembler in Direct Linking Loader

The assembler should give the following information to the loader:

- ***The length of the object code segment.***
- ***A list of external symbols (could be used by diff. segment).***
- ***List of External symbols(The segment itself is using)***