Software Systems

INTRODUCTION TO LOADER LECTURE01 2023

Dr. Azam E. Al-Rawachy
SOFTWARE DEPARTMENT | COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS

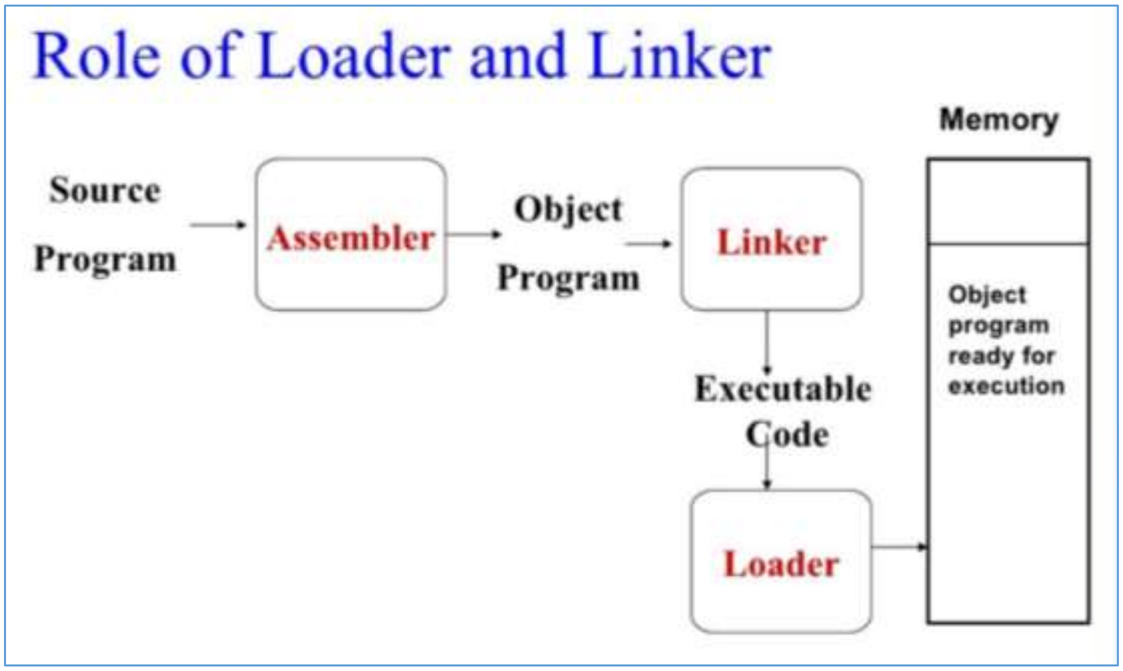
1 Introduction

The Source Program written in assembly language or high-level language will be converted to an object program, which is in the machine language form for execution. This conversion either from an assembler or from a compiler contains translated instructions and data values from the source program or specifies addresses in primary memory where these items are to be loaded for execution. This contains the following three processes, and they are

Loading - Which allocates memory location and brings the object program into memory for execution - (Loader)

Linking- Which combines two or more separate object programs and supplies the information needed to allow references between them - (Linker)

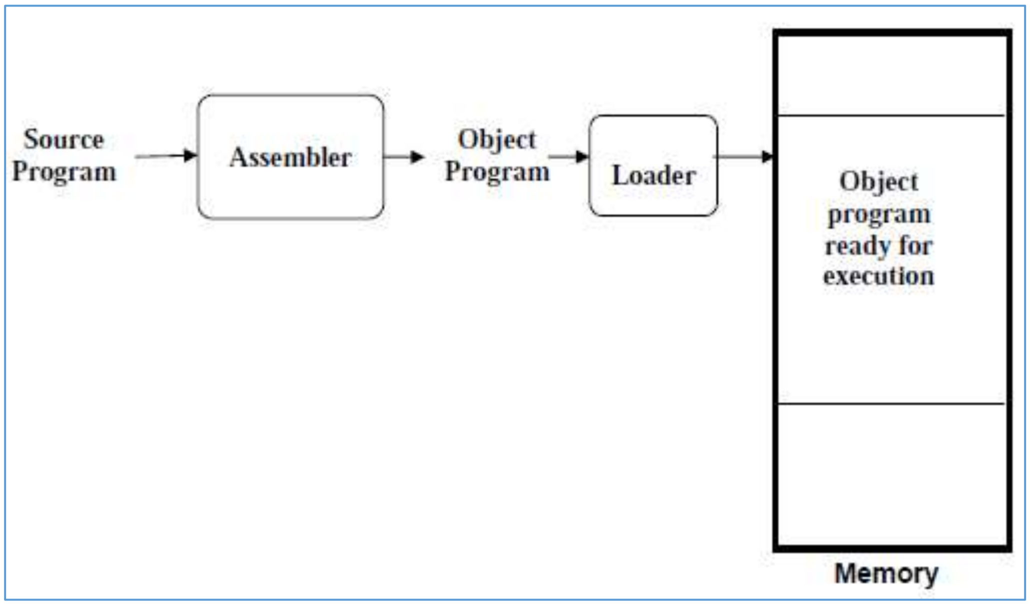
Relocation - Which modifies the object program so that it can be loaded at an address different from the location originally specified - (Linking Loader)



2 LOADER

Loader is the program that accomplished the loading task. Loading is the process of bringing a program into the main memory so that it can run. On most modern systems, each program is loaded into a fresh address space, which means that all programs are loaded at a known fixed address, and can be linked for that address. Loading is pretty simple from this point and requires the following steps:

- Read enough header information from the object file to find out how much address space is needed.
- Allocate that address space, in separate segments if the object format has separate segments.
- Read the program into the segments in the address space.
- Create a stack segment if the architecture needs one.
- Start the program.

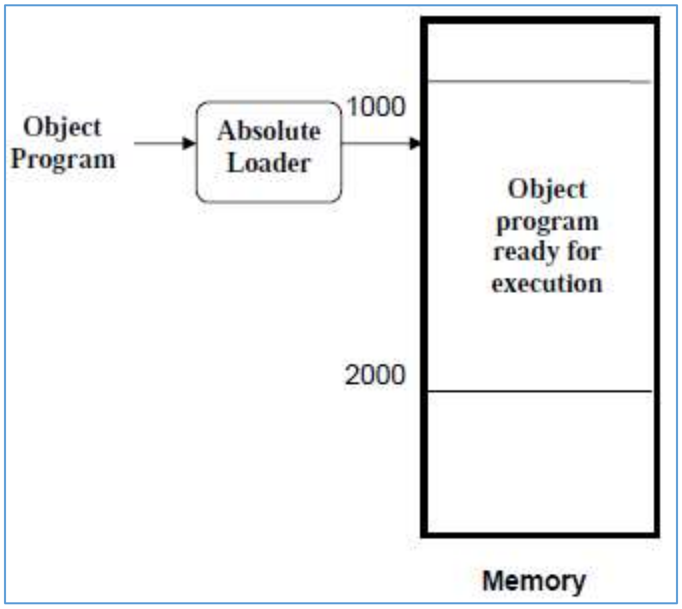


3 Type of Loaders

The different types of loaders are **absolute loader**, **relocating loader (relative loader)**, **bootstrap loader**, and, **direct linking loader**. The following sections discuss the functions of the first two types of loaders while the remaining types are out of the scope of this lecture.

3.1 Absolute Loader

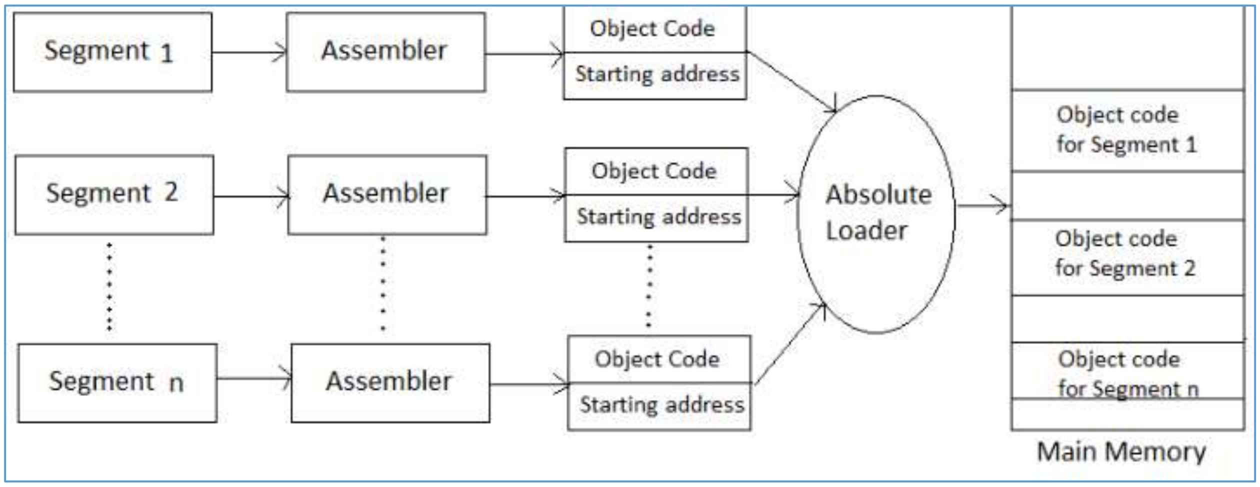
The operation of the absolute loader is very simple. The object code is loaded to specified locations in the memory. In the end, the loader jumps to the specified address to begin the execution of the loaded program. The role of the absolute loader is as shown in the figure below.



This type of loader is called absolute loader because no relocating information is needed, rather it is obtained from the programmer or assembler. The starting address of every module is known to the programmer, this corresponding starting address is stored in the object file then the task of the loader becomes very simple that is to simply place the executable form of the machine instructions at the locations mentioned in the object file.

The programmer should take care of two things:

- The specification of starting address of each module to be used. If some modification is done in the address of the immediate next modules, the programmer then must make necessary changes in the starting addresses of respective modules.
- The second thing is while branching from one segment to another the absolute starting address of the respective module is to be known by the programmer so that such address can be specified at respective JMP instruction.



3.2 Advantages and disadvantage of the Absolute Loader

Absolute loader is simple and efficient; however, its scheme has several disadvantages. One of the most obvious is the need for a programmer to specify the actual address at which it will be loaded into memory. Usually, we do not know exactly when jobs will be submitted, and how long they will run; We would often like to run several independent programs together, sharing memory and resources between them. This means that we do not know in advance where a program will be loaded. Multiprogramming and efficient sharing of the machine make it necessary to have a relocatable loader instead of an absolute one.

The algorithm for this type of loader is given here.

```
Begin
   Read Header record
   Verify program name and length
   Read first Text record
   While record type is! = 'E' do
         begin
         {   if object code is in character form, convert into internal representation}
             move object code to the specified location in memory
             read next object program record
         end
   Jump to address specified in End record
End
```

3.3 RELOCATING LOADERS (Relative Loader)

Absolute loader is simple and efficient, but the scheme has potential disadvantages One of the most disadvantages is the programmer has to specify the actual starting address, from where the program to be loaded. This does not create difficulty if one program to run, but not for several programs. Further, it is difficult to use subroutine libraries efficiently.

To avoid possible reassembling of all subroutines when a single subroutine is changed and to perform the tasks of allocation and linking for the programmer the relocating loaders is introduced. The execution of the object program is done using any part of the available & sufficient memory. The object program is loaded into memory wherever there is room for it. The assembler assembles each procedures segment independently. The assembler would also provide the loader with additional information, such as the length of the entire program.

RELOCATING
LOADERS

3.3.1 Methods for specifying relocation

Use of modification record and, use of relocation bit, are the methods available for specifying relocation. In the case of ***modification record***, a modification record M is used in the object program to specify any relocation.

Relocation Loader Algorithm

```
Begin
  Get PROGADDR from OS
  While not end of input do
      {  Read next input record
         While record type is! = 'E' do
           {
                 Read next input record

                 While record type = 'T' do
                   {
                         Move object code from record to location ADDR + Specified address
                   }

                 While record type = 'M' do
                         Add PROGADDR at location PROGADDR+ Specified address
           }
      }
End
```