LOADERS

Contents

INTRODUCTION

- The Source Program written in assembly language or high level language will be converted to object program, which is in the machine language form for execution.

- This conversion either from assembler or from 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.

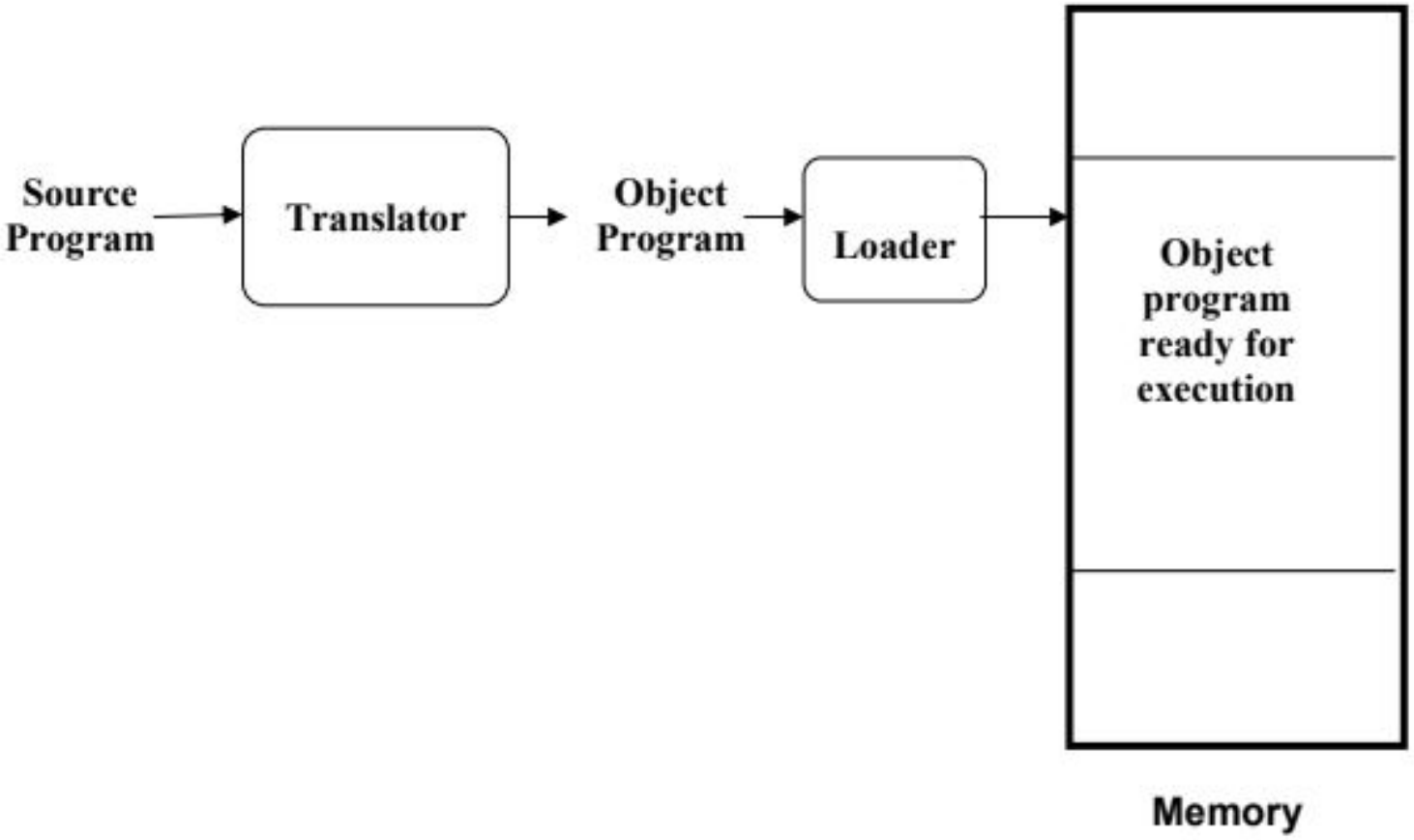


Figure 3.1 : The Role of Loader

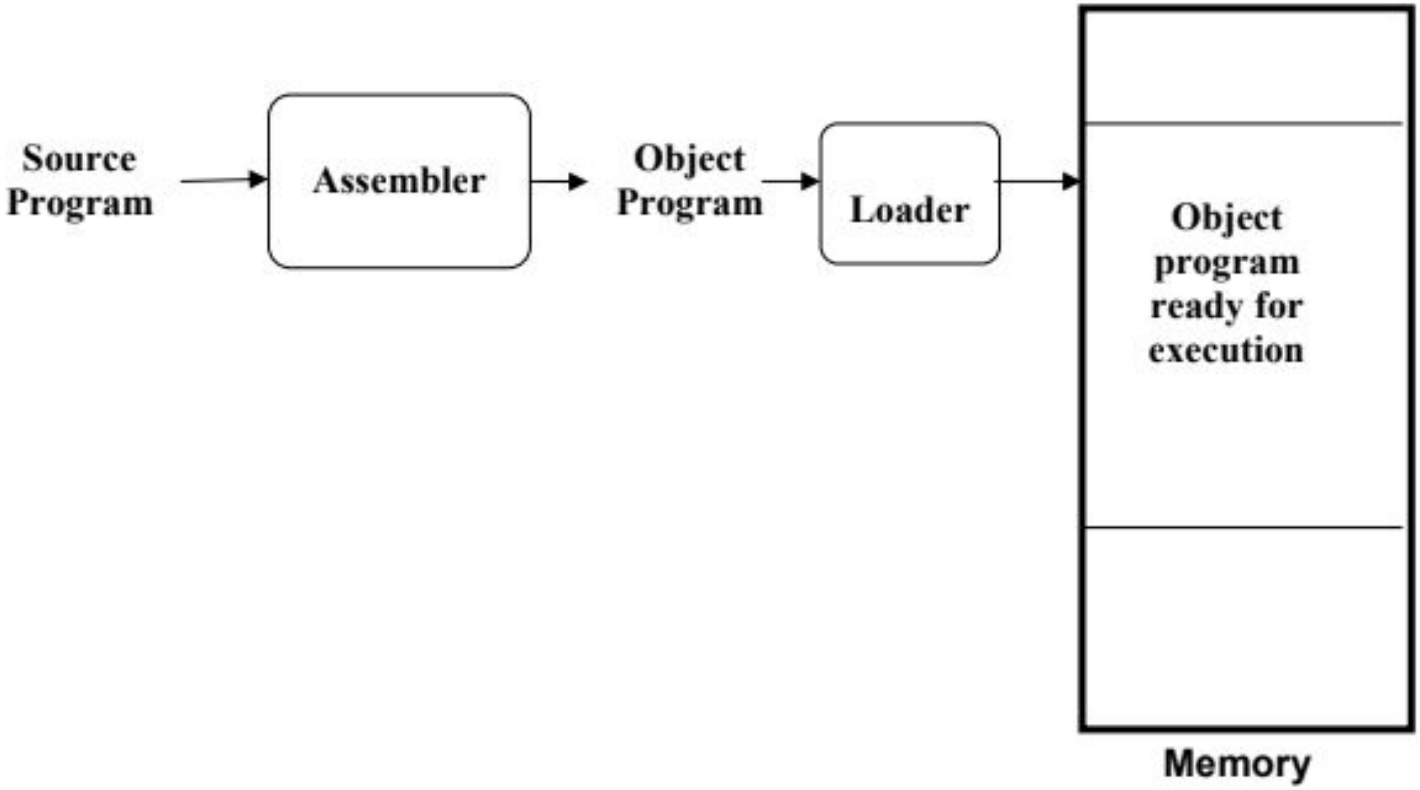


Figure 3.2: The Role of Loader with Assembler

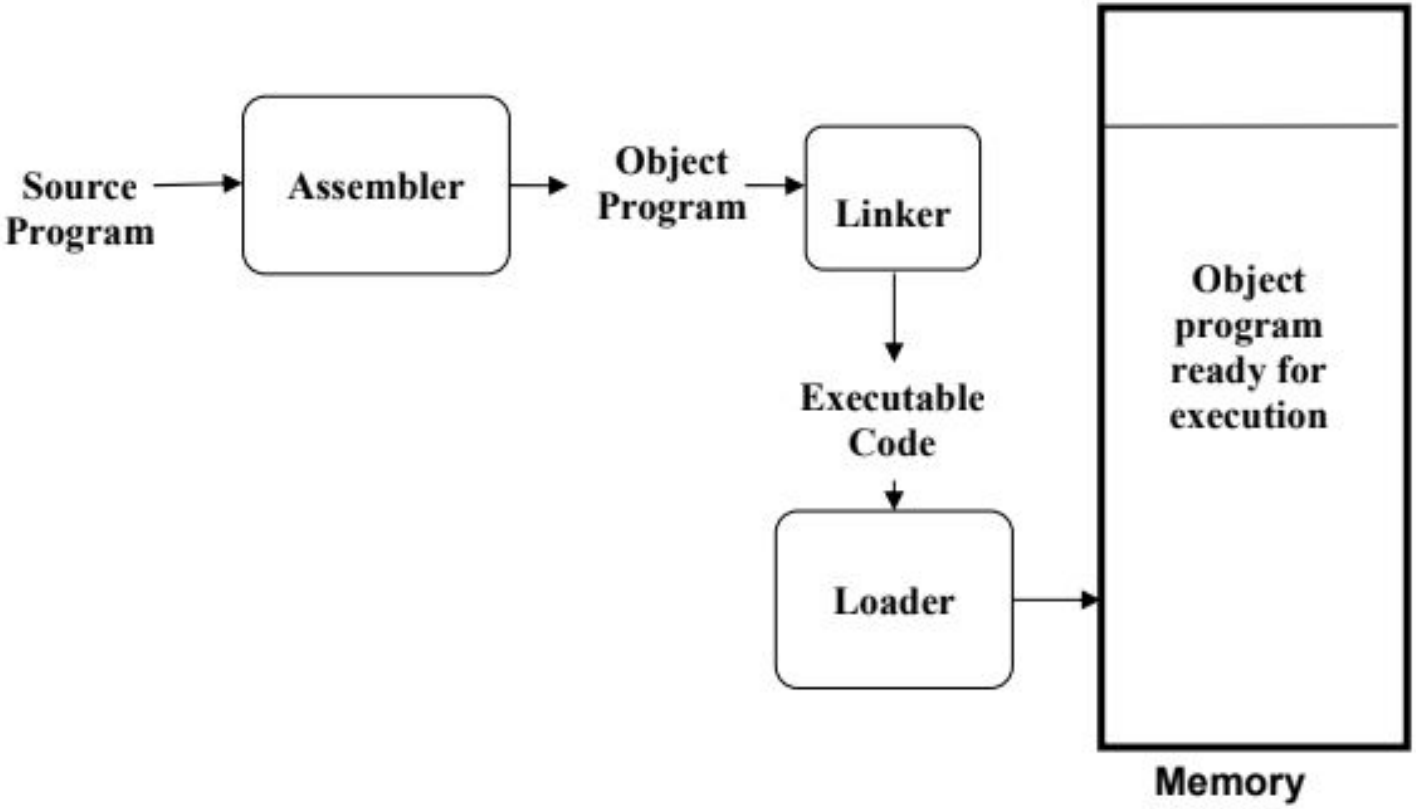


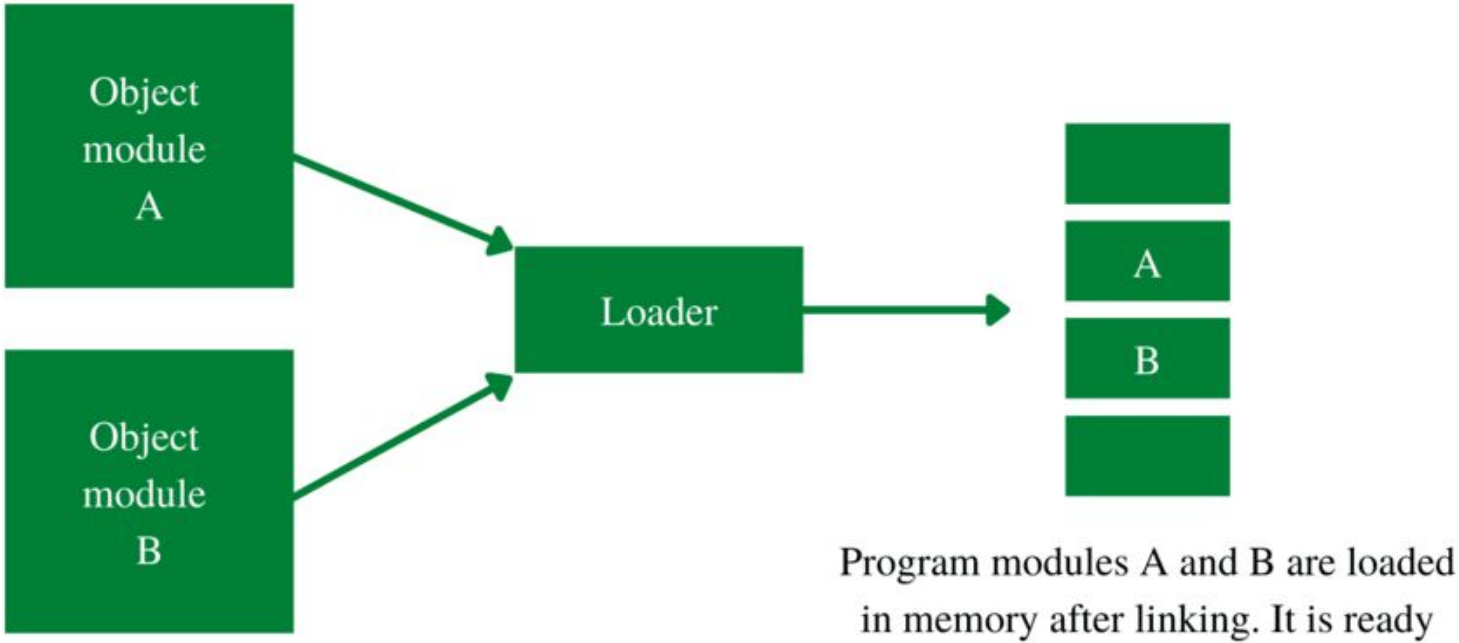
Figure 3.3 : The Role of both Loader and Linker

Loader & its funtions:

It brings object program into memory and starts its execution.

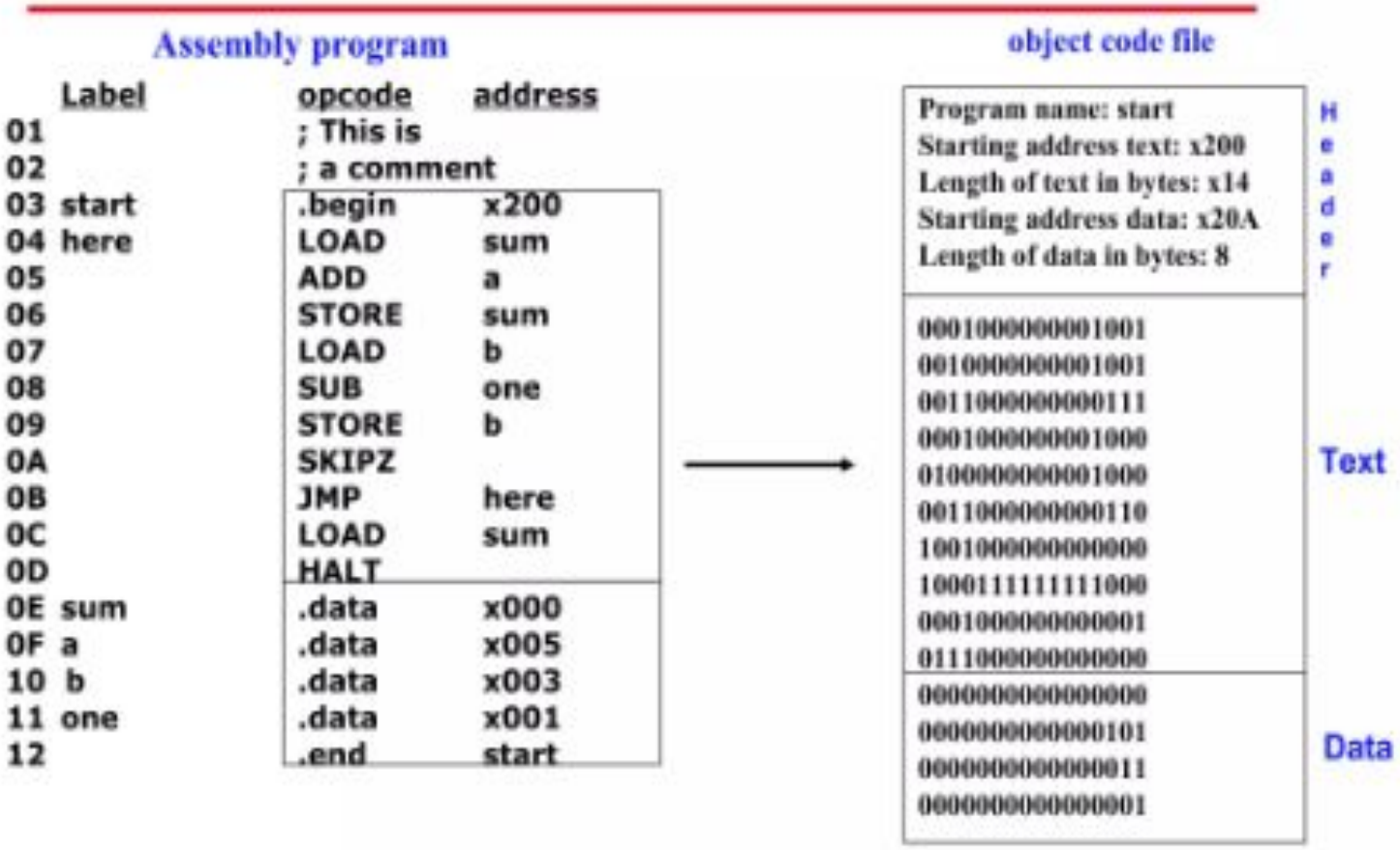
Loader performs its task via four functions, these are as follows:

1. **Allocation:** It allocates memory for the program in the main memory.
2. **Linking:** It combines two or more separate object programs or modules and supplies necessary information.
3. **Relocation:** It modifies the object program so that it can be loaded at an address different from the location.
4. **Loading:** It brings the object program into the main memory for execution.



Program modules A and B are loaded in memory after linking. It is ready for execution.

General loading scheme


Assembly program
object code file
Label opcode address
01 ; This is
02 ; a comment
03 start .begin x200
04 here LOAD sum
05 ADD a
06 STORE sum
07 LOAD b
08 SUB one
09 STORE b
0A SKIPZ
0B JMP here
0C LOAD sum
0D HALT
0E sum .data x000
0F a .data x005
10 b .data x003
11 one .data x001
12 .end start
Program name: start
Starting address text: x200
Length of text in bytes: x14
Starting address data: x20A
Length of data in bytes: 8
Header
0001000000001001
0010000000001001
0011000000000111
0001000000001000
0100000000001000
0011000000000110
1001000000000000
1000111111111000
0001000000000001
0111000000000000
Text
0000000000000000
0000000000000101
0000000000000011
0000000000000001
Data

1) Allocation

- In order to allocate memory to the program, the loader allocates the memory on the basis of the size of the program, this is known as allocation.

- The loader gives the space in memory where the object program will be loaded for execution.

2) Linking

The linker resolves the symbolic reference code or data between the object modules by allocating all of the user subroutine and library subroutine addresses. This process is known as **linking**

Eg. printf(), scanf()

3) Relocation

- There are some address-dependent locations in the program, and these address constants must be modified to fit the available space, this can be done by loader and this is known as **relocation**.
- In order to allow the object program to be loaded at a different address than the one initially supplied, the loader modifies the object program by modifying specific instructions.

4) Loading

- The loader loads the program into the main memory for execution of that program.
- It loads machine instruction and data of related programs and subroutines into the main memory, this process is known as **loading**. The loader performs loading; hence, the assembler must provide the loader with the object program.

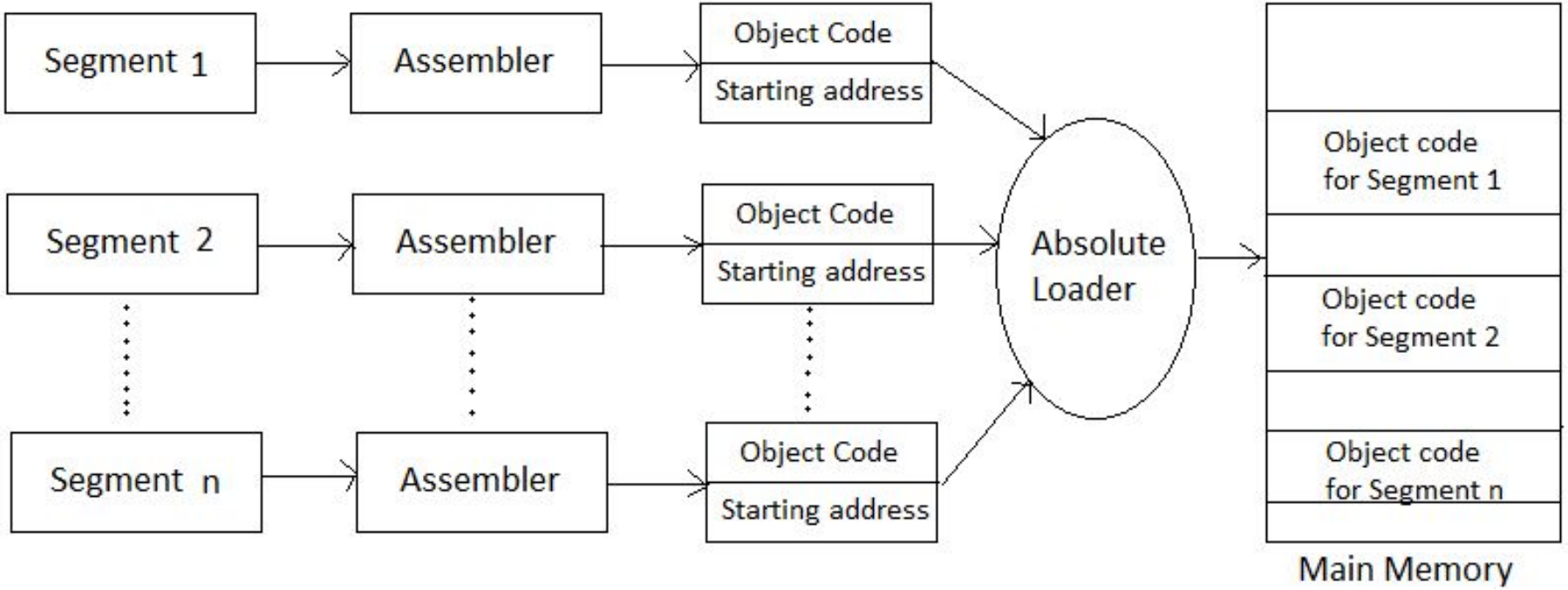
Types of Loaders

Types of Loaders

- Absolute loader,
- Compile and go Loader
- Bootstrap loader,
- Relocating loader (relative loader)
- Direct linking loader.

ABSOLUTE LOADER

- The absolute loader transfers the text of the program into memory at the address provided by the assembler after reading the object program line by line.
- Information that the object program must communicate from the assembler to the loader:
-
- **Machine instructions**
- **Start of execution**


Segment 1
Assembler
Object Code
Starting address
Segment 2
Assembler
Object Code
Starting address
Segment n
Assembler
Object Code
Starting address
Absolute Loader
Object code for Segment 1
Object code for Segment 2
Object code for Segment n
Main Memory

```
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 specified location in memory
        read next object program record
        end
jump to address specified in End record
end
```

Absolute Loader

Assembler object code file

Program name: start Starting address text: x200 Length of text in bytes: x14 Starting address data: x20A Length of data in bytes: 8	Header
0001000000001001 0010000000001001 0011000000000111 0001000000001000 0100000000001000 0011000000000110 1001000000000000 1000111111111000 0001000000000001 0111000000000000	Text section
0000000000000000 0000000000000101 0000000000000011 0000000000000001	Data section

Absolute loader:

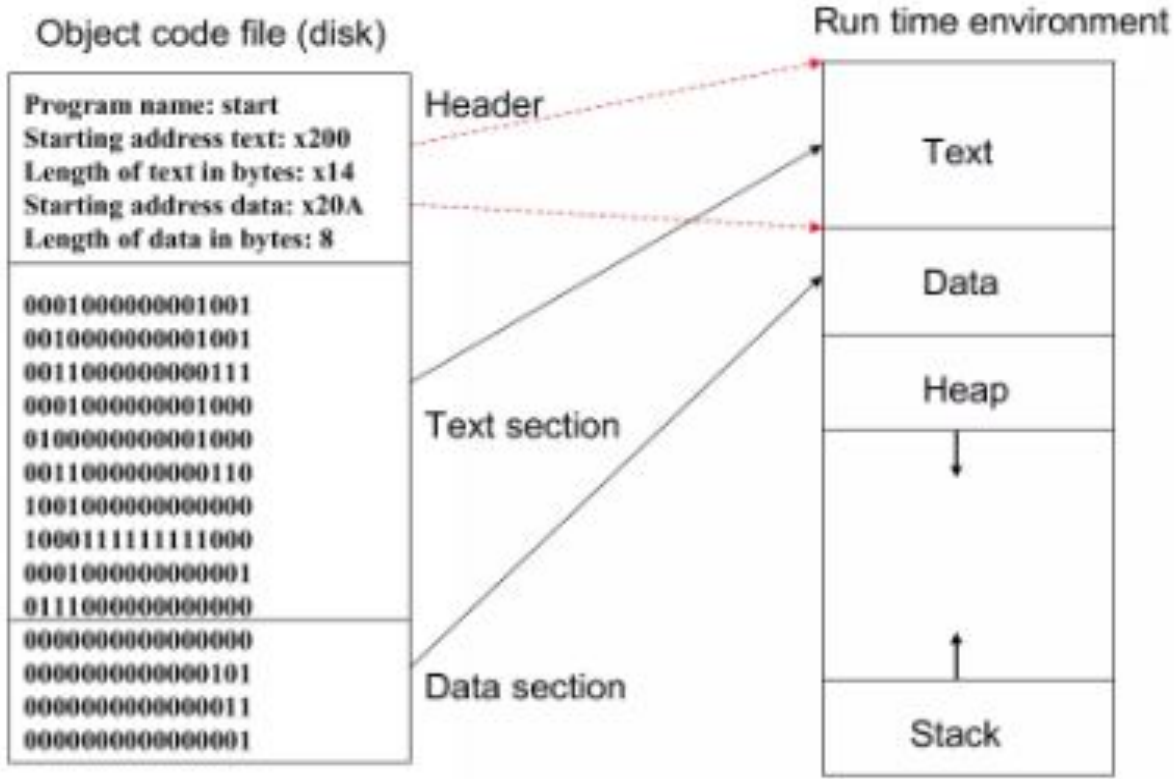
The absolute loader will load the program at memory location x200:

1.- The header record is checked to verify that the correct program has been presented for loading.

2.- Each text record is read and moved to the indicate address in memory

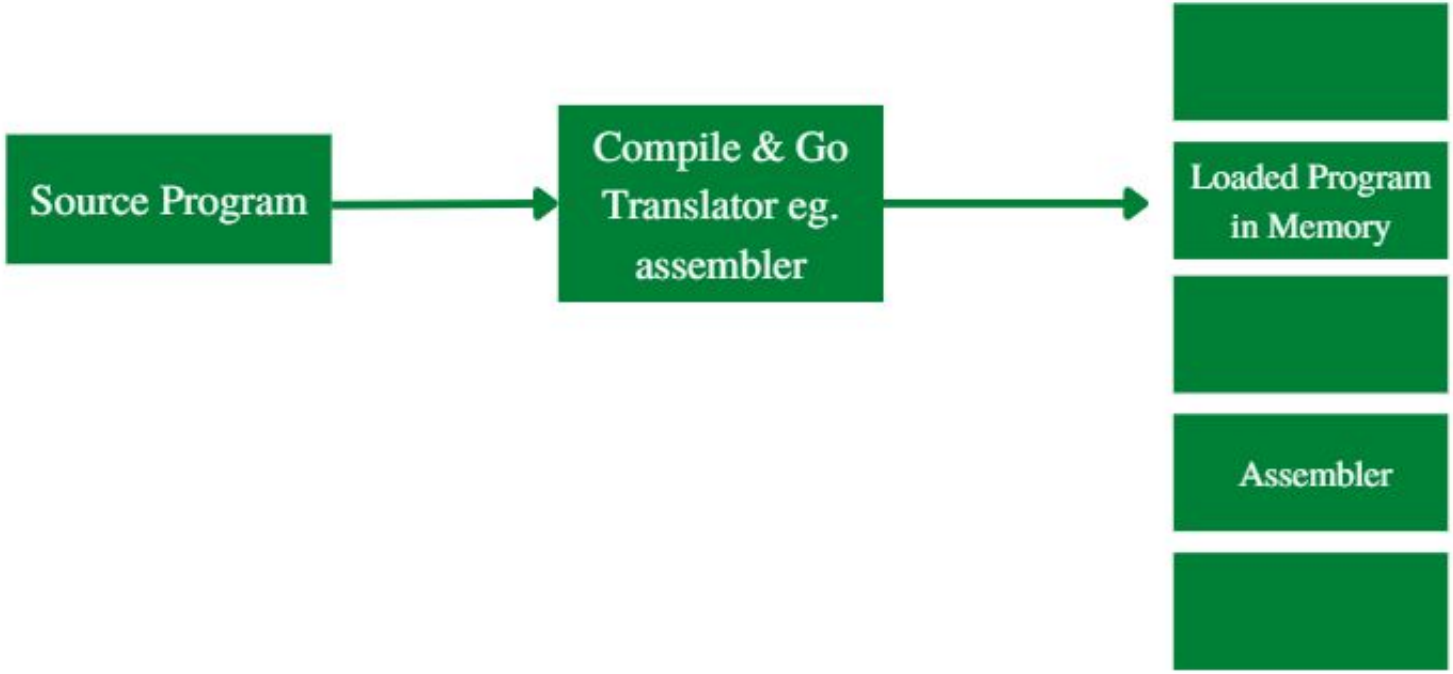
3.- When the "end" record (EOF) is encountered, the loader jumps to the specified address to begin execution.

Loading object code into memory



Compile and Go

- Assembler does the process of compiling.
- In this scheme, the source code goes into the translator line by line, and then that single line of code loads into memory.
- Line-by-line code goes to the translator so there is no proper object code.
- Because of that, if the user runs the same source program, every line of code will again be translated by a translator. So here re-translation happens.
- WATFUR-77, a FORTRAN compile



Compile and go loader scheme

Simple bootstrap loader

- When a computer is first turned on or restarted, a special type of absolute loader, called bootstrap loader is executed.
- This bootstrap loads the first program to be run by the computer -- usually an operating system.
- The bootstrap itself begins at address 0.
- It loads the OS starting address 0x80. No header record or control information, the object code is consecutive bytes of memory.