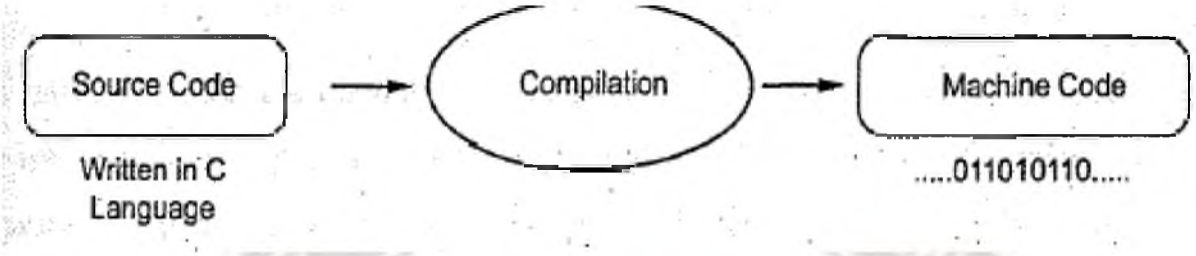
ASSEMBLY, LINKING AND LOADING

Introduction

The compilation process in C is converting an understandable human code into a machine understandable code.



Compilation process in C involves four steps:

(i) Preprocessing,

(ii) Compiling,

(iii) Assembling, and

(iv) Linking.

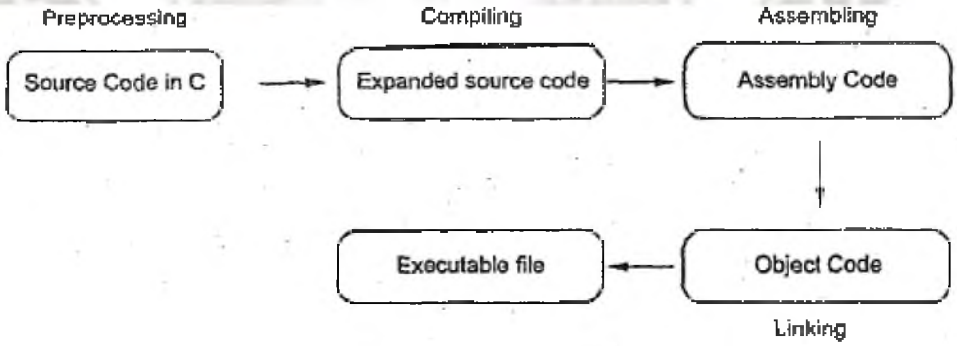
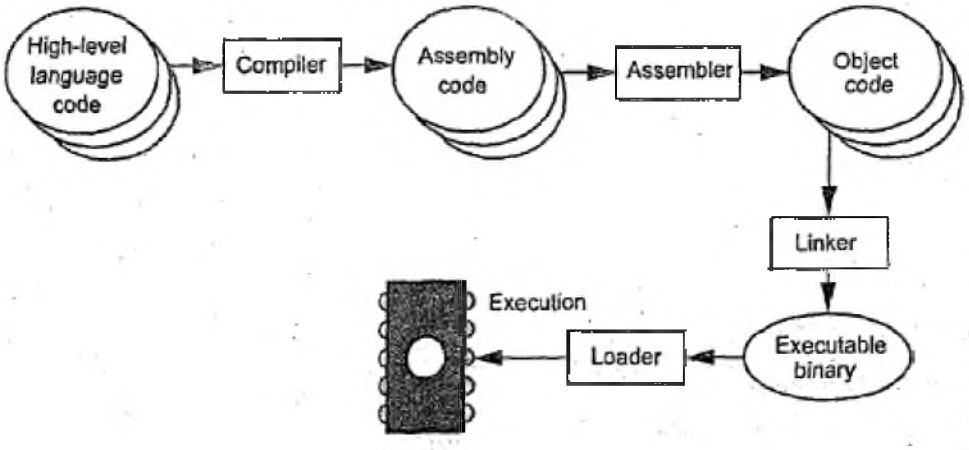


Fig 6.12 Steps involved in compilation process

(1) Program Generation Work Flow:



Most of the compilers do not directly generate machine code. A compiler is a software that converts the source code (or) high-level language into lite instruction-level program in the form of human-reddable assembly language.

The assembler's job is to translate symbolic assembly language statements into a bit-level representations of instructions known as object code.

The final steps in determining the addresses of instructions and data are performed by the linker, which produces an executable binary file. The program that brings the program into memory for execution is called a loader.

Absolute And Relative Addresses

There are two types of addressing:

(i) Absolute addresses, and

(ii) Relative addresses.

Assemblers

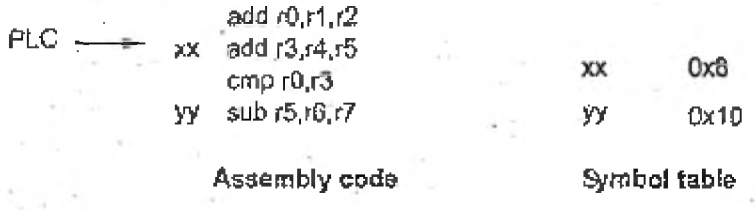
When translating an assembly code into object code, the assembler must translate opcodes and format the bits in each instruction, and then translate th? labels into addresses.

Labels make the assembly process more complex, but they are the mo\i important abstraction provided by the assembler. Label processing requires two passes through the assembly source code:

(i) The first pass scans the code to determine the address of each label.

(ii) The second pass assembles the instructions using the label values compuicj in the first pass.

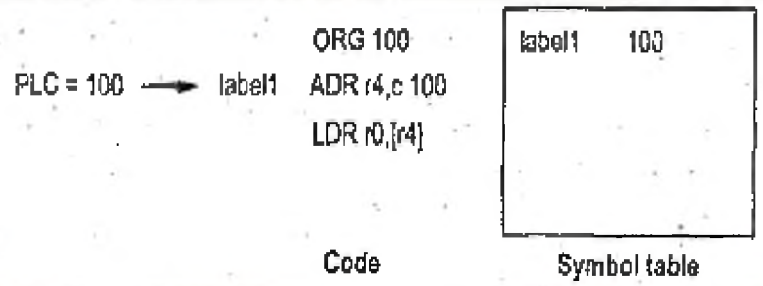
(1) Symbol Table



During scanning, the current location in memory is kept in a Program Location Counter (PLC). At the start of.the first pass, the PLC is set io the program's starting address and the assembler points to the first line.

After examining the line, the assembler updates the PLC to the next location and looks at the next instruction.

If the instruction begins with a label, a new entry is made in the symbol table, which includes the label name and its value. The value of the label is equal to the current value of the PLC. At the end of the first pass, the assembler rewinds to the beginning of the assembly language file to make the second pass.



During the second pass, when a label name is found, the label is looked up in the symbol table and its value substituted into the appropriate place in the instruction.

(2) Object Code Formats

The assembler produces an object file that describes the instructions, data and any addressing information in the binary format. A commonly used object file format, originally developed for Unix is Common Object File Format (COFF).

Linking

Linker:

A linker is a software tool that plays a crucial role in the compilation process of a program; It takes the object code generated by an assembler and combines it with other necessary libraries and modules to create an executable file.

The linker takes care of resolving references between different parts of your program. When you write code, you often divide it into multiple source files or

the modules. The linker ensures that all the necessary functions and variables from different modules are correctly connected and allowing your program to run smoothly.

The linker operates on the object files created by the assembler and modifies the assembled code to make the necessary links between the files.

Some labels will be both defined and used in the same file.

```
label1  LDR r0,[r1]
        ...
        ADR a
        ...
        B label2
        ...
var1    % 1
```

External references	Entry points
a	label1
label2	var1

File 1

```
label2  ADR var1
        ...
        B label3
        ...
x       % 1
y       % 1
a       % 10
```

External references	Entry points
var1	label2
label3	x
	y
	a

File 2

The place in the file where a label is defined is known as an entry point. The place in the file where the label is used is called an external reference.

The main job of the loader is to resolve external references based on available entry points. External references are identified in the object code by their relative symbol identifiers.

(1) Unking Process:

The linker proceeds into two phases:

(i) First, it determines the address of the start of each object file. When the loader is run or by creating a load map file that gives the order in which files are to be placed in memory and the length of each object file, it is easy to compute the starting address of an each file.

(ii) At the start of second phase, the loader merges all the symbol tables from the object files into a single and large table. It then edits the object files to change the relative addresses into addresses. This is typically performed by having the assembler write extra bits into the object file to identify the instructions and fields that refer to labels.

(2) Dynamically Linked Libraries

Static linkers merge all the necessary object code and libraries into a single executable file, resulting in a self-contained program.

Dynamic linkers allow the program to be loaded into memory at runtime and link to shared libraries which enabling more flexibility and an efficient memory usage.

Object Code Design

(1) Memory Map Design

The linker allows us to control where object code modules are placed in memory. We may need to control the placement of several types of data:

(i) Interrupt vectors and other information for I/O devices must be placed in a specific locations.

(ii) Memory management tables must be set up.

(iii) Global variables used for communication between processes must be put in locations that are accessible to all the users of that data.

(2) Reentrancy:

Many programs should be designed to be reentrant. A program is reentrant if it can be interrupted by another call to the function without changing the results of either call.

A program is recursive If the program changes the value of global variables, it friay give a different answer.

(3) Relocatability:

A program is relocatable if it can be executed when loaded into different parts of memory. It requires some sort of support from hardware that provides address calculation.