جامـــعـة المـــسـتـقـبـل
AL MUSTAQBAL UNIVERSITY

قســم الامـــن الســـــــيبرانـــــــي

Department of Cyber Security

Subject:

Compiler Design

Class:

Third

Lecturer:

Asst. Prof. Dr. Ali Kadhum Al-Quraby

Lecture: (1)

Introduction

❖ Programming Language

In computer programming, a programming Language serves as a means of communication between the person with a problem and the computer used to help solve it. An effective programming language enhances both the development and the expression of computer programs. It must bridge the gap between the often-unstructured nature of human thought and precision required for computer execution.

A hierarchy of programming languages based on increasing machine independence includes the following:

1. **Machine – Level Languages.**
2. **Assembly Languages.**
3. **High – level Language.**
4. **Problem – Oriented Languages.**

1. **Machine – Level Languages: -** is the lowest form of computer language. Each instruction in a program is represented by a numeric code, and numerical addresses are used throughout the program to refer to memory locations in the computer's memory.
2. **Assembly Languages: -** is essentially a symbolic version of a machine-level language. Each operation code is given a symbolic code such as ADD for addition and MUL for multiplication.

   Assembly-language systems offer certain diagnostic and debugging assistance that is normally not available at the machine level.

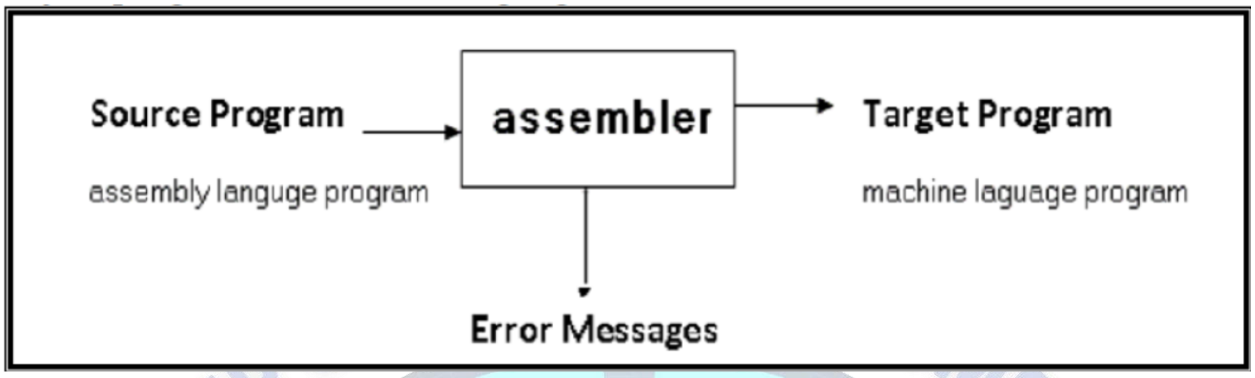
3. **High level language: -** such as FORTRAN, PASCAL, C++, …,etc. it offers most of the features of an assembly language. While some facilities for accessing system-level features may not be provided, a high-level language offers a more enriched set of language features such as structured control constructs, nested statements, blocks, and procedures.
4. **A problem-oriented language: -** treating problems in a specific application or problem area. Such as (SQL) for database retrieval applications and COGO for civil engineering applications.

Programming language (high level language) can be depicted as notations for describing computation to people and to machine. The world as we know it depends on programming languages, because all the software running on all the computers was written in some programming language. But, before a program can be run, it first must be translated into a form in which it can be executed by a computer. The software to do this translation is called ***Compiler.***

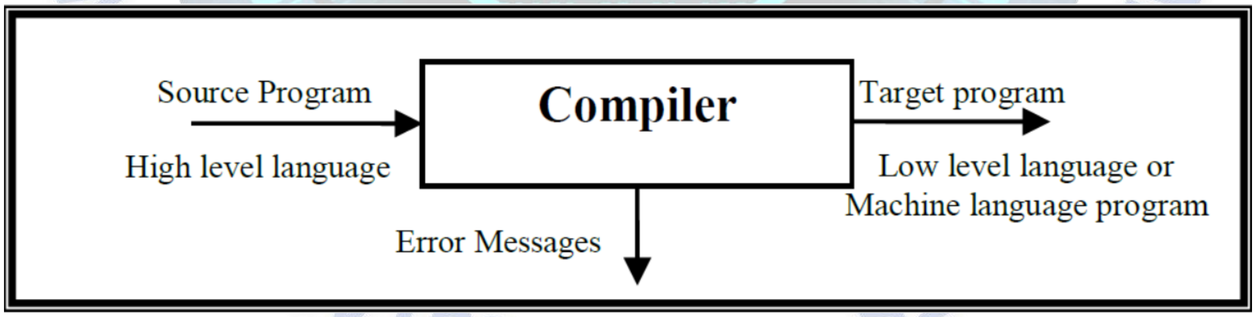
❖ Translator

A translator is program that takes as input a program written in a given programming language (the source program) and produce as output program in another language (the object or target program). As an important part of this translation process, the compiler reports to its user the presence of errors in the source program.

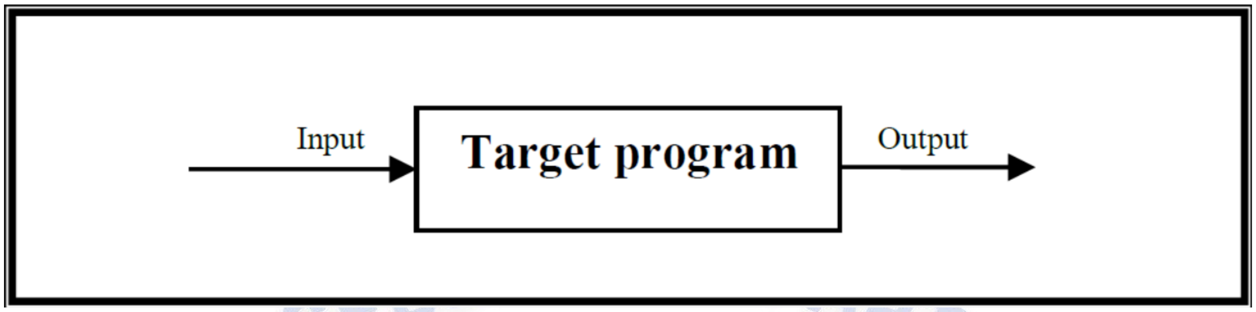
If the source language being translated is assembly language, and the object program is machine language, the translator is called ***Assembler.***



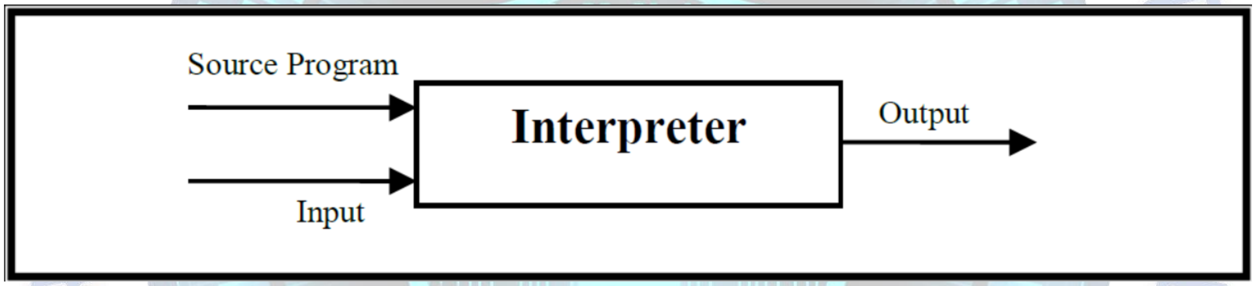
A translator, which transforms a high-level language such as C in to a particular computers machine or assembly language, called Compiler. A compiler translates (or ***compiles***) a program written in a high-level programming language that is suitable for human programmers into the low-level machine language that is required by computers. During this process, the compiler will also attempt to spot and report obvious programmer mistakes.



If the target program is an executable machine – language program, it can then be called by the user to process inputs and produce outputs.



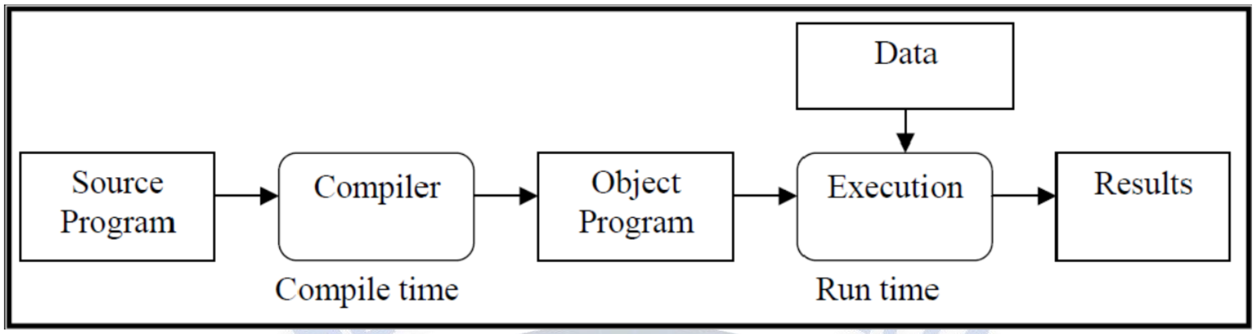
Another common kind of translator (language processors) called an **Interpreter**, in this kind, instead of producing a target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user, Figure below shows the role of interpreter



The machine language target program produced by a compiler is usually faster than an interpreter at mapping inputs to outputs. An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

Note: - The execution of a program written in a high-level language is basically take two steps: -

1- The source program must first be compiled (translated into the object program)
2- The object program is loaded into memory to be executed.

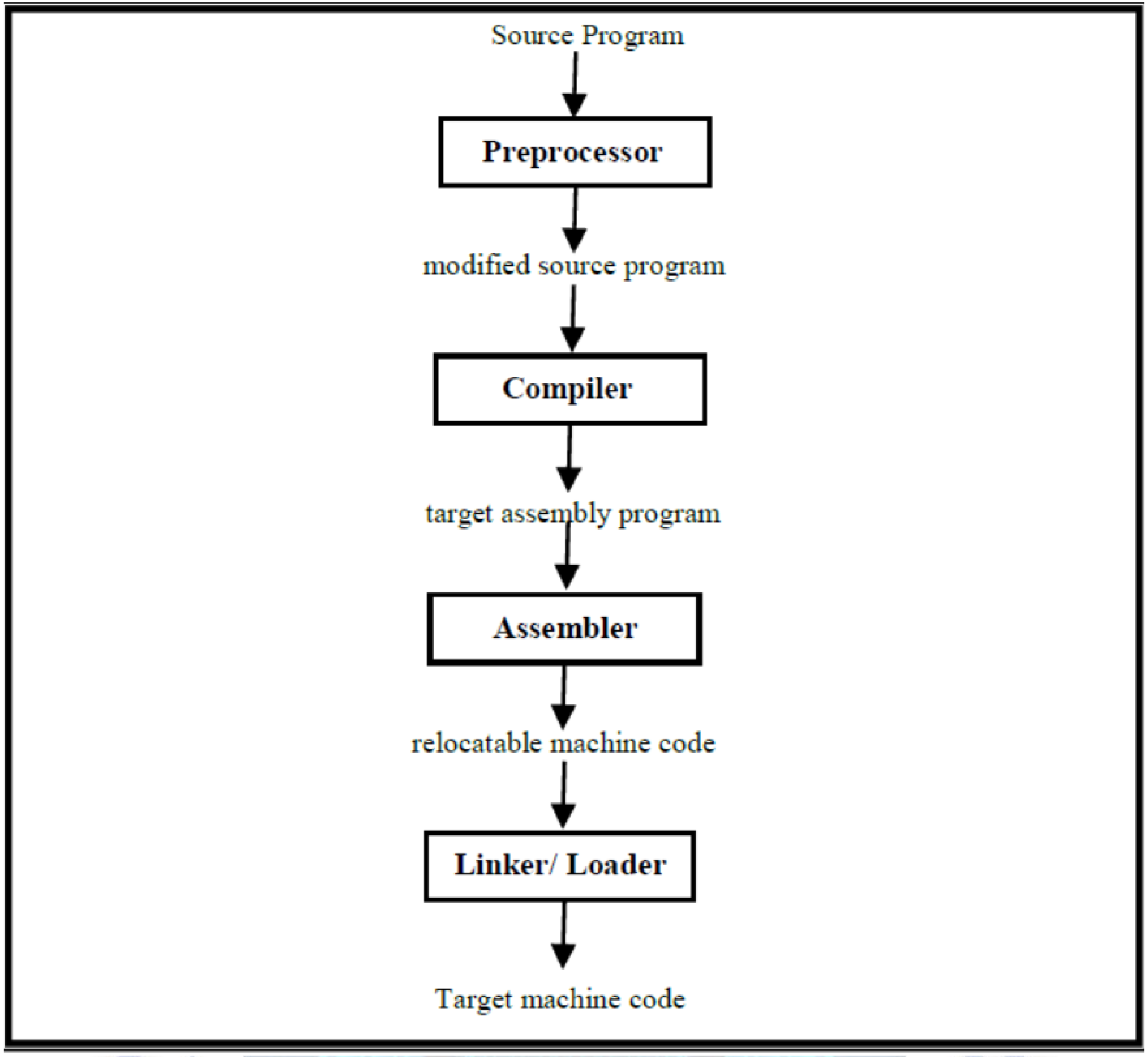


The time used to compile the program called ***<u>Compile time</u>*** and time that used to run the object program is called ***<u>Run time</u>***.

In addition to a compiler, several other programs may be required to create an executable target program, as shown in figure below. A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes entrusted to a separate program, called a ***<u>preprocessor.</u>***

The modified source program is then fed to a compiler. The compiler may produce an assembly-language program as its output, because assembly language is easier to produce as output and is easier to debug. The assembly language is then processed by a program called an ***<u>assembler</u>*** that produces relocatable machine code as its output.

Large programs are often compiled in pieces, so the relocatable machine code may have to be linked together with other relocatable object files and library files into the code that actually runs on the machine. The ***<u>linker</u>*** resolves external memory addresses, where the code in one file may refer to a location in another file. The ***<u>loader</u>*** then puts together all of the executable object files into memory for execution.



❖ The Structure of Compiler

The Compiler consists of two parts:

1. Analysis.

2. Synthesis.

The ***analysis*** part breaks up the source program into constituent pieces and imposes a grammatical structure on them. It then uses this structure to create an intermediate representation of the source program. If the analysis part detects that the source program is either syntactically ill formed or semantically unsound, then it must provide informative messages, so the user can take corrective action. The analysis part also collects information about the source program and stores it in a data structure called a ***symbol table****,* which is passed along with the intermediate representation to the synthesis part.

The ***synthesis*** part constructs the desired target program from the intermediate representation and the information in the symbol table. The analysis part is often called the ***front end*** of the compiler; the synthesis part is the ***back end***. If we examine the compilation process in more detail, we see that it operates as a sequence of ***phases****,* each of which transforms one representation of the source program to another. A typical decomposition of a compiler into phases is shown in the following Figure. In practice, several phases may be grouped together, and the intermediate representations between the grouped phases need not be constructed explicitly. The symbol table, which stores information about the entire source program, **is used by all phases of the compiler**. Some compilers have a **machine-independent optimization phase** between the front end and the back end. The purpose of this optimization phase is to perform transformations on the intermediate representation, so that the back end can produce a better target program than it would have otherwise produced from an unoptimized intermediate representation. Since optimization is optional, one or the other of the two optimization phases shown in Fig. 1.6 may be missing.

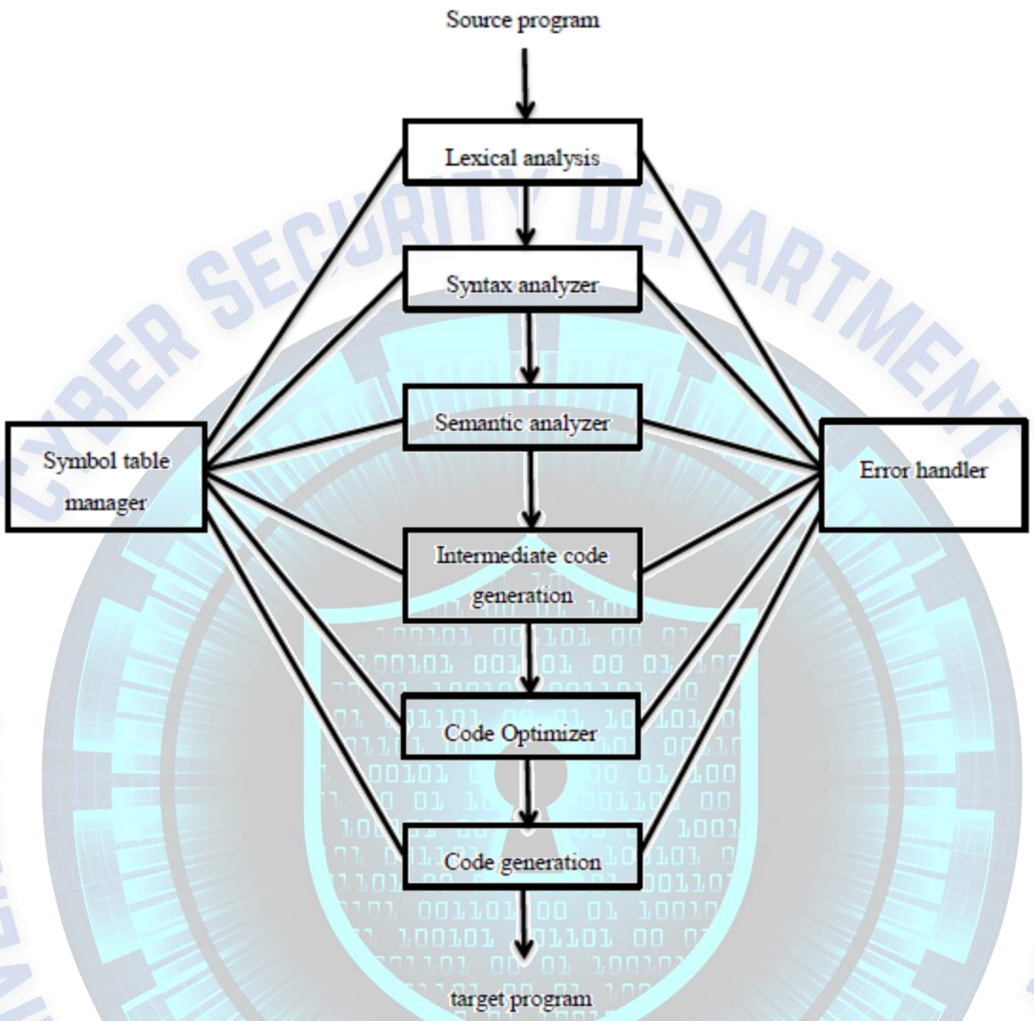


Figure shows the phases of compilers