INTRODUCTION

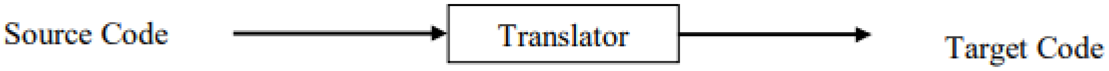
TRANSLATORS

- A translator is a program that takes as input a program written in one language and produces as output a program in another language.
- Beside program translation, the translator performs another very important role, the error-detection. Any violation of d HLL (High Level Language) specification would be detected and reported to the programmers.

Important Role of Translator are:

- Translating the HLL program input into an equivalent ml program.
- Providing diagnostic messages wherever the programmer violates specification of the HLL

A translator or language processor is a program that translates an input program written in a programming language into an equivalent program in another language.



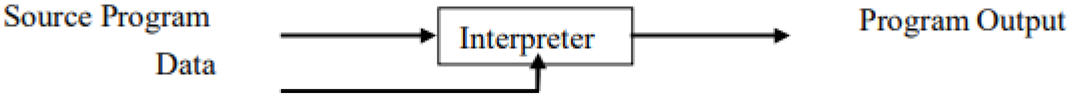
Execution in Translator

Types of Translators:

- Interpreter
- Assembler
- Compiler

Interpreter

An interpreter is a program that appears to execute a source program as if it were machine language. It is one of the translators that translate high level language to low level language.



Execution in Interpreter

During execution, it checks line by line for errors. Languages such as BASIC, SNOBOL, LISP can be translated using interpreters. JAVA also uses interpreter. The process of interpretation can be carried out in following phases.

- Lexical analysis
- Syntax analysis
- Semantic analysis
- Direct Execution

Example: BASIC, Lower Version of Pascal, SNOBOL, LISP & JAVA

Advantages:

- Modification of user program can be easily made and implemented as execution proceeds.
- Type of object that denotes various may change dynamically.

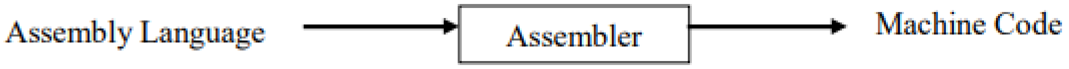
- Debugging a program and finding errors is simplified task for a program used for interpretation.
- The interpreter for the language makes it machine independent.

Disadvantages:

- The execution of the program is slower
- Memory consumption is more

Assembler

- Programmers found it difficult to write or read programs in machine language. They begin to use a mnemonic (symbols) for each machine instruction, which they would subsequently translate into machine language. Such a mnemonic machine language is now called an assembly language.
- Programs known as assembler were written to automate the translation of assembly language in to machine language. The input to an assembler program is called source program, the output is a machine language translation (object program).
- It translates assembly level language to machine code.



Example: Microprocessor 8085, 8086.

Advantages:

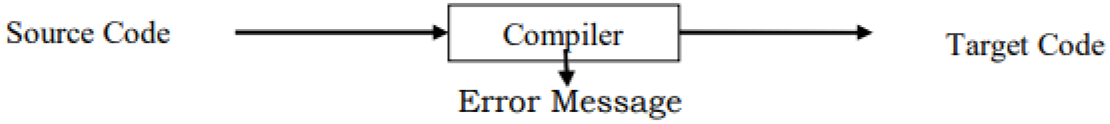
- Debugging and verifying
- Making compilers->Understanding assembly coding techniques is necessary for making compilers, debuggers and other development tools.
- Optimizing code for size.
- Optimizing code for speed.

Disadvantages:

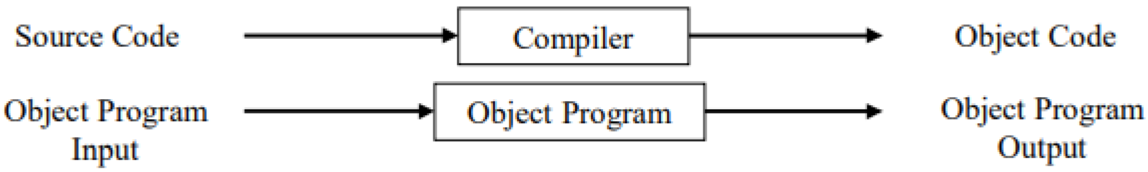
- Development time. Writing code in assembly language takes much longer than writing in a high-level language.
- Reliability and security. It is easy to make errors in assembly code.
- Debugging and verifying. Assembly code is more difficult to debug and verify because there are more possibilities for errors than in high-level code.

Compiler

Compiler is a translator program that translates a program written in (HLL) the source program and translate it into an equivalent program in (MLL) the target program. As an important role of a compiler is error showing to the programmer.

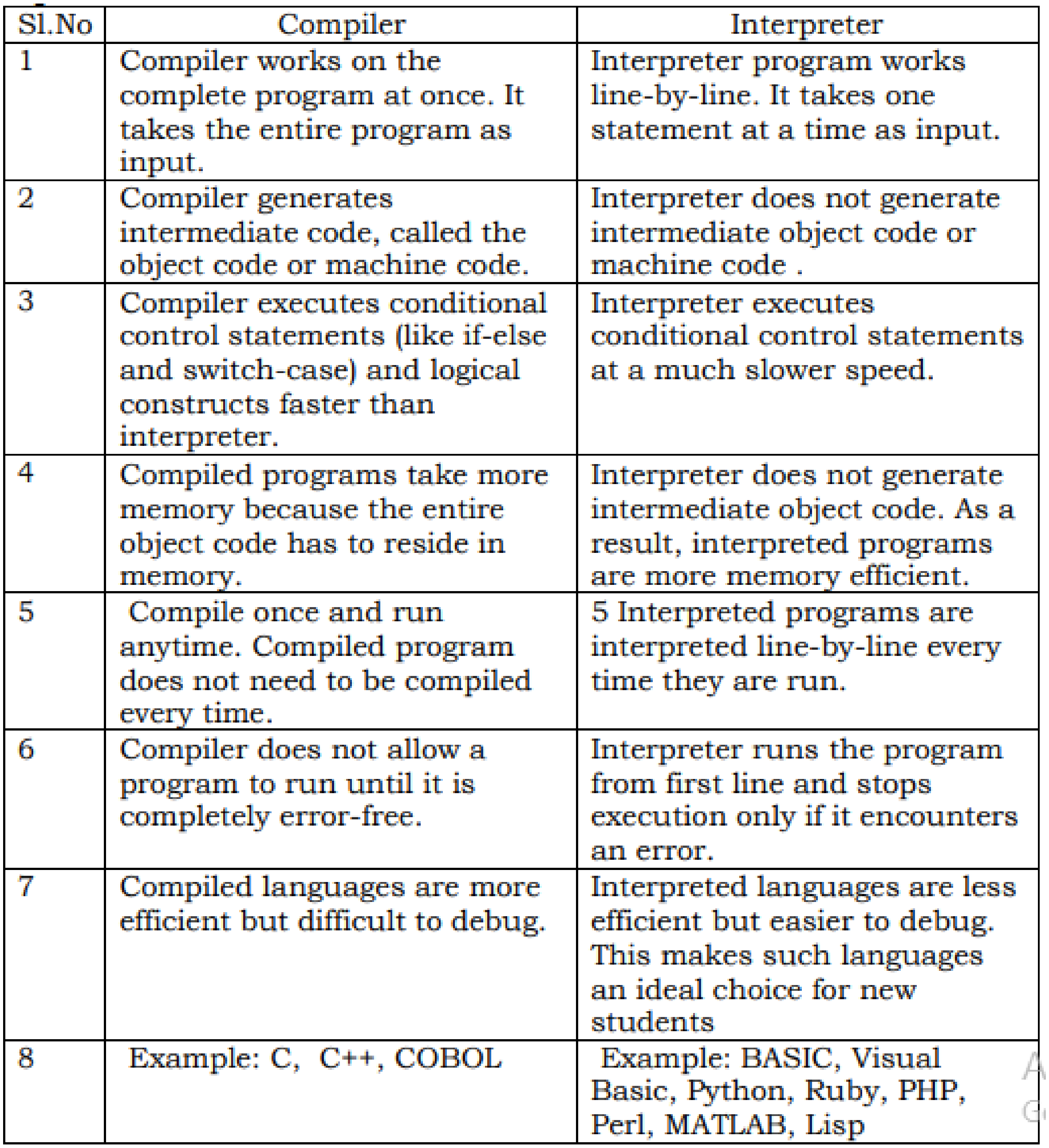


Executing a program written in HLL programming language is basically of two parts. The source program must first be compiled translated into an object program. Then the results object program is loaded into a memory executed.



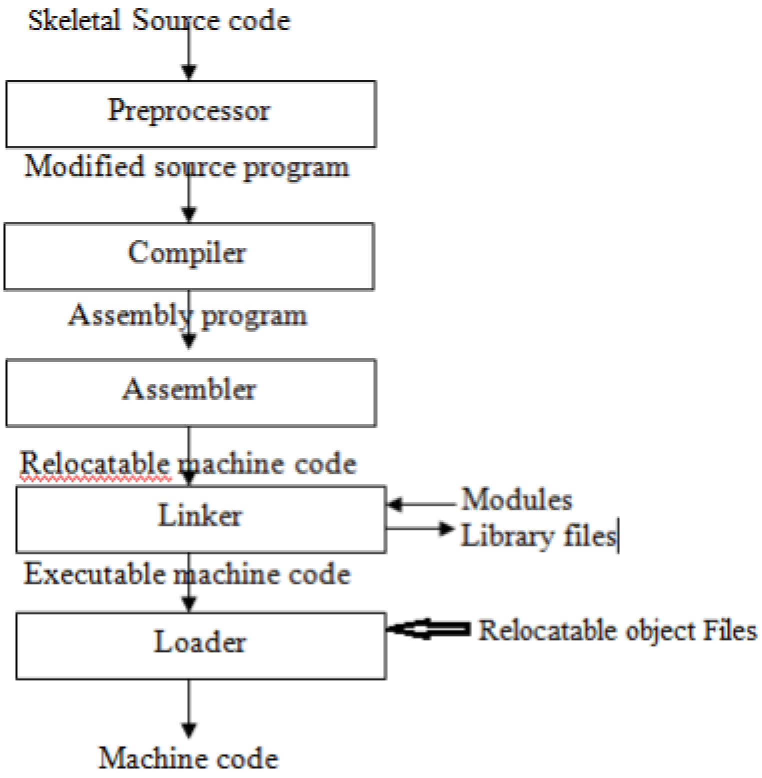
Example: C, C++, COBOL, higher version of Pascal.

Difference between Compiler and Interpreter



Sl.No	Compiler	Interpreter
1	Compiler works on the complete program at once. It takes the entire program as input.	Interpreter program works line-by-line. It takes one statement at a time as input.
2	Compiler generates intermediate code, called the object code or machine code.	Interpreter does not generate intermediate object code or machine code .
3	Compiler executes conditional control statements (like if-else and switch-case) and logical constructs faster than interpreter.	Interpreter executes conditional control statements at a much slower speed.
4	Compiled programs take more memory because the entire object code has to reside in memory.	Interpreter does not generate intermediate object code. As a result, interpreted programs are more memory efficient.
5	Compile once and run anytime. Compiled program does not need to be compiled every time.	5 Interpreted programs are interpreted line-by-line every time they are run.
6	Compiler does not allow a program to run until it is completely error-free.	Interpreter runs the program from first line and stops execution only if it encounters an error.
7	Compiled languages are more efficient but difficult to debug.	Interpreted languages are less efficient but easier to debug. This makes such languages an ideal choice for new students
8	Example: C, C++, COBOL	Example: BASIC, Visual Basic, Python, Ruby, PHP, Perl, MATLAB, Lisp

Language processors (COUSINS OF THE COMPLIER or Language Processing System)



1. Preprocessors:

- It produces input to Compiler. They may perform the following functions.

Macro Processing:

- A preprocessor may allow a user to define macros that are shorthand's for longer constructs.

File inclusion:

- A preprocessor may include header files into the program text. For example, the C preprocessor causes the contents of the file to replace the statement #include when it processes a file containing this statement.

Rational preprocessors:

- These preprocessors augment older language with more modern flow of control and data structuring facilities. If constructs like while-statements does not exist in the programming language, then this preprocessor provides it. Language extensions:
- These preprocessor attempts to add capabilities to the language by what amounts to build in macros. For example, Equal a database query language embedded in C. Statement beginning with ## are taken as preprocessor to be database access statements, unrelated to C and are translated into procedure calls on routines that perform the database access.

2. Complier:

- It converts the source program (HLL) into target program (LLL).

3. Assemblers:

- It converts an assembly language (LLL) into machine code. Some compilers produce assembly for further processing.
- Other compilers perform the job of the assembler, producing relocatable machine code that can be passed directly to the loader/link-editor.
- Assembly code is a mnemonic version of the machine code, in which names are used instead of binary codes for operation and names are also given to memory addresses. A typical sequence of assembly instructions might be

```
MOV a, R1
ADD #2, R1
MOV R1, b
```

4. Loader and Link Editors:

Loader:

- The process of loading consists of taking relocatable machine code, altering the relocatable addresses and placing the altered instructions and data in memory at the proper locations.
- The Link-editor allows us to make a single program from several files of relocatable machine code. These files may have been the result of several different compilations, and one or more may be library files of routines provided by the system and available to any program that needs them.

Link Editor:

- It allows us to make a single program from several files of relocatable machine code.