CSC 218 Foundation of Sequential Program
Lecture Note 3
(Assemblers Specification and Translation of P/L)

Translators

A translator is a programming language processor that converts a computer program from one language to another. It takes a program written in source code and converts it into machine code. It discovers and identifies the error during translation.

Purpose of Translator

It translates high-level language program into a machine language program that the central processing unit (CPU) can understand. It also detects errors in the program.

Different Types of Translators

There are 3 different types of translators as follows:

Compiler

A compiler is a translator used to convert high-level programming language to low-level programming language. It converts the whole program in one session and reports errors detected after the conversion. Compiler takes time to do its work as it translates high-level code to lower-level code all at once and then saves it to memory. A compiler is processor-dependent and platform-dependent. But it has been addressed by a special compiler, a cross-compiler and a source-to-source compiler. Before choosing a compiler, user has to identify first the Instruction Set Architecture (ISA), the operating system (OS) and the programming language that will be used to ensure that it will be compatible.

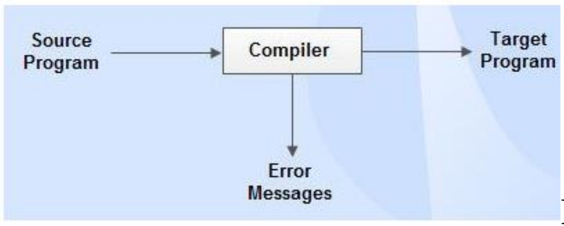


Fig.: Compiler

Interpreter

Just like a compiler, is a translator used to convert high-level programming language to low-level programming language. It converts the program one at a time and reports errors detected at once, while doing the conversion. With this, it is easier to detect errors than in a compiler. An interpreter is faster than a compiler as it immediately executes the code upon reading the code. It is often used as a debugging tool for software development as it can execute a single line of code at a time. An interpreter is also more portable than a compiler as it is not processor-dependent, you can work between hardware architectures.

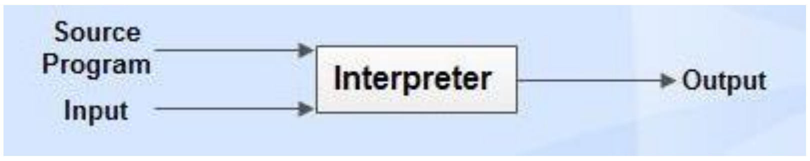


Fig.: Interpreter

Assembler

An assembler is is a translator used to translate assembly language to machine language. It is like a compiler for the assembly language but interactive like an interpreter. Assembly language is difficult to understand as it is a low-level programming language. An assembler translates a low-level language, an assembly language to an even lower-level language, which is the machine code. The machine code can be directly understood by the CPU.

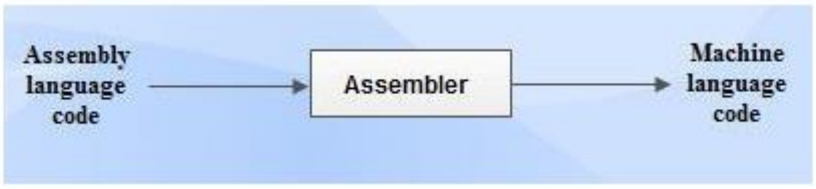
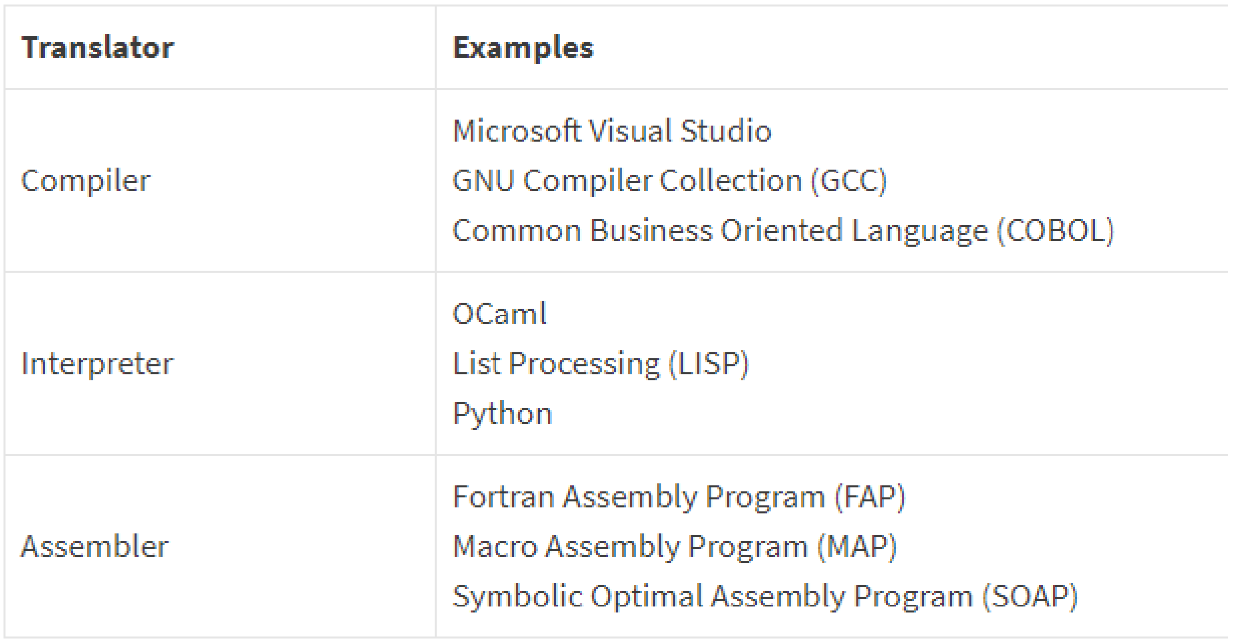


Fig.: Assembler

Examples of Translators

Here are some examples of translators per type:



Translator	Examples
Compiler	Microsoft Visual Studio GNU Compiler Collection (GCC) Common Business Oriented Language (COBOL)
Interpreter	OCaml List Processing (LISP) Python
Assembler	Fortran Assembly Program (FAP) Macro Assembly Program (MAP) Symbolic Optimal Assembly Program (SOAP)

Advantages and Disadvantages of Translators

Here are some advantages of the Compiler:

- The whole program is validated so there are no system errors.
- The executable file is enhanced by the compiler, so it runs faster.
- User do not have to run the program on the same machine it was created.

Here are some disadvantages of the Compiler:

- It is slow to execute as you have to finish the whole program.
- It is not easy to debug as errors are shown at the end of the execution.
- Hardware specific, it works on specific machine language and architecture.

Here are some advantages of the Interpreter:

- You discover errors before you complete the program, so you learn from your mistakes.
- Program can be run before it is completed so you get partial results immediately.

- You can work on small parts of the program and link them later into a whole program.

Here are some disadvantages of the Interpreter:

- There's a possibility of syntax errors on unverified scripts.
- Program is not enhanced and may encounter data errors.
- It may be slow because of the interpretation in every execution.

Here are some advantages of the Assembler:

- The symbolic programming is easier to understand thus time-saving for the programmer.
- It is easier to fix errors and alter program instructions.
- Efficiency in execution just like machine level language.

Here are some disadvantages of the Assembler:

- It is machine dependent, cannot be used in other architecture.
- A small change in design can invalidate the whole program.
- It is difficult to maintain.

INTRODUCTION TO ASSEMBLY LANGUAGE

Assembly Language is a low-level programming language. It helps in understanding the programming language to machine code. In computers, there is an assembler that helps in converting the assembly code into machine code executable. Assembly language is designed to understand the instruction and provide to machine language for further processing. It mainly depends on the architecture of the system whether it is the operating system or computer architecture.
Assembly Language mainly consists of mnemonic processor instructions or data, and other statements or instructions. It is produced with the help of compiling the high-level language source code like C, C++. Assembly Language helps in fine-tuning the program.

Why is Assembly Language Useful?

Assembly language helps programmers to write human-readable code that is almost similar to machine language. Machine language is difficult to understand and read as it is just a series of numbers. Assembly language helps in providing full control of what tasks a computer is performing.

Example:

Find the below steps to print “Hello world” in Windows

1. Open the notepad.
2. Write below code

```
global  _main
extern  _printf
section .text
_main:
push    message
call    _printf
add     esp, 4
ret
message:
db  'Hello, World!', 10, 0
```

3. Save the file with any name example XYZ.asm, the extension should be “.asm”.
4. The above file needs to compile with the help of an assembler that is NASM (Netwide Assembler).
5. Run the command nasm –f win32 XYZ.asm
6. After this, Nasm creates one object file that contains machine code but not the executable code that is XYZ.obj
7. To create the executable file for windows Minimal GNU is used that provides the GCC compiler.
8. Run the command gcc –o XYZ.exe XYZ.obj

9. Execute the executable file now “XYZ”
10. It will show the output as “Hello, world”.

Why You Should Learn Assembly Language?

The learning of assembly language is still important for programmers. It helps in taking complete control over the system and its resources. By learning assembly language, the programmer is able to write the code to access registers and able to retrieve the memory address of pointers and values. It mainly helps in speed optimization that increases efficiency and performance.

Assembly language learning helps in understanding the processor and memory functions. If the programmer is writing any program that needs to be a compiler that means the programmer should have a complete understanding of the processor. Assembly language helps in understanding the work of processors and memory. It is cryptic and symbolic language.

Assembly Language helps in contacting the hardware directly. This language is mainly based on computer architecture and it recognizes the certain type of processor and its different for different CPUs. Assembly language refers as transparent compared to other high-level languages. It has a small number of operations but it is helpful in understanding the algorithms and other flow of controls. It makes the code less complex and easy debugging as well.

Features

The features of the assembly language are mentioned below:

1. It can use mnemonic than numeric operation code and it also provides the information of any error in the code.
2. This language helps in specifying the symbolic operand that means it does not need to specify the machine address of that operand. It can be represented in the form of a symbol.
3. The data can be declared by using decimal notation.

Assemblers

The assemblers are used to translate the assembly language into machine language. There are two types of assembler are:

1. **Single-pass assembler:** A single assembler pass is referred as the complete scan of source program input to assembler or equivalent representation and translation by the statement on the basis of statement called as single pass assembler or one pass translation. It isolates the label, mnemonics and operand field of the system. It validates the code instructions by looking it up in mnemonic code table. It enters the symbol found in the label field and the address of the text available machine word into the symbol table. This pass is fast and effected, and no need to construct the intermediate code.
2. **Multi-pass assembler:** In this, an assembler goes through assembly language several times and generates the object code. In this last pass is

called a synthesis pass and this assembler requires any form of an intermediate code to generate each pass every time. It is comparatively slower than single pass assembler but there can be some actions that can be performed more than once means duplicated.

Advantages and Disadvantages

Mentioned are some advantages and disadvantages:

Advantages

Below are the advantages:

1. It allows complex jobs to run in a simpler way.
2. It is memory efficient, as it requires less memory.
3. It is faster in speed, as its execution time is less.
4. It is mainly hardware oriented.
5. It requires less instruction to get the result.
6. It is used for critical jobs.
7. It is not required to keep track of memory locations.
8. It is a low-level embedded system.

Disadvantages

Below mentioned are the disadvantages:

1. It takes a lot of time and effort to write the code for the same.
2. It is very complex and difficult to understand.
3. The syntax is difficult to remember.
4. It has a lack of portability of program between different computer architectures.
5. It needs more size or memory of the computer to run the long programs written in Assembly Language.