ISSN No:-2456 – 2165

The Comparative Study of Role of Compilers in Computer Architecture

Veena . A. More
Bachelor of Computer Application (BCA)
A. S. Patil College of Commerce (Autonomous)
Vijayapur, Karnataka
veenamore1103@gmail.com

Laxmi C. Bagalkot
Bachelor of Computer Application (BCA)
A. S. Patil College of Commerce (Autonomous)
Vijayapur, Karnataka

Abstract:-Computer is a fusion of software and hardware. Hardware is simply a chunk of computer and its functions are being controlled by compatible software. Hardware understands instructions in the form of electronic charge, which is the counterpart of binary language in software programming. Binary language has only two alphabets, 0 and 1. To instruct, the hardware codes must be written in binary format, which is simply a series of 1s and 0s. It would be a difficult and cumbersome task for computer programmers to write such codes, which is why we have compilers to write such codes.

Keywords:-Optimization, Lexemes, Loader, Parser, YACC**.**

I. INTRODUCTION

Compilers fixes the gap between high-level languages & low level language that are convenient for us to use, low-level languages that can be executed efficiently by machines. Compilers are the most important tool for programmers, and consequently much effort has been expended since the dawn of the age of computing to produce high-quality compilers.

The goal of this paper is to explain how the program accepts a source code and generates machine code computer architecture. Eventually this should lead to a more in-depth understanding of compilers. The role of compiler is a well researched area. Typically, design of the compiler use a parser generator framework like YACC to construct lexer, scanner, parser, etc.

A. WHAT IS COMPILER?

A compiler is a program that processes statements written in a programming language and converts them into machine language or also known as code that a computer's processor uses. A programmer writes language statements using an editor. The file is created that contains the source statements. The programmer then runs the compiler, which contains the source statements.

When we run, the compiler first it analyzes the language statements syntactically one after the other and then, in one or More successive passes, builds the output code, making sure those statements that refer to other statements are referred to correctly in the final code. After compilation the output is known as object code. Here object code is machine code that the processor can execute one instruction at a time.

II. LANGUAGE PROCESSING SYSTEM

Before going into the concepts of compilers, we should understand a few other tools that work closely with compilers.

A. Preprocessor

It is generally called as a part of compiler. It is a tool that generates input for compilers. It shares with macro-processing, augmentation, file inclusion, language extension, etc.

B. Interpreter

An interpreter translates high-level language into low-level machine language. It reads a statement from the input, translates it to an intermediate code, executes it, and then takes the next statement which is present in sequence. If it generates error it stops its execution and reports it.

C. Assembler

An assembler converts assembly language programs into machine code. The output of an assembler is known as object file, which contains a combination of machine instructions and data required to place these instructions in memory.

D. Linker

It is a computer program which links and merges various object files together in able to make an executable file. All files could have been compiled via separate assemblers. The major work of a linker is to find and locate referenced

module/routines in a program and to determine the memory location where these codes will be loaded.

E. Loader

The task of loader is to load all executable files into memory and execute them. The memory space is created by loader when it calculates the size of a program (instructions and data). To initiate execution it initializes various registers.

F. Cross-Compiler

A compiler which runs on one platform, it has the ability to generate executable code for other platform.

G. Source-to-Source Compiler

A compiler which takes the source code of one programming language and translates it into the source code of another programming language

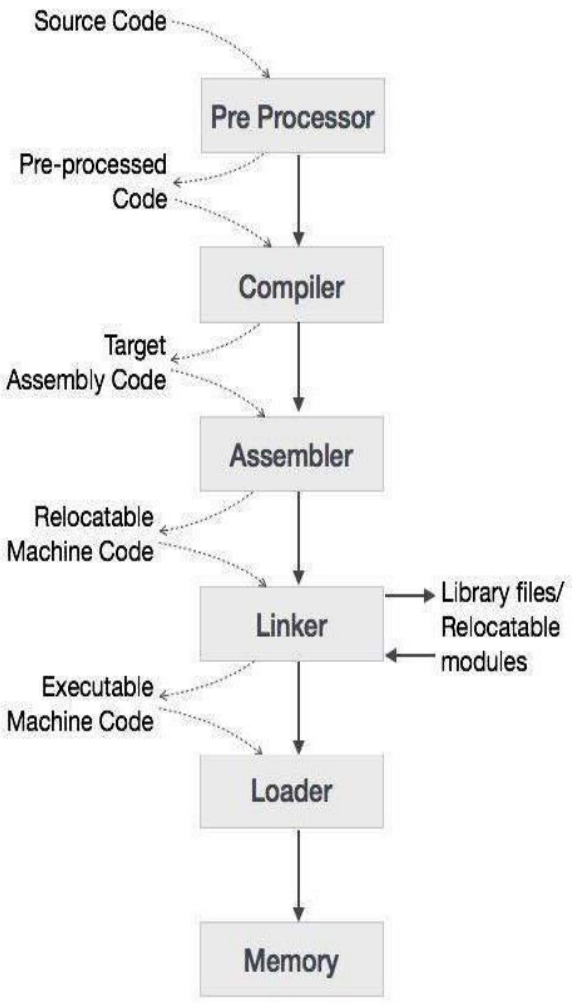

Figure 1: Language Processing System

III. COMPILER ARCHITECTURE

A compiler can be divided into two phases based on the way they compile.

A. Analysis Phase

The analysis phase which is known as the front end of the compiler reads the source program, splits it into parts and then verifies for lexical, grammar and syntax errors. The analysis phase creates an intermediate code representation of the source program and symbol table as output, but it should be fed to Synthesis phase as input.

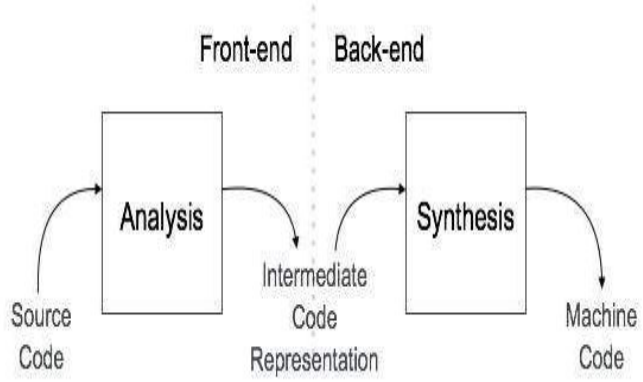

Figure 2: Compiler Architecture

B. Synthesis Phase

The synthesis phase which is known as the back end of the compiler generates the target program via intermediate source code representation and symbol table.

C. Phase and Pass of Compiler.

- **Pass** : compiler traversing the entire program.
- **Phase** : It is the stage where it takes input from the prior stage, processes and generates output which can be used as input for the next stage

IV. CONCLUSION

An effective compiler allows a more efficient execution of application programs for a given computer architecture, Both the compiler writer and machine designer have multiple objectives. When the compiler writer and the machine designer work on their objectives also having in mind the objectives of the other then it is possible achieve what they need to accomplish and also what can be accomplished using the feature or optimization built by them.

The compiler writer will benefit by having more space to research on the optimization techniques of the compiler and

the machine designer will profit on the reduced execution time it takes to implement because of the well structured architecture that complies with the needs of a compiler writer and also achieves its primary goals.

REFERENCES

[1]. William A. Wulf, ―Compilers and Computer Architecture‖, IEEE 0018-9162.
[2]. John Hennessey and David Patterson, Computer Architecture – A Quantitative Approach (fifth edition)
[3]. Torben Egidius Mogense, Basics of Compiler Design
[4]. Donavan, John J. Systems Programming. ew ork , D sseldorf McGraw-Hill, 1972. Print.
[5]. Lee, Gyungho, and Pen-Chung Yew. Interaction between Compilers and Computer Architectures. Boston: Kluwer Academic, 2001. Print.
[6]. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-035-computer-language-engineering-sma-5502-fall-2005/lecture-notes/14_wrapup.pdf
[7]. Kurt Keutzer , Wayne Wolf Anatomy of a Hardware Compiler AT&T Bell Laboratories Murray Hill NJ, 1988 ACM O-8979 l-269- l/88/0006/0095.
[8]. Mano, M. Morris. Computer Systems Architecture. Rockville, Mar.: Computer Science, 1982. Print.
[9]. Hamacher, V. Carl., Zvonko G. Vranesic, and Safwat G. Zaky. Computer Organization. New York: McGraw-Hill, 1984. Print.
[10]. http://en.wikipedia.org/wiki/Optimizing_compiler#Specific_techniques.
[11]. https://www.inkling.com/read/computer-architecture-hennessy-5th/appendix-a/section-a-8.
[12]. http://www.tkt.cs.tut.fi/tools/public/tutorials/synopsys/design_compiler/ gsdc.html#arch_opt