Representation of Intermediate Code Generation- a Compiler

Neha Bhateja, Nishu Sethi
Amity School of Engineering & Technology
Amity University Haryana, Gurgaon, India

Abstract- Compiler is a translator which is used to convert one language into another language. The compilation process to generate a target code goes through a different phase. This paper gives a brief description about the different phases of the compiler. It describes the intermediate code generation process with different ways.

Keywords: compiler, parser, tokens, Quadruples, Triples.

I. INTRODUCTION

A compiler is a set of programs (instructions) or a translator which is used to convert a given source program written in any programming language (high level language) into another target language (called computer language or knowns as binary code form called as object code) [1,2]. Compiler reads the whole source code at once and if required it generate the errors and displayed.

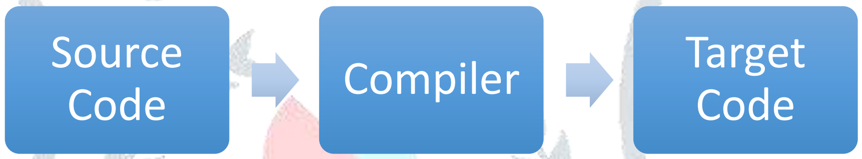

Fig.1: Process of Compiler

II. PHASES OF A COMPILER

The compilation process is categorized into two parts:

a) **Analysis Phase** – The Front End
the purpose of the analysis phase is to divide the whole source code into parts and rearrange these parts into a meaningful structure. The analysis phase perform by the compiler is machine independent [2]. The analysis phase contains the following phases of the compiler:
   (a) Lexical analysis
   (b) Syntax Analysis
   (c) Semantic Analysis

b) **Synthesis Phase** – the Back End
This phase accepts the input in terms of intermediate code and generates the targe code as output [1]. The synthesis phase is machine dependent phase. The following sub-phases comes under the synthesis phase:
   a) Intermediate code generation
   b) Code optimization
   c) Code generation

All the phases of compiler are interacting with symbol table and error handler.

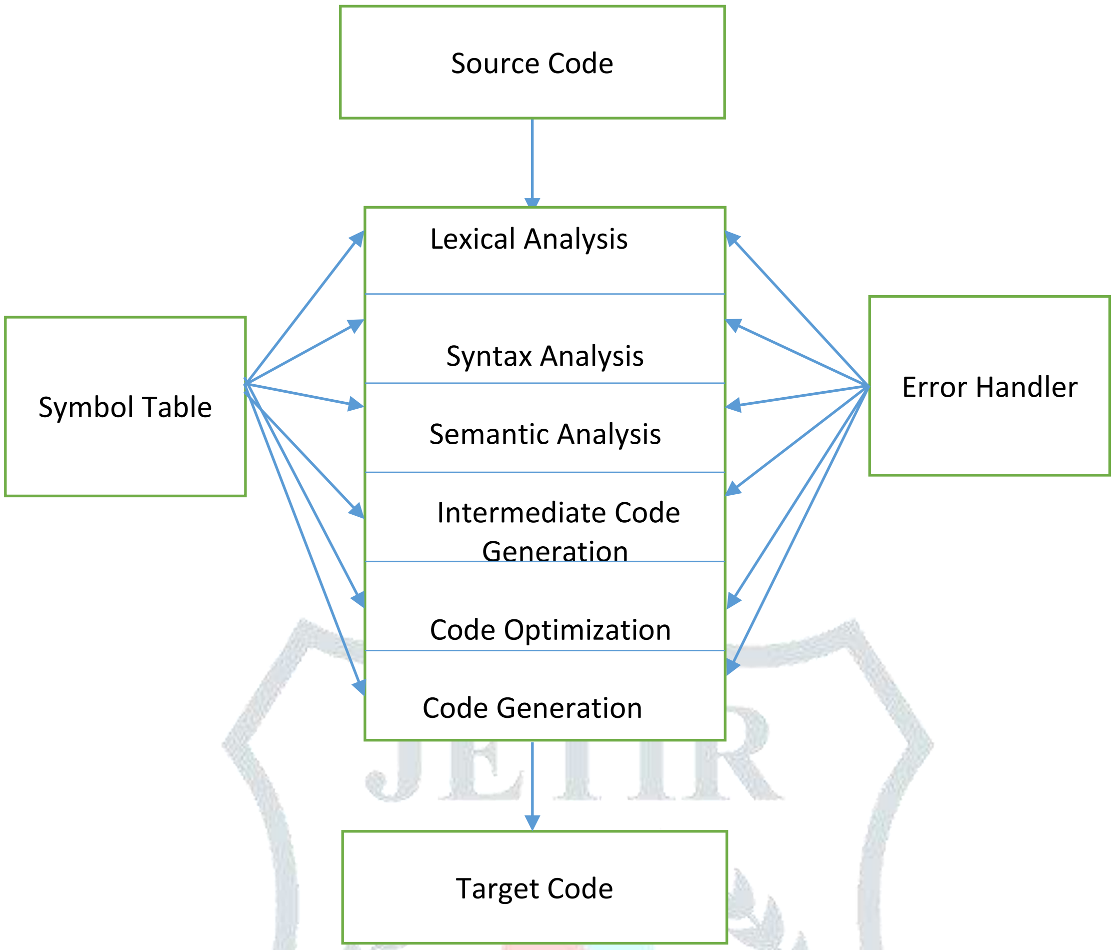

Fig. 2. Phases of Compiler

Lexical Analysis: This is the initial phase of the compiler. It is also known as scanning or lexer. It accepts the source code as input and perform the scanning on it to generates the tokens. The tokens are referring to the string of individual characters with meaning.

Syntax Analysis: This is second phase of the compiler. This phase accepts the input in terms of tokens from the previous phase (i.e. lexical analysis phase) and generate the parse tree.

Semantic Analysis: the role of the semantic analyzer is to check the meaning of the program. During this phase logical errors are checked. For example: variable undeclared, array out of bound, divide by zero etc.

Intermediate code generation: this phase of compiler is responsible for generating the intermediate representation of the source code. Intermediate code representation is an abstract code that is different from the source code. Intermediate representation can be done by various ways [6].

Code optimization: during this phase, optimization can be performed on the intermediate representation code. the reason behind performing the optimization is to increase the efficiency of the code. The code optimization is performed by various methods: dead code elimination, elimination of common-subexpression, loop optimization [5].

Code Generation: This is the last phase of the compilation process. The enhanced (optimized) intermediate code is transformed into the final

III. Why intermediate code generation??

Suppose we have n different programming languages and m different type of machine. If we want to execute n different programming languages on m different machines, then n*m compilers need to be implement. The implementation of n*m compilers is not easy task.
The above problem can be transform into n + m compiler by introducing a new language IR, known as intermediate representation [4, 7].

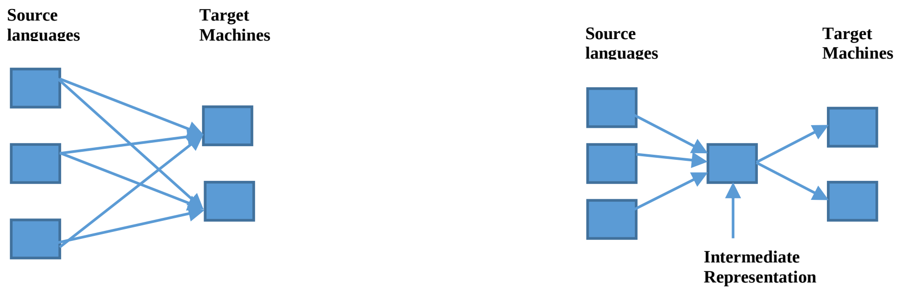

Fig.3. Translation of source code

IV. Different Types of Intermediate Code Representation

Intermediate code representation can be done by the following ways:

1) **Postfix Notation**: the ordinary way of writing an arithmetic expression is the operators between the operands. For example, x + y. In the postfix notation the representation can be done by writing the operands on the left side and operators on the right end. For example, the above expression can be written as xy+.
2) **Syntax Tree:** This is the abstracted form of the parse tree. The nodes and operators of parse tree are turned into their parents and production is represented by a single link in the syntax tree. To recognize the positions of the operands, the expressions are represented in parentheses [7].

For example: (m + n) / (m - n)

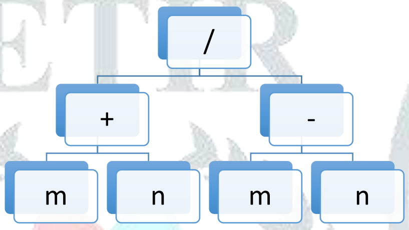

Fig.4. Syntax Tree

3) **Three-Address Code**

The three address code statements are represented in the form a= b op c
a, b and c are the variables and will have memory locations (address) and op is the operator.

For example: the three address code representation for the expression x + y * z + s :
T1 = y * z
T2= x + T1
T3= T2 + s , where T1, T2 and T3 are the temporary variables.

Representation of Three- Address Code

Three address code statements can be represented by the following ways:

a) **Quadruples:** the three address code statements are represented using a record of four fields. The first field of the record represents the operator filed which is followed by the next two operand fields and the last field of the record represents the output.
For example: r = x + y * z + s

Operator	Operand 1	Operand 2	Result
*	Y	z	T1
+	X	T1	T2
+	T2	s	T3
=	T3		r

b) **Triples:** Triples are the form of three address code which do not use any temporary variable to hold the result. When a reference is required, a pointer is used to represent that triple.
For example: x + y * z + s

	Operator	Operand 1	Operand 2
1	*	Y	z
2	+	X	(1)
3	+	(2)	s

V. CONCLUSION

This paper outlines the basic concept of translator, called compiler, which convert the source code into an object code with the help of code generation process. The code generation process accepts the input from the intermediate code generator which represents the code in some intermediate form. The intermediate representation of the source code is done by many ways like postfix notation, syntax tree and three address code.

REFERENCES

[1] Cattell R.G., Automatic derivation of code generators from machine descriptions, ACM
Trans. on Programming Languages and Systems 2(2), 1980, pp.173-190
[2] lfred V.Aho, Ravi Sethi, Jeffery D. Ullman, Addison Wesley, 2007. Compilers- Principles, Techniques, and Tools
[3] William A. Barrett, 2005. Compiler Design, CmpE 152, FALL Version, San Jose State University.
[4] Muchnick, Advanced Compiler Design and Implementation.
[5] Neha Bhateja, A Comprehensive Study on Code Optimization- Levels & Techniques, 2017, IJARCS, ISSN No. 0976-5697 page 1897-1899.
[6] http://www.ijetae.com/files/Volume4Issue3/IJETAE_0314_87.pdf
[7] https://www.geeksforgeeks.org/intermediate-code-generation-in-compiler-design

JETIR