Compiler & It's Phases

Piyush chutani[1], Rohan khanna[2], Puneet bhalla[3]

(Student, Dept. of computer science . DCE, Gurgaon, India)

ABSTRACT

Compiler is the set of program that translate a program written in source language into target language (machine language). In this paper we discuss about compiler, analysis synthesis model of compiler, phases of compiler, need of compiler and error handler. As we know source program or code written in high level language and convert into machine language. After compilation target program can be executed to process input and produce output. Each phase perform their own task and produce output. These phases linked to each other and output of one phase passes to other and final intermediate program is produced from compiler. Working of each phase is described in this paper.

Key words- *Design, Token, attribute, model, memory.*

1. INTRODUCTION

Compiler is a program that compiles a source program into target program. Compilation is the term or concept in production of any software. It act as link between application written in high level platform and low level platform(where the application executed on machine.) The compilation means translation of high level code into machine level code.

The output of translation is independent executable program that can run directly.

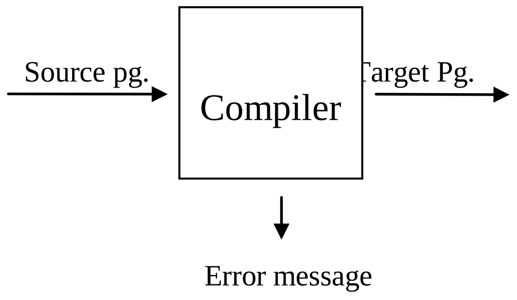


Fig. 1. Compiler structure

2. ANALYSIS SYNTHSIS MODEL OF COMPUTER

In the analysis synthesis model we done compilation in two phase. The two phases are

2.1 Analysis phase - It is the first phase of modal. In analysis phase an intermediate representation is formed from the given source code/program. In this phase information is collected from source code store that information into data structure symbol table. This phase is also called "front end" . Various part of this phase are

- Lexical analyzer
- Syntax analyzer
- Semantic analyzer

2.2 Synthesis phase - It is the second phase of model. In synthesis phase an intermediate representation get from analysis phase are converted into target program. This phase is also called "Back end". Various part of this phase are

- Intermediate code generator
- Code Generator
- Code optimizer

3. PHASES OF COMPILER

There are six phases of compiler. Each phase interacts with a symbol table Manager and error handler. Each phase uses an intermediate form of the program produced by an earlier phase.

3.1 LEXICAL ANALYZER

It is first phase of compiler. It is the phase the program break into lexemes. Each lexeme corresponds to atomic logical entity. For each lexeme the lexical analyzer produce output in the form of token.

Syntax-< Token_name, attribute_value >

- Token _name is an abstract symbol that is used during syntax analysis.
- Attribute_name points to an entry in symbol table for this token.

The information from symbol table is needed for semantic analysis and code generation

Consider the given statement in source code.

Alpha = Beta + bit * 50

This statement converted into lexeme and token is produced for each lexeme.

Table1. lexeme's correspond token

Lexeme	**Token**
Alpha	t1
=	op11
Beta	t2
+	op22
Bit	t3
*	op33
50	number1

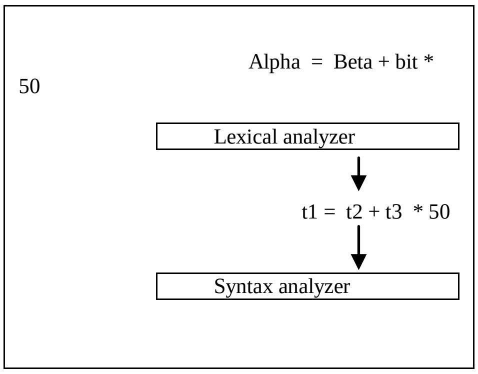


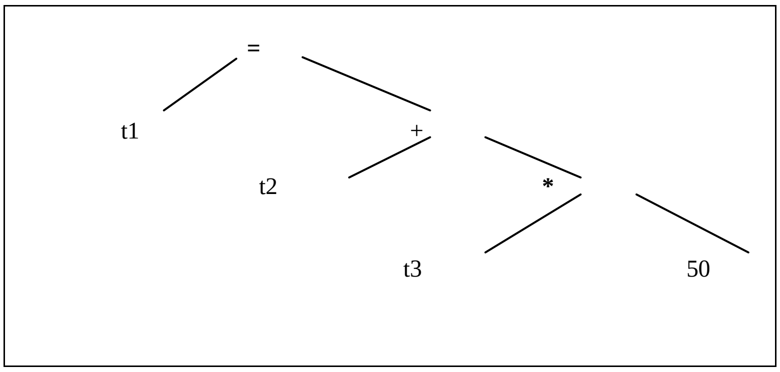
Fig..2. After lexical analyzer

3.2 SYNTAX ANALYZER

It is second phase of compiler. The input to this phase is the group of token produced by lexical analyzer. In this parser check that received tokens are correct order. It arrange group of token in grammatical phrase. It perform various task such as

- **Validate syntax-** Check whether group of token meets grammatical scheme.
- **Generate syntax tree-** Construct the syntax (parser) tree.
- **Syntax error is produced-** Message of error is shown if any.

Syntax Tree-The parse tree describe the syntactic structure of input. It also reflects the structure of program.



. Fig.3.Syntax tree

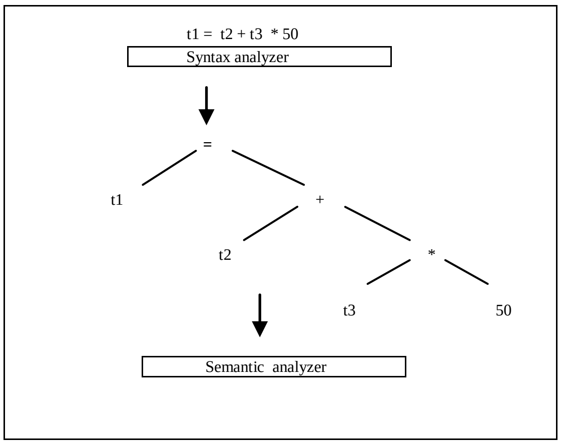


Fig..4. After syntax analyzer

3.3 SEMANTIC ANALYZER

It is the third phase of compiler. The input to this phase is syntax tree produce by syntax analyzer.

Function of semantic analyzer

Disambiguate overloaded operators- Specify the meaning of each overloaded operator if any.

Uniqueness checking- Name of the variable is unique.

Type checking - It is the process of checking type of variable. This may occur either at compile time or at run time.

Type coercion- It refers to type conversion. If operator applied to floating point number and integer then compiler may coercion the integer into floating point number. Example shows below.

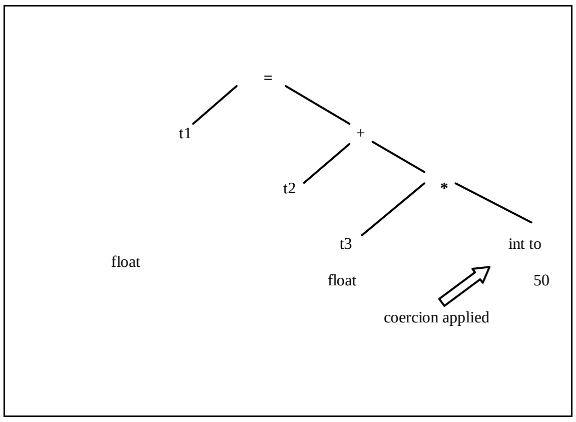


Fig.5. Coercion concept

3.4 INTERMEDIATE CODE GENERATOR

It is the fourth phase of compiler. In this source program is converted into machine independent intermediate language. In this the intermediate representation as a program for an abstract machine. It is the solution for avoiding the construction P x Q compilers. Where

P = numbers of source language

Q= number of object language

There are two properties for intermediate representation

- It is easy to produce.
- It is easy to easy to translate into target (machine) language.

Three address code

Str1 := int to float(50)

Str2 := t3 * Str1

Str3 := t2 + Str2

t1 := Str3

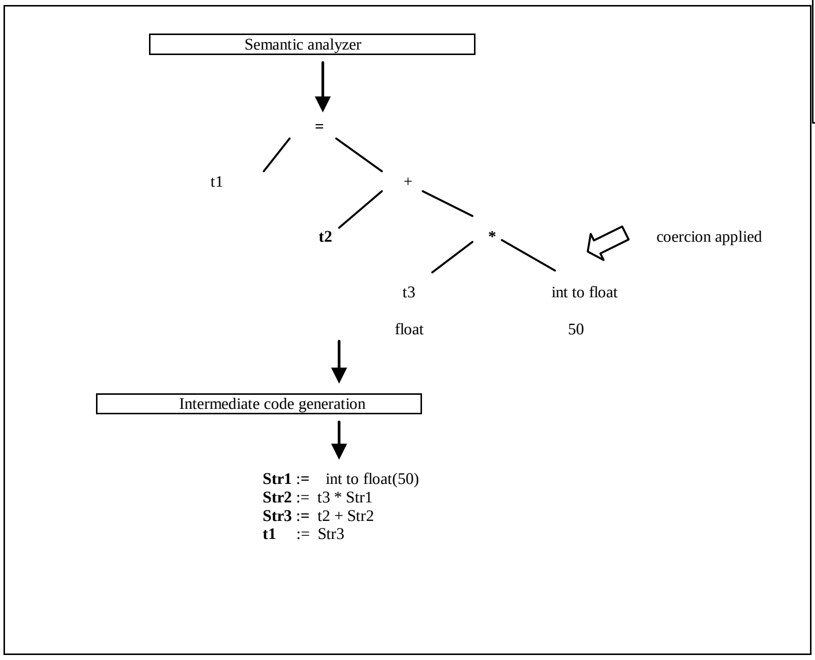


Fig.6. After Intermediate code generation

3.5 CODE OPTIMIZATION

It is the fifth phase of compiler. This phase is used for improvement in intermediate code received from intermediate code generation. Aim of this phase to produce better target code. The term better refers to less coding, easily running and less time consuming. Here conversion is take place of integer to floating point. It can be done once for all time compilation.

Apply code optimization

Str1 = t3 + 50.0
t1 = t2 + Str1

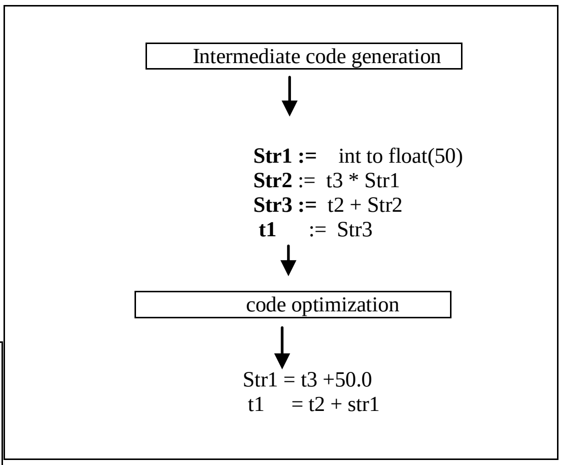
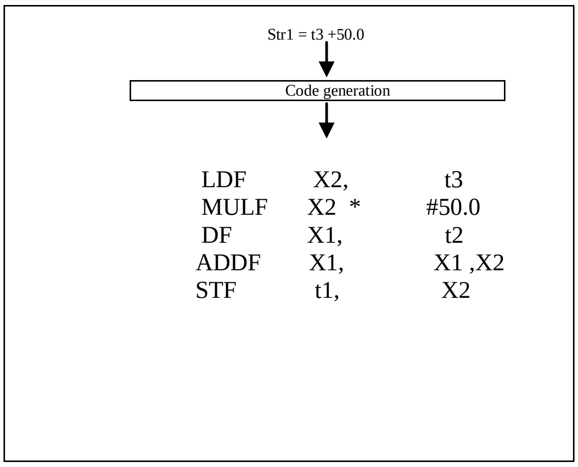


Fig.7. After code optimization

3.6 CODE GENERATION

It is the sixth phase of compiler. Its input is the intermediate code generated from previous phase. Then it produced the target program. The target program consist register address, memory locations and various machine code. Here we are using two registers named X1 and X2 and corresponding procedure followed.

```
LDF        X2,       t3
MULF       X2  *     #50.0
LDF        X1,       t2
ADDF       X1,       X1 ,X2
STF        t1,       X2
```



```
LDF      X2,       t3
MULF     X2  *     #50.0
DF       X1,       t2
ADDF     X1,       X1 ,X2
STF      t1,       X2
```

Fig.8. After code generation

4. BLOCK DIAGRAM OF COMPILER

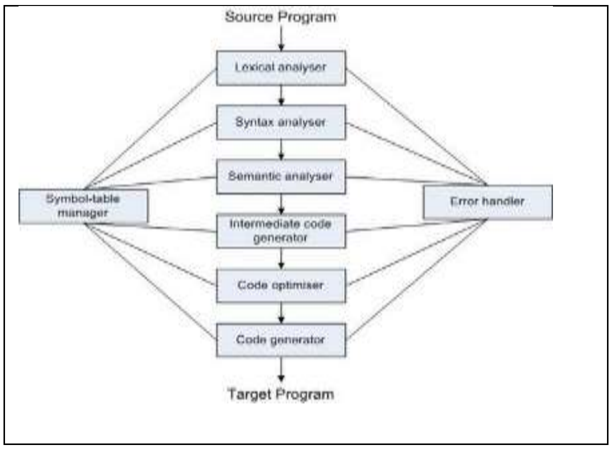


Fig.9. phases of compiler (ref. www.google.com)

5. ERROR HANDLING

The compiler perform a special function called error handling. The error handling include the detection and reporting of errors. When each phase of compiler is performing their working if any error exist their then error handler automatically called and report as an error and give the details of error. Such as it give the numbers of line at which error is happened and detail of error (what type of error is happened). So it interact with all phases of compiler.

6. NEED OF COMPILER

Various point clear the need of compiler are.

- Efficient for hardware.
- More user friendly.
- Platform independent
- Easy to avoid error
- Fill the semantic gap b/w higher level language and machine language.

7. CONCLUSION

The compiler externally seen as a single a single unit but internally it work's in different phases. Each phase has its own work. The output of one phase is passes to next phase. The detect the error in source program and make easier for user to correct the program. We need one time compilation. It become intermediate between higher level language and machine language.

8. REFRENCES

1. Dr. Matt Poole 2002, Compilers edited by Mr. Christopher Whyley, 2nd Semester 2006/2007.
2. “Compilers” http://www.info.univ-tours.fr/˜mirian/.
3. “Basics of Compiler Design” http://www.diku.dk/_torbenm/Basics