Programming Languages Design and Implementation

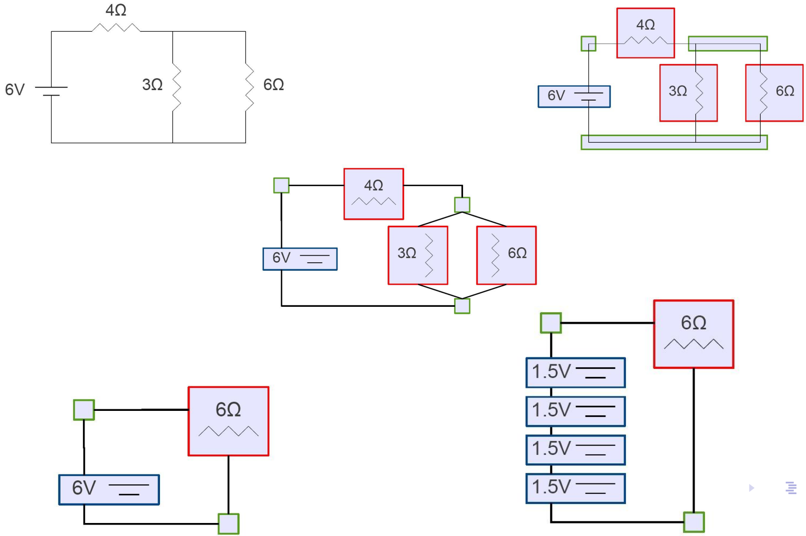
M. T. Bennani
Assistant Professor, FST - El Manar University, URAPOP-FST

-Academic Year 2020-2021-

Why Is Compilers Interesting?

- Understand how programming languages operate
- Learn how to develop programming languages
- Discover the basic concepts of the languages design
- Create ambitious software or libraries.

How Compilers Work



From the description to the implementation

- Lexical Analysis (Scanning): Identify the logical pieces of a given description.
- Syntactic Analysis (Parsing): Distinguish how the elements relate to each other.
- Semantic Analysis: Recognize the meaning of the entire structure.
- Generate the intermediate representation: Design a possible structure (composition).
- Optimize the intermediate representation: Simplify the produced composition.
- Code Generation: Generate a low layer program.
- Optimization: Improve the previous output.

1. Lexical Analysis

Source File

```
while (y   z){
  int x = a+b;
  y += x;
}
```

Output

```
T_While
T_ParLeft
T_Identifier y
T_Lessthan
T_ParLeft
T_Identifier z
T_ParRight
T_BracOpen
T_Int
T_Identifier x
T_Assign
T_Identifier a
T_Plus
T_Identifier b
T_Semicolon
T_Identifier y
T_PlusAssign
T_Identifier x
T_Semicolon
T_BracClose
```

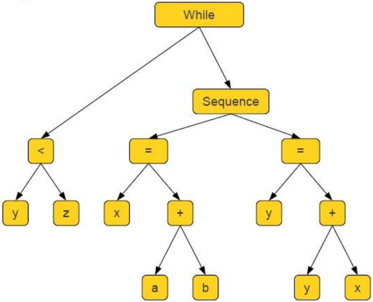
2. Syntactic Analysis

Input

The set of tokens generated by the lexical analyzer

Output

Abstract syntax tree



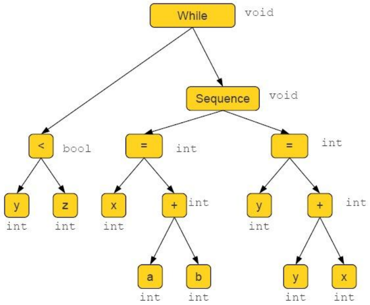
3. Semantic Analysis

Input

The abstract syntax tree generated by the syntactic analyzer

Output

Enhanced Abstract syntax tree



4. Generation of the intermediate representation

Input

The enhanced abstract syntax tree generated by the semantic analyzer

Output

Intermediate representation:

```
Loop : x = a + b
       y = x + y
       _t1 = y < z
       if _t1 goto Loop
```

5. Optimization of the intermediate representation

Input

The intermediate representation generated by the IR generator

Output

Optimized Intermediate representation:

```
       x = a + b
Loop : y = x + y
       _t1 = y < z
       if _t1 goto Loop
```

6. Low Level Code Generation

Input

The enhanced intermediate representation generated by the IR optimizer

Output

Low level code:

```
        add $1, $2, $3
Loop :  add $4, $1, $4
        slt $6, $1, $5
        beq $6, loop
```

slt: Set on less than. If $1 is less than $5, $6 is set to one. It gets zero otherwise.

beq: Branch on equal. Branches if the two registers are equal.

7. Code Optimization

Input

Low Level Code generated by the LLC Generator

Output

Optimized Low level code:

```
         add $1, $2, $3
  Loop:  add $4, $1, $4
         blt $1, $5, loop
```

blt: Branch on lower than. Branches if the two registers are equal.

References

- **Compilers: Principles, Techniques, and Tools** (Second Edition), Alfred Aho, Monica Lam, Ravi Sethi, and Jeffrey Ullman. Addison-Wesley, Published August 2006.
- **Modern Compilers Implementation in Java** (Second Edition), Andrew Appel and Jens Palsberg. Cambridge University Press, Published October 2002.
- **Engineering: A Compiler** (Second Edition), Keith Cooper and Linda Torczon. Morgan Kaufmann Editions, Published February 2011.