Complier Design
Assignment - Unit 1

Submitted By:

Habibur Rahman

17BCS071

Submitted To:

Mr. Sarfaraz Masood

B-Tech VIth Semester (Computer Science)

Department of Computer Engineering,

Faculty of Engineering and Technology,

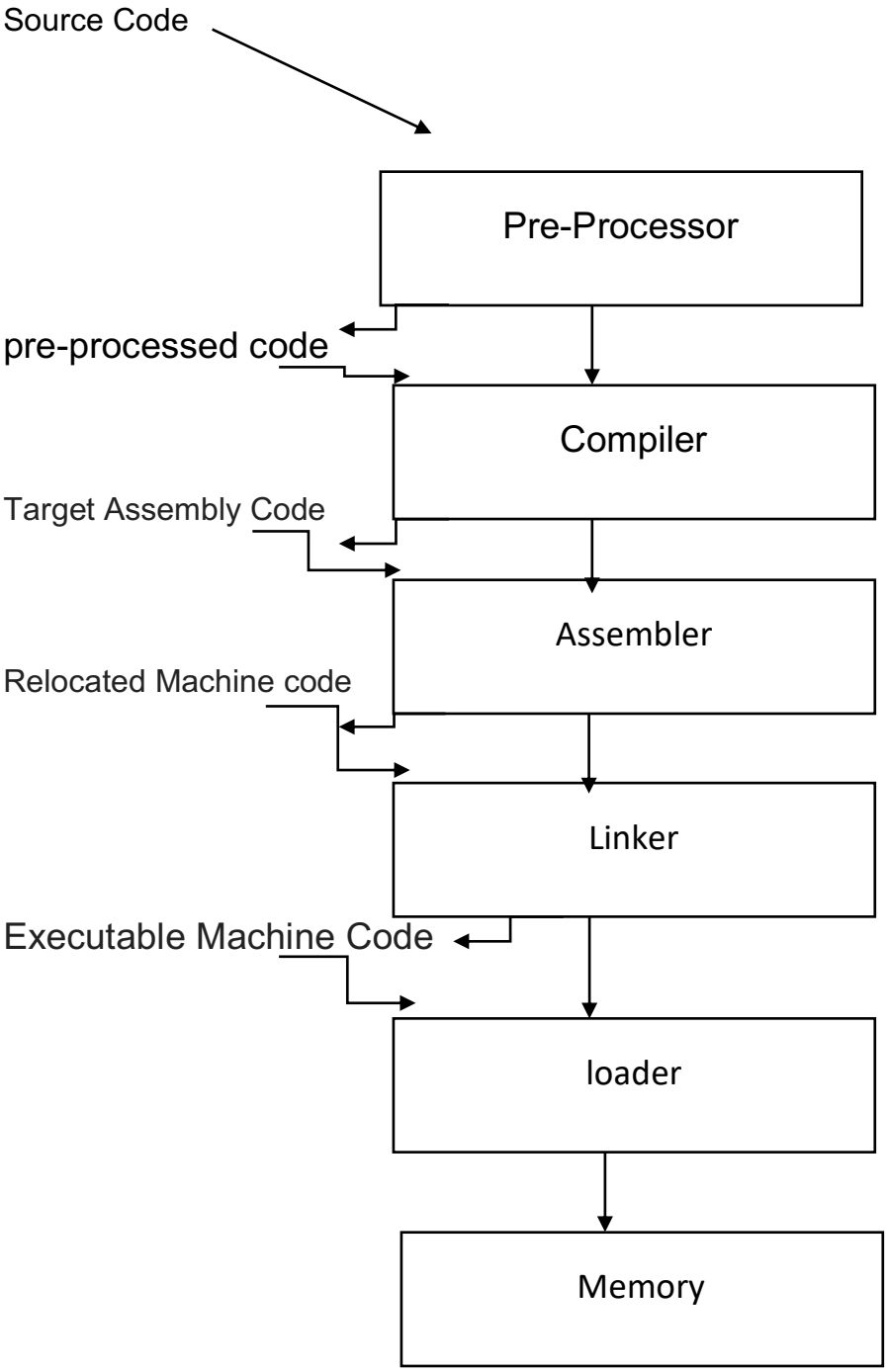
Jamia Millia Islamia ,

New Delhi

Session: 2019-20

Q1 a. Explain the Language processing System along with its components in details.

Ans.)



Steps for Language processing systems

Language Processing system consist of:

- Preprocessor: The preprocessor is considered as a part of the Compiler. It is a tool which produces input for Compiler. It deals with macro processing, augmentation, language extension, etc.
- Interpreter: An interpreter is like Compiler which translates high-level language into low-level machine language. The main difference between both is that interpreter reads and transforms code line by line. Compiler reads the entire code at once and creates the machine code.
- Assembler: It translates assembly language code into machine understandable language. The output result of assembler is known as an object file which is a combination of machine instruction as well as the data required to store these instructions in memory.
- Linker: The linker helps you to link and merge various object files to create an executable file. All these files might have been compiled with separate assemblers. The main task of a linker is to search for called modules in a program and to find out the memory location where all modules are stored.
- Loader: The loader is a part of the OS, which performs the tasks of loading executable files into memory and run them. It also calculates the size of a program which creates additional memory space.
- Cross-compiler: A cross compiler is a platform which helps you to generate executable code.
- Source-to-source Compiler: Source to source compiler is a term used when the source code of one programming language is translated into the source of another language.

Compiler Construction Tools

Compiler construction tools were introduced as computer-related technologies spread all over the world. They are also known as a compiler- compilers, compiler- generators or translator.

These tools use specific language or algorithm for specifying and implementing the component of the compiler.

- Scanner generators: This tool takes regular expressions as input. For example LEX for Unix Operating System.
- Syntax-directed translation engines: These software tools offer an intermediate code by using the parse tree. It has a goal of associating one or more translations with each node of the parse tree.
- Parser generators: A parser generator takes a grammar as input and automatically generates source code which can parse streams of characters with the help of a grammar.
- Automatic code generators: Takes intermediate code and converts them into Machine Language
- Data-flow engines: This tool is helpful for code optimization. Here, information is supplied by user and intermediate code is compared to analyze any relation. It is also known as data-flow analysis. It helps you to find out how values are transmitted from one part of the program to another part.

Q1b. Differentiate between Assembler, interpreter and Compiler using suitable examples.

Ans.)

Assembler –

The Assembler is used to translate the program written in Assembly language into machine code. The source program is a input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.

Example: GNU, GAS

Interpreter –

The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter. If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message. The interpreter moves on to the next line for execution only after removal of the error. An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code.
Example: Perl, Python and MATLAB.

Compiler –

The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a Compiler.
Example: C, C++, C#, Java

In a compiler, the source code is translated to object code successfully if it is free of errors. The compiler specifies the errors at the end of compilation with line numbers when there are any errors in the source code. The errors must be removed before the compiler can successfully recompile the source code again.

Q2 a. How is the Analysis part of the compiler different from its synthesis part Highlight using suitable example.

Ans.)

There are two parts to compilation: **analysis and synthesis**. The analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program. The synthesis part constructs the desired target program from the intermediate representation.

Analysis Phase: -

Lexical Analysis: - For example the assignment statement

position := initial + rate * 60

would be group into the following tokens: -

1. The identifier position.
2. The assignment symbol :=.
3. The identifier initial.
4. The plus sign.
5. The identifier rate.
6. The multiplication sign.
7. The number 60.

Syntax Analysis:- It involves grouping the tokens of the source program into grammatical phrases that are used by the compiler to synthesize output.

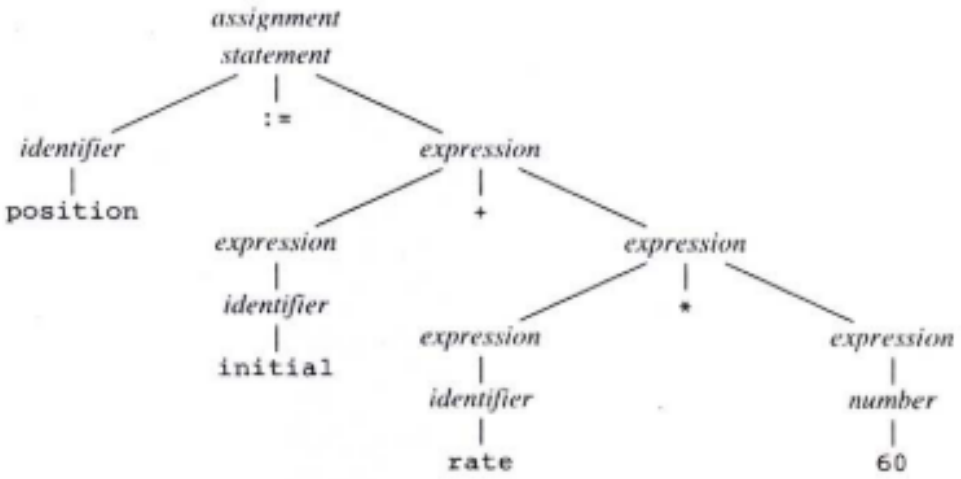
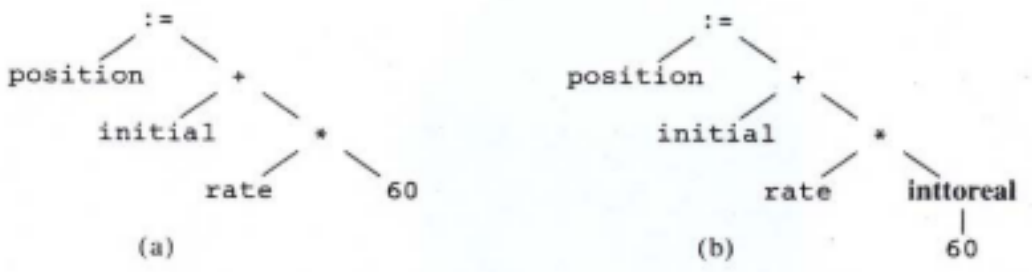


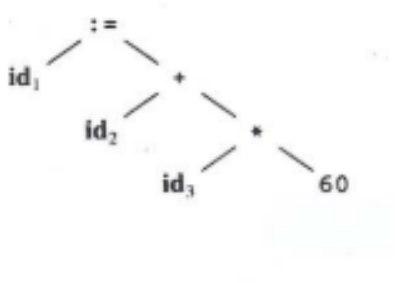
Fig. 1 Parse Tree for position := initial + rate * 60

Semantic Analysis:-

The semantic analysis phase checks the source program for semantic errors and gathers type information for the subsequent code-generation phase. It uses the hierarchical structure determined by the syntax-analysis phase to identify the operators and operands of expressions and statements.

Inside a machine, the bit pattern representing an integer is generally different from the bit pattern for a real, even if the integer and the real number happen to have the same value. Suppose, for example, that all identifiers in Figure 1 have been declared to be reals and that 60 by itself is assumed to be an integer Type checking of Figure (a) reveals that * is applied to a real, rate, and an integer, 60. The general approach is to convert the integer into a real. This has been achieved in Figure (b) by creating an extra node for the operator **int to real** that explicitly converts an integer into a real. Alternatively, since the operand of **int to real** is a constant, the compiler may instead replace the integer constant by an equivalent real constant.





Synthesis Phase: -

Synthesis phase is divided into three sub parts

1. Intermediate code generation
2. Code optimization
3. Code generation

Intermediate code generation: - The intermediate representation can have a variety of forms. We consider an intermediate form called "three-address code," which is like the assembly language for a machine in which every memory location can act like a register. Three-address code consists of a sequence of instructions, each of which has at most three operands. The source program in (1) might appear in three-address code as

temp1 := inttoreal(60)

temp2 := id3 * temp1

temp3 := id2 + temp2

id1 := temp3

Code Optimization: - The code optimization phase attempts to improve the intermediate code, so that faster-running machine code will result.

temp1 := id3 * 60.0

id1 := id2 + temp1

Code generation:- The final phase of the compiler is the generation of target code, consisting normally of relocatable machine code or assembly code. Memory locations are selected for each of the variables used by the program. Then, intermediate instructions are each translated into a sequence of machine instructions that perform the same task. A crucial aspect is the assignment of variables to registers.

MOVF id3, R2

MULF #60.0, R2

MOVF id2, R1

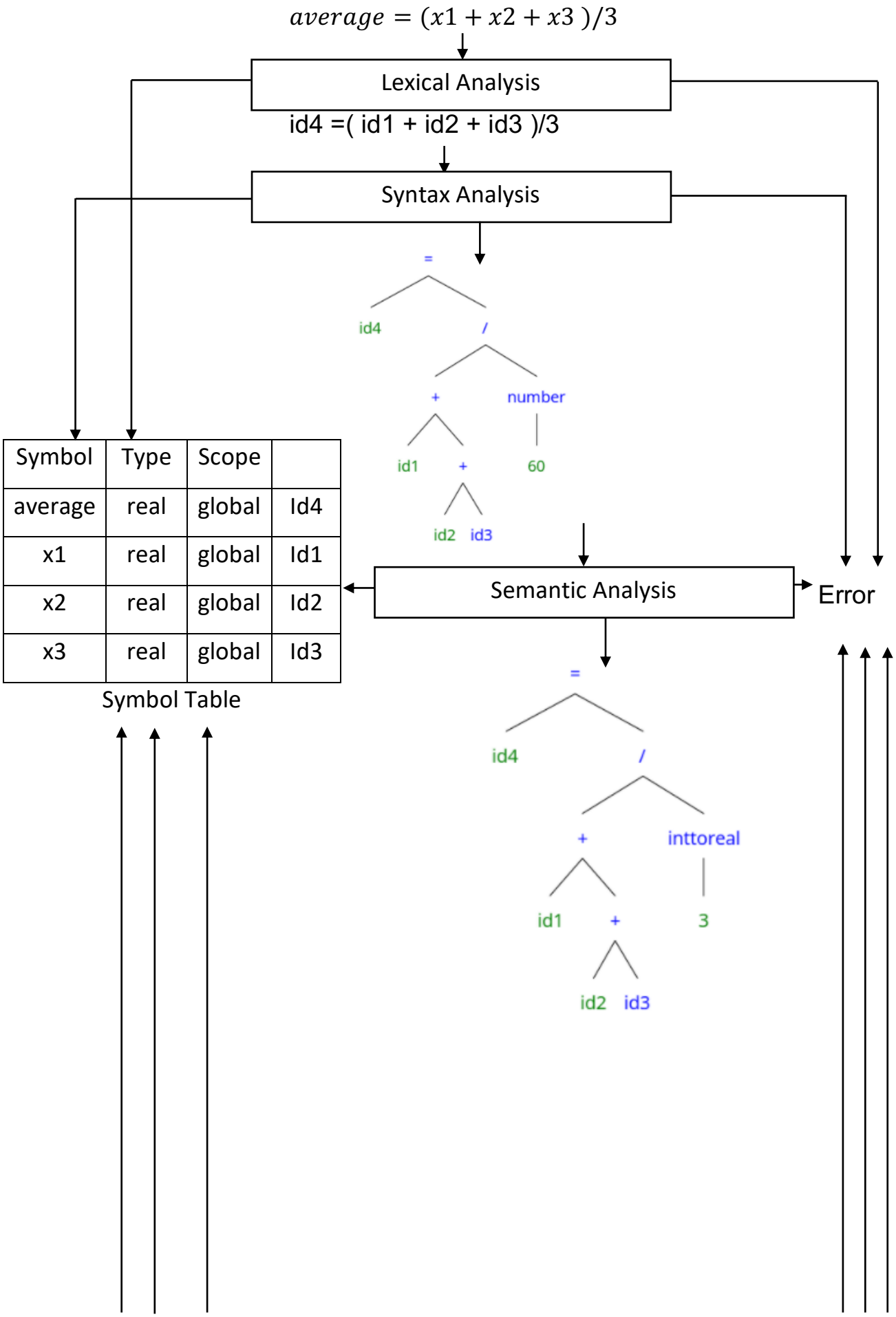
ADDF R2, R1

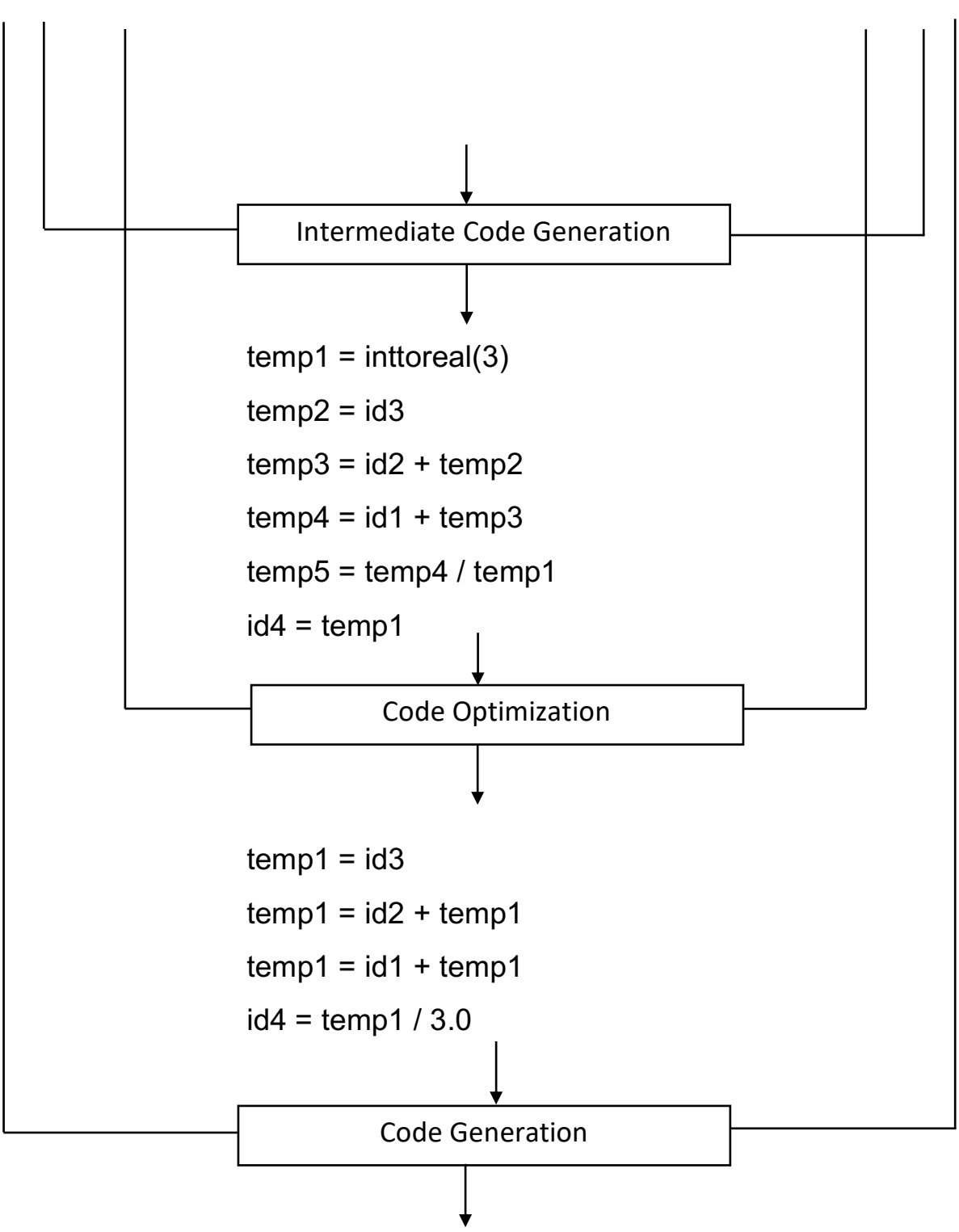
MOVE R1, id1

Q2 b. Trace the steps followed in the compiler during the conversion of the high level statement (given below) to the corresponding Assembly code .

$$average = (x1 + x2 + x3\ )/3$$

Solution


$average = (x1 + x2 + x3\ )/3$
Lexical Analysis
id4 =(id1 + id2 + id3)/3
Syntax Analysis
=
id4
/
+
number
id1
+
60
id2
id3
Symbol
Type
Scope
average
real
global
Id4
x1
real
global
Id1
x2
real
global
Id2
x3
real
global
Id3
Symbol Table
Semantic Analysis
Error
=
id4
/
+
inttoreal
id1
+
3
id2
id3


Intermediate Code Generation
temp1 = inttoreal(3)
temp2 = id3
temp3 = id2 + temp2
temp4 = id1 + temp3
temp5 = temp4 / temp1
id4 = temp1
Code Optimization
temp1 = id3
temp1 = id2 + temp1
temp1 = id1 + temp1
id4 = temp1 / 3.0
Code Generation
MOVF id3 R1
MOVF id2 R0
ADDF R0 R1
MOVF id1 R1
ADDF R0 R1
DIVF R0 #3.0
MOVF R0 id4