SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Engineering

Name: _ Date of Performance: _ _ /_ _/20_ _
Class: B.E. Computer Date of Completion: _ _ /_ _/20_ _
Division : Grade:
Batch:
Roll No: Sign. of Teacher with Date:
Subject: Advanced Computer Network Lab

Experiment No. 5

Aim: Implementation of Code Generator

1. Objective: To understand and simulate code generation phase of compiler.

2. **Background:**

Code generation can be considered as the final phase of compilation. Through post code generation, optimization process can be applied on the code, but that can be seen as a part of code generation phase itself. The code generated by the compiler is an object code of some lower-level programming language, for example, assembly language. We have seen that the source code written in a higher-level language is transformed into a lower-level language that results in a lower-level object code, which should have the following minimum properties:

It should carry the exact meaning of the source code.

It should be efficient in terms of CPU usage and memory management

Code generator is expected to have an understanding of the target machine's runtime environment and its instruction set. The code generator should take the following things into consideration to generate the code:

- **Target language** : The code generator has to be aware of the nature of the target language for which the code is to be transformed. That language may facilitate some machine-specific instructions to help the compiler generate the code in a more convenient way. The target machine can have either CISC or RISC processor architecture.
- **IR Type** : Intermediate representation has various forms. It can be in Abstract Syntax Tree (AST) structure, Reverse Polish Notation, or 3-address code.
- **Selection of instruction** : The code generator takes Intermediate Representation as input and converts (maps) it into target machine's instruction set. One representation can have many ways (instructions) to convert it, so it becomes the responsibility of the code generator to choose the appropriate instructions wisely.

- **Register allocation** : A program has a number of values to be maintained during the execution. The target machine's architecture may not allow all of the values to be kept in the CPU memory or registers. Code generator decides what values to keep in the registers. Also, it decides the registers to be used to keep these values.
- **Ordering of instructions** : At last, the code generator decides the order in which the instruction will be executed. It creates schedules for instructions to execute them.

3. Pre-lab Task:

In addition to the basic conversion from an intermediate representation into a linear sequence of machine instructions, a typical code generator tries to optimize the generated code in some way.

Tasks which are typically part of a sophisticated compiler's "code generation" phase include:

- Instruction selection: which instructions to use.
- Instruction scheduling: in which order to put those instructions. Scheduling is a speed optimization that can have a critical effect on pipelined machines.
- Register allocation: the allocation of variables to processor registers
- Debug data generation if required so the code can be debugged.

Instruction selection is typically carried out by doing a recursive postorder traversal on the abstract syntax tree, matching particular tree configurations against templates; for example, the tree W := ADD(X,MUL(Y,Z)) might be transformed into a linear sequence of instructions by recursively generating the sequences for t1 := X and t2 := MUL(Y,Z), and then emitting the instruction ADD W, t1, t2.

In a compiler that uses an intermediate language, there may be two instruction selection stages — one to convert the parse tree into intermediate code, and a second phase much later to convert the intermediate code into instructions from the instruction set of the target machine. This second phase does not require a tree traversal; it can be done linearly, and typically involves a simple replacement of intermediate-language operations with their corresponding opcodes. However, if the compiler is actually a language translator (for example, one that converts Eiffel to C), then the second code-generation phase may involve *building* a tree from the linear intermediate code.

4. In-lab Task:

Lex program Template :

```
%{
   - - - - - - - - - - - -
    - - - - - - - - - - - -

%}

%%
 - - - - - - - - - - - -|
 - - - - - - - - - - - -  { strcpy(yylval.vname,yytext);
                            return NAME;
                          }
 - - - - - - - - - - - -     ;
.|\n              { return yytext[0]; }
%%
```

Yacc Program Template:

```
%{

- - - - - - - - - - - - -
- - - - - - - - - - - - -
- - - - - - - - - - - - -
%}

%union
{
      - - - - - - - - - - - - -
}

%left '+' '-'
%left '*' '/'

%token <vname> NAME
%type <vname> expression

%%

input   : line '\n' input
        | '\n' input
        | ;


line    : NAME '=' expression  {
                        fprintf(fpOut,"MOV %s, AX\n",$1);
                              }
       ;


expression: - - - - - - - - - - - - -     {
                        fprintf(--------------------------);
                        fprintf(--------------------------);
                              }

             | - - - - - - - - - - - - -


             | - - - - - - - - - - - - -         {

             | NAME '/' NAME         {
                        fprintf(fpOut,"MOV AX, %s\n",$1);
                        fprintf(-----------------------------------);
                              }
             | NAME                  {
                        fprintf(fpOut,"MOV AX, %s\n",$1);
                        strcp$$, $1y();
                              }
            ;

%%
```

```
FILE *yyin;
main()
{
	FILE *fpInput;
	fpInput = fopen("----------,”r");
	 fpOut = fopen("----------","w");
	yyin = fpInput;
	yyparse();
	fcloseall();
}
```

5. Post-lab Task:

Outcomes:

. . . . . .
. . . . . .
. . . . . .
. . . . . .

Questions:

Q.1. Explain design issues is code generator.

Q.2. Discuss Code generation phase.