

Government College of Engineering, Jalgaon

(An Autonomous Institute of Govt. of Maharashtra)

Name of Student:.....

PRN:.....

Course Teacher: Sharayu Bonde

Date of Performance:

Date of completion:

Experiment No.: Group B: 03

Title: Write a C program to generate machine code from abstract syntax tree generated by the parser

Theory:

Code generator converts the intermediate representation of source code into a form that can be readily executed by the machine. A code generator is expected to generate a correct code. Designing of code generator should be done in such a way so that it can be easily implemented, tested and maintained. The following issue arises during the code generation phase:

1. Input to code generator –

The input to code generator is the intermediate code generated by the front end, along with information in the symbol table that determines the run-time addresses of the data-objects denoted by the names in the intermediate representation. Intermediate codes may be represented mostly in quadruples, triples, indirect triples, Postfix notation, syntax trees, DAG's etc. Assume that they are free from all of syntactic and state semantic errors, the necessary type checking has taken place and the type-conversion operators have been inserted wherever necessary.

2. Target program –

Target program is the output of the code generator. The output may be absolute machine language, relocatable machine language, assembly language. Absolute machine language as an output has advantages that it can be placed in a fixed memory location and can be immediately executed.

Relocatable machine language as an output allows subprograms and subroutines to be compiled separately. Relocatable object modules can be linked together and loaded by linking loader.

Assembly language as an output makes the code generation easier. We can generate symbolic instructions and use macro-facilities of assembler in generating code.

3. Memory Management –

Mapping the names in the source program to addresses of data objects is done by the front end and the code generator. A name in the three address statement refers to the symbol table entry for name. Then from the symbol table entry, a relative address can be determined for the name.

4. Instruction selection –

Selecting best instructions will improve the efficiency of the program. It includes the instructions that should be complete and uniform. Instruction speeds and machine idioms also plays a major role when efficiency is considered. But if we do not care about the efficiency of the target program then instruction selection is straight-forward.

For example, the respective three-address statements would be translated into latter code sequence as shown below:

P:=Q+R

S:=P+T

MOV Q, R0

ADD R, R0

MOV R0, P

```

MOV P, R0
ADD T, R0
MOV R0, S

```

Here the fourth statement is redundant as the value of the P is loaded again in that statement that just has been stored in the previous statement. It leads to an inefficient code sequence. A given intermediate representation can be translated into many code sequences, with significant cost differences between the different implementations. A prior knowledge of instruction cost is needed in order to design good sequences, but accurate cost information is difficult to predict.

5. Register allocation issues –

Use of registers make the computations faster in comparison to that of memory, so efficient utilization of registers is important. The use of registers are subdivided into two subproblems: During Register allocation – we select only those set of variables that will reside in the registers at each point in the program.

During a subsequent Register assignment phase, the specific register is picked to access the variable. As the number of variables increase, the optimal assignment of registers to variables becomes difficult. Mathematically, this problem becomes NP-complete. Certain machine requires register pairs consist of an even and next odd-numbered register. For example

M a, b

These types of multiplicative instruction involve register pairs where a, the multiplicand is an even register and b, the multiplier is the odd register of the even/odd register pair.

6. Evaluation order –

The code generator decides the order in which the instruction will be executed. The order of computations affects the efficiency of the target code. Among many computational orders, some will require only fewer registers to hold the intermediate results. However, picking the best order in general case is a difficult NP-complete problem.

7. Approaches to code generation issues:

Code generator must always generate the correct code. It is essential because of the number of special cases that a code generator might face. Some of the design goals of code generator are:

- Correct
- Easily maintainable
- Testable
- Maintainable

Sharayu Nivrutti Bonde
Course Teacher