Assignment no.B5
Roll no.4202

1 Title

Code generation using DAG / labeled tree

2 Problem Definition

Code generation using DAG / labeled tree

3 Objective

1. 1. To understand how code is generated
2. 2. To learn how to implement code generation using DAG

4 Pre-requisite

1. 1. Basic knowledge of tree data structure
2. 2. Knowledge of phases of compilation

5 Software and Hardware Requirements

1. 1. 64 bit open source Linux OS - Fedora 20
2. 2. Eclipse IDE installed on machine

6 Mathematical Model :

Let S be the solution perspective of the system such that,
S =$\{S_t, E_t, I, O, DD, NDD, F_{me}, Sc, Fc\}$
where,

S_t = Start state where message to be encoded is read.

I represents set of input
I = {operators,operands,stack}
operators = +,-,*,operands = [a-z][A-Z]
O represents assembly level code
O = code

F_{me}=set of functions.
F_{me}={f1,f2,f3}
where,
f1= f1 represents function to generate tree
operator = parent node parentnode label = max(lchild,rchild)
f2= f2 is the function to calculate frequency of words
frequency={f—f ∈ I+}
f3= f3 is the function to swap top two registers

f4= f4 is the function to generate assembly code

DD (Deterministic Data) = I = Operators , operands

NDD (Non Deterministic Data) = Assembly level code

Sc = Code generated performs fuctions of input expression

Fc = Incorrect code is generated.

E_t=displaycode.

Theory

The DAG Representation of Basic Blocks

We construct a DAG for a basic blockas follows:

1. There is a node in the DAG for each of the initial values of the variables appearing in the basic block.
2. There is a node N associated with each statement s within the block. The children of N are those nodes corresponding to statements that are the last definitions, prior to s, of the operands used by s.
3. Node N is labeled by the operator applied at s, and also attached to N is the list of variables for which it is the last definition within the block.
4. Certain nodes are designated output nodes. These are the nodes whose variables are live on exit from the block; that is, their values may be used later, in another block of the flow graph.

The DAG representation of a basic block lets us perform several codeimproving transformations on the code represented by the block.

1. We can eliminate local common subexpressions, that is, instructions that compute a value that has already been computed.
2. We can eliminate dead code, that is, instructions that compute a value that is never used.
3. We can reorder statements that do not depend on one another; such reordering may reduce the time a temporary value needs to be preserved in a register.
4. We can apply algebraic laws to reorder operands of three-address instructions, and sometimes thereby simplify the computation.

Machine Instructions for Operations

For a three-address instruction such as x = y + z, do the following:

1. Use getReg(x = y + z) to select registers for x, y, and z. Call these Rx, Ry, and Rz.
2. If y is not in Ry (according to the register descriptor for Ry), then issue an instruction LD Ry, y', where y' is one of the memory locations for y (according to the address descriptor for y).
3. Similarly, if z is not in Rz, issue and instruction LD Rz, Zl, where Zl is a location for z.
4. Issue the instruction ADD Rx, Ry, Rz·

7 Conclusion

We have thus understood how code can be generated using DAG/labelled tree.