Assignment-02

Q.1 Symbol table management and Data Structure used for symbol table.

Ans.

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler.

Implementation

If a compiler is to handle a small amount of data, then the symbol table can be implemented as an unordered list, which is easy to code, but it is only suitable for small tables only. A symbol table can be implemented in one of the following ways:

- Linear (sorted or unsorted) list
- Binary Search Tree
- Hash table

Among all, symbol tables are mostly implemented as hash tables, where the source code symbol itself is treated as a key for the hash function and the return value is the information about the symbol.

Operations

A symbol table, either linear or hash, should provide the following operations.

<u>insert():</u>

This operation is more frequently used in the analysis phase, i.e., the first half of the compiler where tokens are identified and names are stored in the table. This operation is used to add information in the symbol table about unique names occurring in the source code. The format or structure in which the names are stored depends upon the compiler in hand.

<u>lookup():</u>

lookup() operation is used to search a name in the symbol table to determine:

- if the symbol exists in the table.
- if it is declared before it is being used.
- if the name is used in the scope.
- if the symbol is initialized.
- if the symbol is declared multiple times.

Q.2 Principal source of optimization.

Ans.

Optimization is a program transformation technique, which tries to improve the code by making it consume less resources (i.e. CPU, Memory) and deliver high speed.

In optimization, high-level general programming constructs are replaced by very efficient low-level programming codes. A code optimizing process must follow the three rules given below:

- The output code must not, in any way, change the meaning of the program.
- Optimization should increase the speed of the program and if possible, the program should demand less number of resources.
- Optimization should itself be fast and should not delay the overall compiling process.

Efforts for an optimized code can be made at various levels of compiling the process.

- At the beginning, users can change/rearrange the code or use better algorithms to write the code.
- After generating intermediate code, the compiler can modify the intermediate code by address calculations and improving loops.
- While producing the target machine code, the compiler can make use of memory hierarchy and CPU registers.

Optimization can be categorized broadly into two types : machine independent and machine dependent.

Machine-dependent Optimization

Machine-dependent optimization is done after the target code has been generated and when the code is transformed according to the target machine architecture. It involves CPU registers and may have absolute memory references rather than relative references. Machine-dependent optimizers put efforts to take maximum advantage of memory hierarchy.

Dead-code Elimination

Dead code is one or more than one code statements, which are:

- Either never executed or unreachable,
- Or if executed, their output is never used.

Thus, dead code plays no role in any program operation and therefore it can simply be eliminated.

Partial Redundancy

Redundant expressions are computed more than once in parallel path, without any change in operands.whereas partial-redundant expressions are computed more than once in a path, without any change in operands.

Q.3 Data flow analysis .

Ans.

It is the analysis of flow of data in control flow graph, i.e., the analysis that determines the information regarding the definition and use of data in a program. With the help of this analysis optimization can be done. In general, its process in which values are computed using data flow analysis.The data flow property represents information which can be used for optimization.

Basic Terminologies :–

- Definition Point: a point in a program containing some definition.
- Reference Point: a point in a program containing a reference to a data item.
- Evaluation Point: a point in a program containing evaluation of expression.

Data Flow Properties :–

- Available Expression – A expression is said to be available at a program point x iff along paths its reaching to x. A Expression is available at its evaluation point.
  A expression a+b is said to be available if none of the operands gets modified before their use.

Reaching Definition :– A definition D is reaches a point x if there is path from D to x in which D is not killed, i.e., not redefined.

Live variable :– A variable is said to be live at some point p if from p to end the variable is used before it is redefined else it becomes dead.

Busy Expression :– An expression is busy along a path iff its evaluation exists along that path and none of its operand definition exists before its evaluation along the path.

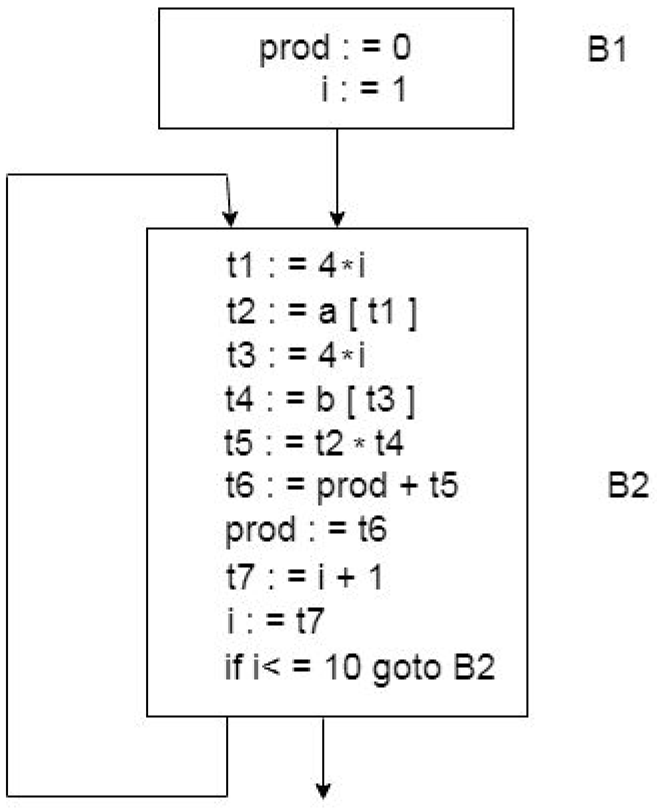
Q.4 What is basic blocks and flow graphs? Also represent basic block with the help of DAG.

Ans. Basic Block is a straight line code sequence which has no branches in and out branches except to the entry and at the end respectively. Basic Block is a set of statements which always executes one after another, in a sequence.
The first task is to partition a sequence of three-address code into basic blocks. A new basic block is begun with the first instruction and instructions are added until a jump or a label is met. In the absence of jump control moves further consecutively from one instruction to another.
Flow Graph:

Flow graph is a directed graph. It contains the flow of control information for the set of basic block.

A control flow graph is used to depict that how the program control is being parsed among the blocks. It is useful in the loop optimization.

Flow graph for the vector dot product is given as follows:



- Block B1 is the initial node. Block B2 immediately follows B1, so from B2 to B1 there is an edge.
- The target of jump from last statement of B1 is the first statement B2, so from B1 to B2 there is an edge.
- B2 is a successor of B1 and B1 is the predecessor of B2.

Q.5 What is peephole Optimization.

Ans.

Definition:-

Peephole optimization is a simple and effective technique for locally improving target code. This technique is applied to improve the performance of the target program by examining the short sequence of target instructions (called the peephole) and replacing these instructions with shorter or faster sequences whenever possible. Peephole is a small, moving window on the target program.

Characteristics of Peephole Optimization so that peephole optimization can be applied to the target code using the following characteristic:

1. Redundant instruction elimination

• Especially the redundant loads and stores can be eliminated in this type of transformations. Example: MOV R0,x MOV x,R0 • We can eliminate the second instruction since x is in already R0. But if MOV x, R0 is a label statement then we can not remove it.

2. Unreachable code

• Especially the redundant loads and stores can be eliminated in this type of transformations.

• An unlabeled instruction immediately following an unconditional jump may be removed.

• This operation can be repeated to eliminate the sequence of instructions.

Example:

define debug 0

If(debug) { Print debugging information }

-> In the intermediate representation the if statement may be translated as if-

If debug=1 goto L1

goto L2 L1:

print debugging information L2:

• Additionally, One obvious peephole optimization is to eliminate jumps over jumps. Thus no matter what the value of debugging, can replaced by:

If debug goto L2

debug≠1 Print debugging information L2:

• Now, since debug set to 0 at the beginning of the program, constant propagation program, should replace by

If 0≠1 goto L2≠1 Print debugging information L2:

• As the argument of the first statement evaluates to a constant true, it can replace by goto L2.

• Then all the statements that print debugging aids are manifestly unreachable and can manifestly eliminate one at a time.

3. The flow of control optimization

• The unnecessary jumps can be eliminated in either the intermediate code or the target code by the following types of peephole optimizations.

• We can replace the jump sequence.

Goto L1 …… L1:

goto L2 By the sequence

Goto L2 ……. L1:

goto L2

• If there are no jumps to L1 then it may be possible to eliminate the statement L1: goto L2 provided it is preceded by an unconditional jump. Similarly, the sequence If a

4. Algebraic simplification

• So Peephole optimization is an effective technique for algebraic simplification.

• The statements such as x = x + 0 or x := x* 1 can be eliminated by peephole optimization.

5. Reduction in strength

- Certain machine instructions are cheaper than the other.
- In order to improve the performance of the intermediate code, we can replace these instructions by equivalent cheaper instruction.
- So For example, x2 is cheaper than x * x. Similarly, addition and subtraction are cheaper than multiplication and division. So we can add an effectively equivalent addition and subtraction for multiplication and division.

6. Machine idioms

- So The target instructions have equivalent machine instructions for performing some operations.
- Hence we can replace these target instructions by equivalent machine instructions in order to improve the efficiency.
- Example: Some machines have auto-increment or auto-decrement addressing modes.decrement These modes can be used in a code for a statement like i=i+1.