Assignment No.14

AIM:

Write a program to implement code optimization for a given block of three address codes.

THEORY:

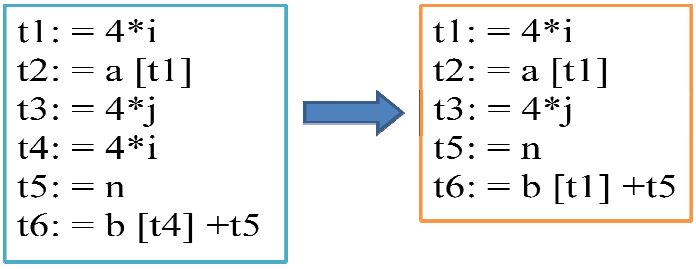
A transformation of a program is called local if it can be performed by looking only at the statements in a basic block; otherwise, it is called global.
Many transformations can be performed at both the local and global levels. Local transformations are usually performed first.

- Function-Preserving Transformations
  - Common Sub expressions elimination
  - Copy Propagation
  - Dead-Code Eliminations
  - Constant folding
- Loop Optimizations
  - Code Motion
  - Induction Variables
  - Reduction In Strength

There are a number of ways in which a compiler can improve a program without changing the function it computes.
The following transformations are common examples of such function-preserving transformations. The other transformations come up primarily when global optimizations are performed.

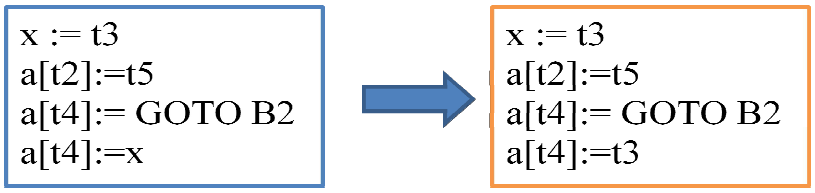
- Common Sub expressions elimination - An occurrence of an expression E is called a common sub-expression if
  - E was previously computed, and
  - The values of variables in E have not changed since the previous computation.

We can avoid recomputing the expression if we can use the previously computed value.

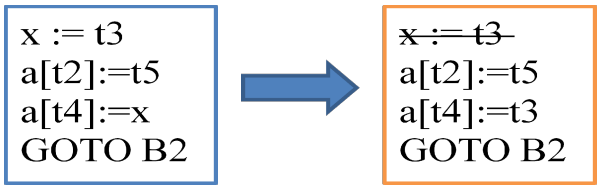


- Copy Propagation - Assignments of the form f : = g called copy statements, or copies for short. The idea behind the copy-propagation transformation is to use g for f, whenever possible after the copy statement f: = g. Copy propagation means use of one variable instead of another.

  Example



- Dead-Code Eliminations - A variable is live at a point in a program if its value can be used subsequently; otherwise, it is dead at that point. A related idea is dead or useless code, statements that compute values that never get used. While the programmer is unlikely to introduce any dead code intentionally, it may appear as the result of previous transformations. An optimization can be done by eliminating dead code.



- Constant folding - We can eliminate both the test and printing from the object code. More generally, deducing at compile time that the value of an

expression is a constant and using the constant instead is known as constant folding. One advantage of copy propagation is that it often turns the copy statement into dead code. For example,
a=3.14157/2 can be replaced by
a=1.570 there by eliminating a division operation.

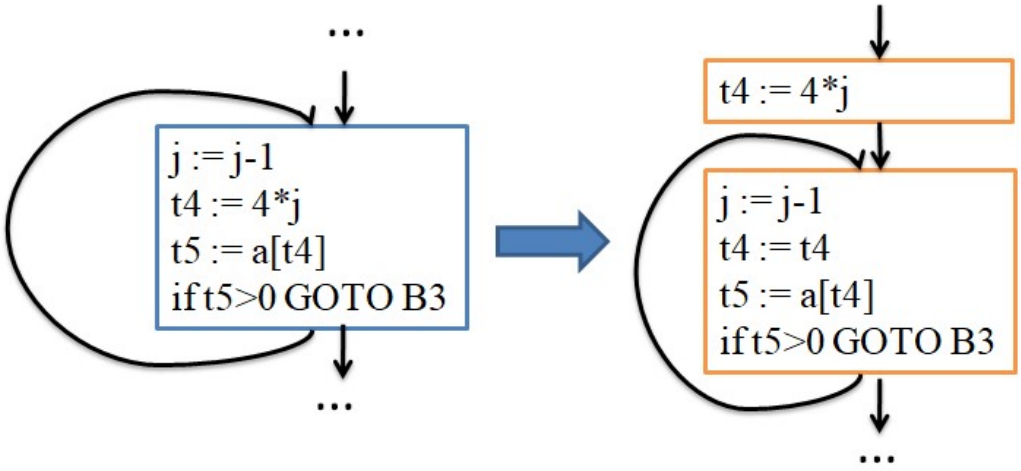
We now give a brief introduction to a very important place for optimizations, namely loops, especially the inner loops where programs tend to spend the bulk of their time. The running time of a program may be improved if we decrease the number of instructions in an inner loop, even if we increase the amount of code outside that loop. Three techniques are important for loop optimization:

- Code Motion - An important modification that decreases the amount of code in a loop is code motion. This transformation takes an expression that yields the same result independent of the number of times a loop is executed and places the expression before the loop. The notion "before the loop" assumes the existence of an entry for the loop. For example, evaluation of limit-2 is a loop-invariant computation in the following while-statement: Code motion will result in the equivalent of

```
while (i <= limit-2)
/* statement does not change limit*/
```

```
t= limit-2;
while (i<=t)
/* statement does not change limit or t */
```

- Induction Variables - Loops are usually processed inside out. When there are two or more induction variables in a loop, it may be possible to get rid of all but one, by the process of induction-variable elimination.



- Reduction In Strength - Reduction in strength replaces expensive operations by equivalent cheaper ones on the target machine. Certain machine instructions are considerably cheaper than others and can often be used as special cases of more expensive operators. For example, x^2 is invariably cheaper to implement as x*x than as a call to an exponentiation routine. Fixed-point multiplication or division by a power of two is cheaper to implement as a shift. Floating-point division by a constant can be implemented as multiplication by a constant, which may be cheaper.

INPUT:

a : = b + c
b : = a – d
c : = b + c
d : = a – d

OUTPUT:

a : = b + c
b : = a – d
c : = b + c
d : = b

CONCLUSION:

- There are a number of ways in which a compiler can improve a program without changing the function it computes.
- Code optimization technique – common subexpression elimination is implemented.

REFERENCES:

- Compilers - Principles, Techniques and Tools - A.V. Aho, R. Shethi and J. D. Ullman (Pearson Education)