Unit 5

Question bank

Q.1. What are the properties of code generation phase?

Ans: Code generation phase should have the following minimum properties:

(i) It should carry the exact meaning of the source code.

(ii) It should be efficient in terms of CPU usage and memory management.

Q.2. What are basic blocks?

Ans: Basic block contains a sequence of statement. The flow of control enters at the beginning of the statement and leave at the end without any halt (except may be the last instruction of the block).

Q.3. Discuss the peephole optimization?

Ans: Peephole optimization is a type of Code Optimization performed on a small part of the code. It is performed on the very small set of instructions in a segment of code.
The small set of instructions or small part of code on which peephole optimization is performed is known as peephole or window.

It basically works on the theory of replacement in which a part of code is replaced by shorter and faster code without change in output.

Q.4. Explain the peephole optimization objectives ?

Ans:
The objective of peephole optimization is:

a. To improve performance
b. To reduce memory footprint
c. To reduce code size

Q.5. What is code 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.

Q.6. Explain machine dependent and independent code optimization?

Ans: **Machine-independent Optimization:**

In this optimization, the compiler takes in the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations. For example:

```
do
{
  item = 10;
  value = value + item;
} while(value<100);
```

This code involves repeated assignment of the identifier item, which if we put this way:

```
Item = 10;
do
{
  value = value + item;
} while(value<100);
```

should not only save the CPU cycles, but can be used on any processor.

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. For example Peephole optimization is the machine dependent optimization.

Q.7.. What is common sub-expression and how to eliminate it? Explain with example.

Ans: Common sub-expression (CS) is instances of identical expressions during code optimization. Common sub-expression elimination (CSE) is a compiler optimization that searches for instances of identical expressions (i.e., they all evaluate to the same value), and analyses whether it is worthwhile replacing them with a single variable holding the computed value.

In the common sub-expression, we don't need to be computed it over and over again. Instead of this we can compute it once and kept in store from where it's referenced when encountered again.

a : = b + c
b : = a - d
c : = b + c

d : = a - d

In the above expression, the second and forth expression computed the same expression. So the block can be transformed as follows:

a : = b + c
b : = a - d
c : = b + c
d : = b

Q.8. What is 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 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.

Q.9. Write the algorithm for partitioning into Blocks.

Ans: **Algorithm:** Partition into basic blocks

Input: It contains the sequence of three address statements

Output: it contains a list of basic blocks with each three address statement in exactly one block

Method: First identify the leader in the code. The rules for finding leaders are as follows:

- The first statement is a leader.
- Statement L is a leader if there is an conditional or unconditional goto statement like: if....goto L or goto L
- Instruction L is a leader if it immediately follows a goto or conditional goto statement like: if goto B or goto B

For each leader, its basic block consists of the leader and all statement up to. It doesn't include the next leader or end of the program

Q.10. Construct the DAG for following statement

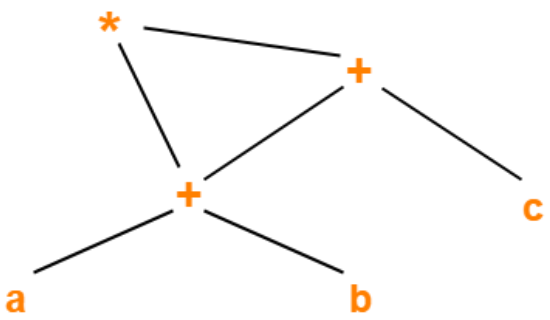
(a + b) * (a + b + c)

Ans: Three Address Code for the given expression is-

T1 = a + b

T2 = T1 + c

T3 = T1 * T2



Directed Acyclic Graph

Q.11. Write different applications of DAG.

Ans:

DAGs are used for the following purposes-

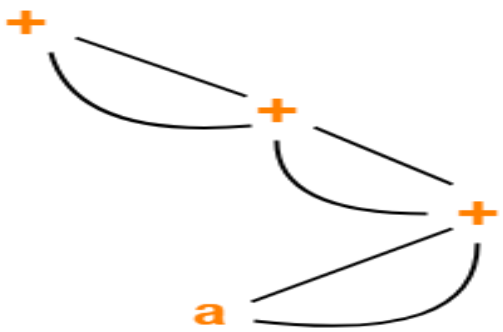
- To determine the expressions which have been computed more than once (called common sub-expressions).
- To determine the names whose computation has been done outside the block but used inside the block.
- To determine the statements of the block whose computed value can be made available outside the block.
- To simplify the list of Quadruples by not executing the assignment instructions x:=y unless they are necessary and eliminating the common sub-expressions.

Q.12. Consider the following expression and construct a DAG for it-

(((a + a) + (a + a)) + ((a + a) + (a + a)))

Ans:

- Directed Acyclic Graph for the given expression is-



Directed Acyclic Graph

Q.13. Consider the following basic block-

B10:

S1 = 4 x I

S2 = addr(A) – 4

S3 = S2[S1]

S4 = 4 x I

S5 = addr(B) – 4

S6 = S5[S4]

S7 = S3 x S6

S8 = PROD + S7

PROD = S8

S9 = I + 1

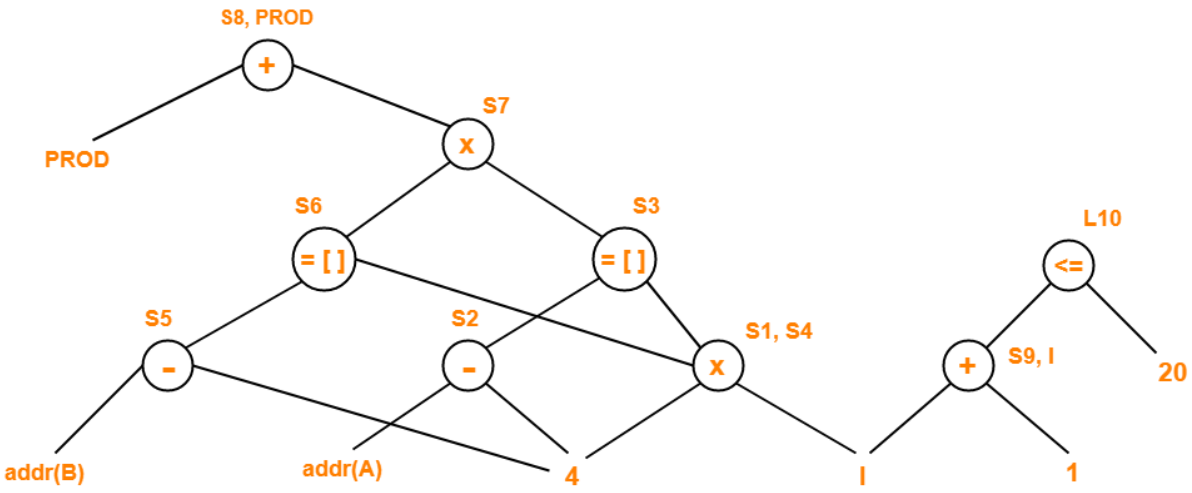
I = S9

If I <= 20 goto L10

(i)Draw a directed acyclic graph and identify local common sub-expressions.
(ii)After eliminating the common sub-expressions, re-write the basic block.

Ans:

(i) Directed Acyclic Graph for the given basic block is-



Directed Acyclic Graph

(ii) In this code fragment,

- 4 x I is a common sub-expression. Hence, we can eliminate because S1 = S4.
- We can optimize S8 = PROD + S7 and PROD = S8 as PROD = PROD + S7.
- We can optimize S9 = I + 1 and I = S9 as I = I + 1.

After eliminating S4, S8 and S9, we get the following basic block-

B10:

S1 = 4 x I

S2 = addr(A) – 4

S3 = S2[S1]

S5 = addr(B) – 4

S6 = S5[S1]

S7 = S3 x S6

PROD = PROD + S7

I = I + 1

If I <= 20 goto L10

Q.14. Explain loop unrolling and loop jamming?

(a) Loop Unrolling:

Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. We basically remove or reduce

iterations. Loop unrolling increases the program's speed by eliminating loop control instruction and loop test instructions.

Example:

```
Initial code:

for (int i=0; i<5; i++)
    printf("Pankaj\n");

Optimized code:

printf("Pankaj\n");
printf("Pankaj\n");
printf("Pankaj\n");
printf("Pankaj\n");
printf("Pankaj\n");
```

(b) Loop Jamming:

Loop jamming is the combining the two or more loops in a single loop. It reduces the time taken to compile the many number of loops.

Example:

```
Initial Code:

for(int i=0; i<5; i++)
    a = i + 5;
for(int i=0; i<5; i++)
    b = i + 10;

Optimized code:
for(int i=0; i<5; i++)
{
 a = i + 5;
 b = i + 10;
}
```

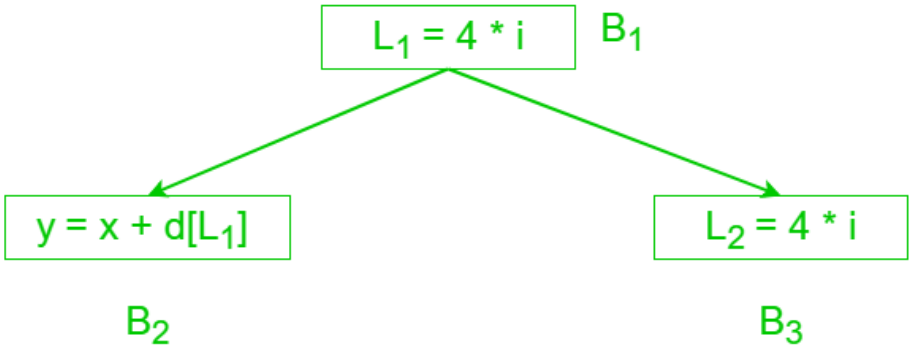
Q.15. Explain various **Data** Flow Properties in code optimization.

Ans: various **Data** Flow Properties are as following:

(a)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.

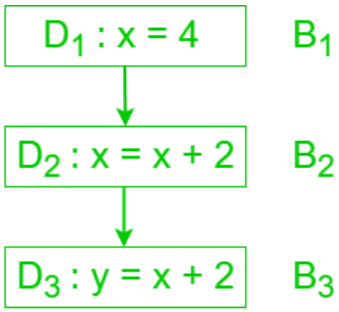
Example –



Expression 4 * i is available for block B_2 , B_3

Advantage –
It is used to eliminate common sub expressions.

(b)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.
Example –

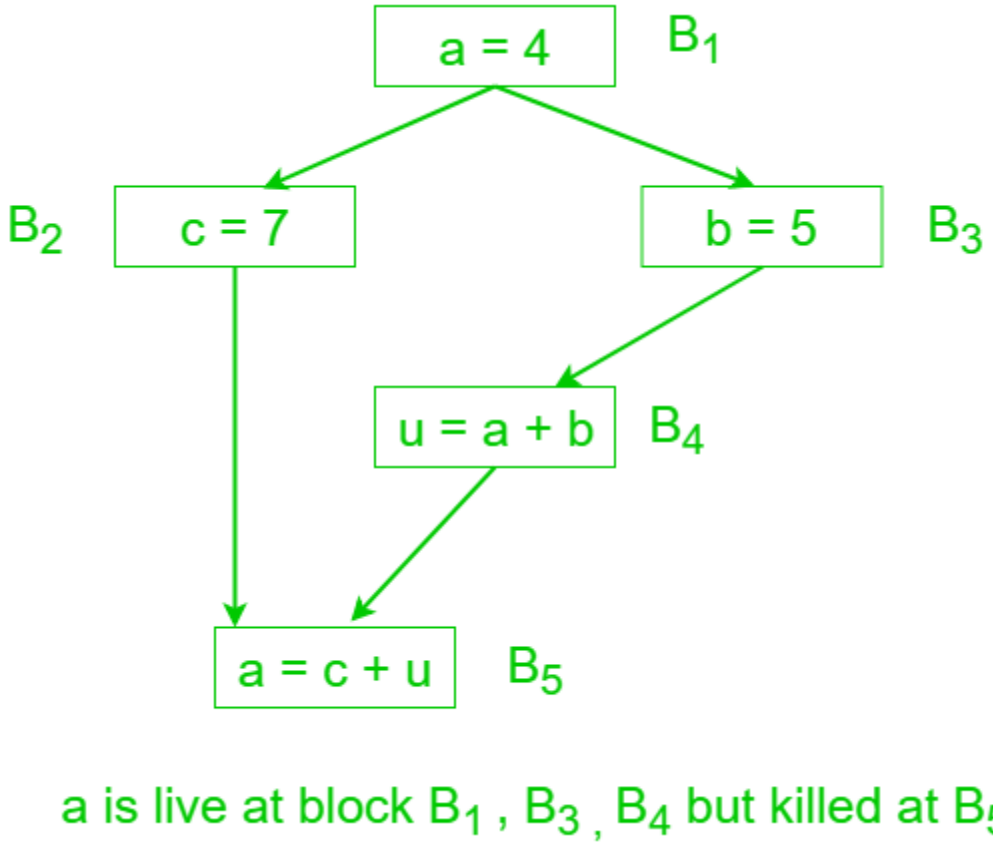


D_1 is reaching definition for B_2 but not for B_3 since it is killed by D_2

Advantage –
It is used in constant and variable propagation.

(c)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.

Example –



Advantage –

1. It is useful for register allocation.
2. It is used in dead code elimination.

(d)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.

Advantage –

It is used for performing code movement optimization.

Q.16. Explain techniques of the peephole optimization?

Ans:

Peephole Optimization Techniques:

1. **Redundant load and store elimination:**
   In this technique the redundancy is eliminated.

Initial code:

```
y = x + 5;
i = y;
z = i;
w = z * 3;
```

Optimized code:

```
y = x + 5;
```

```
i = y;
w = y * 3;
```

2. **Constant folding:**

   The code that can be simplified by user itself, is simplified.

   Initial code:

   ```
   x = 2 * 3;
   ```

   Optimized code:

   ```
   x = 6;
   ```

3. **Strength Reduction:**
   The operators that consume higher execution time are replaced by the operators consuming less execution time.

   Initial code:

   ```
   y = x * 2;
   ```

   Optimized code:

   ```
   y = x + x;    or     y = x << 1;
   ```

   Initial code:

   ```
   y = x / 2;
   ```

   Optimized code:

   ```
   y = x >> 1;
   ```

4. **Null sequences:**
   Useless operations are deleted.
5. **Combine operations:**
   Several operations are replaced by a single equivalent operation.

Q.17. Explain the Design Issues of code generation phase?

Ans: In the code generation phase, various issues can arises:

1. Input to the code generator

2. Target program
3. Memory management
4. Instruction selection
5. Register allocation
6. Evaluation order

1. Input to the code generator

- The input to the code generator contains the intermediate representation of the source program and the information of the symbol table. The source program is produced by the front end.
- Intermediate representation has the several choices:
  a) Postfix notation
  b) Syntax tree
  c) Three address code
- We assume front end produces low-level intermediate representation i.e. values of names in it can directly manipulated by the machine instructions.
- The code generation phase needs complete error-free intermediate code as an input requires.

2. Target program:

The target program is the output of the code generator. The output can be:

a) **Assembly language:** It allows subprogram to be separately compiled.

b) **Relocatable machine language:** It makes the process of code generation easier.

c) **Absolute machine language:** It can be placed in a fixed location in memory and can be executed immediately.

3. Memory management

- During code generation process the symbol table entries have to be mapped to actual p addresses and levels have to be mapped to instruction address.
- Mapping name in the source program to address of data is co-operating done by the front end and code generator.

- Local variables are stack allocation in the activation record while global variables are in static area.

4. **Instruction selection:**

- Nature of instruction set of the target machine should be complete and uniform.
- When you consider the efficiency of target machine then the instruction speed and machine idioms are important factors.
- The quality of the generated code can be determined by its speed and size.

Example:

The Three address code is:

a:= b + c
d:= a + e

Inefficient assembly code is:

MOV b, R0	R0→b
ADD c, R0	R0→ c + R0
MOV R0, a	a → R0
MOV a, R0	R0→ a
ADD e, R0	R0 → e + R0
MOV R0, d	d → R0

5. **Register allocation**

Register can be accessed faster than memory. The instructions involving operands in register are shorter and faster than those involving in memory operand.

The following sub problems arise when we use registers:

Register allocation: In register allocation, we select the set of variables that will reside in register.

Register assignment: In Register assignment, we pick the register that contains variable.

Certain machine requires even-odd pairs of registers for some operands and result.

For example:

Consider the following division instruction of the form:

D x, y

Where,

x is the dividend even register in even/odd register pair

y is the divisor

Even register is used to hold the reminder.

Old register is used to hold the quotient.

6. **Evaluation order**

The efficiency of the target code can be affected by the order in which the computations are performed. Some computation orders need fewer registers to hold results of intermediate than others.

Q.18. Consider the following code-

```
prod = 0 ;
i = 1 ;
do
{
prod = prod + a[ i ] x b[ i ] ;
i = i + 1 ;
} while (i <= 10) ;
```

(i) Compute the three address code.
(ii)Compute the basic blocks and draw the flow graph.

Ans: (i)
Three address code for the given code is-

```
prod = 0
```

i = 1

T1 = 4 x i

T2 = a[T1]

T3 = 4 x i

T4 = b[T3]

T5 = T2 x T4

T6 = T5 + prod

prod = T6

T7 = i + 1

i = T7

if (i <= 10) goto (3)

(ii)

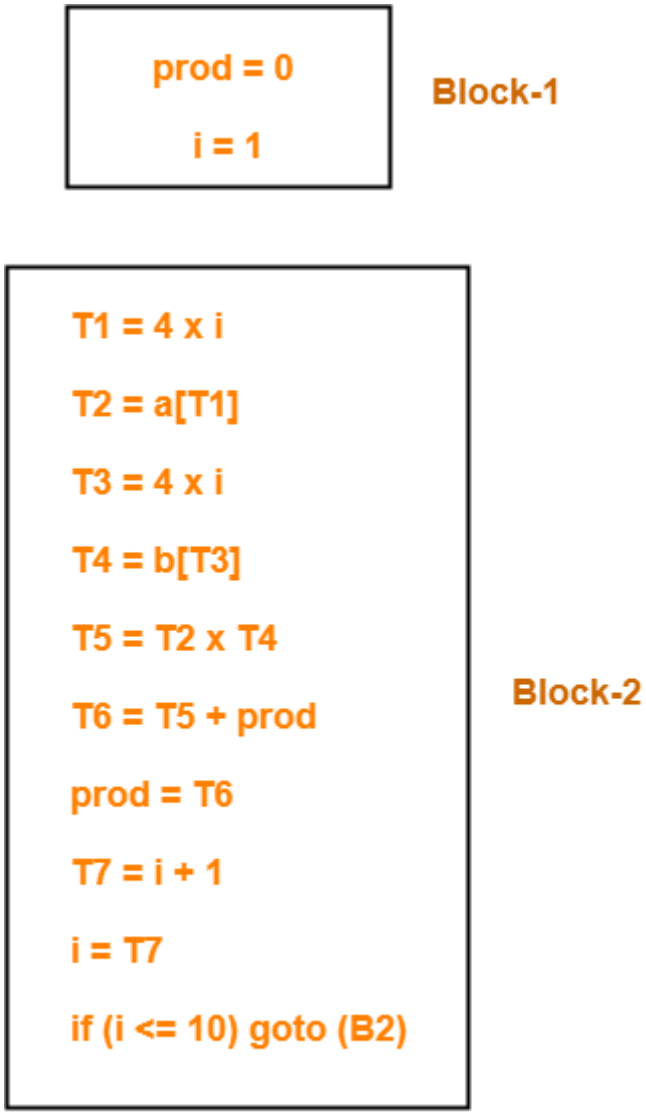
Step-01:

We identify the leader statements as-

- prod = 0 is a leader because first statement is a leader.
- T1 = 4 x i is a leader because target of conditional or unconditional goto is a leader.

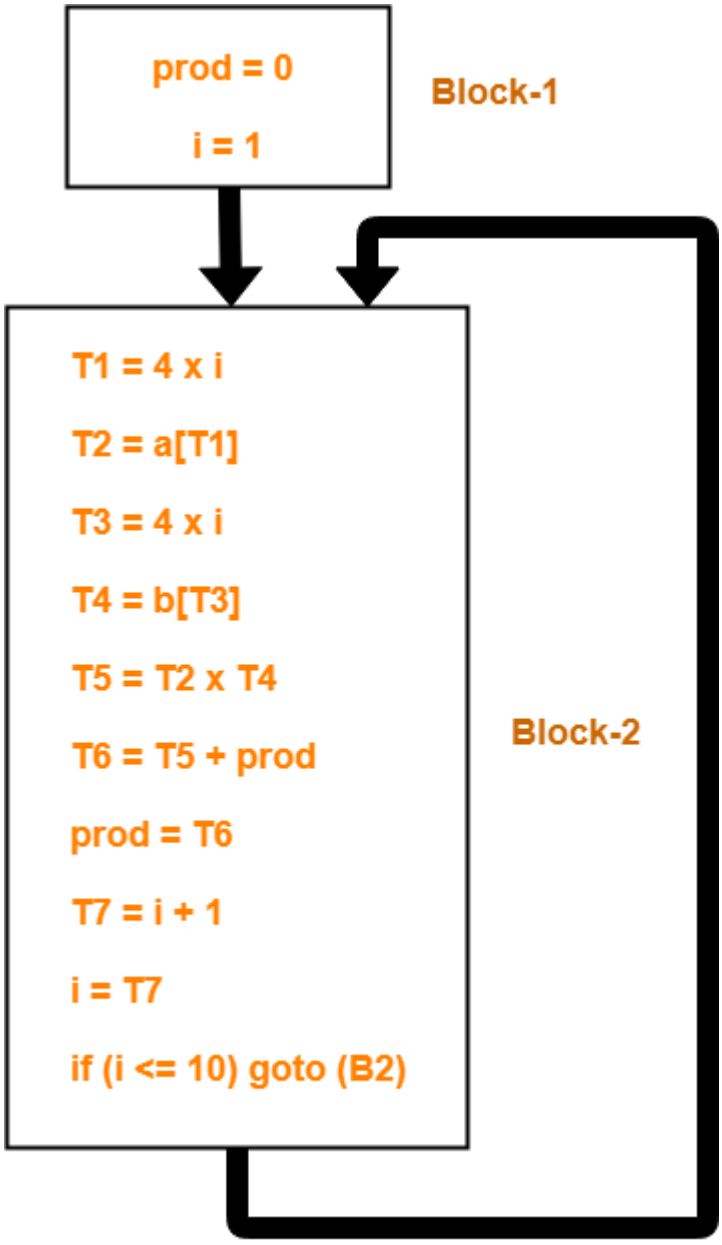
Step-02:

The above generated three address code can be partitioned into 2 basic blocks as-



Step-03:

The flow graph is-



Flow Graph

Q.19. Write a short note with example to optimize the code:

a. Dead code elimination

b. Code motion

c. Reduction in strength

Ans:

a. <u>Dead code elimination</u>: Variable propagation often leads to making assignment statement into dead code.

Befo elimination:

```
c = a * b
x = a
till
d = a * b + 4
```

After elimination:

```
c = a * b
till
d = a * b + 4
```

b. <u>Code motion</u>:

Reduce the evaluation frequency of expression.

- Bring loop invariant statements out of the loop.

```
a = 200;
 while(a>0)
 {
    b = x + y;
    if (a % b == 0}
    printf("%d", a);
  }


//This code can be further optimized as
a = 200;
b = x + y;
while(a>0)
 {
    if (a % b == 0}
    printf("%d", a);
  }
```

c. Reduction in strength: Strength reduction means replacing the high strength operator by the low strength.

```
i = 1;
while (i<10)
{
   y = i * 4;
}


//After Reduction
i = 1
t = 4
{
  while( t<40)
  y = t;
```

```
  t = t + 4;
}
```

Q.20. Describe the following terms:

(a). Flow graph

(b). Dominators

(c). Natural loops

(d). Inner loops

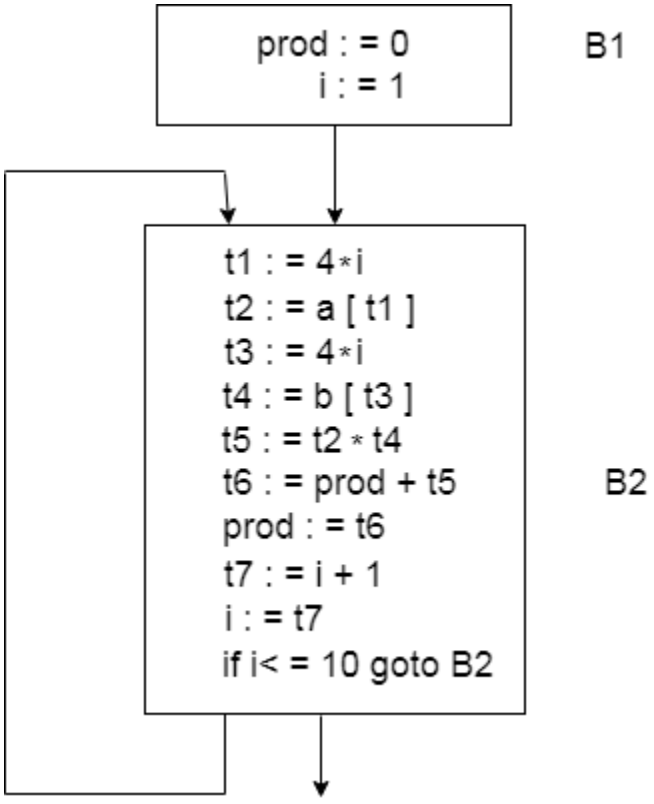
(e). Reducible flow graphs

Ans:

(a). **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:



(b)**Dominators:**

In a flow graph, a node d dominates node n, if every path from initial node of the flow graph to n goes through d. This will be denoted by d dom n. Every initial node dominates all the remaining nodes in the flow graph and the entry of a loop dominates all nodes in the loop. Similarly every node dominates itself.

Example:
*In the flow graph below,

*Initial node,node1 dominates every node. *node 2 dominates itself

*node 3 dominates all but 1 and 2. *node 4 dominates all but 1,2 and 3.
*node 5 and 6 dominates only themselves,since flow of control can skip around either by goin through the other.

*node 7 dominates 7,8 ,9 and 10. *node 8 dominates 8,9 and 10.
*node 9 and 10 dominates only themselves.

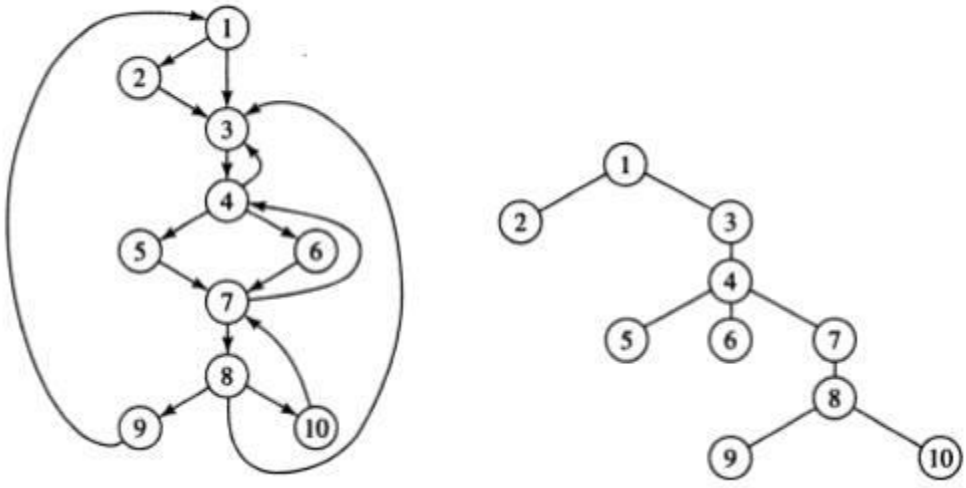


Fig. 5.3(a) Flow graph (b) Dominator tree

The way of presenting dominator information is in a tree, called the dominator tree, in which

- The initial node is the root.
- The parent of each other node is its immediate dominator.
- Each node d dominates only its descendents in the tree.

The existence of dominator tree follows from a property of dominators; each node has a unique immediate dominator in that is the last dominator of n on any path from the initial node to n. In terms of the dom relation, the immediate dominator m has the property is d=!n and d dom n, then d dom m.

***

D(1)={1}

D(2)={1,2}
D(3)={1,3}
D(4)={1,3,4}
D(5)={1,3,4,5}
D(6)={1,3,4,6}
D(7)={1,3,4,7}
D(8)={1,3,4,7,8}
D(9)={1,3,4,7,8,9}
D(10)={1,3,4,7,8,10}

(c)Natural Loops:

One application of dominator information is in determining the loops of a flow graph suitable for improvement. There are two essential properties of loops:

- Ø A loop must have a single entrypoint, called the header. This entry point-dominates all nodes in the loop, or it would not be the sole entry to the loop.
- Ø There must be at least one way to iterate the loop(i.e.)at least one path back to the headerOne way to find all the loops in a flow graph is to search for edges in the flow graph whose heads dominate their tails. If a→b is an edge, b is the head and a is the tail. These types of

edges are called as back edges.

Example:

In the above graph,

7→4 4 DOM 7

10 →7 7 DOM 10
4→3
8→3
9 →1

The above edges will form loop in flow graph. Given a back edge n → d, we define the natural loop of the edge to be d plus the set of nodes that can reach n without going through d. Node d is the header of the loop.

Algorithm: Constructing the natural loop of a back edge.
Input: A flow graph G and a back edge n→d.

Output: The set loop consisting of all nodes in the natural loop n→d.

Method: Beginning with node n, we consider each node m*d that we know is in loop, to make sure that m's predecessors are also placed in loop. Each node in loop, except for d, is placed once
on stack, so its predecessors will be examined. Note that because d is put in the loop initially, we never examine its predecessors, and thus find only those nodes that reach n without going through d.

```
Procedure insert(m);

if m is not in loop then begin loop := loop U {m};

push m onto stack end;
stack : = empty;

loop : = {d}; insert(n);
while stack is not empty do begin
pop m, the first element of stack, off stack;

        for each predecessor p of m do insert(p)
end
```

(d)Inner loops:

If we use the natural loops as "the loops", then we have the useful property that unless two loops have the same header, they are either disjointed or one is entirely contained in the
other. Thus, neglecting loops with the same header for the moment, we have a natural notion of inner loop: one that contains no other loop.

When two natural loops have the same header, but neither is nested within the other, they are combined and treated as a single loop.

Pre-Headers:

Several transformations require us to move statements "before the header". Therefore begin treatment of a loop L by creating a new block, called the preheader. The pre-header has only the header as successor, and all edges which formerly entered the header of L from outside L instead enter the pre-header. Edges from inside loop L to the header are not changed. Initially the pre-header is empty, but transformations on L may place statements in it.

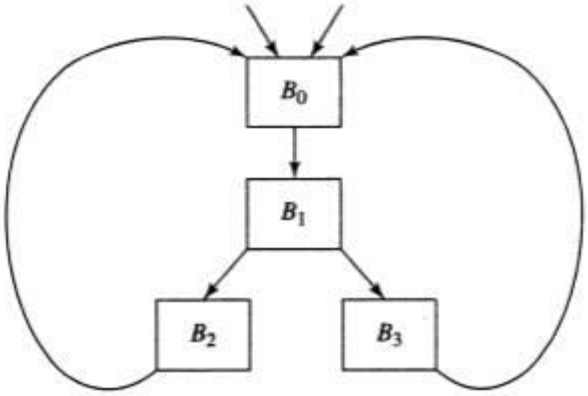


Fig. 5.4 Two loops with the same header

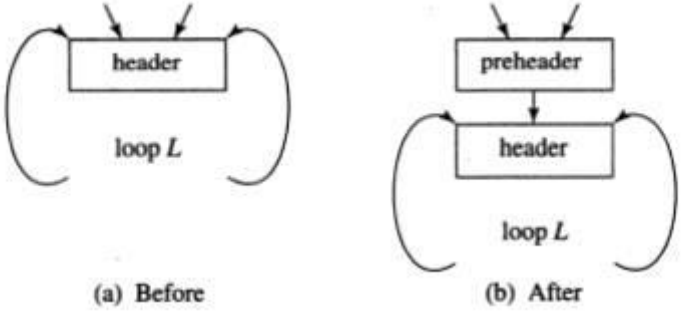


Fig. 5.5 Introduction of the preheader

(e)Reducible flow graphs:

Reducible flow graphs are special flow graphs, for which several code optimization transformations are especially easy to perform, loops are unambiguously defined, dominators can be easily calculated, data flow analysis problems can also be solved efficiently. Exclusive use of structured flow-of-control statements such as if-then-else, while-do, continue, and break statements produces programs whose flow graphs are always reducible.

The most important properties of reducible flow graphs are that

1. There are no umps into the middle of loops from outside;
2. The only entry to a loop is through its header

Definition:

A flow graph G is reducible if and only if we can partition the edges into two disjoint groups, forward edges and back edges, with the following properties.

1. The forward edges from an acyclic graph in which every node can be reached from initial node of G.
2. The back edges consist only of edges where heads dominate theirs tails.

Example: The above flow graph is reducible. If we know the relation DOM for a flow graph, we can find and remove all the back edges. The remaining edges are forward edges. If the forward edges form an acyclic graph, then we can say the flow graph reducible. In the above example remove the five back edges 4→3, 7→4, 8→3, 9→1 and 10→7 whose heads dominate their tails, the remaining graph is acyclic.