UNIT - 5

MACHINE INDEPENDENT OPTIMIZATION

Code Optimization -

- It is a program transformation technique
- It is to reduce the no. of lines in the code.
- It tries to improve the intermediate code by consuming fewer resources (i.e cpu, memory)
- It improves the speed of execution of machine code.

Objectives of the Code optimization -

- The optimised code must be correct
- It must not change the meaning of the program.
- Optimization should improve the performance of the program.
- The optimization process should not delay the overall compiling process.

Types of code optimization -

There are two types of code optimization. They are

1. Machine Independent optimization
2. Machine dependent optimization

Types of Code optimization

Machine Independent optimization

- It improves the intermediate code
- The code doesnot involve any cpu registers or absolute memory locations

Eg: Loop optimization

Machine Dependent optimization

- It improves the target code (Based on the target machine architecture.
- It involves cpu registers and may have absolute memory references rather than relative references

Eg: peephole optimization

Machine Independent Optimization

1. Loop Optimization -

purpose-
- Reducing no. of lines inside a loop
- Reducing no. of loops
- Reducing no. of executions in a loop.

Types of Loop optimization -

a) Code Motion (or) Frequency Reduction

b) Loop unrolling

c) Loop Jamming

a) Code Motion (or) Frequency Reduction.

- Moving the code outside the loop.
- Moving the code from high frequency region to low frequency is called code motion
- Reduces the evaluation frequency of an expression
- Brings loop invariant statements out of the loop.

Eg:-

i)
```
i=0;
while (i<5000)
{
A = sin(x)/cos(x) * i;
i++;
}
```

⟹

```
t = sin(x)/cos(x)
i=0
while (i<5000)
{
A = t * i;
i++;
}
```

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

⟹

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

b) Loop Unrolling -

Reducing the number of times comparision are made in the loop.

```
while (i<10)
{
    x[i] = 0;
    i++;
}
```

Loop condition checked 10 times

⟹

```
while (i<10)
{
    x[i] = 0;
    i++;
    x[i] = 0;
    i++;
}
```

Loop condition checked 5 times

c) Loop Jamming-

Combining two or more loops (or) reducing the no. of loops.

Eg:-

```
for (i=0; i<10; i++)
    for (j=0; j<10; j++)
        x[i][j] = 0;
for (i=0; i<10; i++)
    x[i][i] = 0
```

⟹

```
for (i=0; i<10; i++)
    for (j=0; j<10; j++)
        x[i][j] = 0;
        x[i][i] = 0;
```

3/1/22

Flow Graph-

Data flow analysis - (we can extract useful info)

Flow graph = Nodes + Edges
(Basic Blocks) (control flow)

- Intra procedural analysis
- It determines useful information for optimization.

Data flow properties -

* Available Expression
* Reaching Definitions
* Live variable
* Busy expressions

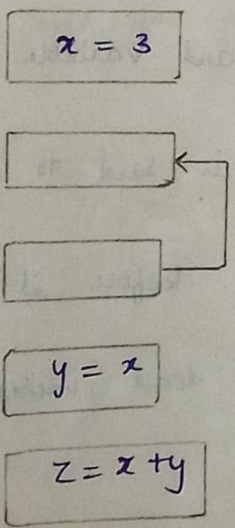
Basic terminologies -

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 some evaluation of expression.

Ex :-



Definition point

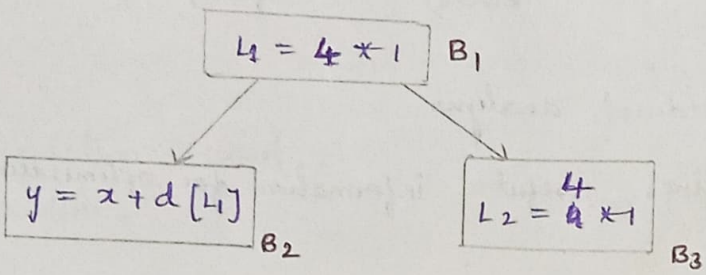
Reference point

Evaluation point

Fig: Program points

Properties:-

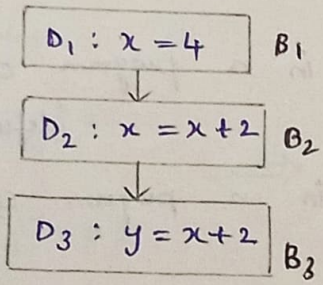
1. Available Expression- A expression is said to be available if none of the operands gets modified before their use



Expression 4 * 1 is available for Block B_2, B_3.

Advantage- It is used to eliminate common sub expressions.

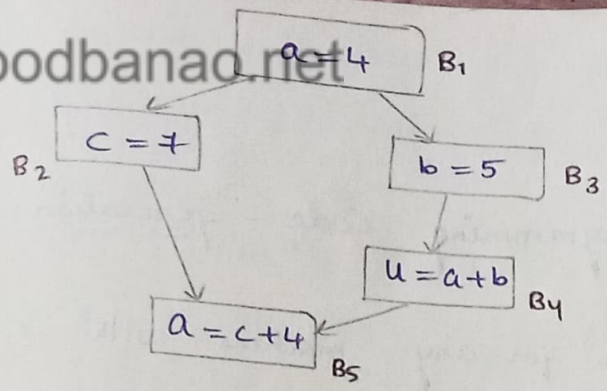
2. Reaching Definition - A definition 'D' reaches a point x if there is a path from D to X in which D is not killed ie not redefined.



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

Advantage- used in constant and variable propogation

3. Live variable- A variable is said to be live, when the value of a is used before it is redefined. Otherwise a is said to be dead variable

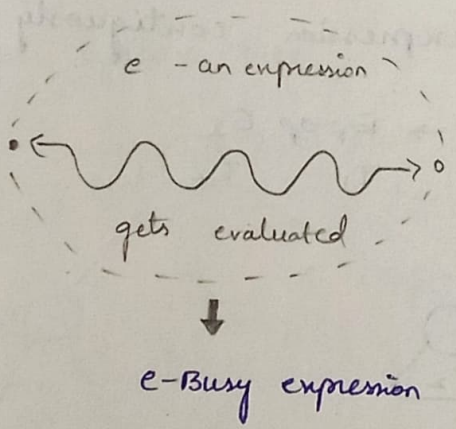


a is live at block B_1, B_3, B_4 but killed at B_5

Advantage-

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

4. Busy variable- An expression is busy at some program point P if it will definitely be evaluated before its value changes



Advantage- It is used for performing code movement optimization.

6/1/23 UNIT 5 -

LOOPS IN FLOW GRAPH -

Loop is a collection of nodes in flow graph such that

i) all such nodes are strongly connected.

ii) collection of nodes has unique entry.

iii) The loop that contains no other loop is called inner loop.

- Some common technologies being used for loop in flow graph -

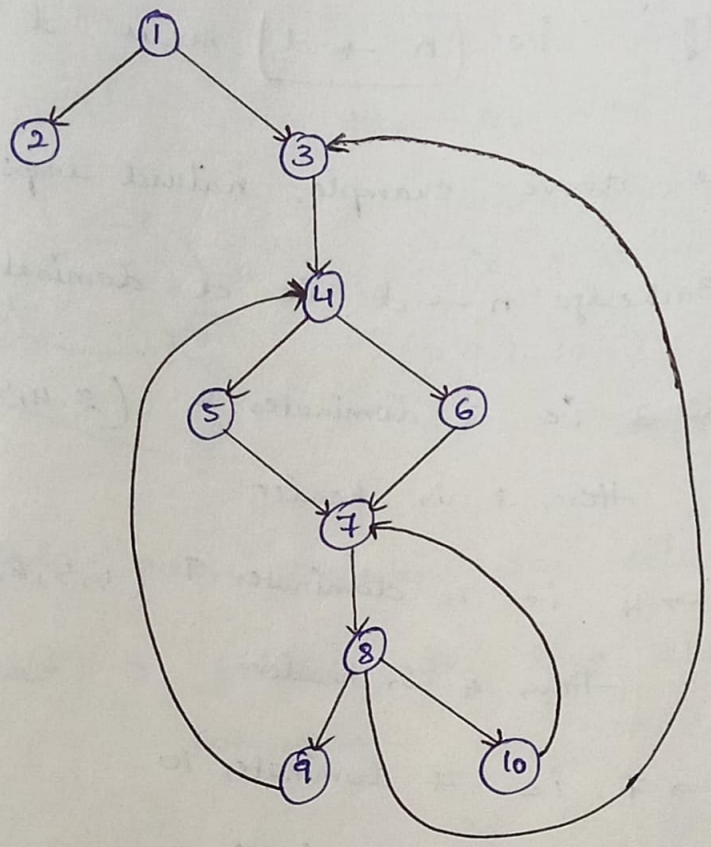
1. Dominators -

Every initial node dominates all the

remaining nodes in the flow graph

- Similarly, every node dominates itself.

Ex :-



Nodes: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Node 1 - Dominates all the nodes in the flow graph (itself also)

Node 2 - Dominates itself

Node 3 - Dominates all nodes except 1 and 2.

Node 4 - Dominates all nodes except 1, 2 and 3

Node 5 and 6 - Dominates themselves.

Node 7 - Dominates 7, 8, 9, 10

Node 8 - Dominates 8, 9, 10

Node 9 and 10 - Dominates themselves.

2. Natural Loops

Natural loops can be defined by 'Back Edge' i.e $n \to d$ means d dominates n.

- In the above example, natural loops are,

Backedge $n \to d$ i.e d dominates n.

i) $8 \to 3$ i.e 3 dominates 8 (3, 4, 5, 6, 7, 8)

Here, 3 is header

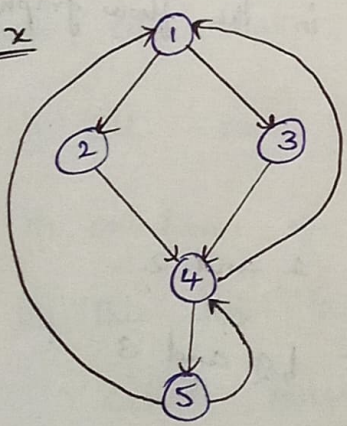
ii) $9 \to 4$ i.e 4 dominates 9 (4, 5, 6, 7, 8, 9)

Here, 4 is header

iii) $10 \to 7$ i.e 7 dominates 10

Here, header is 7

Ex



Natural loops are,

Backedge :

i) $4 \to 1$

1 dominates 4 (1, 2, 3, 4)

ii) $5 \to 4$

4 dominates 5 (4, 5)

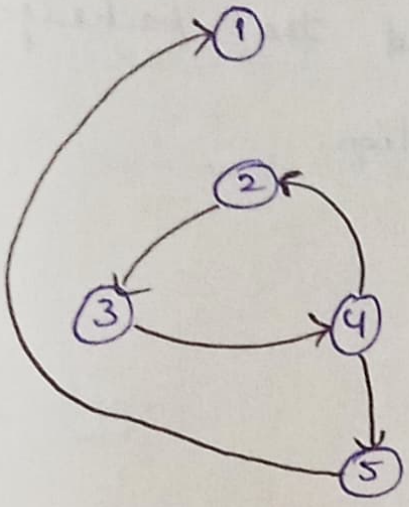
iii) $5 \to 1$

1 dominates 5 (5 → 1, 2, 3, 4, 5)

3. Inner loop

Inner loop is a loop that contains no other loop.

Ex:-



Here the inner loop is,

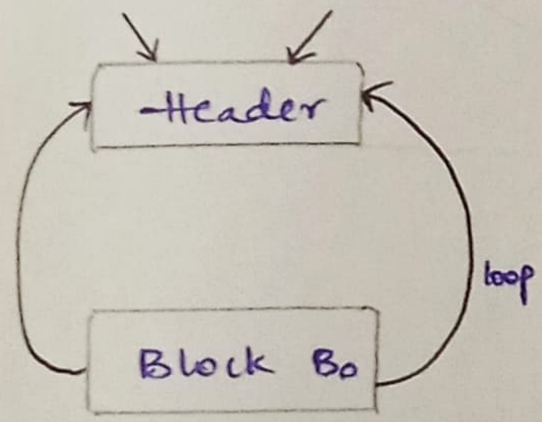
4 → 2

i.e 2 dominates 4

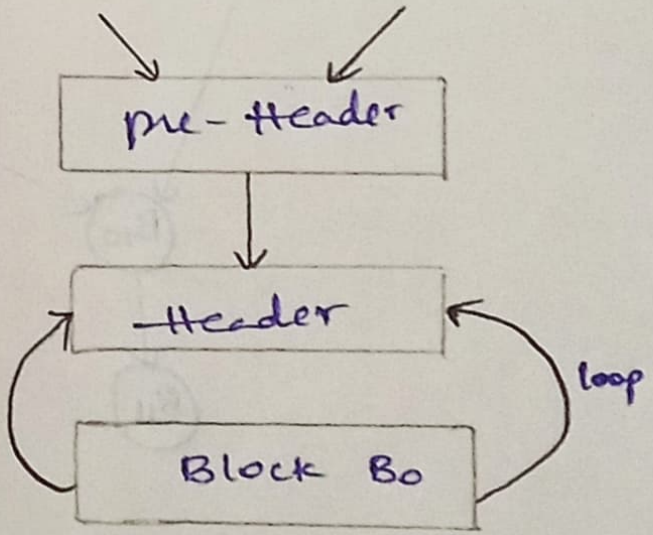
(4 → 2, 3, 4)

4. Pre-Header- means preceding the Actual Header.

- It is used to facilitate the loop transformation computation.



Before pre-Header

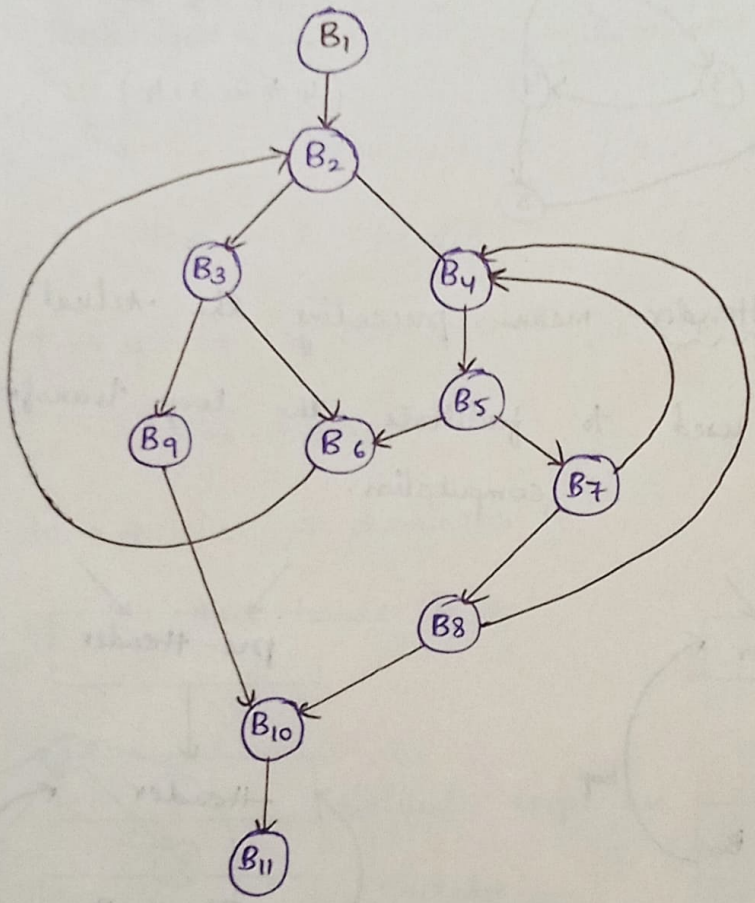


After pre-Header

Example - Consider the following flow graph and find a) Dominators for each basic block

b) Detect all the loops in the flow graph

c) For each loop, find the backedge and header-block information.



NOTE- Every node dominates itself

a) Dominators -

Node B_1 - Dominates all the nodes

Node B_2 - Dominates all nodes except B_1.

Node B_3 - Dominates B_9.

Node B_4 - Dominates B_5, B_7, B_8

Node B_5 - Dominates B_7, B_8

Node B_6 - Dominates itself

Node B_7 - Dominates B_8

Node B_8 - Dominates itself

Node B_9 - Dominates itself.

Node B_{10} - Dominates B_{11}.

Node B_{11} - Dominates itself

b) - c)

Natural loops-

i) $B_6 \rightarrow B_2$ $\therefore B_2$ is header

ii) $B_7 \rightarrow B_4$ $\therefore B_4$ is header

iii) $B_8 \rightarrow B_4$ $\therefore B_4$ is header