Code optimization

The code optimization is Required to Produce an efficient code. The Improvement over Intermediate Code by transformation is called optimization.

Optimization
- Machine Independent
- Machine Dependent

There are three criteria that we applied optimization

(1) Must Presure meaning of Program.

(2) Speedup Program by measureble amount of time.

(3) must warth the effort.

→ Code Optimization Techniques

- Platform Dependent Techniques

(i) Peephole optimization techniques

(II) Instruction Level parallelism

(III) Data Level Parallelism

(IV) Cache optimization

(V) Redundant Resource.

Platform Independent Techniques

Loop optimization
↳ Loop unrolling
→ Codemovement
→ frequency reduction
→ Loop Jamming

Constant folding

Constant Propgation

Common Sub Expression elimination

Peephole optimization

Peephole optimization is simple and effective technique to Improve the Performance of target code by selecting a small set of target Instructions and replacing these Instructions by shorter or faster code. This small set of Instructions or small part of code to which optimization is performed is known as Peephole or window.

Peephole is a machine Dependent optimization.

Characteristics of Peephale optimization

① Redundant Instruction Elimination

Redundant Load & stores Instructions Can be eliminated in this type of transformation.

Ex

$a = b + c$

$d = a + e$

Mov b, R_0

Add c, R_0

Mov R_0, a

Mov a, R_0

Add e R_0

Mov R_0 d

② Strength Reduction

certain machine Instruction are simpler than other so we can replace Complex Instructions by Simpler

Ex

$2*x = x+x$

$x**2 = x*x$

$x/2 = .5*x$

③ Flow of Control optimization

Using peephole optimization, unnecessary jump can be eliminated.

```
if a<b goto TEST            |   if a<b : goto Done
     :                      |          :
TEST : goto Done            |   Test    goto Done
     :                      |          :
     :                      |          :
DONE                        |   DONE
       (Before)             |        (After)
```

④ Algebric simplification — Peephole optimization is an efficent technique for algebric simplification.

Ex

$x = x + 0$

$x = x * 1$

⑤ Machine idioms → The target Instruction have equivalent machine Instructions for performing some operations. so we can replace there target Instructions by equivalent machine Instruction to Improve efficency.

Ex

Auto Increment Instruction — for A+

Auto decrement Instruction — for A-

Loop optimization → Loop optimization is a technique in which optimization is performed on the Loops.

→ Loop optimization Techniques

① Code motion

② Loop unrolling

③ Loop Jamming

④ Reduction in strength

⑤ Reduction Variable Elimination

Code motion (Loop Invariant Computation)

* Code motion is a technique which moves the code outside the loop

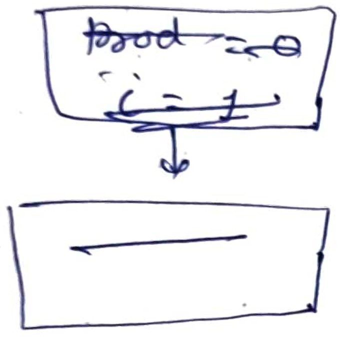
* if there is an Expression in the Loop where Result remains unchanged even after executing the loop for several times then such Expression should be placed outside the Loop.

Before Code motion	after Code motion
`while(i<=max)`	`n = max-1`
`{`	`while(i<=n)`
`sum = sum+a[i]`	`{`
`}`	`sum = sum+a[i]`
	`}`

Induction Variable — A variable to be Induction variable, if the value of variable gets changed everytime, It is either Incremented or decremented by some constant

When there are two or more variable in a loop, It may be Possible to get rid of all but one

Example



Ex → Induction Variable

```
i = i+1
t2 = 4*i
t3 = a[t2]
if t3<v goto B2
```

$t_2 = 4*i$

```
t2 = t2+4
t3 = a[t2]
if t3<v goto B2
```

Reduction In strength →

strength Reduction means replacing the high strength operato
by the low strength operatar.

Before

```
for (i = 1; i<=50; i++ )
  {
    count = i*7;
  }
```

after

```
t = 7
for(i =1; i<=50; i++)
  {
    count = t;
      t = t+7;
  }
```

Loop unrolling → is a method in which no of jumps and tests can be reduced by writing the code multiple times It reduces execution time but Increases memory Load.

Ex

Before Loop Unrolling

```
int i = 1
while (i<=100)
{
  a[i] = b[i];
      i++;
 }
```

after Loop Unrolling

```
int i =1
while ( i<=100)
{
  a[i] = b[i]
    i++;
  a[i] = b[i]
    i++
 }
```

Loop Jamming (Loop fusion) - In Loop Jamming method several loops are merged to one Loops. when two adjacent loops would the same no of items then their bodies can be merged.

Before Loop Jamming

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

After Loop Jamming

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

Basic Block → A basic block is a sequence of Consecutive statements in which the flow of Control enters at the begening and leaves at the end without branching or halt.

1.

Algorithm to Convert 3-address code to Basic blocks

① Determine the Leader from three-address Code
(a) 1st statement of 3-address code.
(b) Target statement of unconditional/Conditional goto is a "Leader".

① Statement Immediately following unconditional or Conditional goto is a "Leader".

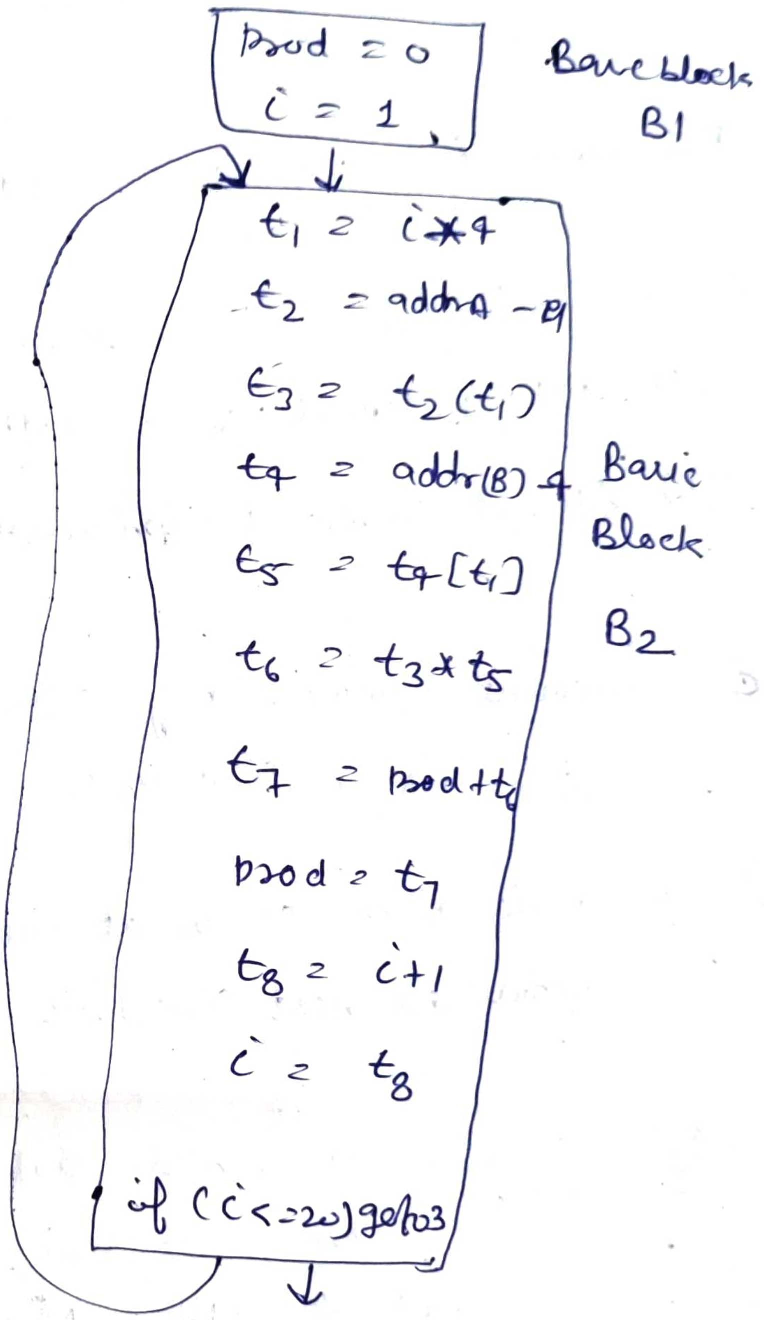
② The basic block starts at one "Leader" and ends just before the next "Leader".

Flow Graph → A flow graph is a directed graph whose nodes are basic blocks and edges are used to add the flow of Information from one block to another.

Loop in flow graph — Loop is a Collection of nodes in a flow graph such that there is a Path from any node to any other node which that loop. There is always one Path from a node outside the loop to the node Inside the loop.

Example   Find the basic block, flow graph, and loops for the following 3-address code.

```
(1)  prod = 0  (Leader)
(2)  i = 1
(3)  t1 = i*4        (Leader)
(4)  t2 = addr(A)-4
(5)  t3 = t2[t1]
(6)  t4 = addr(B)-4
(7)  t5 = t4[t1]
(8)  t6 = t3*t5
(9)  t7 = prod+t6
(10) prod = t7
(11) t8 = i+1
(12) i = t8
(13) if(i<=20) goto(3)
14   - -  (Leader)
```



Dead Code elimination

A variable is said to be dead at a point in a program if the value contained in it is never been used.

The code containing such a variable is said to be dead code.

The optimization can be perform by eliminating such a dead code.

```
   i=0;
 if( i==1)
 {
   a = x+5;
 }
```

In this, if statement is dead code as this condition will never be satisfied. So this is dead code.

```
        {
          a = x+5;
        }
```

Constant folding

→ In this we evaluate constant expression at compile time and replace the constant expression by their values.

"Replacement of Run-time computation by the compile time computation is called constant folding."

So the Expression $2*3.14$ is replaced by 6.28

③ Common Sub expression alimenation

The Common Sub Expression is an Expression appearing repeatedly in the Program which is Computed previously.

Example -1

Before Common Sub Expression	After Common Sub expression Elimenation
$a = b+c$	$a = b+c$
$b = a-d$	$b = a-d$
$c = b+c$	$c = b+c$
$d = a-d$	$d = b$

Example -2

$t_1 = 4*i$	$t_1 = 4*i$
$t_2 = addr(A)-4$	$t_2 = addr(A)-4$
$t_3 = t_2[t_1]$	$t_3 = t_2[t_1]$
$t_4 = t_4*i$	$t_5 = addr(B)-4$
$t_5 = addr(B)-4$	$t_6 = t_5[t_1]$
$t_6 = t_5[t_4]$	

Value number :- A value is a number associated with each variable used within the basic block

* It uniquely identifies the place in the basic block where the variable was last assigned a value.

* The value number of all the variables are Initialized at the start of basic block.

Ex

a = x+y —(1)

b = x+y —(2)

c = a+i —(3)

x = y —(4)

d = b+i —(5)

a = a*d —(6)

x	0
y	1
a	2

(i) Value table

b = x+y
= a

x	0
y	1
a	2
b	2

(ii) Value table

x	0
y	1
a	2
b	2
i	3
c	4

(iii) Value table

x	1
y	1
a	2
b	2
i	3
c	4

(iv)

d = b+i
= c

x	1
y	1
a	2
b	2
i	3
c	4
d	4

(v)

a = a*d

x	1
y	1
a	5
b	2
i	3
c	4
d	4

(vi)

Directed Acyclic Graph (DAG) →

A ~~DAG~~ ~~als~~ Directed Acyclic graph Called DAG is a ~~derived~~ Directed graph that Contains no cycle.

* A DAG is used to optimize basic block.
* A DAG is used to eliminate Common Sub Expression.
* DAG Specify how the value Computed by each statement in a basic block is used in subsequent statements of the block.
* To apply Transformation on basic block a DAG is ~~Converted~~ Constructed from 3 address code.

Algorithm for Construction of DAG

(1) In a DAG leaf nodes, represent (Identifier, names or Constants)

(2) Interior nodes represent operator.

(3) While Constructing DAG, A check is made to find, if there is an existing node with the same children. A new node is created only when such a node does not exist. It helps to detect Common Sub Expression.

(4) The assignment of the form $x = y$ must not be performed unless and until it is not.

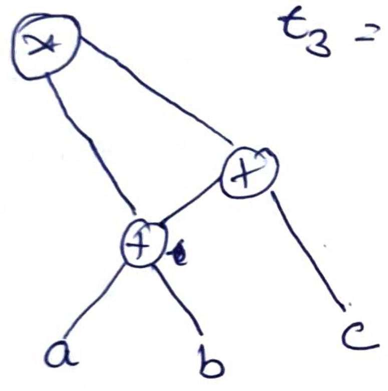
<u>Example</u> – Construct DAG for the given Expression

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

<u>Ans-1</u> – Three address Code for the given Expression

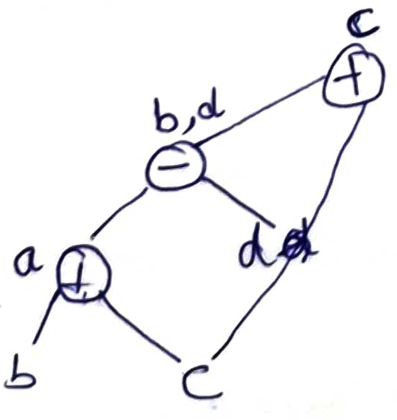
$t_1 = a + b$
$t_2 = t_1 + c$
$t_3 = t_1 * t_2$

<u>DAG</u>



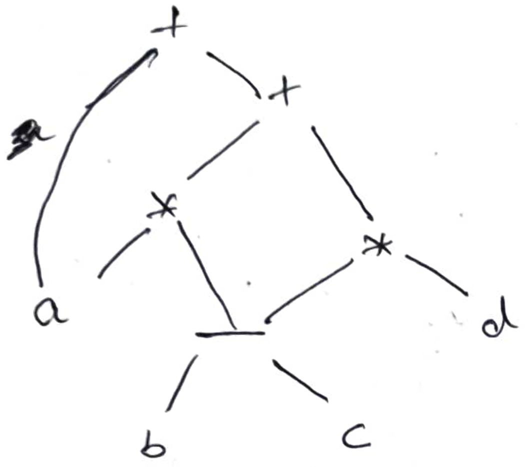
<u>Ex</u> <u>DAG</u> →

$a = b + c$
$b = a - d$
$c = b + c$
$d = a - d$



Example

$a + a * (b-c) + (b-c) * d$



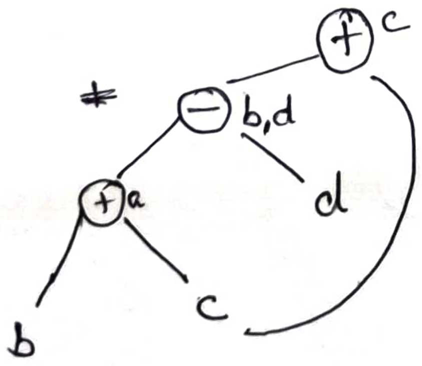
Ex

$a = b + c$

$b = a - d$

$c = b + c$

$d = a - ~~d~~$



Construct the DAG for the following 3- address Code

(1) $t_1 = i * 4$

(2) $t_2 = addr(a) - 4$

(3) $t_3 = t_2[t_1]$

(4) $t_4 = i * 4$

(5) $t_5 = addr(b) - 4$

(6) $t_6 = t_5[t_4]$

(7) $t_7 = t_3 * t_6$

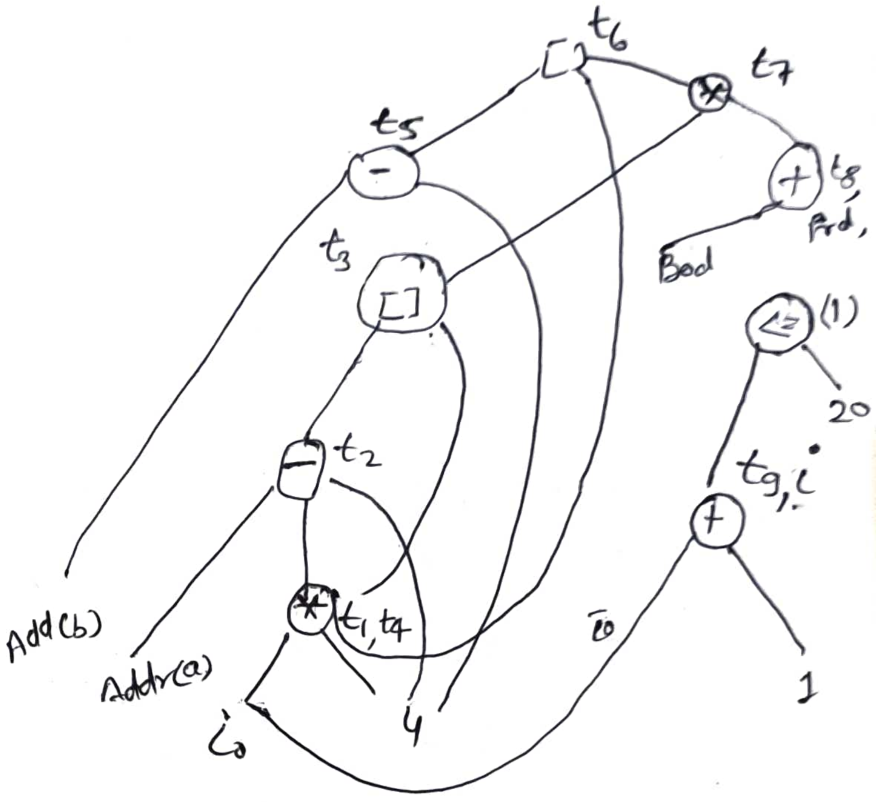
(8) $t_8 = prod + t_7$

(9) $Prod = t_8$

(10) $t_9 = i + 1$

(11) $i = t_9$

(12) if $i <= 20$ goto (1)



$a = b * c$

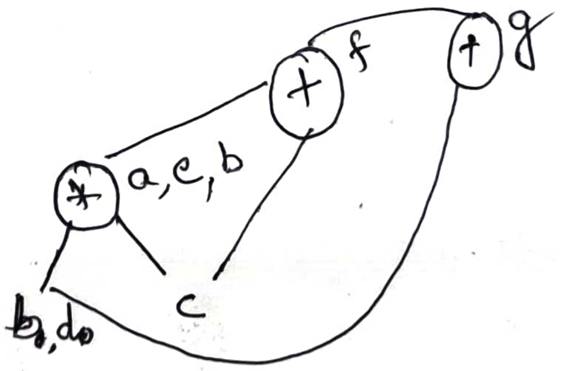
$d = b$

$e = d * c$

$b = e$

$f = b + c$

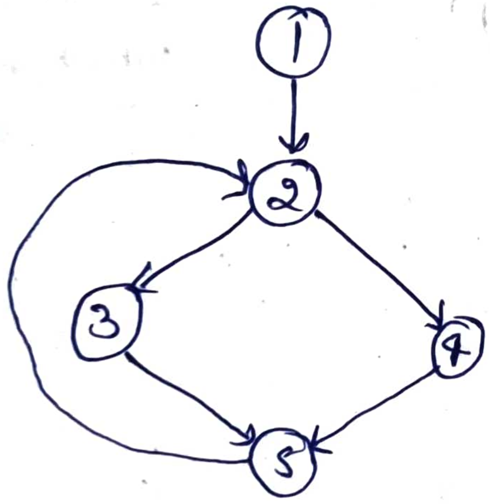
$g = f + d$



Loops In Flow graph

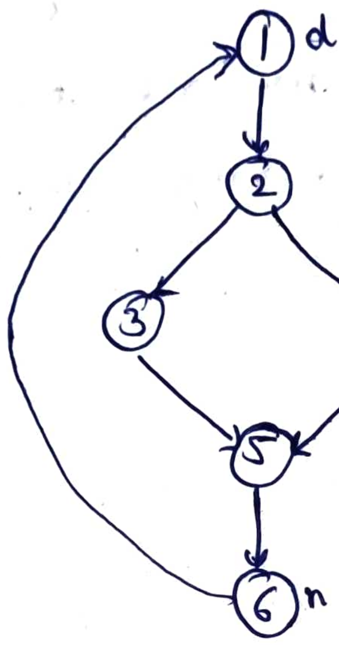
Dominators → A node d is said to dominate node n in a flow graph if every Path to node n from Initial node goes through d only.

Every Initial node dominates all the remaining node in a flow graph. every nodes also dominates itself.



Node 1	- dominates	- Nodes 2, 3, 4, & 5 in addition to itself
Node 2		Node 3, 4 & 5 in addition to itself
Node 3	dominates itself	
Node 4	"	
Node 5	"	

Natural loops → A natural loop can be defined by a back edge n → d such that there exist a collection of all nodes that can reach to n without going through d.



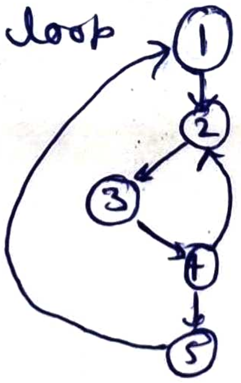
Natural Loop

d + {all nodes that can reach to n without going through d }

natural loop 6 → 1

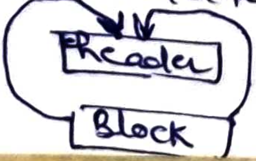
{ 2, 3, 4, 5, 6 1 }

Inner Loop → Inner loop is a loop that contains no other loop

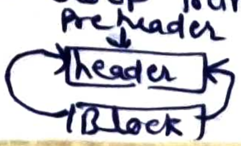


Inner loop 4 → 2
[2, 3, 4]

Preheader → The Preheader is a new block created such that successor of this block is header block. It is added to faciliates loop transformation optimization.



Header
Block



Preheader
header
Block