Principle source of optimization (code optimization)

It comes under back end of the compiler reducing the amount of space and runs faster.

→ The code produced by the straight forward compiling algorithm can often be made to run faster and take less space.

→ This improvement means the straight forward algorithm which takes less space is achieved by the program transformations that are traditionally called as optimization.

optimization is classified into two categories

1. machine independent optimization
2. machine dependent optimization

Machine independent optimization:-

They are program transformation that improve target code without taking into consideration any properties of target machine. (It transfers the program to improve the target code with out taking any help from target machine).

Machine dependent optimization:-

→ These are based on register allocation and utilization of special machine instruction sequence.

⇒ Code optimization techniques

→ Compile time evaluation
→ Variable propagation
→ dead code elimination
→ code motion
→ Induction variable and strength reduction

To convert the code from the code optimization to code Generator There are 5 steps.

Compile time Evaluation:-

a) $x = 5 \times (45.0/5.0) \times r$

at compile time.

b) $x = 37$

$y = x/3.6$

Evaluate $\frac{37}{3.6}$

→ In compile time evaluation it evaluates the place where the evaluation needed.

→ In compile time evaluation we doesn't need to insert the values. It will directly insert the values.

Variable propagation:-

before optimization	After optimization
c = a * b	c = a * b
x = a	x = a
till	till
d = x * b + 4	d = a * b + 4.

These are identified as common sub expressions.

deadcode elimination:-

Before elimination	After elimination
c = a * b	c = a * b
x = b	till
till	d = a * b + 4
d = a * b + 4	

x = b is not used so it is considered as dead code and it is also called as dead state. In this stage the

dead state is removed.

code motion :-

→ It reduces the evaluation frequency of expression.

→ It brings loop invariant statement out of the loop.

before optimization

```
a=200;
while (a>0)
{
  b=x+y;        → out of the loop.
  if(a%b==0)
  printf("%d",a);
}
```

After optimization

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

Induction Variable and strength reduction :- → replace the high strength operator with low strength operator then we can reduce the strength

→ The Induction Variable is used for loop

Eg:- before reduction

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

After reduction

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

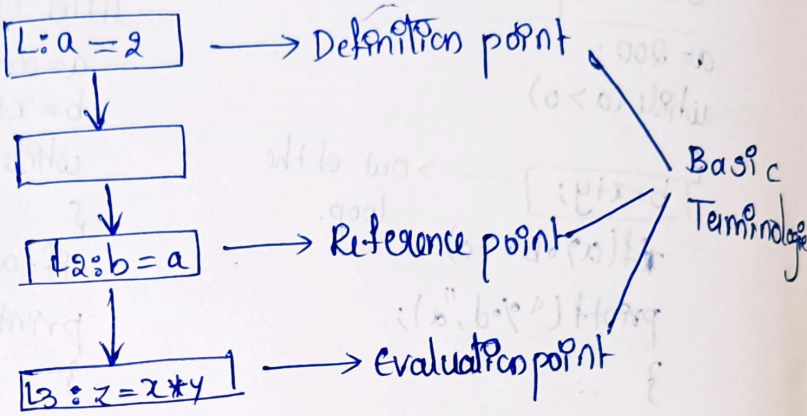
two techniques of code optimization
local optimization
global optimization

Data Flow Analysis:-

The important optimization techniques under global optimization is Data flow Analysis.

→

→ It is a analysis that determines the information regarding the definition and use of data in program, with the help of this, global optimization can be done. (the flow of data)



L: a = 2
L2: b = a
L3: z = x * y

→ In data flow analysis in data we have Variables, Values, assigned variables etc. we will arrange all that data into flow graph so that we can analyse the data easily

→ The ~~follow~~ flow graph is made by using the definition and usage of data. whenever we are making the flow there are various point that we can determine in Data flow graph. the points are Definition point, Reference point, Evaluation point.

→ we can when we see the expression L: a = 2, we are defining 'a' variable and the 'a' variable is

assigned to the value i.e, 2. So this is called Definition point.

→ After defining the value of 'a' and then we use 'a' as reference to another variable 'b' then it is called Reference point. i.e L2: b=a

→ In L3: x=x*y this expression we have to evaluate each and every term so it is known as Evaluation point.

→ Data flow properties:-

1) Available expression
2) Reaching definition.

(1) Available expression:-

An expression 'a+b' is said to be available at a program point 'x', if none of its operands get modified before their use. It is used to eliminate common subexpressions.

B1: L1=4*i

B2:- Y=X+L1 B3: P=L1*3

L1, i.e 4*i is available for block B2 & B3

Reaching definition :-

A definition D is reaching to a point x if D is not killed or redefined before that point. It is used in constant / variable propagation

B1 : D1 : x = 4

B2 : D2 : x = x + 2

B3 : D3 : y = x + 2

D1 is reaching definition for B2 but not for B3 since it is killed by

Loop optimization :-

It is most valuable machine-independent optimization because programs inner loop takes bulk time of a programmer

→ If we decrease the number of instructions in an inner loop then the running time of a program may be improved even if we increase the amount of code outside that loop.

→ for loop optimization the following three techniques are important:-

1. code motion
2. Induction-variable elimination
3. strength reduction.

1) code motion :-

It is used to decrease the amount of code in loop. This transformation takes a statement or expression which can be moved outside the loop body without affecting the semantics of the program

Eg:- while (i<= limit-2) // stmt does not change limit

After code motion the result is as follows.

```
a = limit-2;
while (i<=a)
```

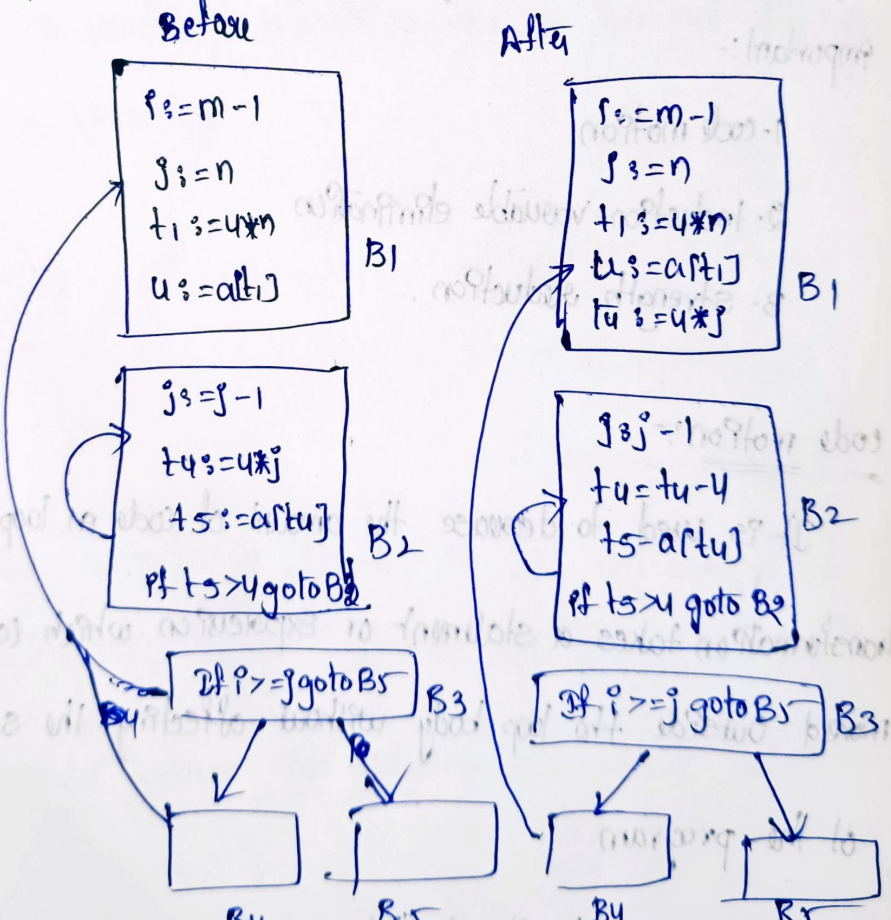
while (i<=a) //stmt does not change limit a

In this while stmt, the limit-a Equation is a loop invariant Equation.

2. Induction - Variable Elimination.

→ It is used to replace variable from inner loop

→ It can reduce the number of additions in a loop. It improves both code space and runtime performance.



3) Reduction in strength :-

→ strength reduction is used to replace the expensive operation by the cheaper once on the teuget machine.

→ Addition of a constant is cheaper than a multiplication so we can replace multiplication with an addition with the loop

→ Multiplication is cheaper than exponent. So we can replace exponent with multiplication within the loop

ex > mul > add

example :-

before

```
while (i<10)
{
  j=3*i+1;
  a[j]=a[j]-2;
  i=i+2
}
```

After:-

```
s=3*i+1;
while (i<10)
{
  j=s;
  a[j]=a[j]-2;
  i=i+2;
  s=s+6;
}
```

In the above code.

It is cheaper to Compute

s=s+6 than

s=3*6.