CODE OPTIMI~~S~~ZATION

Code Optimization Introduction :

Optimization

- Machine Independent
  - ① Loop optimizations
    - ⓐ codemotion (or) frequency reduction
    - ⓑ loop unrolling.
    - ⓒ loop jamming.
  - ② folding
    - constant propagation
  - ③ Redundancy elimination.
  - ④ Strength reduction.
- Machine dependent
  - ① Register allocation.
  - ② Use of addressing mode.
  - ③ peephole optimization.
    - ⓐ Redundant load/store.
    - ⓑ flow of control optimizations.
    - ⓒ strength reduction.
    - ⓓ use of machine idioms.

M/C Independent

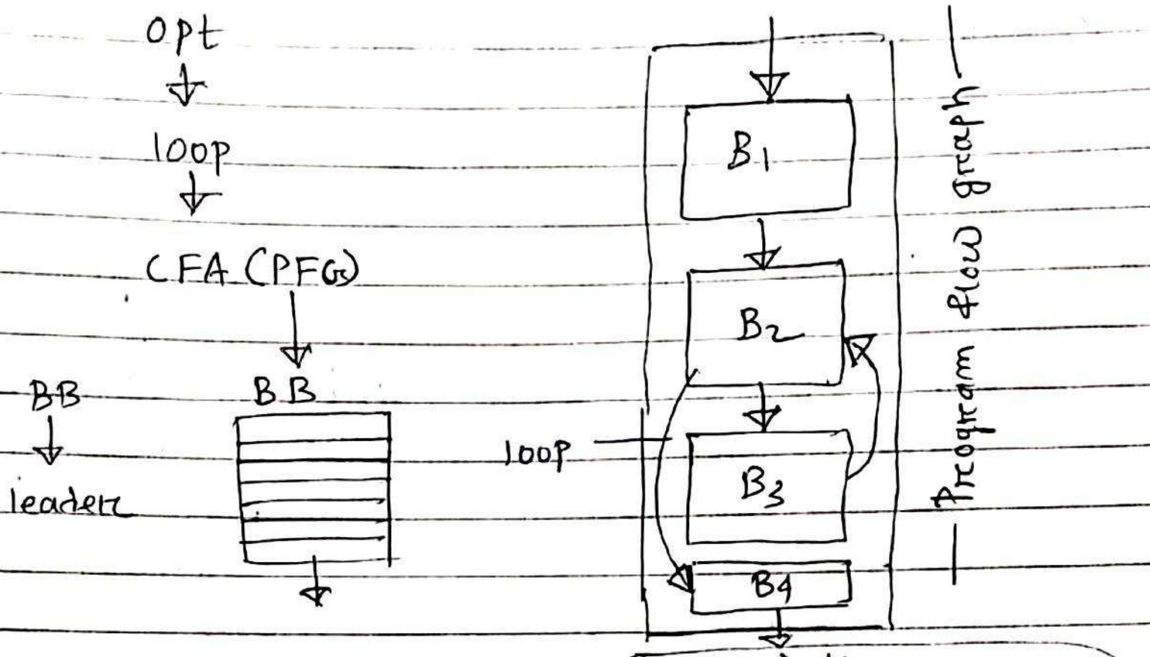
① loop optimizations

→ To apply loop optimizations, we must first detect loops.

→ For detecting loops we use control flow analysis (CFA) using ~~#~~ program flow graph.

→ To find PFG, we need to find basic blocks.

A Basic block is a sequence of 3-address ~~code~~ statements where control enters at the beginning and leaves only at the end without any jumps or halts.



(cycle indicate a loop).

• Algorithm to find the Basic Block:

→ In~~or~~ order to find the basic blocks, we need to find the leaders in programm. Then a basic block will start from one leader to the next leader but not including next leader.

• Identifying leaders in a basic block:

① A statement is a leader.

② statement that is target of conditionas or unconditionas statement is deader.

③ Statement that follows immediately a conditionas or unconditionas statement is a leader.

Example = (find out leaders)

Programm –

```
fact(x)
{ int ~~x~~ f=1
  for (i=2; i<=x; i++)
      f=f*i;
  return f;
}
```

3 add code –

```
*1) f=1          → leader.      (not including other
 2) i=2    B1                    leader) - we make
                                 block)
leader ← *3) if (i>x) goto (9)  B2
*4) t1=f*i;  → leader
 5) f=t1;
 6) t2=i+1;      B3
 7) i=t2;
 8) goto(3).
*9) Goto calling program (return) → leader
                         B4
```

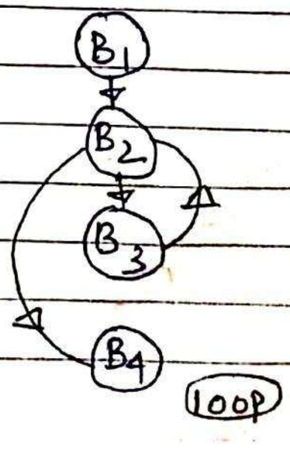
→ using Algo we find '4'-leaders. and '4' block.

→ no. of Basic Block depends on. number of leaders.

→ with 'n' leader g we get 'n' Block.

Program Flow graph –

applys-CFA



B2 and B3 falling -loop

loop

- Type of loop optimization :-

① **Frequency reduction** :

Moving the code from high frequency region to low frequency region is called code motion.

Ex:

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

$\Downarrow$

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

② **loop unrolling** : effectively redue the number of comparision:

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

$\Downarrow$

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

$\boxed{0\ 1}\boxed{2\ 3}\boxed{4\ 5}\boxed{6\ 7}\boxed{8\ 9}$

③ **Loop jamming** : combining the bodies of two loops.

```
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;
    }
}
```

② **folding** =

Replacing an expression that can be computed at compile time by its values.

Ex: 2+3+C+B = 5+C+B

③ **Redundancy Elimination** = (DAG)

A = B+C
D = 2+B+3+C

D = 2+3+A.
= 5+A

④ Strength reduction = Replacing a costly operation cheaper one.

Ex: $\frac{B = A * 2}{B = A << 1}$.

⑤ Algebraic simplification =

A = A+0 } eleminate
x = x*1 } such statements.

M/c dependent opt =

CO-DISS
① Register allocation
- local allocation
- global allocation.

CO-DISS
② use of addressing modes.

③ peephole optimization =
(a) Redundant load and store elimination =

x = y+z
- MOV y, R0
- add z, R0
- MOV R0, x

a = b+c	Mov b, R0
d = a+e	add c, R0
	Mov R0, a ✗
	Mov a, R0 ✗
	add e, R0
	MOV R0, d.

(c) flow of control optimization =

Avoid jumps on jumps

```
L1: jum ~~L2~~ L4
  :
L2: jum L3
  :
L3: jum L4
```

eliminate dead code

```
# define x 0
if(x)
{
    ↑
    |  dead code
    ↓
}
```

(d) use of m/c idioms =

```
i = i+1 | mov R0, i  }
        | add R0, 1  } ⇒ inc i
        | mov i, R0  }
```