CODE OPTIMIZATION

(Improve target code)

- Run faster
- Take less space

Machine independent code optimization

NOT REQUIRED: Machine instruction, Register allocation

(Does not require properties of target machine)

Dead code elimination + Variable propagation

```
c = a*b;
x = a;
for (i=1; i<=N; i++)
{
    d = (x*b) + i;
}
```

→

```
c = a*b;
for (i=1; i<=N; i++)
{
    d = c + i;
}
```

Code motion: Bring loop invariant statements outside

```
for (i=1; i<=N; i++)
{
    x = 10;
    y = x + i;
}
```

→

```
x = 10;
for (i=1; i<=N; i++)
{
    y = x + i;
}
```

```
for (i=1; i<=N; i++)
{
    x = 10;
    y = x + 3;
}
```

→ (dead code elimination)

```
x = 10;
y = x + 3;
```

eliminate loop.

Reducing strength

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

→ '*' is replaced by '+'

```
i = 1;
t = 4;
while (t<40)
{
    y = t;
    t = t+4;
}
```

* → Booth Algorithm

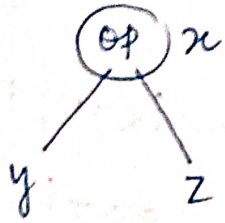
\+ →

```
i = 1;
t = 0;
while (t<40)
{
    t = t+4;
}
y = t;
```

DIRECTED ACYCLIC GRAPH (DAG)

Case-1

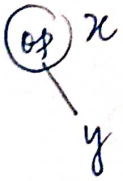
$x = y \text{ op } z$



Case-2

$x = \text{op } y$

or $x = x \text{ op } y$

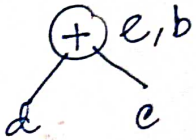


Case-3

$x = y$

eg. $e = d \times c$

$b = e$

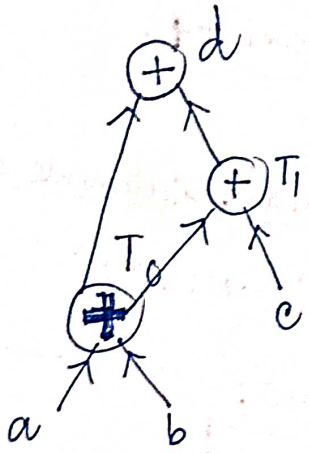


Eg.-1

$T_0 = a + b$

$T_1 = T_0 + c$

$d = T_0 + T_1$



Eg.2

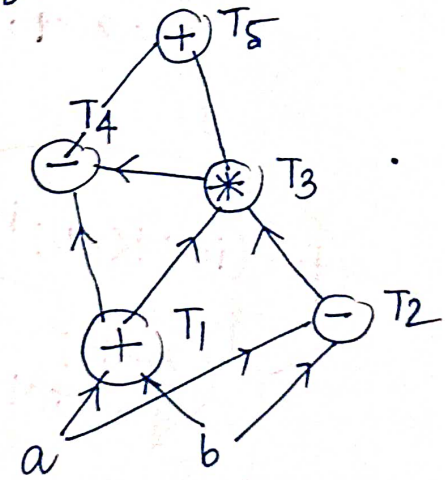
$T_1 = a + b$

$T_2 = a - b$

$T_3 = T_1 * T_2$

$T_4 = T_1 - T_3$

$T_5 = T_4 + T_3$



Eg.3 ① $a = b * c$ ✓

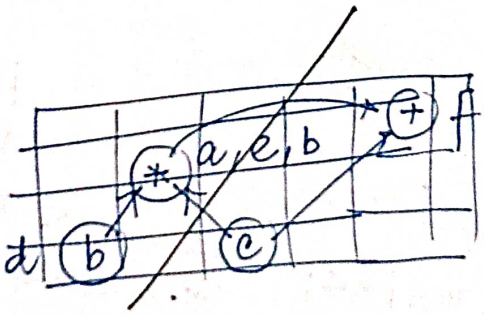
② $d = b$ ✓

③ $e = d * c$

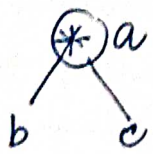
④ $b = e$

⑤ $f = b + c$

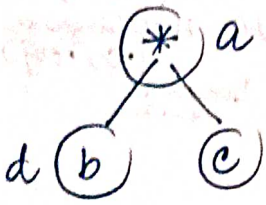
⑥ $g = f + d$



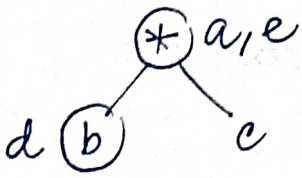
$a = b * c$



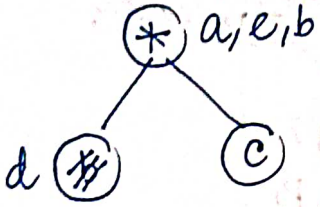
$a = b * c$
$d = b$



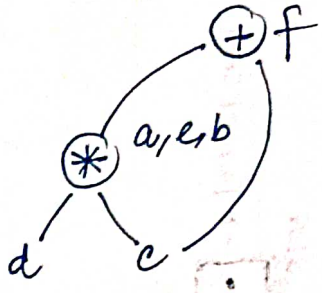
$a = b * c$
$d = b$
$e = d * c$



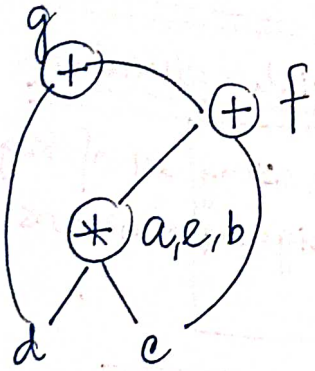
$a = b * c$
$d = b$
$e = d * c$
$b = e$



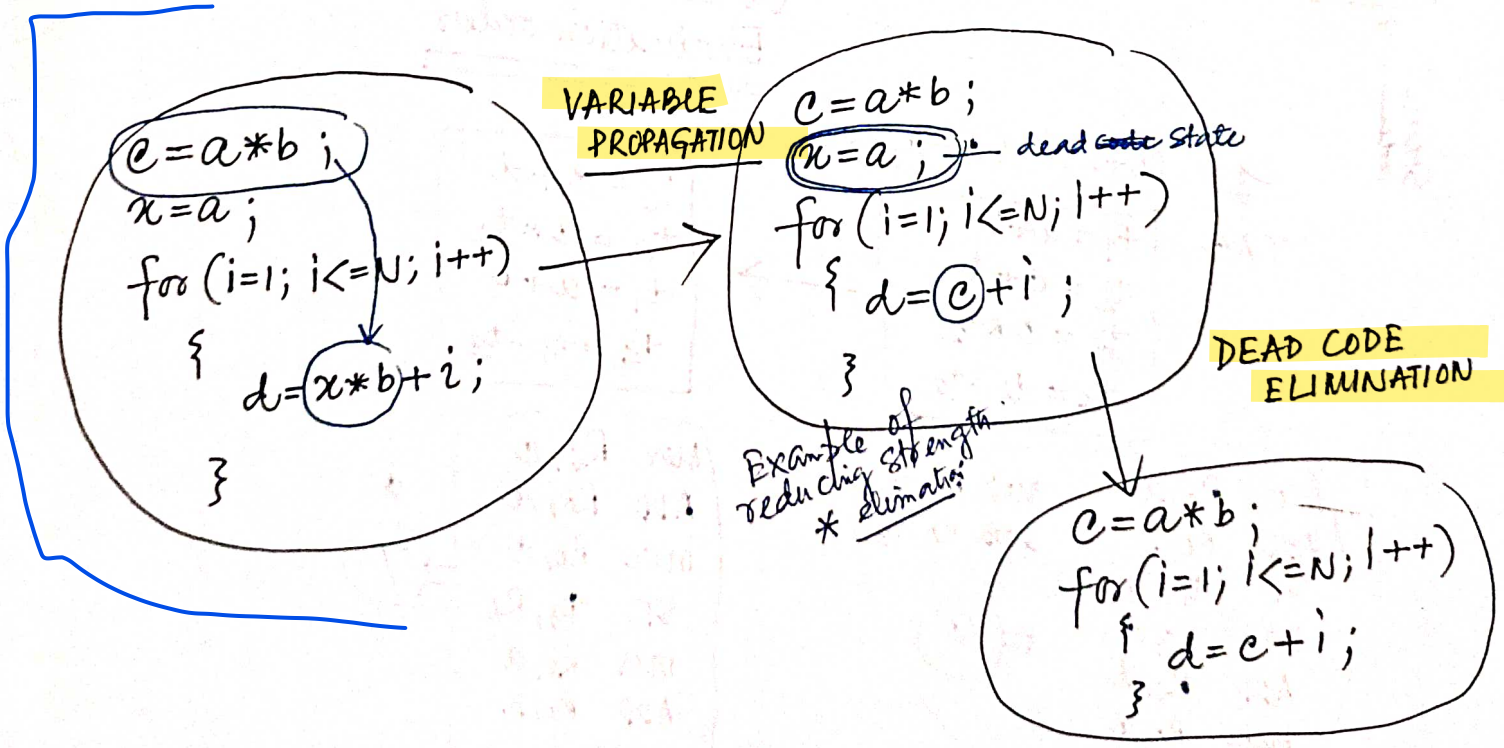
$a = b * c$
$d = b$
$e = d * c$
$b = e$
$f = b + c$



$g = f + d$



$a = d * c$
$f = a + c$
$g = d + f$



Peephole optimization

Redundant Instruction Elimination

```
int f (int x)
{
  int y, z;
  y = 10;
  z = x + y;
  return z;
}
```

Var: x, y, z
instruction: 04

```
int f (int x)
{
  int y;
  y = 10;
  y = y + x;
  return y;
}
```

```
int f (int x)
{
  int y = 10;
  return x + y;
}
```

```
int f (int x)
{
  return x + 10;
}
```

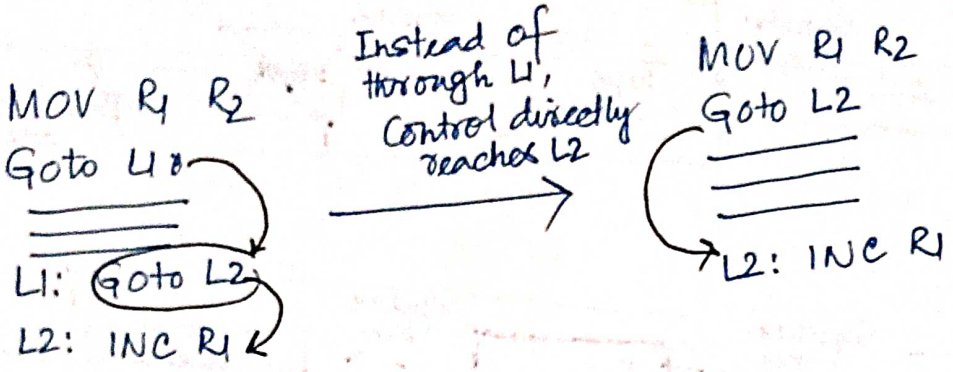
Var: x
instruction: 01

Unreachable Code

```
int f (int x)
{
  return x + 10;
  printf ("value of %d", x + 10);
}
```

Unreachable code
(Can be removed)

Flow of control optimization



Algebric Simplification

$a = a + 0$ ——— Can be removed.

$a = a + 1$ → INC a ✓ (increment a)

a++

$x = 2 * y$ → $x = y + y$ because + is lightweight than *.