COMMON SUB EXPRESSION ELIMINATION

```
a = b * c + g;
d = b * c * e;

T1 = b*c;
a = T1+g;
d = T1*e;
```

CONSTANT FOLDING AND CONSTANT PROPAGATION

```
Function()
{
int x = 14;
int y = 7 - x / 2;
return y * (28 / x + 2);
}

x=14;y=0; return 0;

Function()
{
return 0;
}
```

LOOP INVARIANT CODE MOTION

```
for (int i=0; i<n; ++i) {
    x = y+z;
    a[i] = 6*i + x*x;
}

After moving outside the loop
x=y+z;
T1=x*x;
for (int i=0; i<n; ++i) {
    a[i] = 6*i + T1;
}
```

PARTIAL REDUNDANCY ELIMINATION (PRE)

```
if (some_condition) {
   // some code that does not alter x
   y = x + 4;
}
else {
  // other code that does not alter x
}
for(i=0;i<n;i++)
{
   // other code that does not alter x
}
z = x + 4;
Other code y and z values are not changed;
z=y; Z can be replaced by y;
```

An interesting property of PRE is that it performs (a form of) common subexpression elimination and loop-invariant code motion at the same time.

LOOP UNROLLING AND FUNCTION INLINING

Example 1. Before loop Unrolling

```
1.int x;
2.for (x = 0; x < 10; x++)
3.{
```

```
4.    delete(x);
5.}
```

After loop Unrolling

```
delete(0);
delete(1);
delete(2);
delete(3);
delete(4);
delete(5);
delete(6);
delete(7);
delete(8);
delete(9);
```

Example 2. Before loop Unrolling

```
for (i = 1; i < 9; i++)
{
    if (i mod 2 = 0) then do_even_stuff(i);
    else do_odd_stuff(i);
}
```

After loop Unrolling

```
do_odd_stuff(1);
do_even_stuff(2);
do_odd_stuff(3);
do_even_stuff(4);
do_odd_stuff(5);
do_even_stuff(6);
```

VECTORIZATION AND CONCURRENTIZATION

```
for (i = 0; i < 5; i++)
{
    c[i] = a[i] + b[i];
}
```

One processor - 10 sec

```
c0 = a0+b0; 1 processor
c1 = a1+b1; 2 processor
c2 = a2+b2; 3 processor
c3 = a3+b3; 4 processor
c4 = a4+b4; 5 processor 2 sec
```

UNREACHABLE CODE

```
#include<stdio.h>

void ced55()
{
some code;
}

int main()
{
int a,b,c;
c=a+b;
printf("%d\n",c);

d= a*b*c; //dead code
return 0;

int i=1;
some code(1);
```

```
}

ALGEBRAIC SIMPLIFICATIONS

a = b+b+b+b+b;
a = 5*b;

d = b*c*a*d;
b*c
b*c*a
b*c*a*d

b*c-1processor a*d-2processor
b*c*a*d;

function (int n)
{

}

a= function(x); return value;
b=function(y); return value;(if x=y)
c=function(z); return value;

if x=y=z
a=function(x);
replace b and c with a

suppose b and c are not altering after this code, replacement is possible

if b and c are altering, then replacement not possible.

variable table

variable numbers of a,b,c will be same.

if x!=y!=z
```