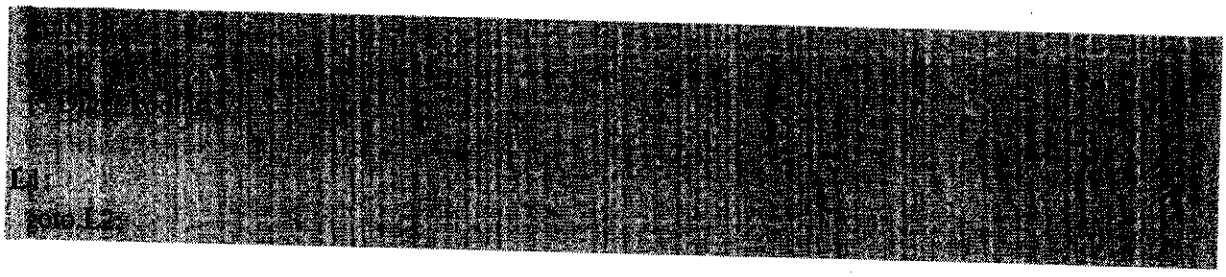
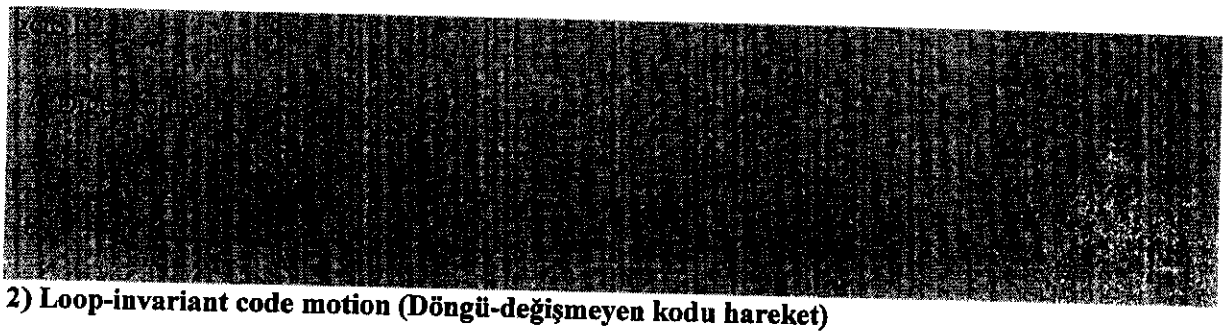
1) Branch Elimination (Şube Eliminasyon)

Aşağıdaki kod parçasındaki L1 ve L2 tek bir L2 ile değiştirilebilir.



Şube eleme yöntemi uygulandıktan sonra.



2) Loop-invariant code motion (Döngü-değişmeyen kodu hareket)

Döngü içerisindeki değişmeyen kod parçaları döngü dışına alınarakekstra tekrarlar engellenir.

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

Değişmeyen kod parçaları çıkarıldıktan sonra

```
x = y + z;
t1 = x * x;
for (int i = 0; i < n; i++) {
   a[i] = 6 * i + t1;
}
```

3) Common subexpression elimination (Ortak alt ifade eliminasyonu)

Hesaplanan değeri tutan tek bir değişken tutarak optimizasyon sağlanır.

Örneğin:

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

b*c değerini tutan bir değişken kullanılırsa.

```
tmp = b * c;
a = tmp + g;
```

```
d = tmp * d;
```

Faydaları:

[redacted]

4) Constant propagation (Sabit Yayılma)

Sabit yayılma derleme zamanında ifadelerde bilinen sabit değerleri ikame işlemidir.

örn1 :

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

x 'leri eleme

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

y'leri eleme x ve y ölü kod pozisyonundadır.

```
int x = 14;
int y = 0;
return 0;
```

örn 2:

x kullanımı kaldırılır.

[redacted]

sabit yayılma uygulandıktan sonra..

[redacted]

5) Dead code elimination (Ölü kod eleme)

Koda erişlilmiyor veya programı etkilemeyen tanımlamalar ölü kodlardır.

1) int foo(void)
{
int a = 24;
int b = 25; /* *Ölü değişkene atama* */
int c;
c = a << 2;
return c;
b = 24; /* *Ulaşılmıyor* */
return 0;
}
	2)
	
	int global;
	void f ()
	{
	int i;
	i = 1; /* dead store */
	global = 1; /* dead store */
	global = 2;
	return;
	global = 3; /* unreachable */
	}

6) Function Inlining

Geri dönen fonksiyonlardaki masrafları azaltmak.

Örnek:

```
int add (int x, int y)
{
 return x + y;
}

int sub (int x, int y)
{
 return add (x, -y);
}
```

```
1)
int sub (int x, int y)
{
 return x + -y;
}
2) En optimize hali
int sub (int x, int y)
{
 return x - y;
}
```

7) Instruction Combining (Kod birleştirme)

Kaynak kodda iki iki talimatı birleştirerek tek bir talimat haline getirme.

```
int i;
  void f (void)
  {
   i + +;
   i + +;
  }
```

```
int i;

void f (void)

{ i + = 2;

}
```

8) Instruction scheduling (Kod Zamanlaması)

Yürütme zamanını en aza indirmek asıl amaçtır. Yürütme zamanı en aza indirmek için verilebilecek en iyi örnek kodun paralelleştirilmesidir. Bu nasıl yapılır: seri bir biçimde, bir birinin işlevini bekleyen kodları en aza indirerek bu işlemleri paralel bir biçimde işlemleri yapması gerçekleştirerek yürütme zamanı azaltılabilir.

9) Interprocedural optimization

Yeniden hesaplama kaldırmak için hesaplama gibi çok sık kullanılan kod parçalarını fonksiyonlaştırmak.

```
Program example;
 integer b;          %A variable "global" to the procedure Silly.
 Procedure Silly(a,x)
```

```
    if x < 0 then a:=x + b else a:=-6;
    End Silly;    %Reference to b, not a parameter, makes Silly "impure" in general.
    integer a,x;          %These variables are visible to Silly only if parameters.
    x:=7; b:=5;
    Silly(a,x); Print x;
    Silly(x,a); Print x;
    Silly(b,b); print b;
    End example;
```

10) Alias Optimization

```
const int const_array[];

void f (int *p, int i)
{
 int x, y;
 const int *q = &const_array[i];
 x = *q;
 *p = 5;
 y = *q;
 g (x, y);
}
```

```
const int const_array[];

void f (int *p, int i)
{
 int x, y;
 const int *q = &const_array[i];
 x = *q;
 *p = 5;
 g (x, x);
}
```

11) Expression Simplification(ifade Basitleştirme)

void f (int i)
{
a[0] = i + 0;
a[1] = i * 0;
	void f (int i)
	{
	a[0] = i;
	a[1] = 0;

```
  a[2] = i - i;
  a[3] = 1 + i + 1;
}
```

```
  a[2] = 0;
  a[3] = 2 + i;
}
```

12) Forward Store

Global değişkenler döngü dışına taşınarak bant genişliği gereksinimleri azaltılabilir.

```
int sum;

void f (void)
{
 int i;

 sum = 0;
 for (i = 0; i < 100; i++)
  sum += a[i];
}
```

```
int sum;

void f (void)
{
 int i;
 register int t;

 t = 0;
 for (i = 0; i < 100; i++)
  t += a[i];
 sum = t;
}
```

13) Loop Fusion

Birbirine benzeyen döndüler içindeki elamanlar birleştirilerek. Döngü yükü azaltılır.

Örnek: İki döngü birleştirilerek tek döngü haline getirilir.

```
for (i = 0; i < 300; i++)
  a[i] = a[i] + 3;

for (i = 0; i < 300; i++)
  b[i] = b[i] + 4;
```

```
for (i = 0; i < 300; i++)
 {
  a[i] = a[i] + 3;
  b[i] = b[i] + 4;
 }
```