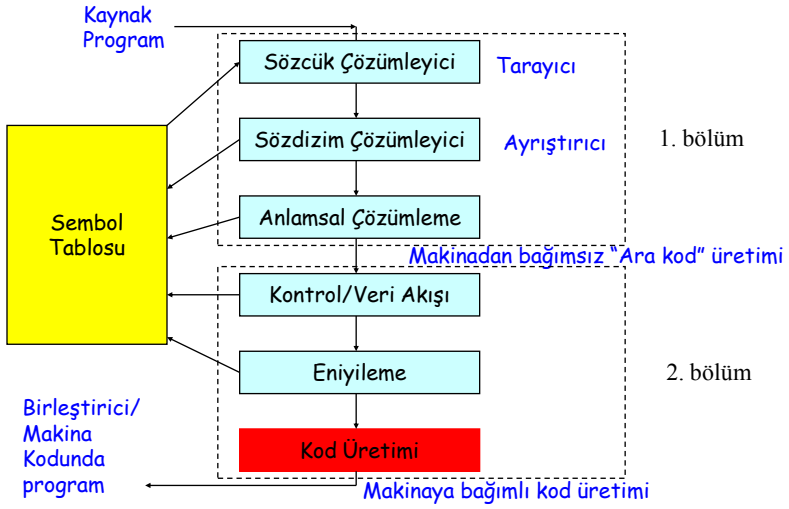
KOD ÜRETİMİ

1

Kod Üretimi



2

Kod Üretici

- Kod üreticinin giriş bilgileri:
  - Kaynak program için üretilen arakod
    - Her türlü hatadan arınmış olarak (örneğin reel ile dizi indislenemez, vs.)
    - Tip sınaması yapılmış olarak (gerekli ise tip dönüşümlerini gerçekleyecek kod eklenmiş halde)
  - Sembol Tablosu bilgileri

3

Amaç Program

- Amaç program şu üç türden birinde olabilir:
  - Makina dilinde ve mutlak (absolute): belleğin sabit bir adresine yerleştirilip hemen yürütülebilir
  - Makine dilinde ve yeniden yüklenebilir (relocatable): farklı zamanlarda derlenen modüllere izin verir. Bir "bağlayıcı-yükleyici" yardımıyla, herhangi bir bellek alanına yüklenip yürütülmeye hazır hale gelir
  - Simgesel dilde: "birleştirici program" (assembler) tarafından birleştirilmeye gerek duyar

4

Komut Seçimi

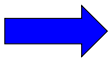
- Üç adresli kod oldukça basit olduğundan, arakod komutlarını makina dili komut dizilerine dönüştürmekte sorun çıkmaz.
- Ancak hedef makinanın donanımı belirleyici olacaktır.
- Dikkate alınması gerekenler şunlardır:
  - Gereksiz işlemlerden kaçınma
  - Uygun makina dili komutlarının seçimi
  - Etkin saklayıcı kullanımı

5

Komut Üretimi

- Eğer etkin kod hedeflenmiyor ise, her işlem için uygun bir komut dizisi (kalıp) tanımlanıp uygulanabilir.

- Örnek:

x=y+z 

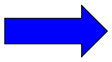
```
MOV y,R0
ADD z,R0
MOV R0,x
```

6

Örnek:

- Örnek uygulama:

a=b+c;
d=a+e;



1. MOV b,R0
2. ADD c,R0
3. MOV R0,a
4. MOV a,R0
5. ADD e,R0
6. MOV R0,d

- 4. numaralı komut gereksiz çünkü a zaten R0 içinde
- 3. numaralı komut da gereksiz eğer bu noktadan sonra "a" tekrar kullanılmayacak ise.

SONUÇ: üretilen kodun çok etkin olduğu söylenemez

7

Temel Blok

- Temel Blok: Kontrol akışının bir uçtan girip, dallanma veya durma gerçekleşmeden diğer uçtan çıktığı, birbirini izleyen komutların oluşturduğu blok.
- Örnek: $\sum_{i=1}^{20} A_i * B_i$
- Temel bloklar incelenerek daha iyi kod üretilebilir

```
1)  p=0
2)  i=1
```
→ 1. temel blok

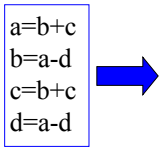
```
3)  t1=4*i
4)  t2=a[t1]
5)  t3=4*i
6)  t4=b[t3]
7)  t5=t2*t4
8)  t6=p*t5
9)  p=t6
10) t7=i+1
11) i=t7
12) if i<=20 goto 3)
```
→ 2. temel blok

```
13) .....
```
→ 3. temel blok

8

Temel Bloklar Üzerinde Eniyileme İşlemleri

- Ortak alt ifadelerin kaldırılması:



--- buradaki b, bir önceki satırda yer alan b'den farklıdır

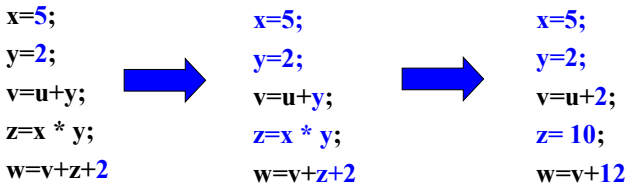
- Ölü kodun kaldırılması:

"x=y+z" gibi bir komut var ancak "x" daha sonra hiç kullanılmamış ise, bu komut bütünüyle silinebilir

9

Temel Bloklar Üzerinde Eniyileme İşlemleri

- Sabitlerin yayılımı: derleme sırasında hesaplanabilir ifadeler belirlenir ve değerleriyle yer değiştirilir



10

Temel Bloklar Üzerinde Eniyileme İşlemleri

- Çevrim değişmezinin çevrim dışına aktarılması

```
for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            .... a[i , j] .....
```

```
for (i=0; i<N; i++) {
    taban= a + (i * dim1);
        for (j=0; j<N; j++) {
            .... taban + j .....
```

11

Eniyileme Örneği: matris çarpımı

- Eniyileme işlemi uygulanmamış matris çarpımı:

```
for i = 1 to N do
    for j = 1 to N do
        A[i,j] =  0
        for k = 1 to N do
            A[i,j] =  A[i,j] + B[i,k] x C[k,j]
        od
    od
od
```

Dikkat: A[i,j] =Adr(A)+((i-1)*K1+(j-1))*K2  6 adet işlem

12

Matris Çarpımı – İşlem Sayısı

- En içte yer alan çevrim (indisi k olan)
  - 24 tamsayı işlemi
  - 3 yükleme (load)
  - 1 kayan noktalı toplama
  - 1 kayan noktalı çarpma
  - 1 yazma (store)

  Toplam: 30 işlem içerir

13

Eniyileme Sonrası Matris Çarpımı

Eniyileme işlemi uygulanmış matris çarpımı:

```
for i = 1 to N do
    b  = & B[i,1]
    for j = 1 to N do
          a  = & A[i,j]   ---------- A[i,j] çevrim değişmezi, çevrim dışına al
          for k = 1 to N do
              *a = *a + *b x C[k,j]
               b = b + Kb ----------- B adresindeki değişim: B[i,k]->B[i,k+1]
          od                          bu değişimi gerçekleyecek kod ekle
    od
od
```

SONUÇ: en içteki çevrim sadece 7 adet tamsayi işleme sahip
(C[k,j]: 6 adet ve b + K_b : 1 adet)

14

Bağlama Bağlı Çeviri ve Saklayıcı Kullanımı

- Makina komutlarının büyük çoğunluğu operanlardan birinin saklayıcı içinde olmasını gerektirir
- Saklayıcı-bellek arasında aktarım zaman alacağından, bilgilerin saklayıcı içinde tutulmasi tercih edilir
- Genel yaklaşım: bilgi saklayıcı içinde yer alıyor ve yakın zamanda kullanılacak ise, saklayıcıda kalmasını sağla
- Sorun, tüm verileri tutacak sayıda saklayıcının bulunmamasıdır.

15

Basit Kod Üreticisi

- Şu bilgilere gerek duyulacaktır:
  - Hangi saklayıcılar kullanımda ve ne içeriyorlar
  - Bir değişkenin değerine nerede ulaşılabilir (bellek, saklayıcı, her ikisi ..)
  - Blok içinde, ilerde, hangi değişkenlere gerek duyulacak
  - Değerleri saklayıcı içinde bulunan hangi değişkenlere blok dışında da erişilecek (bunların değerleri blok sonunda belleğe aktarılmalıdır)

16

Saklayıcı ve Adres Tanımlayıcıları

Saklayıcı Tanımlayıcısı:

Bir saklayıcının o anda ne taşıdığını gösterir. Bir saklayıcı bir anda sıfır veya daha fazla değişkenin değerini taşıyabilir

örnek: a = b; Eğer b R0 içinde yer alıyor ise, a da R0 içinde yer alacaktır → R0:{a,b}

Adres Tanımlayıcısı:

Bir değişkenin geçerli olan en son değerine nerede ulaşılabileceğini gösterir. Bu yer bellek, saklayıcı, yığın, veya bunların birkaçından oluşan bir küme olabilir. Bu bilgi sembol tablosuna eklenip, erişim için uygun yöntem belirlenir.

17

Bir Sonraki Kullanım (Next Use)

- Bir sonraki kullanım (BSK):

  Eğer i. dörtlük x'e değer atıyor, j. dörtlük x'i operand olarak kullanıyor ve i.→j. dörtlüğe kadar x'e atama yapılmadan gelinebiliyor ise, j. dörtlük x'in i. dörtlükte atanan değerini kullanıyordur.

  - Sonuç: x'in BSK vardır ve BSK = j
  - Bir değişkenin BSK'ı varsa, değişkene "canlı", aksi durumda ise "ölü" niteliği kazandırılır

18

Bir Sonraki Kullanım Bulunması

- Bir sonraki kullanımın bulunması:

  Temel blok sondan başa doğru taranır. "i: x = y op z" dörtlüğü ile karşılaşınca:

  1. i. dörtlüğe, x,y ve z için sembol tablosundan elde edilen canlılık ve BSK bilgilerini ekle
  2. Sembol tablosunun x girişine şu atamaları yap:
     i. ölü
     ii. BSK'ı yok
  3. Sembol tablosunun y ve z girişlerine şu atamaları yap:
     i. canlı
     ii. BSK = i

19

GetReg Fonksiyonu

"x = y op z" kodu için x değerini taşıyacak olan L bul.

- Eğer

  i) y değeri başka bir değişkenin de değerini taşımayan bir saklayıcı içinde yer alıyor ise,

  ii) y işlem sonunda canlı olmayacak ve BSK'ı bulunmayacak ise,

  L olarak y'i taşıyan saklayıcıyı getir. y tanımlayıcısını güncelle ve artık L içinde yer almadığını göster.

2. Eğer 1. adım mümkün olmazsa, boş bir saklayıcıyı L olarak getir

20

GetReg Fonksiyonu(2)

3. Eğer 2. adım da mümkün olmazsa,
   i) eğer x'in BSK'ı yoksa, veya
   ii) op saklayıcı gerektiren bir işlem ise,
   o anda kullanılmayan (ancak dolu) bir R saklayıcısı bul ve R'nin içeriğini M bellek gözüne taşı "MOV R, M" (eğer önceden taşınmamış ise). M adres tanımlayıcısını güncelle, ve R'i L olarak getir.

4. Eğer uygun bir saklayıcı bulunamaz ise, x bellek adresini L olarak getir

21

Basit Kod Üretici Algoritma

Her "x = y op z" düzenindeki kod için şu işlemleri yürüt

- GetReg fonksiyonunu çağır ve işlem sonucunun yerleştirileceği yer olan L'i bul. L genellikle bir saklayıcı olacaktır ancak bellek adresi de olabilir.

- "y"e ait adres tanımlayıcısından bulunduğu yer olan y' bilgisini al. Eğer y hem bellek, hem de saklayıcı içinde yer alıyor ise, saklayıcıyı tercih et. Eğer "y" L içinde bulunmuyor ise, "MOV y',L" komutu ile "y"nin bir kopyasını L'e yerleştir.

22

Basit Kod Üretici Algoritma(2)

3. "OP z', L" komutunu üret. z', "z" değerinin bulunduğu yerdir (saklayıcı tercih edilir). "x"e ait adres tanımlayıcısını güncelle ve "x" değerinin L içinde olduğunu göster. Eğer L bir saklayıcı ise, tanımını güncelle ve "x" değerini taşıdığını göster ve "x"i diğer saklayıcı tanımlayıcılarından sil.

4. Eğer "y" ve "z" daha sonra kullanılmıyor ve blok sonunda canlı değilseler, saklayıcı tanımlayıcılarını güncelle ve artık "y" ve "z" değerlerini taşımadıklarını göster.

23

Örnek Kod Üretimi

- Örnek: d = (a-b) + (a-c) + (a-c)
- Üretilen dörtlük dizisi:

```
t1=a-b
t2=a-c
t3=t1+t2
d=t2+t3
```

- d: blok sonunda canlı
- t1,t2,t3: geçici değişkenler, MOV ile taşınmaz iseler, bellekte değiller
- a,b,c: bellekte bulunuyorlar

24

Örnek Kod Üretimi(2)

Komut	Üretilen Kod	Sak. Tanımlayıcı	Adres Tanımla.
t1=a-b	MOV a,R0 SUB b,R0	R0: {t1}	t1:{R0}
t2=a-c	MOV a, R1 SUB c, R1	R0: {t1} R1: {t2}	t1:{R0} t2:{R1}
t3=t1+t2	ADD R1,R0	R0: {t3} R1: {t2}	t2:{R1} t3:{R0}
d=t2+t3	ADD R1,R0 MOV R0, d	R0: {d}	d:{R0 ve bellek}

25