Geração de Código
Cap. 8

Introdução

- Fase final para um compilador
- Entrada é uma representação intermediária do código fonte e a saída é um programa para ser executado em uma máquina alvo
- O programa para a máquina alvo deve preservar o significado semântico do programa fonte e deve utilizar eficientemente os recursos do processador
  - Deve executar de maneira eficiente!
- Desafio: gerar código ótimo para uma máquina alvo é um problema indecidível
  - Muitos subproblemas são intratáveis
- Nesta etapa, utilizaremos muitos conceitos de Arq. de Computadores!

Blocos Básicos

- **Ajudam a entender e manipular determinadas etapas da geração de código**
- **Blocos básicos são seqüências maximais de instruções de 3-end. consecutivas**
- **Código da RI (em 3-end.) é dividido em blocos de instruções com as seguintes características:**
  - Fluxo de controle entra no bloco básico através da 1a. Instrução desse bloco
  - Não existem desvios ou *labels* no “meio” de um bloco básico
  - O controle de execução deve sair do bloco básico apenas pela última instrução

Blocos Básicos

- **Algoritmo para construção de blocos básicos**

- Entrada: seqüência de instruções de 3-end
- Saída: lista de blocos básicos
- Método: determina as instruções que são líderes, a primeira instrução de um bloco básico. As instruções que são líderes determinam a criação de novos blocos básicos. As demais instruções são incluídas, na seqüência, em blocos básicos já existentes. As regras para criar líderes são:
  - A 1a. Instrução do programa é líder
  - Qualquer instrução que é alvo de um desvio condicional ou incondicional é um líder
  - Qualquer instrução que segue (está após) um desvio condicional ou incondicional é um líder

Blocos Básicos

- **Exemplo: Determinar os blocos básicos para o seguinte código:**

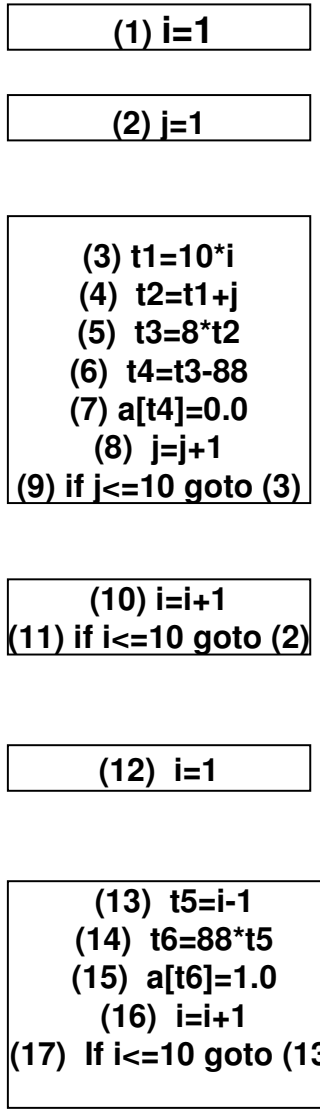
```
(1) i=1
(2) j=1
(3)  t1=10*i
(4)  t2=t1+j
(5)  t3=8*t2
(6)  t4=t3-88
(7)  a[t4]=0.0
(8)  j=j+1
(9)  if j<=10 goto (3)
(10)  i=i+1
(11)  if i<=10 goto (2)
(12)  i=1
(13)  t5=i-1
(14)  t6=88*t5
(15)  a[t6]=1.0
(16)  i=i+1
(17)  If i<=10 goto (13)
```

```
(1) i=1

(2) j=1

(3) t1=10*i
(4)  t2=t1+j
(5)  t3=8*t2
(6)  t4=t3-88
(7) a[t4]=0.0
(8)  j=j+1
(9) if j<=10 goto (3)

(10) i=i+1
(11) if i<=10 goto (2)

(12)  i=1

(13)  t5=i-1
(14)  t6=88*t5
(15)  a[t6]=1.0
(16)  i=i+1
(17)  If i<=10 goto (13)
```

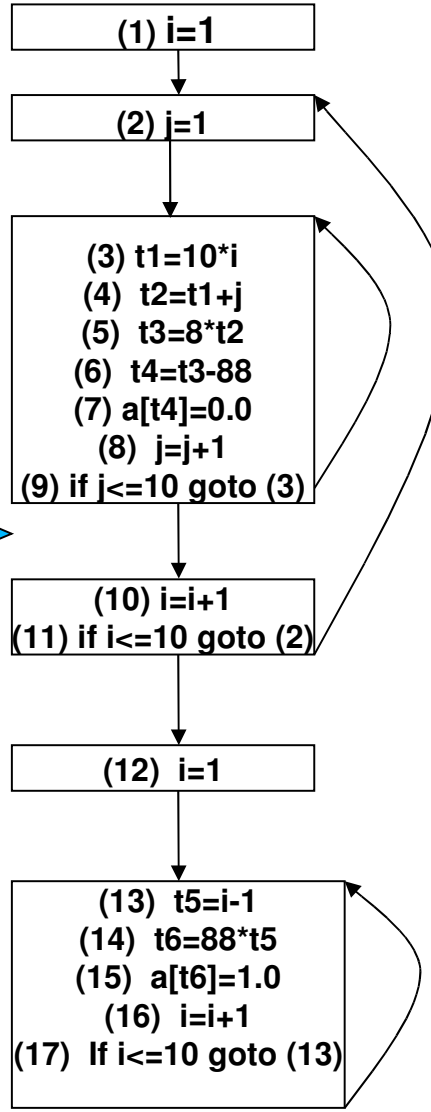
Blocos Básicos

- **Exemplo: Determinar os blocos básicos para o seguinte código:**

(1) i=1
(2) j=1
(3) t1=10*i
(4) t2=t1+j
(5) t3=8*t2
(6) t4=t3-88
(7) a[t4]=0.0
(8) j=j+1
(9) if j<=10 goto (3)
(10) i=i+1
(11) if i<=10 goto (2)
(12) i=1
(13) t5=i-1
(14) t6=88*t5
(15) a[t6]=1.0
(16) i=i+1
(17) If i<=10 goto (13)



Formação do CFG



Informação de “próximo-uso” (next-use)

- Uma informação importante para a geração de código é saber quando o valor variável será usado
  - Se o valor que está atualmente em um registrador não será usado novamente, então, esse registrador pode ser reusado por outra variável

- Use de um nome em 3-end é definido como segue:
  - Em um ponto (declaração) i, a variável x recebe um valor; Se a declaração j tem x como um de seus operandos e o controle pode ir de i para j, através de um caminho que não altera o valor de x, então dizemos que j usa x computado em i

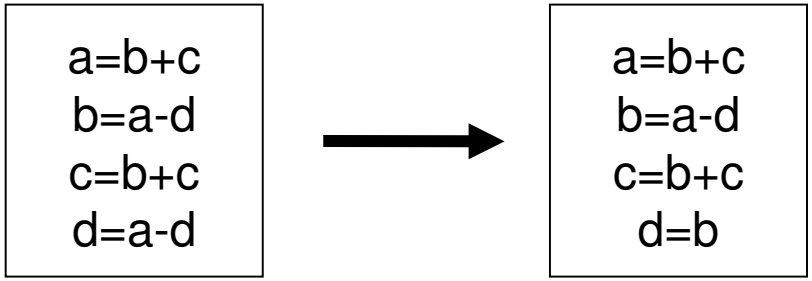
- O que queremos é determinar para cada declaração x=y op z, qual é o próximo uso de x,y e z

Informação de “próximo-uso” (next-use)

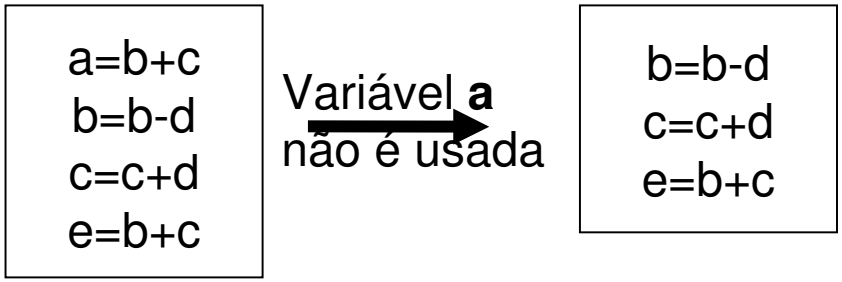
- **Algoritmo: determinar a longevidade (liveness) e informação de próximo-uso para cada declaração em um bloco básico**
- Entrada: Um bloco básico B com código em 3-end. Assumimos que a TS pode armazenar a informação de próximo-uso
- Saída: em cada declaração ***i***: x=y op z, determinamos longevidade e próximo-uso de x, y e z
- Método: Iniciar na última declaração de B e pesquisar por todas as declarações de B até a primeira instrução. Em cada declaração ***i***: x=y op z, fazer:
  1. Armazenar nas estruturas de ***i*** a informação atual da TS sobre o próximo-uso e longevidade de x, y e z
  2. Na TS, alterar a informação de longevidade e próximo-uso de x para 0 e 0
  3. Na TS, alterar y e z para 1 e na informação de próximo-uso indicar a posição de ***i***

Otimizações básicas em nível de BB

- *Common Subexpression Elimination*: eliminação de expressões pode ser detectada observando a construção de DAGs. Para um nó M a ser adicionado no DAG, já existe um nó N com os mesmos filhos, mesma ordem dos filhos e mesmo operador



- *Dead Code Elimination*: Eliminação de variáveis que não são mais utilizadas



Otimizações básicas em nível de BB

- Identidades Algébricas: consiste em substituir instruções que envolvem operações aritméticas por operações mais simples e com o mesmo valor semântico

x+0=0+x=x
x-0=x
x*1=1*x=x
x/1=x

- *Redução de força*

x^2=x*x
2*x=x+x
x/2=x*0.5

- *Expressões constantes: avaliar expressões com constantes em tempo de compilação e substituí-las pelo seu valor*

2*3.14=6.28