

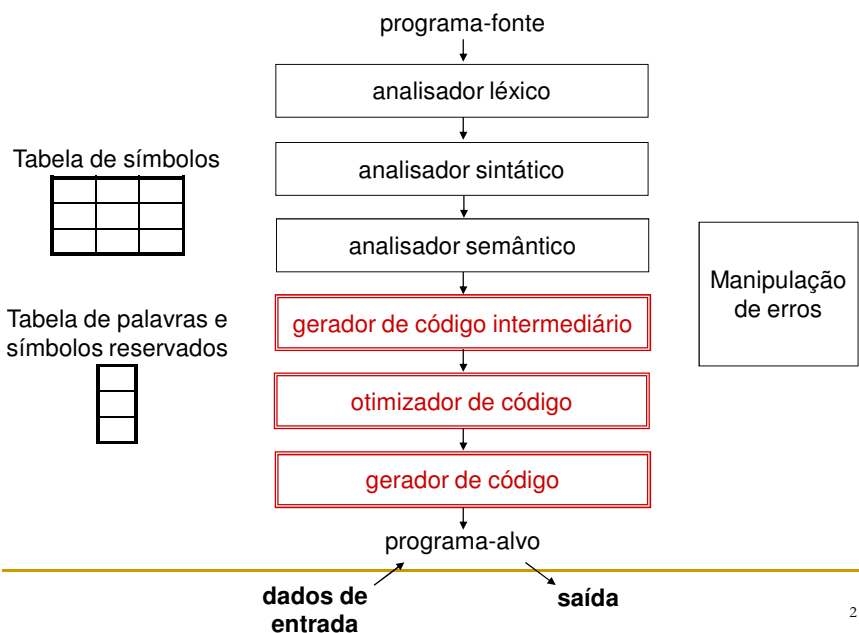
Geração e Otimização de Código

Representação de código intermediária
Código de três endereços, P-código
Técnicas para geração de código
Otimização de código

Prof. Thiago A. S. Pardo
taspardo@icmc.usp.br

1

Estrutura geral de um compilador



Geração de código

- A geração de código é **uma das tarefas mais complexas** do compilador
 - Depende da linguagem-fonte, da máquina-alvo e seu ambiente de execução e sistema operacional
- Portanto, é **conveniente dividir esta etapa** em etapas menores
 - Geração de código intermediário, geração de código objeto, otimização
 - Compilação de várias passagens

3

Geração de código intermediário

- Até o momento, temos utilizado a **árvore sintática** (mesmo que implícita) como representação interna por excelência, juntamente com as informações da tabela de símbolos
- Entretanto, ela **não corresponde em nada com o código-objeto** que queremos gerar, além de poder ser muito complexa
- Pode ser interessante gerar um **código mais próximo do objeto** antes de fazer a tradução final
 - **Código intermediário**

4

Geração de código intermediário

■ Características

- ❑ Pode assumir muitas formas
- ❑ Em geral, é alguma forma de linearização da árvore sintática
- ❑ Pode ser muito abstrato (como a árvore sintática) ou mais próximo do código-objeto
- ❑ Pode ou não usar informações sobre a máquina-alvo e o ambiente de execução (como disponibilidade de registradores, tamanho dos tipos, etc.)
- ❑ Pode ou não incorporar informações da tabela de símbolos (como escopo, níveis de aninhamento, etc.), dispensando ou não a tabela na geração de código-objeto

5

Geração de código intermediário

■ Vantagens

- ❑ Bom para quando se quer produzir código-objeto extremamente eficiente
- ❑ Se for genérico o suficiente, pode ser a base para geração de código para várias máquinas
 - Aumenta a portabilidade (o tradutor do código intermediário para o objeto ainda é necessário)

6

Geração de código intermediário

- Veremos aqui as características gerais de 2 formas populares de código intermediário
 - ▣ Código de três endereços
 - ▣ P-código
 - Também possuem diversas formas distintas na literatura e na prática de compiladores

7

Código de três endereços

8

Código de três endereços

- **Instrução mais básica:** $x = y \text{ op } z$
 - O operador op é aplicado a y e z e o resultado é armazenado em x
- Origem do nome desse código: **três endereços de memória** envolvidos no cômputo
 - y e z (mas não x) podem ser constantes ou literais, na realidade
- Exemplo

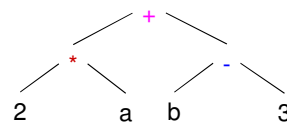
$2 * a + (b - 3) \longrightarrow$
 $t1 = 2 * a$
 $t2 = b - 3$
 $t3 = t1 + t2$

9

Código de três endereços

- **Atenção**
 - É necessário que se gerem **variáveis temporárias** (t1, t2 e t3 no exemplo) com nomes diferentes dos possíveis identificadores no programa
 - As variáveis podem ser mantidas na memória ou em registradores
 - As **variáveis temporárias** correspondem aos nós internos da árvore sintática subjacente à expressão

$2 * a + (b - 3) \longrightarrow$
 $t1 = 2 * a$
 $t2 = b - 3$
 $t3 = t1 + t2$

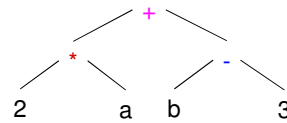


10

Código de três endereços

- No exemplo, o código intermediário corresponde à **linearização da árvore** quando ela é percorrida em **pós-ordem**
 - Qualquer alternativa é possível, dependendo somente da semântica da linguagem
 - Dependendo da alternativa, o código intermediário pode mudar (por exemplo, faz-se primeiro a subtração e depois a soma)

$2 * a + (b - 3) \Rightarrow$
 $t1 = 2 * a$
 $t2 = b - 3$
 $t3 = t1 + t2$



11

Código de três endereços

- Para acomodar todas as possibilidades de uma linguagem de programação, o **código precisa ser mais flexível**
 - **Nem sempre se têm três elementos**
 - Essa é uma das razões da diversidade de formas de código intermediário encontradas
 - Por exemplo, para operadores unários, pode-se ter o código $t2 = -t1$
 - Outras possibilidades: cópia, salto incondicional e condicional, índices, endereços

12

Código de três endereços

- Um exemplo de uma linguagem fictícia e seu possível código intermediário

```
{ Programa exemplo
  -- computa o fatorial
}
read x; { inteiro de entrada }
if 0 < x then { não computa se x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact { fatorial de x como saída }
end
```



```
read x
t1 = x > 0
if_false t1 goto L1
fact = 1
label L2
t2 = fact * x
fact = t2
t3 = x - 1
x = t3
t4 = x == 0
if_false t4 goto L2
write fact
label L1
halt
```

13

Código de três endereços

- Estrutura de dados
 - ❑ Nem sempre os códigos são armazenados textualmente como no exemplo
 - ❑ Listas encadeadas e arranjos podem ser usadas
 - ❑ Em vez de armazenar nomes de identificadores, pode ser interessante guardar ponteiros para os identificadores na tabela de símbolos (caso ela esteja sendo usada)
 - ❑ Muitas vezes, a noção de tuplas é utilizada (implementadas via registros, por exemplo)
 - Haverá, no máximo, quádruplas, sendo que algumas posições podem ficar vazias

14

Código de três endereços

■ Exemplo de representação em tuplas

```
read x
t1 = x > 0
if_false t1 goto L1
fact = 1
label L2
t2 = fact * x
fact = t2
t3 = x - 1
x = t3
t4 = x == 0
if_false t4 goto L2
write fact
label L1
halt
```



```
(rd,x,_,_)
(gt,x,0,t1)
(if_f,t1,L1,_)
(asn,1,fact,_)
(lab,L2,_,_)
(mul,fact,x,t2)
(asn,t2,fact,_)
(sub,x,1,t3)
(asn,t3,x,_)
(eq,x,0,t4)
(if_f,t4,L2,_)
(wri,fact,_,_)
(lab,L1,_,_)
(halt,_,_,_)
```

15

P-código

16

P-código

■ Um pouco da história

- ❑ Criado como código-alvo de montagem padrão para os compiladores de PASCAL dos anos 70 e 80
- ❑ Criado como código para uma máquina hipotética chamada P-máquina, sendo que diversos montadores para máquinas reais foram disponibilizados
 - Aumento da portabilidade
- ❑ Foi redescoberto como um bom código intermediário

17

P-código

■ Assume que

- ❑ O ambiente de execução é baseado em pilhas
- ❑ Dados do ambiente de execução são conhecidos (como tamanho de tipos)

■ Exemplo de P-código para $2*a+(b-3)$

```
ldc 2      ; carrega constante 2
lod a      ; carrega valor da variável a
mpi        ; multiplicação de inteiros
lod b      ; carrega valor da variável b
ldc 3      ; carrega constante 3
sbi        ; subtração de inteiros
adi        ; adição de inteiros
```

18

P-código

- Outro exemplo: $x := y + 1$

```
lda x      ; carrega endereço de x
lod y      ; carrega valor de y
ldc 1      ; carrega constante 1
adi        ; adição
sto        ; armazena topo no endereço
           ; abaixo do topo & retira os dois
```

19

P-código

- Exemplo para o programa fictício completo

```
{ Programa exemplo
-- computa o fatorial
}
read x; { inteiro de entrada }
if 0 < x then { não computa se x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact { fatorial de x como saída }
end
```

20

P-código

■ Exemplo para o programa fictício completo

<pre>{ Programa exemplo -- computa o fatorial } read x; { inteiro de entrada } if 0 < x then { não computa se x fact := 1; repeat fact := fact * x; x := x - 1 until x = 0; write fact { fatorial de x } end</pre>	<pre>ldax ; carrega endereço de x rdi ; lê um inteiro, armazena no ; endereço no topo da pilha (& o retira) lodx ; carrega o valor de x ldc 0 ; carrega a constante 0 grt ; retira da pilha e compara os dois valores do topo ; coloca na pilha o resultado booleano fjp L1 ; retira o valor booleano, salta para L1 se falso lda fact ; carrega endereço de fact ldc 1 ; carrega constante 1 sto ; retira dois valores, armazena primeiro ; em endereço representado pelo segundo lab L2 ; definição do rótulo L2 lda fact ; carrega endereço de fact lod fact ; carrega valor de fact lod x ; carrega valor de x mpi ; multiplica sto ; armazena topo em endereço do segundo & retira lda x ; carrega endereço de x lod x ; carrega valor de x ldc 1 ; carrega constante 1 sbi ; subtrai sto ; armazena (como no caso anterior) lod x ; carrega valor de x ldc 0 ; carrega constante de 0 equ ; teste de igualdade fjp L2 ; salta para L2 se falso lod fact ; carrega valor de fact wri ; escreve topo da pilha & retira lab L1 ; definição do rótulo L1 stp</pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

P-código

■ Em relação ao código de três endereços

- Mais próximo do código de máquina
- Exigem menos endereços (quase todas as instruções vistas até agora tinham 1 ou nenhum endereço)
- Menos compacto (isto é, tem mais instruções)
- Não é “auto-suficiente”, pois assume a existência de uma pilha
- Pode ser representado e implementado da mesma forma que o código de três endereços

Gerando código

23

Geração de código

- Como se gera o código (intermediário ou final) para um programa?
 1. Gramática de atributos
 2. Procedimentos/funções de geração
 - Baseados na gramática de atributos definida
 - Ou *ad hoc*

24

Gramática de atributos

- Exemplo de gramática de atributos para geração do P-código (atributo pcod)

$\text{exp} \rightarrow \text{id} = \text{exp} \mid \text{aexp}$
 $\text{aexp} \rightarrow \text{aexp} + \text{fator} \mid \text{fator}$
 $\text{fator} \rightarrow (\text{exp}) \mid \text{num} \mid \text{id}$



Valor da cadeia
 Concatena e não pula linha
 Concatena e pula linha

Regra Gramatical	Regras Semânticas
$\text{exp}_1 \rightarrow \text{id} = \text{exp}_2$	$\text{exp}_1.\text{pcod} = \text{"lda"} \parallel \text{id.strval} \parallel \text{"stn"}$
$\text{exp} \rightarrow \text{aexp}$	$\text{exp}.\text{pcod} = \text{aexp}.\text{pcod}$
$\text{aexp}_1 \rightarrow \text{aexp}_2 + \text{fator}$	$\text{aexp}_1.\text{pcod} = \text{aexp}_2.\text{pcod} \parallel \text{fator}.\text{pcod} \parallel \text{"adi"}$
$\text{aexp} \rightarrow \text{fator}$	$\text{aexp}.\text{pcod} = \text{fator}.\text{pcod}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{pcod} = \text{exp}.\text{pcod}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{pcod} = \text{"ldc"} \parallel \text{num.strval}$
$\text{fator} \rightarrow \text{id}$	$\text{fator}.\text{pcod} = \text{"lod"} \parallel \text{id.strval}$

Gramática de atributos

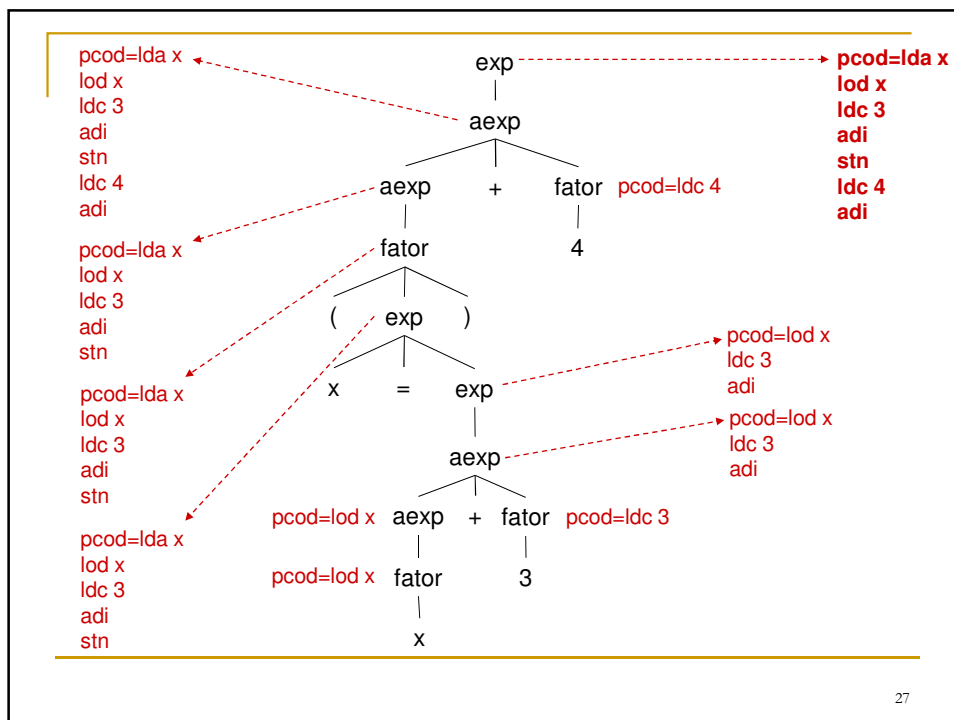
- Exercício: analise a cadeia $(x=x+3)+4$

$\text{exp} \rightarrow \text{id} = \text{exp} \mid \text{aexp}$
 $\text{aexp} \rightarrow \text{aexp} + \text{fator} \mid \text{fator}$
 $\text{fator} \rightarrow (\text{exp}) \mid \text{num} \mid \text{id}$



Valor da cadeia
 Concatena e não pula linha
 Concatena e pula linha

Regra Gramatical	Regras Semânticas
$\text{exp}_1 \rightarrow \text{id} = \text{exp}_2$	$\text{exp}_1.\text{pcod} = \text{"lda"} \parallel \text{id.strval} \parallel \text{"stn"}$
$\text{exp} \rightarrow \text{aexp}$	$\text{exp}.\text{pcod} = \text{aexp}.\text{pcod}$
$\text{aexp}_1 \rightarrow \text{aexp}_2 + \text{fator}$	$\text{aexp}_1.\text{pcod} = \text{aexp}_2.\text{pcod} \parallel \text{fator}.\text{pcod} \parallel \text{"adi"}$
$\text{aexp} \rightarrow \text{fator}$	$\text{aexp}.\text{pcod} = \text{fator}.\text{pcod}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{pcod} = \text{exp}.\text{pcod}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{pcod} = \text{"ldc"} \parallel \text{num.strval}$
$\text{fator} \rightarrow \text{id}$	$\text{fator}.\text{pcod} = \text{"lod"} \parallel \text{id.strval}$



Gramática de atributos

- Exemplo de gramática de atributos para geração do código de 3 endereços (atributo tacode)

$\text{exp} \rightarrow \text{id} = \text{exp} \mid \text{aexp}$
 $\text{aexp} \rightarrow \text{aexp} + \text{fator} \mid \text{fator}$
 $\text{fator} \rightarrow (\text{exp}) \mid \text{num} \mid \text{id}$



Gera novo nome temporário, que é guardado no atributo "nome"

Cadeia vazia

Regra Gramatical	Regras Semânticas
$\text{exp}_1 \rightarrow \text{id} = \text{exp}_2$	$\text{exp}_1.\text{name} = \text{exp}_2.\text{name}$ $\text{exp}_1.\text{tocode} = \text{exp}_2.\text{tocode} ++$ $\text{id}.\text{strval} "=" \text{exp}_2.\text{name}$
$\text{exp} \rightarrow \text{aexp}$	$\text{exp}.\text{name} = \text{aexp}.\text{name}$ $\text{exp}.\text{tocode} = \text{aexp}.\text{tocode}$
$\text{aexp}_1 \rightarrow \text{aexp}_2 + \text{fator}$	$\text{aexp}_1.\text{name} = \text{newtemp}()$ $\text{aexp}_1.\text{tocode} =$ $\text{aexp}_2.\text{tocode} ++ \text{fator}.\text{tocode}$ $++ \text{aexp}_1.\text{name} "=" \text{aexp}_2.\text{name}$ $ "+" \text{fator}.\text{name}$
$\text{aexp} \rightarrow \text{fator}$	$\text{aexp}.\text{name} = \text{fator}.\text{name}$ $\text{aexp}.\text{tocode} = \text{fator}.\text{tocode}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{name} = \text{exp}.\text{name}$ $\text{fator}.\text{tocode} = \text{exp}.\text{tocode}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{name} = \text{num}.\text{strval}$ $\text{fator}.\text{tocode} = ""$
$\text{fator} \rightarrow \text{id}$	$\text{fator}.\text{name} = \text{id}.\text{strval}$ $\text{fator}.\text{tocode} = ""$

Gramática de atributos

- Exercício em duplas para entregar: analise a cadeia $(x=x+3)+4$

$\text{exp} \rightarrow \text{id} = \text{exp} \mid \text{aexp}$
 $\text{aexp} \rightarrow \text{aexp} + \text{fator} \mid \text{fator}$
 $\text{fator} \rightarrow (\text{exp}) \mid \text{num} \mid \text{id}$



Gera novo nome temporário, que é guardado no atributo "nome"

Cadeia vazia

Regra Gramatical	Regras Semânticas
$\text{exp}_1 \rightarrow \text{id} = \text{exp}_2$	$\text{exp}_1.\text{name} = \text{exp}_2.\text{name}$ $\text{exp}_1.\text{tocode} = \text{exp}_2.\text{tocode} ++ \text{id}.\text{strval} "=" \text{exp}_2.\text{name}$
$\text{exp} \rightarrow \text{aexp}$	$\text{exp}.\text{name} = \text{aexp}.\text{name}$ $\text{exp}.\text{tocode} = \text{aexp}.\text{tocode}$
$\text{aexp}_1 \rightarrow \text{aexp}_2 + \text{fator}$	$\text{aexp}_1.\text{name} = \text{newtemp}()$ $\text{aexp}_1.\text{tocode} =$ $\text{aexp}_2.\text{tocode} ++ \text{fator}.\text{tocode}$ $++ \text{aexp}_1.\text{name} "=" \text{aexp}_2.\text{name}$ $ "+" \text{fator}.\text{name}$
$\text{aexp} \rightarrow \text{fator}$	$\text{aexp}.\text{name} = \text{fator}.\text{name}$ $\text{aexp}.\text{tocode} = \text{fator}.\text{tocode}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{name} = \text{exp}.\text{name}$ $\text{fator}.\text{tocode} = \text{exp}.\text{tocode}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{name} = \text{num}.\text{strval}$ $\text{fator}.\text{tocode} = ""$
$\text{fator} \rightarrow \text{id}$	$\text{fator}.\text{name} = \text{id}.\text{strval}$ $\text{fator}.\text{tocode} = ""$

Geração de código

Gramática de atributos

- Útil para apresentar com clareza as relações entre as seqüências de código de diferentes partes da árvore sintática e para comparar diferentes métodos de geração de código...
- mas não é prática, pois
 - Grande quantidade de cópia de cadeias e desperdício de memória
 - É desejável que se gerem pequenos trechos de código a medida que se analisa a cadeia e que se gravem esses trechos em um arquivo ou em outra estrutura de dados
 - A gramática de atributos pode ficar muito complicada

Procedimentos/funções para geração de código

- Com base na gramática de atributos e na árvore sintática
 - Similar à forma que se fazia a análise semântica
 - Aparência geral de um procedimento nessa linha (que deve ser instanciado de acordo com a geração de código pretendida)

```
procedure genCod(T: nó-árvore);
begin
  if T não é nulo then
    gere código de preparação para T;
    genCod(filho à esquerda de T);
    gere código de preparação para T;
    genCod(filho à direita de T);
    gere código para implementação de T;
end;
```

31

Procedimentos/funções para geração de código

- Com base na gramática de atributos e na árvore sintática
 - Similar à forma que se fazia a análise semântica
 - Exemplo instanciado básico para a gramática anterior

```
procedure genCod(T: nó-árvore);
begin
  if T não é nulo then
    if ('+') then
      genCode(t->leftchild);
      genCode(t->rightchild);
      write("adi");
    else if ('=') then
      write("lda"+id.strval);
      genCode(t->leftchild);
      write("strn");
    else if ('num') then write("ldc"+num.strval);
    else if ('id') then write("lod"+id.strval);
end;
```


Procedimentos/funções para geração de código

- Solução *ad hoc*

- Geração de código amarrada aos procedimentos sintáticos

```
função fator(Seg): string;
Início
  declare cod: string;
  se (simbolo='(') então início
    obtem_simbolo(cadeia,simbolo);
    cod=exp(Seg+'(');
    se (simbolo=')')
      então obtem_simbolo(cadeia,simbolo);
      senão ERRO(Seg);
    fim
  senão se (simbolo='num') então início
    cod="ldc "+cadeia;
    obtem_simbolo(cadeia,simbolo);
    fim
  senão se (simbolo='id') então início
    cod="lod "+cadeia;
    obtem_simbolo(cadeia,simbolo);
    fim
  senão ERRO(Seg);
  retorne cod;
fim
```

Procedimentos/funções para geração de código

- Solução *ad hoc*

- Geração de código amarrada nos procedimentos sintáticos

- Alternativamente, podem-se chamar **outras funções e procedimentos** (com ou sem parâmetros que indiquem que código gerar) **que gerem o código**, em vez de embutir a geração diretamente no próprio procedimento sintático

Geração de código

■ Ponto importante

- O código-alvo (objeto) desejado pode ser similar ao P-código ou outra linguagem de montagem qualquer (caso da LALG)
 - Nesse caso, um **montador** também é necessário

35

Geração de código

■ Geração de código-alvo a partir do código intermediário

- Passo necessário se código intermediário é utilizado e é diferente do código-alvo desejado
- **Pode ser um processo complicado** se o código intermediário não contém informações da máquina-alvo e de seu ambiente de execução

36

Geração de código

- Geração de código-alvo a partir do código intermediário
 - Em geral, dependendo do código com que se está lidando, se requer uma ou ambas destas técnicas: **expansão de macros** e **simulação estática**
 - **Expansão de macros:** encara cada linha de código como uma macro e a substitui por uma porção de código correspondente do código-alvo
 - Pode gerar código ineficiente ou redundante
 - **Simulação estática:** requer uma simulação direta dos efeitos do código intermediário e a geração do código-alvo correspondente a estes efeitos
 - Pode ser muito simples ou muito sofisticada

37

Geração de código

- **Exemplo de expansão de macro**
 - Supondo que temos código de três endereços como código intermediário e queremos o P-código como código-alvo
 - A expressão $a=b+c$ pode ser substituída pela sequência abaixo
 - lda a
 - lod b (ou ldc b, se b é constante)
 - lod c (ou ldc c, se c é constante)
 - adi
 - sto

38

Geração de código

■ Exemplo de expansão de macro

- Supondo que temos código de três endereços como código intermediário e queremos o P-código como código-alvo

$t1 = x + 3$
 $x = t1$
 $t2 = t1 + 4$



```
lda t1
lod x
ldc 3
adi
sto
lda x
lod t1
sto
lda t2
lod t1
ldc 4
adi
sto
```

39

Geração de código

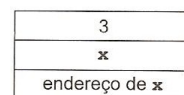
■ Exemplo de simulação estática

- Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo

- Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

```
lda x
lod x
ldc 3
adi
stn
ldc 4
adi
```

Processando as
3 primeiras instruções



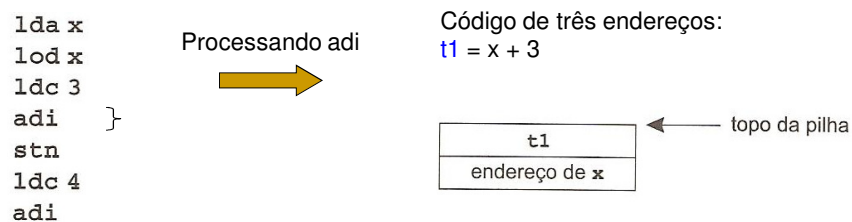
← topo da pilha

40

Geração de código

■ Exemplo de simulação estática

- Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
- Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

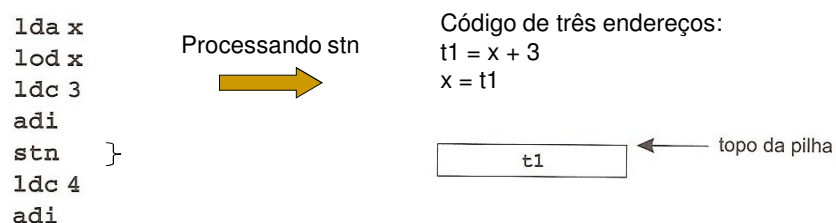


41

Geração de código

■ Exemplo de simulação estática

- Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
- Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

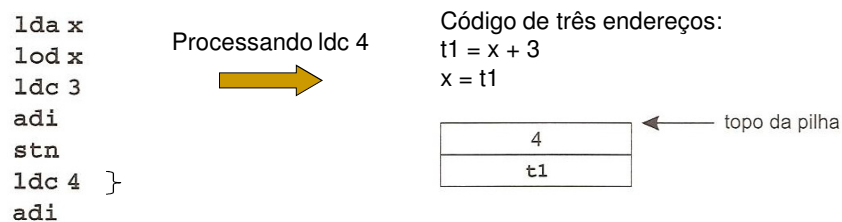


42

Geração de código

■ Exemplo de simulação estática

- Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
- Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento

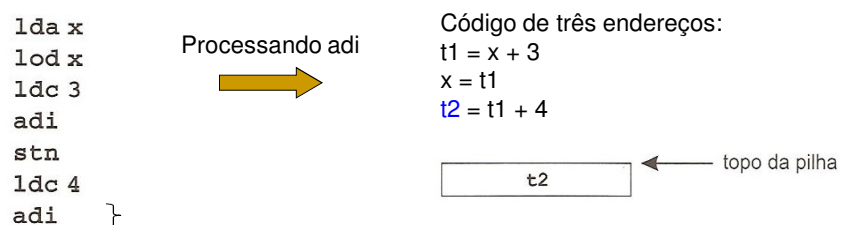


43

Geração de código

■ Exemplo de simulação estática

- Supondo que temos P-código como código intermediário e queremos o código de três endereços como código-alvo
- Precisamos gerar os temporários e, por isso, é necessário que se simule a pilha e seu funcionamento



44

Otimizando código

45

Otimização

- **Definição**: processo de se melhorar a qualidade do código gerado pelo compilador
 - Nome enganoso, pois é raro se gerar o código ótimo
- **Qualidade de código** pode ser medida por vários ângulos
 - Quais?

46

Otimização

- **Definição:** processo de se melhorar a qualidade do código gerado pelo compilador
 - Nome enganoso, pois é raro se gerar o código ótimo
- **Qualidade de código** pode ser medida por vários ângulos
 - Velocidade
 - Tamanho do código
 - Memória utilizada para temporários

47

Otimização

- Análise do código e de sua execução permitem determinar o que otimizar
 - Algumas técnicas de otimização são simples e impactantes
 - Algumas são caras (em termos de tempo), complexas e com pouco ganho
 - Há muitas técnicas de otimização, algumas boas para algumas linguagens, outras não
 - O projetista do compilador deve fazer esse julgamento!

48

Otimização

■ Principais fontes de otimização de código

- **Alocação de registradores:** quanto mais e melhor usados, maior a velocidade
- **Operações desnecessárias**
 - Expressões de valores invariáveis, mas avaliadas várias vezes no programa
 - Código inatingível ou morto (por exemplo, if com condição falsa sempre ou procedimento nunca ativado)
 - Salto para outros saltos (poderia ir direto)

49

Otimização

■ Principais fontes de otimização de código

- **Operações caras**
 - Redução de força: trocam-se expressões caras por mais baratas (por exemplo, multiplicação por 2 pode ser implementada como uma transposição)
 - Empacotamento e propagação de constantes: reconhecimento e troca de expressões constantes pelo valor calculado (por exemplo, troca-se 2+5 por 7)
 - Procedimentos: pode-se melhorar fazendo-se alinhamento de procedimentos (inserção de seus códigos no corpo do programa), identificação e remoção de recursão de cauda
 - Uso de dialetos de máquina: instruções mais baratas oferecidas por máquinas específicas
- **Previsão do comportamento do programa**
 - Conhecimento do comportamento do programa para otimizar saltos, laços e procedimentos ativados mais frequentemente

50

Otimização

■ Tipos de otimização

- Quanto ao instante em que são realizadas
- Quanto à área do programa em que se aplicam

■ Quanto ao instante

- Empacotamento de constantes pode ser feito durante análise sintática
- Otimização de saltos poderia ser feita após a geração do código-alvo
 - Otimizações de buracos de fechadura (vêm-se apenas pequenas porções de código)

51

Otimização

■ Quanto ao instante

- A maioria das otimizações é feita sobre o código intermediário ou durante a geração do código-alvo
 - Otimizações de **nível de fonte**: depende do programa (por exemplo, quantas vezes as variáveis são acessadas determina quais variáveis ficarão em registradores)
 - Otimizações de **nível de alvo**: depende da máquina-alvo e do ambiente de execução (por exemplo, número de registradores disponíveis)

52

Otimização

■ Quanto ao instante

- Análise do efeito de uma otimização sobre outras
 - Faz sentido propagar constantes antes de tratar código inatingível (pode ser mais fácil identificá-los)

x=1; ... y=0; ... if (y) x=0; ...	Propagação de constantes →	x=1; ... y=0; ... if (0) x=0; ...	Remoção de código inatingível →	x=1; ... y=0; ...
--------------------------------------------------	----------------------------------	--------------------------------------------------	---------------------------------------	----------------------------

53

Otimização

■ Quanto à área do programa

- Otimizações locais: que se aplicam a segmentos de código de linha reta, ou seja, que não contêm saltos para dentro ou fora da sequência; sequência maximal de código de linha reta é chamada "bloco básico"
 - Relativamente fáceis de efetuar
- Otimizações globais: que se estendem para além dos blocos básicos, mas que sejam confinadas a um procedimento individual
 - Exigem análise de fluxo de dados
- Otimizações interprocedimentos: que se estendem para além dos limites dos procedimentos, podendo atingir o programa todo
 - As mais complexas, exigindo diversos tipos de informações e rastreamentos do programa

54