Universidade Federal de São Carlos – Departamento de Computação
Construção de Compiladores e Construção de Compiladores 1
Profa. Helena Caseli

Sexta Lista de Exercícios – Geração e Otimização de Código

1) Cite quais são os 2 tipos de código intermediário apresentados em aula e suas características principais. Quais são as diferenças entre eles?

R. O código de três endereços é uma forma de representação intermediária na qual cada instrução pode envolver, no máximo, três endereços de memória:

x = y op z

em que op é um operador aritmético (+ ou -, por exemplo) ou algum outro operador que possa operar sobre os valores de y e z. Atenção: o uso de x é diferente do uso de y e z já que y e z podem representar constantes ou literais sem endereços na execução, enquanto é necessário conhecer o endereço de x para que a atribuição possa ser realizada. Isso fica bem claro com o P-código. Outra característica relevante do código de três endereços é a necessidade de utilizar temporários (t1, t2 etc.) para armazenar os valores intermediários das operações. Esses temporários correspondem aos nós interiores da árvore sintática e representam seus valores computados; podem ser armazenados na memória ou em registradores.

O P-código, por sua vez, surgiu como um código de montagem alvo padrão produzido pelos compiladores Pascal e foi projetado como código de uma máquina hipotética baseada em pilhas (P-máquina) com interpretador para diversas máquinas reais. Como foi projetado para ser executado diretamente, o P-código contém uma descrição implícita de um ambiente de execução entre outros detalhes abstraídos quando o consideramos como código intermediário nesse curso.

Código de três endereços X P-código

Código de três endereços	*P-código*
- é mais compacto (menos instruções) *- é auto-suficiente no sentido de que não precisa de uma pilha para representar o processamento* *- precisa de temporários para armazenar os valores das computações intermediárias*	*- é mais próximo do código de máquina* *- as instruções exigem menos endereços (os endereços omitidos estão na pilha implícita)* *- não precisa de temporários, uma vez que a pilha contém todos os valores temporários*

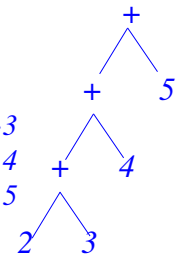
2) Apresente a sequência de instruções de código de três endereços correspondente a cada uma das expressões aritméticas a seguir. Quais são as árvores sintáticas abstratas que correspondem à geração de código?

a) 2+3+4+5
b) 2+(3+(4+5))
c) a*b+a*b*c

R.

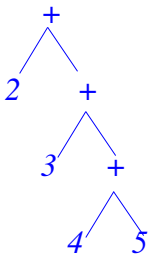
a)

t1 = 2+3
t2 = t1+4
t3 = t2+5



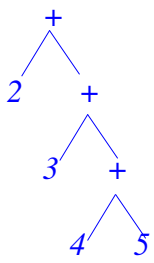
ou

t1 = 4+5
t2 = 3+t1
t3 = 2+t2



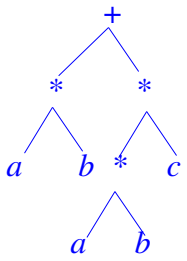
b)

t1 = 4+5
t2 = 3+t1
t3 =2+t2



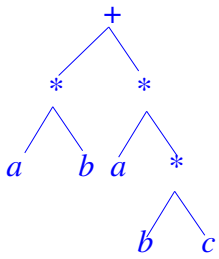
c)

```
t1 = a*b
t2 = a*b
t3 = t2*c
t4 = t1+t3
```



ou

```
t1 = a*b
t2 = b*c
t3 = a*t2
t4 = t1+t3
```



3) Apresente a sequência de instruções de P-código correspondente às expressões aritméticas do exercício anterior.

R.

a)

```
ldc 2
ldc 3
adi
ldc 4
adi
ldc 5
adi
```

ou *(gerando num passo posterior à análise)*

```
ldc 4
ldc 5
adi
ldc 3
adi
ldc 2
adi
```

b)

```
ldc 2
ldc 3
ldc 4
ldc 5
adi
adi
adi
```

ou

```
ldc 4
ldc 5
adi
ldc 3
adi
ldc 2
adi
```

c)

```
lod a
lod b
mpi
lod a
lod b
mpi
lod c
mpi
adi
```

4) Escreva a gramática de atributos para geração de <u>código de três endereços</u> para a gramática de expressões aritméticas de inteiros a seguir. Utilizando a gramática resultante, gere o código de três endereços para <u>todas</u> as expressões da questão 2.

exp → exp soma termo | termo
soma → + | -
termo → termo mult fator | fator
mult → *
fator → (exp) | num | id

R.

Regra gramatical	*Regra semântica*
exp → exp soma termo	*exp_1.nome = newtemp()* *exp_1.tacode = exp_2.tacode ++ termo.tacode ++* *exp_1.nome || "=" || exp_2.nome || soma.nome || termo.nome*
exp → termo	*exp.nome = termo.nome* *exp.tacode = termo.tacode*
soma → +	*soma.nome = "+"* *soma.tacode = ""*
soma → -	*soma.nome = "-"* *soma.tacode = ""*
termo → termo mult fator	*$termo_1$.nome = newtemp()* *$termo_1$.tacode = $termo_2$.tacode ++ fator.tacode ++* *$termo_1$.nome || "=" || $termo_2$.nome || mult.nome || fator.nome*
termo → fator	*termo.nome = fator.nome* *termo.tacode = fator.tacode*
*mult → **	*mult.nome = "*"* *mult.tacode = ""*
fator → (exp)	*fator.nome = exp.nome* *fator.tacode = exp.tacode*
fator → num	*fator.nome = num.strval* *fator.tacode = ""*
fator → id	*fator.nome = id.strval* *fator.tacode = ""*

5) Considerando-se a mesma gramática do exercício anterior, escreva a gramática de atributos para a geração de P-código. Utilizando a gramática resultante, gere o código de três endereços para todas as expressões da questão 2.
R.

Regra gramatical	*Regra semântica*
exp → exp soma termo	*exp_1.pcode = exp_2.pcode ++ termo.pcode ++ soma.pcode*
exp → termo	*exp.pcode = termo.pcode*
soma → +	*soma.pcode = "adi"*
soma → -	*soma.pcode = "sbi"*
termo → termo mult fator	*$termo_1$.pcode = $termo_2$.pcode ++ fator.pcode ++ mult.pcode*
termo → fator	*termo.pcode = fator.pcode*
*mult → **	*mult.pcode = "mpi"*
fator → (exp)	*fator.pcode = exp.pcode*
fator → num	*fator.pcode = "ldc" || num.strval*
fator → id	*fator.pcode = "lod" || id.strval*

6) Escreva o procedimento correspondente para a geração de código de três endereços para a gramática da questão 4.

R.

```
fucntion genTCode(T: no_arvore): string;
var  temp: string;
begin
        if T não é nulo then
                if (T.val='+') then
                        temp = newtemp();
                        write(temp+"="+genTCode(T->filhoesq)+"+"+genTCode(T->filhodir));
                        return temp;
                else if (T.val='-') then
                        temp = newtemp();
                        write(temp+"="+genTCode(T->filhoesq)+"-"+genTCode(T->filhodir));
                        return temp;
                else if (T.val='*') then
                        temp = newtemp();
                        write(temp+"="+genTCode(T->filhoesq)+"*"+genTCode(T->filhodir));
                        return temp;
                else if (T.val='num') then return num.strval;
                else if (T.val='id') then return id.strval;
end;
```

7) Escreva o procedimento correspondente para a geração de P-código para a gramática da questão 4.

R.

```
procedure genPCode(T: no_arvore);
begin
        if T não é nulo then
                if (T.val='+') then
                        genPCode(T->filhoesq);
                        genPCode(T->filhodir);
                        write("adi");
                else if (T.val='-') then
                        genPCode(T->filhoesq);
                        genPCode(T->filhodir);
                        write("sbi");
                else if (T.val='*') then
                        genPCode(T->filhoesq);
                        genPCode(T->filhodir);
                        write("mpi");
                else if (T.val='num') then write("ldc"+num.strval);
                else if (T.val='id') then write("lod"+id.strval);
end;
```

8) Apresente as instruções de três endereços correspondentes às expressões em C a seguir.

a) (x=y=2)+3*(x=4)

b) a[a[i]]=b[i=2]

c) p->next->next = p->next

R. *a)*

```
y = 2
x = 2
x = 4
t1 = 3*4
t2 = 2 + t1
```

b)

```
i = 2
t1 = 2*elem_size(b)
t2 = &b+t1
t3 = *t2
t4 = i
t5 = t4*elem_size(a)
t6 = &a+t5
t7 = *t6
t8 = t7*elem_size(a)
t9 = &a+t8
*t9 = t3
```

c)

```
t1 = p + field_offset(*p,next);
t2 = p + field_offset(*p,next);
t3 = t2 + field_offset(*p,next);
*t3 = t1
```

9) Apresente a sequência de instruções de P-código correspondente às expressões em C do exercício anterior.

R. *a)*

```
lda x
lda y
ldc 2
stn
stn
ldc 3
lda x
ldc 4
stn
mpi
adi
```

b)

```
lda a
lda a
lod i
ixa elem_size(a)
ind 0
ixa elem_size(a)
lda b
lda i
ldc 2
stn
ixa elem_size(b)
ind 0
sto
```

c)

```
lod p
ldc field_offset(*p,next)
ixa 1
ldc field_offset(*p,next)
ixa 1
lod p
ind field_offset(*p,next)
sto
```

10) Cite, em ordem decrescente de vantagem em relação a custo, as várias fontes de otimização apresentadas em aula explicando o que vem a ser cada uma delas e dizendo "o que" elas otimizam (tamanho, velocidade etc.).

R.

1. *Alocação de registradores*
   O bom uso dos registradores é a característica mais importante do código eficiente, uma vez que é por meio deles que as operações são efetuadas. Quanto maior o número de registradores e melhor seu uso, maior a velocidade do código gerado.
2. *Remoção de operações desnecessárias*
   Às vezes um código-fonte dá origem (após a tradução para código intermediário ou alvo) a operações que não serão executadas ou que poderiam ser realizadas de uma maneira mais "inteligente" como as subexpressões comuns, os códigos inatingíveis e os saltos desnecessários. Ao se evitar a geração de código para essas operações redundantes ou desnecessárias realiza-se uma otimização do tamanho do código gerado.
3. *Adaptação de operações caras*
   Algumas operações podem ser substituídas por uma versão mais "barata" computacionalmente (executa em tempo menor ou envolvendo menos registradores etc.) o que é chamado de redução de força. Além disso, essa otimização também diz respeito ao empacotamento de constantes (troca de expressões constantes pelo valor calculado), alinhamento de procedimentos (inserir o código do procedimento onde ele é chamado evitando-se, assim, todas as operações de ativação) e remoção de redução em cauda (quando a última instrução do procedimento é uma chamada a si próprio), e uso de dialetos de máquina. Todas essas alterações são realizadas com o intuito de aumentar a velocidade de execução do código-alvo.
4. *Previsão de comportamento do programa*
   O conhecimento apropriado do comportamento dos programas a serem escritos na linguagem, por parte do projetista do compilador, permite que se possa otimizar as operações usadas com maior frequência aumentando, assim, a velocidade.