GERAÇÃO DE CÓDIGO

Alexandre Agustini
Politécnica/PUCRS

Rascunho

Geração de código

Objetivo:

Gerar um código ***equivalente ao programa fonte*** em uma linguagem de máquina ou linguagem de montagem, de acordo com a arquitetura onde será executado o programa.

Tarefas:

- Reservar espaço para variáveis
- Alocar o uso dos registradores da máquina
- Gerar código para as construções da linguagem

Geração de código

A geração de código é apresentada utilizando-se um código intermediário, esta alternativa apresenta algumas vantagens:

- facilita o transporte entre máquinas;
- possibilita a utilização de otimizadores de código independentes;
- não exige o estudo e aprendizado de uma arquitetura específica.

Entre as representações mais comuns pode-se destacar:

- árvores sintáticas (abstratas);
- máquinas virturais de pilha;
- máquinas virtuais com código de três endereços.

Neste material é utilizada a notação de uma máquina de pilha, apresentada e estendida conforme a necessidade.

Declarações (Globais)

- reservam espaço para as variáveis
- inclui o endereço na Tabela de Símbolos

Obs.: como está se gerando uma linguagem de montagem, podemos considerar

endereço = rótulo (nome) da posição de memória

- alternativas para nomes internos gerados pelo compilador:

i. extrair do nome das variáveis no programa fonte (ex.: _idade)

ii. gerado internamente (ex.: V001, V002, ...)

Declarações (2)

- Procedimentos auxiliares:

InclTS(id, rotVar, tam)	insere na TS, associado a ID, um rótulo e tamanho
rotVar := novoRotVar()	função que gera rótulos para variáveis (V001, V002, ..)
geraCod(...)	gera uma linha de código objeto

- PseudoInstruções:

DS N	Reserva N posições a partir do endereço atual
DC N	Armazena N no endereço corrente e reserva esta posição

- Instruções:

JMP N	Desvia para endereço N
NOP	No OPeration (Faz nada)

Declarações (3)

```
Prog  → { geraCod( 'JMP  _start' ) }
            Decl
            { geraCod( '_start :  NOP ' ) }
        Cmdos

Decl  →  Decl  id : Tipo ;  {  temp = novoRotVar() ;
                               inclTS( id.val, temp, Tipo.tam) ;
                               geraCod( temp, 'DS', tipo.tam) }
         | ε

Tipo  →  INTEGER   { tipo.tam = 4 }
      |  DOUBLE        { tipo.tam = 8 }
      |  array [ num ] of Tipo1  { Tipo.tam = tipo1.tam*num.valor }
```

Declarações - exemplo

Programa:

```
a : integer ;
v : array [5] of integer;
```

Código:

```
        JMP _start
_a      DS 4
_v      DS 20
_start: NOP
```

Expressões e atribuição

- Procedimentos auxiliares:

pesqRotTS(ID)	pesquisa na TS o rótulo associado ao identificador ID

- Instruções:

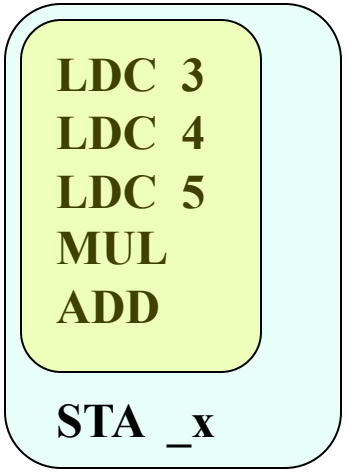
LDA N	(Load Address) Empilha o conteúdo do endereço N
STA N	(Store Address) Desempilha valor e armazena no endereço N
LDC N	(Load Constant) Empilha constante N
ADD	Desempilha RT1 e RT2 e empilha RT2+RT1
MUL	Desempilha RT1 e RT2 e empilha RT2+RT1

Expressões e atribuição - exemplo

Cmdo → id := Exp { rotVar:=pesqRotTS(id.val); geraCod('STA', rotVar) }

Exp → id { rotVar := pesqRotTS(id.val) ; exraCod('LDA',rotVar) }
| num { geraCod('LDC', num.val) }
| Exp + Exp { geraCod('ADD') }
| Exp * Exp { geraCod('MUL') }
| (Exp) { }
| ...

Exemplo: ‘x := 3 + 4 * 5'



Comandos de controle

- Procedimentos auxiliares:

novoRotCod()	função que gera rótulos para código (R001, R002, ..)

- Instruções:

JZER N	(Jump Zero) Desempilha e se valor desempilhado for zero desvia para o endereço N
JPOS N	(Jump Positive)Desempilha e se valor desempilhado for positivo desvia para o endereço N
GRT	Desempilha RT1 e RT2, empilha 1 se RT2>RT1, c.c. empilha 0
LES	Desempilha RT1 e RT2, empilha 1 se RT2<RT1, c.c. empilha 0
EQ	Desempilha RT1 e RT2, empilha 1 se RT2==RT1, c.c. empilha 0

Comandos de controle (2)

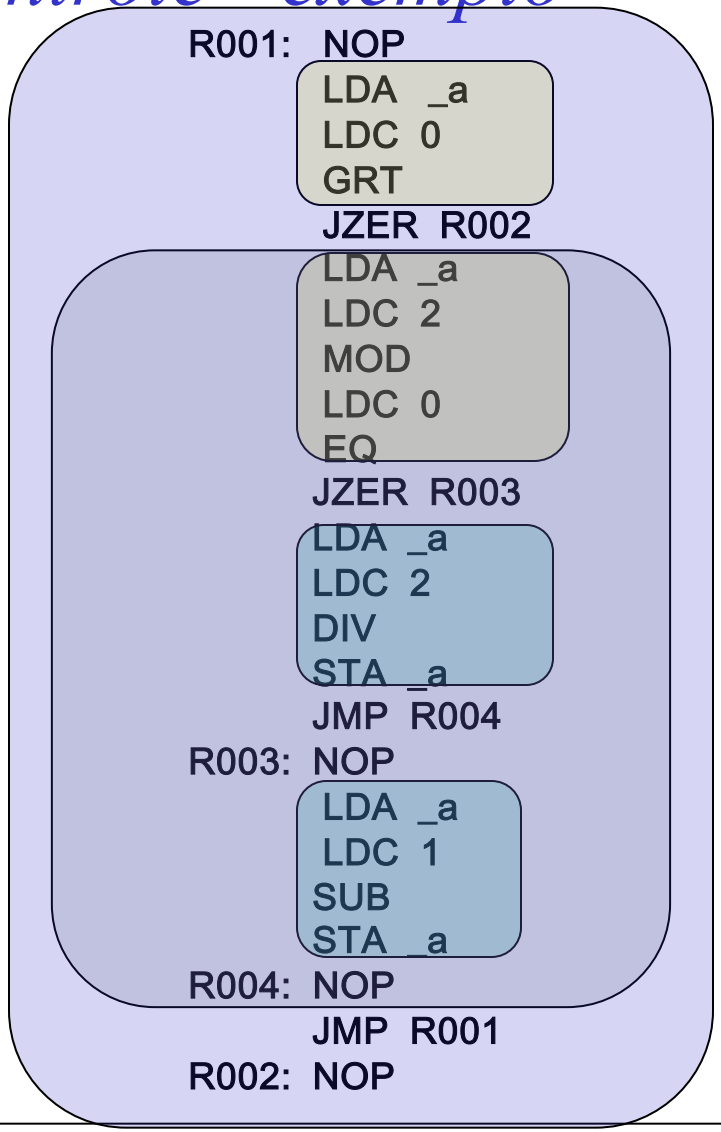
```
Stack<Integer> pRot = new Stack<Integer>();
int proxRot = 1;
```

```
Cmdo  → IF {  pRot.push(proxRot);  proxRot += 2; }
          Exp  {  geraCod( 'JZER', pRot.peek())  }
           THEN Cmdo  {  geraCod( 'JMP', pRot.peek()+1);
                         geraCod(pRot.peek(), ':NOP') }
           ELSE  Cmdo  {  geraCod(pRot.peek()+1, ':NOP' ) ;
                          pRot.pop();}
        | WHILE {  pRot.push(proxRot);  proxRot += 2;
                   geraCod(', pRot.peek(), ':NOP') }
          Exp    { geraCod( 'JZER', ', pRot.peek()+1)  }
          DO Cmdo  {  geraCod( 'JMP', pRot.peek());
                      geraCod(pRot.peek()+1, ':NOP')
                      pRot.pop()}
```

Comandos de controle - exemplo

Exemplo:

```
while (a > 0)
      if  (a % 2 == 0)
          a  = a / 2
       else
          a = a - 1
```



Procedimentos não-locais

Problema: na geração de código de procedimentos, que podem ser ativados recursivamente, onde alocar espaço para as variáveis locais ? O que fazer com os parâmetros ?

Solução: para resolver os problemas acima utiliza-se um REGISTRO DE ATIVAÇÃO, para cada procedimento há um registro de ativação contendo:

- variáveis locais utilizadas no procedimento;
- parâmetros;
- Informações de controle.

Registro de ativação

A cada ativação de um procedimento deve ser criado um novo registro na pilha do sistema (Heap); ao término do procedimento (retorno) este registro pode ser destruído.

Na arquitetura da máquina devemos ter:

- um apontador para o topo da pilha (SP=Stack Pointer)
- um apontador apontando para o início do registro de ativação (BP=base pointer); variáveis são acessadas por meio de um deslocamento (displacement) em relação ao BP;

Ao entrar em um procedimento:

- empilha o endereço de retorno;
- salva o BP antigo;
- ajusta o novo BP;
- aloca espaço para as variáveis.

Quando termina:

- desaloca espaço das variáveis;
- restaura valor do BP;
- retorna, desempilhando o valor de retorno.

Subprogramas

- Instruções:

AMEM N	(Alloc Memory) aloca N palavras na pilha (SP:=SP+N)
DMEM N	(DeAlloc Memory) SP:=SP-N
SETBP	faz BP apontar para o topo da pilha
POPBP	desempilha RT1 e faz BP ← RT1
PUSHBP	empilha valor de BP
LDO N	(Load Offset) empilha o conteúdo da posição de memória BP+N
STO N	desempilha palavra e coloca na posição BP+N
DROP	tira uma palavra do topo da pilha

Subprogramas - exemplos

Programa		Código gerado
Procedure P	P:	NOP
		PUSHBP
		SETBP
var x,a,b,c		AMEM 4
x := z		LDA 500 (endereço de 'z')
		STO 1
p		JSR P
a :=x * z		LDO 1
		LDA 500
		MUL
		STO 2
...		
end		DMEM 4
		POPBP
		RTS

Parâmetros por valor

Parâmetros são colocados no topo da pilha e alocados por deslocamentos negativos em relação ao BP.

Exemplo:

```
Procedure  p (  t : integer );
 var a : integer;
  ...
  t := a * 3 .....................................    LDO  1 (BP+1 =  'a')
                                                     LDC 3
                                                     MUL
                                                     STO  -2  (BP-2 = 't' (primeiro parâmetro))
```

- No caso de parâmetros por referência o que é empilhado é o endereço dos parâmetros reais e não seu conteúdo.

- Instruções:

STI	(Store Indirect) desempilha RT1 e RT2 e armazena RT1 no endereço RT2
LDI	(Load Indirect) desempilha RT1 e empilha valor do endereço RT1

Parâmetros por referência

Exemplo:

```
Procedure  p( var: r:integer;  v:integer);
 ...
 r := r*v  .....
```