

# Geração de Código para LALG

Ambiente de execução para LALG  
Máquina hipotética  
Repertório de instruções

Prof. Thiago A. S. Pardo

1

# Geração de código para LALG

- Código-alvo será um código de montagem
  - Portanto, requer interpretação posterior

2

## Geração de código para LALG

- **Técnica** para geração de código
  - **Ad hoc**: geração de código **atrelada aos procedimentos sintáticos**
    - Geração diretamente dos procedimentos sintáticos ou...
    - via chamada a procedimentos/funções de geração com argumentos específicos
      - **Tabela de símbolos** dará o suporte necessário para a geração

3

## Geração de código para LALG

- **Máquina hipotética** e seu **ambiente de execução**
  - **MaqHipo**, com ambiente baseado em **pilhas** simplificado
    - Basicamente, área de código e de dados
      - Área de código: simulada em um vetor nomeado C
      - Área de dados: simulada em um vetor nomeado D
    - Ponteiros/registadores
      - Contador de programa i
      - Topo da pilha s

4

## Geração de código para LALG

- Área de código C
  - A geração consiste no **preenchimento deste vetor** conforme o programa for sendo compilado
    - Posição atual marcada por C[i]
    - Ao final, o vetor C é gravado em um arquivo de saída
      - As instruções não podem ir direto para o arquivo conforme forem sendo geradas
        - Ainda poderão ser modificadas
        - O processo pode parar por algum motivo e o tempo de escrita em arquivo seria perdido

5

## Geração de código para LALG

- Área de código C
  - Cada posição do vetor contém uma instrução na linguagem de montagem
  - **Tabela de símbolos** acrescida de **2 campos** para suporte à geração
    - Endereço relativo de identificadores na pilha
    - Endereço da primeira instrução de procedimentos
  - Esse vetor será **interpretado posteriormente**

6

## Geração de código para LALG

### ■ Área de dados D

- Vetor que se comporta como uma pilha
  - Topo marcado por D[s]
- **Só existirá** realmente durante a **execução**
- As instruções funcionarão sobre seu topo e, muitas vezes, sobre a posição imediatamente abaixo do topo (no caso de uma operação binária)

7

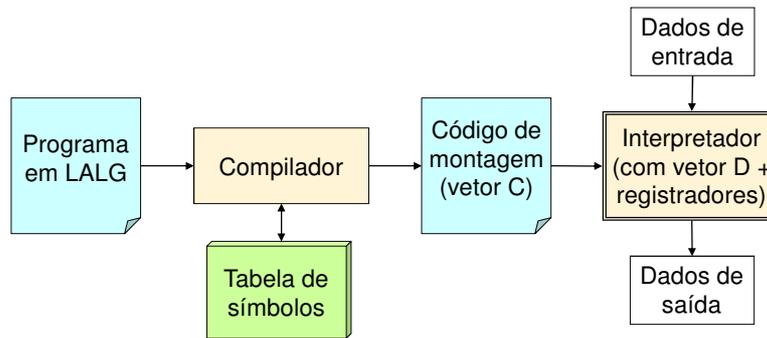
## Geração de código para LALG

- **Uma vez gerado código** no vetor C, a **execução é simples** (via um interpretador)
  - As instruções indicadas pelo registrador i são executadas até que seja encontrada a instrução de parada ou ocorra algum erro
    - Conforme as instruções são executadas, a pilha é manipulada
  - A execução de cada instrução aumenta de 1 o valor de i, exceto as instruções que envolvem desvios
  - Como só há os tipos inteiro e real na LALG, a pilha D pode ser um **vetor de reais**

8

## Geração de código para LALG

### ■ Interpretador



9

## Repertório de instruções

- **Grupos de instruções** que funcionam para LALG, mas que **funcionariam também para o básico de outras linguagens**, como **Pascal e C**
  - Avaliação de expressões
  - Atribuição
  - Comandos condicionais e iterativos
  - Entrada e saída
  - Alocação de memória
  - Inicialização e finalização de programa
  - Procedimentos
- Junto com cada **instrução**
  - **interpretação durante a execução em cor verde**
  - **ações durante compilação em cor vermelha**

10

## Instruções para avaliação de expressões

- CRCT k

{carrega constante k na pilha D}

$s := s + 1$

$D[s] := k$

- CRVL n

{carrega valor de endereço n na pilha D}

$s := s + 1$

$D[s] := D[n]$

11

## Instruções para avaliação de expressões

- SOMA

{soma topo da pilha com seu antecessor, desempilha-os e empilha resultado}

$D[s-1] := D[s-1] + D[s]$

$s := s - 1$

- SUBT

{subtrai o elemento do topo do antecessor, desempilha-os e empilha o resultado}

$D[s-1] := D[s-1] - D[s];$

$s := s - 1$

12

## Instruções para avaliação de expressões

- **MULT**

{multiplica elemento do topo pelo antecessor, desempilha-os e empilha resultado}

$D[s-1] := D[s-1] * D[s]$

$s := s - 1$

- **DIVI**

{divide elemento do antecessor pelo do topo, desempilha-os e empilha resultado }

$D[s-1] := D[s-1] \text{ div } D[s]$

$s := s - 1$

13

## Instruções para avaliação de expressões

- **INVE**

{inverte sinal do topo}

$D[s] := - D[s]$

- **CONJ**

{conjunção de valores lógicos: F=0 e V=1}

se  $D[s-1]=1$  e  $D[s]=1$  então  $D[s-1]:=1$

senão  $D[s-1]:=0$

$s := s - 1$

14

## Instruções para avaliação de expressões

- DISJ

{disjunção de valores lógicos}

se  $D[s-1]=1$  ou  $D[s]=1$  então  $D[s-1]:=1$

senão  $D[s-1]:=0$

$s:=s-1$

- NEGA

{negação lógica}

$D[s]:=1-D[s]$

15

## Instruções para avaliação de expressões

- CPME

{comparação de menor}

se  $D[s-1]<D[s]$  então  $D[s-1]:=1$

senão  $D[s-1]:=0$

$s:=s-1$

- CPMA

{comparação de maior}

se  $D[s-1]>D[s]$  então  $D[s-1]:=1$

senão  $D[s-1]:=0$

$s:=s-1$

16

## Instruções para avaliação de expressões

- CFIG  
{comparação de igualdade}  
se  $D[s-1]=D[s]$  então  $D[s-1]:=1$   
senão  $D[s-1]:=0$   
 $s:=s-1$
- CDES  
{comparação de desigualdade}  
se  $D[s-1]<>D[s]$  então  $D[s-1]:=1$   
senão  $D[s-1]:=0$   
 $s:=s-1$

17

## Instruções para avaliação de expressões

- CPMI  
{comparação  $\leq$  }  
se  $D[s-1]\leq D[s]$  então  $D[s-1]:=1$   
senão  $D[s-1]:=0$   
 $s:=s-1$
- CMAI  
{comparação  $\geq$  }  
se  $D[s-1]\geq D[s]$  então  $D[s-1]:=1$   
senão  $D[s-1]:=0$   
 $s:=s-1$

18

## Instruções para avaliação de expressões

### ■ Exemplo

□  $1 - a + b$

- CRCT 1
- CRVL  $a^*$
- SUBT
- CRVL  $b^*$
- SOMA

\* endereços na pilha D obtidos da tabela de símbolos

19

## Instrução de atribuição

### ■ ARMZ n

{armazena o topo da pilha no endereço n de D}

$D[n] := D[s]$

$s := s - 1$

20

## Instrução de atribuição

### ■ Exemplo

□  $a := a + 1$

- CRVL  $a^*$
- CRCT 1
- SOMA
- ARMZ  $a^*$

21

## Instruções para comandos condicionais e iterativos

### ■ DSVI p

{desvio incondicional para a instrução de endereço p}

$i := p$

### ■ DSVF p

{desvie para a instrução de endereço p caso a condição resultante seja falsa}

???

22

## Instruções para comandos condicionais e iterativos

- DSVI p  
{desvio incondicional para a instrução de endereço p}  
i:=p
- DSVF p  
{desvie para a instrução de endereço p caso a condição resultante seja falsa}  
se D[s]=0 então i:=p  
senão i:=i+1  
s:=s-1

23

## Instruções para comandos condicionais e iterativos

- Seqüência de instruções
  - If E then C1 else C2

Seqüência de instruções?

24

## Instruções para comandos condicionais e iterativos

### ■ Seqüência de instruções

#### □ If E then C1 else C2

```
... } E  
DSVF k1  
... } C1  
DSVI k2  
k1 ... } C2  
k2 ...
```

No lugar de k1 e k2 devem aparecer índices reais de C

k1 é determinado quando se encontra o else, e k2 quando termina o comando if

→ necessidade de se voltar no vetor C para substituir k1 e k2 por índices reais

25

## Instruções para comandos condicionais e iterativos

### ■ Seqüência de instruções

#### □ If E then C

```
... } E  
DSVF k1  
... } C  
k1 ...
```

26

## Instruções para comandos condicionais e iterativos

- Seqüência de instruções

- While E do C

```
k1 ... } E
  DSVF k2
  ... } C
  DSVI k1
k2 ...
```

27

## Instruções para comandos condicionais e iterativos

- Seqüência de instruções

- Repeat C until E

???

28

## Instruções para comandos condicionais e iterativos

- Seqüência de instruções

- Repeat C until E

```
k1 ... } C
E
DSVF k1
```

29

## Instruções para entrada e saída

- LEIT

{lê um dado do arquivo de entrada}

$s := s + 1$

$D[s] := \text{próximo valor da entrada}$

- IMPR

{imprime valor inteiro no arquivo de saída}

Imprime  $D[s]$

$s := s - 1$

30

## Instruções para entrada e saída

### ■ Exemplo

#### □ Read a, b

- LEIT
- ARMZ a\*
- LEIT
- ARMZ b\*

31

## Instruções para entrada e saída

### ■ Exemplo

#### □ Write x, x\*y

- CRVL x\*
- IMPR
- CRVL x\*
- CRVL y\*
- MULT
- IMPR

32

## Instrução de alocação de memória

- ALME m

{reserva m posições na pilha D, sendo que m depende do tipo da variável}

o valor de s é armazenado no campo de endereço relativo da variável correspondente na tabela de símbolos (em tempo de compilação)

S:=S+m

- Na LALG, por simplicidade, todos os tipos ocuparão uma única posição
  - m=1

33

## Instrução de alocação de memória

- Exemplo

```
program p1;  
var x, y: integer;  
...
```



ALME 1  
ALME 1



### Tabela de símbolos

<u>Cadeia</u>	<u>Token</u>	...	<u>End. relativo</u>
x	id		0
y	id		1
...			

34

## Instruções para inicialização e finalização de programas

- INPP

{inicia programa - será sempre a 1ª instrução}

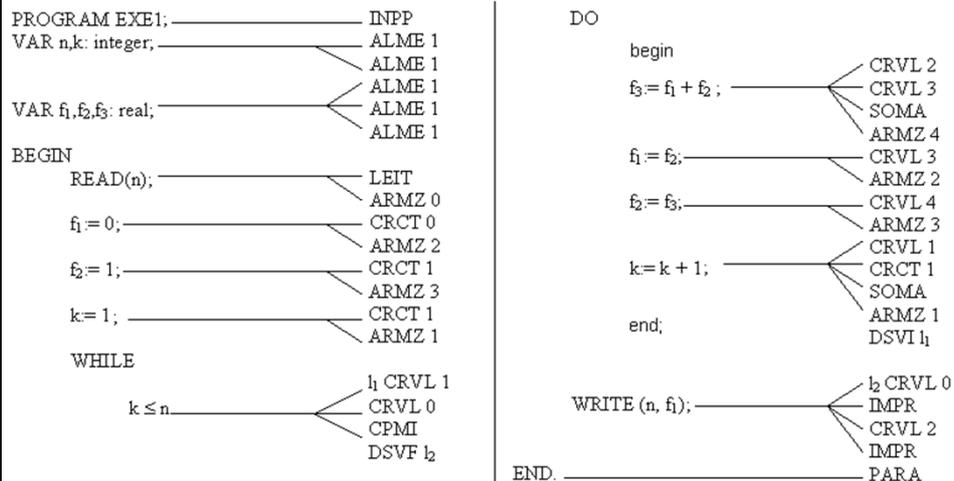
s := -1

- PARA

{termina a execução do programa}

35

## Exemplo completo



No lugar dos rótulos simbólicos l1 e l2 devem aparecer endereços reais que serão calculados após sua geração, tendo-se que voltar no vetor C

## Exercício

- Gere código para o programa abaixo e interprete o código gerado

```
program exemplo1;  
var a, b: integer;  
begin  
  read(a,b);  
  write(a,b);  
end.
```

37

## Exercício

- Gere código para o programa ao lado e interprete o código gerado

```
program exemplo2;  
var a, b: integer;  
var c: real;  
begin  
  read(a,b);  
  c:=5;  
  while a<b do  
  begin  
    a:=a+1;  
    c:=c*a;  
  end;  
  write(c);  
end.
```

38

## Exercício

- Gere código para o programa ao lado e interprete o código gerado

```
program exemplo3;
var x, y: integer;
begin
  read(x);
  y:=x*x;
  if (x<y) then
  while (x<y) do
  begin
  y=y-2;
  write(y);
  end;
  else write(x);
  write(x*y);
  end.
```

39

## Instruções para procedimentos

- **Características** da LALG
  - **Passagem** de parâmetros por **valor**
  - Somente **procedimentos globais**

40

## Instruções para procedimentos

- **Ao chamar procedimento**
  - Empilha-se endereço de retorno (ainda não definido, pois depende do número de parâmetros)
  - Empilham-se valores de parâmetros
  - Salta-se para 1ª instrução de procedimento
- **No início do procedimento**
  - Desvia-se para programa principal
  - Copia valores dos parâmetros passados
- **No fim do procedimento**
  - Libera memória (variáveis locais e parâmetros)
  - Retorna do procedimento

41

## Instruções para procedimentos

- **PUSHER e**  
{empilha endereço de retorno}  
 $s := s + 1$   
 $D[s] := e$
- **CHPR p**  
{desvia para instrução de índice p no vetor C, obtido na tabela de símbolos}  
 $i := p$

42

## Instruções para procedimentos

- DESM m

{desaloca m posições de memória, a partir do topo s de D, restaurando os valores do topo a partir de m}

deve retirar (ou tornar inacessível) da tabela de símbolos as posições desalocadas (em tempo de compilação)

s:=s-m

- RTPR

{retorna do procedimento}

i:=D[s]

s:=s-1

43

## Instruções para procedimentos

- COPVL

sem efeito na execução

coloca na tabela de símbolos o endereço do parâmetro, associando parâmetros atuais com formais (em tempo de compilação)

- PARAM n

{aloca memória e copia valor da posição n de D}

s:=s+1

D[s]:=D[n]

44

## Instruções para procedimentos

- Ao chamar procedimento

- PUSHER e
- {PARAM n}
- {.....}
- CHPR p

- No início do procedimento

- {DSVI k}
- {COPVL}
- {.....}

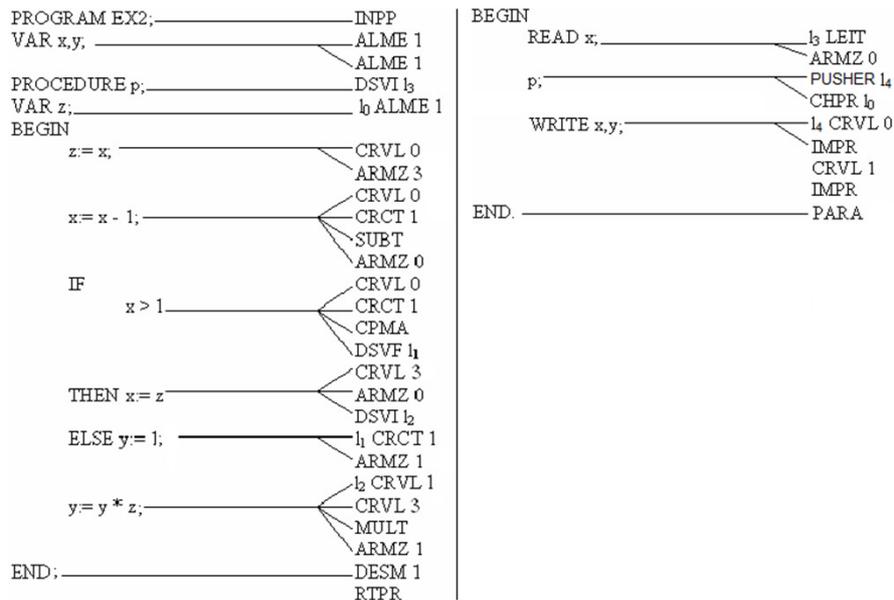
Deve levar em conta a posição usada pelo endereço de retorno na pilha D, isto é, soma-se 1 ao endereço obtido para o primeiro parâmetro ou variável local

- No fim do procedimento

- DESM n
- RTPR

45

## Exemplo: sem parâmetros



## Exemplo: com parâmetros

```
Program ex3;  
var a,b: integer;  
procedure proc(x,y:integer);  
    var l: integer;  
    begin  
        l:= x + y;  
        x:= l;  
    end;  
begin  
    read(a,b);  
    proc(a,b);  
end.
```

```
1.INPP  
2. ALME 1  
3. ALME 1  
4. DSVI 16  
5. COPVL  
6. COPVL  
7. ALME1  
8.CRVL 3  
9. CRVL 4  
10. SOMA  
11. ARMZ 5  
12. CRVL 5  
13. ARMZ 3  
14. DESM 3  
15. RTPR  
16. LEIT  
17. ARMZ 0  
18. LEIT  
19. ARMZ 1  
20. PUSHER 24  
21. PARAM 0  
22. PARAM 1  
23. CHPR 5  
24. PARA
```

## Instruções para procedimentos

- Ao se gerar código para um procedimento, coloca-se o endereço de sua primeira instrução na tabela de símbolos
- Ao se retornar do procedimento, a pilha deve estar exatamente como estava antes da chamada do procedimento

## Exercício

- Gere código para o programa ao lado e interprete o código gerado

```
program exemplo2;
var a: real;
var b: integer;
procedure nomep(x: real);
var a, c: integer;
begin
  read(c,a);
  if a<x+c then
  begin
    a:=c+x;
    write(a);
  end
  else c:=a+x;
  end;
begin {programa principal}
  read(b);
  nomep(b);
end.
```