

Geração de Código para LALG

Ambiente de execução para LALG

Máquina hipotética

Repertório de instruções

Prof. Thiago A. S. Pardo

Geração de código para LALG

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

Geração de código para LALG

■ Técnica para geração de código

- Ad hoc: geração de código atrelada às regras sintáticas

- Geração disparada pelas regras sintáticas ou...

- via chamada a procedimentos/funções de geração com argumentos específicos

- Tabela de símbolos dará o suporte necessário para a geração

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/registradores
 - Contador de programa i
 - Topo da pilha s

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

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**

Geração de código para LALG

■ Área de dados D

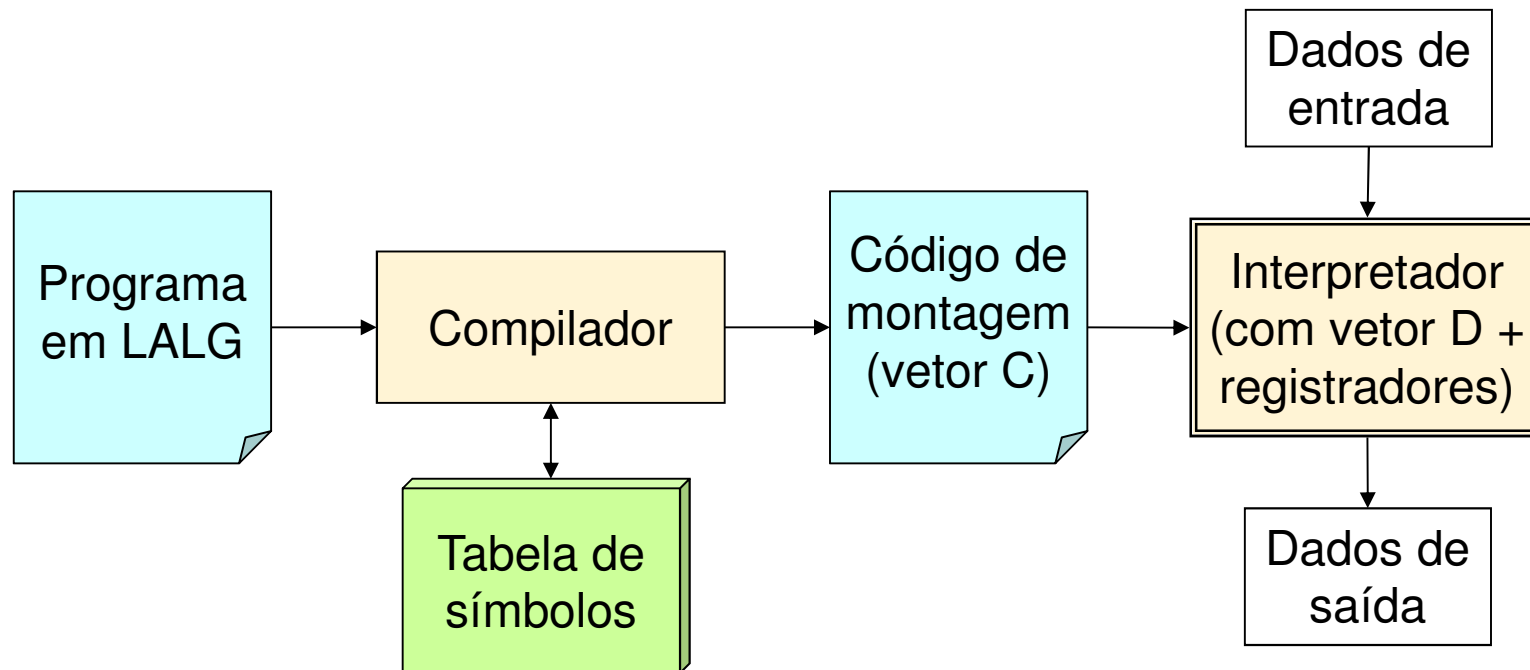
- ❑ Vetor que se comporta como uma pilha
 - Topo marcado por $D[s]$
- ❑ **Só existirá** realmente durante a **execução**
- ❑ As instruções funcionarão sobre seu topo e, muitas vezes, sobre a posição imediatamente abaixo do topo (no caso de uma operação binária)

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

Geração de código para LALG

■ Interpretador



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**

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}

$s := s + 1$

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

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}

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

$s := s - 1$

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 }

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

$s := s - 1$

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 $D[s-1] := 0$

$s := s - 1$

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}

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

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 $D[s-1] := 0$

$s := s - 1$

Instruções para avaliação de expressões

■ CPIG

{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 $D[s-1]:=0$

$s:=s-1$

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 $D[s-1] := 0$

$s := s - 1$

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

Instrução de atribuição

- ARMZ n

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

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

$s := s - 1$

Instrução de atribuição

■ Exemplo

□ $a := a + 1$

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

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}

???

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

Instruções para comandos condicionais e iterativos

- Sequência de instruções

- If E then C1 else C2

Sequência de instruções?

Instruções para comandos condicionais e iterativos

■ Sequência 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

Instruções para comandos condicionais e iterativos

■ Sequência de instruções

□ If E then C

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

Instruções para comandos condicionais e iterativos

■ Sequência de instruções

□ While E do C

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

Instruções para comandos condicionais e iterativos

- Sequência de instruções

- Repeat C until E

???

Instruções para comandos condicionais e iterativos

- Sequência de instruções

- Repeat C until E

k1 ... } C

E

DSVF k1

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 $D[s]$

$s := s - 1$

Instruções para entrada e saída

■ Exemplo

□ Read a, b

- LEIT
- ARMZ a^*
- LEIT
- ARMZ b^*

Instruções para entrada e saída

■ Exemplo

□ Write x, x^*y

- CRVL x^*
- IMPR
- CRVL x^*
- CRVL y^*
- MULT
- IMPR

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

→ $m = 1$

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
...			

Instruções para inicialização e finalização de programas

- INPP

{inicia programa - será sempre a 1ª instrução}

$s := -1$

- PARA

{termina a execução do programa}

Exemplo completo

```

PROGRAM EXE1; _____ INPP
VAR n,k: integer; _____ ALME 1
                                ALME 1
                                ALME 1
VAR f1,f2,f3: real; _____ ALME 1
                                ALME 1
BEGIN
  READ(n); _____ LEIT
                                ARMZ 0
  f1 := 0; _____ CRCT 0
                                ARMZ 2
  f2 := 1; _____ CRCT 1
                                ARMZ 3
  k := 1; _____ CRCT 1
                                ARMZ 1
  WHILE
    k ≤ n _____ l1 CRVL 1
                                CRVL 0
                                CPMI
                                DSVF l2

```

```

DO
  begin
    f3 := f1 + f2; _____ CRVL 2
                                CRVL 3
                                SOMA
                                ARMZ 4
    f1 := f2; _____ CRVL 3
                                ARMZ 2
    f2 := f3; _____ CRVL 4
                                ARMZ 3
    k = k + 1; _____ CRVL 1
                                CRCT 1
                                SOMA
                                ARMZ 1
                                DSVI l1
  end;

  WRITE (n, f1); _____ l2 CRVL 0
                                IMPR
                                CRVL 2
                                IMPR
END. _____ PARA

```

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.
```

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.
```

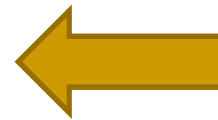
Exercício

- 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.
```

Exercício

- | | |
|-------------|-------------|
| 0) INPP | 16) DSVF 24 |
| 1) ALME 1 | 17) CRVL 1 |
| 2) ALME 1 | 18) CRCT 2 |
| 3) LEIT | 19) SUBT |
| 4) ARMZ 0 | 20) ARMZ 1 |
| 5) CRVL 0 | 21) CRVL 1 |
| 6) CRVL 0 | 22) IMPR |
| 7) MULT | 23) DSVI 13 |
| 8) ARMZ 1 | 24) DSVI 27 |
| 9) CRVL 0 | 25) CRVL 0 |
| 10) CRVL 1 | 26) IMPR |
| 11) CPME | 27) CRVL 0 |
| 12) DSVF 25 | 28) CRVL 1 |
| 13) CRVL 0 | 29) MULT |
| 14) CRVL 1 | 30) IMPR |
| 15) CPME | 31) PARA |



```
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.
```