# Geração de Código Objeto (Hipo)

**Fonte:** http://www.icmc.sc.usp.br/~gracan/download/sce126/

# Introdução

- Até agora, falou-se de:
  - Analisador Léxico
  - Analisador Sintático
  - Ações semânticas
    - verificação de declarações de identificadores, compatibilidade de tipos, etc.
- Há, ainda, a fase de tradução propriamente dita, do código fonte (LALG) para um código objeto
  - O código objeto pode ser:
    - um ASSEMBLY de uma determinada máquina
    - um pseudo-código de uma máquina hipotética
      - interpretado posteriormente,
      - pode ser executado em qualquer máquina que execute o interpretador

# Introdução

- Seja qual for a opção, é necessário que se conheçam as características da máquina-objeto
  - 1º caso: o computador e seu Assembly
  - 2º caso: a máquina hipotética e seu pseudo-código

- Nossa máquina hipotética será chamada de **MaqHipo** e o código da LALG será gerado para ela

# Características da MaqHipo

- MaqHipo será uma máquina à pilha
- Sua estrutura básica permitiria a interpretação de linguagens mais complexas que LALG, como PASCAL ou C

# Características da MaqHipo

- MaqHipo terá uma memória composta de 3 áreas:
  - **Área de Código**
    - Vetor que conterá as instruções geradas pelo compilador
  - **Área de Dados**
    - Pilha que conterá os valores manipulados pelas instruções da MaqHipo
  - **Área dos Registradores de base**
    - Pilha que armazena dados sobre o escopo
  - Supõe-se que cada área tem endereços numerados com 0, 1, 2, ...

# Características da MaqHipo

- Área de Código
  - Simulada como um array (chamaremos de C)
  - Cada registro do array é uma instrução ou pseudo-código
    - a Geração de Código consiste no preenchimento desse array, que será "interpretado" posteriormente

# Características da MaqHipo

- Área de Dados
  - Simulada por uma pilha (chamaremos de **D**)
  - Armazena os valores manipulados pelas instruções
  - **Só existirá realmente durante a interpretação do código gerado**
  - Toda instrução agirá sobre o topo desta pilha ou sobre o topo e seu antecessor (no caso de operações binárias)

# Características da MaqHipo

- Área dos Registradores de Base
  - Pilha de registradores (chamaremos de **B**)
  - Cada registrador, quando ativo, aponta para um índice de **D** que marca o início de um novo escopo
    - No caso de LALG, só haverá 2 casos possíveis:
      - Ou o escopo é 0 (programa principal) ou 1 (procedimento declarado no programa principal)
      - Portanto, a pilha B não é necessária
      - Se o aninhamento de procedimentos fosse permitido, haveria diferentes bases, que marcariam o início dos valores manipulados pelas instruções daquele escopo, na pilha de dados D

# Características da MaqHipo

- Além das 3 áreas, a MaqHipo possui registradores especiais:
  - o registrador de programa **i** aponta para próxima instrução a ser executada, portanto **C(i)**
  - o registrador **s** indica o topo da pilha **D**, portanto **D(s)**
  - o registrador **b** indica o topo da pilha **B**, portanto **B(b)**
    - Assim como a pilha B, o registrador b não será necessário no nosso projeto

# Características da MaqHipo

- Uma vez que o programa da MaqHipo está carregado na região C e os registradores inicializados, o funcionamento da máquina é muito simples:
  - As instruções indicadas pelo registrador i são executadas até que seja encontrada a instrução de parada, ou ocorra algum erro. A execução de cada instrução aumenta de 1 o valor de i, exceto as instruções que envolvem desvios
  - O array C será construído pelo compilador que o gera como saída. Esta saída passa a ser a entrada de um programa interpretador deste código. As áreas D e B e os registradores i, s, b, estão fora do escopo do compilador e portanto são definidos e manipulados pelo interpretador

# Características da MaqHipo

- Como a área D só manipula dados e LALG só manipula inteiros e reais, então

  var
  D: array [0 .. tpilha] of real;
  s: -1 .. tpilha;

- A Tabela de Símbolos – TS – deverá ser aumentada com 2 novos campos: end_rel (endereço relativo à base na pilha D) e prim_instr (endereço da 1ª instrução no array C, no caso do identificador ser do tipo procedimento).

# Repertório de Instruções

- Os slides a seguir apresentam o conjunto de instruções para MaqHipo
- Ao lado de cada instrução, há a ação que deve ser tomada na pilha de Dados, durante sua interpretação

# Repertório de Instruções

- Instruções para Avaliação de Expressões

  ex.: 1- a + b

  CRCT 1

  CRVL a* (endereço de a na pilha D, obtido da TS)

  SUBT

  CRVL b*

  SOMA

  *(1) CRCT k*

  {carrega constante k no topo da pilha D}

  s:= s+1;

  D[s]:= k

# Instruções

*(2) CRVL n*

{carrega valor de endereço n no topo da pilha D}
s:= s+1;
D[s]:= D[n]

*(3) SOMA*

{soma o elemento antecessor com o topo da pilha;
desempilha os dois e empilha o resultado}
D[s-1]:= D[s-1] + D[s];
s:= s-1

*(4) SUBT*

{subtrai o antecessor pelo elemento do topo}
D[s-1]:= D[s-1] - D[s];
s:= s-1

# Instruções

*(5)MULT*

{multiplica elemento antecessor pelo elemento do topo}
D[s-1]:= D[s-1] * D[s];
s:= s-1

*(6)DIVI (opcionalmente, podemos ter um DIVR – divisão real)*

{divide o elemento antecessor pelo elemento do topo}
D[s-1]:= D[s-1] div D[s];
s:= s-1

*(7)INVE*

{inverte sinal do topo}
D[s]:= - D[s]

# Instruções

*(8)CONJ*
{conjunção de valores lógicos. F=0; V=1}
se D[s-1] = 1 e D[s] = 1 então D[s-1]:= 1
senão D[s-1]:= 0;
s:= s-1

*(9)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

*(10) NEGA*
{negação lógica}
D[s]:= 1 - D[s]

# Instruções

*(11) CPME*

{comparação de menor entre o antecessor e o topo}
se D[s-1] < D[s] então D[s-1]:= 1
senão D[s-1]:= 0;
s:= s-1

*(12) CPMA*

{comparação de maior}
se D[s-1] > D[s] então D[s-1]:= 1
senão D[s-1]:= 0;
s:= s-1

*(13) CPIG*

{comparação de igualdade}
se D[s-1] = D[s] então D[s-1]:= 1
senão D[s-1]:= 0;
s:= s-1

# Instruções

*(14) CDES*
{comparação de desigualdade}
se D[s-1] <> D[s] então D[s-1]:= 1
senão D[s-1]:= 0;
s:= s-1

*(15) CPMI*
{comparação menor-igual }
se D[s-1] <= D[s] então D[s-1]:= 1
senão D[s-1]:= 0;
s:= s-1

*(16) CMAI*
{comparação maior-igual}
se D[s-1] >= D[s] então D[s-1]:= 1
senão D[s-1]:= 0;
s:= s-1

# Instruções

- Comando de Atribuição
  - V:=E

    onde E é uma expressão que será previamente avaliada, cujo resultado, portanto, se encontra no topo da pilha

    ex: a := a + 1

    CRVL a*
    CRCT 1
    SOMA
    ARMZ a*

    **Obs.**: Resultado de qquer operação deve estar sempre no topo da pilha

    *(17) ARMZ n*

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

    D[n]:= D[s];

    s:= s-1

# Instruções

- Comandos Condicionais e Iterativos
  - Instruções de Desvio – em while e if
  - If E then $C_1$ else $C_2$

    **... } E**
    **DSVF k1**
    **...} $C_1$**
    **DSVI k2**
    **k1 ... } $C_2$**
    **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 array C para substituir k1 e k2 por índices reais

# Exemplos

*If E then C*

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

*While E do C*

k1 ... } E
DSVF k2 ⟶ determinado a posteriori
... } C
DSVI k1 ⟶ determinado a priori
k2 ...

# Instruções

*(18) DSVI p*

{desvio incondicional para a instrução de endereço p}

i:= p

*(19) DSVF p*

{desvio condicional para a instrução de endereço p; o desvio será executado caso a condição resultante seja falsa; o valor da condição estará no topo}

se D[s]=0 então i:= p

s:= s-1

# Instruções

- Comandos de E/S

Read a, b
LEIT
ARMZ a*
LEIT
ARMZ b*

endereços na pilha D que foram guardados na TS durante a declaração de a e b (ALME m)

Write x, x*y
CRVL x
IMPR
CRVL x
CRVL y
MULT
IMPR

# Instruções

*(20) LEIT*

{lê um dado de entrada para o topo da pilha}

s:= s+1;

D[s]:= "valor da entrada"

*(21) IMPR*

{imprime valor o valor do topo da pilha na saída}

"imprimir D[s]";

s:= s-1

# Instruções

- Alocação de Memória

  *(22) ALME m*

  {reserva m posições na pilha D; m depende do tipo da variável}

  s:= s + m

  - **Efeito colateral:** O valor de s é armazenado no campo de end_rel da variável correspondente, na TS

  Ex:

  programa exemplo;

  var a, b: integer → ALME 1

  ALME 1

  - Ao mesmo tempo, na TS, coloca-se no campo de end_rel de a e b, os valores s+1 e s+2 (0 e 1), respectivamente (pois são as primeiras variáveis declaradas)

# Instruções

- Inicialização e Finalização

*(23) INPP*

{inicia programa – será sempre a 1ª instrução}

s:= -1

*(24) PARA*

{termina a execução do programa}

# Exemplo

Endereços das variáveis:
n(0), K(1), f1(2), f2(3), f3(4)

| | | | |
|---|---|---|---|
| *program exe1;* | 0. INPP | *begin* | |
| *var n, k : integer;* | 1. ALME 1 | *f3 := f1 + f2;* | 18. CRVL 2 |
| | 2. ALME 1 | | 19. CRVL 3 |
| *var f1, f2, f3 : real;* | 3. ALME 1 | | 20. SOMA |
| | 4. ALME 1 | | 21. ARMZ 4 |
| | 5. ALME 1 | *f1 := f2;* | 22. CRVL 3 |
| *begin* | | | 23. ARMZ 2 |
| *read(n);* | 6. LEIT | *f2 := f3;* | 24. CRVL 4 |
| | 7. ARMZ 0 | | 25. ARMZ 3 |
| *f1 := 0;* | 8. CRCT 0 | *k := k + 1;* | 26. CRVL 1 |
| | 9. ARMZ 2 | | 27. CRCT 1 |
| *f2 := 1;* | 10. CRCT 1 | | 28. SOMA |
| | 11. ARMZ 3 | | 29. ARMZ 1 |
| *k := 1;* | 12. CRCT 1 | *end;* | 30. **DSVI 14** |
| | 13. ARMZ 1 | *write(n);* | **31**. CRVL 0 |
| *while k <= n do* | **14**. CRVL 1 | | 32. IMPR |
| | 15. CRVL 0 | *write(f1);* | 33. CRVL 2 |
| | 16. CPMI | | 34. IMPR |
| | 17. **DSVF 31** | *end.* | 35. PARA |