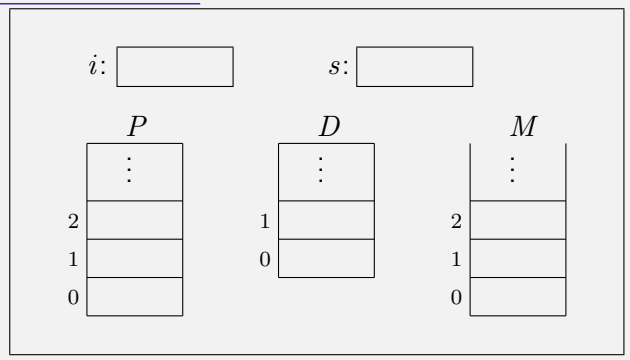
Máquina Virtual

MEPA: Máquina de Execução para Pascal

- Máquina virtual
- Permite separar idiossincrasias da arquitetura real
- Concentração nos aspectos semânticos da linguagem
- Facilita portabilidade
- Facilita o projeto
- Desenvolvimento gradual para justificar as decisões
- Implementações possíveis:
  - interpretador
  - montagem (*assembly*)

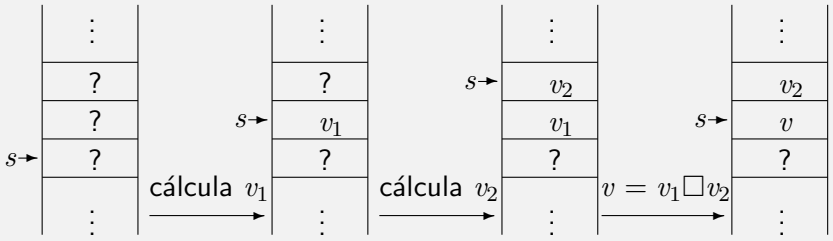
Características gerais



- Máquina a pilha (recursão)
- P – região de programa (instruções da MEPA); formato não especificado
- M – região de pilha de dados (inteiros); sem limites
- i – registrador de programa, inicialmente 0 (uso $P[i]$)
- s – registrador de topo da pilha (uso $M[s]$)
- D – registradores de base (*display*)

Expressões

- Tradução de uma expressão da forma $E_1 \square E_2$, onde $\square$ é um operador binário (segue a lógica de expressões pós-fixas):
  - instruções que implementam E_1 (valor v_1 no topo da pilha)
  - instruções que implementam E_2 (valor v_2 no topo da pilha)
  - instrução que implementa $\square$: desempilha v_1 e v_2 e empilha $v = v_1 \square v_2$
- Esquematicamente:



- Regra geral: a avaliação de qualquer expressão faz o topo da pilha subir de uma unidade, com o resultado no topo.
- Tradução de operadores unários é análoga e segue a mesma regra.
- Tradução de outras construções (constantes, chamadas de funções) também deverá seguir a mesma regra.

Instruções para expressões (exemplos)

CRCT k (Carregar constante):
$s \leftarrow s + 1; M[s] \leftarrow k$

CRVL n (Carregar valor):
$s \leftarrow s + 1; M[s] \leftarrow M[n]$

SOMA (Somar):
$M[s-1] \leftarrow M[s-1] + M[s]; s \leftarrow s - 1$

SUBT (Subtrair):
$M[s-1] \leftarrow M[s-1] - M[s]; s \leftarrow s - 1$

INVR (Inverter sinal):
$M[s] \leftarrow -M[s]$

CMME (Comparar menor):
$M[s-1] \leftarrow$ **se** $M[s-1] < M[s]$ **então** 1 **senão** $0; s \leftarrow s - 1$

Obs.: A instrução *CRVL* será modificada mais adiante.

Exemplo de tradução de expressão

- Expressão: $a + (b \textbf{ div } 9 - 3) * c$
- Suposição: os endereços atribuídos pelo compilador às variáveis são a: 100, b: 102 e c: 99
- Tradução:

```
CRVL  100
CRVL  102
CRCT  9
DIVI
CRCT  3
SUBT
CRVL  99
MULT
SOMA
```

- Verifica-se facilmente que o nível da pilha subirá de 1

Comando de atribuição

- Forma do comando: $v := E$, onde v é uma variável simples e E é uma expressão qualquer
- A avaliação da expressão deixa o valor no topo da pilha
- Instrução de armazenamento (será modificada mais adiante):
  ARMZ n (Armazenar valor):
  $M[n] \leftarrow M[s]; s \leftarrow s - 1$
- Exemplo (com os mesmos endereços): $a := a + b * c$

```
CRVL  100
CRVL  102
CRVL  99
MULT
SOMA
ARMZ  100
```

- O nível final da pilha não muda
- Todos os comandos deverão seguir esta regra

Comandos condicionais e iterativos

- Instruções adicionais (*NADA* apenas por conveniência):
  DSVS p (Desviar sempre):
  $i \leftarrow p$
  DSVF p (Desviar se falso):
  se $M[s] = 0$ **então** $i \leftarrow p$ **senão** $i \leftarrow i + 1; s \leftarrow s - 1$
  NADA (Nada):
- Utilizaremos rótulos simbólicos em lugar de endereços de programa (interpretador da MEPA aceita rótulos)
- Traduções

if E **then** C_1 **else** C_2:

```
    ...      (E)
    DSVF  L1
    ...      (C1)
L1: NADA
    ...      (C2)
L2: NADA
```

while E **do** C:

```
L1: NADA
    ...      (E)
    DSVF  L2
    ...      (C)
    DSVS  L1
L2: NADA
```

- Verifica-se facilmente que estes comandos mantêm o nível da pilha.

Exemplo de comando condicional

```
if a>b then q := p and q
    else if a< 2*b then p := true
            else q := false
```

```
        CRVL  A          CRVL  B
        CRVL  B          MULT
        CMMA             CMME
        DSVF  L3         DSVF  L5
        CRVL  P          CRCT  1
        CRVL  Q          ARMZ  P
        CONJ             DSVS  L6
        ARMZ  Q     L5:  NADA
        DSVS  L4         CRCT  0
   L3:  NADA             ARMZ  Q
        CRVL  A     L6:  NADA
        CRCT  2     L4:  NADA
```

Obs.: As letras A, B, ... correspondem aos endereços atribuídos pelo compilador às variáveis a, b, ...

Exemplo de comando repetitivo

```
while s<=n do
  s := s+3
```

```
   L1:  NADA
        CRVL  S
        CRVL  N
        CMEG
        DSVF  L2
        CRVL  S
        CRCT  3
        CRVL  S
        MULT
        SOMA
        ARMZ  S
   L8:  NADA
```

Obs.: Novamente, é fácil verificar que estes comandos mantêm o nível da pilha.

Entrada e saída

- Instruções adicionais:

  LEIT (Leitura):
  $s \leftarrow s + 1$; $M[s] \leftarrow$ próximo valor de entrada

  IMPR (Impressão):
  imprimir $M[s]$; $s \leftarrow s - 1$

- Exemplo:

```
read(a,b,c);
write(a+b*c)
```

```
LEIT          CRVL  A
ARMZ  A       CRVL  B
LEIT          CRVL  C
ARMZ  B       MULT
LEIT          SOMA
ARMZ  C       IMPR
```

- Verifica-se novamente que estes comandos mantêm o nível da pilha.

Programas simples

- Instruções adicionais (ambas a serem modificadas mais adiante):

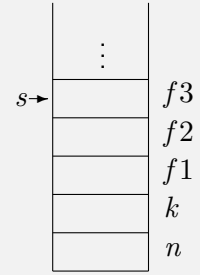
  INPP (Iniciar programa principal):
  $s := -1$

  AMEM m (Alocar memória):
  $s \leftarrow s + m$

  PARA (Parar):
  pára a execução da MEPA

- Exemplo:

```
program Exemplo;
  var n, k: integer;
    f1, f2, f3: integer;
begin
  read(n);
  f1 := 0; f2 := 1; k := 1;
  while k<=n do begin
      f3 := f1+f2;
      f1 := f2; f2 := f3;
      k := k+1
      end;
  write(n,f1)
end.
```



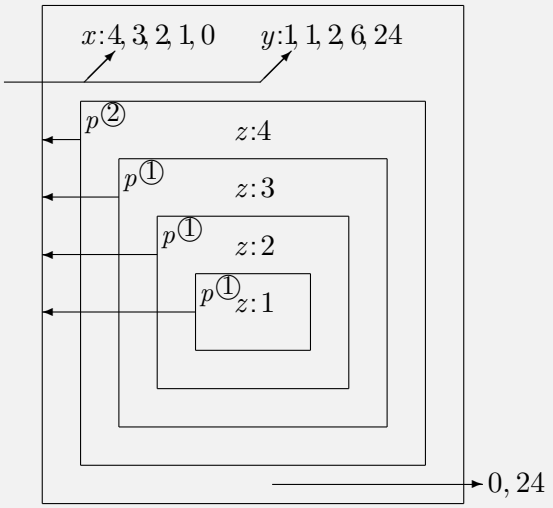
Programas simples (cont.)

```
      INPP          program
      AMEM  2       var n, k
      AMEM  3       f1, f2, f3
      LEIT
      ARMZ  0       read(n)
      CRCT  0
      ARMZ  2       f1 := 0
      CRCT  1
      ARMZ  3       f2 := 1
      CRCT  1
      ARMZ  1       k := 1
L1:   NADA          while
      CRVL  1
      CRVL  0
      CMEG          k <= n
      DSVF  L2      do
      CRVL  2
      CRVL  3
      SOMA
      ARMZ  4       f3 := f1 + f2
      CRVL  3
      ARMZ  2       f1 := f2
      CRVL  4
      ARMZ  3       f2 := f3
      CRVL  1
      CRCT  1
      SOMA
      ARMZ  1       k := k + 1
      DSVS  L1
L2:   NADA
      CRVL  0
      IMPR
      CRVL  2
      IMPR          write(n,f1)
      PARA          end.
```


Procedimentos sem parâmetros

Exemplo:

```
program Exemplo;
  var x, y: integer;
  procedure p;
    var z: integer;
  begin
    z := x; x := x−1;
    if z>1 then p①
           else y := 1;
    y := y∗z
  end; {p}
begin
  read(x);
  p②;
  write(x,y)
end.
```



Procedimentos sem parâmetros (cont.)

- Problema: coexistem várias instâncias da variável z.
- O número de instâncias é imprevisível – depende dos dados.
- Acesso à uma única instância (última) dentro de cada ativação de p.
- Na versão atual da MEPA, a variável z deve ter endereço fixo.
- Solução: salvar e restaurar os valores das variáveis locais na pilha.
- Redefinição e instruções adicionais:

AMEM m,n (Alocar memória):
para $k := 0$ **até** $n-1$ **faça**
$\{s := s+1; M[s] := M[m+k]\}$

DMEM m,n (Desalocar memória):
para $k := n-1$ **até** 0 **faça**
$\{M[m+k] := M[0]; s := s-1\}$

CHPR p (Chamar procedimento):
$s := s+1; M[s] := i+1; i := p$

RTPR p (Retornar de procedimento):
$i := M[s]; s := s-1$

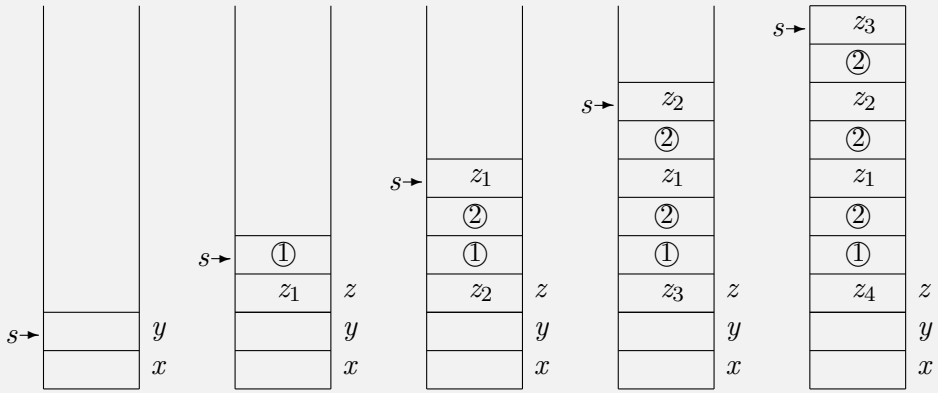
- Notar que a instrução *RTPR* sempre enontra a informação correta no topo da pilha (comandos deixam a pilha no nível original).

Procedimentos sem parâmetros (cont.)

```
      INPP          program
      AMEM  0,2     var x, y
      DSVS  L1
L2:   NADA          procedure p
      AMEM  2,1     var z
      CRVL  0
      ARMZ  2       z := x
      CRVL  0
      CRCT  1
      SUBT
      ARMZ  0       x := x − 1
      CRVL  2
      CRCT  1
      CMMA          if z > 1
      DSVF  L3      then
      CHPR  L2  p
      DSVS  L4
L3:   NADA          else
      CRCT  1
      ARMZ  1       y := 1
L4:   NADA
      CRVL  1
      CRVL  2
      MULT
      ARMZ  1       y := y ∗ z
      DMEM  2,1     end
      RTPR
L1:   NADA
      LEIT
      ARMZ  0       read(x)
      CHPR  L2  p
      CRVL  0
      IMPR
      CRVL  1
      IMPR          write(x,y)
      DMEM  0,2     end.
      PARA
```

Procedimentos sem parâmetros (cont.)

- Configurações da pilha inicial e após cada chamada e alocação; os endereços de retorno empilhados estão indicados de maneira simbólica:

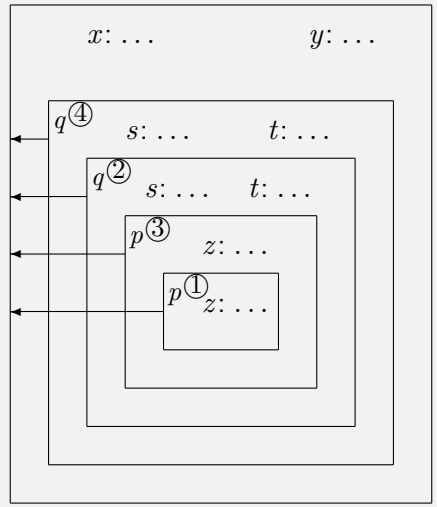


- Após cada retorno (precedido de desalocação), a configuração anterior será restaurada.

Procedimentos sem parâmetros (cont.)

Outro exemplo (esboço):

```
program Exemplo;
  var x, y:
  procedure p;
    var z;
  begin
  ... p① ...
  end {p};
  procedure q;
    var s, t: integer;
  begin
  ... q② ... p③ ...
  end {q};
begin
... q④ ...
end.
```



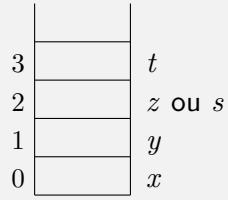
Procedimentos sem parâmetros (cont.)

- O exemplo demonstra que as variáveis z (do procedimento p) e as variáveis s e t (do procedimento q) nunca são acessíveis ao mesmo tempo. Conseqüentemente, o compilador pode atribuir a estas variáveis endereços compartilhados; os seus conteúdos serão salvos e restaurados quando necessário. Assim, os endereços das variáveis seriam:

x: 0	z: 2	s: 2
y: 1		t: 3

- As instruções de alocação correspondentes seriam:

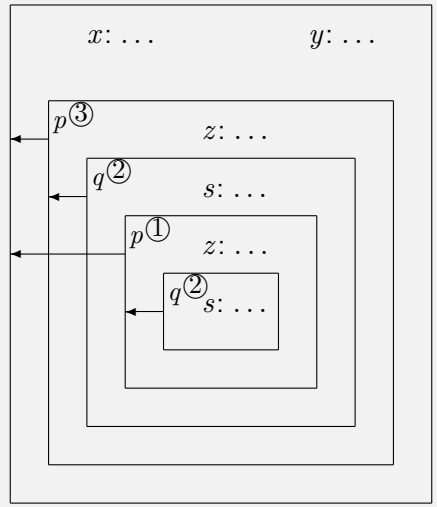
AMEM $0, 2$ **var** x, y
...
AMEM $2, 1$ **var** z
...
AMEM $2, 2$ **var** s, t



Procedimentos sem parâmetros (cont.)

Mais um exemplo (esboço):

```
program Exemplo;
  var x, y: integer;
  procedure p;
    var z: integer;
    procedure q;
      var s: integer;
    begin
    ... p① ...
    end {q};
  begin
  ... q② ...
  end {p};
begin
... p③ ...
end.
```



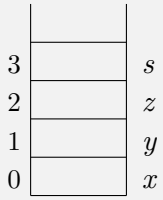
Procedimentos sem parâmetros (cont.)

- Neste exemplo, o procedimento q está encaixado dentro do procedimento p; conseqüentemente, durante a execução de q, as variáveis de p podem ser acessadas. Assim, as variáveis de p e de q não podem compartilhar os mesmos endereços da pilha e sua atribuição seria:

  x: 0 z: 2 s: 3
  y: 1

- As instruções de alocação correspondentes seriam:

  AMEM $0, 2$ **var** x, y
  ...
  AMEM $2, 1$ **var** z
  ...
  AMEM $3, 1$ **var** s

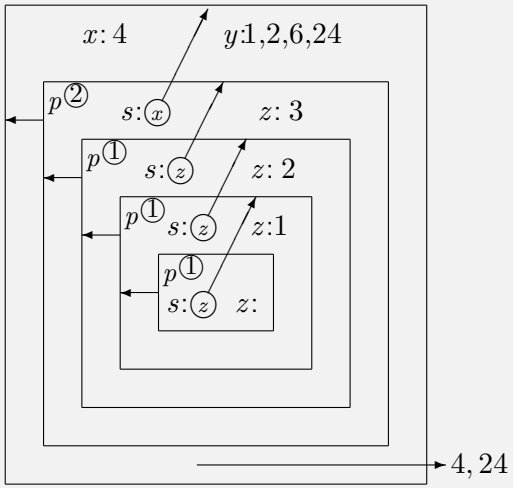


Observações

- O projeto corrente da MEPA poderia ser estendido para procedimentos (ou funções) com parâmetros passados **estritamente** por valor, isto é, parâmetros que não denotem referências a outras variáveis ou parâmetros.
- Conceitualmente, parâmetros passados por valor são variáveis locais inicializadas com os valores passados na chamada.
- Os valores dos parâmetros (resultado de cálculo de expressões antes da chamada) estariam nas posição da pilha imediatamente abaixo do endreço de retorno, em posições conhecidas.
- O tratamento destas variáveis poderia ser semelhante ao de variáveis, definindo-se instruções para salvar e restaurar posições correspondentes.
- Em certos casos (objetos grandes como matrizes) esta técnica de salvar e restaurar pode trazer muita ineficiência.
- Ver implementação completa na Seção 8.9 do texto de referência.

Parâmetros por referência

```
program Exemplo;
  var x, y: integer;
  procedure p(var s: integer);
    var z: integer;
  begin
    if s=1
      then y:=1
      else begin
             z := s−1;
             p(z)①;
             y := y∗s
           end
  end {p};
begin
  x := 4;
  p(x)②;
  write(x,y)
end.
```

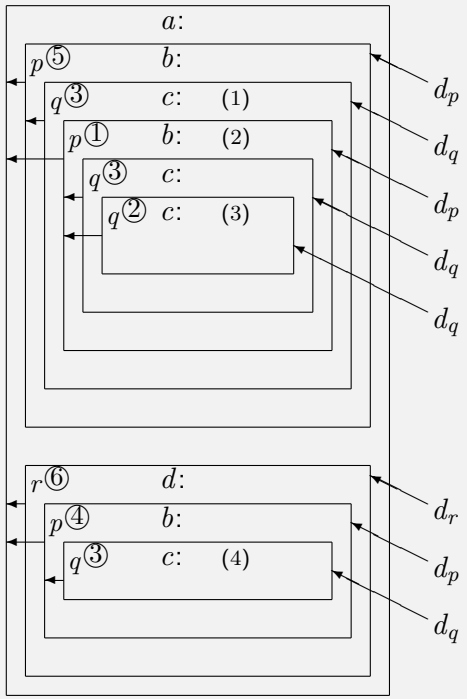


Parâmetros por referência (cont.)

- O exemplo demonstra que, num dados instante, dentro do corpo do procedimento p, pode haver acesso a duas instâncias da variável z: à corrente e à de uma ativação anterior, através do parâmetro formal s.
- A MEPA, no estágio atual, atribuiria o mesmo endereço fixo a todas as instâncias da variável z, tornando difícil acesso simultâneo a mais de uma instância.
- O problema está no conceito de endereço fixo (absoluto).
- A MEPA será modificada de maneira a associar com cada variável endereçamento relativo (deslocamento) dentro de cada procedimento (incluindo o programa principal).
- Um *registrador de base* será associado com cada procedimento e conterá endereço do início das suas variáveis no instante de ativação (*registro de ativação*).
- O par (registrador de base, deslocamento) forma o *endereço textual* de uma variável.
- Para ilustrar, consideraremos um exemplo mais complexo, mas com procedimentos sem parâmetros que serão introduzidos depois.

Endereçamento relativo

```
program Exemplo;
  var a: integer;
  procedure p;
    var b: integer;
    procedure q;
      var c: integer;
    begin
    ... p①; ... q②; ...
    end {q};
  begin ... q③; ... end {p};
  procedure r;
    var d: integer;
  begin ... p④; ... end {r};
begin
  ... p⑤;
  ... r⑥;
  ...
end.
```

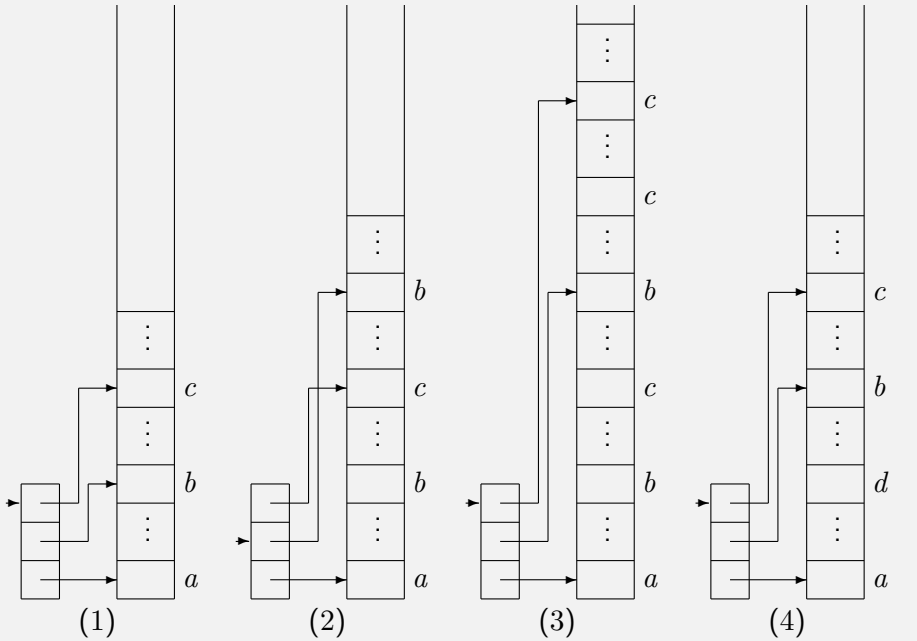


Endereçamento relativo (cont.)

- Os registradores de base d_p, d_q e d_r poderiam ser todos distintos, mas esta política acarretaria um número exagerado de registradores num programa grande (centenas ou milhares de procedimentos).
- No exemplo, d_p e d_q têm que ser distintos porque existe acesso simultâneo às variáveis dos procedimentos respectivos (dentro de q.
- Dentro do procedimento r, não existe acesso às variáveis de p e de q, e dentro de p e de q não existe acesso às variáveis de r (ver flechas estáticas).
- Nota-se que é possível atribuir registradores aos procedimentos com base no encaixamento estático (disciplina de pilha):

  d_p: $D[1]$ d_q: $D[2]$ d_r: $D[1]$
- Por questão de uniformidade, será atribuído o registrador $D[0]$ ao programa principal cujas variáveis também serão referenciadas por endereços textuais.
- Serão redefinidas várias instruções, particularmente as que manipulam variáveis.

Endereçamento relativo: pilha



Endereçamento relativo: redefinições

- Acesso às variáveis: deve levar em consideração o enedereçamento textual

  CRVL m, n (Carregar valor):
  $s := s+1; M[s] := M[D[m]+n]$

  ARMZ m, n (Armazenar valor):
  $M[D[m]+n] := M[s]; s := s-1$
- Alocação de memória: ficou mais simples porque as variáveis não precisam mudar de lugar

  AMEM n (Alocar memória):
  $s := s+n$

  DMEM n (Desalocar memória):
  $s := s-n$

Endereçamento relativo: redefinições (cont.)

- Procedimentos: a instrução de chamada *CHPR* não mudou; *ENPR* acerta a entrada do vetor de registradores de base; *RTPR* restaura esta entrada

  CHPR p (Chamar procedimento):
  $s := s+1; M[s] := i+1; i := p$

  ENPR k (Entrar no procedimento):
  $s := s+1; M[s] := D[k]; D[k] := s+1$

  RTPR k (Retornar do procedimento):
  $D[k] := M[s]; i := M[s-1]; s := s-2$
- A instrução *RTPR* será modificada em breve.

Endereçamento relativo: parâmetros

- Problema original: parâmetros passados por referência.
- Solução: empilhar todos os parâmetros:
  - quando por valor, os próprios resultados das expressões
  - quando por referência, endereços das variáveis na pilha
- Instruções:

  CREN m, n (Carregar endereço):
  $s := s+1; M[s] := D[m]+n$

  CRVI m, n (Carregar valor indiretamente):
  $s := s+1; M[s] := M[M[D[m]+n]]$

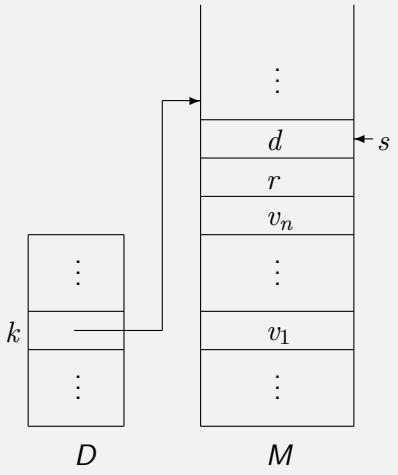
  ARMI m, n (Armazenar indiretamente):
  $M[M[D[m]+n]] := M[s]; s := s-1$

  RTPR k, n (Retornar do procedimento):
  $D[k] := M[s]; i := M[s-1]; s := s-(n+2)$
- O programa é tratado como um procedimento de nível 0:

  INPP (Iniciar programa principal):
  $s := -1; D[0] := 0$

Configuração da MEPA: procedimento

Após uma chamada de procedimento da forma $p(e_1, e_2, \ldots, e_n)$ e a execução da instrução *ENPR k*, a configuração esquemática da MEPA será:



onde:

- k é o nível textual do procedimento p
- $v_1, \ldots, v_n$ são os valores calculados e empilhados de $e_1, \ldots, e_n$ (deslocamentos negativos)
- r é o endereço de retorno
- d é o valor prévio de $D[k]$ salvo na pilha
- $D[k]$ aponta para a primeira variável de p (se existir)

Exemplo com procedimento

```
program Exemplo;
  var k: integer;
  procedure p(n: integer; var g: integer);
    var h: integer;
  begin
    if n<2 then  g := g+n
           else begin
                  h := g;
                  p(n−1,h);
                  g := h;
                  p(n−2,g)
                  end;
    write(n,g)
  end {p∗};
begin
  k := 0;  p(3,k)
end.
```

Exemplo com procedimento: tradução

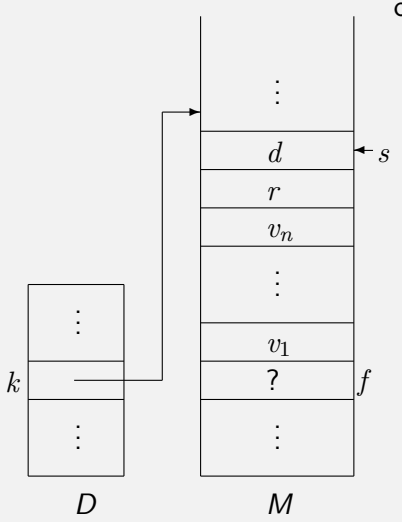
```
      INPP                  program
      AMEM    1             var k
      DSVS    L1
L2:   ENPR    1             procedure p
      AMEM    1             var h
      CRVL    1, −4
      CRCT    2
      CMME                  if n < 2
      DSVF    L3            then
      CRVI    1, −3
      CRVL    1, −4
      SOMA
      ARMI    1, −3  g := g + n
      DSVS    L4
L3:   NADA                  else
      CRVI    1, −3
      ARMZ    1, 0   h := g
      CRVL    1, −4
      CRCT    1
      SUBT
      CREN    1, 0
      CHPR    L2     p(n − 1, h)
      CRVL    1, 0
      ARMI    1, −3  g := h
      CRVL    1, −4
      CRCT    2
      SUBT
      CRVL    1, −3
      CHPR    L2     p(n − 2, g)
L4:   NADA
      CRVL    1, −4
      IMPR
      CRVI    1, −3
      IMPR                  write(n, g)
      DMEM    1             end
      RTPR    1, 2
L1:   NADA
      CRCT    0
      ARMZ    0, 0   k := 0
      CRCT    3
      CREN    0, 0
      CHPR    L2     p(3, k)
      DMEM    1             end.
      PARA
```


Funções

- O valor final da chamada deve ficar no topo da pilha, coerente com a implementação de expressões.
- Reserva de uma posição antes de avaliar os parâmetros (*AMEM 1*).
- Dentro do corpo da função, esta posição será usada como o zero-ésimo parâmetro.
- O resto é idêntico às chamadas de procedimentos.

Configuração da MEPA: função

Após uma chamada de função da forma $f(e_1, e_2, \ldots, e_n)$ e a execução da instrução *ENPR k*, a configuração esquemática da MEPA será:



onde:

- k é o nível textual da função f
- f é a posição reservada para o resultado (deslocamento negativo)
- $v_1, \ldots, v_n$ são os valores calculados e empilhados de $e_1, \ldots, e_n$ (deslocamentos negativos)
- r é o endereço de retorno
- d é o valor prévio de $D[k]$ salvo na pilha
- $D[k]$ aponta para a primeira variável de f (se existir)

Exemplo com função

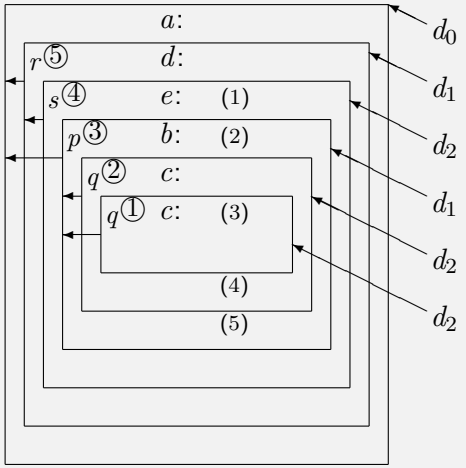
```
program Exemplo4;
  var m: integer;
  function f(n: integer; var k: integer): integer;
    var p, q: integer;
  begin
    if n<2
      then begin f := n; k := 0 end
      else  begin
              f := f(n−1,p)①+f(n−2,q)②;
              k := p+q+1;
            end;
    write(n,k)
  end; {f}
begin
  write(f(3,m)③,m)
end.
```

Exemplo com função: tradução

```
      INPP              program
      AMEM   1          var m
      DSVS   L1
L2:   ENPR   1          function f
      AMEM   2
      CVEL   1,-4
      CRCT   2
      CMME              if n < 2
      DSVF   L3         then
      CRVL   1,-4
      ARMZ   1,-5
      CRCT   0
      ARMI   1,-3  k := 0
      DSVS   L4
L3:   NADA              else
      AMEM   1
      CRVL   1,-4
      CRCT   1
      SUBT
      CREN   1,0
      CHPR   L2
      AMEM   1
      CRVL   1,-4
      CRCT   2
      SUBT
      CREN   1,1
      CHPR   L2
      SOMA
      ARMZ   1,-5  f := f(...) + f(...)
      CRVL   1,0
      CRVL   1,1
      SOMA
      CRCT   1
      SOMA
      ARMI   1,-3  k := p + q + 1
L4:   NADA
      CRVL   1,-4
      IMPR
      CRVI   1,-3
      IMPR              write(n, k)
      DMEM   2          end
      RTPR   1,2
L1:   NADA
      AMEM   1
      CRCT   3
      CREN   0,0
      CHPR   L2
      IMPR
      CRVL   0,0
      IMPR              write(f(3, m), m)
      DMEM   1          end.
      PARA
```


Rótulos e comandos de desvio

```
program Exemplo;
  var a: integer;
  procedure p;
    label 100;
    var b: integer;
    procedure q;
      var c: integer;
    begin ...q①; goto 100; ... end {q};
  begin ...q②; ... 100: ... end {p};
  procedure r;
    var d: integer;
    procedure s;
      var e: integer;
    begin ... p③; ... end {s};
  begin ... s④; ... end {r};
begin
... r⑤; ...
end.
```



Rótulos e comandos de desvio (cont.)

- Na versão atual da MEPA, cada chamada de uma rotina de nível k salva o valor de $D[k]$ com a instrução *ENPR k*.
- O valor de $D[k]$ é restaurado no retorno com *RTPR k,n*.
- Os valores de k são dados de maneira explícita pelo código.
- Introduzindo-se desvios, o número de registradores de base a restaurar é imprevisível.
- No exemplo, quando é executado o comando de desvio na configuração (3), devem ser restaurados $D[2]$ (correspondente ao procedimento s) e de $D[1]$ (correspondente ao procedimento p).
- A operação de desvio deverá simular a restauração de um número variável de registradores; para isto precisa encontrar a informação necessária na pilha.
- A instrução *CHPR* será modificada para empilhar também o nível léxico da rotina que executou a chamada.

Rótulos e comandos de desvio: instruções

- m é o nível textual da rotina de origem:
  CHPR p, m (Chamar procedimento):
  $M[s+1] := i+1; M[s+2] := m; s := s+2; i := p$
- *RTPR* leva em conta a existência de mais um campo mas não o utiliza:
  RTPR k, n (Retornar do procedimento):
  $D[k] := M[s]; i := M[s-2]; s := s-(n+3)$
- *DSVR* restaura os registradores de base necessários:
  DSVR p, j, k (Desviar para rótulo):
  $t1 := k;$
  enquanto $t1 \neq j$ **faça**
  $\{t2 := M[D[t1]-2]; D[t1] := M[D[t2]-1]; t1 := t2\};$
  $i := p$
- *ENRT* restaura o nível da pilha:
  ENRT j, n (Entrar no rótulo):
  $s := D[j]+n-1$