

Geração de Código para uma Máquina Hipotética a Pilha

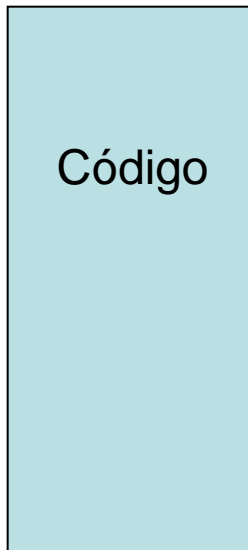
MEPA

Máquina de Execução para Pascal e
sua linguagem de montagem

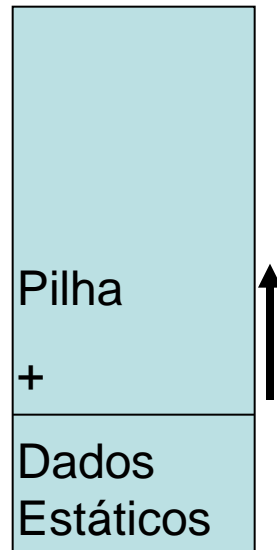
(capítulo 8 e capítulo 9.5 do livro Kowaltowski)

MEPA

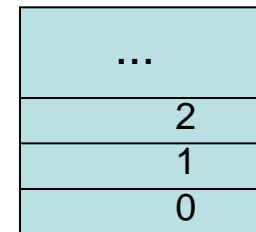
- É composta de 3 Regiões
- Não tem HEAP, pois FRANKIE não permite variáveis dinâmicas



Região de programa:
 $P(i)$



Região da Pilha de
Dados: $M(s)$



Display: $D(b)$

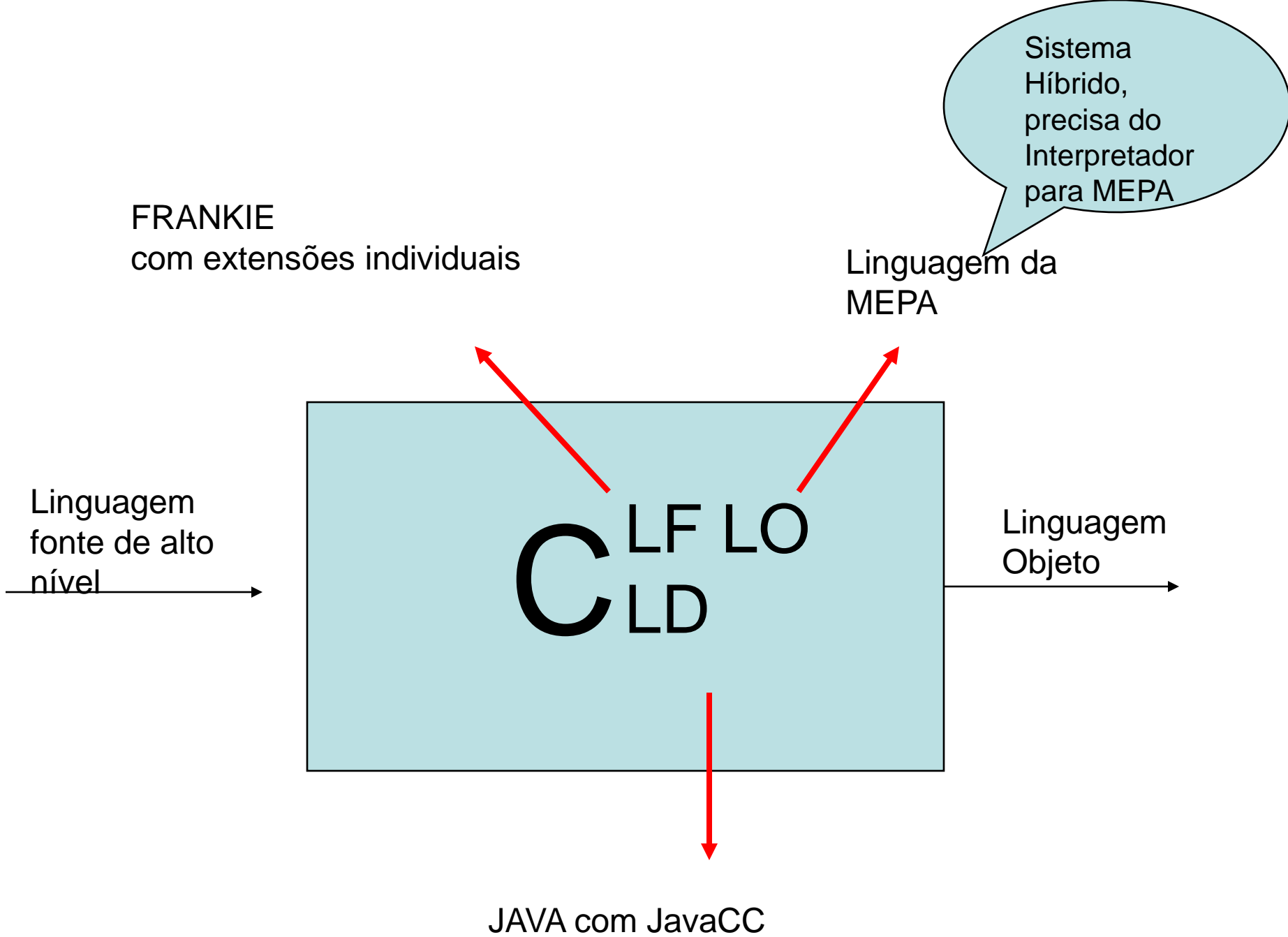
(Vetor dos
Registradores
de Base que
apontam para
M)

Geração de Código

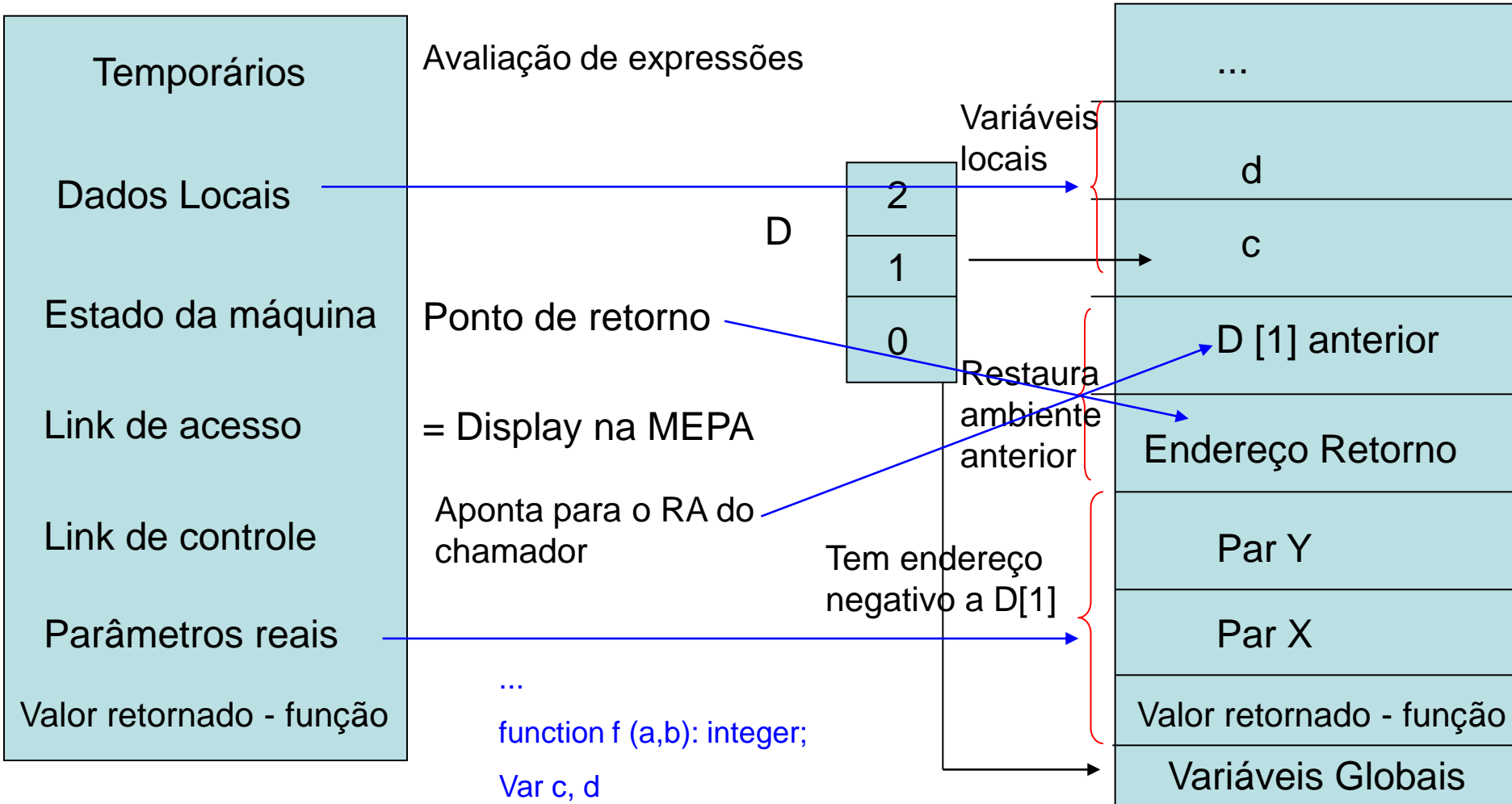
- Uma vez que o programa da MEPA está carregado na região P, e os registradores têm seus valores iniciais, o funcionamento da máquina é muito simples:
 - As instruções indicadas pelo registrador i são executadas até que seja encontrada a instrução de parada, ou ocorra algum erro.
 - A execução de cada instrução aumenta de 1 o valor de i , exceto as instruções que envolvem desvios.
- Em nível de projeto, o **vetor P** será construído pelo **compilador** que o gera como saída. Esta saída passa a ser a entrada de um **programa interpretador** deste código.

Porque falei em vetor P e não arquivo para P???

- As áreas **M e D** e os registradores i, s, b , estão fora do escopo do compilador e, portanto, são definidos e manipulados no programa **interpretador**, como mostra a especificação inicial para nosso projeto:



Registro de ativação para funções



Registro de ativação (RA) geral

```

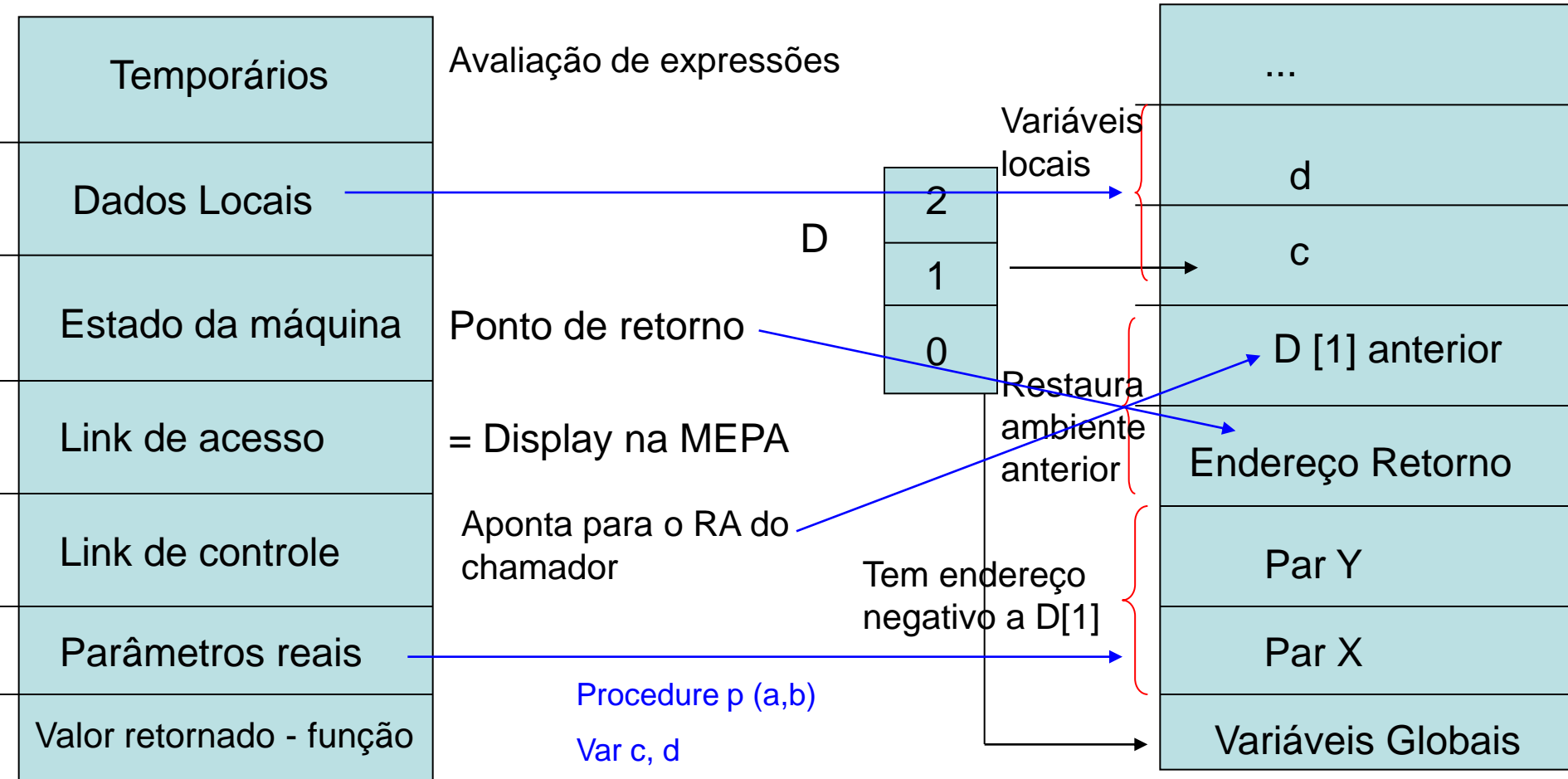
function f (a,b): integer;
Var c, d
Begin .... f:= ...
end;
Begin ...write(f(X,Y));... End.
    
```

M
Na Mepa...

Gerencia as info necessárias para uma única execução de uma função

Registro de ativação para procedimentos

Gerencia as info necessárias para uma única execução de um procedimento



Registro de ativação (RA) geral

Procedure p (a,b)

Var c, d

Begin ... end;

Begin

p(x,y)

End.

M

Na Mepa...

Geração de Código para a MEPA

Veremos agora a geração de código para:

1. Avaliação de expressões
2. Comando de atribuição
3. Comandos Iterativos e Condicionais
4. Procedimentos pré-definidos de E/S
5. Programas sem procedimentos
6. Procedimentos sem parâmetros

Vamos anotar no conjunto de instruções os números acima e redefinir algumas instruções em certos momentos, pois nos moveremos do simples para o complexo

Linguagem de Montagem da MEPA, separados
por contextos de uso

1. Avaliação de expressões – 16 instruções

1. SOMA

2. SUBT

3. MULT

4. DIVI

5. INVR

6. CONJ

7. DISJ

8. NEGA

9. CMME

Usados sem
alteração

10. CMMA

11. CMIG

12. CMDG

13. CMEG

14. CMAG

15. CRCT k

16. CRVL n

$s := s + 1$

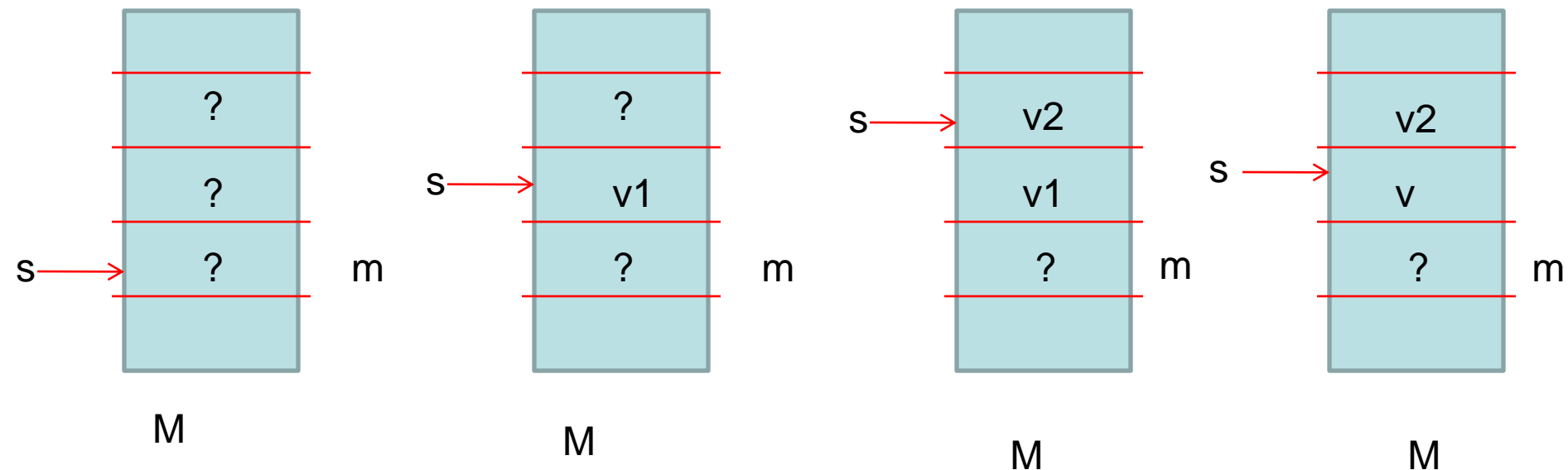
$M[s] := M[n]$

alterado

Uso da notação polonesa (posfixa)

1. Avaliação de expressões - cálculos

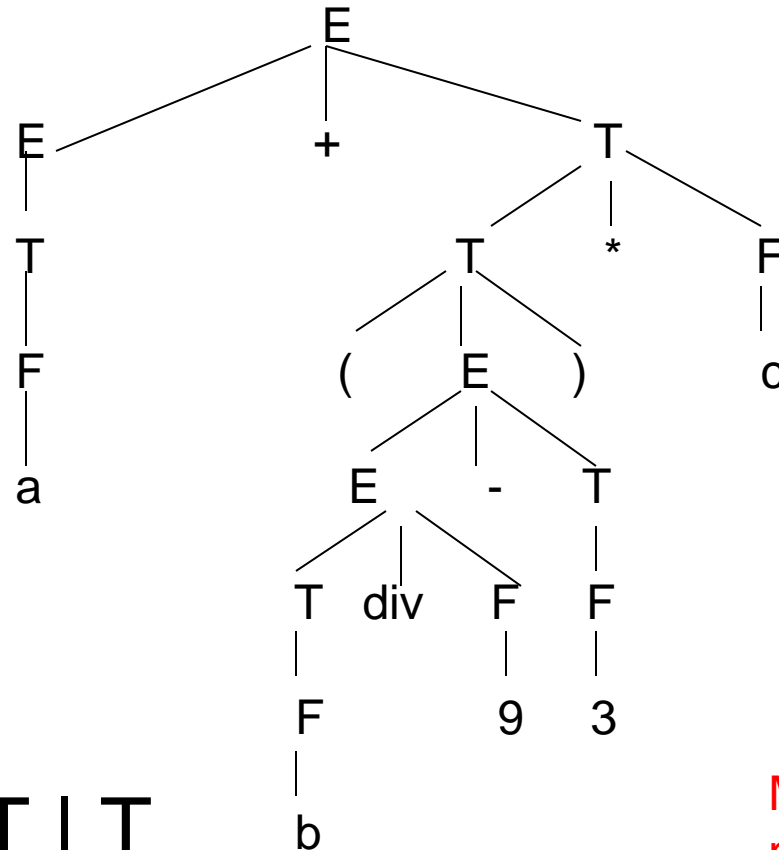
- $E = E1 \text{ op } E2$
- Avaliar $E1$ e $E2$ e depois operar
- Guarda-se os valores $v1$ e $v2$ de $E1$ e $E2$ na própria pilha M , v será o valor final de E



Ex1: $a + (b \text{ div } 9 - 3) * c$
(operadores com prioridade diferente)

- a, b, c estão em 100, 102 e 99
- Com os valores -2, 10 e 100
- $s = 104$
- Observem que o código para expressões fica na notação polonesa posfixa
- **PORQUE????**

Árvore de Derivação para a expressão $a + (b \text{ div } 9 - 3) * c$



$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T \text{ div } F \mid T * F \mid F$

$F \rightarrow (E) \mid \text{not } F \mid \text{ID} \mid \text{NRO}$

Mepa é uma máquina a pilha, assim sempre tem acesso aos elementos do topo/topo -1

Programa-objeto da tradução

CRVL 100 (a)

CRVL 102 (b)

CRCT 9 (c)

DIVI

CRCT 3

SUBT

CRVL 99

MULT

SOMA

Ex2: 1- a + b (operadores de mesma prioridade)

- a, b estão em 100, 102
- Com os valores -2, 10
- $s = 104$

- Mas podemos também deixar os endereços genéricos (A e B)

Programa-objeto da tradução

CRCT	1
CRVL	A (endereço de a na pilha M, obtido da TS)
SUBT	
CRVL	B
SOMA	

2. Comando de atribuição

- Atribuição a variáveis simples: $V := E$
- A expressão E já sabemos traduzir
- n é o endereço atribuído pelo compilador à variável V

ARMZ n

$M[n] := M[s]$ alterado

$s := s - 1$

Ex3: $a := a + b * c$

- a, b, c estão em 100, 102 e 99
- Com os valores -2, 10 e 100
- Observem que o valor final do registrador s após a execução de uma atribuição será igual ao seu valor inicial
- Esta propriedade valerá para todo comando em FRANKIE; exceções: desvio e chamadas de procedimentos e funções

Programa-objeto da tradução

CRVL 100

CRVL 102

CRVL 99

MULT

SOMA

ARMZ 100

3. Comandos Iterativos e Condicionais

- Os esquemas do IF e WHILE usam:

1. DSVS p

2. DSVF p

Usados sem
alteração

3. NADA

p é um número que indica o endereço de programa na MEPA
Aqui usaremos rótulos simbólicos , em vez de números

A diferença entre os comandos em C e Pascal
são somente na forma !!!

If (E) C1 else C2

/ If (E) C

... } tradução de E

DSVF L1

...} tradução de C1

DSVS L2

L1 NADA

... } tradução de C2

L2 NADA

... } tradução de E

DSVF L1

... } tradução de C

L1 NADA

While (E) C

L1 NADA

... } tradução de E

DSVF L2

determinado depois

... } tradução de C

DSVS L1

determinado a priori

L2 NADA

Ex4: if (q) a:= 1 else a := 2

- Neste e nos outros 2 exercícios usaremos os rótulos simbólicos, em ordem crescente: L1, L2, L3

Programa-objeto da tradução

CRVL	Q
DSVF	L1
CRCT	1
ARMZ	A
DSVS	L2
L1	NADA
CRCT	2
ARMZ	A
L2	NADA

```
Ex 5: if (a>b) q := p and q
      else if (a < 2 * b) p := true
            else q := false
```

Programa-objeto da tradução

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

Ex 6: while (s <=n) s := s +3 * s

Programa-objeto da tradução

L7 NADA

CRVL S

CRVL N

CMEG

DSVF L8

CRVL S

CRCT 3

CRVL S

MULT

SOMA

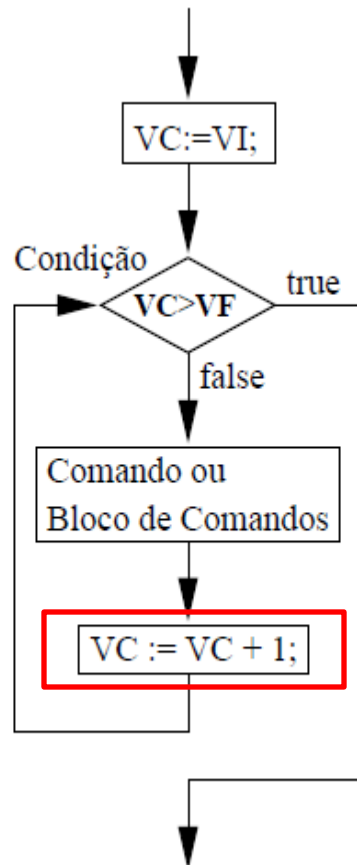
ARMZ S

DSVS L7

L8 NADA

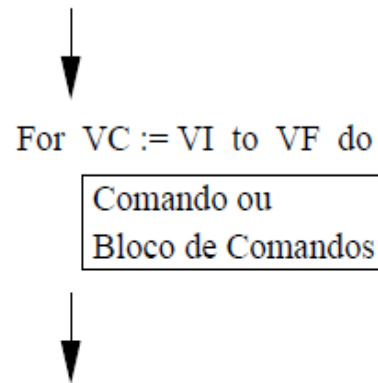
Exercício: Fazer a tradução para o comando FOR e o exercício

<comando repetitivo 2> ::= **for** <identificador> := <expressão>
(**to|downto**) <expressão> **do** <comando>



for $i:=2$ **to** n **do**
 $fatorial:=fatorial * i;$

$fatorial$ 100
 i 101
 n 102



{OBS:}
{VC = Variável de Controle}
{VI = Valor Inicial}
{VF = Valor Final}

For VC := E1 to E2 do C

...} tradução de E1

ARMZ VC

L1 NADA

CRVL VC

...} tradução de E2

CMEG { VC > CF: FIM }

DSVF FIM

...} tradução de C

CRVL VC

CRCT 1

INC

SOMA

ARMZ VC

DSVS L1

FIM NADA

For VC := E1 downto E2 do C

...} tradução de E1

ARMZ VC

L1 NADA

CRVL VC

...} tradução de E2

CMAG { VC < CF: FIM }

DSVF FIM

...} tradução de C

CRVL VC

CRCT 1

DEC

SUBT

ARMZ VC

DSVS L1

FIM NADA

```
for i:=2 to n do
  fatorial:=fatorial * i;
```

```
fatorial 100
i         101
n         102
```

CRCT 2

ARMZ 101

L1 NADA

CRVL 101

CRVL 102

CMEG { VC > CF: FIM }

DSVF FIM

CRVL 100

CRVL 101

MULT

ARMZ 100

```
CRVL 101
CRCT 1
SOMA
ARMZ 101
DSVS L1
FIM NADA
```

INC

E o comando FOR de C é similar?

```
for (expressão1; expressão2; expressão3) comando;
```

As três expressões geralmente tem os seguintes significados:

1. A expressão1 é utilizada para inicializar a variável de controle do laço;
2. A expressão2 é um teste que controla o fim do laço;
3. A expressão3 normalmente faz um incremento ou decremento da variável de controle.

A execução do comando `for` segue os seguintes passos:

1. A expressão1 é avaliada;
2. A expressão2 é avaliada para determinar se o comando deve ser executado;
3. Se o resultado da expressão2 for VERDADE o comando é executado caso contrário o laço é terminado;
4. A expressão3 é avaliada;
5. Voltar para o passo onde 2.

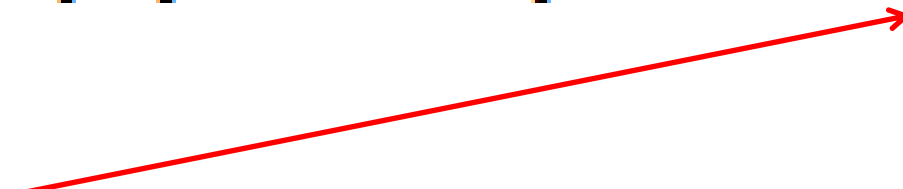
For em C

```
for (i = 0; i <= 100; i++)  
    printf("Numero %d\n", i);
```

É possível omitir qualquer uma das expressões.

Sempre
TRUE

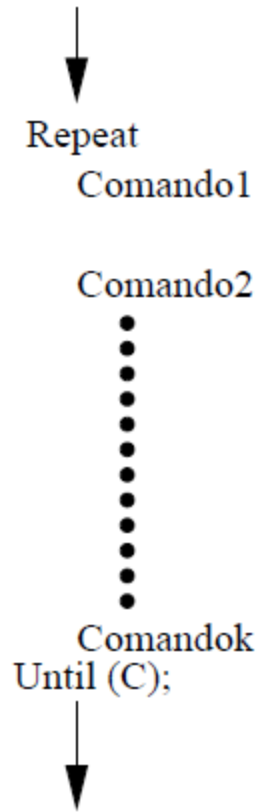
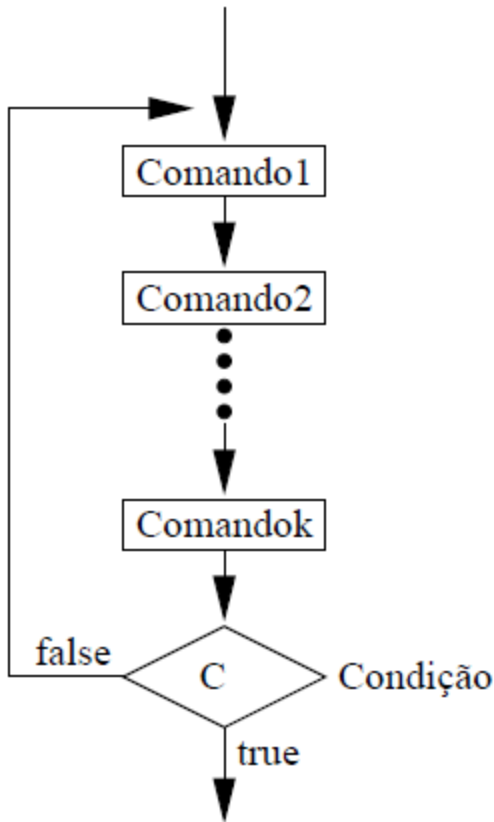
```
for (i = 0; ; i++) {  
    printf("numero %d\n", i);  
    if (i == 5) break;  
}
```



Exercício: Fazer a tradução para o comando REPEAT e o exercício

<comando repetitivo 3> ::= **repeat** <comando> { ; <comando>} **until** <expressão>

::= **do** <comando> { ; <comando>} **while** (<expressão>)



X := 0 ;
REPEAT X := X + 1 ;
UNTIL X = 10 ;
...
x 101

Repeat C {; C} Until E

L1 NADA
... } tradução de C1...Cn

... } tradução de E

DSVF L1 determinado a priori

L2 NADA  não é necessário, mas é bom se parecer com o While

A diferença entre os comandos REPITA em C e Pascal são somente na forma !!!

```
X := 0 ;  
REPEAT X := X + 1 ;  
UNTIL X = 10 ;  
...
```

x 101

L1 NADA

```
CRVL 101  
CRCT 1  
SOMA  
ARMZ 101
```

```
CRVL 101  
CRCT 10  
CMIG
```

```
DSVF L1
```

L2 NADA

Exercício casa: Fazer a tradução para o comando CASE com valores constante únicos e depois a mais geral (Dragão Roxo, inglês, pg. 418)

<comando condicional 2> ::=

case <expressão> **of** <elem_case> { ; <elem_case> } **end**

<elem_case> ::= <constante> : <comando>

Case idade of

V1: write(X);

V2: write(Y);

V3: write(Z);

V4: write(W)

end

1. Traduza E e guarde em t
2. Desvie para o rótulo de **TESTE** (no final da tradução de Cis) que implementa uma tabela que pareia Vi com Ci e DSVS em cada par para o rótulo do Ci
3. Traduza cada Ci (tem um rótulo para cada um) e Gere um DSVS **FIM** após cada um
4. O rótulo **FIM** está no final da tabela

Restrições Semânticas:

1. <expressão> deve ser do tipo ordinal

2. **As constantes devem ser únicas (TABELA no compilador para guardar valor e rótulo)**

3. O tipo das constantes deve ser compatível com o tipo da expressão

4. Procedimentos pré-definidos de E/S

1. LEIT

Usam variáveis simples

2. IMPR

Read (V1, V2, ..., Vn)

Write(E1, E2, ..., En)

Usados sem
alteração

Tradução

LEIT		...} tradução de E1
ARMZ	V1	IMPR
LEIT		...} tradução de E2
ARMZ	V2	IMPR
...		...
LEIT		} tradução de Em
ARMZ	Vn	IMPR

Ex 9:
Read (a, b)
Write (x, x*y)

LEIT
ARMZ A
LEIT
ARMZ B

CRVL X
IMPR
CRVL X
CRVL Y
MULT
IMPR

Vamos agora para a tradução do primeiro programa

- Vejam que vamos usar aqui também **instruções modificadas da MEPA**,
 - assim a tradução deste programa não reflete a versão final que usa registradores de base (o display D).

5. Programas sem procedimentos

1. PARA

(é a última instrução de um programa)

2. INPP

s := -1 alterado

(é a primeira instrução de um programa)

3. AMEN n

(usado para reservar posições de memória para cada linha de declaração de variáveis)

O compilador pode calcular facilmente os endereços das variáveis v_1, v_2, \dots, v_n que deverão ser $0, 1, \dots, n-1$ (quando esta é a primeira declaração)

Usados sem
alteração

K. pg 103 (adaptado para FRANKIE)

```
program exemplo5;
```

```
  int n,k;
```

```
  int f1,f2,f3;
```

```
begin
```

```
  read(n);
```

```
  f1:=0; f2:=1; k:=1;
```

```
  while (k <= n)
```

```
begin
```

```
  f3:= f1 + f2;
```

```
  f1 :=f2; f2 := f3;
```

```
  k := k + 1
```

```
end;
```

```
write(n,f1)
```

```
end.
```

Logo iremos também desalocar as variáveis globais

tradução

INPP

AMEN 2

AMEN 3

LEIT

ARMZ 0

CRCT 0

ARMZ 2

CRCT 1

ARMZ 3

CRCT 1

ARMZ 1

L1 NADA

CRVL 1

CRVL 0

CMEG

DSVF L2

CRVL 2

CRVL 3

SOMA

ARMZ 4

CRVL 3

ARMZ 2

CRVL 4

ARMZ 3

CRVL 1

CRCT 1

SOMA

ARMZ 1

DSVS L1

L2 NADA

CRVL 0

IMPR

CRVL 2

IMPR

PARA

Detalhes de Implementação (rótulos das instruções)

- No lugar de rótulos simbólicos, L_1 e L_2 , podem aparecer endereços reais.
 - L_1 é facilmente calculado, pois é a próxima posição de P ao encontrar um comando que envolve desvio: while, if, etc.
 - Quando L_2 é mencionado pela 1ª vez, não é possível saber seu valor real. O que se faz é guardar o endereço desta instrução e quando L_2 for determinado (ao final do while), deve-se voltar a esta posição e preencher o valor de L_2 .
 - É por isto que é conveniente guardar os código gerados num array, pois precisamos eventualmente "voltar" a uma instrução já gerada.
 - Esta volta para remendar o que já foi gerado é chamada de **técnica de backpatching**, pois preenche informação não especificada de rótulos na geração de código
- Kowaltowski propõe o uso de rótulos da forma L_i , com i inteiro.

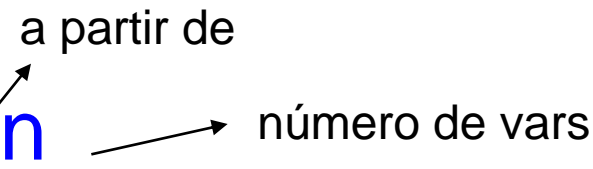
Nosso primeiro programa com procedimentos

- Ainda é uma solução parcial
 - Aqui também não temos uma tradução que usa o display D

6. Procedimentos sem parâmetros

- As Instruções da MEPA usam endereços fixos.
- Como traduzir procedimentos recursivos?
- Num dado instante irão existir várias **encarnações** das variáveis locais dos procedimentos que não podem ocupar a mesma memória!
- **Solução:** usar uma disciplina de pilha para guardar o valor anterior das variáveis locais e assim liberar espaço para a próxima encarnação.
- O valor anterior deve ser restaurado no retorno do procedimento.

6. Procedimentos sem parâmetros

- AMEN m, n 

para $k := 0$ até $n-1$ faça

$s := s + 1$

$M[s] := M[m+k]$

Cada procedimento salva no topo da pilha o conteúdo das variáveis locais

alterados

- DMEN m, n

Para $k := n-1$ até 0 faça

$M[m+k] := M[s]$

$s := s - 1$

Os valores são restaurados no retorno do procedimento

O programa principal também é considerado um procedimento

6. Procedimentos sem parâmetros

Vejam as informações do **Registro de Ativação**

- RTPR

$i := M[s]$

$s := s - 1$

alterados

- CHPR p

$s := s + 1$

$M[s] := i + 1$

$i := p$

- A informação necessária para executar o **retorno de um procedimento** é o endereço da instrução a qual a execução deve retornar.

- O lugar mais natural para guardar esta informação é a **própria pilha**.

- A instrução RTPR sempre encontra a informação no topo da pilha, pois a CHPR guardou ela lá.

Exemplo de programa com procedimento recursivo

```
program ex2;
int x, y;
procedure p;
  int z;
  begin
    z := x; x := x - 1;
    if (z > 1) p
    else y := 1;
    y := y * z
  end;
begin
  read(x);
  p;
  write(x,y)
end.
```

Endereços:

X: 0

Y: 1

Z: 2

Vejam a
página 107 do
Kowaltowski

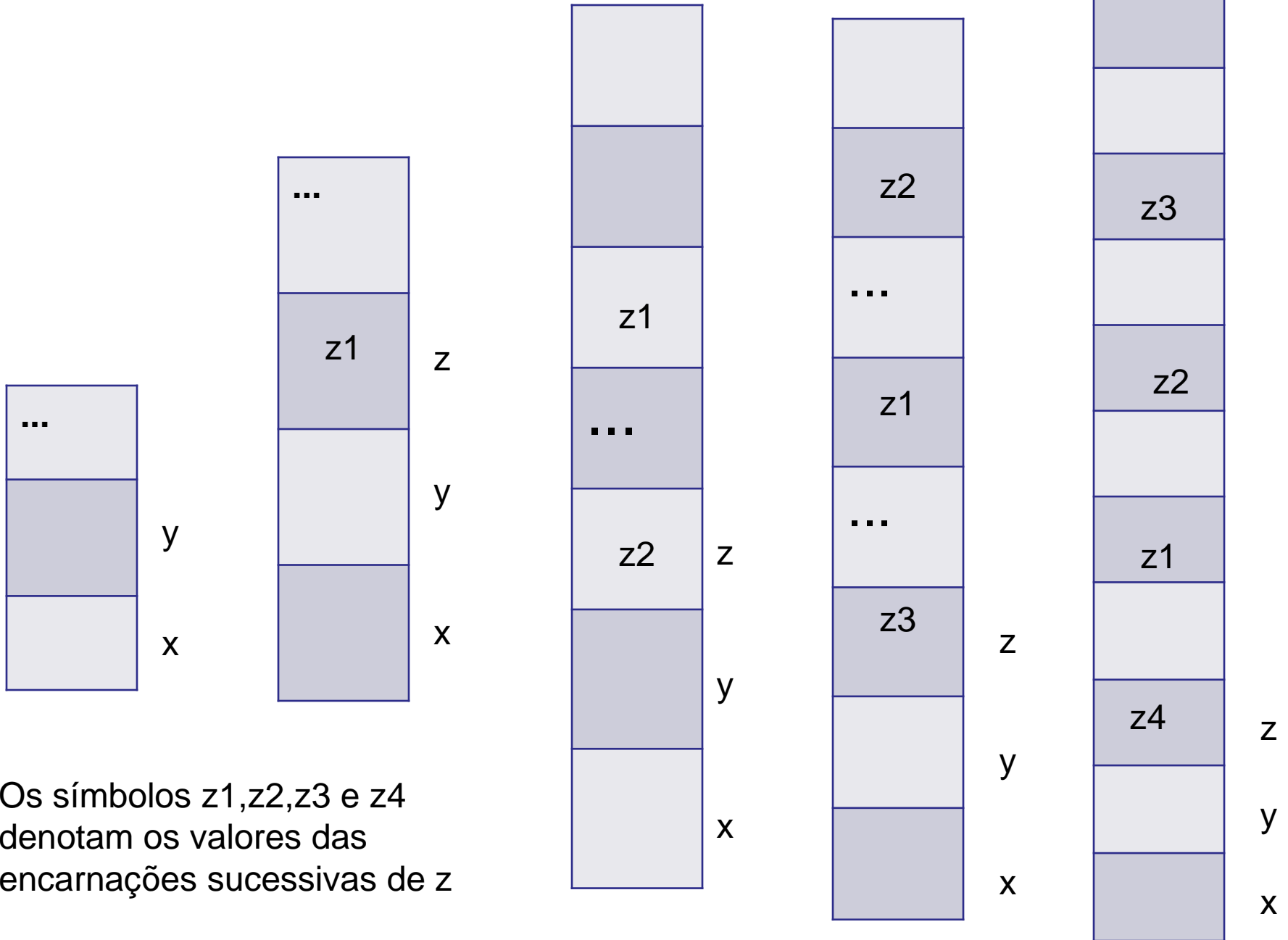
**(Vejam pg 168 do Kowaltowski)
para detalhes de implementação:**

Lembrem que os endereços das variáveis são seqüenciais, pois a varredura do programa é da esquerda para direita.

Assim, com ajuda de uma variável global, o procedimento **geração_código** no compilador que estão desenvolvendo irá atribuir os endereços 0, 1, 2 na área M da MEPA para X, Y, Z.

Tendo estas fixas, o AMEN funciona corretamente.

Configurações da pilha para cada chamada do procedimento p



Os símbolos z_1, z_2, z_3 e z_4 denotam os valores das encarnações sucessivas de z

Tradução do programa

	INPP				
	AMEN 0,2				
	DSVS L1	←			
L2	NADA		L4	NADA	
	AMEN 2,1			CRVL 1	
	CRVL 0			CRVL 2	
	ARMZ 2			MULT	
	CRVL 0			ARMZ 1	
	CRCT 1			DMEN 2,1	
	SUBT			RTPR	
	ARMZ 0		L1	NADA	Programa principal
	CRVL 2			LEIT	
	CRCT 1			ARMZ 0	
	CMMA			CHPR L2	
	DSVF L3			CRVL 0	
	CHPR L2			IMPR	
L3	DSVS L4			CRVL 1	
	NADA			IMPR	
				DMEN 0,2	
				PARA	

Note o uso de DSVS L1 para pular o código do procedimento p ao iniciar-se a execução

Casos de declaração de mais de 1 procedimento, com suas variáveis locais

- 1. Quando não há encaixamento:** não pode haver acesso simultâneo às variáveis locais dos 2. Assim, as mesmas posições fixas de memória podem ser atribuídas às posições de memórias dos 2.
- 2. Quando há encaixamento:** como neste caso pode haver acesso à variável local de um procedimento por outro interno, as variáveis locais não podem ocupar a mesma posição de memória.

Exemplos 7 e 8 (Kowaltowski, pgs.110 e 113)

- Regra Geral para atribuir endereços a variáveis em procedimentos sem parâmetros:
 - Se um procedimento q , que tem k variáveis locais, está diretamente encaixado dentro de um procedimento p cuja última variável local recebeu endereço m , então as variáveis de q recebem os endereços $m+1$, $m+2$, ..., $m+k$.
 - Supor que o programa principal é um procedimento encaixado num outro, tal que $m = -1$, para que as variáveis do pp tenham os endereços $0, 1, \dots, k-1$.
 - Também: A execução do programa mantém a pilha inalterada.

Geração de Código para a MEPA

Veremos agora a geração de código para características mais complexas de FRANKIE:

7. Procedimentos com parâmetros passados por valor
8. Passagem de parâmetros por referência e os Problemas gerados para a implementação anterior (7)

E uma característica bem simples:

9. Implementação de funções

Procedimentos com parâmetros passados por valor (Exemplo)

```
program exemplo9;  
int x,y;  
procedure p (t:int);  
int z;  
begin  
  if (t > 1) p(t-1)  
  else y := 1  
  z := y;  
  y := z * t  
end;  
Begin  
  read(x);  
  p(x); write(x,y)  
End.
```

Se comportam como variáveis locais, cujos valores são inicializados com os valores dos parâmetros reais.

Então também terão endereços fixos de pilha.

Mas, os valores dos parâmetros reais são avaliados na chamada e ficam no topo da pilha, pois são expressões.

Como os valores prévios dos parâmetros passados por valor devem ser restaurados após o fim do procedimento, VAMOS intercambiar os prévios com os efetivos no início do proc e fazer a restauração no fim.

Ótima sacada !

7. Procedimentos com parâmetros passados por valor

- Se comportam como variáveis locais, cujos valores são inicializados com os valores dos parâmetros reais que estão no topo da pilha

a partir de

- **IPVL** m, n → número de parâmetros **Inicializa parâmetros**

para $k := 0$ até $n-1$ faça

$t := M[m+k]$

NOVO

$M[m+k] := M[s-n+k]$

$M[s-n+k] := t$

- **RTPR** m, n **Restaura valor dos parâmetros**

$i := M[s]; s := s - 1;$

para $k := n-1$ até 0 faça

$M[m+k] := M[s]$

alterado

$s := s - 1$

Exemplo de procedimento recursivo com parâmetro passado por valor

```
program exemplo9;  
int x,y;  
procedure p (t:int);  
int z;  
begin  
  if (t > 1) p(t-1)  
  else y := 1  
  z := y;  
  y := z * t  
end;  
Begin  
  read(x); p(x); write(x,y)  
End.
```

Quando a expressão é avaliada ele é deixada no topo da pilha.

Dentro do proc, estes valores iniciam as posições fixas de memória destinadas aos parâmetros

Como os valores prévios destas posições fixas terão que ser **restaurados após o término do proc**, podemos SIMPLEMENTE intercambiar os valores prévios com os efetivos no início da execução e fazer a restauração no fim.

Tradução do Programa

	INPP		
	AMEN 0,2		
L2	DSVS L1		CRVL 2
	NADA		MULT
	IPVL 2,1		ARMZ 1
	AMEN 3,1		DMEN 3,1
	CRVL 2		RTPR 2,1
	CRCT 1		NADA
	CMMA	L1	LEIT
	DSVF L3		ARMZ 0
	CRVL 2		CRVL 0
	CRCT 1		CHPR L2
	SUBT		CRVL 0
	CHPR L2		IMPR
L3	DSVS L4		CRVL 1
	NADA		IMPR
	CRCT 1		DMEN 0,2
	ARMZ 1		PARA
L4	NADA		
	CRVL 1		
	ARMZ 3		
	CRVL 3		

8. Passagem de parâmetros por referência

- Suponha que Frankie tenha somente **passagem de parâmetros passados por referência**
- Para gerenciar var locais de proc sem parâmetros guardávamos as var locais no topo da pilha para liberar espaço (que era fixo) para a nova execução recursiva de um proc, **pois o programa só tinha acesso a instâncias das variáveis locais criadas por último.**
- Mas vejam no próximo programa o que acontece quando temos **proc com variáveis passadas por referência**

```

program exemplo;
int x, y;
procedure p (var s: int);
Int z;
Begin
  if (s = 1) y := 1
  else begin
    z := s - 1;
    p(z);
    y := y * s
  end
End;
Begin
  x := 4;
  p(x);
  write(x,y)
End.

```

Era a z anterior, pois z é
Passado como parâmetro



- Como p é recursivo, podemos ter várias encarnações da var local z.
- O problema é que durante a execução de p os comandos podem fazer **acesso a duas encarnações: a atual e a anterior através do par s**
- **Mas nós havíamos salvo o conteúdo das variáveis locais no topo e só no fim do proc é que poderão ser recuperadas**
- **Não podemos mais usar a implementação anterior (6) !!!!!!!**

Solução

- Vamos atacar a causa do problema: atribuição de ENDEREÇOS FIXOS à **variáveis do programa**.
 - Serão substituídos pelo endereço relativo dentro de cada procedimento (inclui também o programa principal)
 - O compilador associará com cada proc um **REGISTRADOR de BASE (Display D)** e às variáveis locais de cada proc será atribuído um deslocamento fixo, relativo ao conteúdo deste registrador.
 - **O end de cada var será um par**: número do reg de base e o deslocamento, ambos fixos. Este tipo de endereço é chamado **endereço léxico/textual**
 - Os conteúdos dos registradores de base variarão durante a execução do programa, garantindo o acesso correto às variáveis

- A região D da MEPA, será utilizada para formar o vetor de registradores de base.
 - Durante a execução de um proc, o reg de base correspondente apontará para a região da pilha M onde foram alocadas as var locais do proc, formando seu registro de ativação.
- Para uniformizar o esquema de endereçamento, o programa principal possui o registrador de base d_0 .
 - Não será mais necessário guardar e restaurar o valor das variáveis
 - O programa é um procedimento de nível 0
 - Se um procedimento p tem nível m, todos os procs encaixados em m tem nível m+1
 - O nro de registradores corresponde ao nível de encaixamento máximo num programa, o que é reduzido.

Exemplo

(Kowaltowski, pp. 119 – 121)

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

...

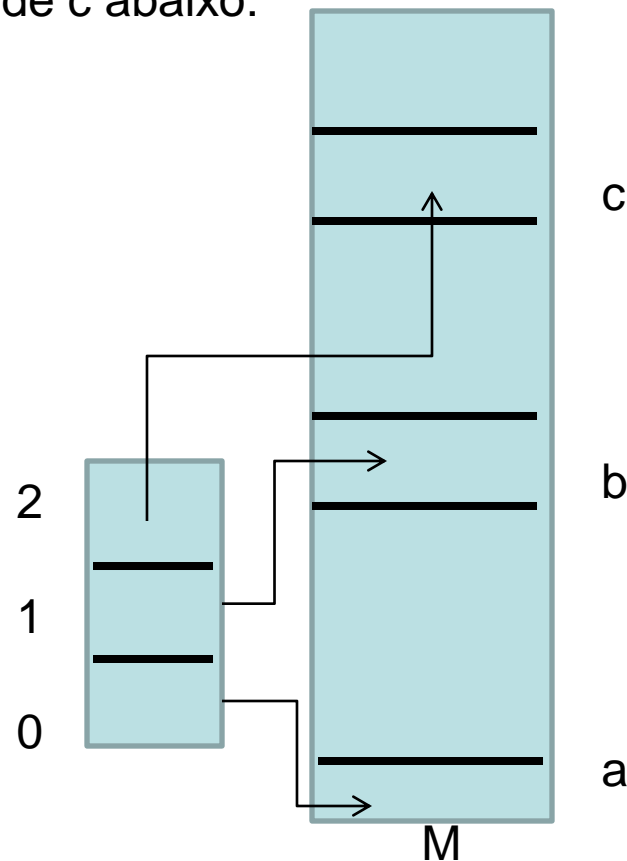
p⁵ ; r⁶ ;

End.

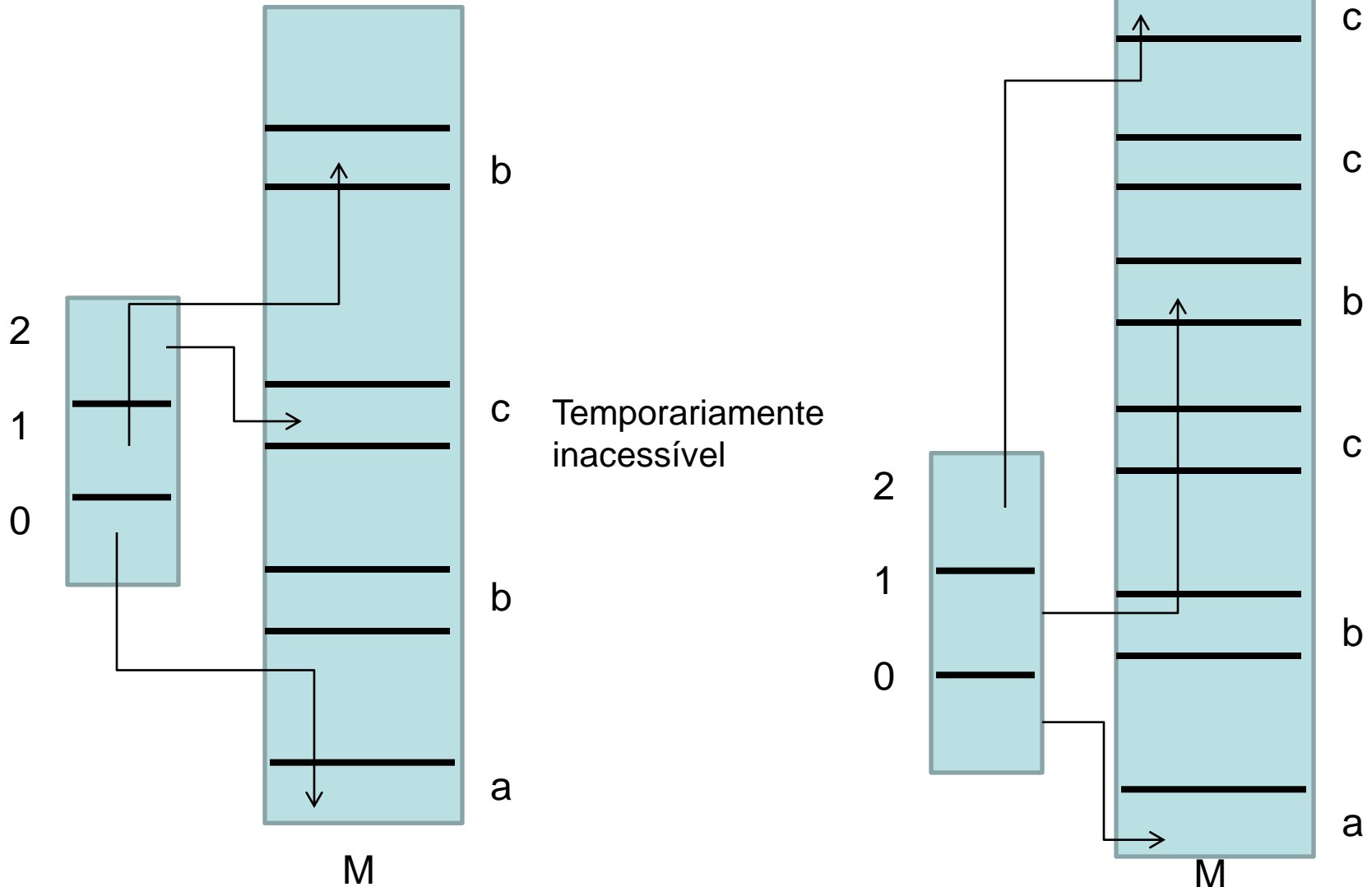
Cada procedimento p, q, r, possui os registradores de base dp, dq, dr, que apontam para as variáveis locais dos procedimentos.

dp e dq são distintos, mas dp e dr ocupam o mesmo.

Após a/p⁵/q³ temos a declaração de c abaixo:



Após $a/p^5/q^3/p^1$ (esq)
Após $a/p^5/q^3/p^1/q^3/q^2$ (dir)



8. Atacando o problema dos endereços fixos

- Como os **reg de base** compartilham o mesmo espaço com os **proc de mesmo nível**, resolveremos isto da mesma forma que fizemos para implementar var locais:
 - Cada **proc** se encarrega de salvar e restaurar o conteúdo **de seu reg de base**, que será guardado na pilha M e isto é realizado pelas instruções: **CHPR p** e **ENTR K**

8 Instruções, com rótulo 8

nível

desloc

- CRVL m,n
- ARMZ m,n
- AMEN n
- DMEN n
- INPP Usados como mostra a folha da MEPA, A
- CHPR p (igual versão 6)
- ENPR k Usado como mostra a folha da MEPA B; primeira instrução de um proc, k é o nível; salva o reg de base
- RTPR k
 $D[k] := M[s]$ Restaura o reg de base
 $i := M[s-1]$
 $s := s - 2$

8. Agora com parâmetros passados por valor e ref

- O lugar natural para guardar parâmetros reais é no topo da pilha.

- No caso de par passados por valor

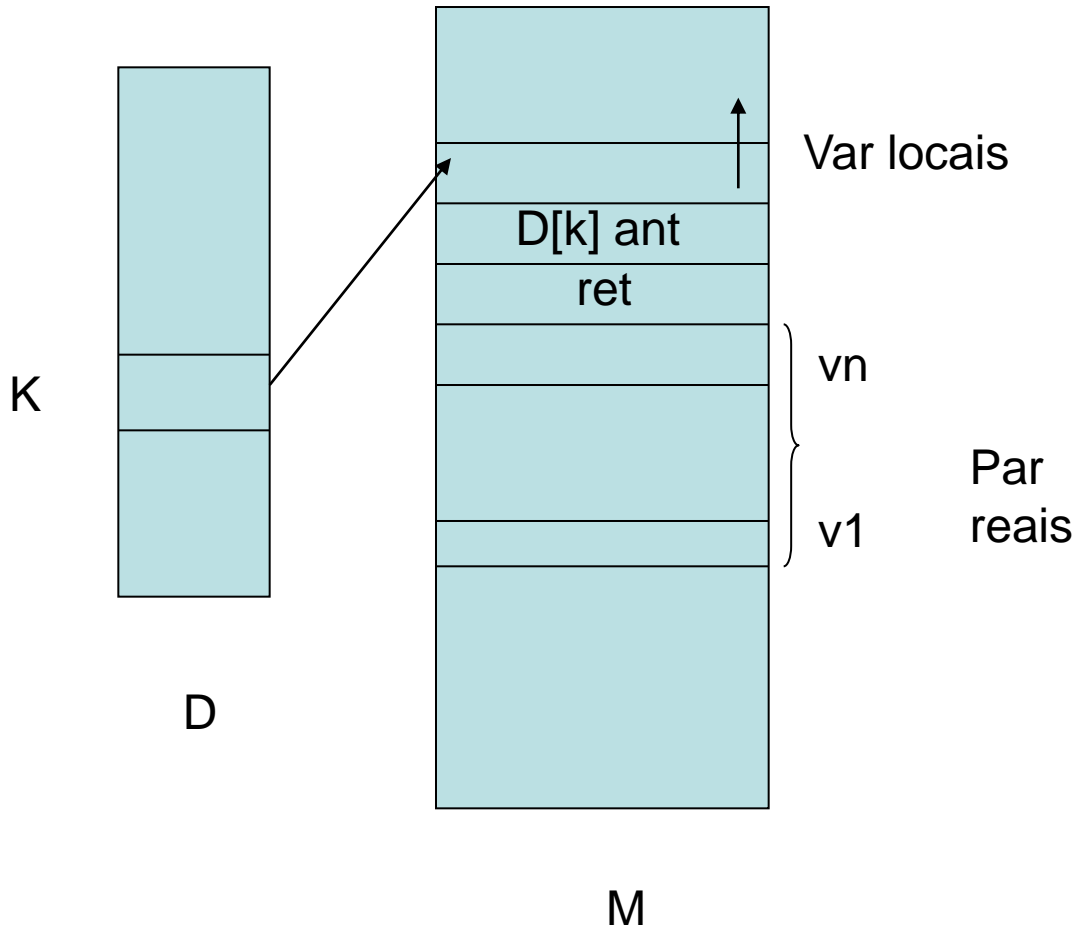
- basta avaliar as expressões

- Para par por passados por referência

- usaremos **endereçamento indireto**, sendo suficiente empilhar o endereço da variável passada como parâmetro real.

Chamada de proc p (E_1, \dots, E_n), com v_1, \dots, v_n sendo os valores dos parâmetros, passados por valor ou ref

Execução das instruções após CHPR e ENPR



O acesso a par formais dentro de p será feito usando-se $D[k]$ e deslocamentos negativos

Se p tem n parâmetros então o i -ésimo terá deslocamento $-(n+3-i)$

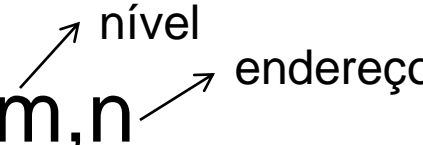
RTPR k, n (MODIFICADA)

$D[k] := M[s]$

$i := M[s-1]$

$s := s - (n+2)$ remove os n par

8. Manipulação de parâmetros passados por ref

- CREN m,n 
- CRVI m,n
- ARMI m,n

Usados como mostra a folha da MEPA, A

CREN é usada para empilhar o valor do par real passado por ref.

CRVI e ARMI serão usadas no corpo do proc para fazer acesso aos parâmetros formais passados por ref

Par passados por valor são tratados como var locais.

Exemplo (Kowaltowski pg 125)

Program ex10;

int k;

Procedure p(n: int; var g: int);

Int h;

Begin

If (n < 2) g := g + n

else begin

h:=g;

p(n-1, h);

g :=h;

p(n-2,g); OBSERVEM a tradução

end; O valor de g já é um end.

Write(n,g)

End;

Begin

k:= 0;

p(3,k)

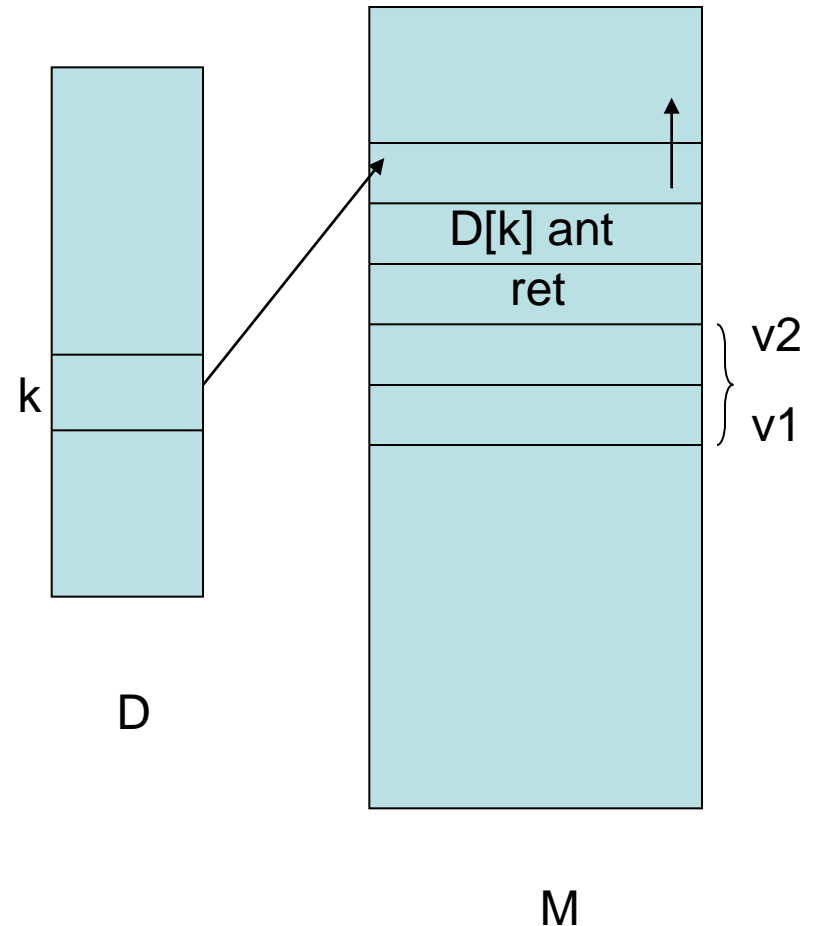
end.

K: 0,0

N:1,-4

G:1,-3

H:1,0



**Se p tem n parâmetros
então o i-ésimo terá
deslocamento - (n+3-i)**

Tradução

K: 0,0
N:1,-4
G:1,-3
H:1,0

INPP

AMEN 1

DSVS L1

L2 ENPR 1

AMEN 1

CRVL 1,-4

CRCT 2

CMME

DSVF L3

CRVI 1,-3

CRVL 1,-4

SOMA

ARMI 1,-3

DSVS L4

L3 NADA

CRVI 1,-3

ARMZ 1,0

CRVL 1,-4

CRCT 1

SUBT

CREN 1,0

CHPR L2

CRVL 1,0

ARMI 1,-3

CRVL 1,-4

CRCT 2

SUBT

CRVL 1,-3

CHPR L2

L4 NADA

CRVL 1,-4

IMPR

CRVI 1,-3

IMPR

DMEN 1

RTPR 1,2 {remove os 2}

L1 NADA {p.p.}

CRCT 0

ARMZ 0,0

CRCT 3

CREN 0,0

CHPR L2

DMEN 1

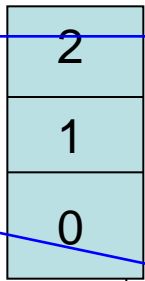
PARA

9. Tratamento de Funções

- A implementação de funções é semelhante a de procedimentos
 - A diferença é que o nome da função corresponde a uma **variável local adicional**
 - O valor desta variável **deverá estar** no topo da pilha após o retorno da função.
- Para fazer isto sem modificar a MEPA:
 - Reservar uma posição no topo **ANTES** de avaliar os **parâmetros reais** da função, usando **AMEM 1**
 - Esta posição será usada como uma variável local, no corpo da função; seu endereço é NEGATIVO e vale **-(n+3)**, sendo n o número de parâmetros da função
 - O valor devolvido pela função fica no topo da pilha após o RTPR, e pode ser usado numa impressão ou atribuição

Temporários
Dados Locais
Estado da máquina
Link de acesso
Link de controle
Parâmetros reais
Valor retornado - função

Avaliação de expressões



Variáveis locais

Restaura ambiente anterior

...
d
c
D [1] anterior
Endereço Retorno
Par Y
Par X
Valor retornado - função
Variáveis Globais

Registro de ativação (RA) geral

```

...
Int f (int a,b);
int c, d;
Begin .... Return ... end;
Begin
...write(f(X,Y));...
End.

```

M
Na Mepa...

Exemplo (Kowaltowski pg 126 e 127) do prog. pag 88 (adaptado para FRANKIE)

Programa ex5;

int m;

int f(n:int; var k: int);

{

int p,q;

if (n <2)

begin

k := 0; return n

end

else begin

k := p + q + 1;

return f(n-1,p) + f(n-2,q)

end;

};

begin

write(f(3,m),m)

end.

m: 0,0

f: 1,-5

n:1,-4

k:1,-3

p: 1,0

q: 1,1

O Comando **Return** faz 2 ações:

- 1) Armazena o valor da EXP no endereço do ID de função e
- 2) DSVS para o fim da função

m: 0,0

f: 1,-5

n:1,-4

k:1,-3

p: 1,0

q: 1,1

Tradução

INPP
 AMEM 1
 DSVS L1
 L2 ENPR 1
 AMEM 2
 CRVL 1,-4
 CRCT 2
 CMME
 DSVF L3
 CRCT 0
 ARMI 1,-3
 CRVL 1,-4
 ARMZ 1,-5
 DSVS L4 return
 DSVS L5
 L3 NADA

CRVL 1,0
 CRVL 1,1
 SOMA
 CRCT 1
 SOMA
 ARMI 1,-3
 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
 DSVS L6 return....
 L4 NADA
 L5 NADA
 L6 NADA
 DMEM 2
 RTPR 1,2
 L1
 NADA
 AMEM 1
 CRCT 3
 CREN 0,0
 CHPR L2
 IMPR
 DMEM 1
 PARA

Exercício casa: implementação de vetores (capítulo 9.5)

- Linearização dos elementos; reservar o bloco, pois o tamanho é conhecido em tempo de compilação
- Instrução AMEM pode ser usada
- Analisem a necessidade de novas instruções para os casos:
 - Elementos usados nas expressões
 - Elementos como variáveis
 - Vetor na passagem de parâmetro por valor e por referência

Não trataremos (capítulo 9)

- Rótulos e Comandos de DESVIO
- Passagem de procedimento e função como parâmetro
- Passagem por nome (corresponde a uma função sem parâmetro)
- Blocos com declarações locais