10. Carga e armazenamento de variáveis indexadas

- **Vetores (array's) unidimensionais**
  - **Carga - exemplo : AUX := X [I]**

```
CRVL   l, I      ∴  T:= T + 1; P[T]:= I;
CVET1  l, X      ∴  P[T]:= P[end(X) + P[T] – 1];
ARMZ   l, AUX    ∴  P[end(AUX)]:= P[T];
                    T:= T –1;
```

  - **Armazenamento – exemplo: X [I] := AUX**

```
CRVL   l, I      ∴  T:= T + 1; P[T]:= I
CRVL   l, AUX    ∴  T:= T + 1; P[T]:= AUX
AVET1  l, X      ∴  P[end(X) + P[T – 1] -1 ]:= P[T];
                    T:= T – 2;
```

  - **Exemplo: trocar os valores de X[I] e X[I+1]**

- **Vetores (array's) bidimensionais**
  - **Carga - exemplo : AUX := M [I, J]**

```
CRCT   -, Ncol   ∴  T:= T + 1; P[T]:= Ncol;
CRVL   l, I      ∴  T:= T + 1; P[T]:= I;
CRVL   l, J      ∴  T:= T + 1; P[T]:= J;
CVET2  l, M      ∴  P[T-2]:= P[end(M) +
                    (P[T-1]–1) * P[T-2] + P[T] -1 ];
                    T:= T – 2;
ARMZ   l, AUX    ∴  P[end(AUX)]:= P[T];
                    T:= T –1;
```

- **Armazenamento – exemplo: M [I, J]:= AUX**

CRCT	-, Ncol	∴	T:= T + 1; P[T]:= Ncol;
CRVL	l, I	∴	T:= T + 1; P[T]:= I
CRVL	l, J	∴	T:= T + 1; P[T]:= J;
CRVL	l, AUX	∴	T:= T + 1; P[T]:= AUX
AVET2	l, M	∴	P[end(M) + (P[T-2] – 1) * P[T-3] + P[T-1] -1]:= P[T]; T:= T – 4;

- **Exemplo: Gerar CI para:**

  1 – Somar os elementos de uma matriz

  2 – Zerar os elementos da diagonal principal

11. Geração de código para procedimentos/funções

Declaração de Procedimento

```
<dcl-proc> ::= proc   id #DP1 <par-formais>  ';'  <bloco>  #DP2
                |  função id #DP1  <par-formais>
                                : <tipo-pre-definido>  ';'  <bloco>  #DP2 ;
```

Chamada de Procedimento

```
<comando> ::= id   <rcomid>
<rcomid> ::=  ...
             |   '('   <expressao>   <rep-lexpr> ')' #CP1

             |    î #CP1 ;
```

AÇÕES DE GC

#DP1 - Gera DSVS __, ?
Guarda end. de DSVS em PP (pilha de proc.)

#DP2 - Gera DMEM __, N
Gera RETU l+1, __
Completa DSVS do topo de PP com "PC"

#CP1 - Gera CALL l+1, a

Geração de código para procedimentos / interpretação de CALL e RETU

- **Informações auxiliares**
  - **Endereço de retorno (ER)**
    - **Definido na chamada**
    - **Usado no retorno**
    - **Não é fixo → Guardado na pilha**
  - **Endereço base do procedimento**
    (posição na pilha de execução onde esta a 1º variável local do procedimento)
    - **Usado na carga/armazenamento de var. locais**
    - **Definido na chamada**
      - **Guardado em uma pilha auxiliar PRA (pilha de registradores (base) de ativação)**
      - **Salvo na pilha (para uso em chamadas paralelas e recursivas)**

- **Interpretação**
  - **CALL l, a → endereço da 1º instrução**
    → nível (léxico) do proc. Chamado

```
TOPO:= TOPO + 1
PILHA[TOPO]:= PC + 1 (* endereço de retorno *)
PC:= a               (* endereço da primeira instrução *)
TOPO:= TOPO + 1
PILHA[TOPO]:=PRA[l]
PRA[l]:= TOPO +1             (* End. Base da proc. *)
```

  - **DMEM - , N**

```
TOPO:= TOPO – N
```

  - **RETU l , - (* ou, RETU l, NPar *)**

```
PRA[l]:= PILHA[TOPO]
PC:= PILHA[TOPO – 1]
TOPO:= TOPO – 2
```

EXEMPLO 1 (* procedure sem parâmetros *)

```
progrma proc1;
var x, y: inteiro;
proc Z;
      var a, b: inteiro;
      {
            a:= x;
            leia(b);
            y:= b;
      };
{
      leia (x);
      Z;
      escreva (x, y);
}.
```

```
1 – INICIO    __, __
2 – AMEM      __,  2
3 – DSVS      __, (?) 13
4 – AMEM      __,  2
5 – CRVL       0,  0
6 – ARMZ       1,  0
7 – LEIA      __, __
8 – ARMZ       1,  1
9 – CRVL       1,  1
10 – ARMZ      0,  1
11 – DMEM     __,  2
12 – RETU      1, __
13 – LEIA     __, __
14 – ARMZ      0,  0
15 – CALL      1,  4
16 – CRVL      0,  0
17 – IMPR     __, __
18 – CRVL      0,  1
19 – IMPR     __, __
20 – FIM      __, __
```

EXEMPLO 2

```
programa proc2;
var x, y : inteiro;
proc A;
      var b : inteiro;
      proc C;
            var d : inteiro;
            {     /* corpo de "C" */
                d:= b + x;
                escreva (d);
            };
      {     /* corpo de  "A" */
          leia (b);
          C;
      };
{     /* corpo do programa principal */
    leia (x);
    A;
}.
```

EXEMPLO 3

```
programa proc3;
var a: inteiro;
proc P;
        var b: inteiro;
        {
                leia (b);
                a:= b;
        };
proc Q;
        var c: inteiro;
        {
                leia (c);
                P;
                a:= a + c;
        };
{
        Q;
        escreva (a);
}.
```

- **Geração de código p/ funções**
  - **Mesmo tratamento que procedimentos**
  - **Nome da função tratado como variável alocada no momento da ativação**
    - **Alocada em PRA[L] – 3**
      - **ARMZ l, id-função ≡ ARMZ l, - 3**
    - **Exemplo: Exercício 4**
      - **Gerar CI**
      - **Interpretar CI**

EXEMPLO 4

```
programa func1;
var A, B, R: inteiro;
funcao SOMA : inteiro;
      {
            leia (A, B);
            soma:= A + B;
      };
{
      R:= SOMA;
      escreva (R);
}.
```

- **Procedimentos com parâmetros**
  - **por valor**
    - **Par. atuais são avaliados e alocados na pilha**
    - **Usados como variáveis locais, porém com deslocamento negativo (ficam abaixo do reg. ativação)**
    - **Fórmula de acesso:**
      PRA[L] – (N + 2 – i),
      onde i é o deslocamento do parâmetro.
    - **Retorno: RETU l, N**
      onde N é o número de parâmetros.

  - **por referência**
    - **Idem a parâmetro por valor com cópia (no retorno) para os parâmetros atuais**
    - **Uso de endereços**
      - **Novas instruções:**

```
CREN      l, a   (* carrega endereço *)
CRVLIND l, a   (* carrega valor indireto *)
ARMZIND l, a   (*armazena valor indireto*)
```

- **funções c/ parâmetros**
  - **Nome da função funciona como um parâmetro adicional**

a = - (N + 3)

nº de parâmetros da função

EXEMPLO 5

```
programa par_valor;
var a, b: inteiro;
proc p (val x, y: inteiro);
        var z: inteiro;
        {
                z:= x + y;
                escreva (z);
        };
{
        leia (a, b);
        p (a, b + 9);
}.
```

EXEMPLO 6

```
programa par-ref;
var z: inteiro;
proc p (ref x: inteiro);
        var a, b: inteiro;
        {
                leia (a, b);
                x:= x + a + b;
        };
{
        z:= 10;
        p(z);
        escreva (z);
}.
```