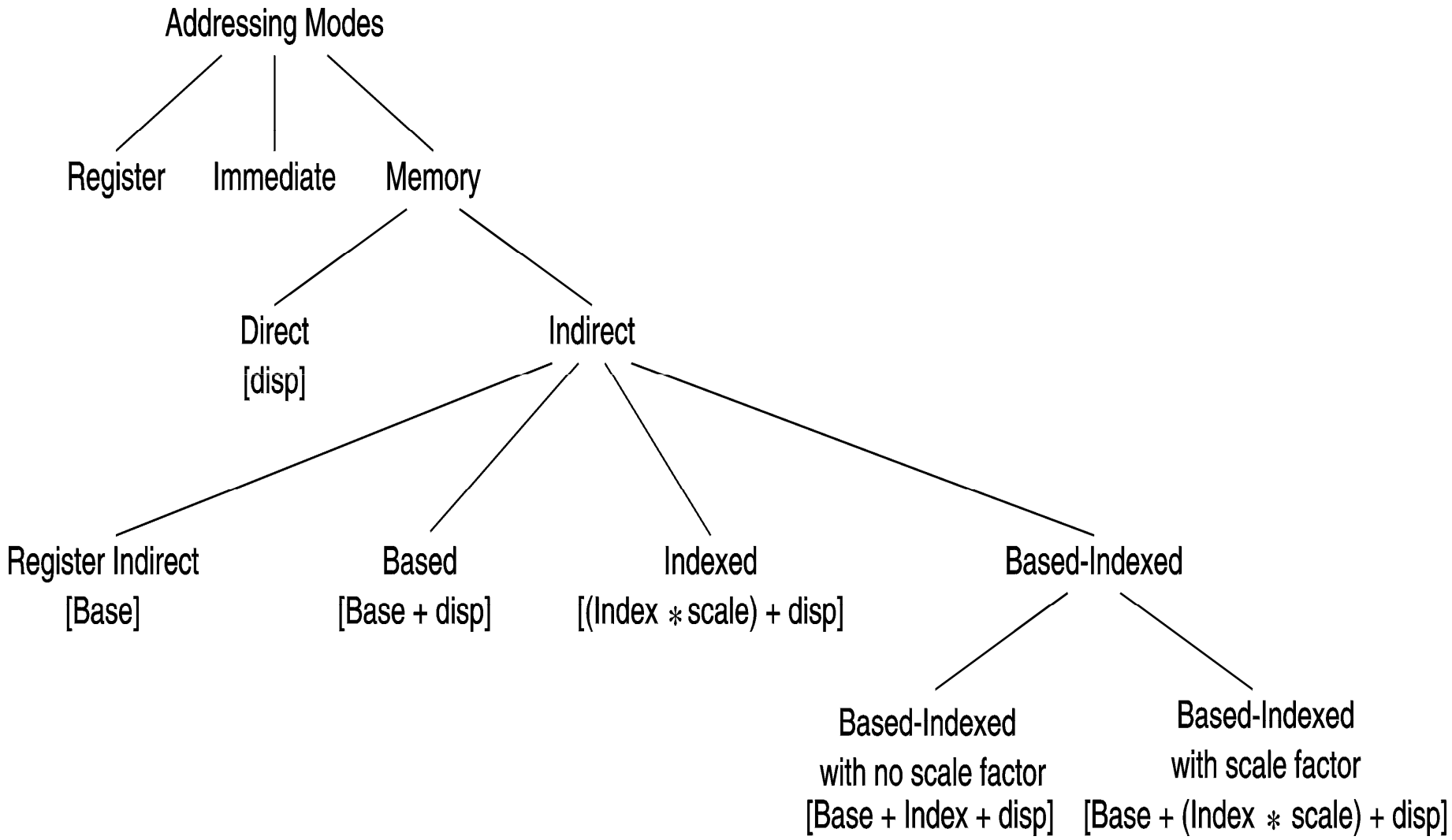
Assembly NASM/IA-32
Modos de endereçamento

Introdução

- Modos de endereçamento no IA-32
- Endereçamento indirecto à memória no IA-32
  - ✓ Endereçamento baseado
  - ✓ Endereçamento indexado
  - ✓ Endereçamento baseado+indexado
- Endereçamento indirecto à memória no NASM/IA-32
- Manipulação de vectores

Modos de Endereçamento no IA-32



Endereçamento Indirecto no IA-32

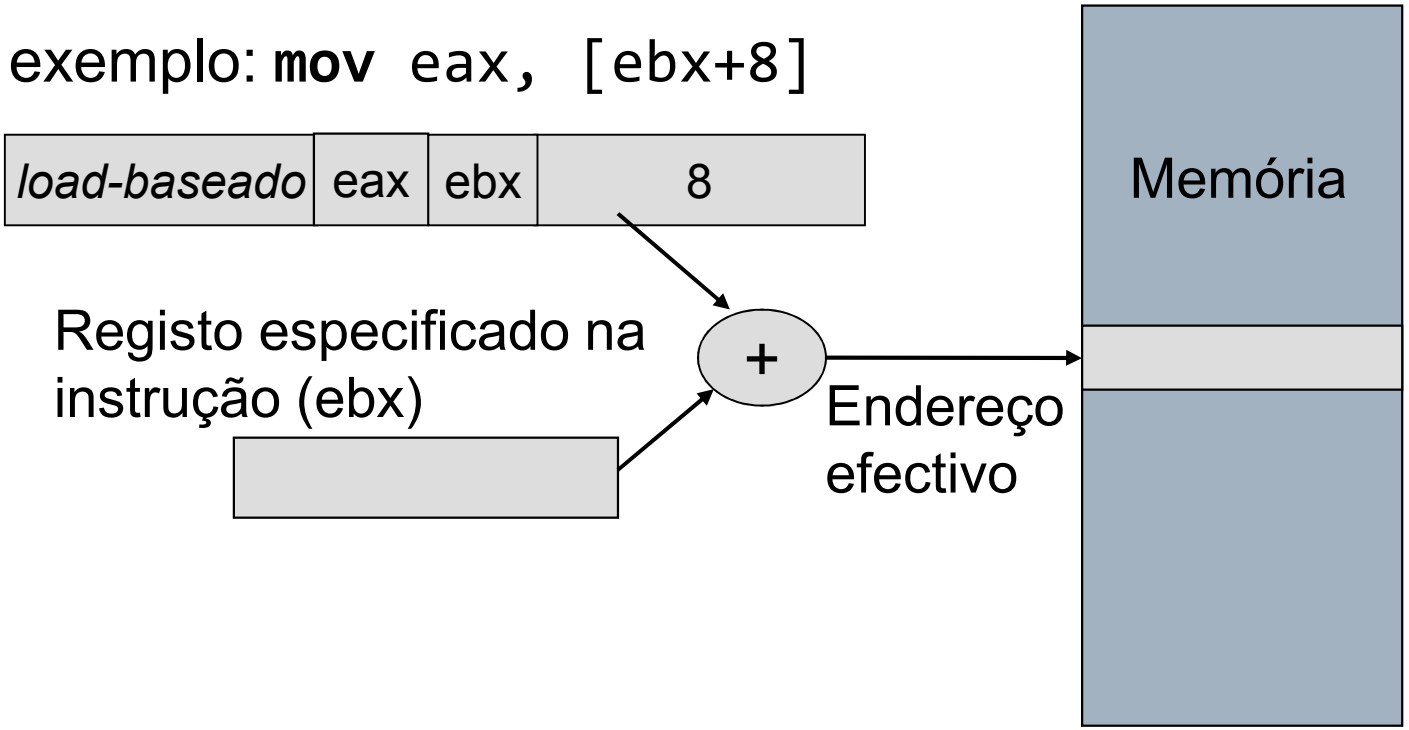
- Forma geral para endereçamento indirecto no IA-32:
  - [Rb+Ri*S+D]
    - D: Constante "deslocamento"
    - Rb: Registo Base: Qualquer dos 8 registos inteiros
    - Ri: Registo índice: Qualquer, excepto ESP
      - Também é pouco provável que seja EBP
    - S: Escala: 1, 2, 4 ou 8
- Casos especiais:
  - [D]
  - [Rb]
  - [Rb+D]
  - [Rb+Ri]
  - [Rb+Ri+D]
  - [Rb+Ri*S]

Endereçamento a 16 ou 32 bits

- Vamos estudar o endereçamento a 32 bits (IA-32)
  - ✓ O endereçamento por omissão no Pentium
- No entanto o Pentium também suporta o endereçamento a 16 bits, originalmente suportado pelo 8086
- A configuração é feita através de um bit (D) no descritor de segmento
  - ✓ D = 0 → ISA a 16 bits – endereços e operandos a 16 bits
  - ✓ D = 1 → ISA a 32 bits – endereços e operandos a 32 bits
- Pode misturar-se operandos de 16 e 32 bits através da aplicação de prefixos especiais ao opcode da instrução

Endereçamento Baseado

- O endereço efectivo é obtido somando o conteúdo de um registo (registo base ou índice) a uma constante



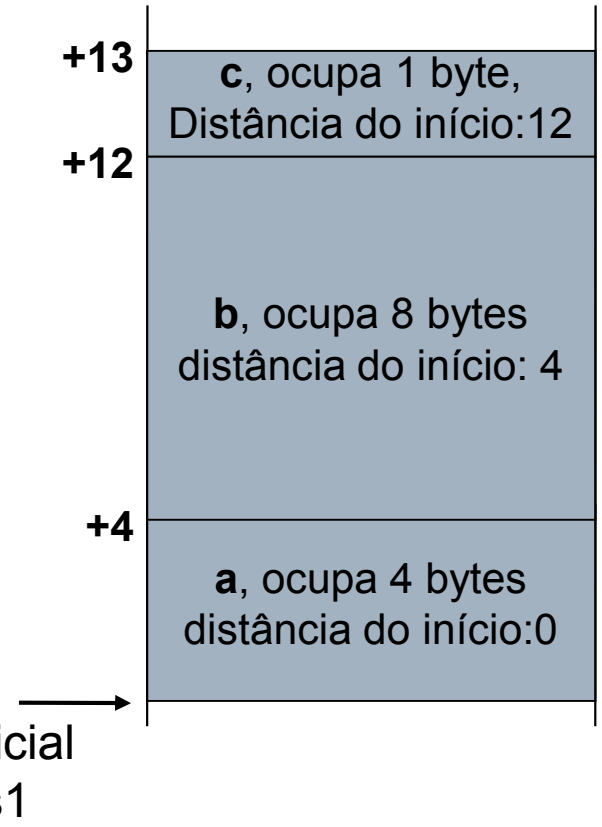
Endereçamento Baseado

- Conveniente para endereçar estruturas

```
struct my_struct {  // estrutura em C
   int a;     // ocupa 4 bytes
   double b; // ocupa 8 bytes
   char c;    // ocupa 1 byte
} s1;
```

Se registo base com endereço de **s1**:

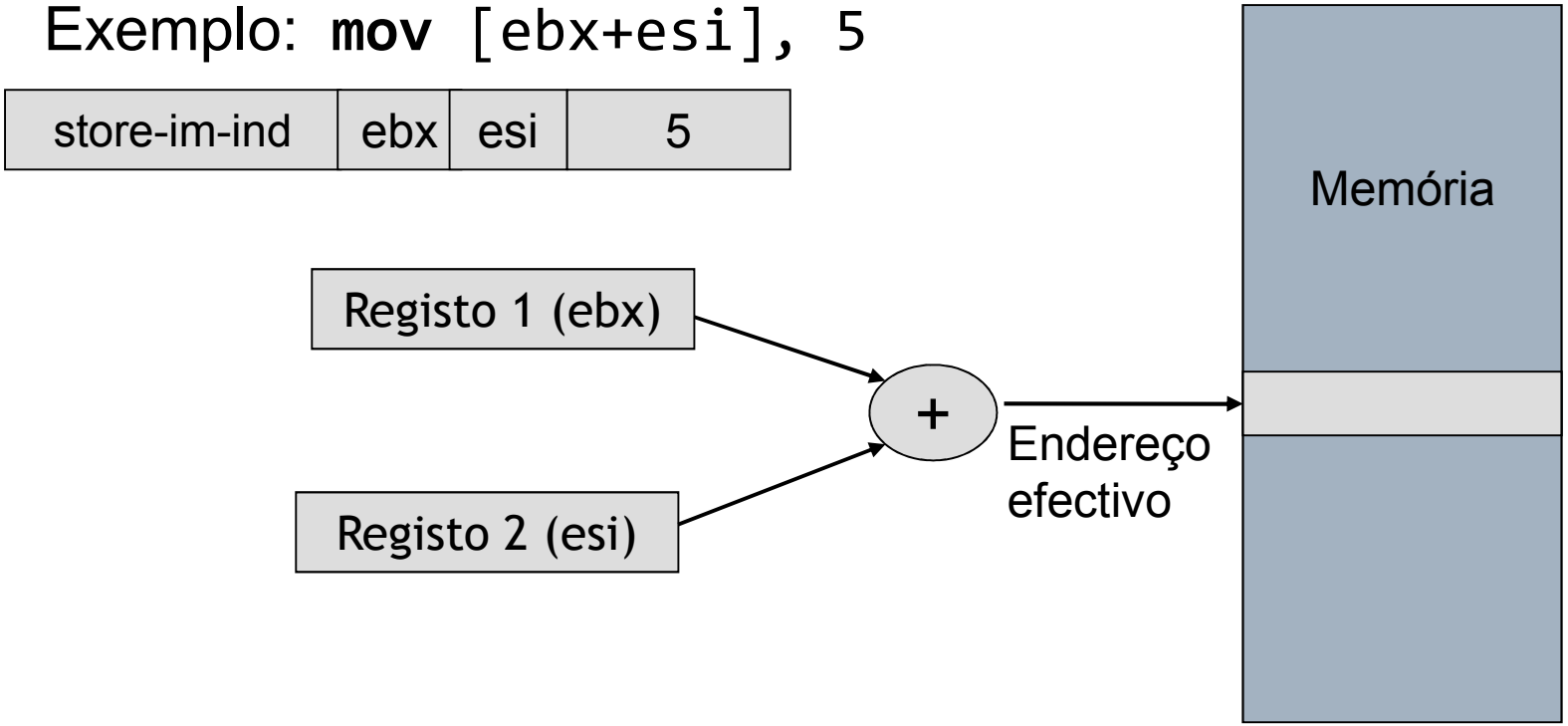
[RegBase + 0] : campo a da estrutura
[RegBase + 4] : campo b da estrutura
[RegBase + 12]: campo c da estrutura



Endereçamento Indexado

- O endereço efectivo é obtido somando o conteúdo de dois registos (registos base e índice).

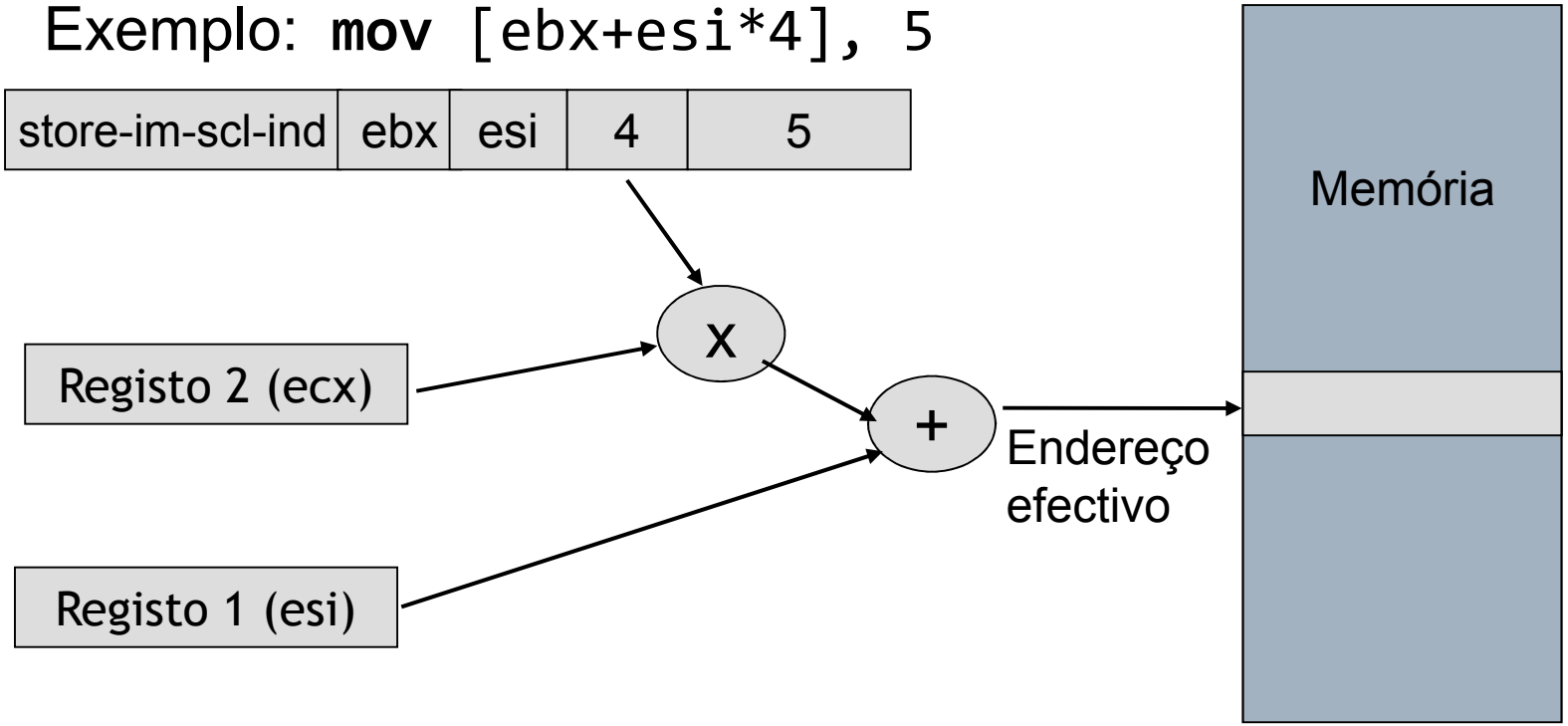
Exemplo: **`mov`** `[ebx+esi], 5`



Endereçamento Indexado com Escala

- O endereço efectivo é obtido somando o conteúdo de dois registos (registos base e índice).

Exemplo: **`mov`** `[ebx+esi*4], 5`



Endereçamento Indexado - Exemplo

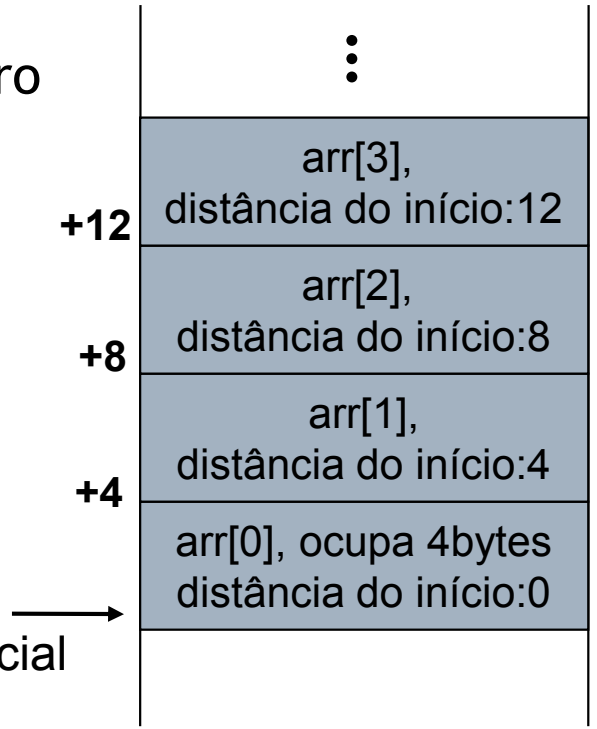
- Conveniente para endereçar vectores

```
int arr[100];
```

Sabendo o endereço de **arr**:
Um registo pode funcionar como base e outro com índice [regBase + *regInd*] onde *regInd* toma os valores 0, 4, 8 …, 396

Exemplo: percorrer o vector

```
        mov ebx, arr
        mov esi, 0
ciclo:  mov eax, [ebx+esi]
        add esi, 4
        cmp esi, 400
        jne ciclo
```



End. Indexado com Escala - Exemplo

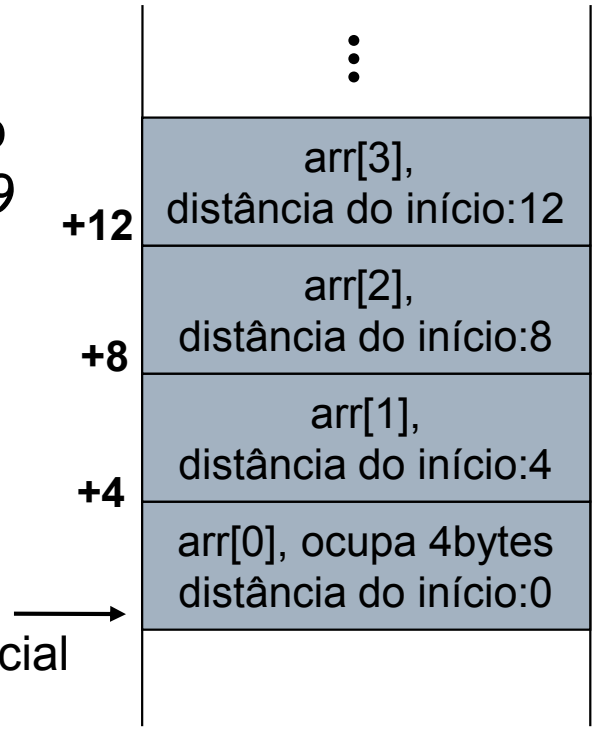
- Conveniente para endereçar vectores

```
int arr[100];
```

Podemos ainda usar o factor de escala para tornar o restante código independente do tamanho dos elementos do vector. O registo de índice toma agora os valores 0,1,2, …, 99

Exemplo: percorrer o vector

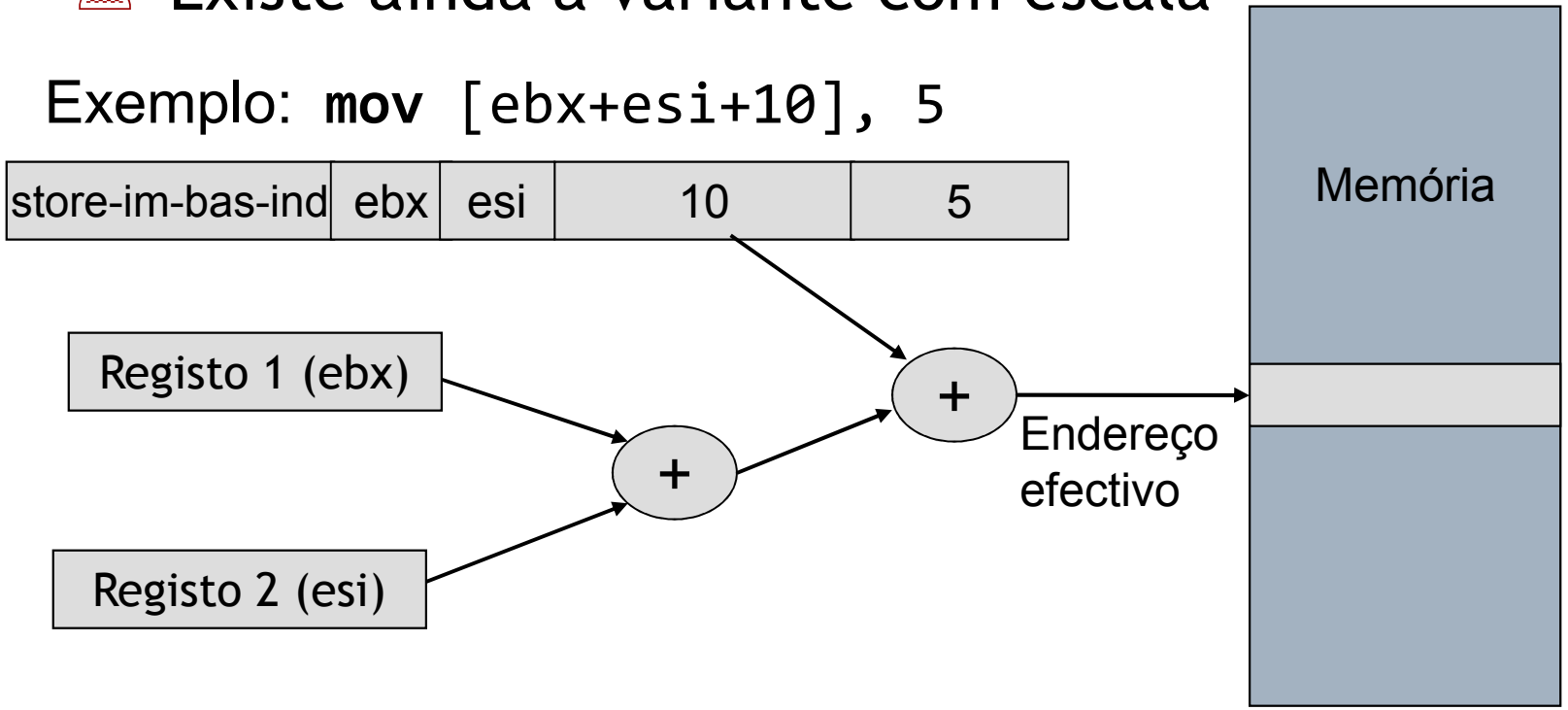
```
        mov ebx, arr
        mov esi, 0
ciclo:  mov eax, [ebx+esi*4]
        inc esi
        cmp esi, 100
        jne ciclo
```



Endereçamento Baseado+Indexado

- O endereço efectivo é obtido somando o conteúdo de dois registos (registos base e índice) mais uma constante (deslocamento)
- Existe ainda a variante com escala

Exemplo: **`mov`** `[ebx+esi+10], 5`



Endereçamento Baseado+Indexado

- Conveniente para endereçar um vector de estruturas

```
struct my_struct {
... // struct com 8 bytes
} arr[100];
```

Suponhamos um registo base (reg1) com o endereço inicial do vector arr

Para aceder ao *elemento i* carrega-se outro registo (reg2) com *i*

[reg1 + reg2*8 + 0] : endereço de arr[i].a
[reg1 + reg2*8 + 4] : endereço de arr[i].b
[reg1 + reg2*8 + 12]: endereço de arr[i].c

Exemplos de Cálculo do Endereço Efectivo

edx	0xf000
ecx	0x100

End. assembly	Cálculo	Endereço
[edx+0x8]	0xf000 + 0x8	0xf008
[edx+ecx]	0xf000 + 0x100	0xf100
[edx+ecx*4]	0xf000 + 4*0x100	0xf400
[edx*2+0x80]	2*0xf000 + 0x80	0x1e080

Modos de Endereçamento no NASM

- O NASM permite uma sintaxe mais abrangente e portanto podemos usar posições de memória em vez de registos e realizar operações algébricas que não as definidas no IA-32

- Exemplos:

```
mov eax, [vector+ecx]
    eax ← Mem[vector+ecx]
mov eax, [vector+(ecx-1)*2]
    eax ← Mem[vector+(ecx-1)*2]
```

- O NASM converte estas notações para as disponíveis no IA-32

A Instrução LEA

- Podemos usar a instrução **`lea`** para calcular um endereço efectivo, por exemplo o endereço do último elemento do vector seguinte

```
vector:    dd 1, 2, 3, 4, 5, 6, 7
```

- Consideremos que:
  - ✓EBX contém o endereço de vector
  - ✓ECX contém o numero de elementos do vector

```
dec ecx
lea edx, [ebx+ecx*4]
```

- Podemos ainda usar a notação mais flexível do NASM

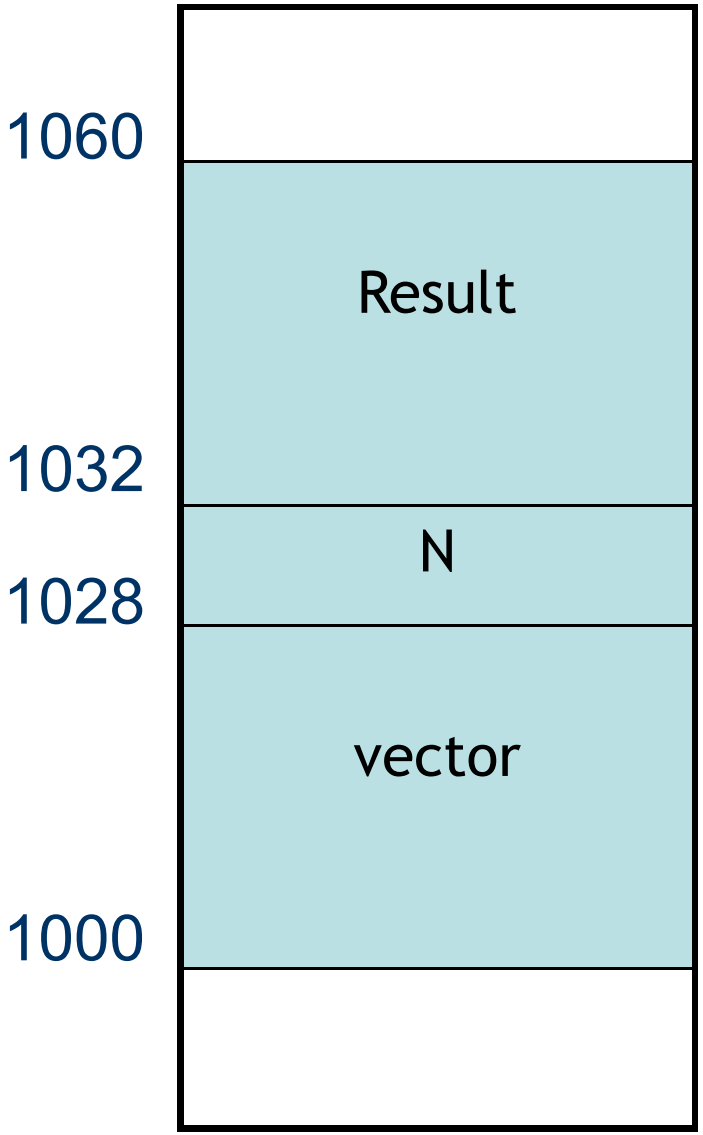
```
lea edx, [vector+(ecx-1)*4]
```

Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector
        mov ecx, [N]
        lea edx, [Result+(ecx-1)*SIZE]
mloop:  mov eax, [ebx]
        mov [edx], eax
        add ebx, SIZE
        sub edx, SIZE
     dec ecx
        jnz mloop
```

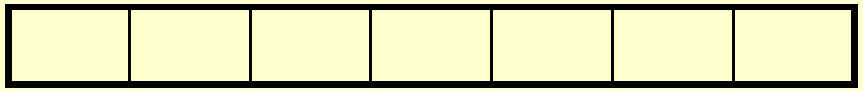
Cópia Invertida de um Vector

- Suponhamos que *vector* começa no endereço 1000
  - ✓ Como *vector* ocupa 7*4 Bytes = 28 Bytes, termina no endereço 1027
- Logo *N* começa no endereço 1028 e vai até 1031 (4 Bytes)
- *Result* começa em 1032 e ocupa os mesmos 28 Bytes que *vector*



Cópia Invertida de um Vector

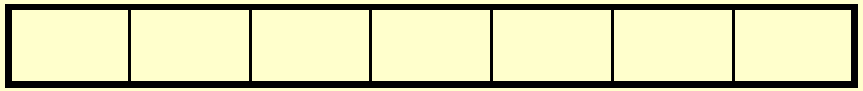
```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector
        mov ecx, [N]
        lea edx, [Result+(ecx-1)*SIZE]
mloop:  mov eax, [ebx]
        mov [edx], eax
        add ebx, SIZE
        sub edx, SIZE
        dec ecx
        jnz mloop
```



Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector
        mov ecx, [N]
        lea edx, [Result+(ecx-1)*SIZE]
mloop:  mov eax, [ebx]
        mov [edx], eax
        add ebx, SIZE
        sub edx, SIZE
        dec ecx
        jnz mloop
```

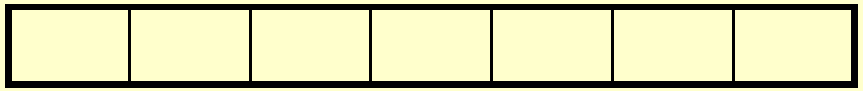
1000



ebx ← 1000

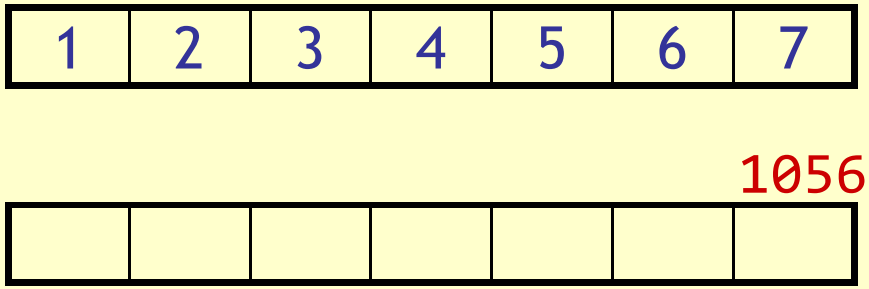
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector        ebx ← 1000
        mov ecx, [N]           ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]
mloop:  mov eax, [ebx]
        mov [edx], eax
        add ebx, SIZE
        sub edx, SIZE
        dec ecx
        jnz mloop
```



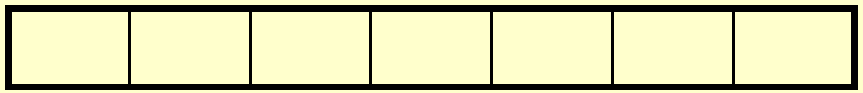
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector       ebx ← 1000
        mov ecx, [N]          ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE] edx ← 1032+6*4
mloop:  mov eax, [ebx]                     = 1056
        mov [edx], eax
        add ebx, SIZE
        sub edx, SIZE
        dec ecx
        jnz mloop
```



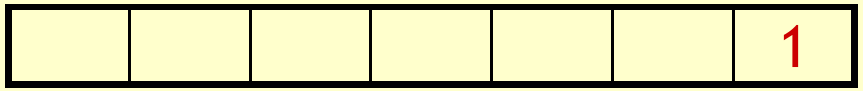
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 1
        mov [edx], eax
        add ebx, SIZE
        sub edx, SIZE
        dec ecx
        jnz mloop
```



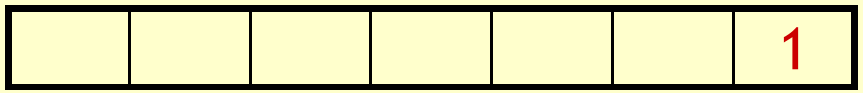
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 1
        mov [edx], eax
        add ebx, SIZE
        sub edx, SIZE
        dec ecx
        jnz mloop
```



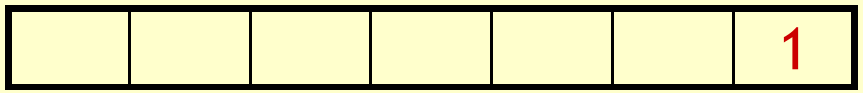
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 1
        mov [edx], eax
        add ebx, SIZE            ebx ← 1004
        sub edx, SIZE            edx ← 1052
        dec ecx                  ecx ← 6
        jnz mloop
```



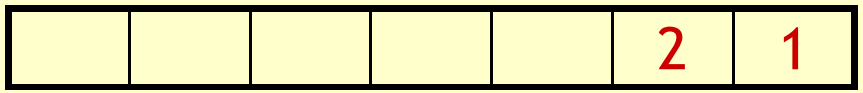
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 2
        mov [edx], eax
        add ebx, SIZE            ebx ← 1004
        sub edx, SIZE            edx ← 1052
        dec ecx                  ecx ← 6
        jnz mloop
```



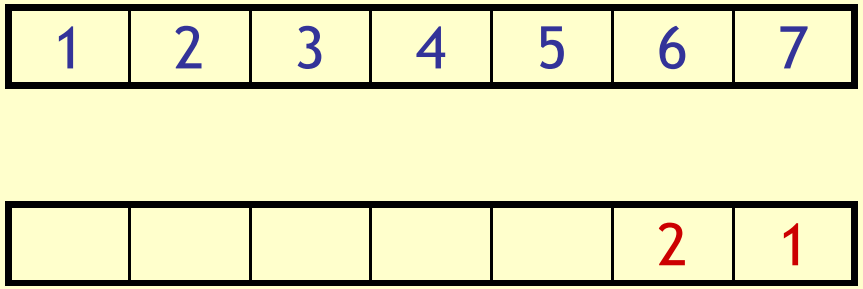
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 2
        mov [edx], eax
        add ebx, SIZE            ebx ← 1004
        sub edx, SIZE            edx ← 1052
        dec ecx                  ecx ← 6
        jnz mloop
```



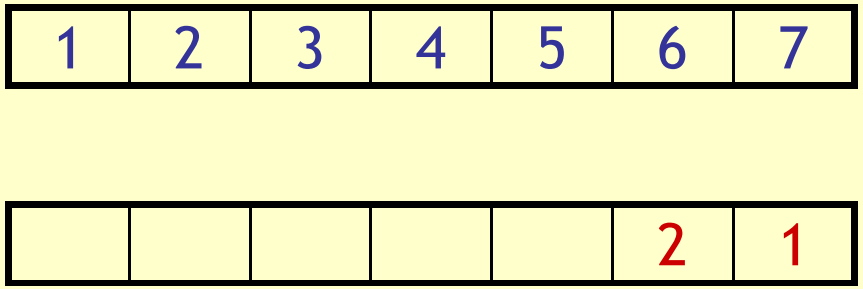
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 2
        mov [edx], eax
        add ebx, SIZE            ebx ← 1008
        sub edx, SIZE            edx ← 1048
        dec ecx                  ecx ← 5
        jnz mloop
```



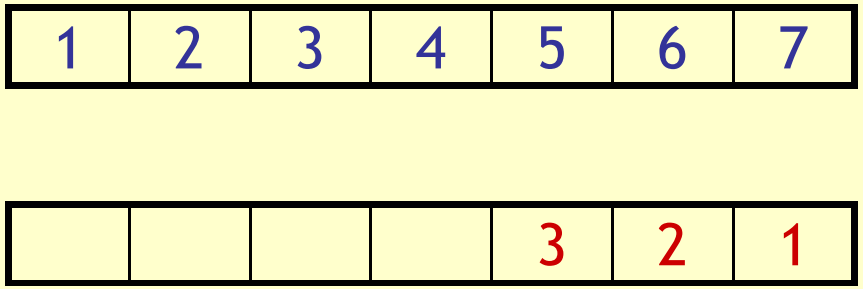
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 3
        mov [edx], eax
        add ebx, SIZE            ebx ← 1008
        sub edx, SIZE            edx ← 1048
        dec ecx                  ecx ← 5
        jnz mloop
```



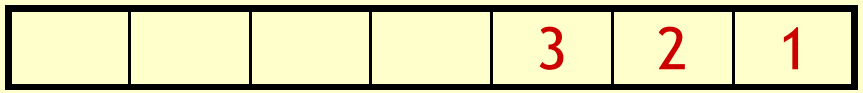
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 3
        mov [edx], eax
        add ebx, SIZE            ebx ← 1012
        sub edx, SIZE            edx ← 1044
        dec ecx                  ecx ← 4
        jnz mloop
```



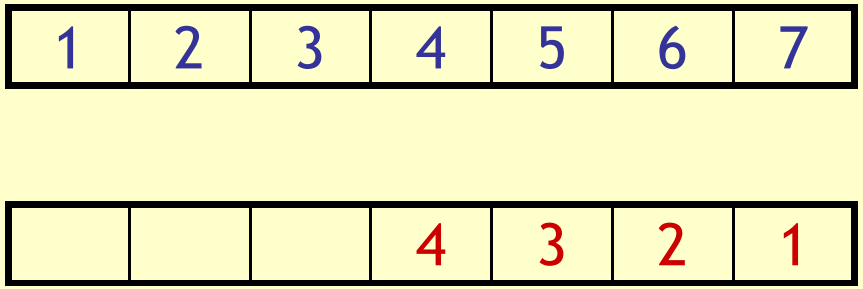
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd
  N:        dd 7
section .bss
  Result:   resd
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 4
        mov [edx], eax
        add ebx, SIZE            ebx ← 1012
        sub edx, SIZE            edx ← 1044
        dec ecx                  ecx ← 4
        jnz mloop
```



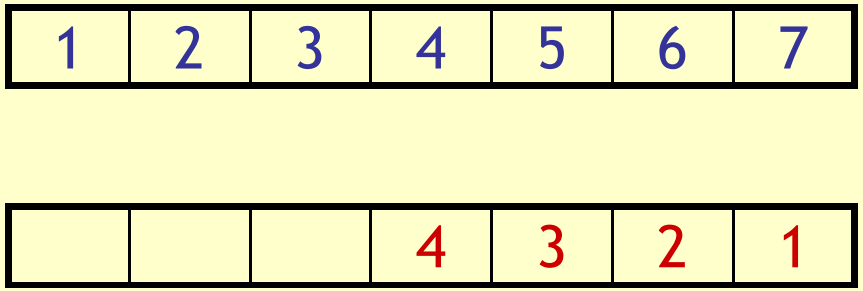
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 4
        mov [edx], eax
        add ebx, SIZE            ebx ← 1016
        sub edx, SIZE            edx ← 1040
        dec ecx                  ecx ← 3
        jnz mloop
```



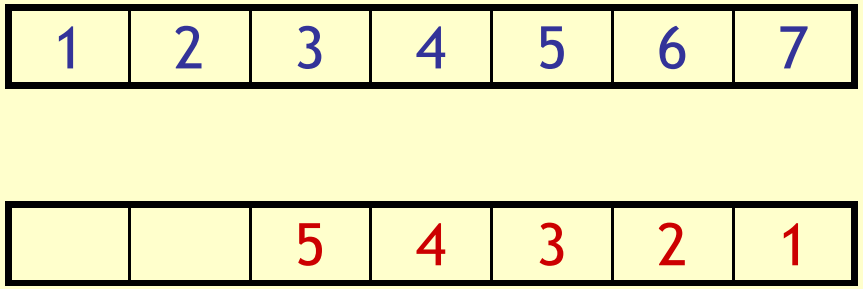
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 5
        mov [edx], eax
        add ebx, SIZE            ebx ← 1016
        sub edx, SIZE            edx ← 1040
        dec ecx                  ecx ← 3
        jnz mloop
```



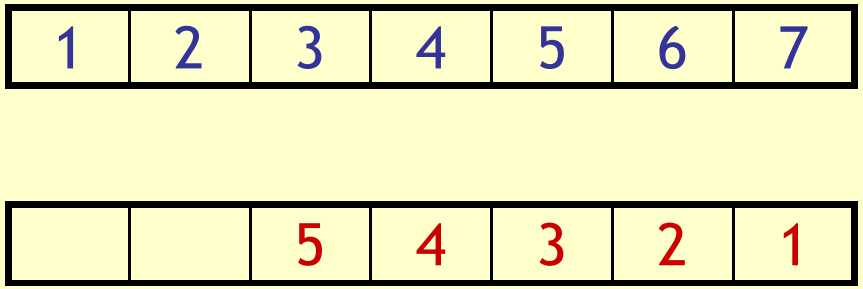
Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 5
        mov [edx], eax
        add ebx, SIZE            ebx ← 1020
        sub edx, SIZE            edx ← 1036
        dec ecx                  ecx ← 2
        jnz mloop
```



Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 6
        mov [edx], eax
        add ebx, SIZE            ebx ← 1020
        sub edx, SIZE            edx ← 1036
        dec ecx                  ecx ← 2
        jnz mloop
```



Cópia Invertida de um Vector

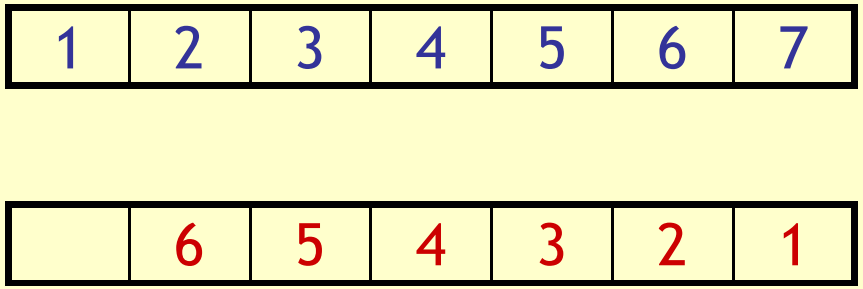
```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 6
        mov [edx], eax
        add ebx, SIZE            ebx ← 1024
        sub edx, SIZE            edx ← 1032
        dec ecx                  ecx ← 1
        jnz mloop
```

1	2	3	4	5	6	7

	6	5	4	3	2	1

Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 7
        mov [edx], eax
        add ebx, SIZE            ebx ← 1020
        sub edx, SIZE            edx ← 1036
        dec ecx                  ecx ← 1
        jnz mloop
```



Cópia Invertida de um Vector

```
SIZE equ 4
global _start
section .data
  vector:   dd 1,2,3,4,5,6,7
  N:        dd 7
section .bss
  Result:   resd 7,6,5,4,3,2,1
section .text
_start: mov ebx, vector          ebx ← 1000
        mov ecx, [N]             ecx ← 7
        lea edx, [Result+(ecx-1)*SIZE]   edx ← 1056
mloop:  mov eax, [ebx]           eax ← 7
        mov [edx], eax
        add ebx, SIZE            ebx ← 1024
        sub edx, SIZE            edx ← 1032
        dec ecx                  ecx ← 0
        jnz mloop
```

Cópia Invertida de um Vector (2)

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov esi, 0
        mov edi, [N]
mloop:  dec edi
        mov eax, [vector+esi*SIZE]  eax ← [1000+0*4]
        mov [Result+edi*SIZE], eax  [1032+6*4] ← eax
        inc esi
        cmp edi, 0 ; também poderia ser cmp esi, [N]
        jne mloop
```

Cópia Invertida de um Vector (2)

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov esi, 0
        mov edi, [N]
mloop:  dec edi
        mov eax, [vector+esi*SIZE]  eax ← [1000+1*4]
        mov [Result+edi*SIZE], eax  [1032+5*4] ← eax
        inc esi
        cmp edi, 0 ; também poderia ser cmp esi, [N]
        jne mloop
```

Cópia Invertida de um Vector (2)

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov esi, 0
        mov edi, [N]
mloop:  dec edi
        mov eax, [vector+esi*SIZE]  eax ← [1000+2*4]
        mov [Result+edi*SIZE], eax  [1032+4*4] ← eax
        inc esi
        cmp edi, 0 ; também poderia ser cmp esi, [N]
        jne mloop
```

Cópia Invertida de um Vector (2)

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov esi, 0
        mov edi, [N]
mloop:  dec edi
        mov eax, [vector+esi*SIZE]  eax ← [1000+3*4]
        mov [Result+edi*SIZE], eax  [1032+3*4] ← eax
        inc esi
        cmp edi, 0 ; também poderia ser cmp esi, [N]
        jne mloop
```

Cópia Invertida de um Vector (2)

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov esi, 0
        mov edi, [N]
mloop:  dec edi
        mov eax, [vector+esi*SIZE]  eax ← [1000+4*4]
        mov [Result+edi*SIZE], eax  [1032+2*4] ← eax
        inc esi
        cmp edi, 0 ; também poderia ser cmp esi, [N]
        jne mloop
```

Cópia Invertida de um Vector (2)

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov esi, 0
        mov edi, [N]
mloop:  dec edi
        mov eax, [vector+esi*SIZE]  eax ← [1000+5*4]
        mov [Result+edi*SIZE], eax  [1032+1*4] ← eax
        inc esi
        cmp edi, 0 ; também poderia ser cmp esi, [N]
        jne mloop
```

Cópia Invertida de um Vector (2)

```
SIZE equ 4
global _start
section .data
  vector:   dd 1, 2, 3, 4, 5, 6, 7
  N:        dd 7
section .bss
  Result:   resd 7
section .text
_start: mov esi, 0
        mov edi, [N]
mloop:  dec edi
        mov eax, [vector+esi*SIZE]  eax ← [1000+6*4]
        mov [Result+edi*SIZE], eax  [1032+0*4] ← eax
        inc esi
        cmp edi, 0 ; também poderia ser cmp esi, [N]
        jne mloop
```