Software Básico

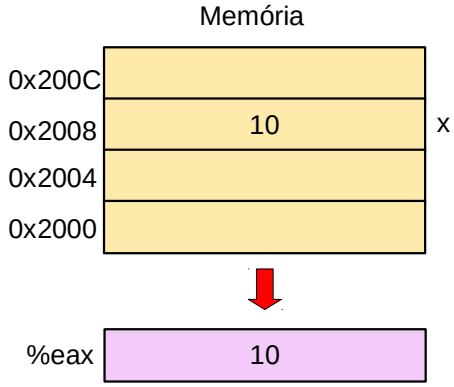
Endereçamento de Memória

Memória: Modo Direto

- O endereço de memória é especificado por uma constante ou rótulo (*label*)

```
.data
    x:    .int    10

.text
main:
    movl x, %eax
```

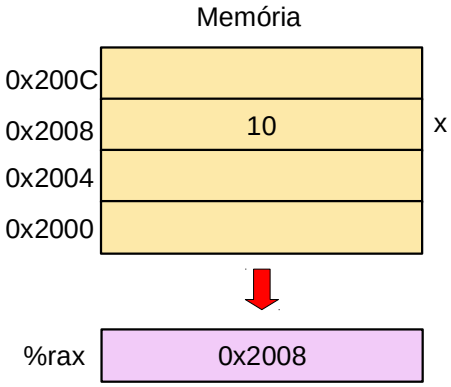


Memória: Modo Direto

- Atenção: se usarmos $x, colocamos o endereço da variável no registrador
- Endereços são 64-bits

```
.data
    x:    .int    10

.text
main:
    movq $x, %rax
```

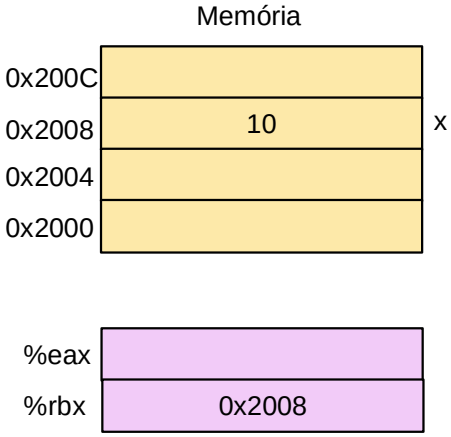


Memória: Modo Indireto

- O endereço de memória está em um registrador
- Formato: (R_{Base})

```
.data
    x:    .int    10

.text
main:
   movl     $1, (%rbx)
   movl (%rbx),  %eax
```

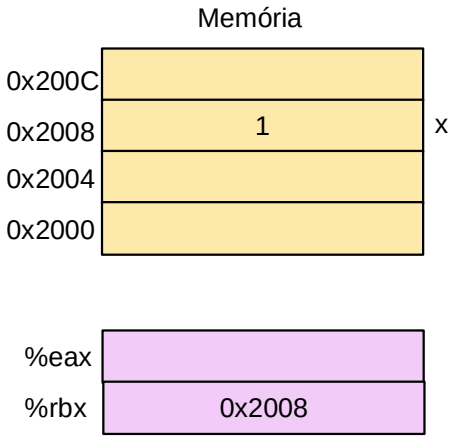


Memória: Modo Indireto

- O endereço de memória está em um registrador
- Formato: (R_{Base})

```
.data
    x:    .int    10

.text
main:
   movl     $1, (%rbx)
   movl (%rbx), %eax
```

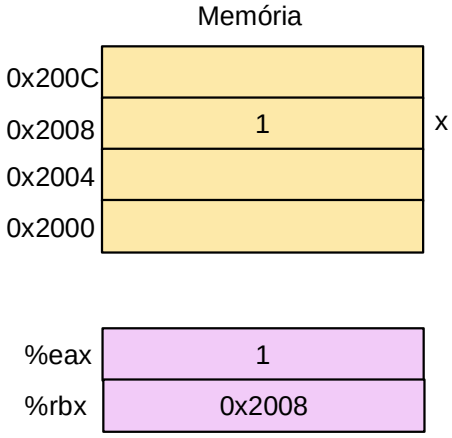


Memória: Modo Indireto

- O endereço de memória está em um registrador
- Formato: (R_{Base})

```
.data
    x:    .int    10

.text
main:
   movl     $1, (%rbx)
   movl (%rbx), %eax
```

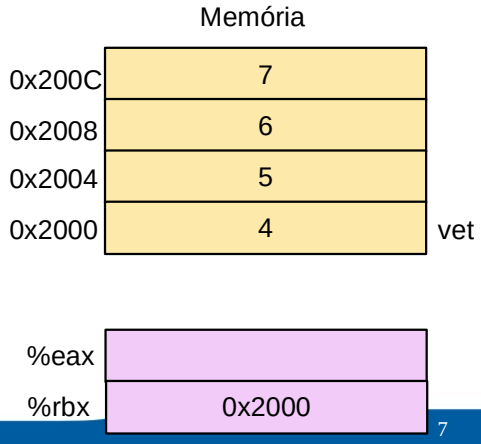


Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: Im(R_{Base}) → Addr = Im + R_{Base}

```
.data
  vet: .int   4,5,6,7

.text
main:
   movl       $1, 4(%rbx)
   movl 8(%rbx),  %eax
```

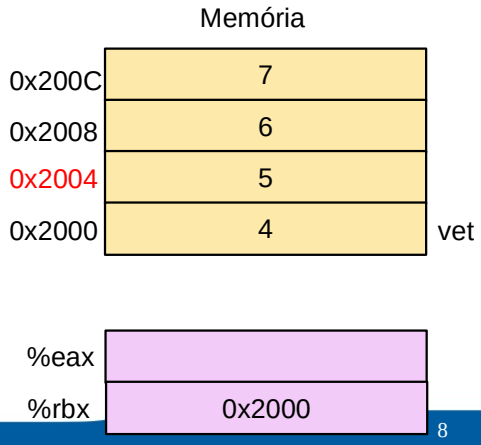


Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: Im(R_{Base}) → Addr = Im + R_{Base}

```
.data
  vet: .int   4,5,6,7

.text
main:
   movl       $1, 4(%rbx)
   movl 8(%rbx),  %eax
```

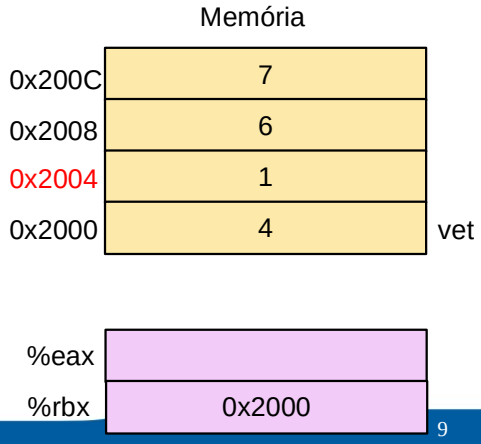


Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: Im(R_{Base}) → Addr = Im + R_{Base}

```
.data
  vet: .int   4,5,6,7

.text
main:
   movl       $1, 4(%rbx)
   movl 8(%rbx),  %eax
```

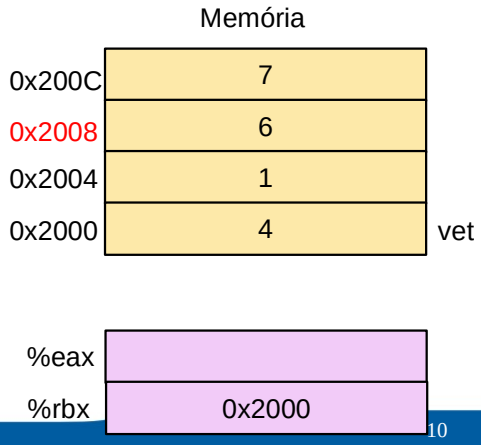


Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: Im(R_{Base}) → Addr = Im + R_{Base}

```
.data
  vet: .int   4,5,6,7

.text
main:
   movl       $1, 4(%rbx)
   movl 8(%rbx),  %eax
```

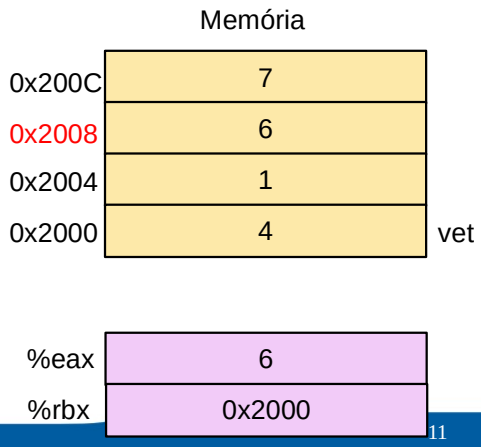


Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: Im(R_{Base}) → Addr = Im + R_{Base}

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl       $1, 4(%rbx)
   movl 8(%rbx),  %eax
```

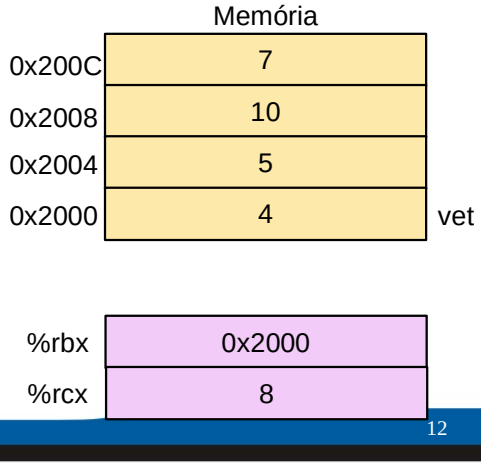


Memória: Indexado (1)

- Base é um registrador tem um endereço de memória
- Índice é um registrador com deslocamento
- Formato: (R_{Base},R_{Idx}) → Addr = R_{Base} + R_{Idx}

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10,  (%rbx,%rcx)
```

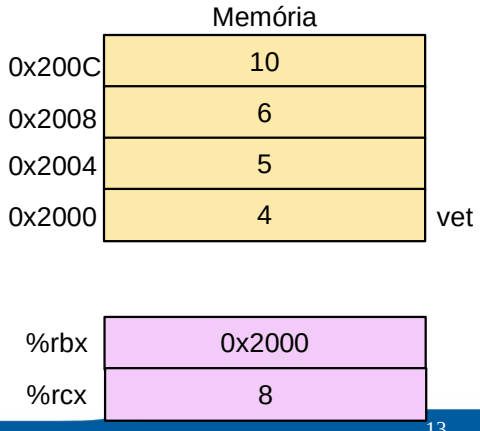


Memória: Indexado (2)

- Base é um registrador tem um endereço de memória
- Índice é um registrador com deslocamento
- Formato: $Im(R_{Base},R_{Idx})$ → $Addr = Im + R_{Base} + R_{Idx}$

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10, 4(%rbx,%rcx)
```

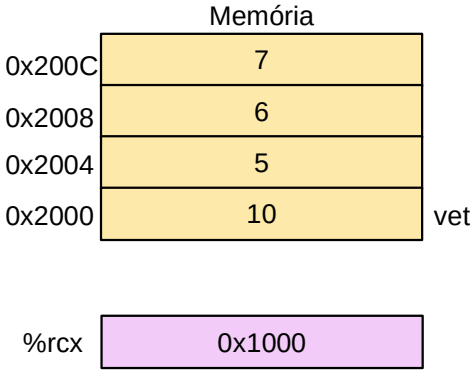


Memória: Indexado e Escalado (1)

- Registrador índice tem um endereço de memória
- A escala s pode ser 1, 2, 4 ou 8
- Formato: $(, R_{Idx}, s)$ → $Addr = s * R_{Idx}$

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10, ( , %rcx, 2)
```

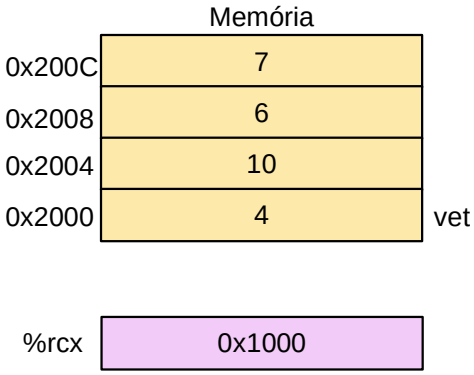


Memória: Indexado e Escalado (2)

- Registrador índice tem um endereço de memória
- A escala s pode ser 1, 2, 4 ou 8
- Formato: Im(, R_{Idx}, s) → Addr = Im + s * R_{Idx}

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10, 4( , %rcx, 2)
```

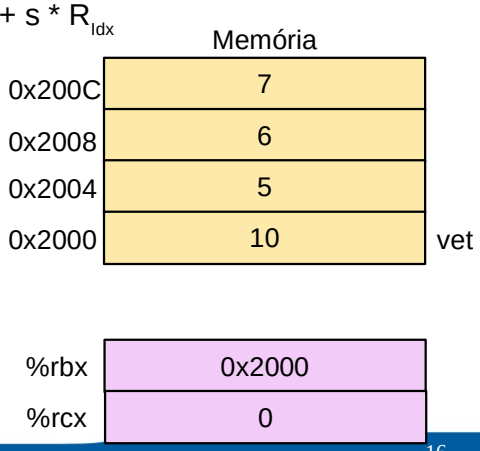


Memória: Indexado e Escalado (3)

- Registrador base tem um endereço de memória
- Registrador índice é usado para deslocamento
- A escala s pode ser 1, 2, 4 ou 8
- Formato: (R_{Base} , R_{Idx}, s) → Addr = R_{Base} + s * R_{Idx}

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10, (%rbx, %rcx, 4)
   incq %rcx
   movl $11, (%rbx, %rcx, 4)
```

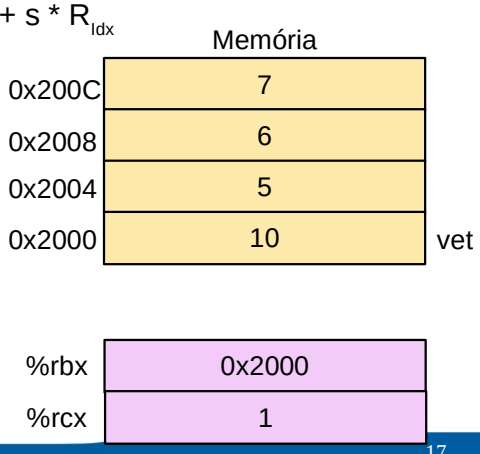


Memória: Indexado e Escalado (3)

- Registrador base tem um endereço de memória
- Registrador índice é usado para deslocamento
- A escala s pode ser 1, 2, 4 ou 8
- Formato: (R_{Base} , R_{Idx}, s) → Addr = R_{Base} + s * R_{Idx}

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10, (%rbx, %rcx, 4)
   incq %rcx
   movl $11, (%rbx, %rcx, 4)
```

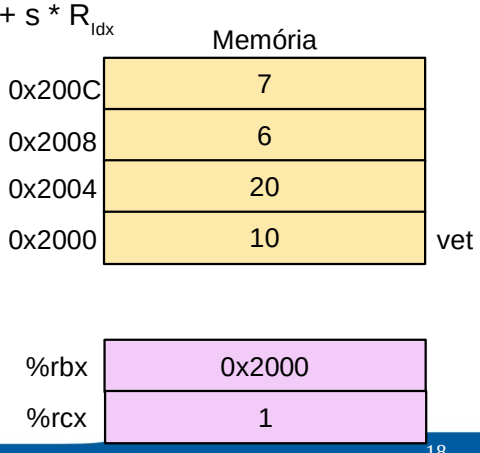


Memória: Indexado e Escalado (3)

- Registrador base tem um endereço de memória
- Registrador índice é usado para deslocamento
- A escala s pode ser 1, 2, 4 ou 8
- Formato: (R_{Base} , R_{Idx}, s) → Addr = R_{Base} + s * R_{Idx}

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10, (%rbx, %rcx, 4)
   incq %rcx
   movl $20, (%rbx, %rcx, 4)
```

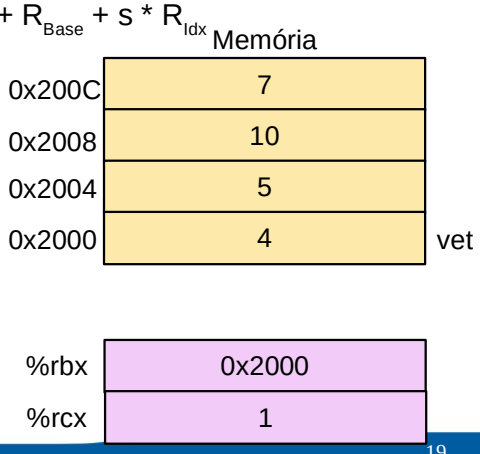


Memória: Indexado e Escalado (4)

- Registrador base tem um endereço de memória
- Registrador índice é usado para deslocamento
- A escala s pode ser 1, 2, 4 ou 8
- Formato: $Im(R_{Base}, R_{Idx}, s)$ → $Addr = Im + R_{Base} + s * R_{Idx}$

```
.data
  vet: .int  4,5,6,7

.text
main:
   movl $10, 4(%rbx, %rcx, 4)
```



Instrução “leaq”

- A instrução “leaq” resolve (calcula) o endereço final e o armazenando em um registrador
- Não há nenhum acesso à memória

```
leaq 4(%rbx), %rax          /* rax = %rbx + 4         */
leaq (%rbx, %rcx, 4), %rdi  /* rdi = %rbx + 4 * %rcx */
leaq 4( , %rcx, 2), %rax    /* rax = 4 + 2 * %rcx    */
```