

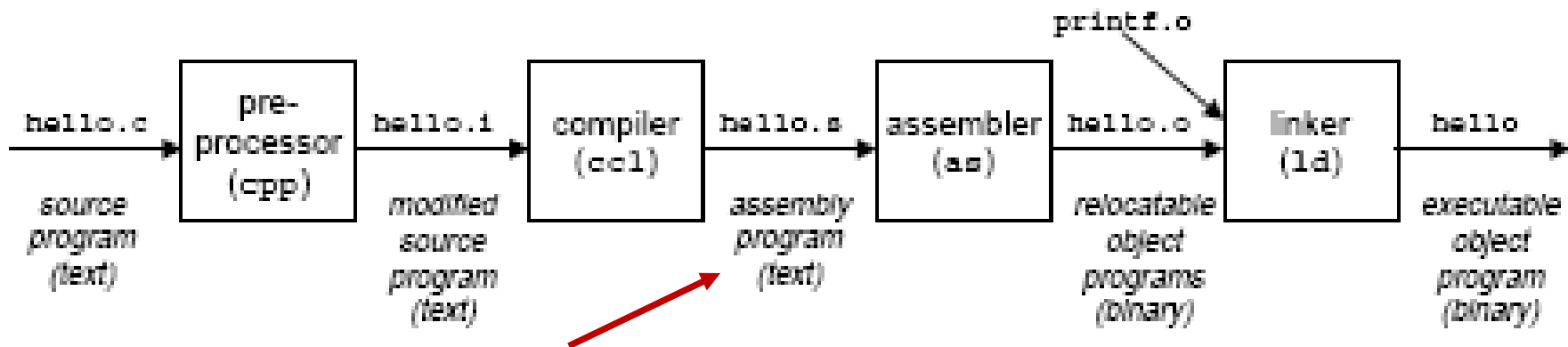
Introdução ao Assembly

Movimentação de Dados
Operações Aritméticas e Lógicas

Noemi Rodriguez
Ana Lúcia de Moura
Raúl Renteria
Alexandre Meslin

<http://www.inf.puc-rio.br/~inf1018>

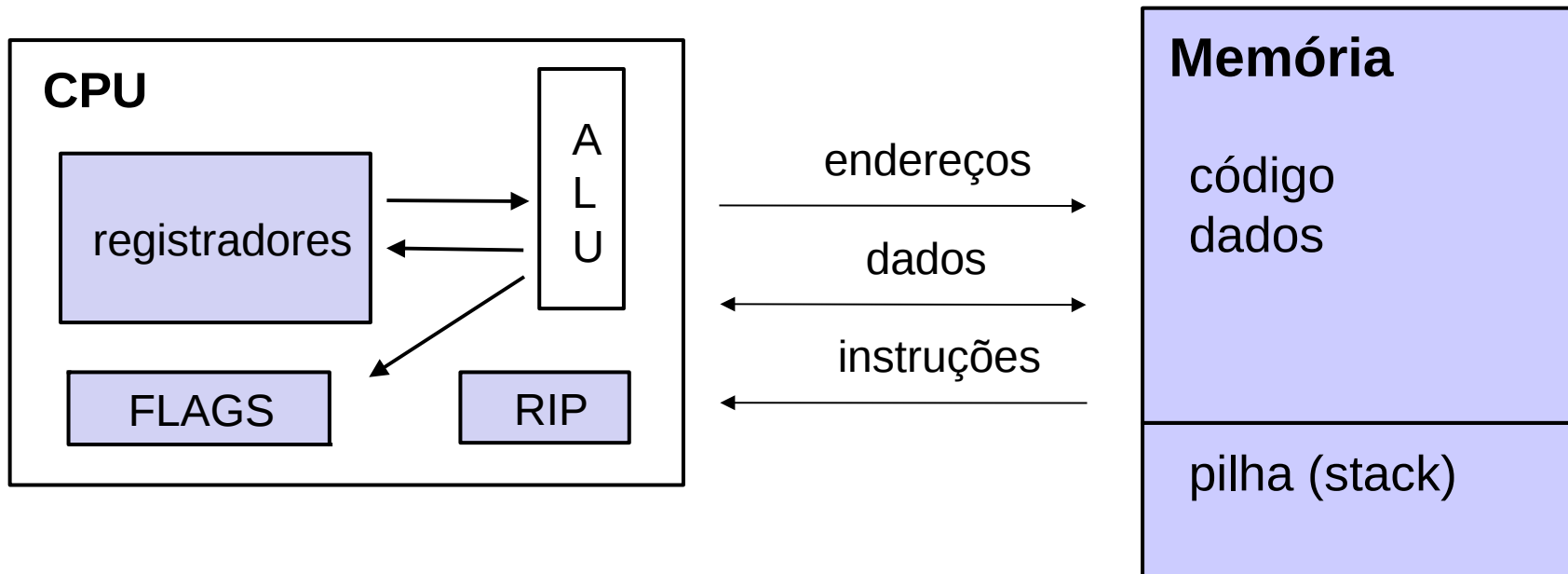
Representação de Programas



O compilador gera o código de um programa conforme

- o conjunto de instruções da máquina alvo
- as regras estabelecidas pela linguagem de programação (C)
- as convenções seguidas pelo sistema operacional

Visão do Programador Assembly



RIP: endereço da próxima instrução

Registradores:

- valores inteiros, endereços

FLAGS: status da última operação

- overflow? zero? resultado < 0 ?

Linguagem de Montagem

Instruções executam operações simples

- operações aritméticas/lógicas
- transferência de dados
- controle do fluxo de execução (desvios, chamadas de função)

Linguagem de Montagem

Instruções executam operações simples

- operações aritméticas/lógicas
- transferência de dados
- controle do fluxo de execução (desvios, chamadas de função)

Tipos de dados básicos

- valores inteiros (1,2,4,8 bytes)
- endereços de memória
- valores em ponto flutuante

Programa em *assembly*

dados globais →
código →
labels →

```
.data
nums: .int 10, -21, -30, 45
...
.text
.globl main
main:
...
movl $0, %ebx      /* i */
movq $nums, %r12   /* p */
L1:
cmpl $4, %ebx      /* if (i == 4) */
je L2               /* goto L2 */
movl (%r12), %eax   /* eax = *p */
...
addl $1, %ebx      /* i++ */
addq $4, %r12      /* p++ */
jmp L1
L2:
...
ret
```

```
int nums[] = {10, -21, -30, 45};
int main() {
    int i, *p;
    for (i = 0, p = nums; i != 4; i++, p++)
        printf("%d\n", *p);
    return 0;
}
```

Registadores

| 63 | 31 | 15 | 8 | 7 | 0 |
|------|-------|-------|-----|---|-------|
| %rax | %eax | %ax | %ah | | %al |
| %rbx | %ebx | %bx | %bh | | %bl |
| %rcx | %ecx | %cx | %ch | | %cl |
| %rdx | %edx | %dx | %dh | | %dl |
| %rsi | %esi | %si | | | %sil |
| %rdi | %edi | %di | | | %dil |
| %rbp | %ebp | %bp | | | %bpl |
| %rsp | %esp | %sp | | | %spl |
| %r8 | %r8d | %r8w | | | %r8b |
| %r9 | %r9d | %r9w | | | %r9b |
| %r10 | %r10d | %r10w | | | %r10b |
| %r11 | %r11d | %r11w | | | %r11b |
| %r12 | %r12d | %r12w | | | %r12b |
| %r13 | %r13d | %r13w | | | %r13b |
| %r14 | %r14d | %r14w | | | %r14b |
| %r15 | %r15d | %r15w | | | %r15b |

Armazenam valores inteiros e endereços (ponteiros)

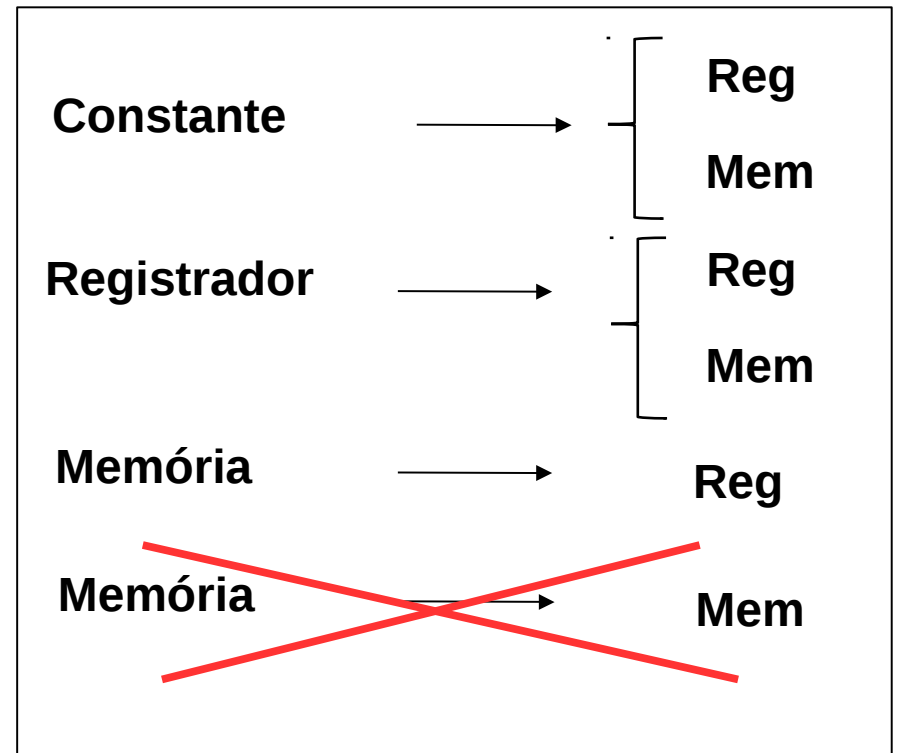
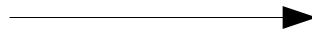
- podem ser usados como valores de 64, 32, 16 e 8 bits

%rbp e **%rsp** são usados como ponteiros para a pilha

Movimentação de Dados

mov

fonte, destino

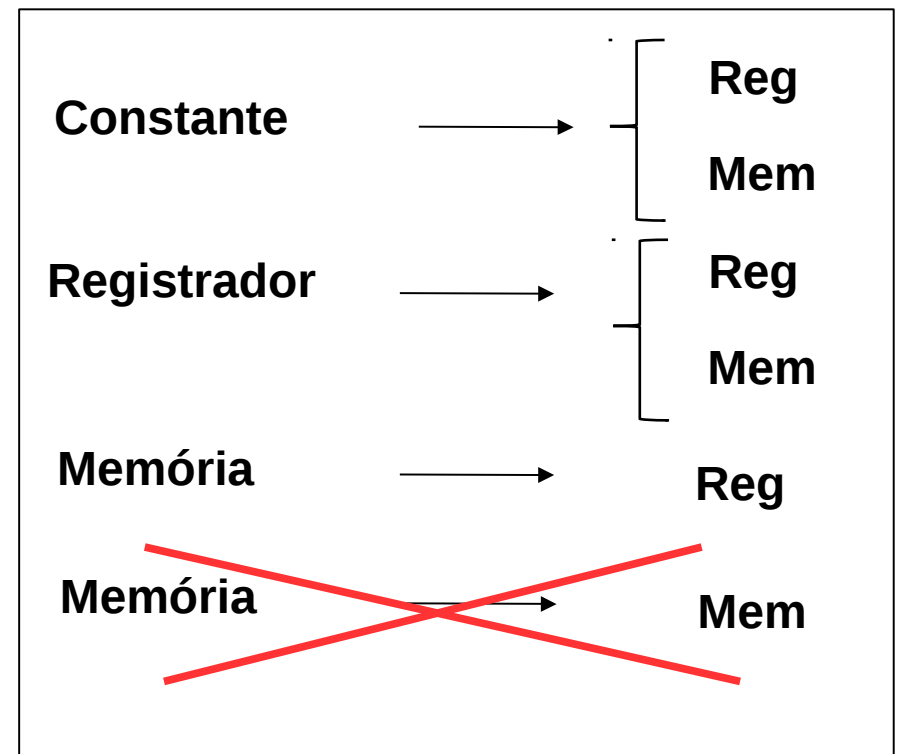


Movimentação de Dados

mov[b w l q] fonte, destino

o sufixo especifica o tamanho dos operandos

```
movl    $0x4050, %eax
movq    $nums, %r12
movl    %ebx, %ecx
movq    %r12, %r13
movl    %edx, (%rbx)
movl    (%r12), %eax
movb    (%r12), %al
```



Constantes e Registradores

Constantes (immediatos) são escritas com um \$ seguido por valor inteiro em notação C

Registradores são especificados pelo **nome**

Constantes e Registradores

Constantes (immediatos) são escritas com um **\$** seguido por valor inteiro em notação C

Registradores são especificados pelo **nome**

```
movl $1024, %eax
```

```
movabsq $-1, %rax /* para constantes de 64 bits */
```

```
movl $0xFF, %ebx
```

```
movb $0, %al
```

```
movl %ebx, %ecx
```

```
movq %r12, %r13
```

Valores em Memória: modo indireto

O **endereço** de memória está em um **registrador**

- o nome do registrador é escrito **entre parênteses**

Valores em Memória: modo indireto

O **endereço** de memória está em um **registrador**

- o nome do registrador é escrito **entre parênteses**

%rbx

0x7fff526a897c

Valores em Memória: modo indireto

O **endereço** de memória está em um **registrador**

- o nome do registrador é escrito **entre parênteses**

`%rbx`

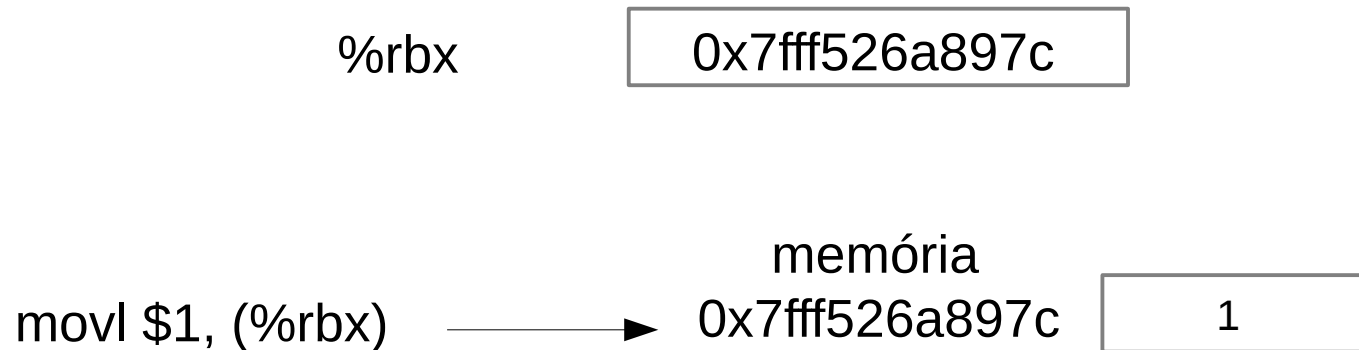
`0x7fff526a897c`

```
movl $1, (%rbx)
```

Valores em Memória: modo indireto

O **endereço** de memória está em um **registrador**

- o nome do registrador é escrito **entre parênteses**



Valores em Memória: modo indireto

O **endereço** de memória está em um **registrador**

- o nome do registrador é escrito **entre parênteses**

`%rbx`

`0x7fff526a897c`

`movl $1, (%rbx)`



memória

`0x7fff526a897c`

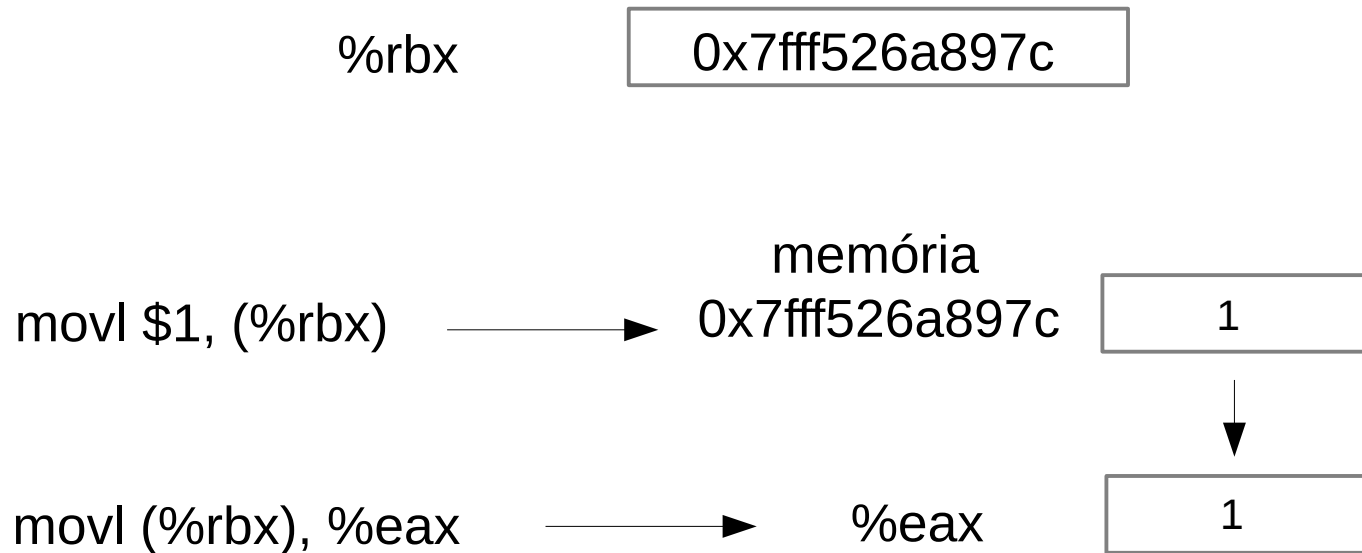
`1`

`movl (%rbx), %eax`

Valores em Memória: modo indireto

O **endereço** de memória está em um **registrador**

- o nome do registrador é escrito **entre parênteses**



Valores em Memória: base-deslocamento

Um registrador tem um **endereço** de memória

- ao endereço é somado um **deslocamento**

Valores em Memória: base-deslocamento

Um registrador tem um **endereço** de memória

- ao endereço é somado um **deslocamento**

%rbx

0x7fff526a8970

Valores em Memória: base-deslocamento

Um registrador tem um **endereço** de memória

- ao endereço é somado um **deslocamento**

%rbx

0x7fff526a8970

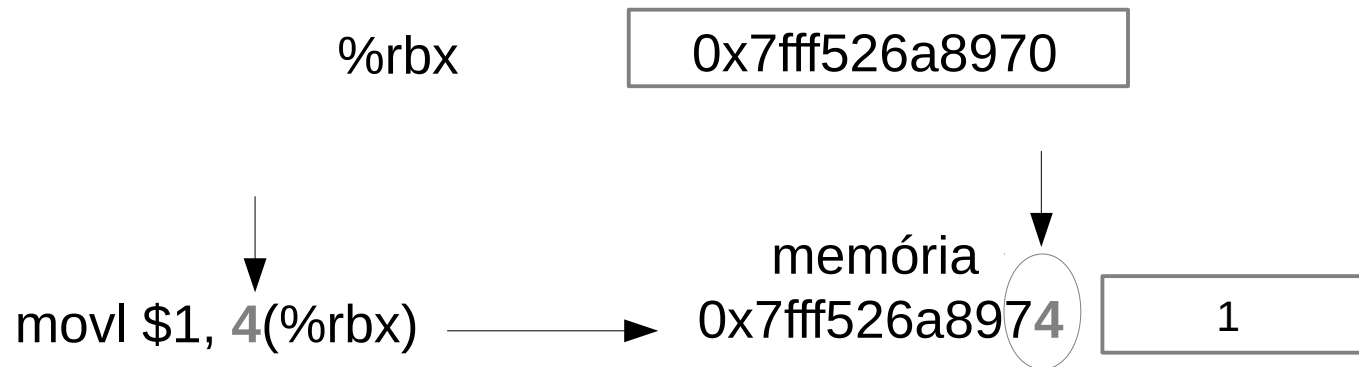


movl \$1, 4(%rbx)

Valores em Memória: base-deslocamento

Um registrador tem um **endereço** de memória

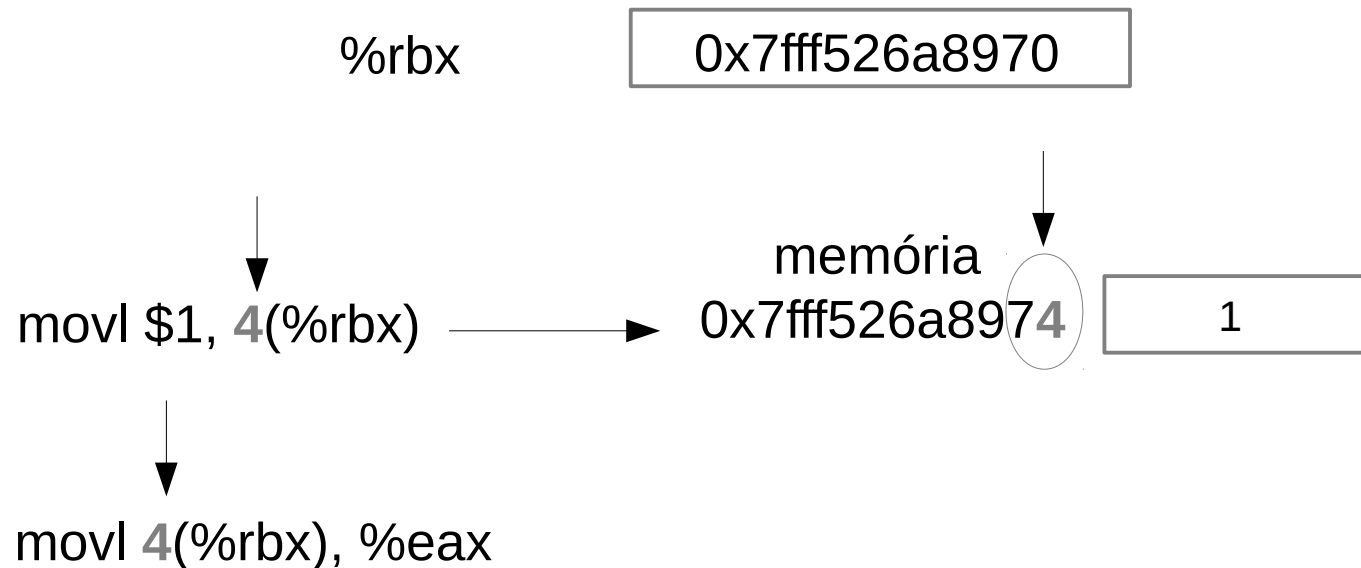
- ao endereço é somado um **deslocamento**



Valores em Memória: base-deslocamento

Um registrador tem um **endereço** de memória

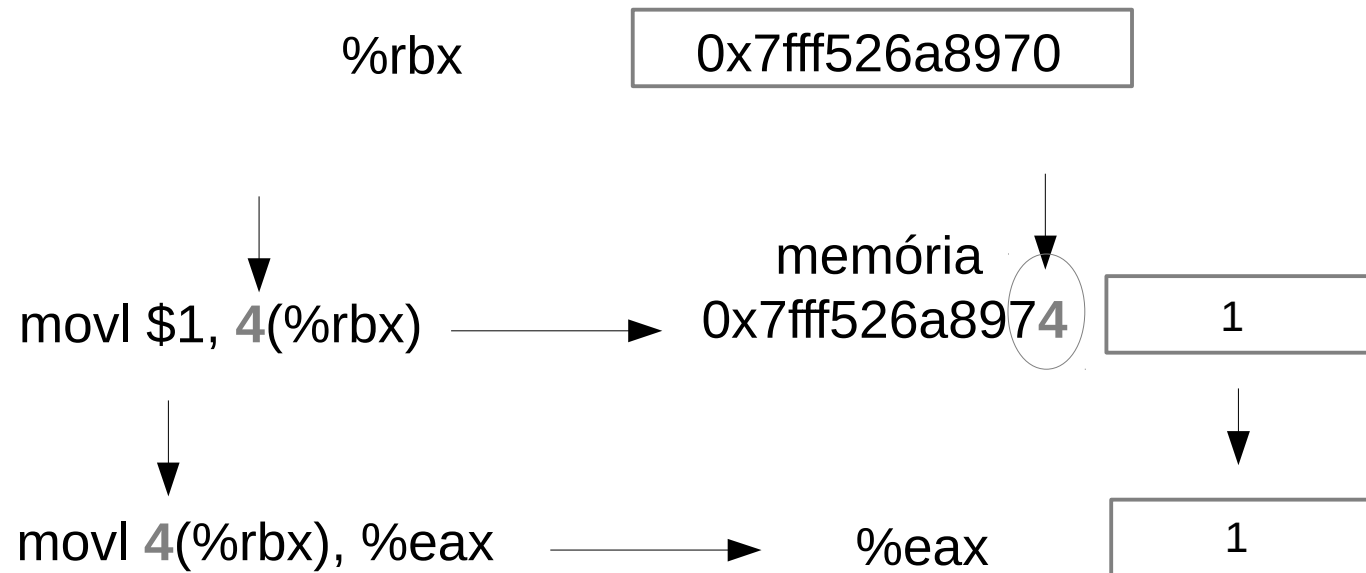
- ao endereço é somado um **deslocamento**



Valores em Memória: base-deslocamento

Um registrador tem um **endereço** de memória

- ao endereço é somado um **deslocamento**



Valores em Memória: modo absoluto

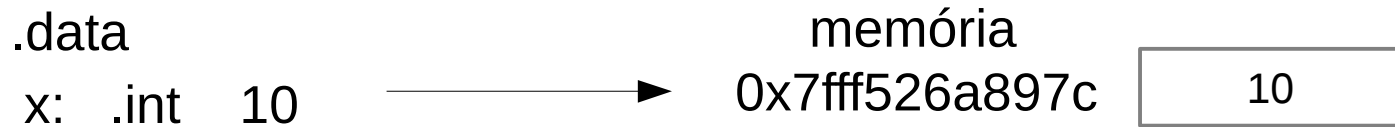
O **endereço** de memória é especificado por uma **constante** ou rótulo (*label*)

- não existe o conceito de variável!

Valores em Memória: modo absoluto

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

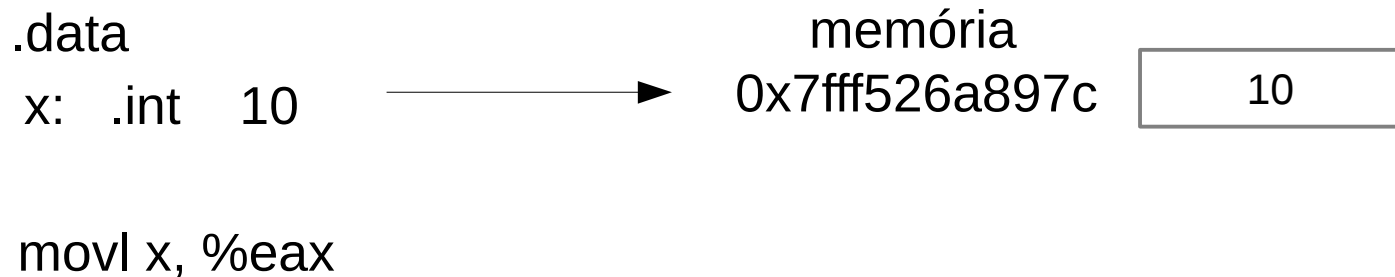
- não existe o conceito de variável!



Valores em Memória: modo absoluto

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

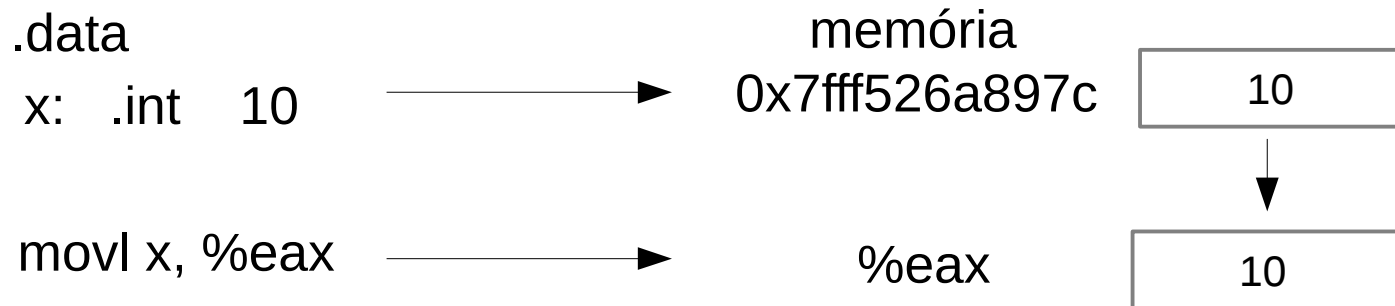
- não existe o conceito de variável!



Valores em Memória: modo absoluto

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

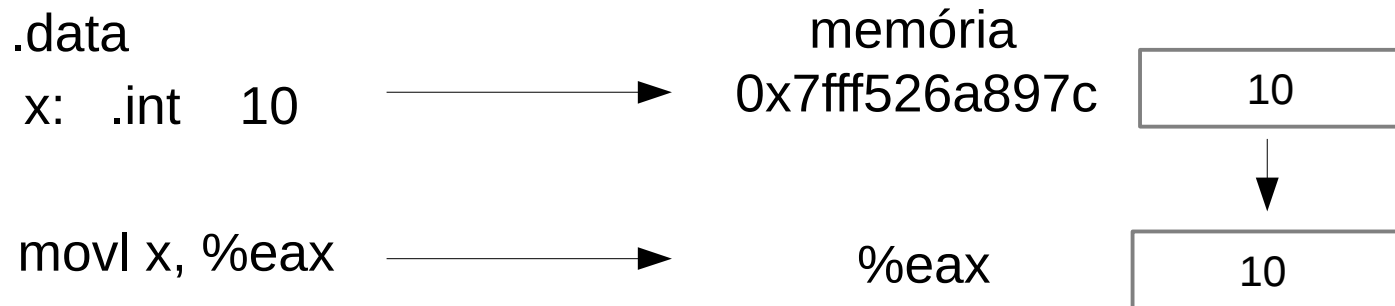
- não existe o conceito de variável!



Valores em Memória: modo absoluto

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

- não existe o conceito de variável!

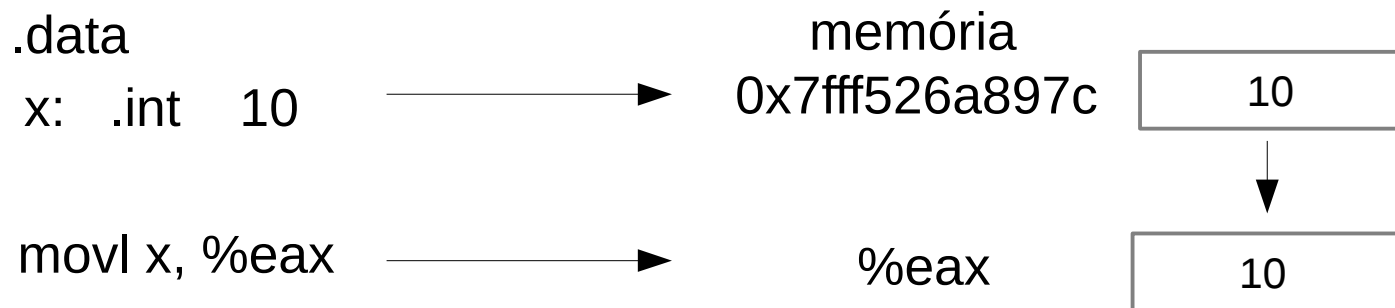


\$x é uma constante (é o **endereço** associado a `x`)

Valores em Memória: modo absoluto

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

- não existe o conceito de variável!



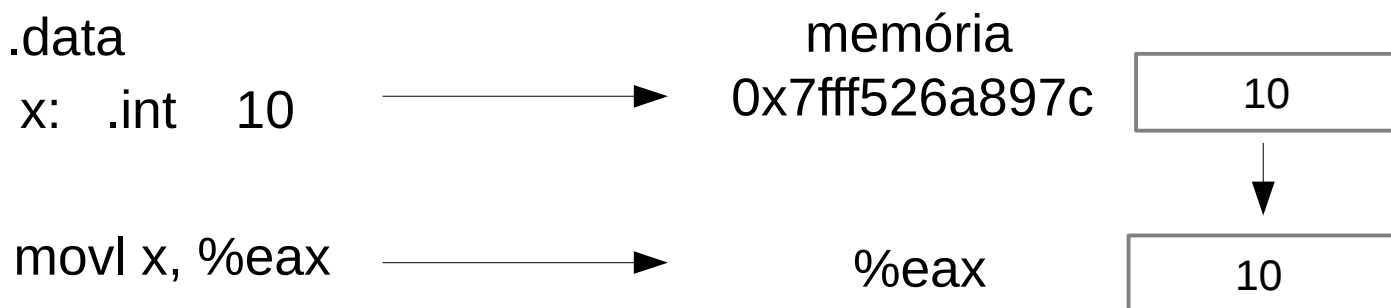
\$x é uma constante (**endereço** associado a x)

```
movq $x, %rbx
```

Valores em Memória: modo absoluto

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

- não existe o conceito de variável!



\$x é uma constante (**endereço** associado a x)



Valores em memória: movimentação

Movimento de dados sempre

CPU ↔ memória

Nunca dois operandos em memória !!!

Valores em memória: movimentação

Movimento de dados sempre

CPU ↔ memória

Nunca dois operandos em memória !!!

~~movl (%rcx), (%rax)~~

~~movl 8(%rbx), (%rcx)~~

~~movl x, (%rax)~~

~~movl 4(%rcx), x~~

Movimentação com Extensão

Extensão com sinal (movs) ou com zeros (movz)

movs[bw][bl][bq][wl][wq][lq]

movz[bw][bl][bq][wl][wq] → não existe movzlq !

Movimentação com Extensão

Extensão com sinal (`movs`) ou com zeros (`movz`)

`movs[bw][bl][bq][wl][wq][lq]`

`movz[bw][bl][bq][wl][wq]` → não existe `movzlq` !

```
movsbl (%r12), %eax
movzbl %al, %ebx
movslq %ebx, %rcx
```

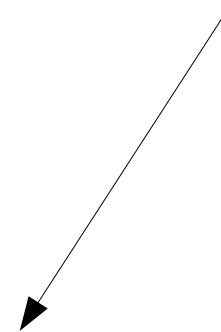
Movimentação com Extensão

Extensão com sinal (`movs`) ou com zeros (`movz`)

`movs[bw][bl][bq][wl][wq][lq]`

`movz[bw][bl][bq][wl][wq]` → não existe `movzlq` !

```
movsbl (%r12), %eax
movzbl %al, %ebx
movslq %ebx, %rcx
```



Instruções que escrevem um valor em registrador de 32 bits **zeram** a parte alta do registrador de 64 bits correspondente ...

Operações Aritméticas: um operando

inc <dest>

dec <dest>

neg <dest>

} registrador ou memória

Operações Aritméticas: um operando

inc <dest>

dec <dest>

neg <dest>

} registrador ou memória

```
incl %eax    /* %eax = %eax + 1 */
incl (%rdx) /* (%rdx) = (%rdx) + 1 */
decl %eax    /* %eax = %eax - 1 */
negl %ebx    /* %ebx = -%ebx */
```

Operações Aritméticas: dois operandos

add <s>,<d>

sub <s>,<d>

imul <s>,<d>

$d = d [+ - *] s$

nunca memória com memória !

Operações Aritméticas: dois operandos

add <s>, <d>

sub <s>, <d>

imul <s>, <d>

$$d = d [+ - *] s$$

nunca memória com memória !

```
addl %ebx, %eax    /* %eax = %eax + %ebx */
addq $4, %rbx      /* %rbx = %rbx + 4 */
addl 4(%r12), %eax /* %eax = %eax + 4(%r12) */
subl %ebx, %eax     /* %eax = %eax - %ebx */
imull %ebx, %eax    /* %eax = %eax * %ebx */
```

Operações Lógicas (bit a bit)

and <s>, <d>

or <s>, <d>

xor <s>, <d>

$d = d [\& | ^] s$

nunca memória com memória !

Operações Lógicas (bit a bit)

and <s>,<d>

or <s>,<d>

xor <s>,<d>

$d = d [\& | \wedge] s$

nunca memória com memória !

```
andl $0x7FFFFFFF,%eax /* %eax = %eax & 0x7FFFFFFF */
andl %ebx,%eax        /* %eax = %eax & %ebx */
orl  (%rcx),%eax      /* %eax = %eax | (%rcx) */
xorl %eax,%ebx        /* %ebx = %ebx ^ %eax */
```

Deslocamento (shift)

shl <i>, <d>

shr <i>, <d>

sar <i>, <d>

$d = d \ll i$

$d = d \gg i$ (lógico)

$d = d \ggg i$ (aritmético)

Deslocamento (shift)

shl <i>,<d>

shr<i>,<d>

sar <i>,<d>

d = d << i

d = d >> i (lógico)

d = d >> i (aritmético)

```
shll $2,%eax      /* %eax = %eax << 2 */
shrl $16,%eax     /* %eax = %eax >> 16 (lógico) */
sarl $3,%ebx      /* %ebx = %ebx >> 3 (aritmético) */
```