Procedimentos
Chamada de Funções e Parâmetros

Noemi Rodriguez
Ana Lúcia de Moura
Raúl Renteria
Alexandre Meslin

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

Memória

Durante a execução de um programa, o SO precisa alocar memória principal para:

- dados globais
- código

} tamanho (fixo) conhecido na compilação

Memória

Durante a execução de um programa, o SO precisa alocar memória principal para:

dados globais
código
} tamanho (fixo) conhecido na compilação

variáveis locais → para cada chamada de função

Memória

Durante a execução de um programa, o SO precisa alocar memória principal para:

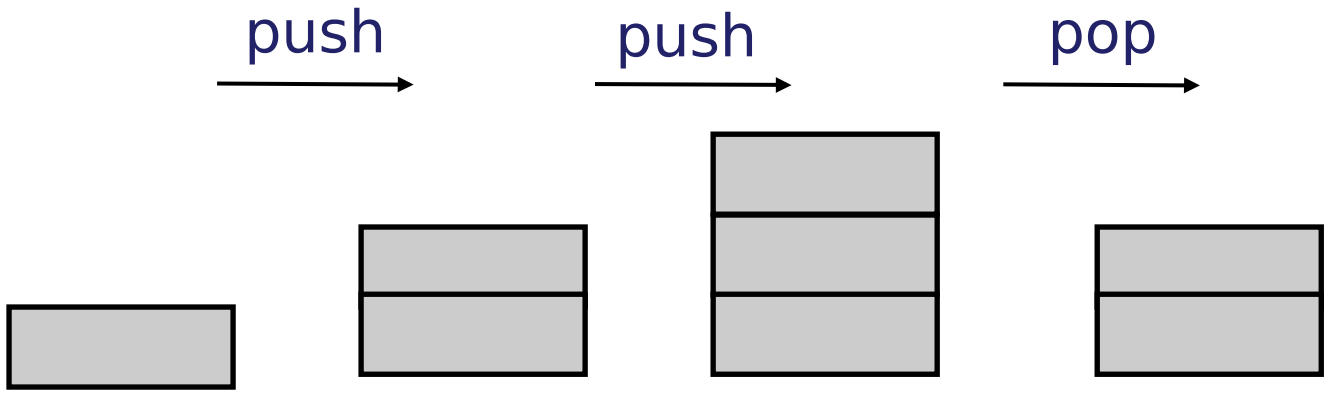
dados globais
código
} tamanho (fixo) conhecido na compilação

variáveis locais → para cada chamada de função

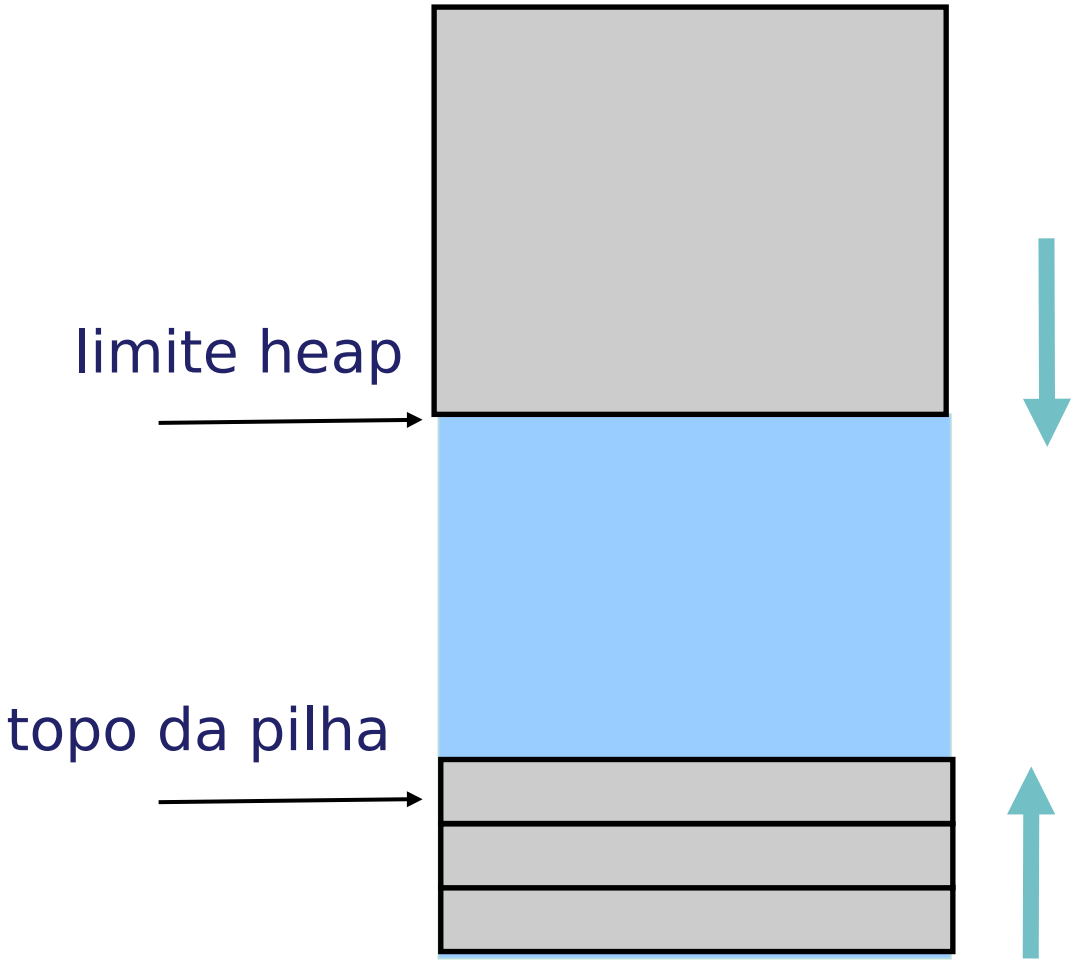
resultados intermediários → quantos? quando?

Pilha de Execução

Para acomodar necessidades de alocação temporária de memória o sistema provê uma **pilha**



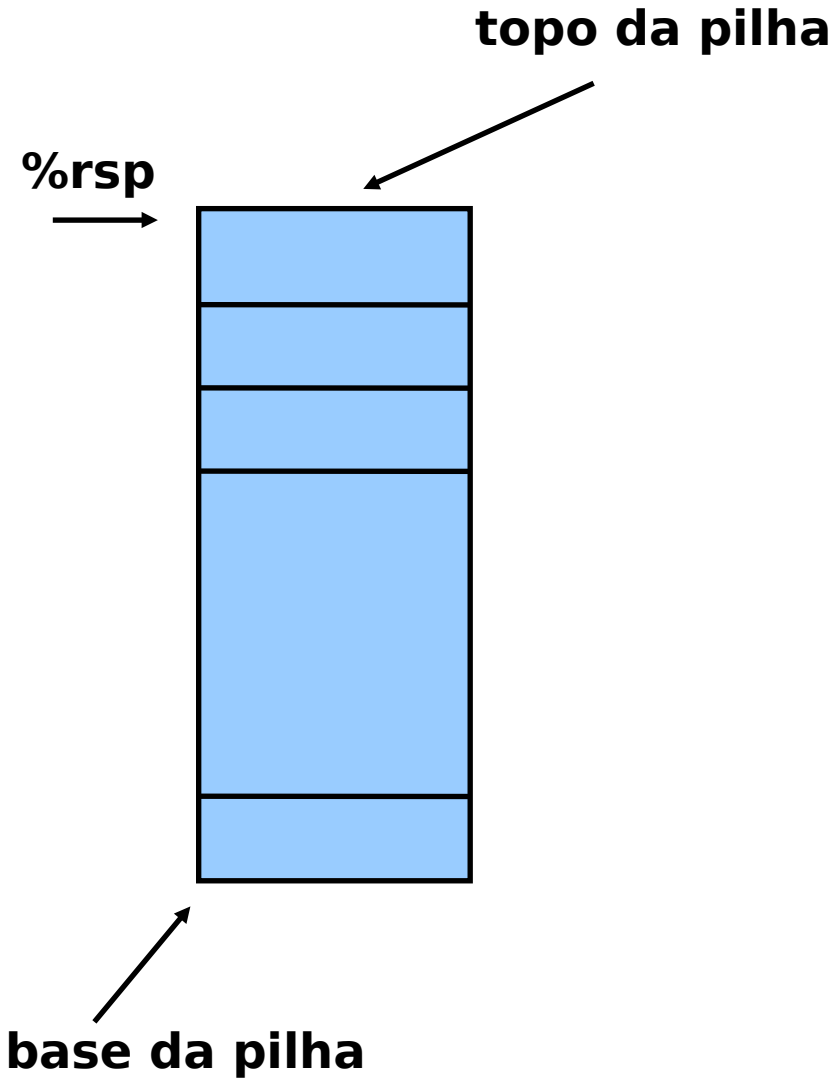
Área de Memória para Pilha



Implementação da Pilha

Registrador dedicado para apontar o topo da pilha

instruções pushq e popq

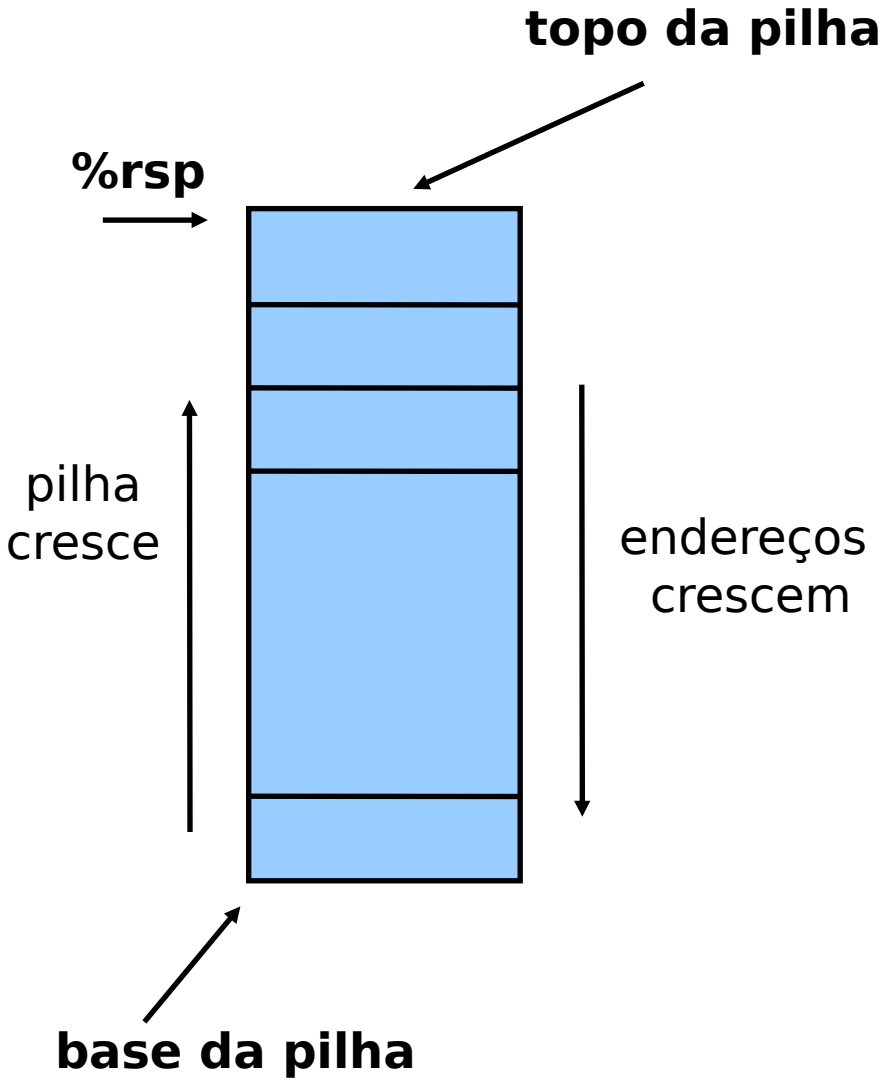


Implementação da Pilha

Registrador dedicado para apontar o topo da pilha

instruções pushq e popq

A pilha “cresce” em direção ao início da memória



Implementação da Pilha

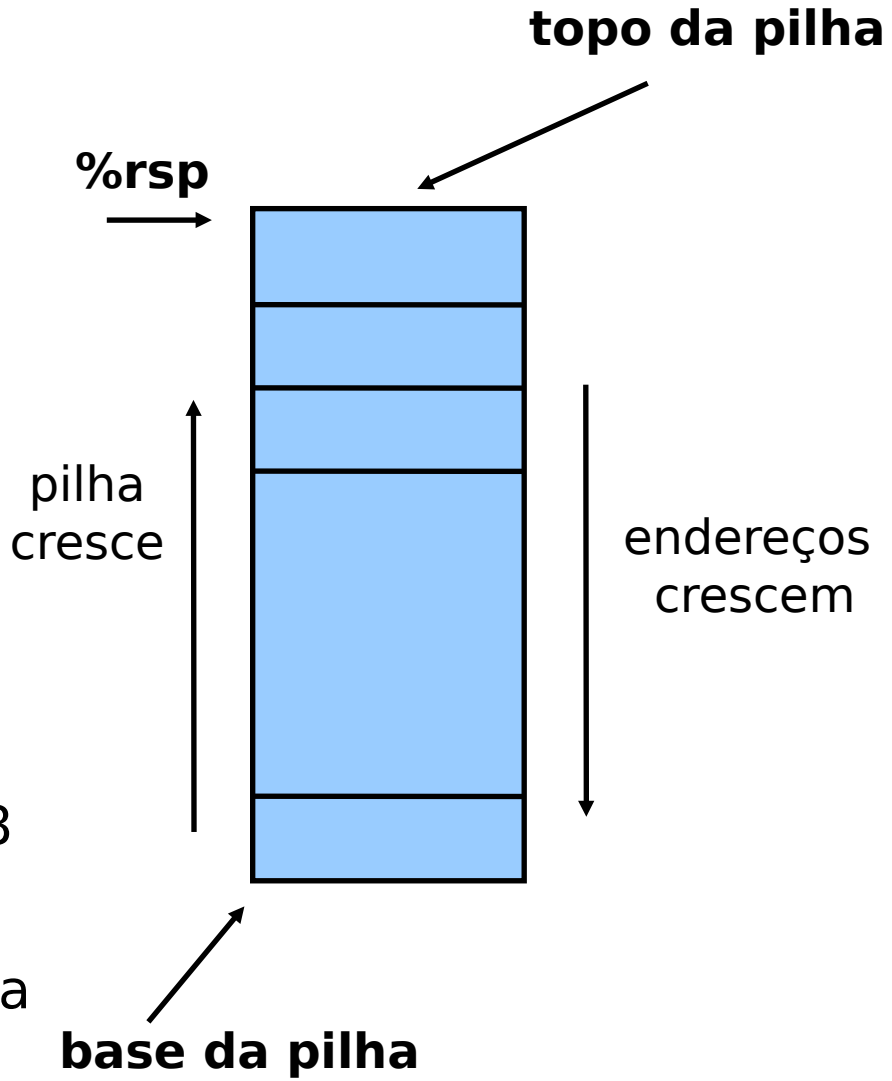
Registrador dedicado para apontar o topo da pilha

instruções pushq e popq

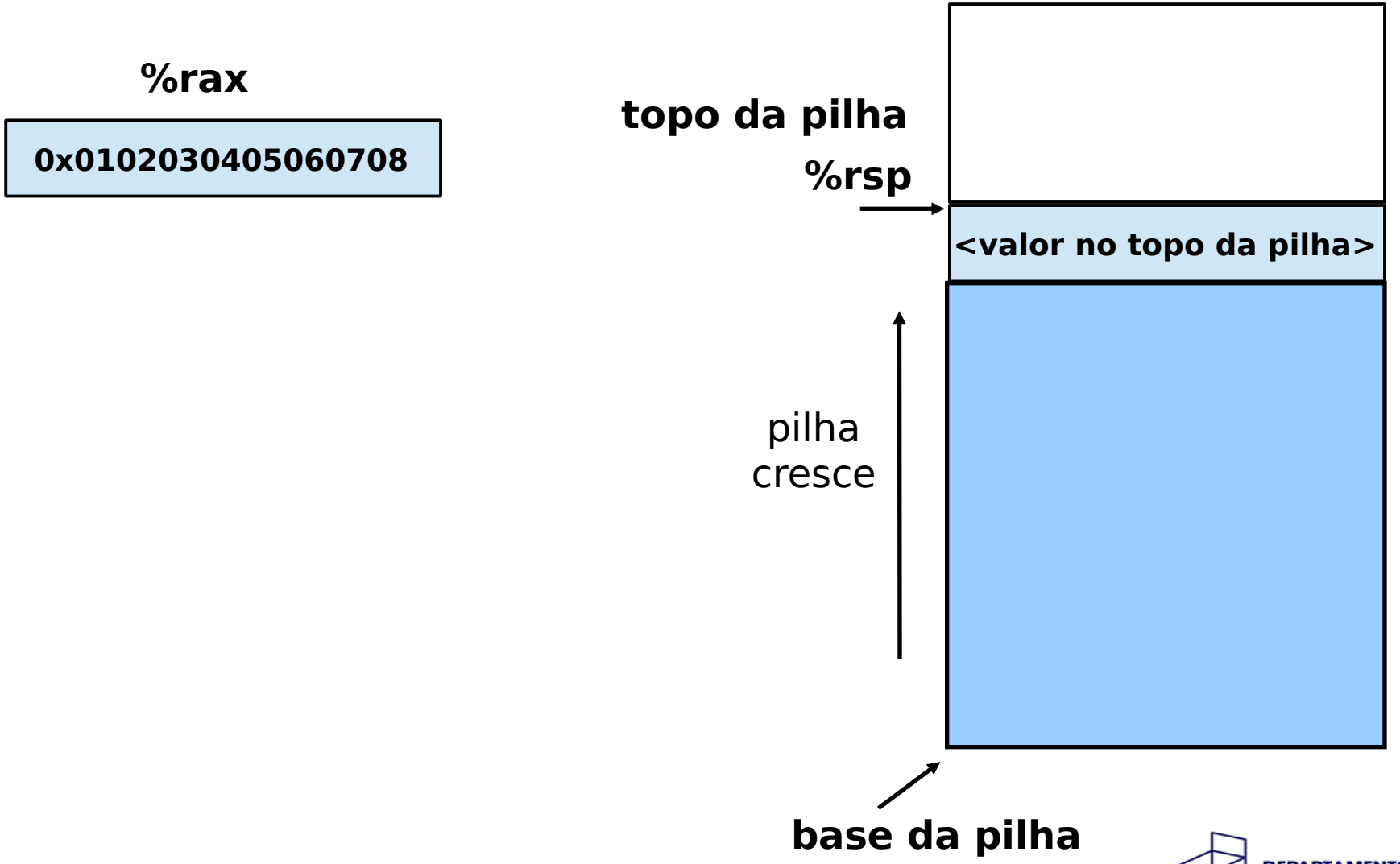
A pilha “cresce” em direção ao início da memória

A unidade de alocação é uma palavra (8 bytes)

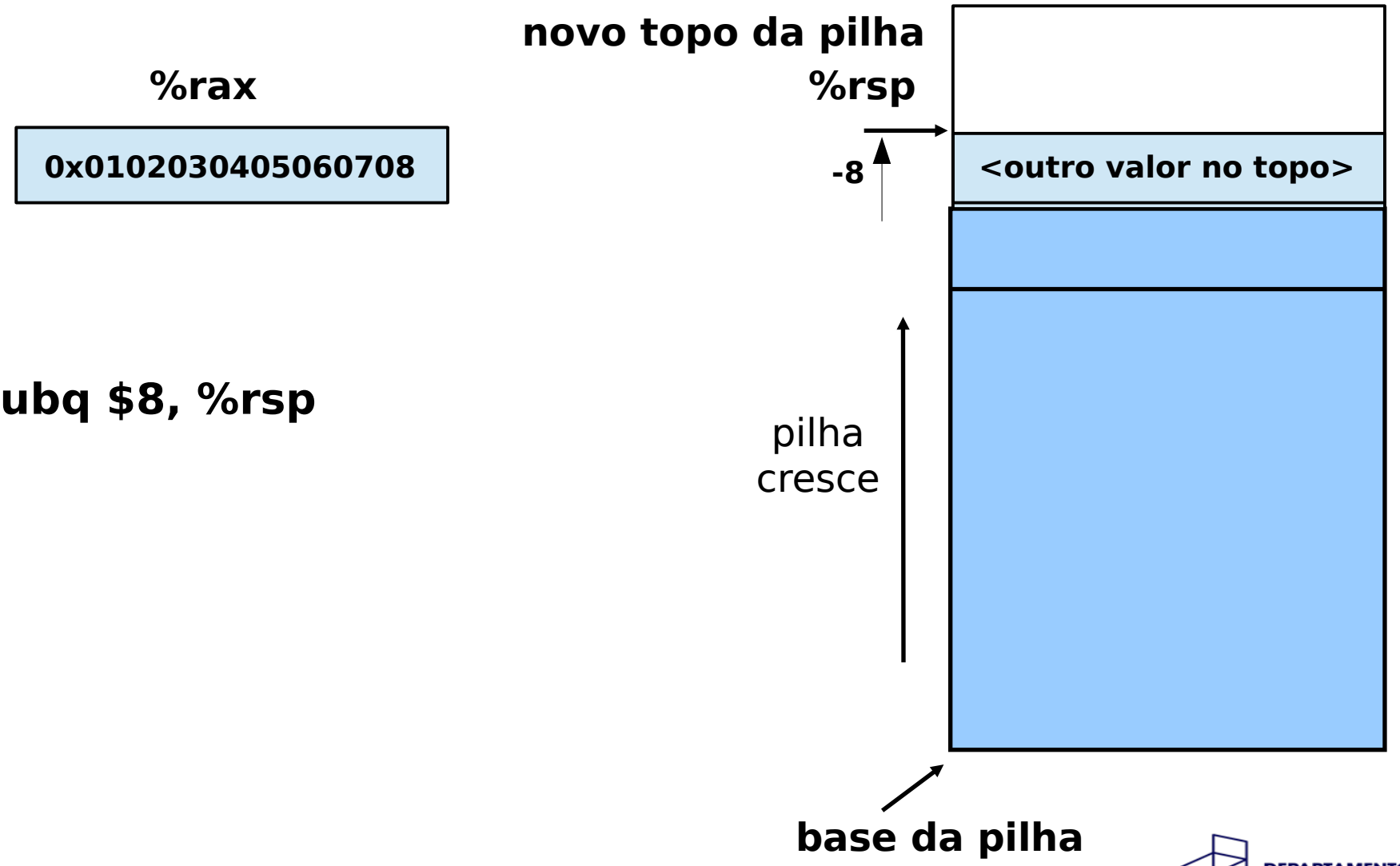
- para alocar espaço subtraimos 8 de %rsp (ou múltiplo de 8)
- para liberar espaço somamos 8 a %rsp (ou múltiplo de 8)



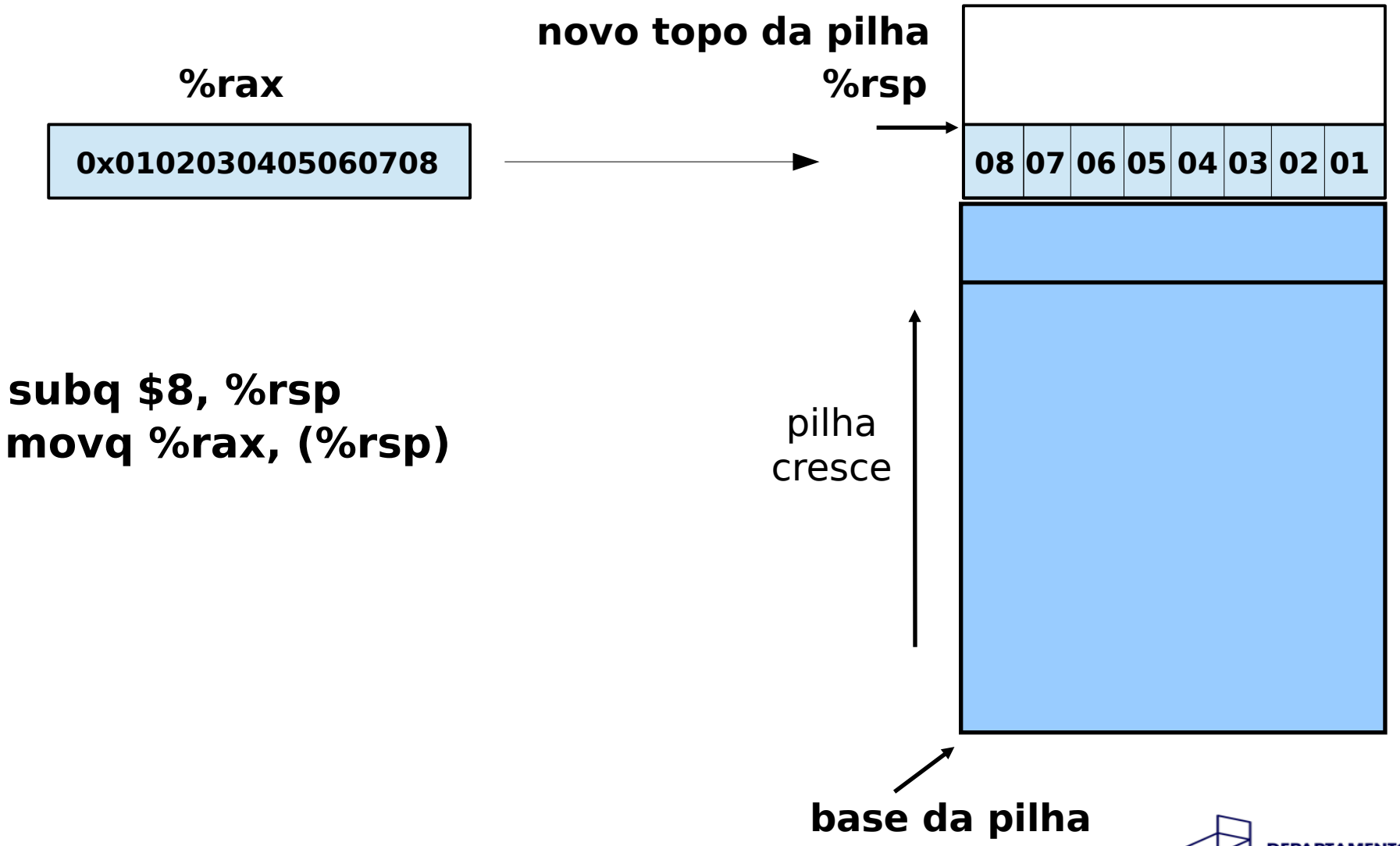
Empilhando um valor



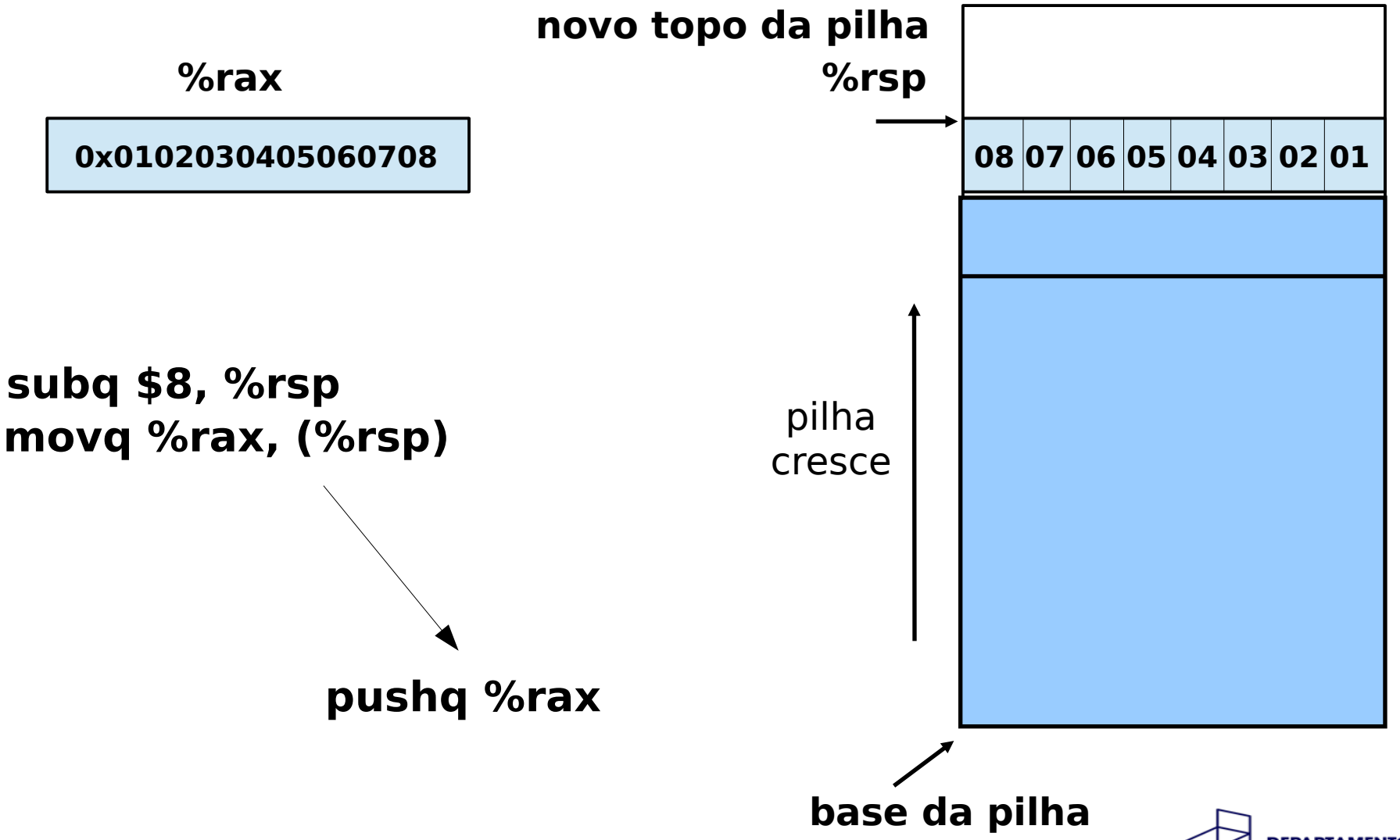
Empilhando um valor



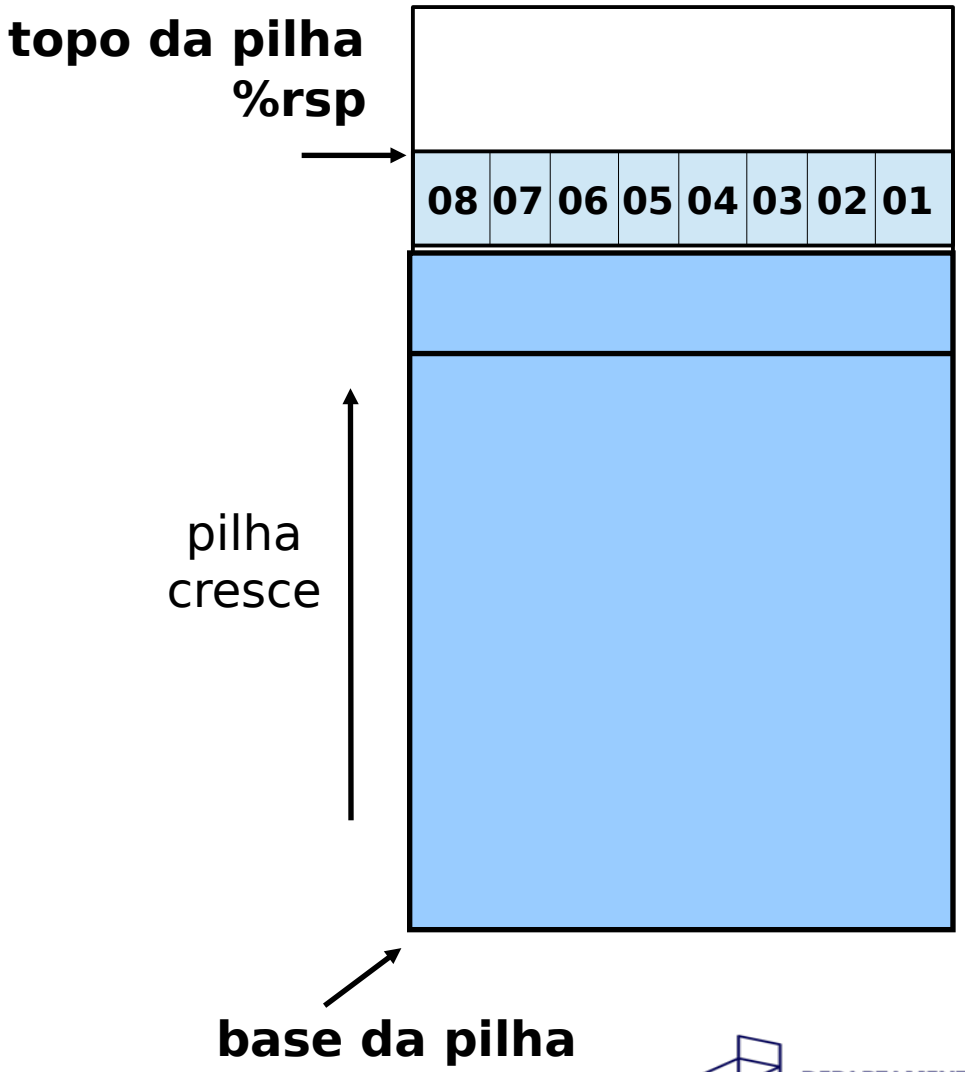
Empilhando um valor



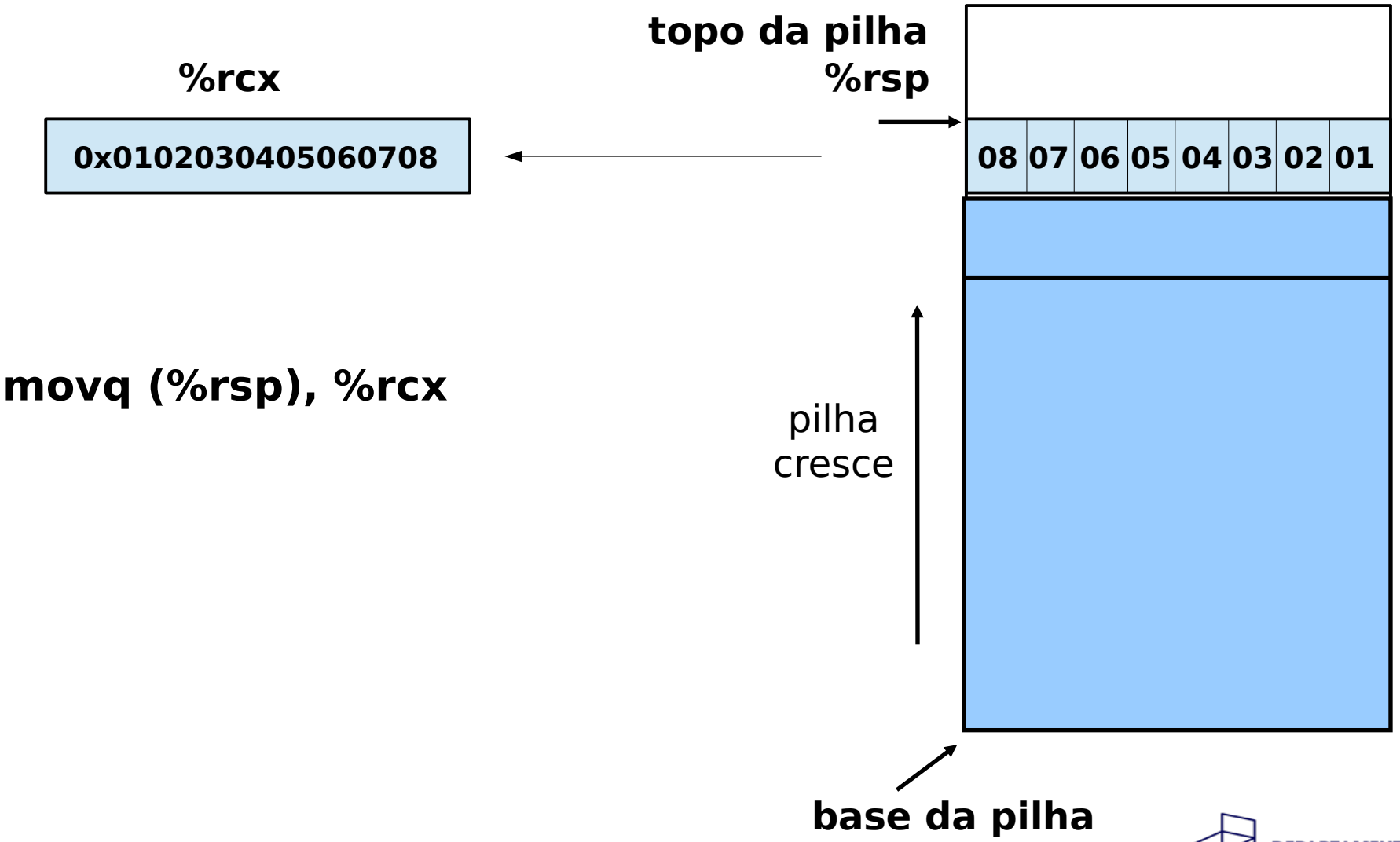
Empilhando um valor



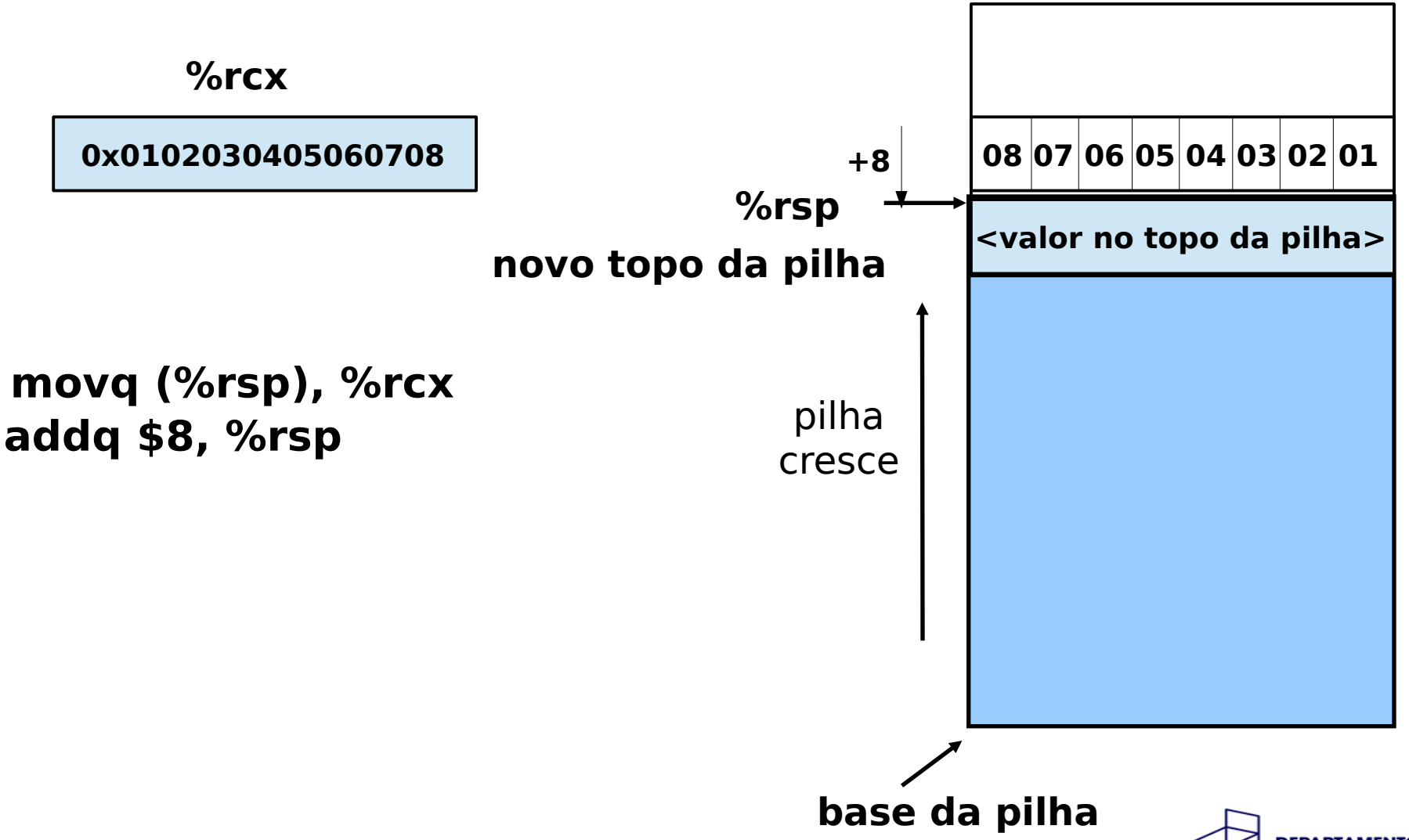
Desempilhando um valor



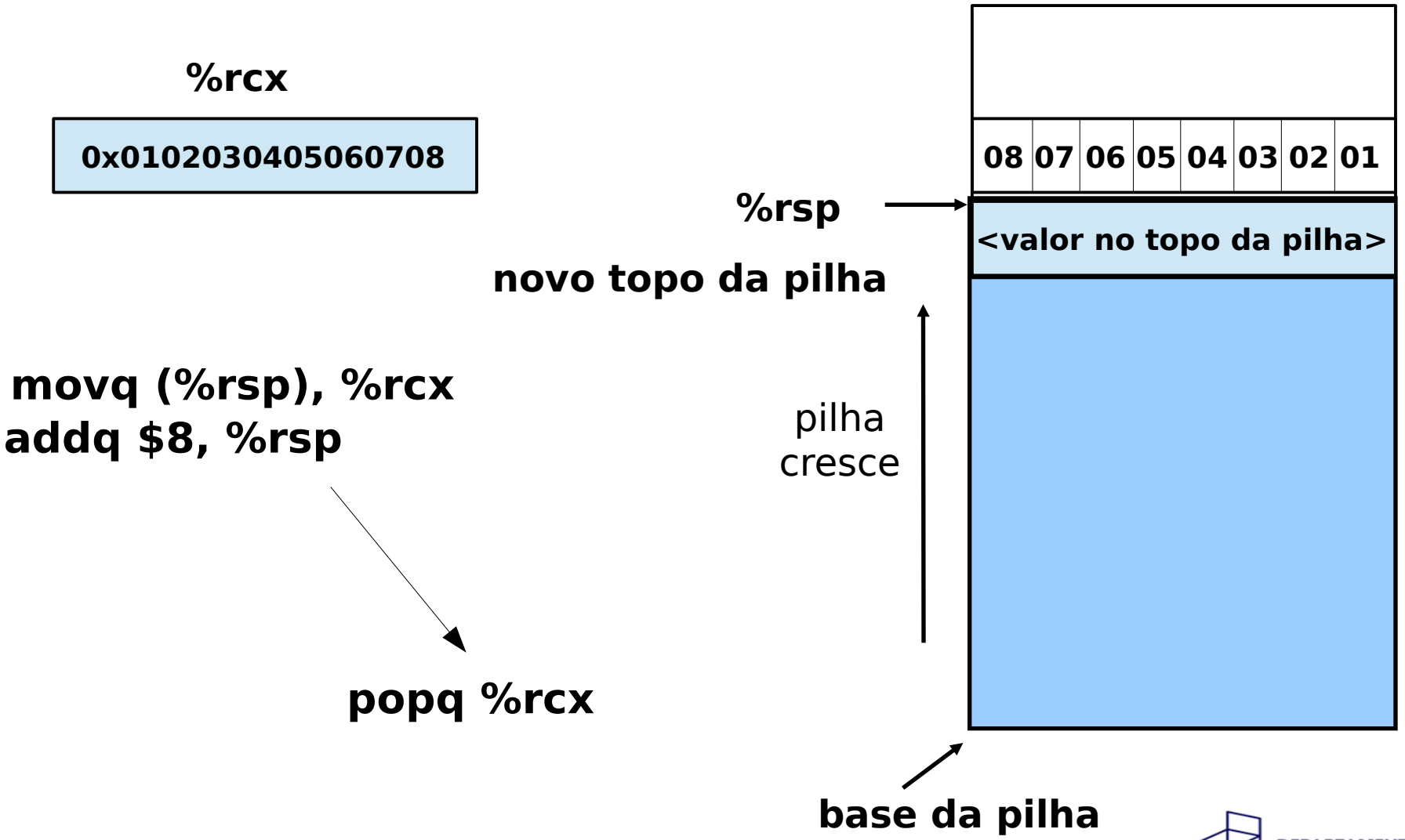
Desempilhando um valor



Desempilhando um valor



Desempilhando um valor



Procedimentos (funções)

Quando chamamos um procedimento, transferimos **dados & controle** de uma parte do código para outra

- chamador ↔ chamado
- parâmetros e valor de retorno

Procedimentos (funções)

Quando chamamos um procedimento, transferimos **dados & controle** de uma parte do código para outra

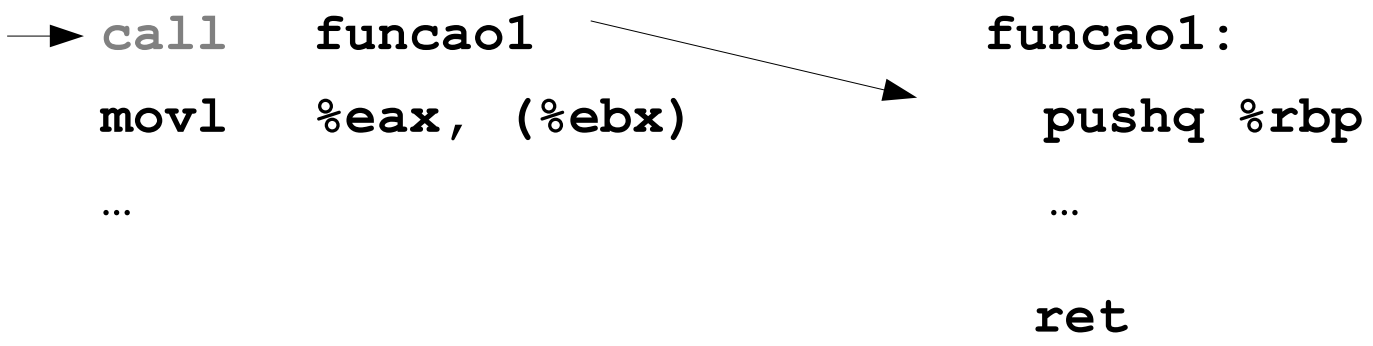
chamador ↔ chamado

parâmetros e valor de retorno

A maioria das arquiteturas provê suporte à implementação de procedimentos com base em **instruções de transferência de controle** e no uso de uma **pilha**

- “memória auxiliar” para armazenamento temporário

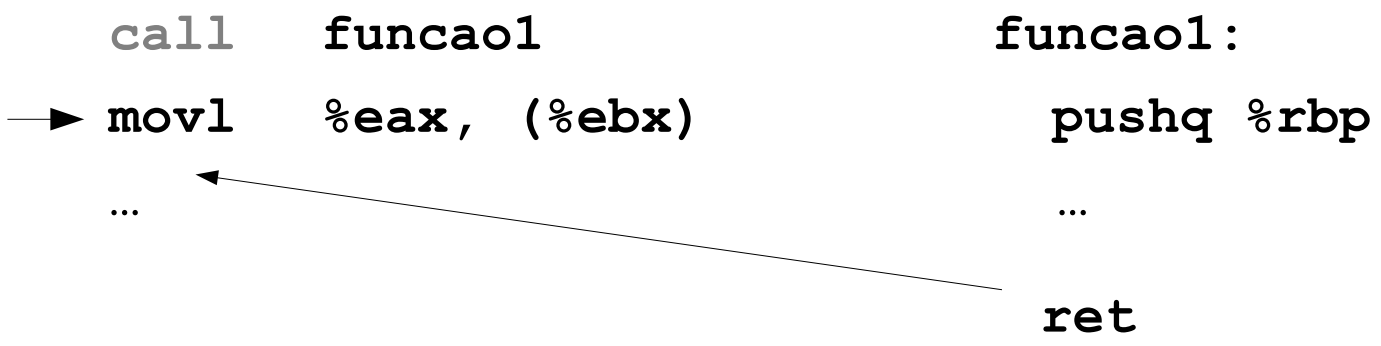
Transferência de Controle



A instrução **call** transfere o controle para a função

- instrução no endereço de memória indicado (associado ao label)

Transferência de Controle



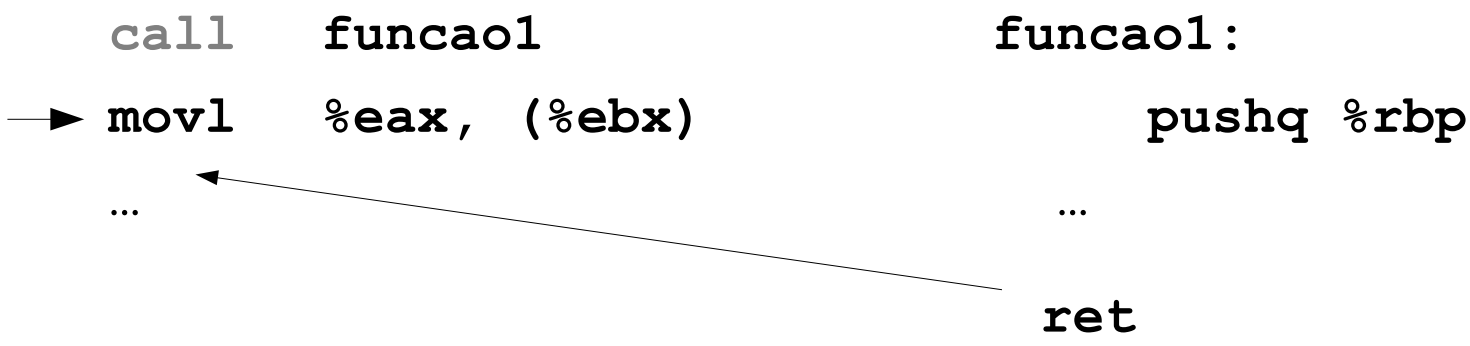
A instrução **call** transfere o controle para a função

- instrução no endereço de memória indicado (associado ao label)

A instrução **ret** transfere o controle para o chamador

- instrução seguinte ao call

Transferência de Controle



A instrução **call** transfere o controle para a função

- instrução no endereço de memória indicado (associado ao label)

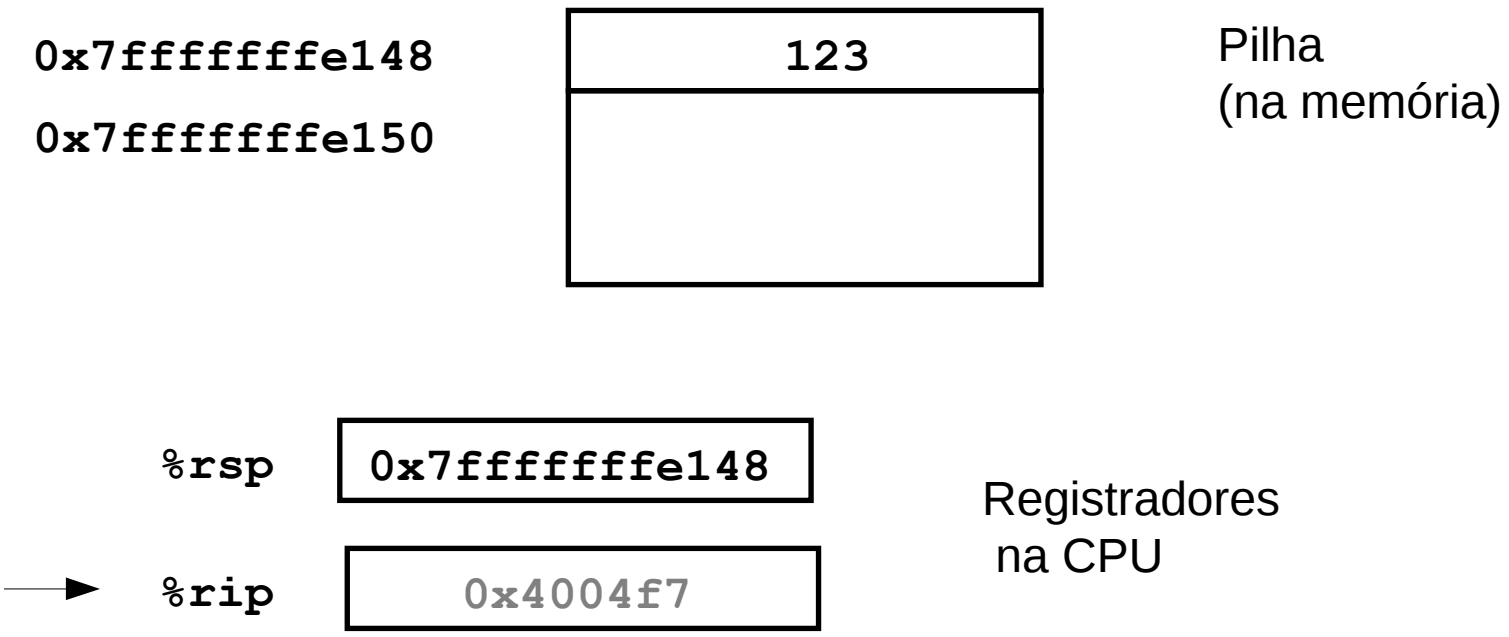
A instrução **ret** transfere o controle para o chamador

- instrução seguinte ao call

O endereço de retorno é armazenado (pelo hardware) na **pilha de execução**!

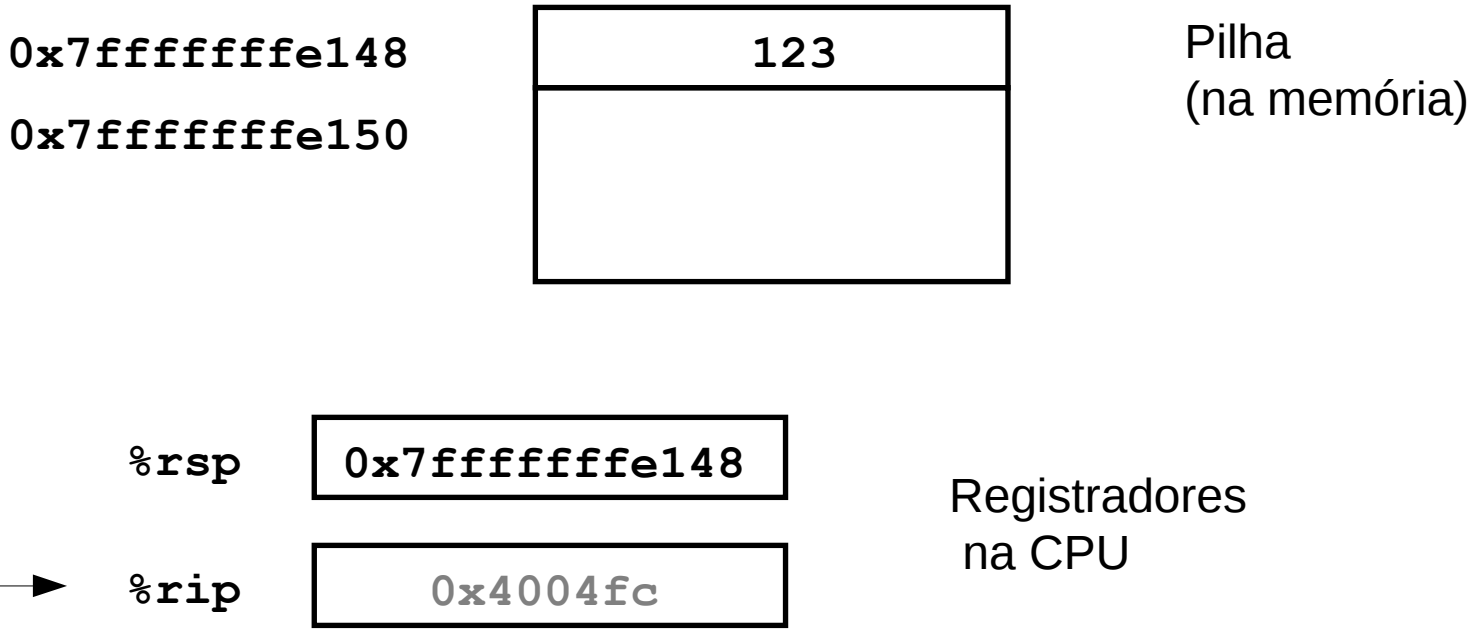
Exemplo de chamada

```
4004f7:   call    sum           /* sum em 4004ed */
4004fc:   movl    %eax, %ebx
```



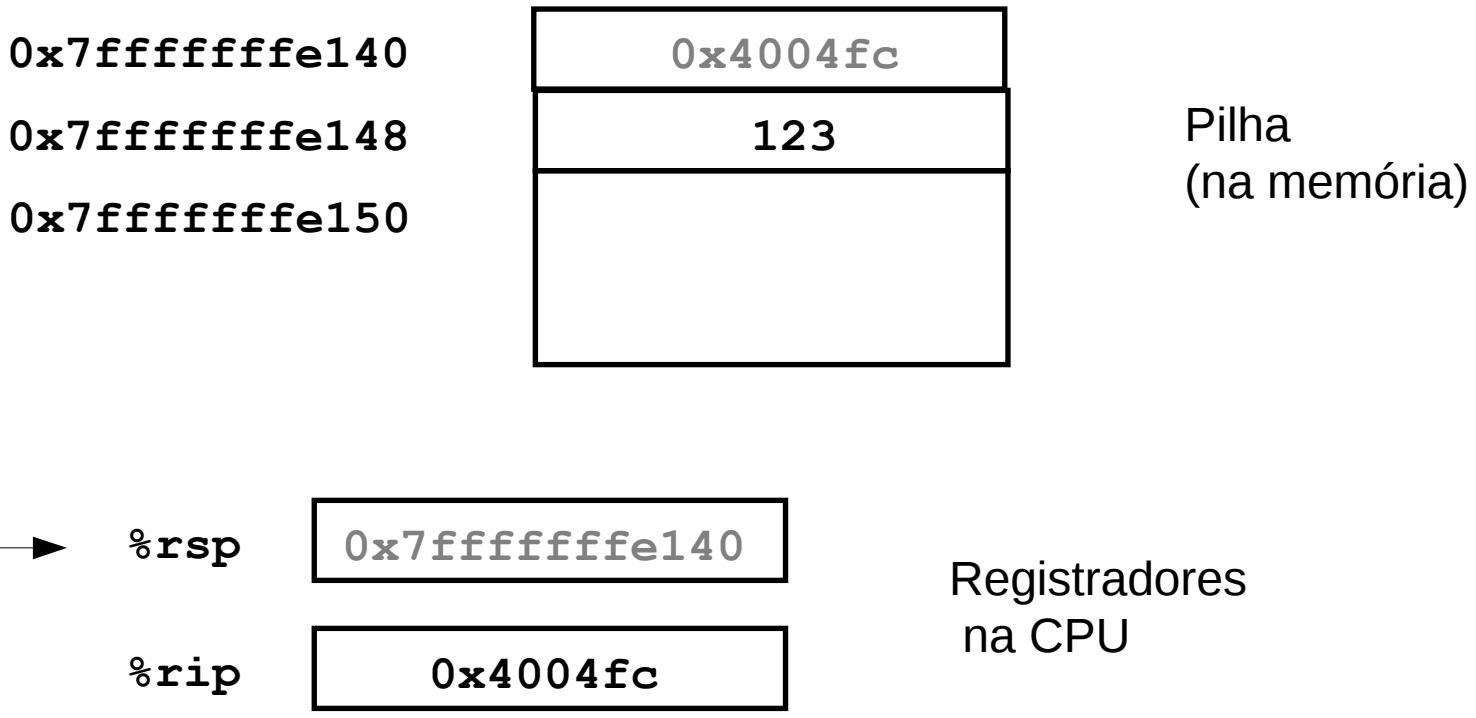
Exemplo de chamada

```
   4004f7:   call    sum           /* sum em 4004ed */
→  4004fc:   movl    %eax, %ebx
```



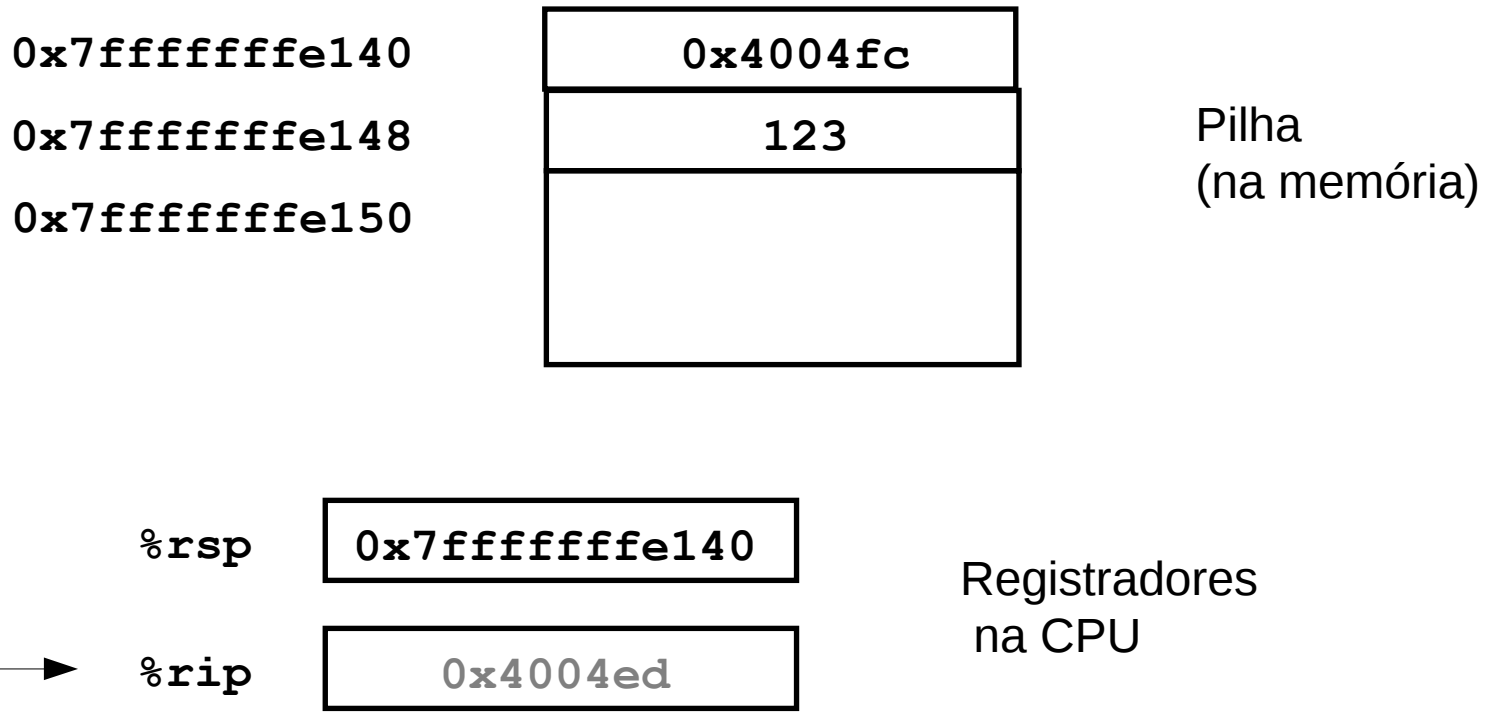
Exemplo de chamada

```
4004f7:   call    sum           /* sum em 4004ed */
4004fc:   movl    %eax, %ebx
```



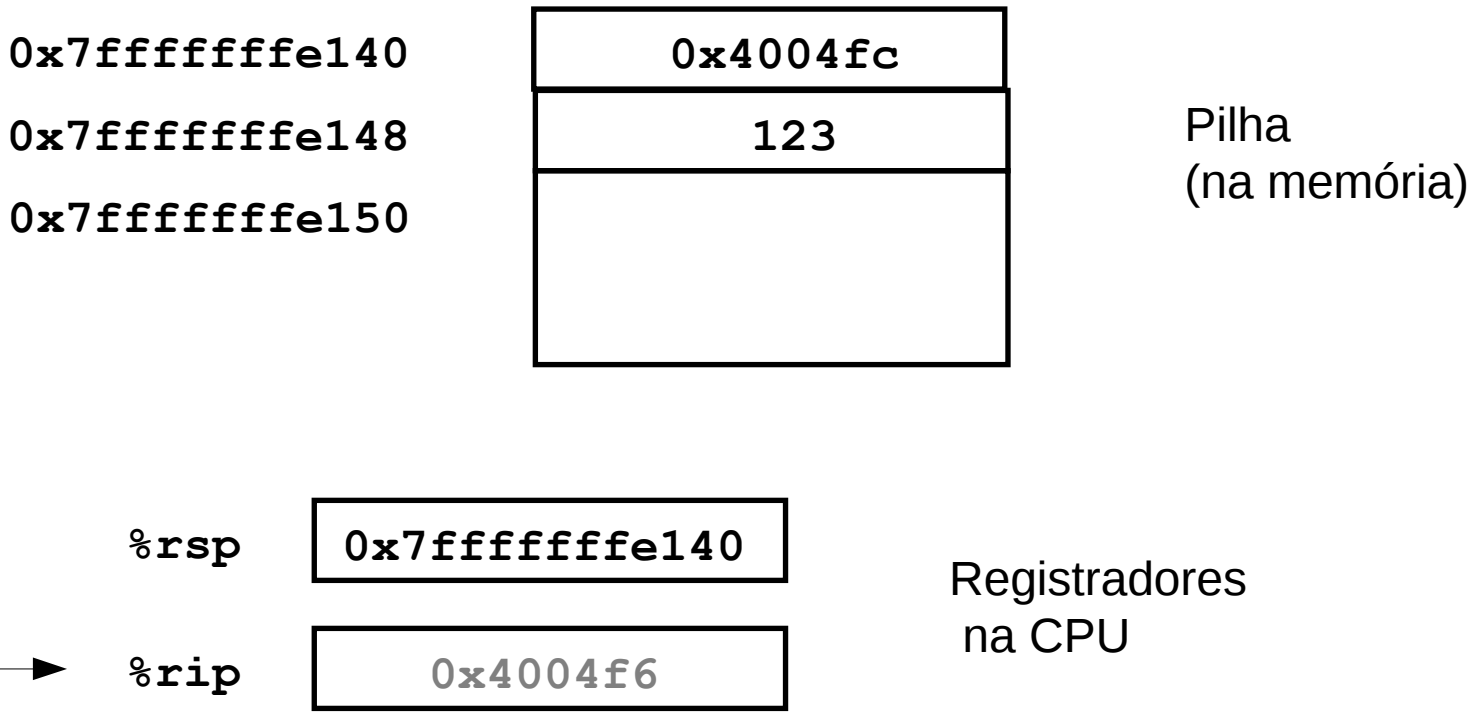
Exemplo de chamada

```
4004f7:   call    sum           /* sum em 4004ed */
4004fc:   movl    %eax, %ebx
```



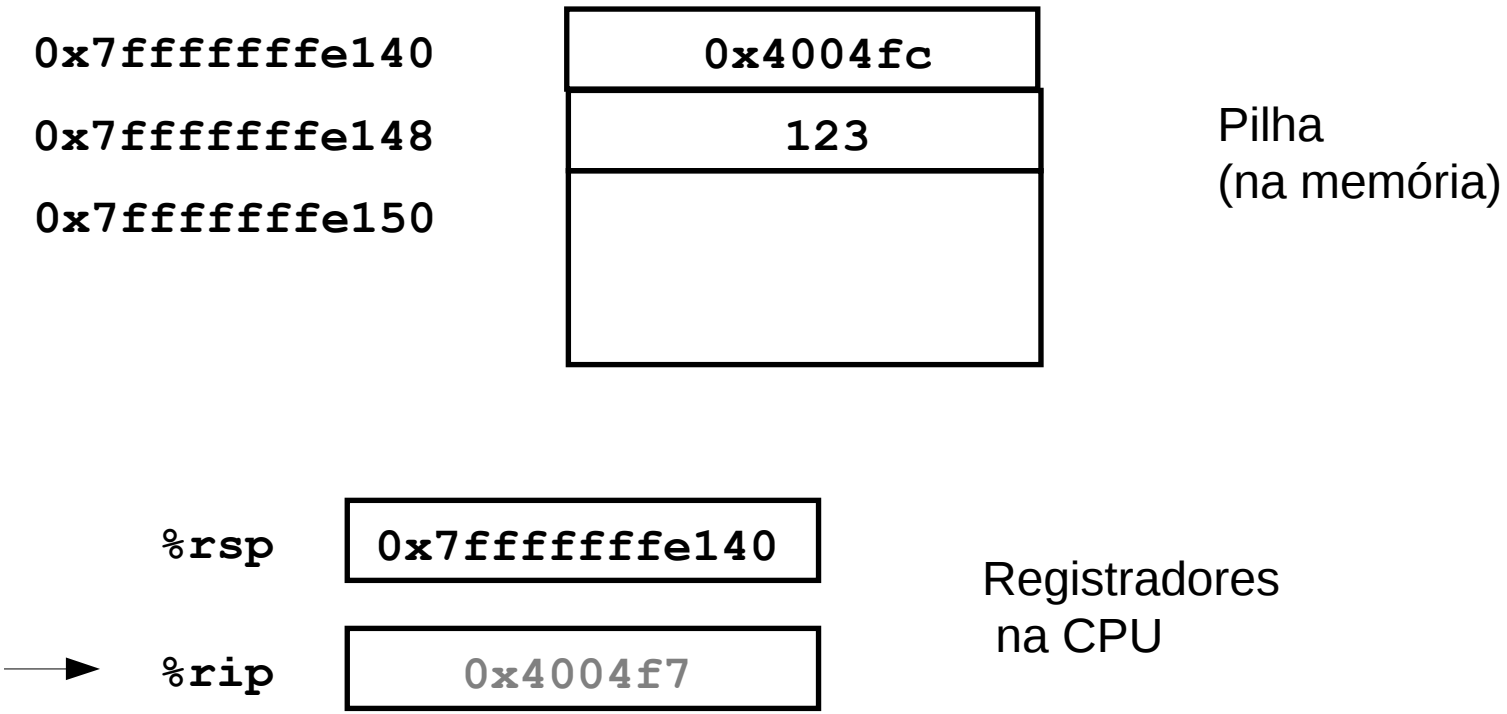
Exemplo de retorno

```
→ 4004f6:  ret
  4004f7:  ...
```



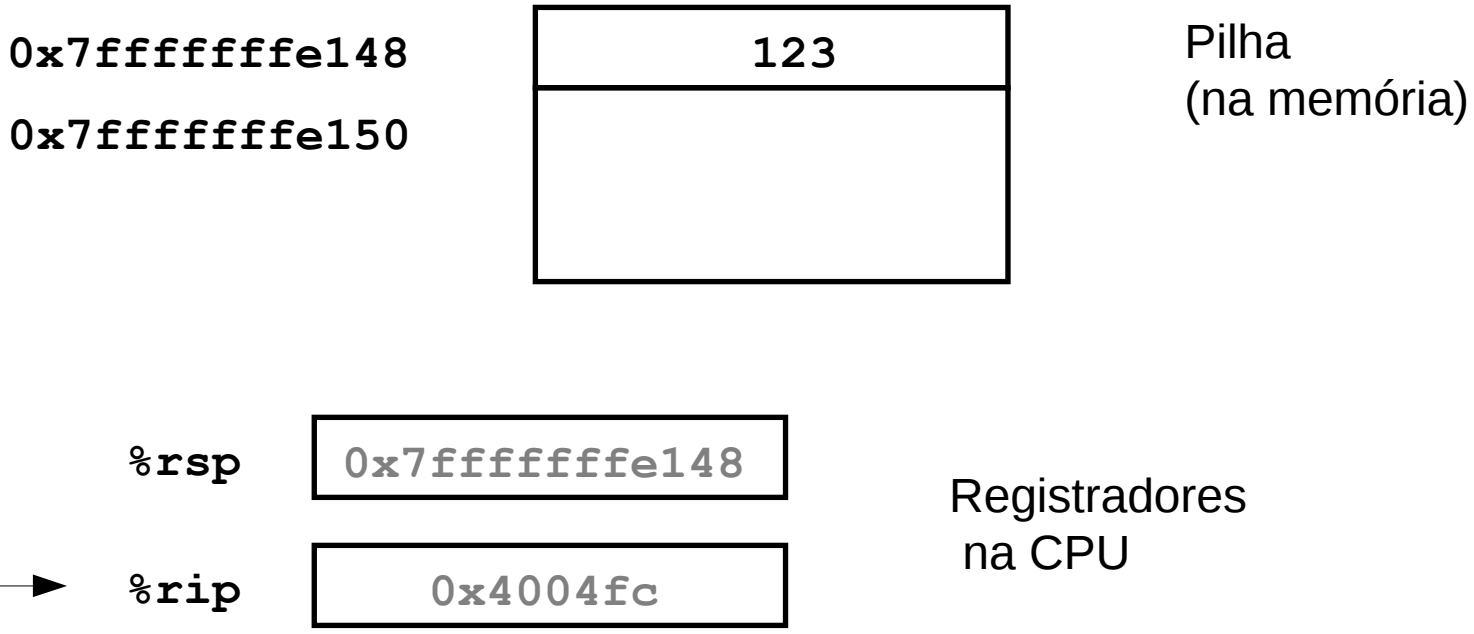
Exemplo de retorno

```
   4004f6:  ret
→  4004f7:  ...
```



Exemplo de retorno

```
4004f6:  ret
4004f7:  ...
```



Passagem de Parâmetros

Instruções **call** e **ret** só transferem controle

- **não cuidam da passagem de valores!**

Passagem de Parâmetros

Instruções **call** e **ret** só transferem controle

- **não cuidam da passagem de valores!**

A convenção para C na plataforma Linux x86-64 (ABI) estabelece a passagem de até 6 valores inteiros (incluindo ponteiros) via registradores

- parâmetros adicionais e estruturas passados na pilha (abaixo do endereço de retorno)

Passagem de Parâmetros

Instruções **call** e **ret** só transferem controle

- **não cuidam da passagem de valores!**

A convenção para C na plataforma Linux x86-64 (ABI) estabelece a passagem de até 6 valores inteiros (incluindo ponteiros) via registradores

- parâmetros adicionais e estruturas passados na pilha (abaixo do endereço de retorno)

Os registradores são usados numa ordem especificada:

→ %rdi, %rsi, %rdx, %rcx, %r8, %r9

Uso dos registradores para parâmetros

Num. do parâmetro	64 bits	32 bits	16 bits	8 bits
1	%rdi	%edi	%di	%dil
2	%rsi	%esi	%si	%sil
3	%rdx	%edx	%dx	%dl
4	%rcx	%ecx	%cx	%cl
5	%r8	%r8d	%r8w	%r8b
6	%r9	%r9d	%r9w	%r9b

Valor de Retorno

A convenção para C na plataforma Linux x86-64 (ABI) estabelece que um valor inteiro (incluindo ponteiro) é retornado no registrador %rax

- ou %eax, dependendo do tamanho

O retorno de estruturas tem tratamento especial

Exemplo de chamada

printf(“%d\n”, sum);

Exemplo de chamada

printf(“%d\n”, sum);

```
Sf:    .string    “%d\n”
```

Exemplo de chamada

printf(“%d\n”, sum);

```
    Sf:    .string    “%d\n”
    ...
    /* assumindo que o valor de sum está em %eax */
    movq  $Sf,    %rdi      /* primeiro parâmetro */
```

Exemplo de chamada

printf(“%d\n”, sum);

```
Sf:    .string    “%d\n”
...
/* assumindo que o valor de sum está em %eax */
movq  $Sf,    %rdi      /* primeiro parâmetro */
movl   %eax, %esi      /* segundo parâmetro */
```

Exemplo de chamada

printf(“%d\n”, sum);

```
    Sf:    .string    “%d\n”
    ...
    /* assumindo que o valor de sum está em %eax */
    movq  $Sf,    %rdi      /* primeiro parâmetro */
    movl   %eax, %esi      /* segundo parâmetro */
    movl   $0, %eax        /* caso especial para printf */
    call     printf
```