

Procedimentos Registro de Ativação

Noemi Rodriguez
Ana Lúcia de Moura

<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

variáveis locais

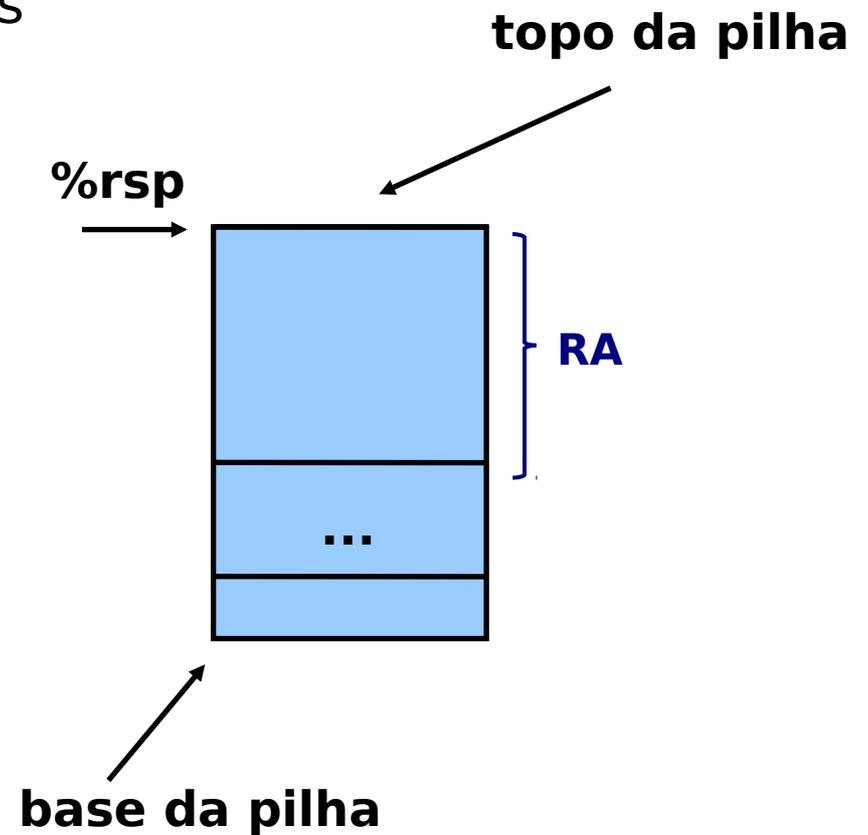
valores temporários

} **para cada chamada de função**

Registro de Ativação

Porção da pilha associada a uma chamada de função

- variáveis locais, valores temporários

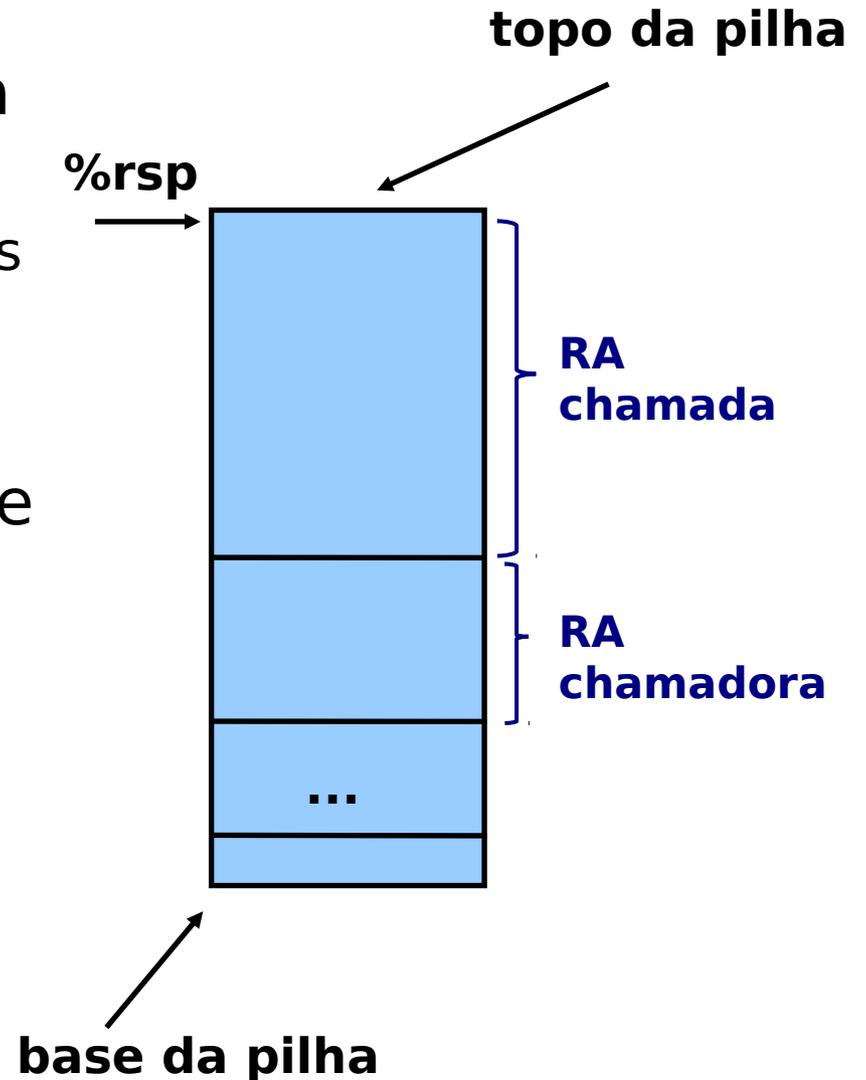


Registro de Ativação

Porção da pilha associada a uma chamada de função

- variáveis locais, valores temporários

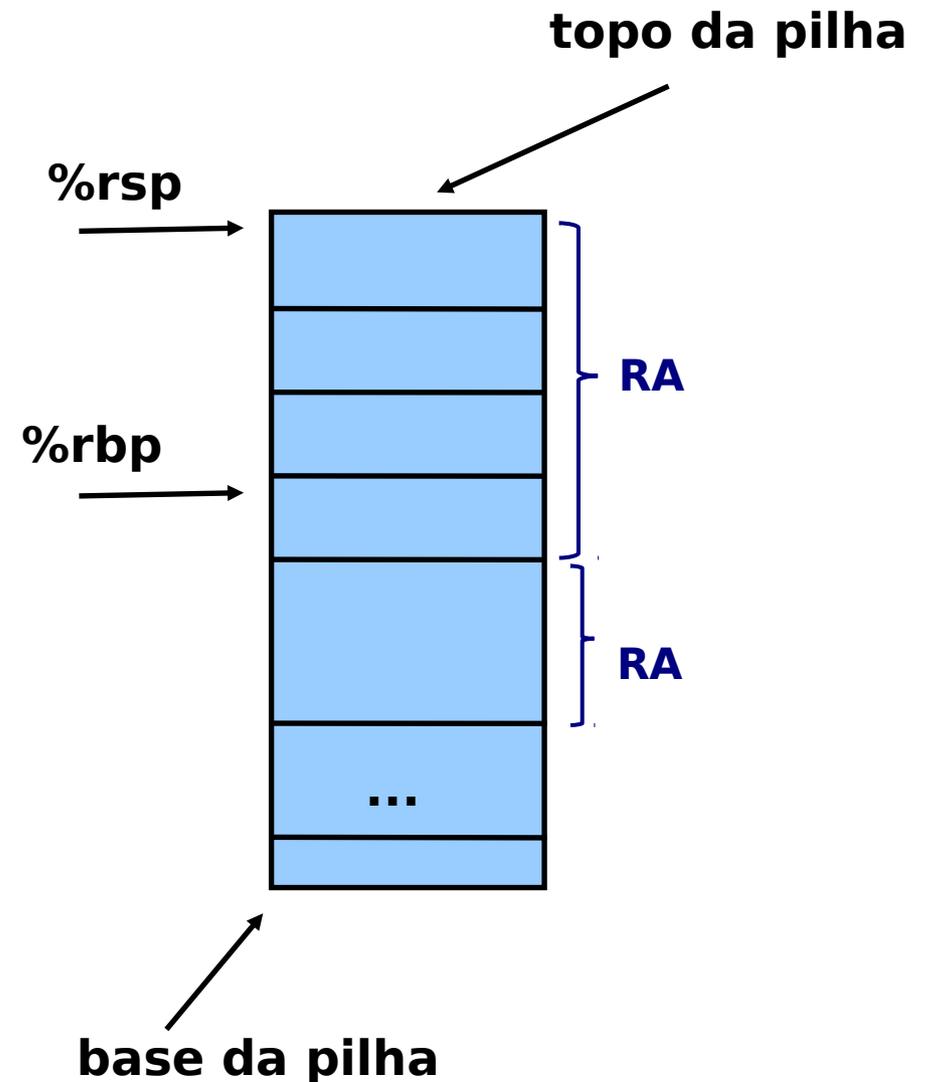
A pilha de execução é também chamada de pilha de registros de ativação



Acesso ao Registro de Ativação

O registrador **%rbp** é usado como **base do registro de ativação**

- acesso a elementos alocados no registro de ativação da função



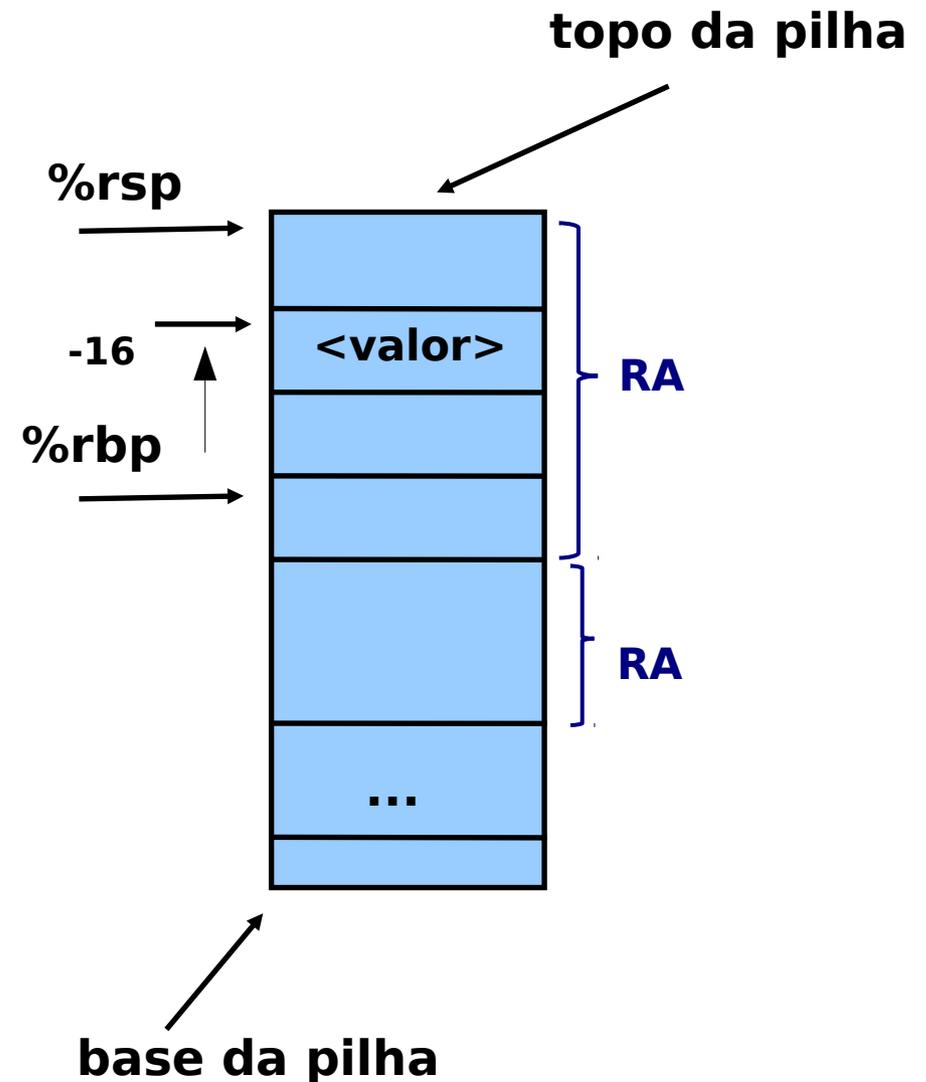
Acesso ao Registro de Ativação

O registrador **%rbp** é usado como **base do registro de ativação**

- acesso a elementos alocados no registro de ativação da função

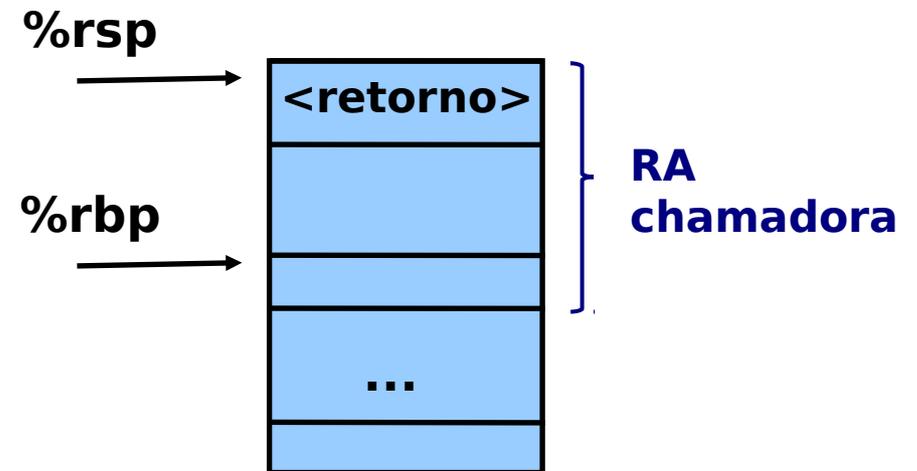
```
movq %r12, -16(%rbp)
```

```
movq -16(%rbp), %r12
```



Início do Procedimento

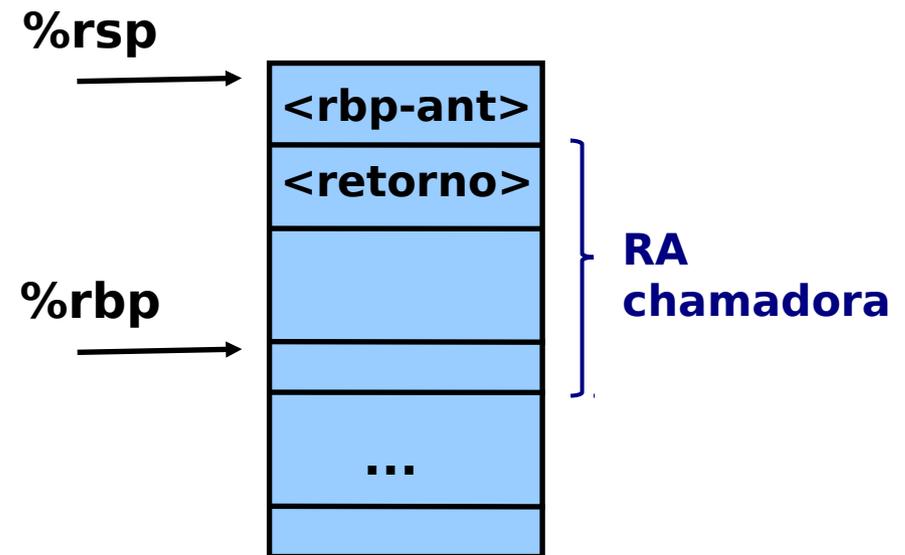
A função chamada deve preservar (guardar) a base do registro de ativação da função chamadora



Início do Procedimento

A função chamada deve preservar (guardar) a base do registro de ativação da função chamadora

```
pushq %rbp
```

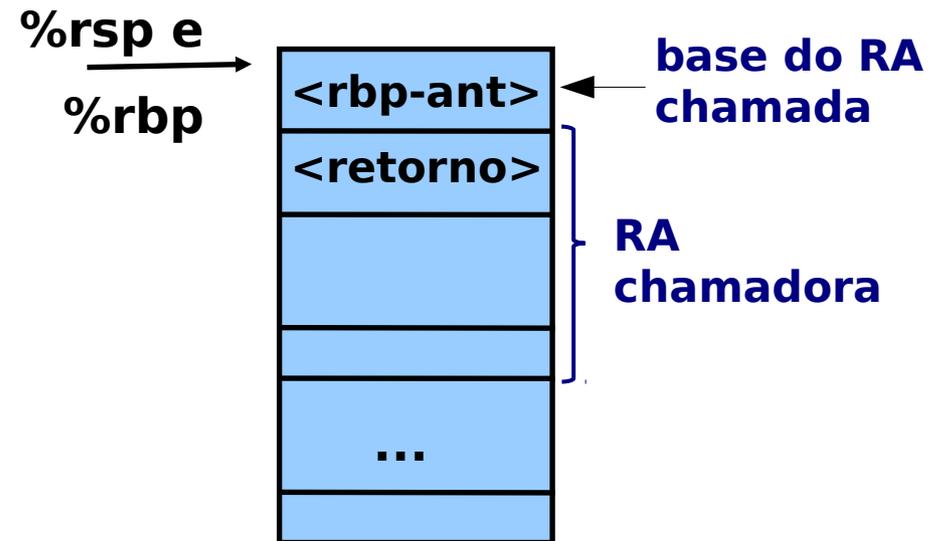


Início do Procedimento

A função chamada deve preservar (guardar) a base do registro de ativação da função chamadora

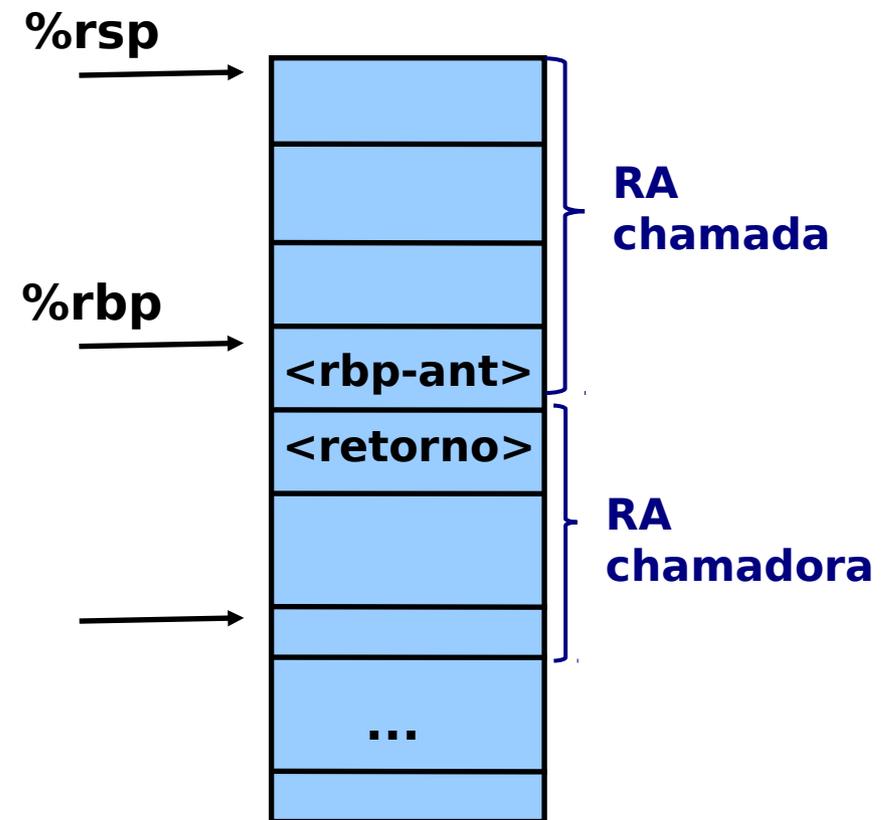
- e preparar o acesso ao seu próprio RA

```
pushq %rbp
movq  %rsp, %rbp
```



Fim do Procedimento

A função chamada deve liberar seu registro de ativação

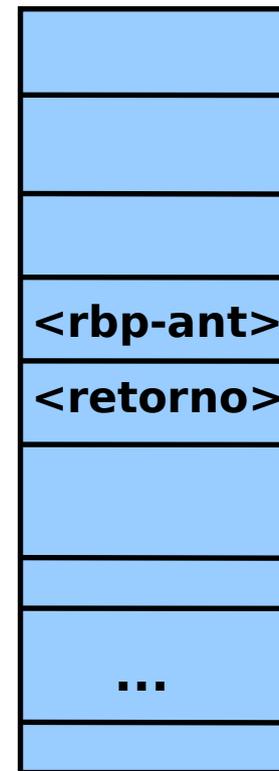


Fim do Procedimento

A função chamada deve liberar seu registro de ativação

```
movq %rbp, %rsp
```

%rsp e %rbp →



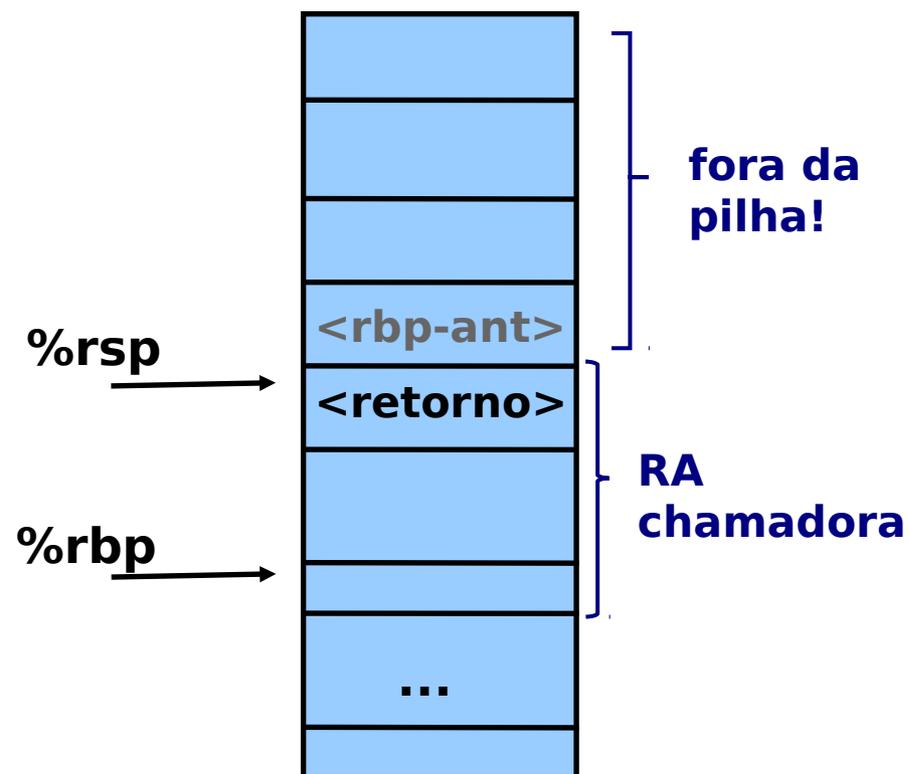
**RA
chamadora**

Fim do Procedimento

A função chamada deve liberar seu registro de ativação

- e recuperar a base do RA da função chamadora

```
movq %rbp, %rsp  
popq %rbp
```

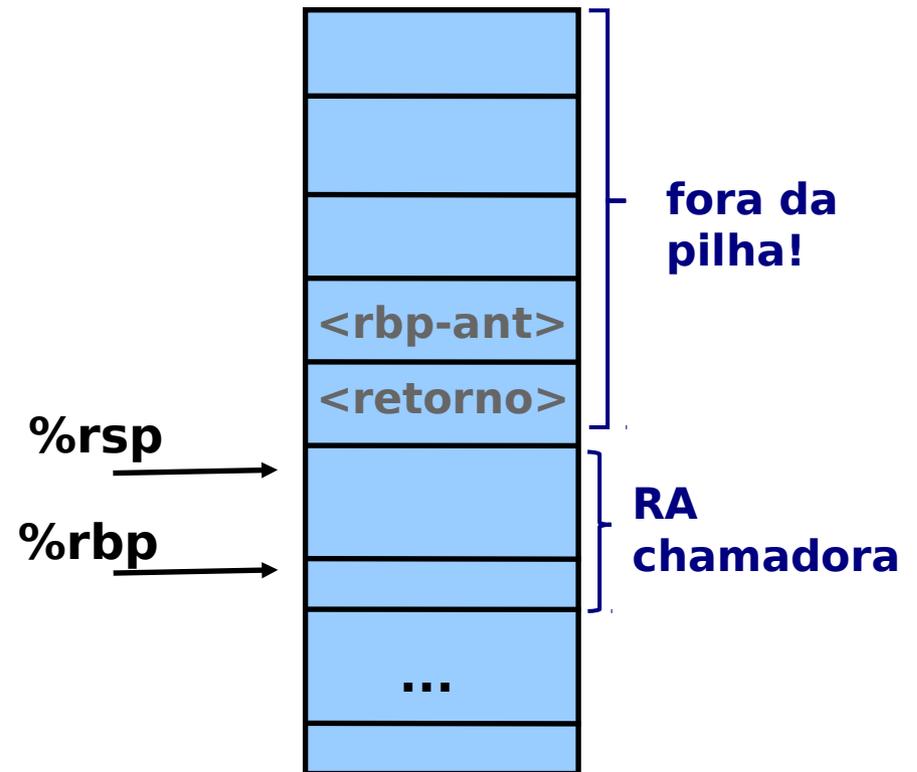


Fim do Procedimento

A função chamada deve liberar seu registro de ativação

- e recuperar a base do RA da função chamadora

```
movq %rbp, %rsp  
popq %rbp  
ret
```



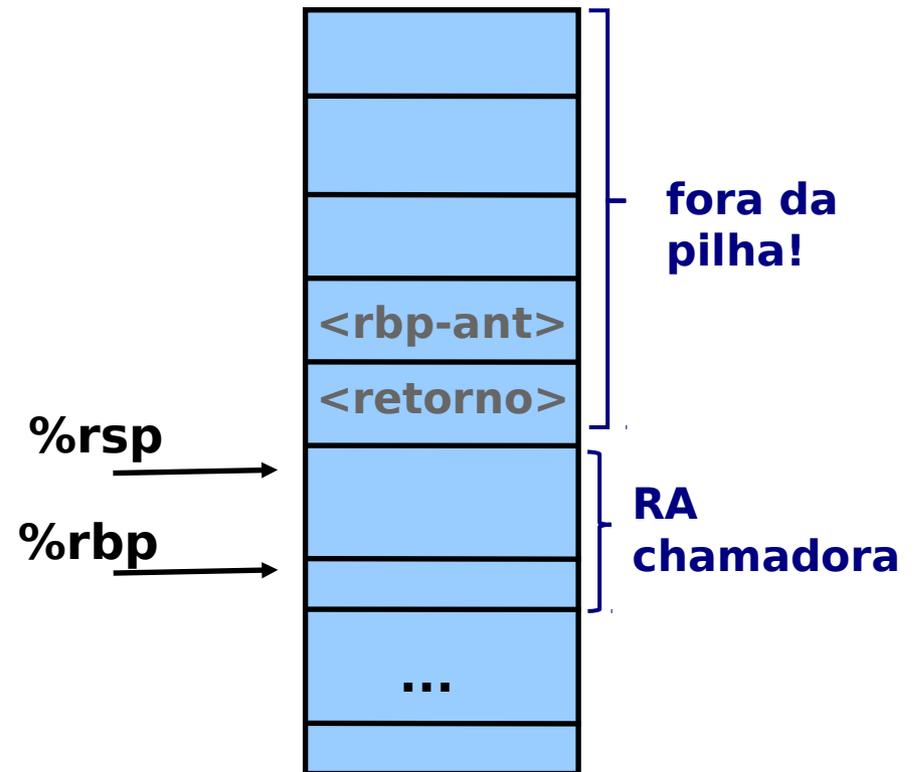
Fim do Procedimento

A função chamada deve liberar seu registro de ativação

- e recuperar a base do RA da função chamadora

```
movq %rbp, %rsp  
popq %rbp  
ret
```

leave



Alocando Espaço no RA

Para alocar espaço no RA subtraímos um múltiplo de 8 de %rsp

- unidade de alocação é uma palavra (8 bytes)

```
pushq %rbp
movq %rsp, %rbp
```

%rsp e %rbp →



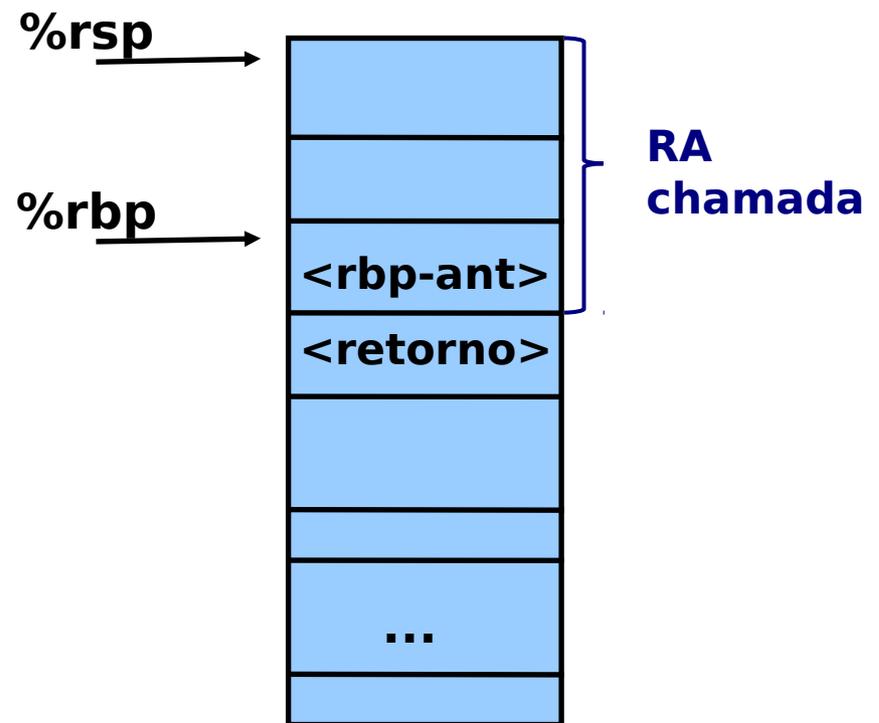
base do RA chamada

Alocando Espaço no RA

Para alocar espaço no RA subtraímos um múltiplo de 8 de `%rsp`

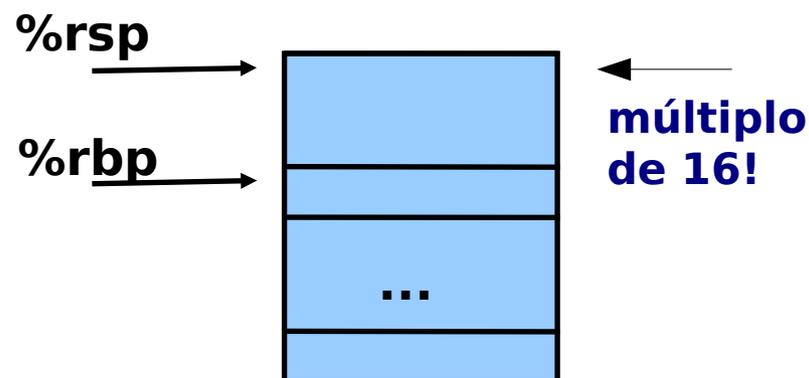
- unidade de alocação é uma palavra (8 bytes)

```
pushq %rbp
movq  %rsp, %rbp
subq  $16, %rsp
```



Alinhamento da Pilha

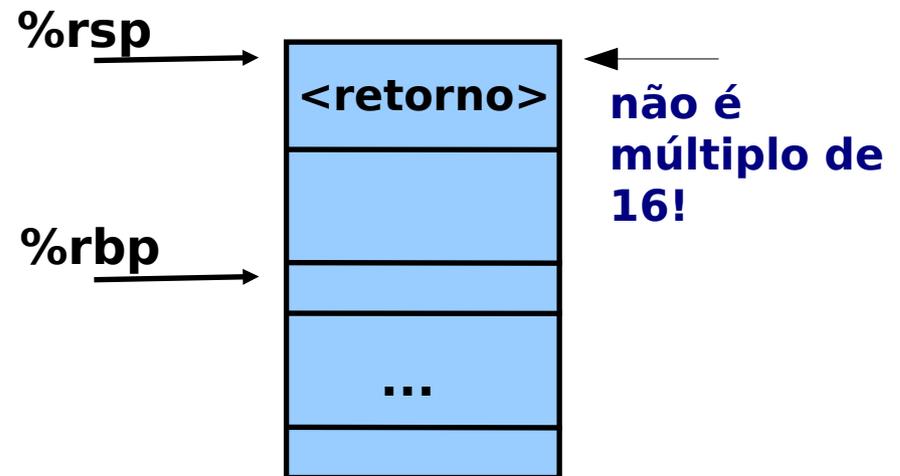
A ABI (convenção) Linux x86-64 estabelece que o topo da pilha (RA) deve estar alinhado num **endereço múltiplo de 16**



Alinhamento da Pilha

A ABI (convenção) Linux x86-64 estabelece que o topo da pilha (RA) deve estar alinhado num **endereço múltiplo de 16**

Logo após a execução do **call**, a pilha fica desalinhada

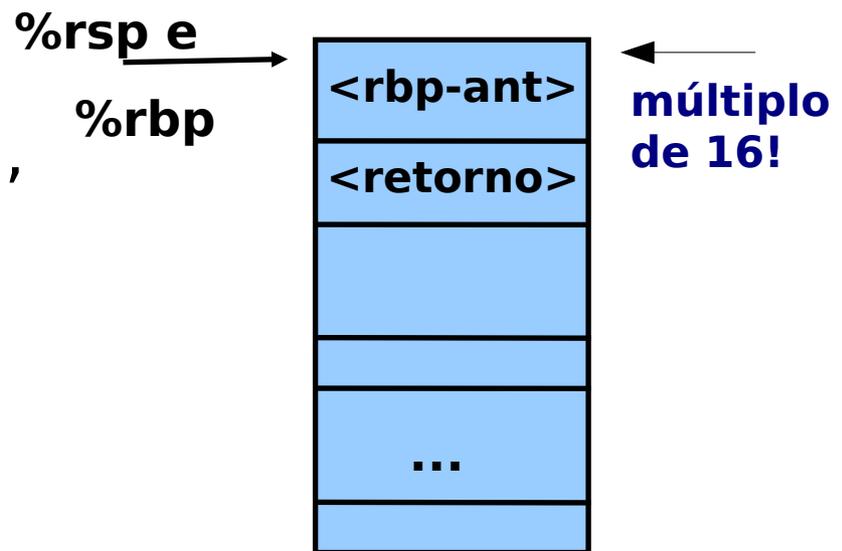


Alinhamento da Pilha

A ABI (convenção) Linux x86-64 estabelece que o topo da pilha (RA) deve estar alinhado num **endereço múltiplo de 16**

Logo após a execução do **call**, a pilha fica desalinhada

O salvamento do **%rbp** realinha a pilha



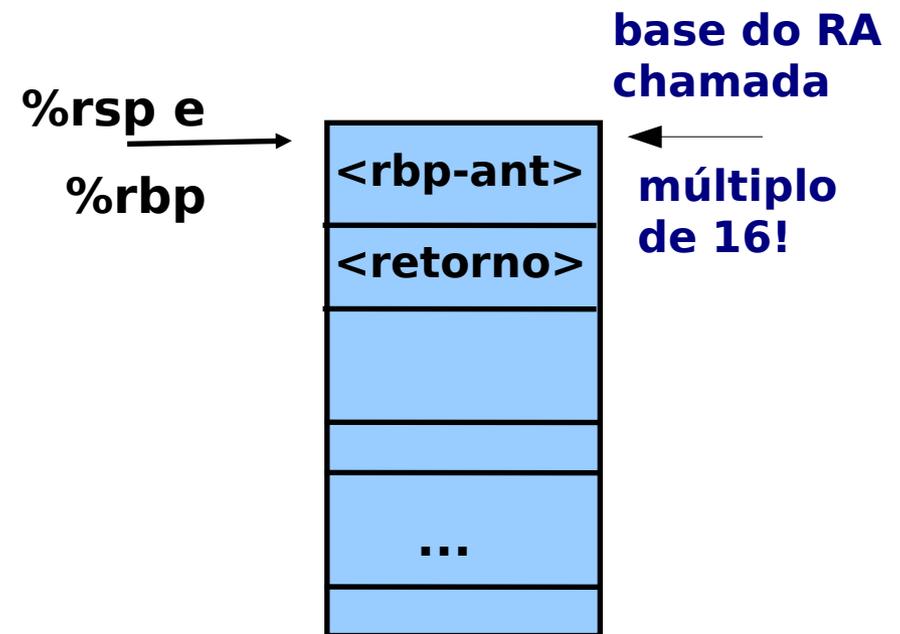
Alinhamento e Alocação

O compilador sabe calcular o tamanho do RA de uma função

- variáveis locais, salvamento de registradores, temporários

Para garantir o alinhamento, a alocação de espaço é feita no início da função

- e sempre é alocado um tamanho múltiplo de 16!



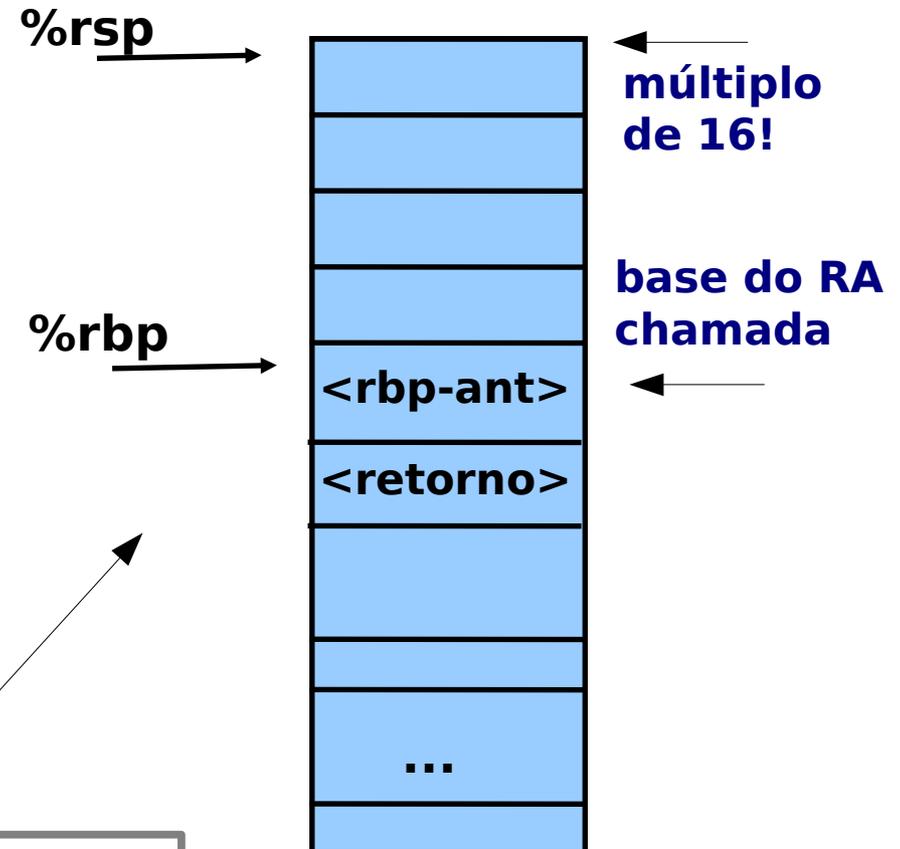
Alinhamento e Alocação

O compilador sabe calcular o tamanho do RA de uma função

- variáveis locais, salvamento de registradores, temporários

Para garantir o alinhamento, a alocação de espaço é feita no início da função

- e sempre é alocado um tamanho múltiplo de 16!



```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
```

Salvamento de Registradores

Funções usam registradores para armazenar valores

- variáveis locais, temporários, parâmetros

Alguns valores não podem ser perdidos ao se chamar uma outra função

- devem ser guardados na pilha (RA): chamadora? chamada?

Salvamento de Registradores

Funções usam registradores para armazenar valores

- variáveis locais, temporários, parâmetros

Alguns valores não devem ser perdidos ao se chamar uma outra função

- devem ser guardados na pilha (RA): chamadora? chamada?

A convenção C (ABI Linux x86-64) determina quais são os registradores **callee-saved** (salvos/guardados pela **função chamada**)

%rbx e %r12 a %r15

Exemplo de Salvamento (prólogo)

main:

```
/* prologo */
```

```
    pushq %rbp                /* guardando base RA chamadora */
```

```
    movq %rsp, %rbp          /* base do novo RA (desta função) */
```

```
    subq $16, %rsp           /* alocando espaço para o RA */
```

```
    movq %rbx, -8(%rbp)      /* guardando valor do %rbx */
```

```
    movq %r12, -16(%rbp)    /* guardando valor do %r12 */
```

Exemplo de Salvamento (epílogo)

```
/* finalizacao */
```

```
movq -8(%rbp), %rbx    /* recupera valor do %rbx */
```

```
movq -16(%rbp), %r12  /* recupera valor do %r12 */
```

```
movq %rbp, %rsp
```

```
popq %rbp              /* recupera valor do %rbp */
```

```
ret
```

Lembrando: Parâmetros e Retorno

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

%rax: valor de retorno (inteiro ou ponteiro)

Exemplo de Chamada

```
printf("%d\n", filtro(*p, LIM);
```

```
movl (%r12), %edi    /* primeiro parâmetro */
```

```
movl $1, %esi       /* segundo parâmetro */
```

```
call filtro
```

```
movq $s1, %rdi     /* primeiro parâmetro */
```

```
movl %eax, %esi    /* segundo parâmetro */
```

```
movl $0, %eax      /* apenas para printf */
```

```
call printf
```

Traduzindo a Função filtro

```
int filtro(int x, int lim) {  
    if (x < lim) return 0;  
    else return x;  
}
```

filtro:

```
/* não precisamos da pilha */
```

```
cmpl %esi, %edi /* x >= lim ? */
```

```
jge else
```

```
movl $0, %edi
```

else:

```
movl %edi, %eax
```

```
ret
```

Exemplo com uso da pilha

```
int g(int x);  
int f(int i, int n) {  
    int a = 0;  
    while (n-- > 0) {  
        a += g(i);  
        i *= 2;  
    }  
    return a;  
}
```

parâmetros:

i -> %edi

n -> %esi

sugestão de uso de
registradores:

a -> %r12d

i -> %r13d

n -> %r14d