Software Básico

Procedimentos
Registro de Ativação

INSTITUTO DE INFORMÁTICA - UFG

Reconhecimento

- Material produzido por:
  - Noemi Rodriguez – PUC-Rio
  - Ana Lúcia de Moura – PUC-Rio
- Adaptação
  - Bruno Silvestre – UFG

Memória

- Durante a execução de um programa, o SO precisa alocar memória principal para:
  - Dados globais
  - Código

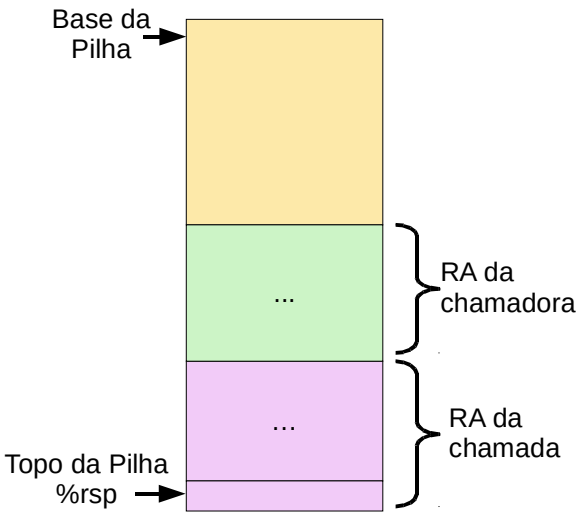
  } Tamanho (fixo) conhecido na compilação

- Variáveis locais
- Valores intermediários

} Para cada chamada de função

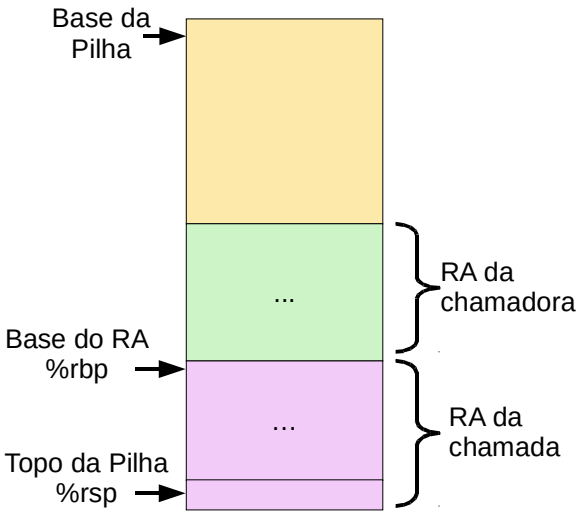
Registro de Ativação (RA)

- RA é <u>porção</u> da pilha associada a cada chamada de função
  - Variáveis locais
  - Valores temporários
- A pilha de execução é também chamada de pilha de registros de ativação



Acessando o Registro de Ativação

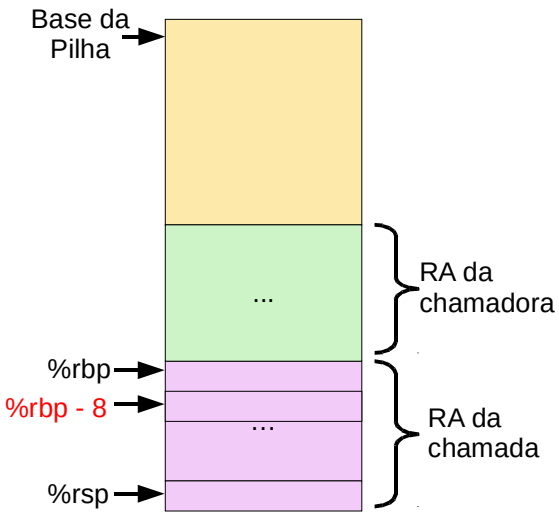
- O registrador **%rbp** é usado como a base do registro de ativação atual
  - Aponta para o endereço de início do RA
- %rbp é usado para acessar os elementos alocados no RA da função



Acessando o Registro de Ativação

- O registrador **%rbp** é usado como a base do registro de ativação atual
- %rbp é usado para acessar os elementos alocados no RA da função

```
movq %r12, -8(%rbp)

movq -8(%rbp), %r12
```

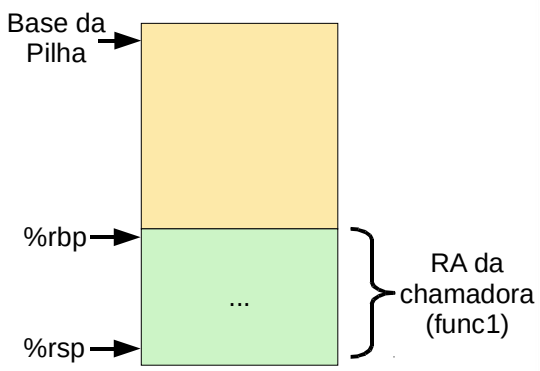


Registro de Ativação:
Início e Término de Procedimento

Início de um Procedimento

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

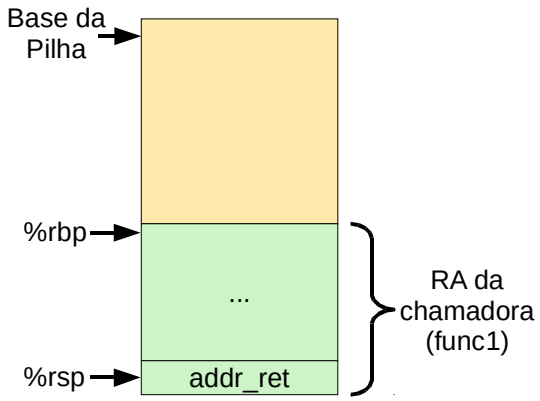
```
func1:
  …
  call   func2
  …

func2:
  …
```



Início de um Procedimento

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

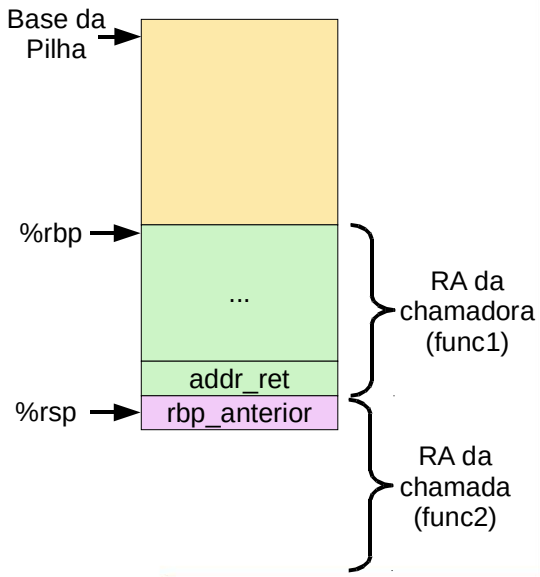
```
func1:
  …
  call   func2
  …

func2:
  …
```



Início de um Procedimento

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

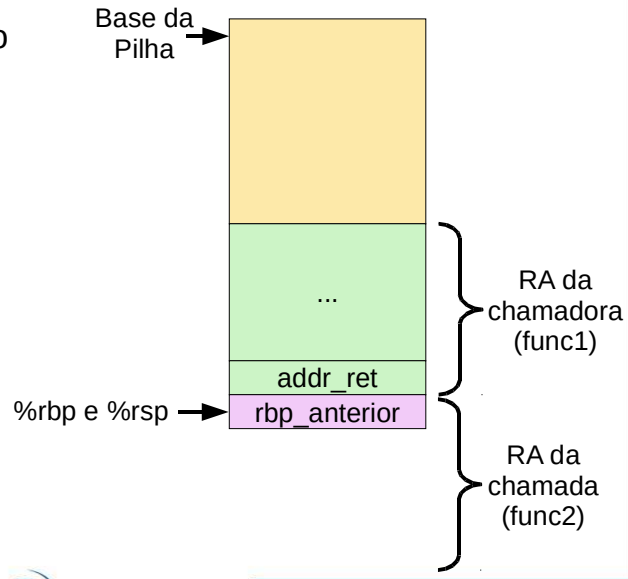
```
func1:
  …
  call   func2
  …

func2:
  pushq %rbp
```



Início de um Procedimento

- Além disso, a função chamada deve preparar um novo registro de ativação

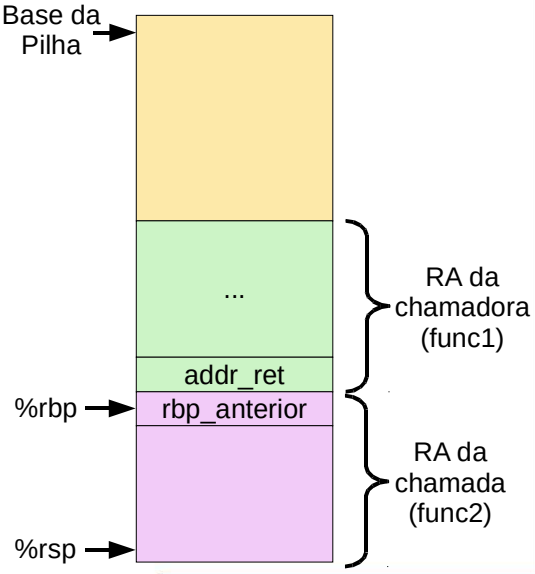
```
func1:
  …
  call   func2
  …

func2:
  pushq %rbp
  movq   %rsp, %rbp
```



Término de um Procedimento

- A função chamada deve liberar o seu registro de ativação

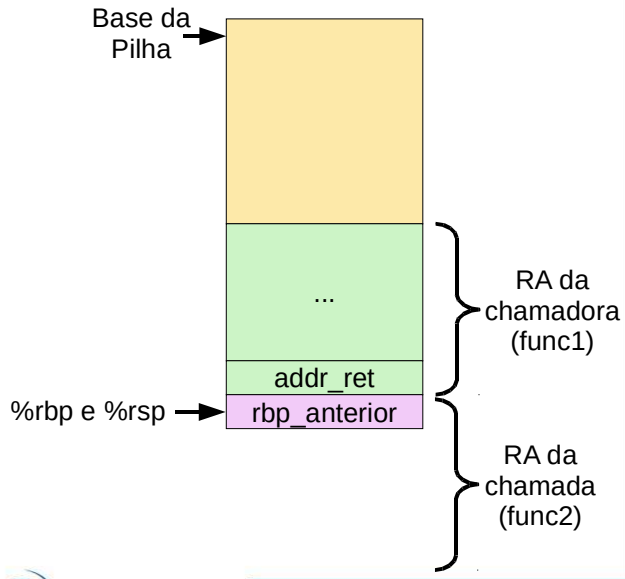
```
func2:
  pushq %rbp
  movq   %rsp, %rbp
  …
```



Término de um Procedimento

- A função chamada deve liberar o seu registro de ativação

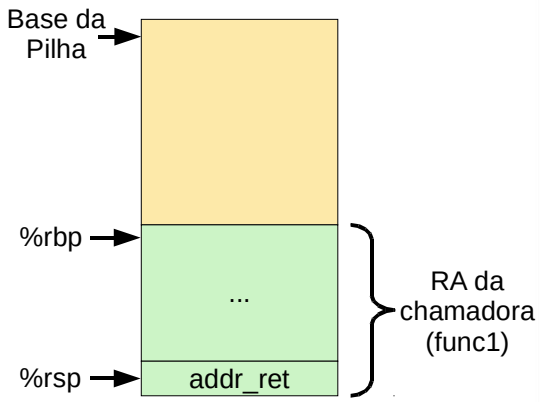
```
func2:
  pushq %rbp
  movq  %rsp, %rbp
  …
  movq %rbp, %rsp
```



Término de um Procedimento

- Além disso, a função chamada deve recuperar a base do registro de ativação da função chamadora

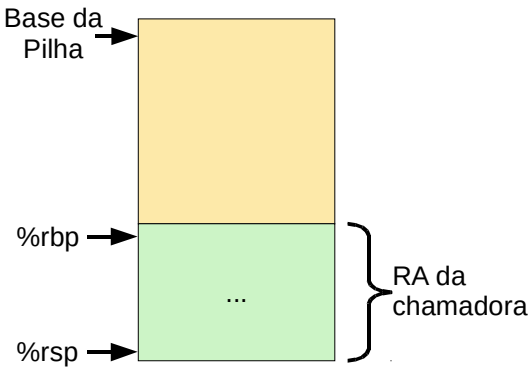
```
func2:
  pushq %rbp
  movq  %rsp, %rbp
  …
  movq %rbp, %rsp
  popq %rbp
```



Término de um Procedimento

- Finalmente, a função chamada pode retornar o controle para a função chamadora

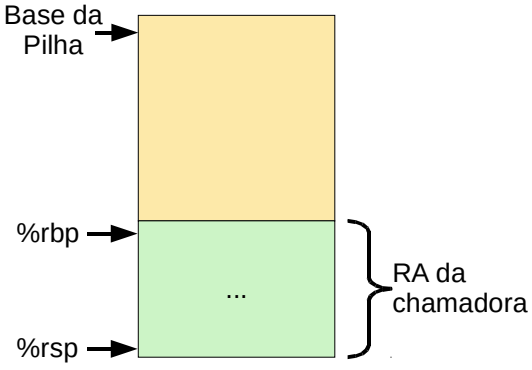
```
func2:
  pushq %rbp
  movq  %rsp, %rbp
  …
  movq %rbp, %rsp
  popq %rbp
  ret
```



Término de um Procedimento

- Finalmente, a função chamada pode retornar o controle para a função chamadora

```
func2:
  pushq %rbp
  movq  %rsp, %rbp
  …

  leave
  ret
```

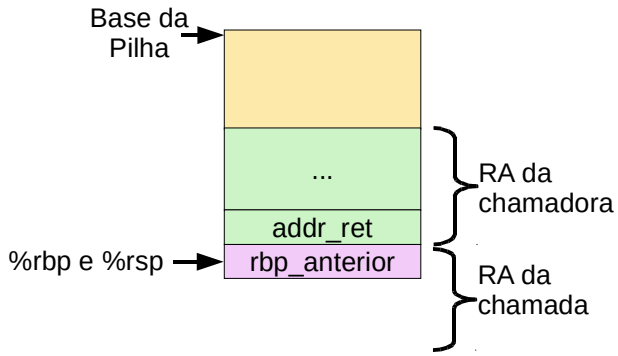


Registro de Ativação: Alocação e Alinhamento

Alocando Espaço

- Para alocar espaço no RA, subtraímos de %rsp um múltiplo de 8
  - Unidade de alocação é uma palavra (8 bytes)

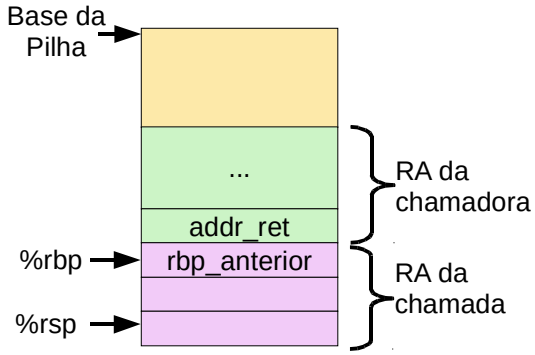
```
func:
  pushq %rbp
  movq  %rsp, %rbp
```



Alocando Espaço

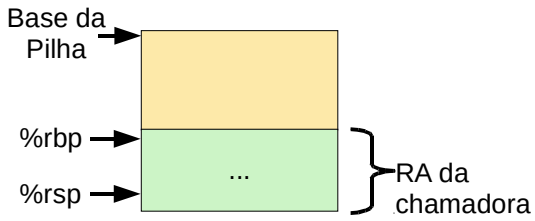
- Para alocar espaço no RA, subtraímos de %rsp um múltiplo de 8
  - Unidade de alocação é uma palavra (8 bytes)

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



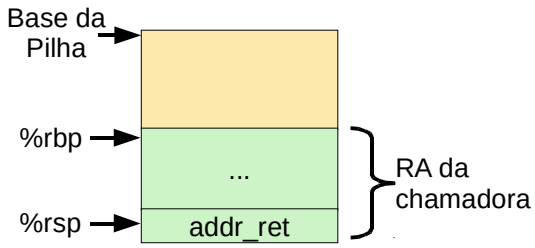
Alinhamento da Pilha

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



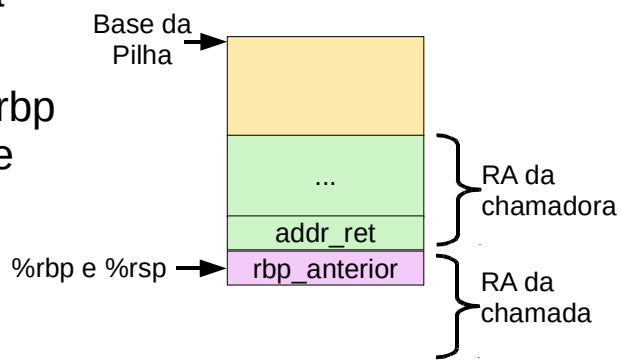
Alinhamento da Pilha

- A ABI Linux x86-64 estabelece que o topo da pilha (RA) deve estar alinhado num endereço múltiplo de 16
- Após a execução da instrução *call* a pilha fica desalinhada



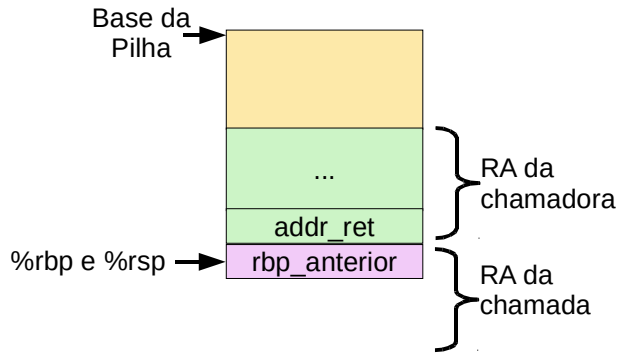
Alinhamento da Pilha

- A ABI Linux x86-64 estabelece que o topo da pilha (RA) deve estar alinhado num endereço múltiplo de 16
- Após a execução da instrução *call* a pilha fica desalinhada
- Mas o salvamento de %rbp alinha a pilha novamente



Alinhamento e Alocação

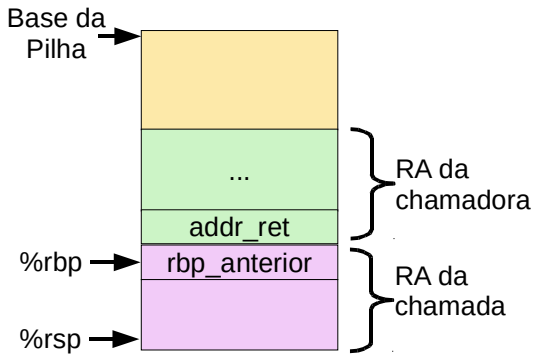
- O compilador sabe calcular o tamanho do RA de uma função
  - Variáveis locais
  - Salvamento de registradores
  - Temporários
  - etc.



Alinhamento e Alocação

- Para garantir o alinhamento, a alocação de espaço é feita no início da função
  - **Sempre é alocado um tamanho múltiplo de 16**

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



Salvamento de Registradores

Salvamento de Registradores

- Funções usam registradores para armazenar valores
  - Variáveis locais, temporários, parâmetros, etc.
- Alguns valores de registradores não podem ser perdidos ao se chamar uma outra função
  - Os registradores devem ser guardados na pilha (RA)
  - Mas, quem faz isso? A função chamadora? A função chamada?
    - Resposta: depende...

Salvamento de Registradores

- A convenção C (ABI Linux x86-64) determina que alguns registradores devem ser salvos pela função chamadora (*caller-saved*)
  - A função chamada pode alterar os valores desses registradores
  - Se a chamadora quiser preservar os valores, ela deve salvar esses registradores antes do *call* e recuperá-los depois do *call*

Salvamento de Registradores

- Por outro lado, alguns registradores devem ser salvos pela função chamada (*callee-saved*)
  - A função chamada deve entregar esses registradores com os mesmos valores que recebeu
    - A função chamadora assume que esses registradores não serão alterados na chamada *call*
  - Se a função chamada quiser usar esses registradores, ela deve salvá-los no início do procedimento, usá-los e restaurá-los ao término do procedimento

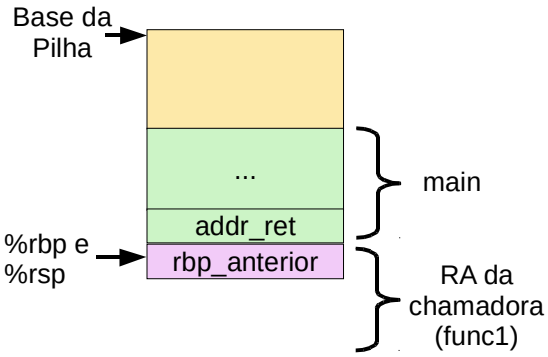
Salvamento de Registradores

REGISTRADOR	RESPONSÁVEL POR SALVAR
%rax	chamadora (caller)
%rbx	chamada (callee)
%rcx	chamadora (caller)
%rdx	chamadora (caller)
%rdi	chamadora (caller)
%rsi	chamadora (caller)
%r8	chamadora (caller)
%r9	chamadora (caller)
%r10	chamadora (caller)
%r11	chamadora (caller)
%r12	chamada (callee)
%r13	chamada (callee)
%r14	chamada (callee)
%r15	chamada (callee)

Exemplo:
caller-saved

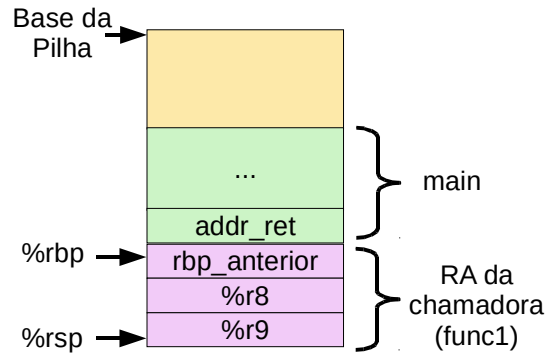
Exemplo: caller-saved

```
func1:
  pushq %rbp
  movq  %rsp, %rbp
```



Exemplo: caller-saved

```
func1:
  pushq %rbp
  movq  %rsp, %rbp
  subl  $16,  %rsp
  …
  movq  %r8, -8(%rbp)
  movq  %r9, -16(%rbp)
  call  func2
```

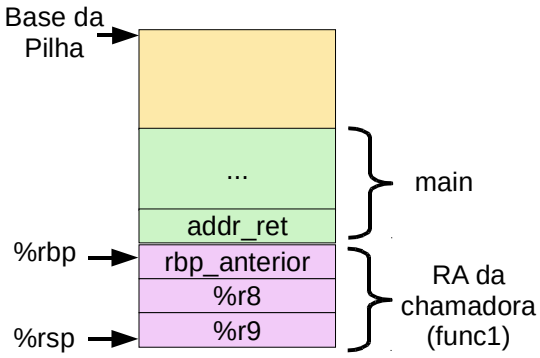


Supondo que %r8 e %r9 possuem valores que não podem ser perdidos, então salvar antes de chamar a função.

Decremento deve ser múltiplo de 16!

Exemplo: caller-saved

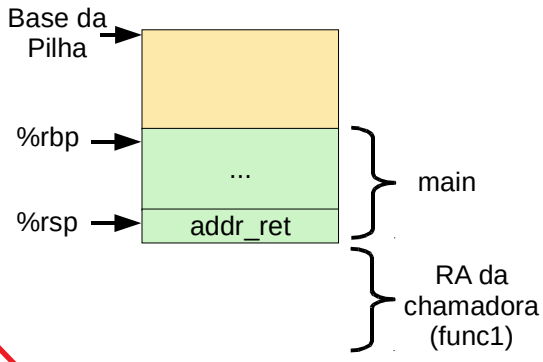
```
func1:
  pushq %rbp
  movq  %rsp, %rbp
  subl  $16,  %rsp
  …
  movq  %r8, -8(%rbp)
  movq  %r9, -16(%rbp)
  call  func2
  movq  -8(%rbp),  %r8
  movq  -16(%rbp), %r9
```



Depois da chamada, retornar os valores salvos para os registradores.

Exemplo: caller-saved

```
main:
  pushq %rbp
  movq  %rsp, %rbp
  subl  $16,  %rsp
  …
  movq  %r8, -8(%rbp)
  movq  %r9, -16(%rbp)
  call  func1
  movq  -8(%rbp),  %r8
  movq  -16(%rbp), %r9
  …
  leave
  ret
```

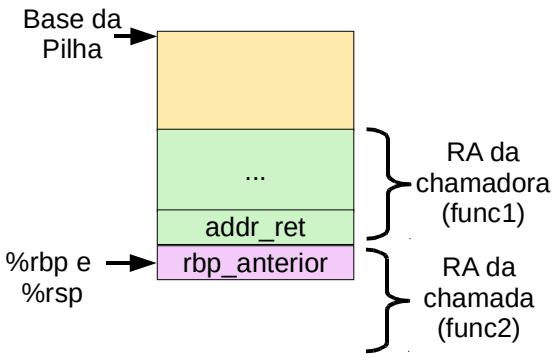


leave desaloca o espaço da pilha no fim da função.

Exemplo: callee-saved

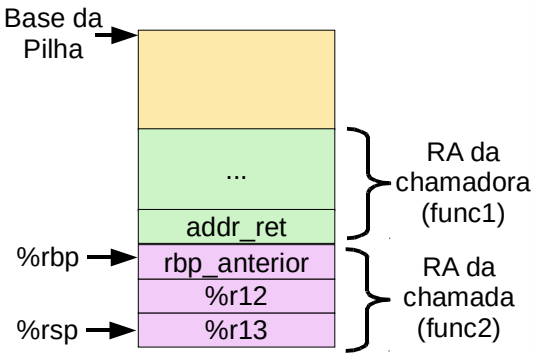
Exemplo: callee-saved

```
func2:
  pushq %rbp
  movq  %rsp, %rbp
```



Exemplo: callee-saved

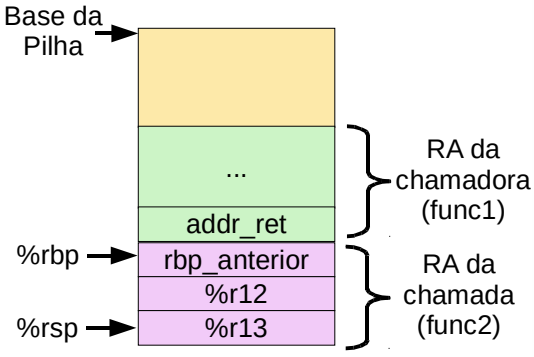
```
func2:
  pushq %rbp
  movq  %rsp, %rbp
  subl  $16,  %rsp
  movq  %r12, -8(%rbp)
  movq  %r13, -16(%rbp)
```



func2 irá usar os registradores %r12 e %r13, então deve salvar os seus valores logo no início.

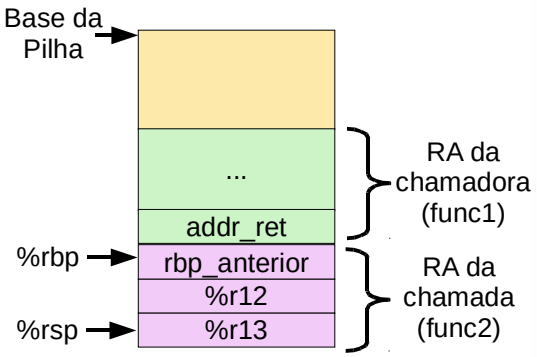
Exemplo: callee-saved

```
func2:
  pushq %rbp
  movq  %rsp, %rbp
  subl  $16,  %rsp
  movq  %r12, -8(%rbp)
  movq  %r13, -16(%rbp)
  …
  # usa %r12 e %r13
```



Exemplo: callee-saved

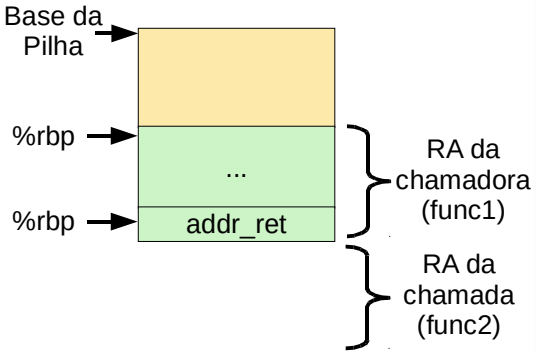
```
func2:
  pushq %rbp
  movq  %rsp, %rbp
  subl  $16,  %rsp
  movq  %r12, -8(%rbp)
  movq  %r13, -16(%rbp)
  …
  # usa %r12 e %r13
  …
  movq -8(%rbp),  %r12
  movq -16(%rbp), %r13
```



Antes do retorno, func2 deve voltar os valores originais que estavam salvos na pilha para os devidos registradores.

Exemplo: callee-saved

```
func1:
  pushq %rbp
  movq  %rsp, %rbp
  subl  $16,  %rsp
  movq  %r12, -8(%rbp)
  movq  %r13, -16(%rbp)
  …
  # usa %r12 e %r13
  …
  movq -8(%rbp),  %r12
  movq -16(%rbp), %r13
  leave
  ret
```

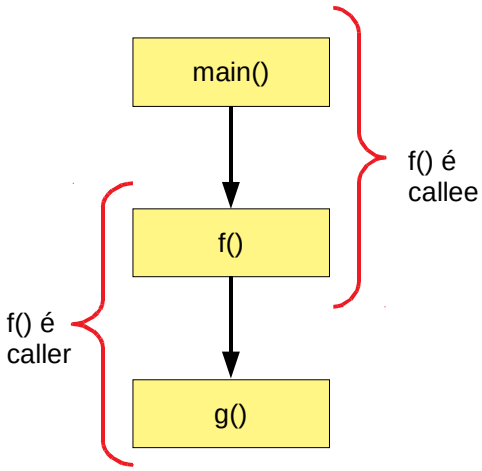


leave libera a memória alocada na pilha.

Outro exemplo…

Exemplo: f() é *caller* e *callee*

```
int g(int x);

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



Exemplo: f() é *caller* e *callee*

- f() é callee
  - Se usar %rbx, %r12, %r13, %r14 ou %r15
    - Salvar no início
    - Recuperar no final
- f() é caller
  - Se há algo importante em %rax, %rcx, …, ou %r11
    - Salvar antes da chamada a g()
    - Recuperar depois da chamada de g()

Dica Geral

- Se a função é folha (não chama nenhuma outra), use de preferência %rax,…,%r11, pois a chamadora já salvou e a função pode alterar sem precisar salvar nada.
- Se a função é intermediária (chama outra), use de preferência %rbx,…,%r15, pois mesmo tendo que salvar no início e recuperar no final, ao chamar outra função esses registradores são preservados pela chamadora. Assim, não precisa salvar a cada chamada.

Exemplo: f() é *caller* e *callee*

```
int g(int x);

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

- Parâmetros:
  - i → %edi
  - n → %esi
- Sugestão de uso de registradores:
  - i → %r12d
  - n → %r13d
  - a → %r14d