

# Procedimentos

## Variáveis Locais

Noemi Rodriguez  
Ana Lúcia de Moura  
Raúl Renteria  
Alexandre Meslin

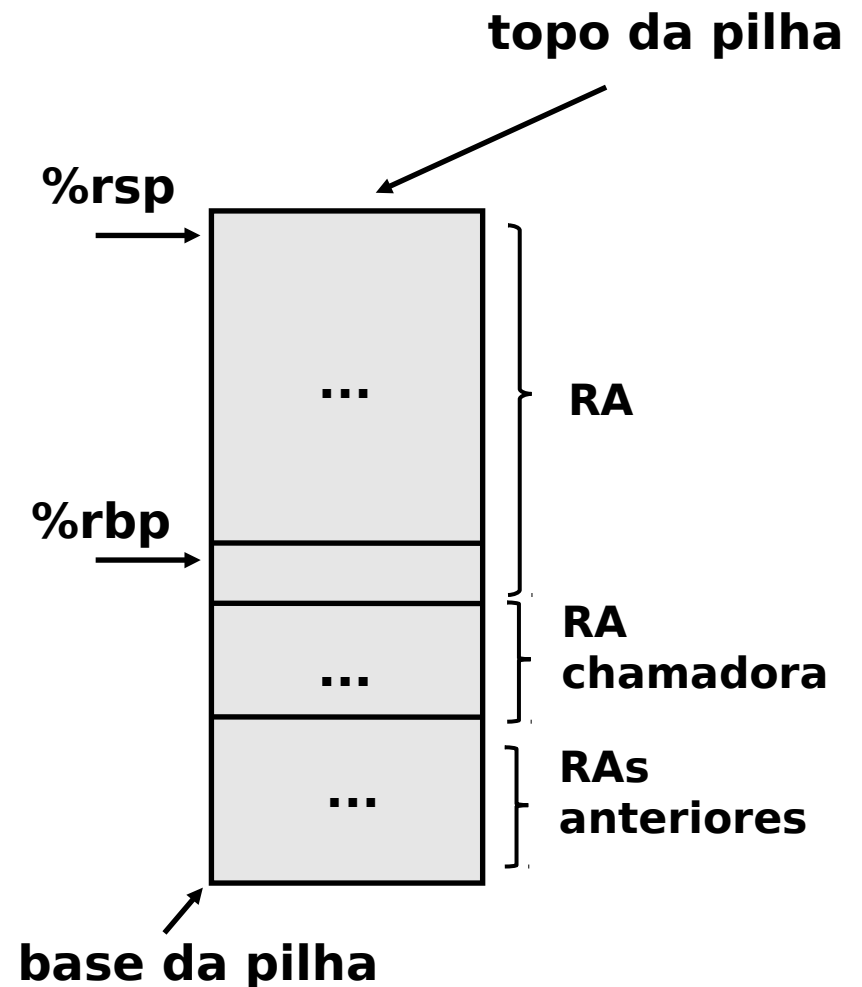
<http://www.inf.puc-rio.br/~inf1018>

# Registro de Ativação

Porção da pilha associada a uma chamada de função

O registrador **%rbp** (*frame* ou *base pointer*) pode ser usado como **base do registro de ativação**

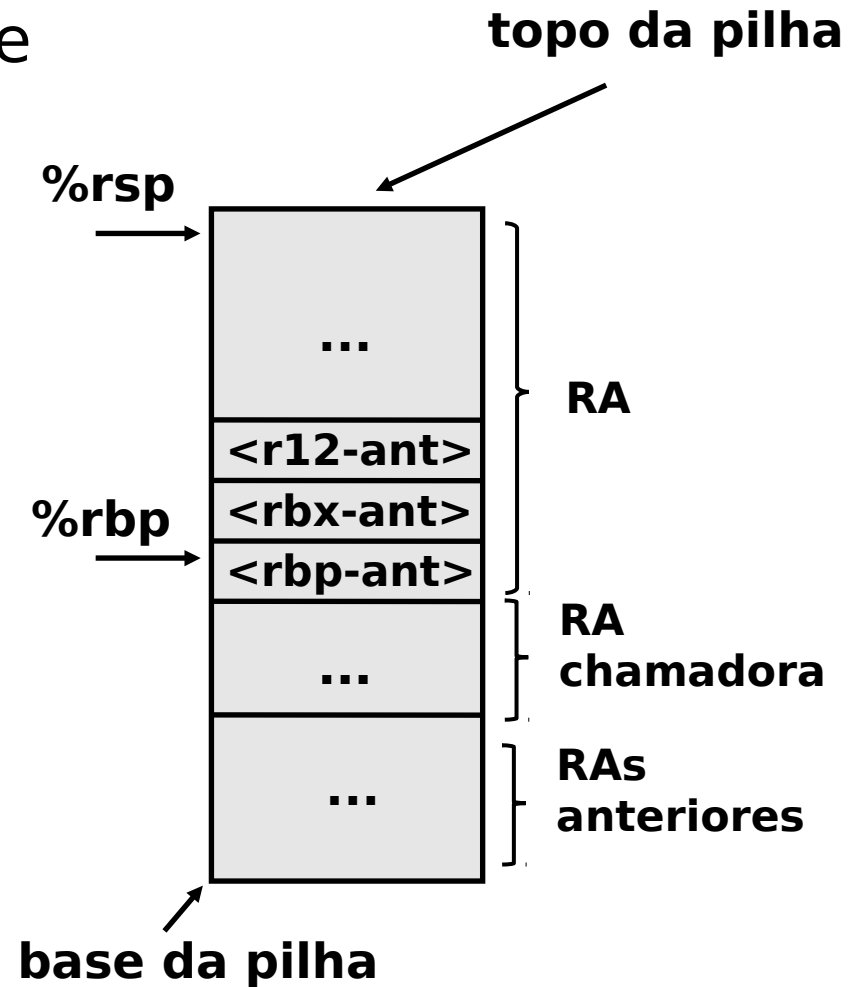
- acesso a elementos alocados no registro de ativação da função



# Registro de Ativação: registradores

RA é usado para guardar valor de registradores callee-saved

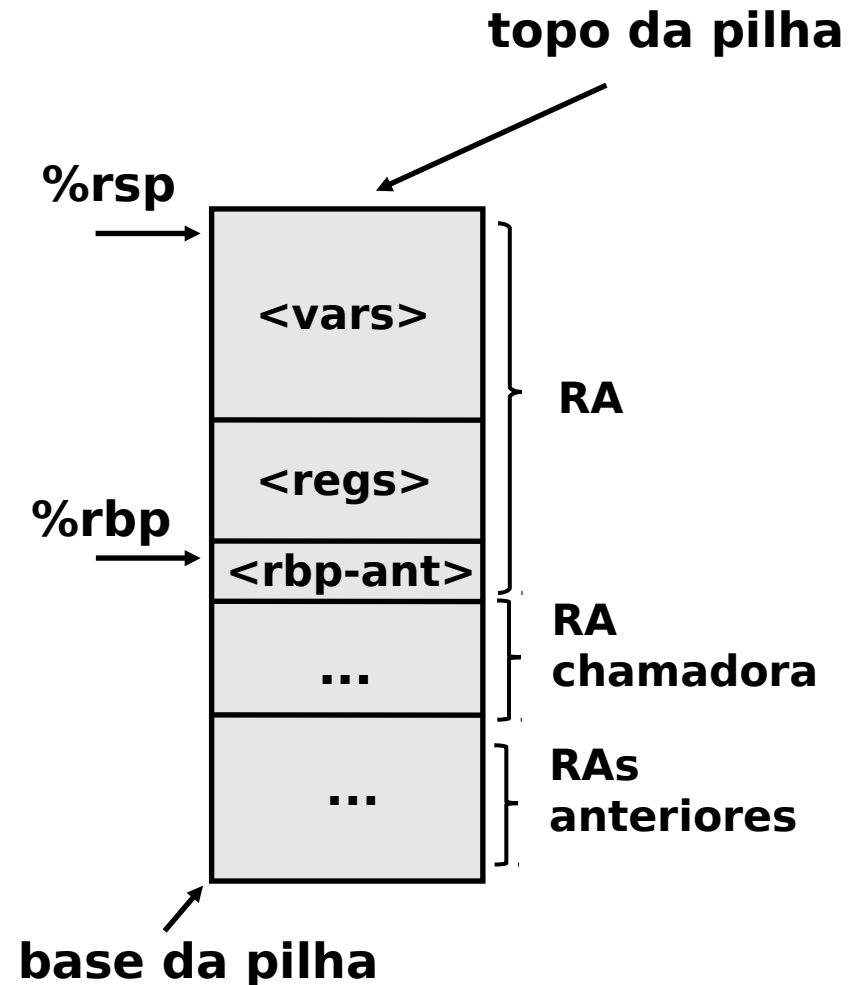
```
foo:
    pushq %rbp
    movq  %rsp,%rbp
    subq  $n,%rsp /* multiplo 16 */
    movq  %rbx, -8(%rbp)
    movq  %r12, -16(%rbp)
```



# Registro de Ativação: variáveis locais

RA também é usado para armazenar variáveis locais

- número de registradores é insuficiente
- arrays e estruturas
- operador & é aplicado a uma variável



# Alocação de Variáveis Locais

---

Não existe convenção para a ordenação das variáveis locais no RA

- apenas a própria função manipula seus endereços

Devem ser respeitadas as convenções de alinhamento!

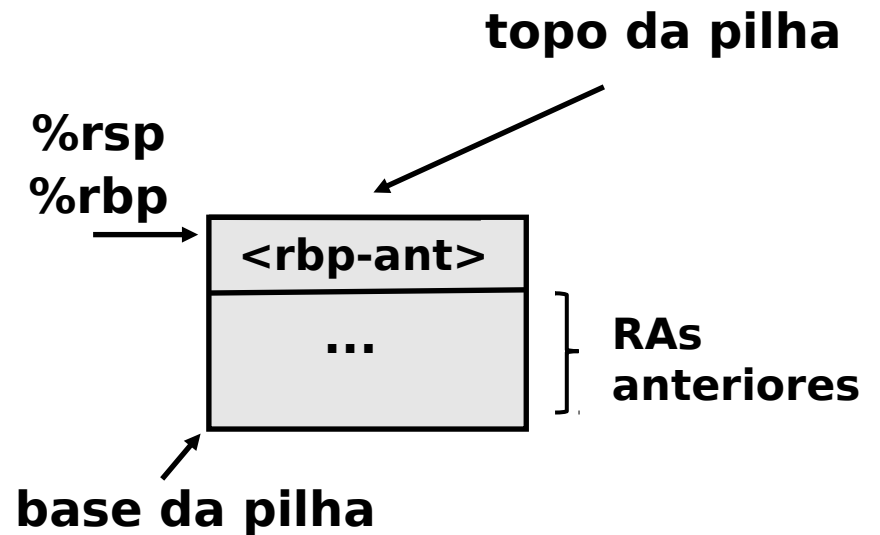
- variáveis escalares, arrays e estruturas

A pilha deve permanecer alinhada em endereço múltiplo de 16

# Alocação de variáveis escalares

```
int foo() {  
  
    char c = 0, d = 1;  
  
    int i = 10;  
  
    long l = 100;  
  
    ...  
}
```

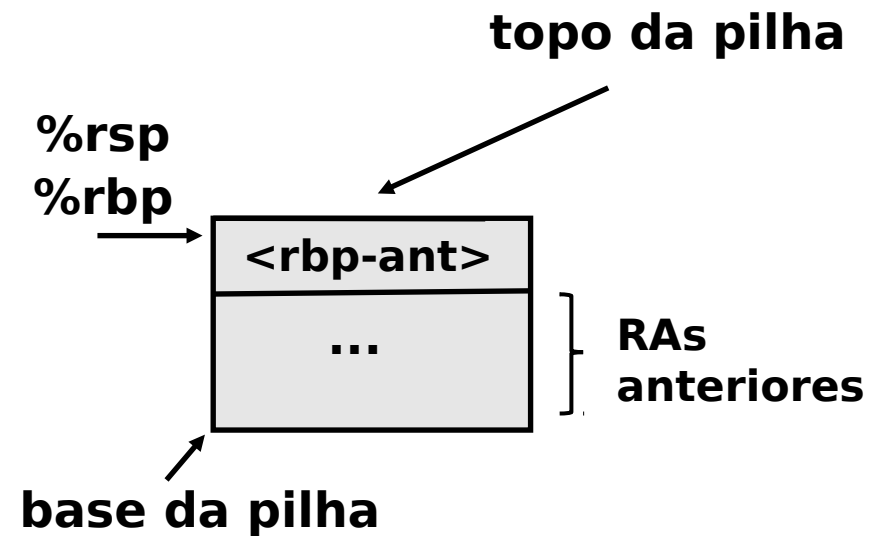
```
foo:  
  
    pushq %rbp  
  
    movq %rsp, %rbp  
  
    ...
```



# Alocação de variáveis escalares

```
int foo() {  
  
    char c = 0, d = 1; —▶ 2 * 1  
  
    int i = 10; —▶ 4  
  
    long l = 100; —▶ 8  
  
    ...
```

```
foo:  
  
    pushq %rbp  
  
    movq %rsp, %rbp  
  
    ...
```

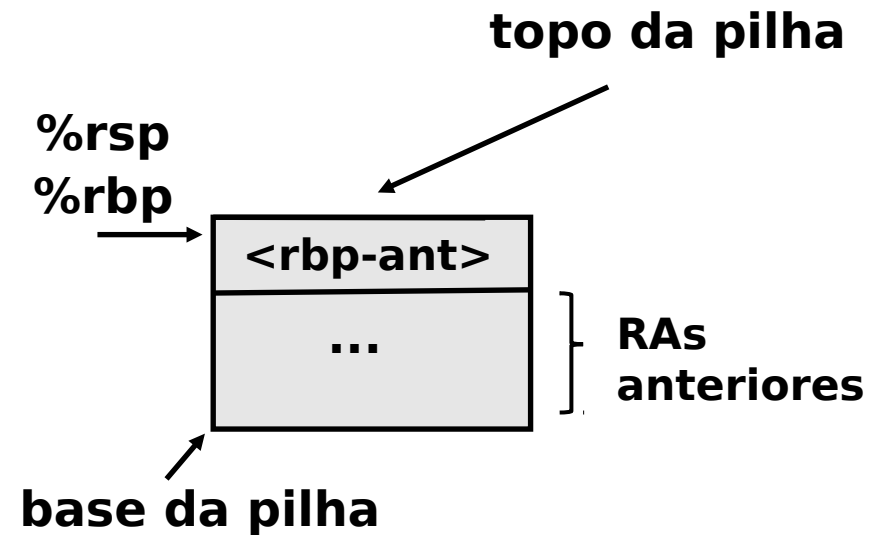


# Alocação de variáveis escalares

```
int foo() {  
    char c = 0, d = 1; → 2 * 1  
    int i = 10; → 4  
    long l = 100; → 8  
    ...  
}
```

14 → 16

```
foo:  
    pushq %rbp  
    movq %rsp, %rbp  
    ...
```



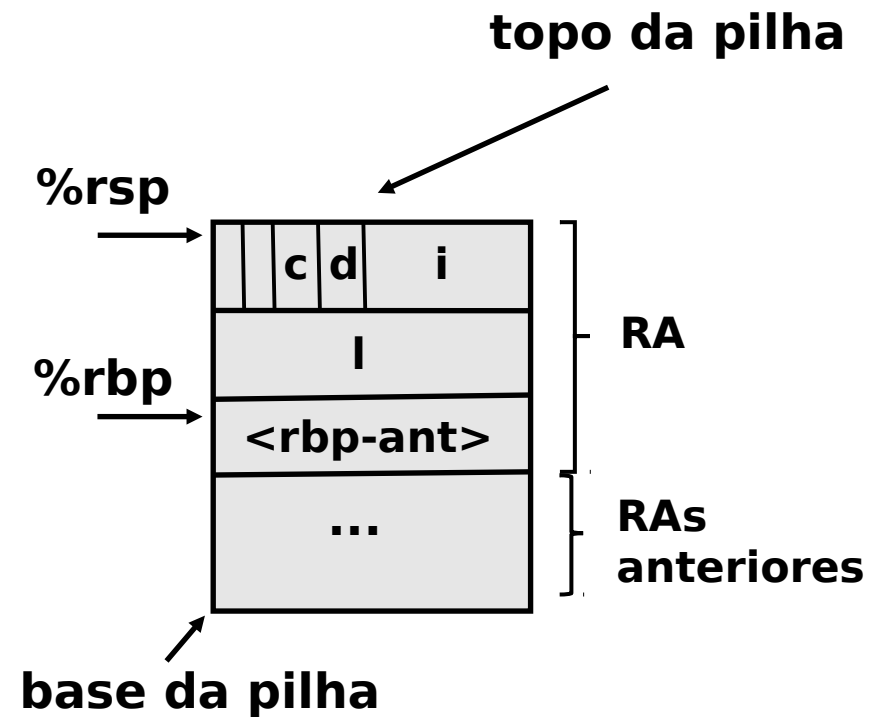


# Alocação de variáveis escalares

```
int foo() {  
  
    char c = 0, d = 1; → 2 * 1  
  
    int i = 10; → 4  
  
    long l = 100; → 8  
  
    ...  
}
```

16

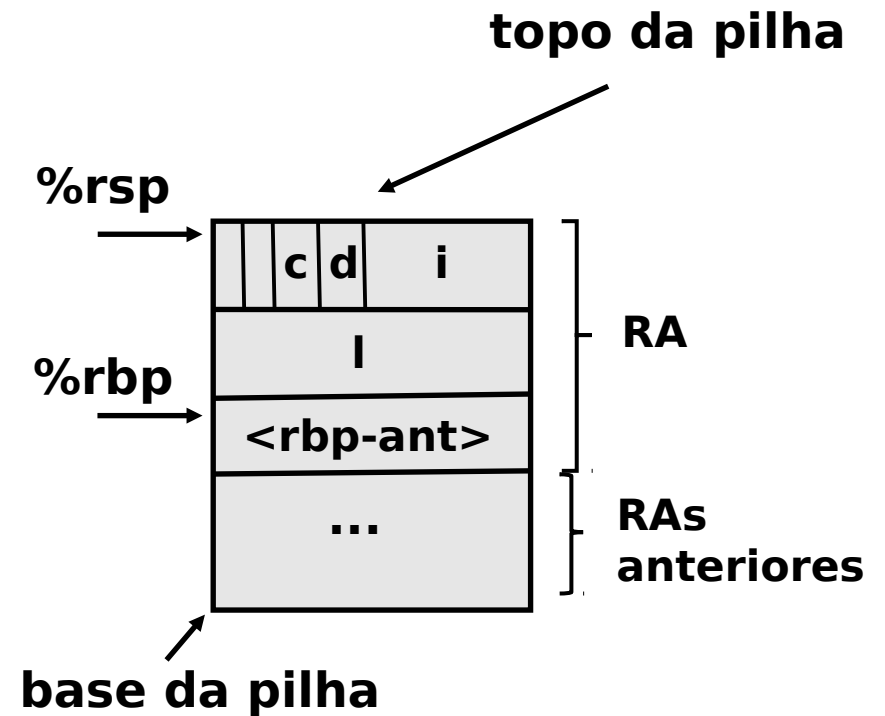
```
foo:  
  
    pushq %rbp  
  
    movq %rsp, %rbp  
  
    subq $16, %rsp  
  
    ...
```



# Acesso às variáveis

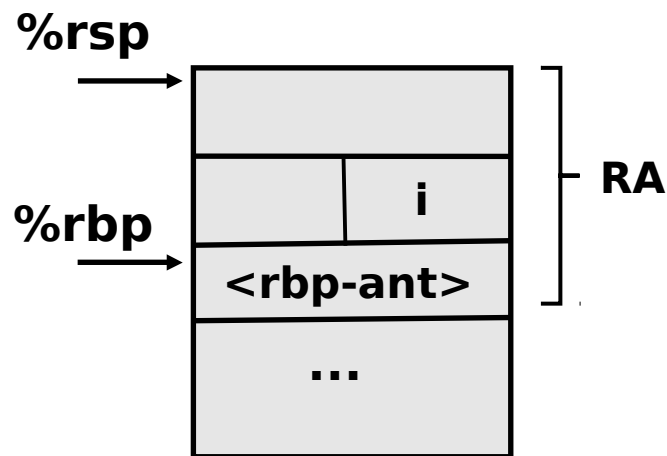
```
int foo() {  
  
    char c = 0, d = 1;  
  
    int i = 10;  
  
    long l = 100;  
  
    ...
```

```
movb $0, -14(%rbp) /* c */  
movb $1, -13(%rbp) /* d */  
movl $10, -12(%rbp) /* i */  
movq $100, -8(%rbp) /* l */
```



# Um Exemplo

```
int foo() {  
    int i;  
    scanf("%d", &i);  
    return i;  
}
```



```
Sf: .string "%d"
```

```
foo:
```

```
    pushq %rbp
```

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

```
    subq $16, %rsp
```

```
    movq $Sf, %rdi
```

```
    leaq -4(%rbp), %rsi
```

```
    call scanf
```

```
    movl -4(%rbp), %eax
```

```
    leave
```

```
    ret
```

# Outro Exemplo

```
int proc() {  
    int x,n=5,s=0;  
    while (n-->0) {  
        scanf ("%d",&x);  
        s += x;  
    }  
    return s;  
}
```

```
Sf: .string "%d"
```

```
proc:
```

```
    pushq %rbp
```

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

```
    subq $32, %rsp
```

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

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

```
    movl $5, %ebx
```

```
    movl $0, %r12d
```

```
loop:
```

```
    cmpl $0, %ebx
```

```
    je fim
```

```
    decl %ebx
```

```
    movq $Sf, %rdi
```

```
    leaq -20(%rbp), %rsi
```

```
    call scanf
```

```
    addl -20(%rbp), %r12d
```

```
    jmp loop
```

```
fim:
```

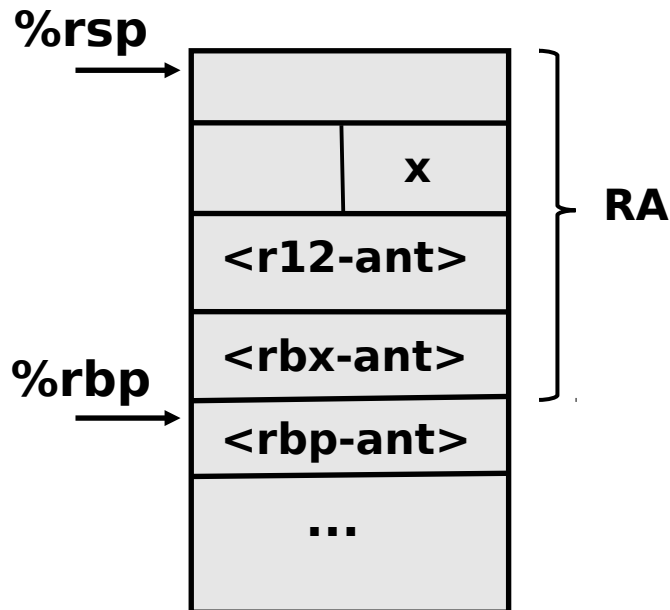
```
    movl %r12d, %eax
```

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

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

```
    leave
```

```
    ret
```

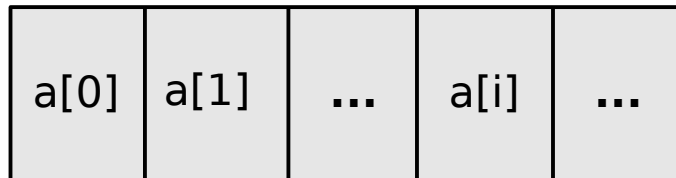


# Alocação de Arrays

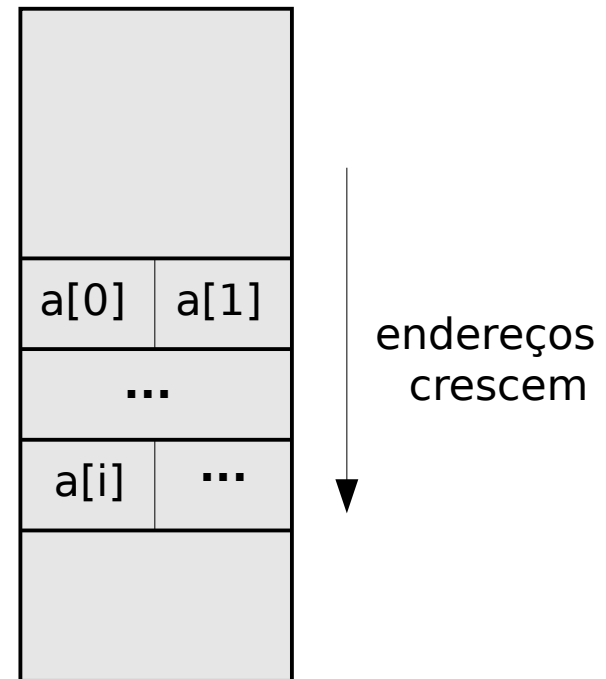
---

Cálculo do endereço dos elementos de um array é feito a partir do seu endereço inicial

o elemento de índice 0 é o primeiro elemento, e deve estar armazenado no **menor** endereço



→  
endereços crescem

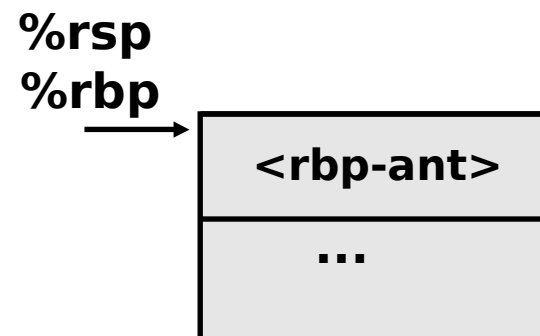


# Exemplo de Arrays Locais

---

```
int foo() {  
    int v1[3];  
    long v2[2];  
    boo(v1, v2);  
    ...  
}
```

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

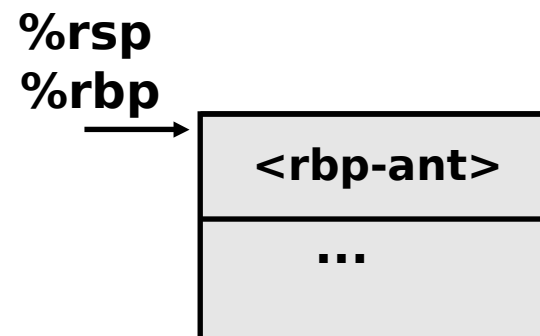


# Exemplo de Arrays Locais

---

```
int foo() {  
    int v1[3]; → 3 * 4  
    long v2[2]; → 2 * 8  
    boo(v1, v2);  
    ...  
}
```

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

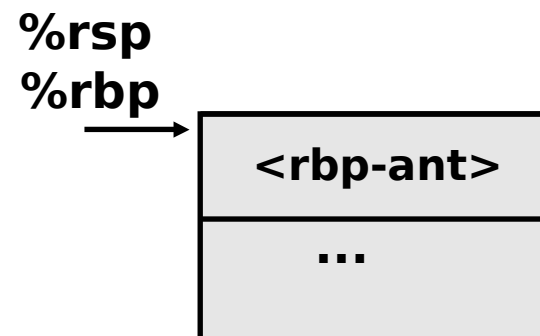


# Exemplo de Arrays Locais

```
int foo() {  
    int v1[3]; → 3 * 4  
    long v2[2]; → 2 * 8  
    boo(v1, v2);  
    ...  
}
```

28 → 32

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

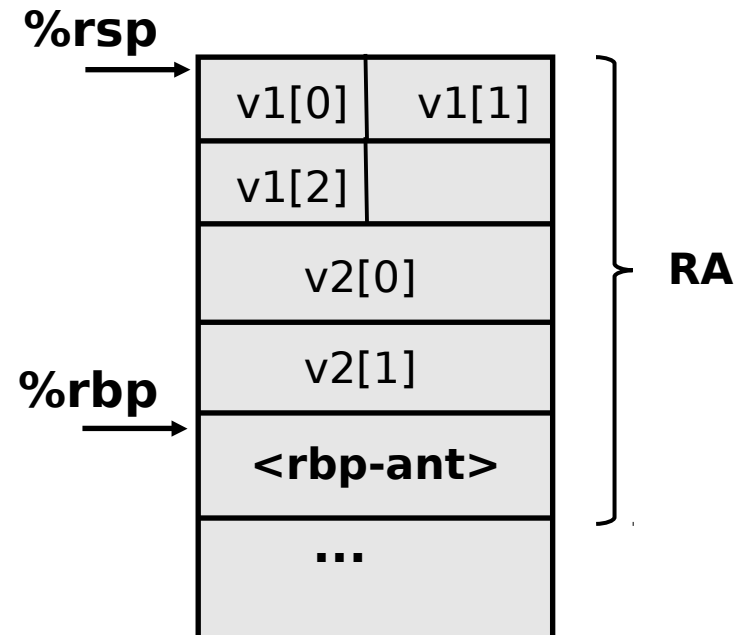




# Exemplo de Arrays Locais

```
int foo() {  
    int v1[3]; → 3 * 4  
    long v2[2]; → 2 * 8  
    boo(v1, v2);  
    ...  
}
```

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



# Exemplo de Arrays Locais

```
int foo() {  
    int v1[3]; → 3 * 4  
    long v2[2]; → 2 * 8  
    boo(v1, v2);  
    ...  
}
```

```
foo:  
    pushq %rbp  
    movq  %rsp, %rbp  
    subq  $32, %rsp  
    leaq  -32(%rbp), %rdi  
    leaq  -16(%rbp), %rsi  
    call  boo  
    ...
```

