Software Básico

Procedimentos
Chamada de Função e Parâmetros

INSTITUTO DE INFORMÁTICA - UFG

Reconhecimento

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

Pilha de Execução

Memória

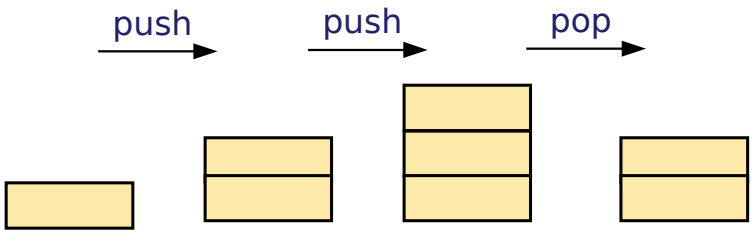
- Durante a execução de um programa, o SO precisa alocar memória principal para:
  - Dados globais
  - Código

  } Tamanho (fixo) conhecido na compilação

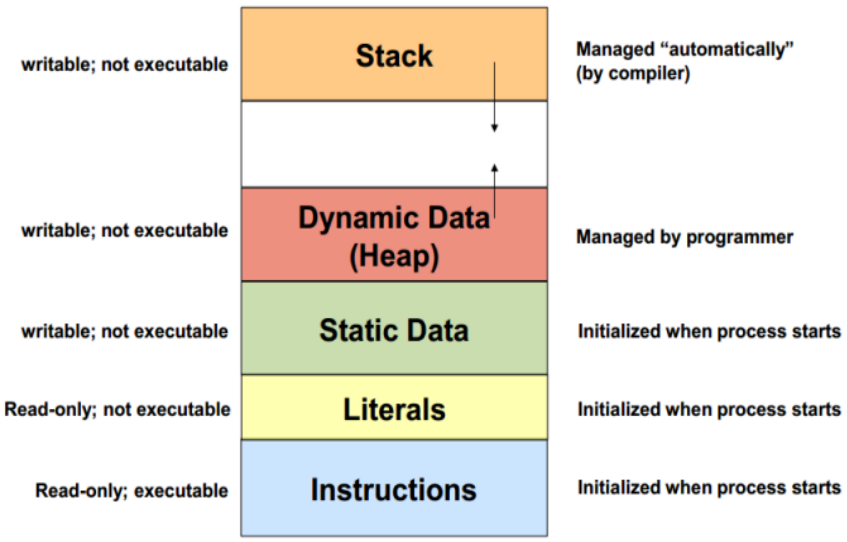
- Variáveis locais → Para cada função
- Resultados intermediários → Quantos? Quais?

Pilha de Execução

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

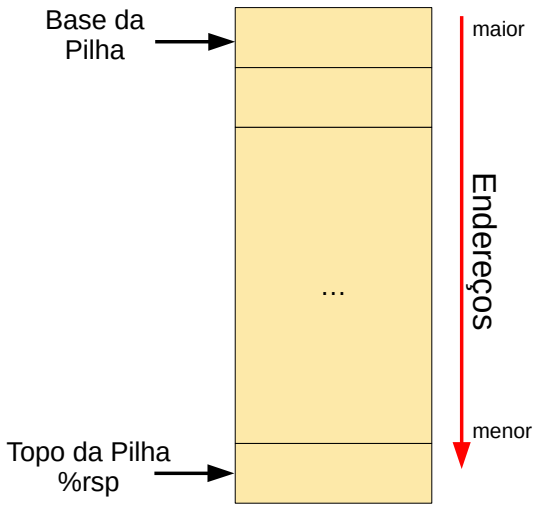


Layout de Memória



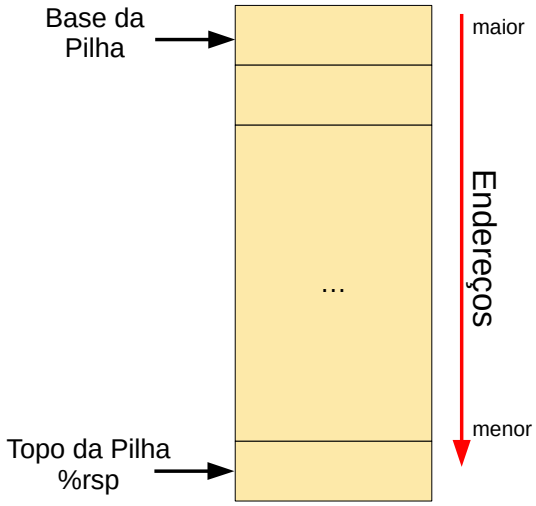
Implementação da Pilha

- Registrador %rbp aponta para a base da pilha
- Registrador %rsp aponta para o topo da pilha
- Instruções:
  - pushq
  - popq
- A pilha “cresce” em direção ao início da memória

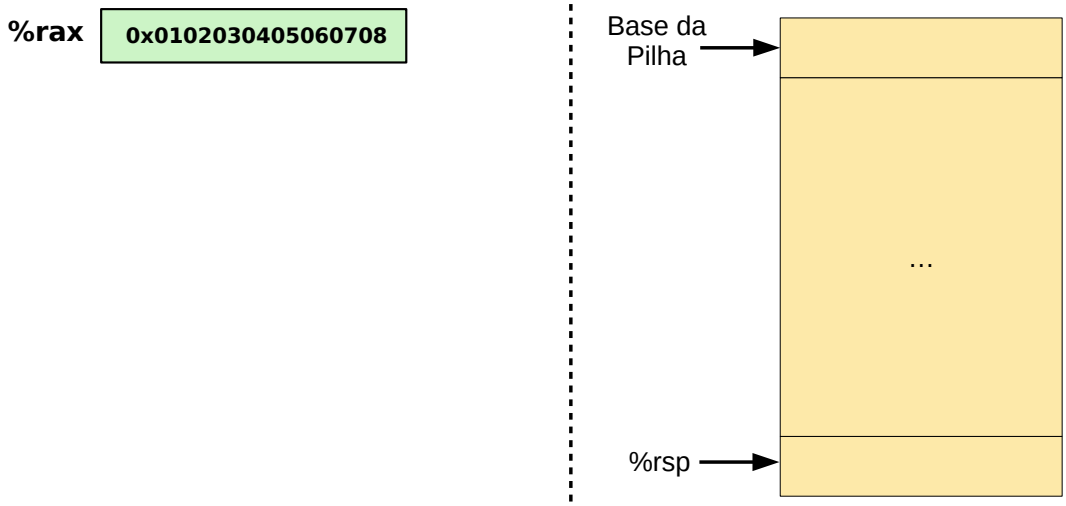


Implementação da Pilha

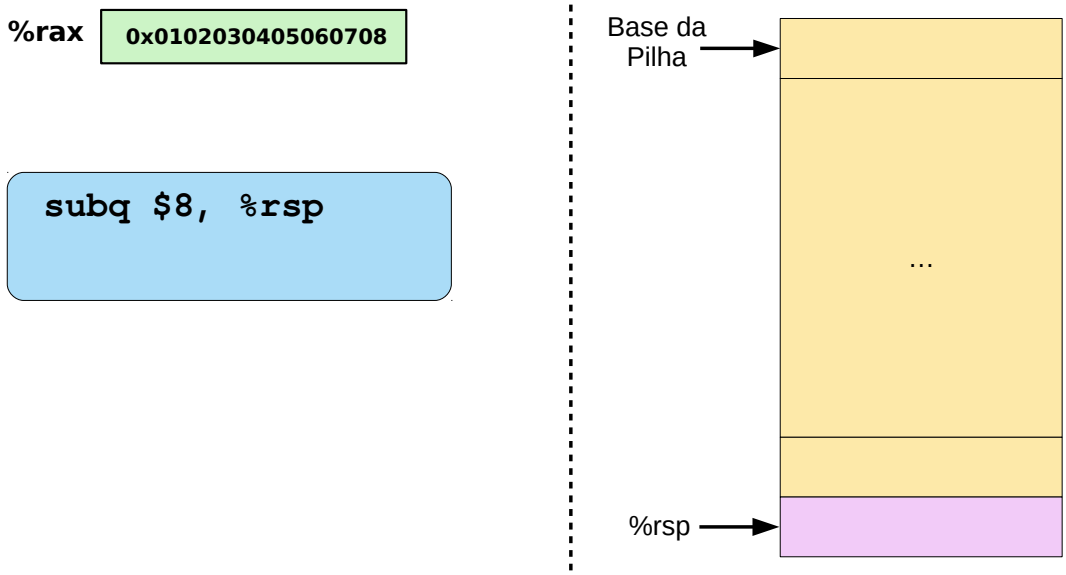
- A unidade de alocação é uma palavra (8 bytes)
  - Para alocar espaço subtraímos 8 de %rsp (ou múltiplo de 8)
  - Para liberar espaço somamos 8 a %rsp (ou múltiplo de 8)



Empilhando um Resultado



Empilhando um Resultado

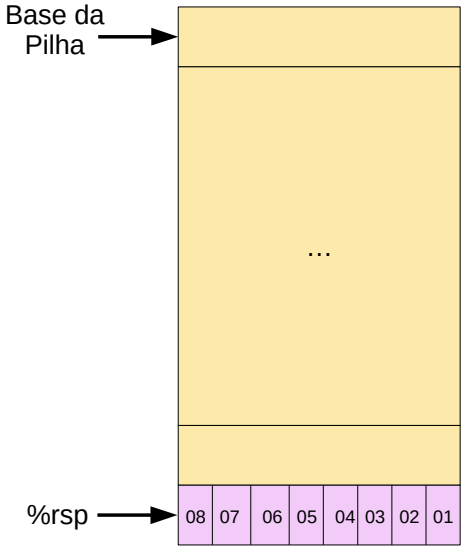



```
subq $8, %rsp
```

Empilhando um Resultado

%rax **0x0102030405060708**

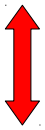
```
subq $8, %rsp
movq %rax, (%rsp)
```



Empilhando um Resultado

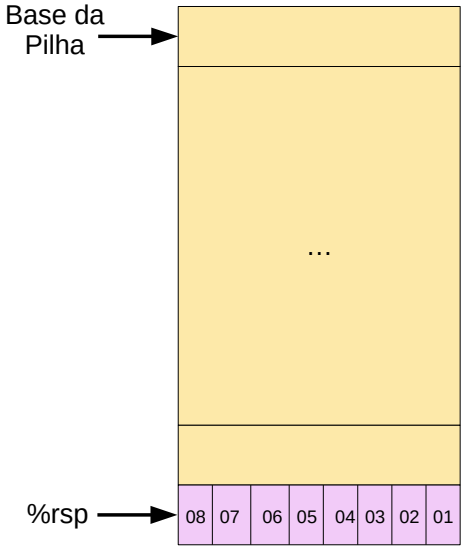
%rax **0x0102030405060708**

```
subq $8, %rsp
movq %rax, (%rsp)
```

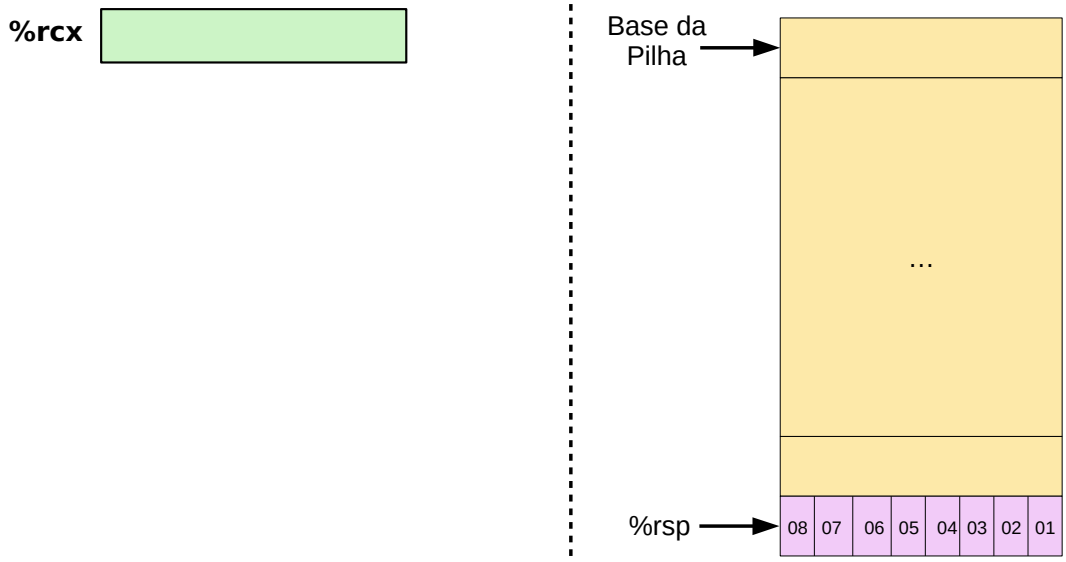


Equivalente

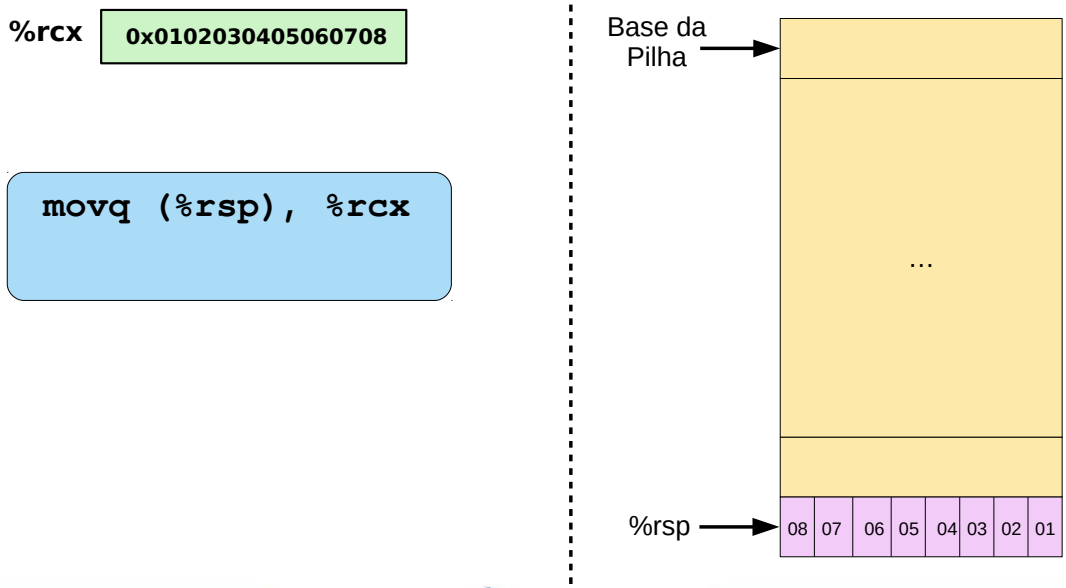
```
pushq %rax
```



Empilhando um Resultado



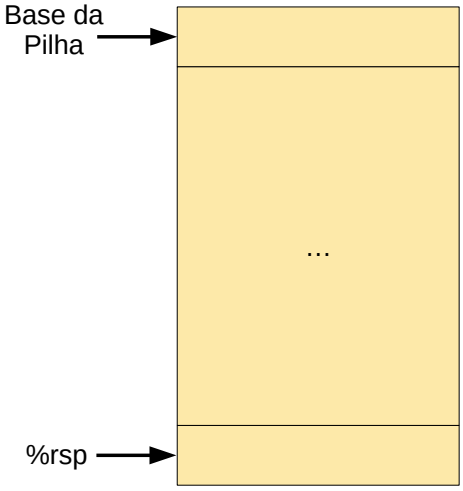
Empilhando um Resultado



Empilhando um Resultado

%rcx **0x0102030405060708**

```
movq (%rsp), %rcx
addq $8, %rsp
```



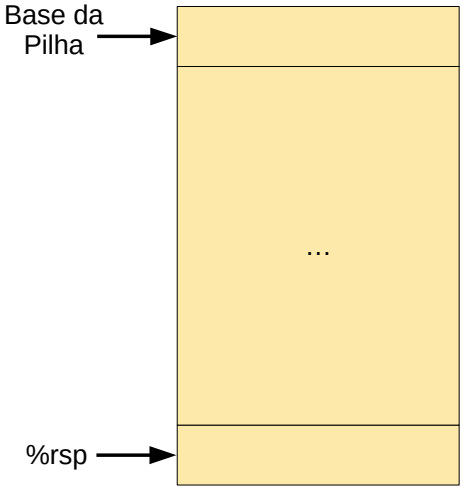
Empilhando um Resultado

%rcx **0x0102030405060708**

```
movq (%rsp), %rcx
addq $8, %rsp
```

Equivalente

```
popq %rcx
```



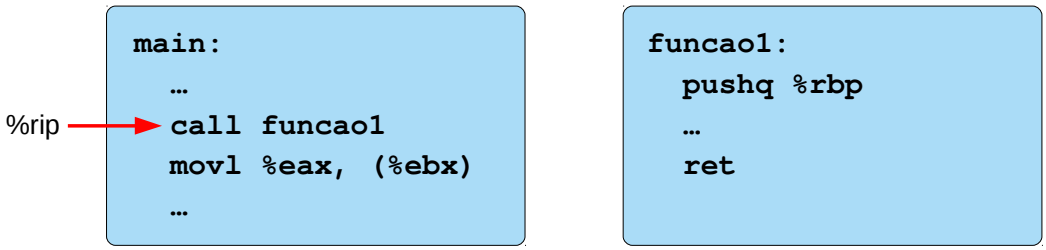
Procedimentos

Procedimentos (Funções)

- Quando chamamos um procedimento, transferimos dados e controle de uma parte do código para outra
  - Chamador (*caller*) ↔ Chamado (*callee*)
  - 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

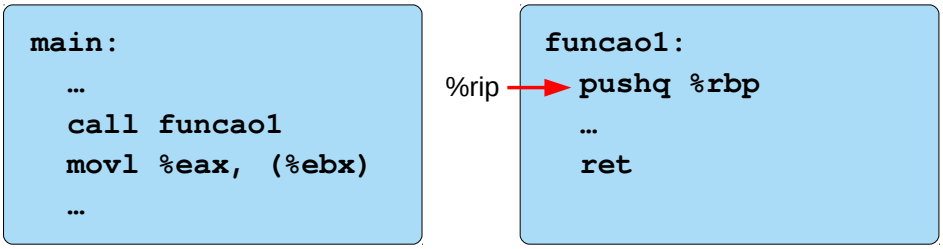
Transferência de Controle

- A instrução *call* transfere o controle para a função
  - Fluxo vai para a instrução no endereço de memória indicado pelo label



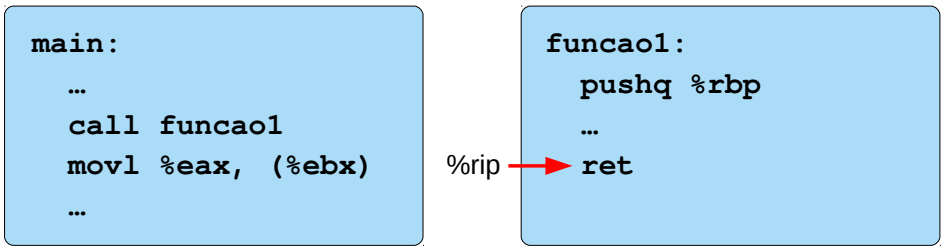
Transferência de Controle

- A instrução *call* transfere o controle para a função
  - Fluxo vai para a instrução no endereço de memória indicado pelo label



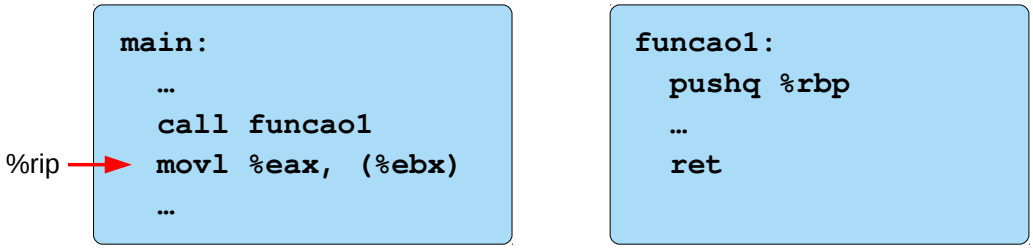
Transferência de Controle

- A instrução *ret* transfere o controle de volta para o chamador
  - A execução continua na instrução após o *call*



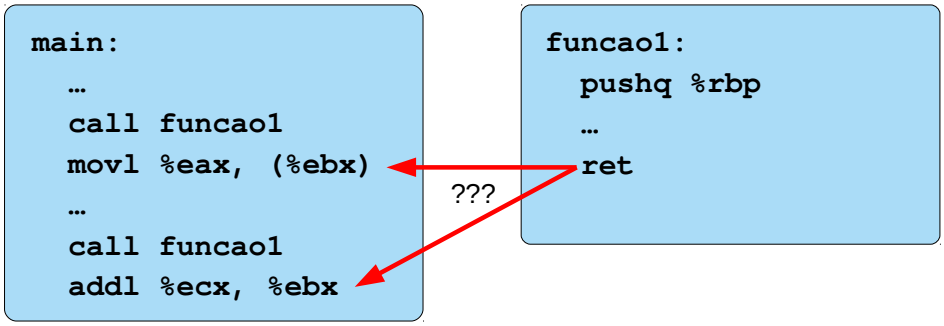
Transferência de Controle

- A instrução *ret* transfere o controle de volta para o chamador
  - A execução continua na instrução após o *call*



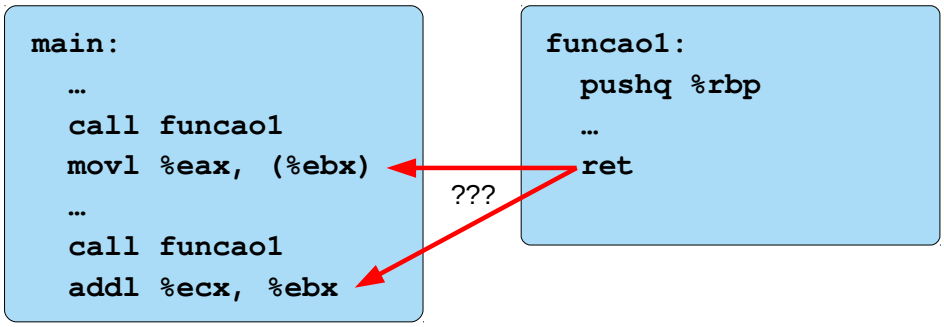
Transferência de Controle

- Como a CPU sabe para onde deve retornar o fluxo?



Transferência de Controle

- Como a CPU sabe para onde deve retornar o fluxo?
  - O endereço de retorno é armazenado na pilha de execução (pela CPU) antes da chamada



Transferência de Controle

```
4004f7:   call   4005ed
4004fc:   movl   %eax, %ebx
```

```
4005ed:   addl %ebx, %eax
4005ef:   ret
```

7fffffffe150	1024
7fffffffe148	
	...

Pilha

%rip	4004f7
%rsp	7fffffffe150

Transferência de Controle

```
4004f7:   call   4005ed
4004fc:   movl   %eax, %ebx
```

```
4005ed:   addl %ebx, %eax
4005ef:   ret
```

7fffffffe150	1024
7fffffffe148	4004fc
	...

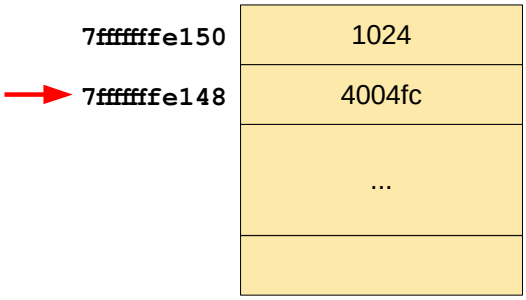
Pilha

%rip	4005ed
%rsp	7fffffffe148

Transferência de Controle

```
4004f7:   call   4005ed
4004fc:   movl   %eax, %ebx
```

```
4005ed:   addl %ebx, %eax
4005ef:   ret
```



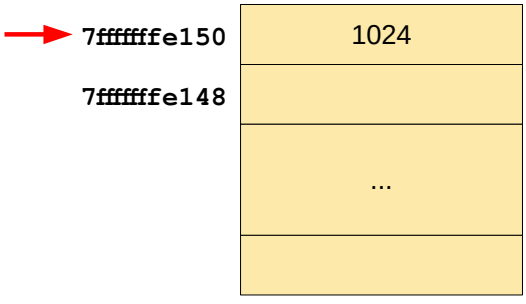
Pilha

%rip	4005ef
%rsp	7fffffffe148

Transferência de Controle

```
4004f7:   call   4005ed
4004fc:   movl   %eax, %ebx
```

```
4005ed:   addl %ebx, %eax
4005ef:   ret
```



Pilha

%rip	4004fc
%rsp	7fffffffe150

Na hora de executar “ret”, o registrador %rsp deve estar apontando para a posição da pilha que contém o endereço de retorno!!

Passagem de Parâmetros

Passagem de Parâmetros

- Instruções *call* e *ret* só transferem controle
  - Não cuidam da passagem de valores
  - Isso é de responsabilidade do programador
- Há convenções para padronizar a passagem dos parâmetros
  - Isso para garantir a interoperabilidade entre compiladores diferentes

Passagem de Parâmetros

- A convenção para C na plataforma Linux x86-64 (ABI) estabelece a passagem de até seis valores inteiros (incluindo ponteiros) via registradores
  - Parâmetros adicionais e estruturas passados na pilha (abaixo do endereço de retorno)

Passagem de Parâmetros

Parâmetro	64 bits	32 bits	16 bits	8 bits
1º.	%rdi	%edi	%di	%dil
2º.	%rsi	%esi	%si	%sil
3º.	%rdx	%edx	%dx	%dl
4º.	%rcx	%ecx	%cx	%cl
5º.	%r8	%r8d	%r8w	%r8b
6º.	%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

```
.data
x:    .int 20
fmt:  .string "%d\n"
```

```
movq $5, %rdi
movl  x, %esi
call somarli
# %eax possui o retorno

movq $fmt, %rdi
movl %eax, %esi
movl   $0, %eax
call printf
```

```
int somarli(long a, int b) {
  return (int)(a + b);
}

int x = 20;

int main() {
  int r = somarli(5, x);
  printf("%d\n", r);
  return 0;
}
```

Exemplo

```
.data
x:    .int 20
fmt: .string "%d\n"
```

```
movq $5, %rdi
movl  x, %esi
call somarli
# %eax possui o retorno

movq $fmt, %rdi
movl %eax, %esi
movl   $0, %eax
call printf
```

```
int somarli(long a, int b) {
  return (int)(a + b);
}

int x = 20;

int main() {
  int r = somarli(5, x);
  printf("%d\n", r);
  return 0;
}
```

Obs: printf pede um valor especial para %eax