“Estou fazendo um sistema operacional gratuito (apenas um hobby, não será grande e profissional como GNU) para 386/486 AT." (Linus Torvalds referindo-se ao Linux; 1991).

# Criando funções no x86-64

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

# Criando funções

- Muitas das convenções sobre funções já foram vistas em aulas passadas quando estávamos chamando funções prontas
- O conceito de criação é o mesmo utilizado no MIPS

# Passagem de parâmetros

- Segundo o System V ABI (Matz, Jaeger, Mitchell; 2014)

| Argumento | Registrador |
|---|---|
| Primeiro | rdi |
| Segundo | rsi |
| Terceiro | rdx |
| Quarto | rcx |
| Quinto | r8 |
| Sexto | r9 |

- Demais parâmetros são empilhados da direita para a esquerda

# Passagem de parâmetros

- Exemplo: como fica a chamada para a função com o seguinte protótipo?

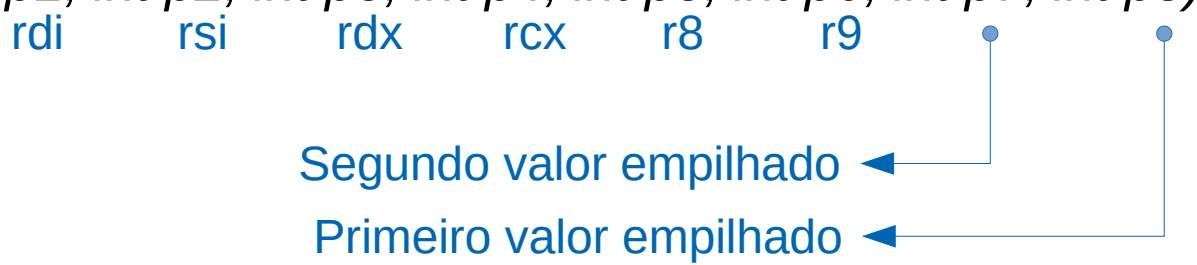
  *void minhaFunc(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8);*

| Argumento | Registrador |
|---|---|
| Primeiro | rdi |
| Segundo | rsi |
| Terceiro | rdx |
| Quarto | rcx |
| Quinto | r8 |
| Sexto | r9 |

# Passagem de parâmetros

- Exemplo: como fica a chamada para a função com o seguinte protótipo?

  *void minhaFunc(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8);*




| Argumento | Registrador |
|---|---|
| Primeiro | rdi |
| Segundo | rsi |
| Terceiro | rdx |
| Quarto | rcx |
| Quinto | r8 |
| Sexto | r9 |

# Salvar ou não salvar, eis a questão

- Sua função deve garantir que os seguintes registradores serão devolvidos ao chamador com os seus conteúdos originais
  - **rsp, rbp, rbx, r12, r13, r14, r15**
- O chamador deve assumir que os demais registradores podem ser alterados pela função
  - **Salve-os antes de chamar a função se necessário**

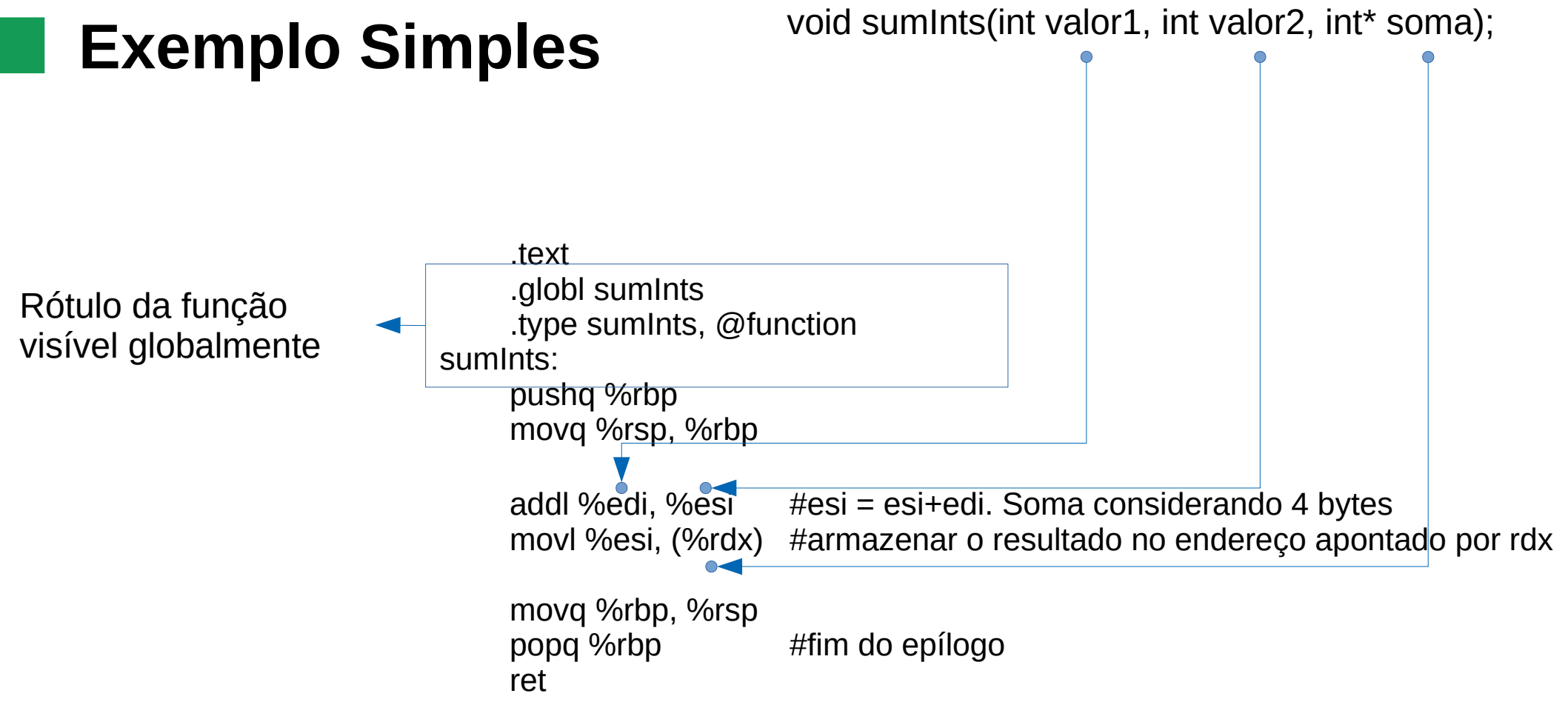
# Regras da pilha

- **Argumentos passados via pilha devem ocupar 8 bytes**
  - Não importa se representa um char, short, int, …
- Variáveis locais que são alocadas na pilha **podem ocupar exatamente** os seus tamanhos
  - Exemplo: duas variáveis locais que representam chars podem ocupar 2 bytes
- rbp (frame pointer) deve ser ajustado na entrada da função, e então **permanecer inalterado dentro da função**
- A pilha deve ter um tamanho múltiplo de 16 antes de chamar uma função
  - Ajuste rsp adequadamente – *stack pointer*
  - O stack pointer pode ser alterado dentro da função
    - Pode mudar de posição conforme novas variáveis precisam ser alocadas
- Variáveis na pilha **devem estar em um endereço que é múltiplo do seu tamanho**
  - Por exemplo, considerando que um inteiro ocupa 4 bytes
    - **Ele pode** estar no endereço -0, -4, -8, … a partir do frame pointer
    - **Não pode** estar no endereço -3 a partir do frame pointer

# Exemplo Simples

- Crie um programa chamado sumInts.s
  - O programa vai conter a função equivalente a
    void sumInts(int valor1, int valor2, int* soma);
    - O resultado é escrito no valor apontado por *soma

# Exemplo Simples



# Usando a função

- Podemos criar um “main” no mesmo arquivo que contém a função e utilizá-la normalmente
  - Como fizemos no MIPS
- Uma solução melhor é criar um arquivo separado para o “main”, e deixar a função em seu próprio arquivo

# Exercícios

1. Crie um arquivo chamado *mainSumInts.s*
    1. Vai conter o “main”
    2. Invoque a função *sumInts* a partir desse arquivo
        1. Basta fazer uma *call*
    3. Pode fazer a chamada para *sumInts* passando valores constantes
    4. Imprima o resultado na tela utilizando o *printf* do C
    5. Para montar e compilar

```
as sumInts.s -o sumInts.o
as mainSumInts.s -o mainSumInts.o
gcc mainSumInts.o sumInts.o -no-pie -o mainSumInts
```

# Exercícios

2. Remova o *.globl sumInts* da função sumInts e compile novamente. O que acontece? Para que serve o *.globl sumInts*?

# Retorno

- Por convenção, o valor de retorno é enviado através do registrador **rax**

# Exercício

3. Modifique a função criada para a seguinte assinatura
   - int sumInts(int valor1, int valor2, int* soma);
     - O resultado é armazenado no endereço apontado por *soma
     - A função retorna
       - 0 caso tudo ocorra normalmente
       - 1 em caso de overflow
     - No *main* imprima o resultado caso tudo ocorra bem, ou imprima “overflow” caso a função retorne esse código de erro.
     - Leia sobre a instrução ADD no manual da Intel e sobre os bits em eflags que podem ser alterados pela instrução
       - Pesquise o jump condicional referente aos bits relevantes do eflags

# Integrando com C

- Você pode chamar sua função *assembly* diretamente em um programa em C
  - **Garanta que você seguiu estritamente as convenções**
    - As convenções são diferentes se você está compilando o programa para 32 ou 64 bits, Windows ou Linux, ...
- Utilize a palavra-chave ***extern***
  - Indica que a função é definida em outro arquivo, e que é papel do *linkeditor* ligá-la ao arquivo

    *extern tipoRetorno rotuloFuncao(tipo param1, tipo param2, …);*

# Exemplo

```c
#include<stdio.h>

extern int sumInts(int v1, int v2, int* ret);

int main(){
	int valor1, valor2, resultado, overflow;
	printf("Digite os valores: ");
	scanf("%d %d", &valor1, &valor2);
	overflow = sumInts(valor1, valor2, &resultado);
	if(!overflow){
		printf("O resultado eh %d\n", resultado);
	}else{
		printf("Ocorreu um overflow!\n");
	}

	return 0;
}
```

UDESC

# Integrando com C

- Para uma solução mais “elegante” você pode criar um arquivo .h para sua função assembly e simplesmente importar o .h para os seus programas
  - Os programas não saberão que a função vem de um arquivo objeto montado a partir de um fonte assembly

# Exercícios

4. Crie **uma função** que **retorna** o enésimo número da sequência de Fibonacci. Considere que N é passado como parâmetro. Crie um *main* em assembly que pede o valor de N para o usuário repetidas vezes, e chama a função de Fibonacci passando esse valor de N. O programa deve exibir o número retornado pela função na tela. O programa termina quando o usuário digitar um valor negativo para N.

5. Crie um função que recebe o endereço de uma string, e a converte os caracteres entre [a-z] para caixa alta, deixando os demais inalterados. Crie um *main* em assembly que solicita uma string do usuário e chama essa função, para que então a string seja impressa na tela.
   - Pesquise sobre as operações and, or e xor
   - Veja que ao realizar um *and* do caractere com *0xDF* o convertemos para caixa alta
     - Seção 12.1 Plantz (2011)

6. Faça o mesmo que no exercício anterior, mas agora crie uma função que converta para caixa baixa
   - *or* com *0x20*

7. Crie **programas em C** que invoquem as funções criadas nos exercícios anteriores.

**8. Desafio:** Crie uma função que converta para caixa alta. A função deve carregar os caracteres de 8 em 8 para algum registrador de 64 bits, e os converter de uma vez para caixa alta (potencialmente 8x mais rápido!).

# Referências


- Bob Plantz. **Introduction to Computer Organization: A Guide to X86-64 Assembly Language and GNU/Linux.** 2011.
- **Intel® 64 and IA-32 Architectures Software Developer's Manual.** Intel, 2019.
- D. Patterson; J. Henessy. **Organização e Projeto de Computadores**: **a Interface Hardware/Software.** 5a Edição. Elsevier Brasil, 2017.
- STALLINGS, W. **Arquitetura e Organização de Computadores.** 10 ed. Prentice Hall. São Paulo, 2018.
- M. Matz, J. Hubička, A. Jaeger, M. Mitchell. **System V Application Binary Interface AMD64 Architecture Processor Supplement.** 2014.