

# Sistemas Hardware-Software

Aula 03 – Arquitetura x86-64

**Engenharia**

Fabio Lubacheski

Maciel C. Vidal

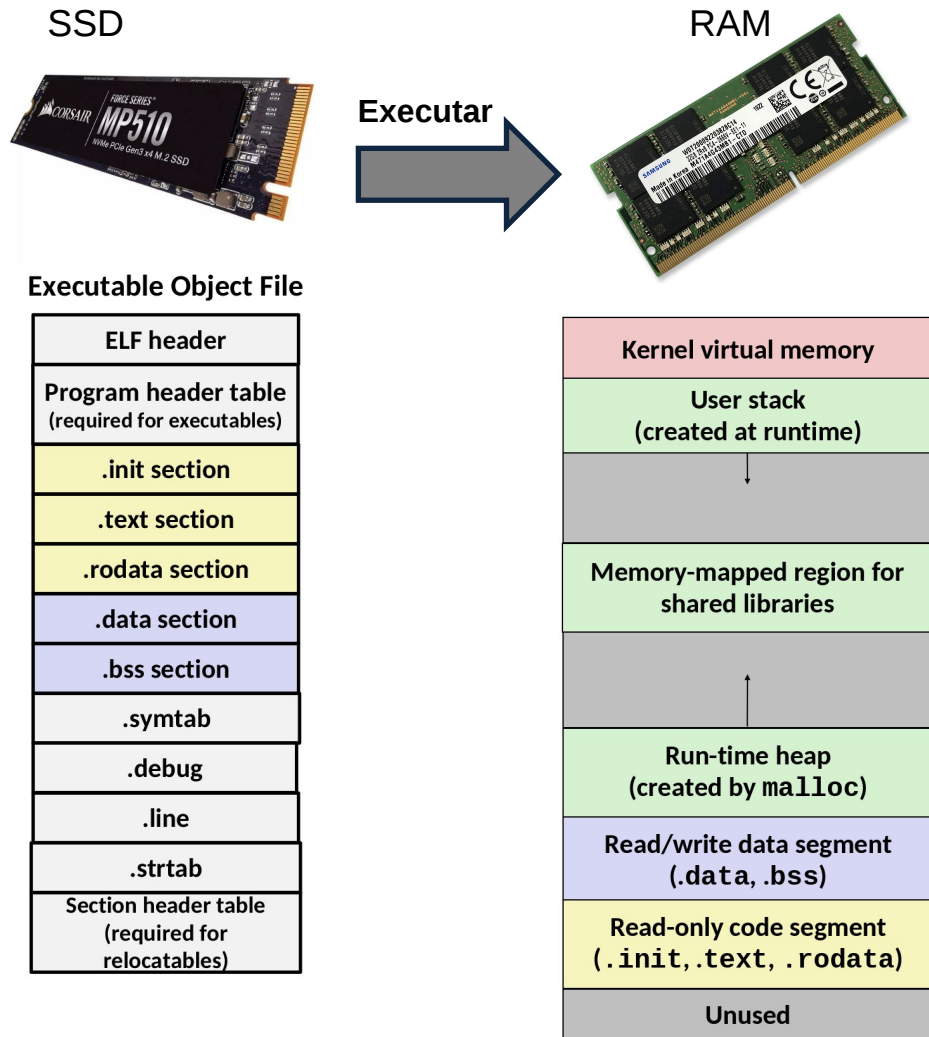
Igor Montagner

Fábio Ayres

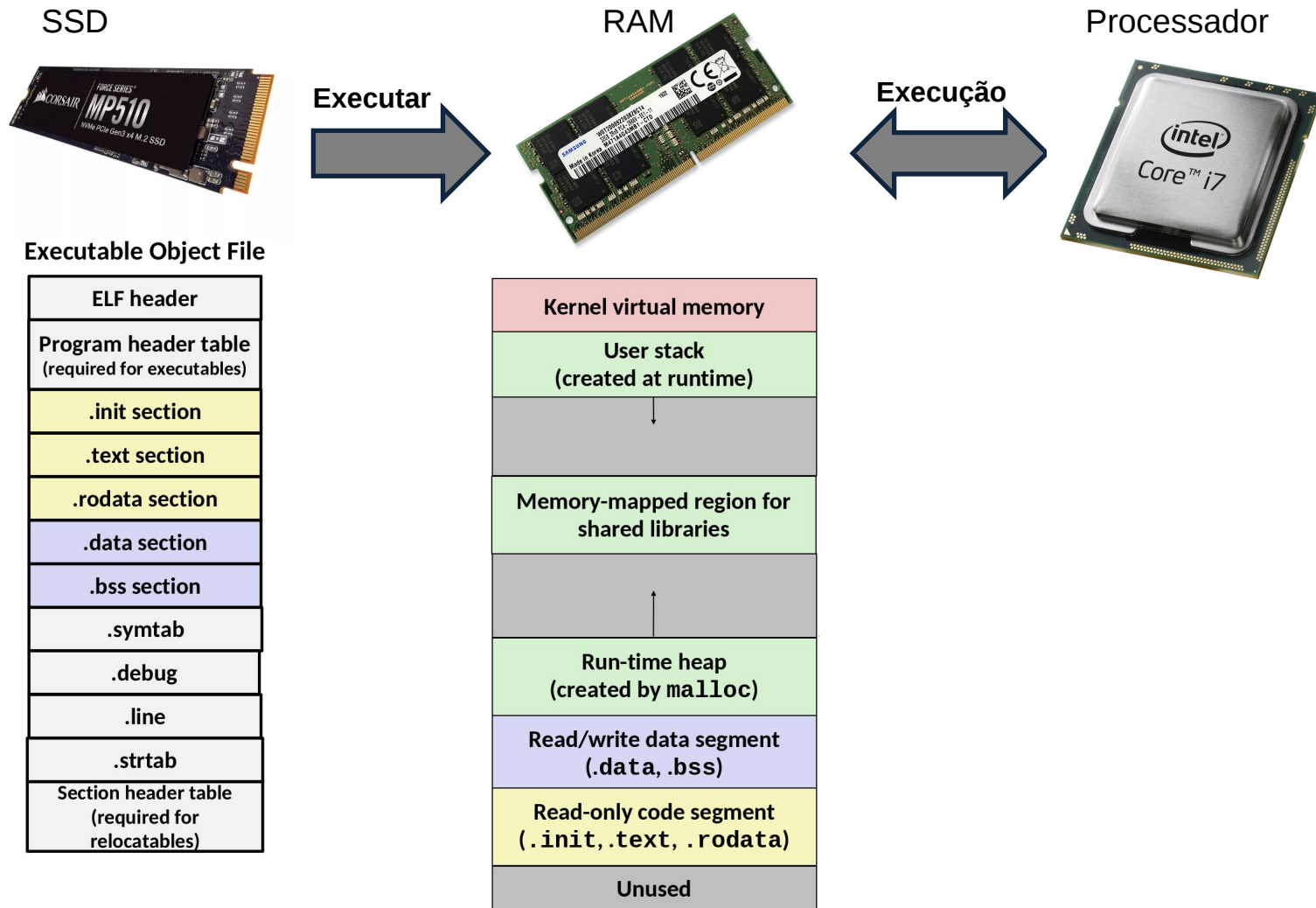
# Aula passada

- Representação em memória de vários tipos em C
- Estrutura de um arquivo executável
  - **.text** guarda nosso código
  - **.data** guarda globais inicializadas
  - **.rodata** guarda constantes
  - **.bss** reserva espaço para globais não inicializadas
- Variáveis locais só existem em tempo de execução

# Aula passada



# Aula de hoje



# Arquitetura x86-64

# Processadores Intel x86

- Dominam o mercado
  - Aprox 80% de market share de PCs!
- **Linhas**
  - Core i3: entry-level
  - Core i5: mainstream
  - Core i7: high-end
  - Core i9: very high-end
  - Core m: mobile (tablets)
  - Xeon: servidores e estações de trabalho
- **Arquitetura: Complex-instruction-set computer (CISC)**
  - Leia o texto abaixo sobre comparação da arquitetura **CISC x RISC**  
<https://diveintosystems.org/book/C5-Arch/index.html>

# Evolução dos processadores Intel/AMD

Nome	Data	Transistores	MHz
<b>8086</b>	<b>1978</b>	<b>29K</b>	<b>5-10</b>
<ul style="list-style-type: none"><li>• Primeiro processador Intel 16-bit</li><li>• Espaço de endereçamento: 1MB</li></ul>			
<b>386</b>	<b>1985</b>	<b>275K</b>	<b>16-33</b>
<ul style="list-style-type: none"><li>• Primeiro processador Intel 32-bit (IA32)</li><li>• Adicionou modo de endereçamento “flat”, capaz de rodar Unix</li></ul>			
<b>Pentium 4E</b>	<b>2004</b>	<b>125M</b>	<b>2800-3800</b>
<ul style="list-style-type: none"><li>• Primeiro processador Intel 64-bit (x86-64)</li></ul>			
<b>Core 2</b>	<b>2006</b>	<b>291M</b>	<b>1060-3500</b>
<ul style="list-style-type: none"><li>• Primeiro processador Intel multi-core</li></ul>			
<b>Core i7</b>	<b>2008</b>	<b>731M</b>	<b>1700-3900</b>
<ul style="list-style-type: none"><li>• Quad-core</li></ul>			

Retro-compatível desde o processador **8086**, de 1978!

# Definições

**Arquitetura** (também conhecida como **ISA: Instruction Set Architecture**):

- registradores, instruções
- Exemplos de ISAs:
  - Intel: x86, IA32, Itanium, x86-64
  - ARM
  - PowerPC

**Microarquitetura:** Implementação da arquitetura

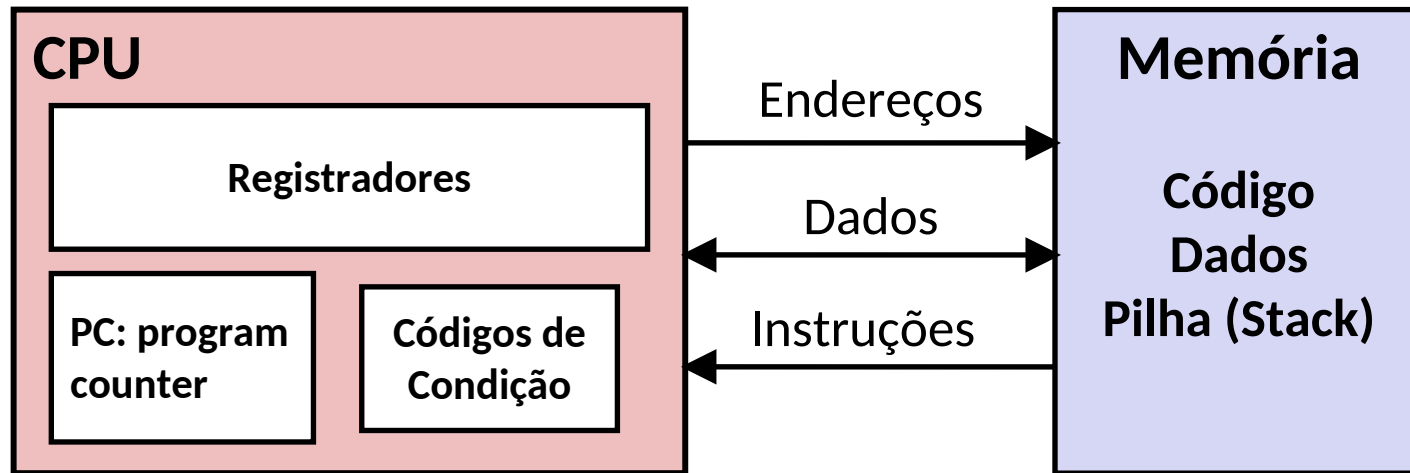
- Tamanho de cache, número de cores, frequência de clock

**Código:**

- **Código de máquina:** sequência de bytes que o processador executa
- **Código assembly:** representação textual mais “amigável” do código de máquina



# A visão do programador



## PC: Program counter

%rip: Endereço da próxima instrução

## Registradores

Dados de uso muito frequente

## Códigos de condição

Informação sobre o resultado das operações aritméticas ou lógicas mais recentes

## Memória

Um vetor de bytes

Armazena código e dados

Armazena estado atual do programa (pilha)

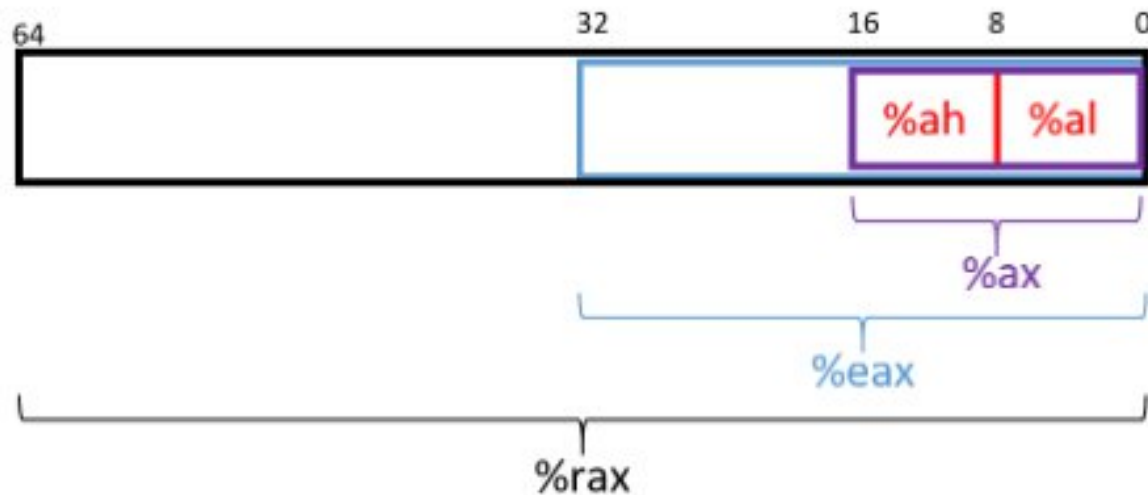
# Registradores inteiros x86-64

64 bits	32 bits	16 bits	8 bits
<code>%rax</code>	<code>%eax</code>	<code>%ax</code>	<code>%al</code>
<code>%rbx</code>	<code>%ebx</code>	<code>%bx</code>	<code>%bl</code>
<code>%rcx</code>	<code>%ecx</code>	<code>%cx</code>	<code>%cl</code>
<code>%rdx</code>	<code>%edx</code>	<code>%dx</code>	<code>%dl</code>
<code>%rdi</code>	<code>%edi</code>	<code>%di</code>	<code>%dil</code>
<code>%rsi</code>	<code>%esi</code>	<code>%si</code>	<code>%sil</code>
<code>%rsp</code>	<code>%esp</code>	<code>%sp</code>	<code>%spl</code>
<code>%rbp</code>	<code>%ebp</code>	<code>%bp</code>	<code>%bpl</code>

64 bits	32 bits	16 bits	8 bits
<code>%r8</code>	<code>%r8d</code>	<code>%r8w</code>	<code>%r8b</code>
<code>%r9</code>	<code>%r9d</code>	<code>%r9w</code>	<code>%r9b</code>
<code>%r10</code>	<code>%r10d</code>	<code>%r10w</code>	<code>%r10b</code>
<code>%r11</code>	<code>%r11d</code>	<code>%r11w</code>	<code>%r11b</code>
<code>%r12</code>	<code>%r12d</code>	<code>%r12w</code>	<code>%r12b</code>
<code>%r13</code>	<code>%r13d</code>	<code>%r13w</code>	<code>%r13b</code>
<code>%r14</code>	<code>%r14d</code>	<code>%r14w</code>	<code>%r14b</code>
<code>%r15</code>	<code>%r15d</code>	<code>%r15w</code>	<code>%r15b</code>

# Registradores inteiros x86-64

A ISA (**Arquitetura**) fornece um mecanismo para acessar as várias partes de um registrador, permitindo acessar os 8 bytes (**%rax**), 4 bytes mais baixos (**%eax**), 2 bytes mais baixos (**%ax**), byte mais baixo (**%al**) e segundo byte mais baixo (**%ah**)



O compilador pode escolher registradores de componentes dependendo do tipo

# Código de funcao1

00000000000000006da <funcao1>:

```
6dd:    89 f8          mov    %edi,%eax
6df:    03            add    (%rsi),%eax
6e1:    c3            retq
```

# Código de funcao1

0000000000000000006da <funcao1>:

6dd: 89 f8

6df: 03

6e1: c3

mov

add

retq

%edi,%eax

(%rsi),%eax

O quê faz MOV

O quê significa esse ( )?

# Atividade prática

## **GDB: Parando programas e examinando registradores**

1. usar GDB para acompanhar a execução de um programa
2. examinar valores dos registradores

# Código de funcao2

```
0x1145 <+0>:    mov     0x2ec9(%rip),%eax
0x114b <+6>:    add     $0x1,%eax
0x114e <+9>:    mov     %eax,0x2ec0(%rip)
0x1154 <+15>:   add     %edi,%eax
0x1156 <+17>:   retq
```

# Código de funcao2

```
0x1145 <+0>:    mov    0x2ec9(%rip),%eax
0x114b <+6>:    add     $0x1,%eax
0x114e <+9>:    mov     %eax,0x2ec0(%rip)
0x1154 <+15>:   add     %edi,%eax
0x1156 <+17>:   retq
```

Quem  
é %rip?

O quê significa  
0x2ec0(%rip)?



# Movendo Dados

**movq Source, Dest**

Tipos de operandos:

- **Imediato (Immediate):** Constantes inteiras
  - Exemplo: **\$0x400**, **\$-533**
  - Não esqueça do prefixo '\$'
  - Codificado com 1, 2, ou 4 bytes
- **Registrador:** Um dos 16 registradores inteiros
  - Exemplo: **%rax**, **%r13**
- **Memória:** 8 bytes (por causa do sufixo 'q') consecutivos de memória, no endereço dado pelo registrador
  - Exemplo mais simples: **(%rax)**
  - Vários outros modos de endereçamento

# movq : Combinações de operandos

	Source	Dest	Src, Dest	C Analog
movq	Imm	Reg	movq \$0x4,%rax	temp = 0x4;
		Mem	movq \$-147, (%rax)	*p = -147;
	Reg	Reg	movq %rax,%rdx	temp2 = temp1;
		Mem	movq %rax, (%rdx)	*p = temp;
	Mem	Reg	movq (%rax),%rdx	temp = *p;

*Não é permitido fazer transferência direta memória-memória com uma única instrução*

# Modos simples de endereçamento

Normal (R)       $\text{Mem}[\text{Reg}[R]]$

- Registrador R especifica o endereço de memória

**`movq (%rcx), %rax`**

Deslocamento (Displacement)       $D(R)$        $\text{Mem}[\text{Reg}[R]+D]$

- Registrador R especifica início da região de memória
- Constante de deslocamento D especifica offset

**`movq 8(%rbp), %rdx`**

# E os tamanhos?

O tamanho do dado é especificado na instrução! MOV não converte tipos!

Usamos um sufixo com o tamanho do tipo:

**Q** = **quad** word (8 bytes)

**L** = **long** word (4 bytes)

**W** = **word** (2 bytes)

**B** = **byte** (1 bytes)

Também podemos ver o tamanho dos registradores usados!

# E os tamanhos?

Cuidado com acessos à memória!

```
movb $-1, (%rsp)
```

Copia um byte no endereço do topo da pilha.

```
movq $-1, (%rsp)
```

Copia 8 bytes no endereço do topo da pilha.

# Exemplo

```
void swap(long *xp, long *yp)
{
    long t0 = *xp;
    long t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
swap:
    movq    (%rdi), %rax
    movq    (%rsi), %rdx
    movq    %rdx, (%rdi)
    movq    %rax, (%rsi)
    ret
```

```

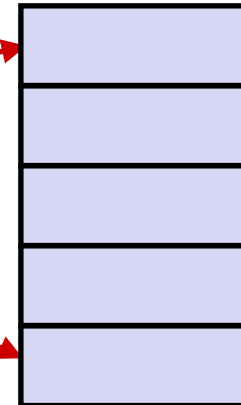
void swap
(long *xp, long *yp)
{
    long t0 = *xp;
    long t1 = *yp;
    *xp = t1;
    *yp = t0;
}

```

## Registers

%rdi	
%rsi	
%rax	
%rdx	

## Memory



### Register Value

%rdi	xp
%rsi	yp
%rax	t0
%rdx	t1

swap:

```

movq    (%rdi), %rax    # t0 = *xp
movq    (%rsi), %rdx    # t1 = *yp
movq    %rdx, (%rdi)    # *xp = t1
movq    %rax, (%rsi)    # *yp = t0
ret

```

## Registers

<b>%rdi</b>	<b>0x120</b>
<b>%rsi</b>	<b>0x100</b>
<b>%rax</b>	
<b>%rdx</b>	

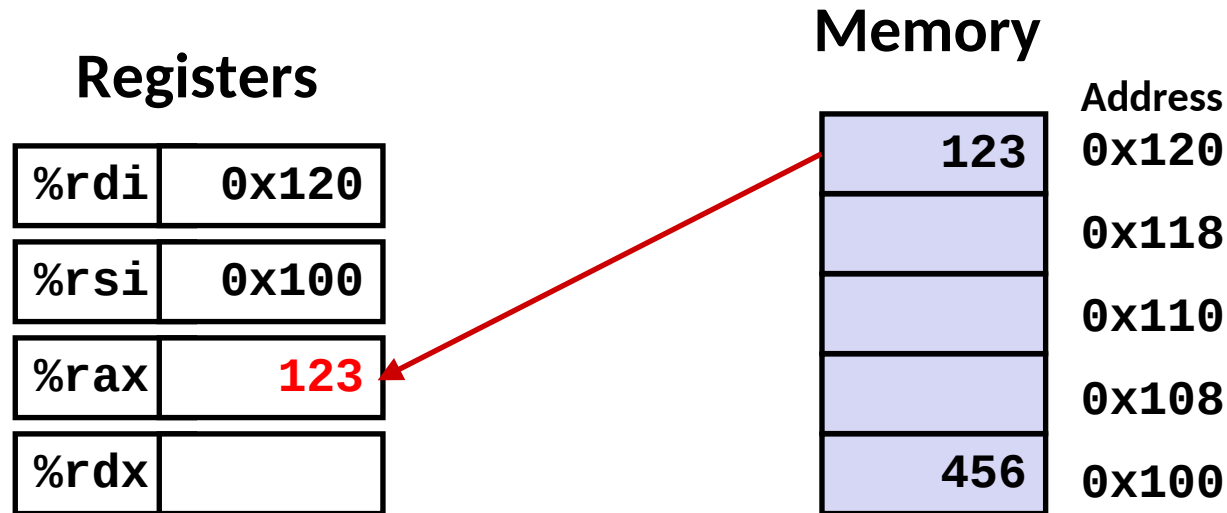
## Memory

Address
<b>123</b> <b>0x120</b>
<b>0x118</b>
<b>0x110</b>
<b>0x108</b>
<b>456</b> <b>0x100</b>

**swap:**

```
movq    (%rdi), %rax    # t0 = *xp
movq    (%rsi), %rdx    # t1 = *yp
movq    %rdx, (%rdi)    # *xp = t1
movq    %rax, (%rsi)    # *yp = t0
ret
```





swap:

```
movq    (%rdi), %rax    # t0 = *xp
movq    (%rsi), %rdx    # t1 = *yp
movq    %rdx, (%rdi)    # *xp = t1
movq    %rax, (%rsi)    # *yp = t0
ret
```

## Registers

%rdi	0x120
%rsi	0x100
%rax	123
%rdx	456

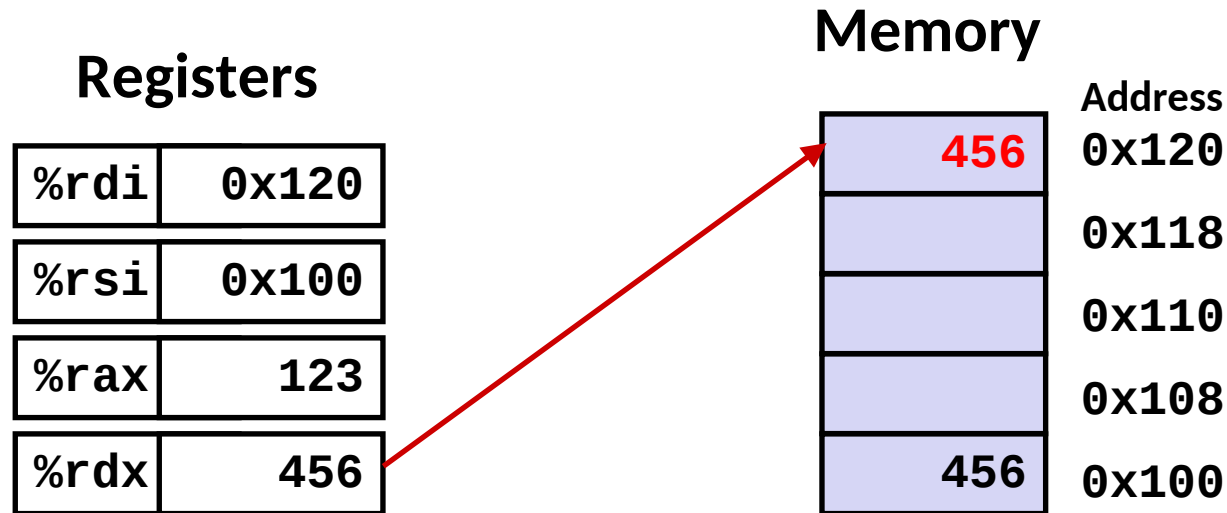
## Memory

Address
0x120
123
0x118
0x110
0x108
0x100
456



swap:

```
movq    (%rdi), %rax    # t0 = *xp
movq    (%rsi), %rdx    # t1 = *yp
movq    %rdx, (%rdi)    # *xp = t1
movq    %rax, (%rsi)    # *yp = t0
ret
```



**swap:**

```
movq    (%rdi), %rax    # t0 = *xp
movq    (%rsi), %rdx    # t1 = *yp
movq    %rdx, (%rdi)    # *xp = t1
movq    %rax, (%rsi)    # *yp = t0
ret
```

## Registers

%rdi	0x120
%rsi	0x100
%rax	123
%rdx	456

## Memory

Address
0x120
456
0x118
0x110
0x108
0x100
123

swap:

```
movq    (%rdi), %rax    # t0 = *xp
movq    (%rsi), %rdx    # t1 = *yp
movq    %rdx, (%rdi)    # *xp = t1
movq    %rax, (%rsi)    # *yp = t0
ret
```

# Modo de endereçamento completo

Forma geral: **D(Rb, Ri, S)**

Representa o valor  $\text{Mem}[\text{Reg}[\mathbf{Rb}] + \mathbf{S} * \text{Reg}[\mathbf{Ri}] + \mathbf{D}]$

Ou seja:

- O registrador **Rb** tem o endereço base
  - Pode ser qualquer registrador inteiro
- O registrador **Ri** tem um inteiro que servirá de índice
  - Qualquer registrador inteiro menos **%rsp**
- A constante **S** serve de multiplicador do índice
  - Só pode ser **1, 2, 4** ou **8**
- A constante **D** é o offset

# Exemplo

<b>%rdx</b>	<b>0xf000</b>
<b>%rcx</b>	<b>0x0100</b>

Expressão	Calculo de endereço	Resultado
<b>0x8(%rdx)</b>	<b>0xf000 + 0x8</b>	<b>0xf008</b>
<b>(%rdx,%rcx)</b>	<b>0xf000 + 0x100</b>	<b>0xf100</b>
<b>(%rdx,%rcx,4)</b>	<b>0xf000 + 4*0x100</b>	<b>0xf400</b>
<b>0x80(,%rdx,2)</b>	<b>2*0xf000 + 0x80</b>	<b>0x1e080</b>

# Atividade prática

## Endereçamento relativo e variáveis globais¶

1. Entender como variáveis globais são acessadas em Assembly

# Para Relembrar! Estrutura de um arquivo executável

## *Executable and Linkable Format (ELF)*

- Formato de arquivo executável em máquinas x86-64 Linux

## Seções importantes

- **.text**: código executável
- **.rodata**: constantes
- **.data**: variáveis globais pré-inicializadas
- **.bss**: variáveis globais não-inicializadas

## Outros formatos:

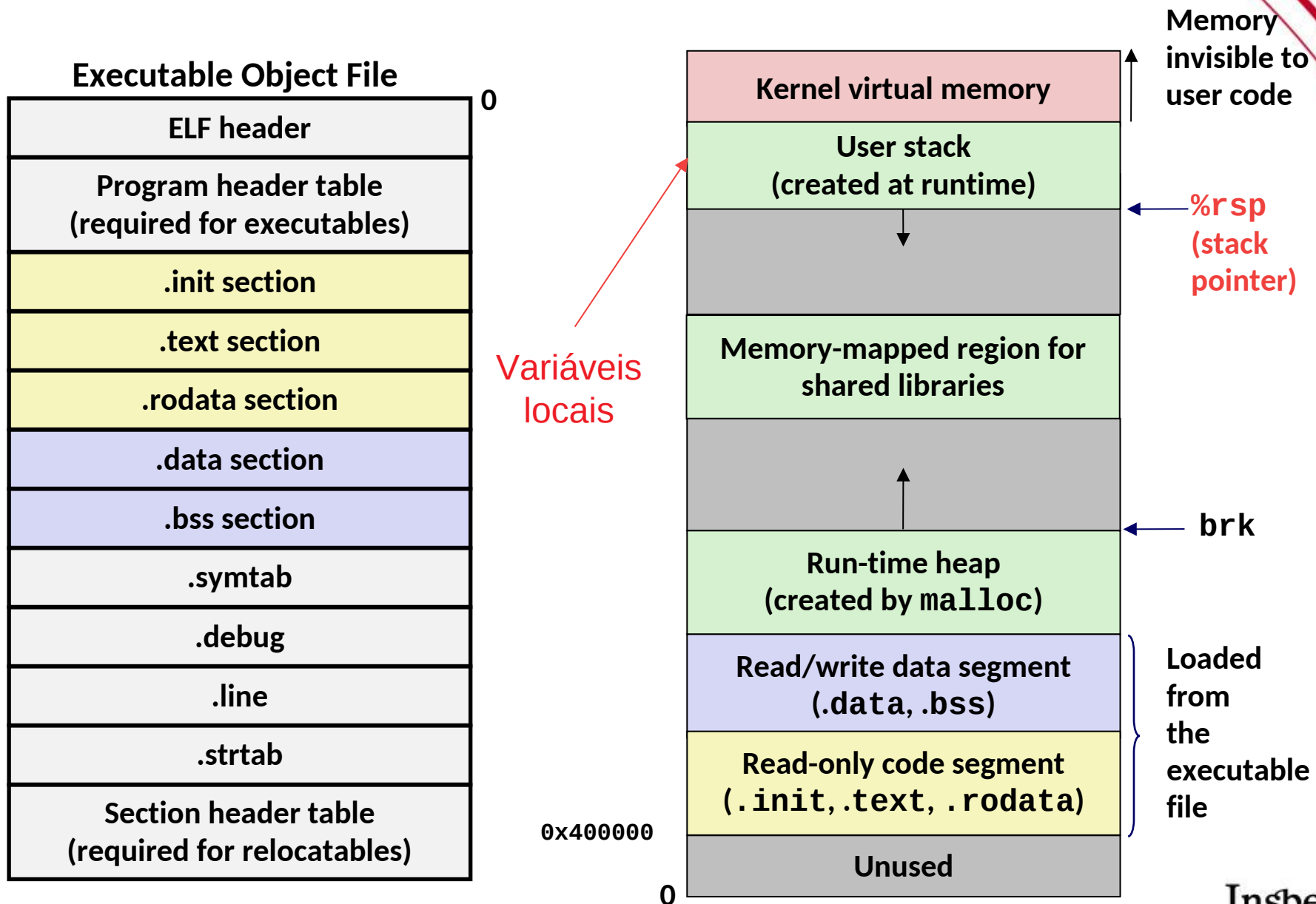
- *Portable Executable (PE)*: Windows
- *Mach-O*: Mac OS-X

## Executable Object File

ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)



# Para Relembrar! Executável na memória



# Insper

[www.insper.edu.br](http://www.insper.edu.br)