

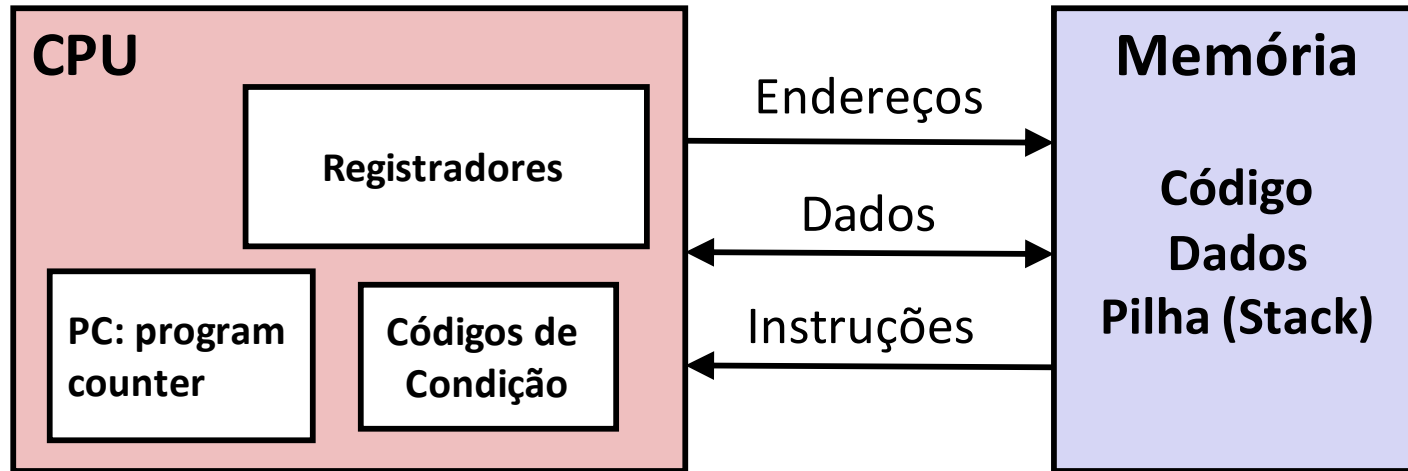
Sistemas Hardware-Software

Aula 5 – Programação em nível de máquina (I)

2020 – Engenharia

Igor Montagner [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br), Fábio Ayres

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

Usado para saltos condicionais

Memória

Um vetor de bytes

Armazena código e dados

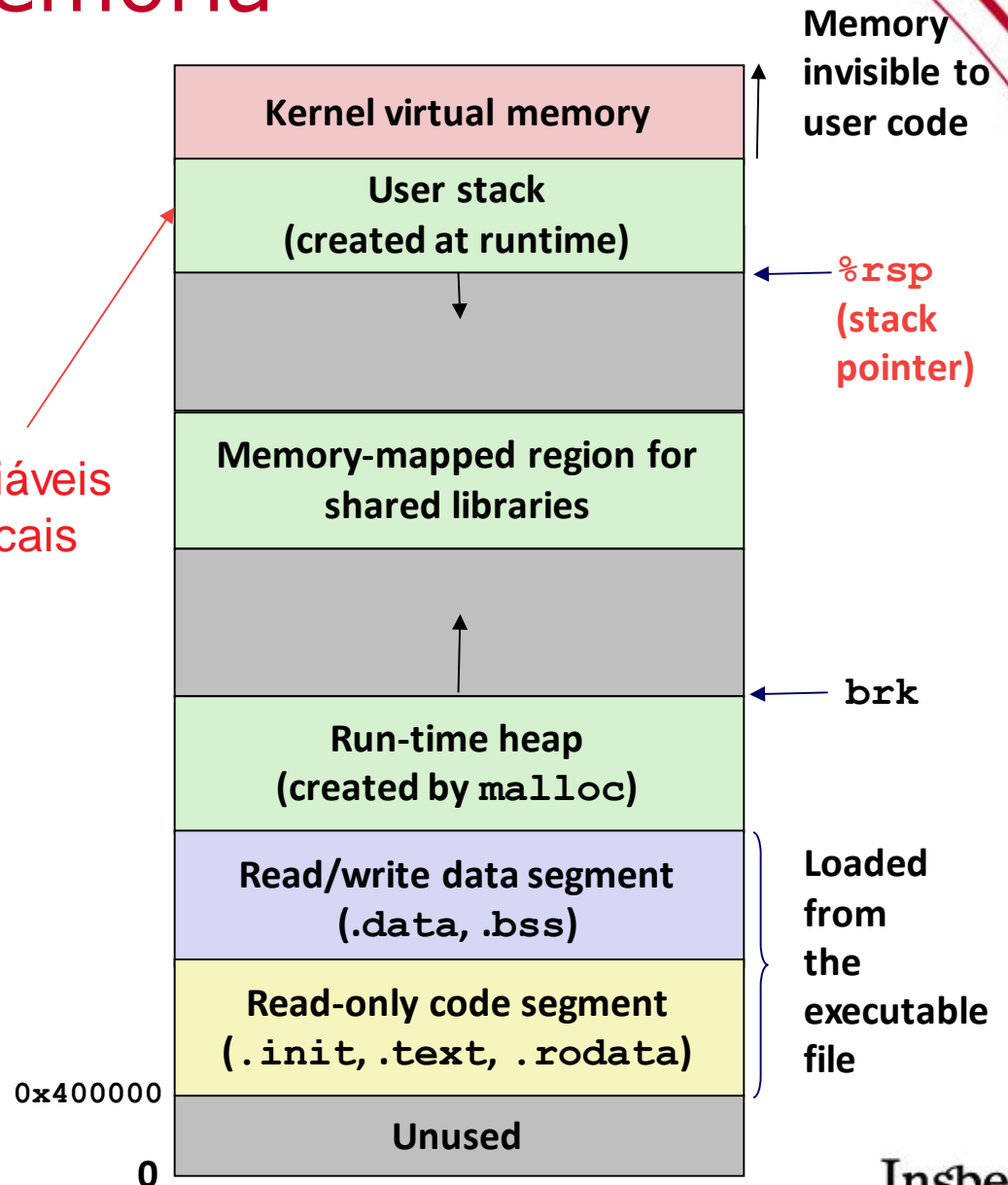
Armazena a pilha: essencial para usar funções

Executável na memória

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)

Variáveis
locais



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

Alguns 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`

Modo de endereçamento completo

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

Representa o valor $Mem[Reg[Rb] + S * Reg[Ri] + D]$

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

lea

“Prima” da instrução `mov`

- Mas ao invés de pegar dados da memória, apenas calcula o endereço de memória desejado
 - Daí vem o nome: *Load Effective Address*

Funcionamento: `lea Mem, Dst`

- **Mem**: operando de endereçamento da forma $D(Rb, Ri, S)$
 - Exemplo: `$0x4(%rax, %rbx, 4)`
- **Dst**: registrador destino
 - Exemplo: `%rsi`

Efeito final: calcula o endereço especificado pelo operando **Mem**, e armazena em **Dst**

lea versus mov

Exemplo:

```
lea $0x4(%rax, %rbx, 8), %rsi
```

Resulta em

$$R[\%rsi] = 4 + R[\%rax] + 8 \times R[\%rbx]$$

Compare com:

```
mov $0x4(%rax, %rbx, 8), %rsi
```

que resulta em

$$R[\%rsi] = M[4 + R[\%rax] + 8 \times R[\%rbx]]$$

(Ou seja, enquanto o **lea** só calcula o endereço, o **mov** vai lá buscar na memória)

Usos da instrução `lea`

`lea`: equivale em C a `p = &v[i]`

`mov`: equivale em C a `p = v[i]`

A instrução `lea` também é muito usada para fazer cálculos matemáticos simples, por exemplo:

```
long m12(long x) {  
    return x*12;  
}
```

```
leaq (%rdi,%rdi,2), %rax    # t <- x + x*2  
salq $2, %rax              # return t << 2
```

Vantagem: `lea` é muito rápida, faz contas com dois registradores e armazena em um terceiro!

Operações aritméticas simples

- Instruções de dois operandos:

<i>Instrução</i>	<i>Cálculo</i>
<code>addq</code>	<code>S, D D = D + S</code>
<code>subq</code>	<code>S, D D = D - S</code>
<code>imulq</code>	<code>S, D D = D * S</code>
<code>salq</code>	<code>S, D D = D << S # Tanto arit. como</code> <code>lógico.</code>
<code>sarq</code>	<code>S, D D = D >> S # Aritmético.</code>
<code>shrq</code>	<code>S, D D = D >> S # Lógico.</code>
<code>xorq</code>	<code>S, D D = D ^ S</code>
<code>andq</code>	<code>S, D D = D & S</code>
<code>orq</code>	<code>S, D D = D S</code>

Não há distinção entre signed e unsigned. (Porque?)

Operações aritméticas simples

- Instrução determina signed vs unsigned
- mul reg – multiplicação sem sinal de reg por %RAX
 - resultado armazenado em %RDX:%RAX
- imul reg – multiplicação com sinal de reg por %RAX
 - resultado armazenado em %RDX:%RAX
- Vale para divisão também!

Operações aritméticas simples

- Instruções de um operando operandos:

<i>Instrução</i>	<i>Cálculo</i>	
<code>incq</code>	<code>D</code>	<code>D = D + 1</code> # Incremento.
<code>decq</code>	<code>D</code>	<code>D = D - 1</code> # Decremento.
<code>negq</code>	<code>D</code>	<code>D = -D</code> # Negativo.
<code>notq</code>	<code>D</code>	<code>D = ~D</code> # Operador "not" bit-a-bit.

- Ver livro para mais instruções

Para referência completa:

<https://software.intel.com/en-us/articles/intel-sdm>

(somente 4684 páginas!)

Funções e engenharia reversa

Hoje o restante da aula será estúdio para a atividade do handout.

Sempre que encontrar uma instrução desconhecida busque por algo do tipo
“ASM x64” + instrução

Insper

www.insper.edu.br