



# RISC-V: The Free and Open RISC Instruction Set Architecture

Rodolfo Azevedo

MC404 – Organização Básica de Computadores e Linguagem de Montagem

<http://www.ic.unicamp.br/~rodolfo/mc404>

# Como codificar as instruções?

# Formatos das Instruções

Formato	Bits																																	
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
<b>R</b>	func7							rs2					rs1					func3			rd				opcode									
<b>I</b>	imm[11:0]											rs1					func3			rd				opcode										
<b>S</b>	imm[11:5]							rs2					rs1					func3			imm[4:0]				opcode									
<b>SB</b>	[12]	imm[10:5]							rs2					rs1					func3			Imm[4:1]				[11]	opcode							
<b>U</b>	imm[31:12]											rd					opcode																	
<b>UJ</b>	[20]	imm[10:1]										[11]	imm[19:12]											rd				opcode						

7 bits

128 instruções possíveis

# Funções

- Trechos de código que executam uma tarefa específica
- Organizadas separadamente para facilitar reuso e legibilidade
- Permite a divisão de tarefas entre desenvolvedores e uso de bibliotecas
- Precisam de convenções para operarem corretamente

Exemplo de função: Menor(int x, y)

# A função Menor(int x, y)

Menor:

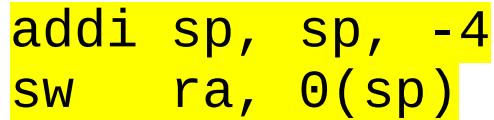
```
    blt a0, a1, fim
    add a0, zero, a1
```

fim:  
ret

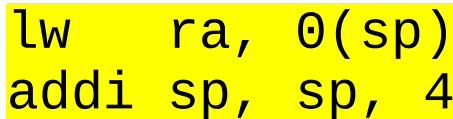
Main:

```
    addi a0, zero, 10
    addi a1, zero, 7
    call Menor
    addi a0, zero, 8
    addi a1, zero, 15
    call Menor
    ret
```

```
addi sp, sp, -4
sw    ra, 0(sp)
```



```
lw    ra, 0(sp)
addi sp, sp, 4
```



# Pilha

- Pilha cresce para baixo
- Registrador sp aponta para o último elemento da pilha
- Inclua sempre a sequência simétrica nas funções

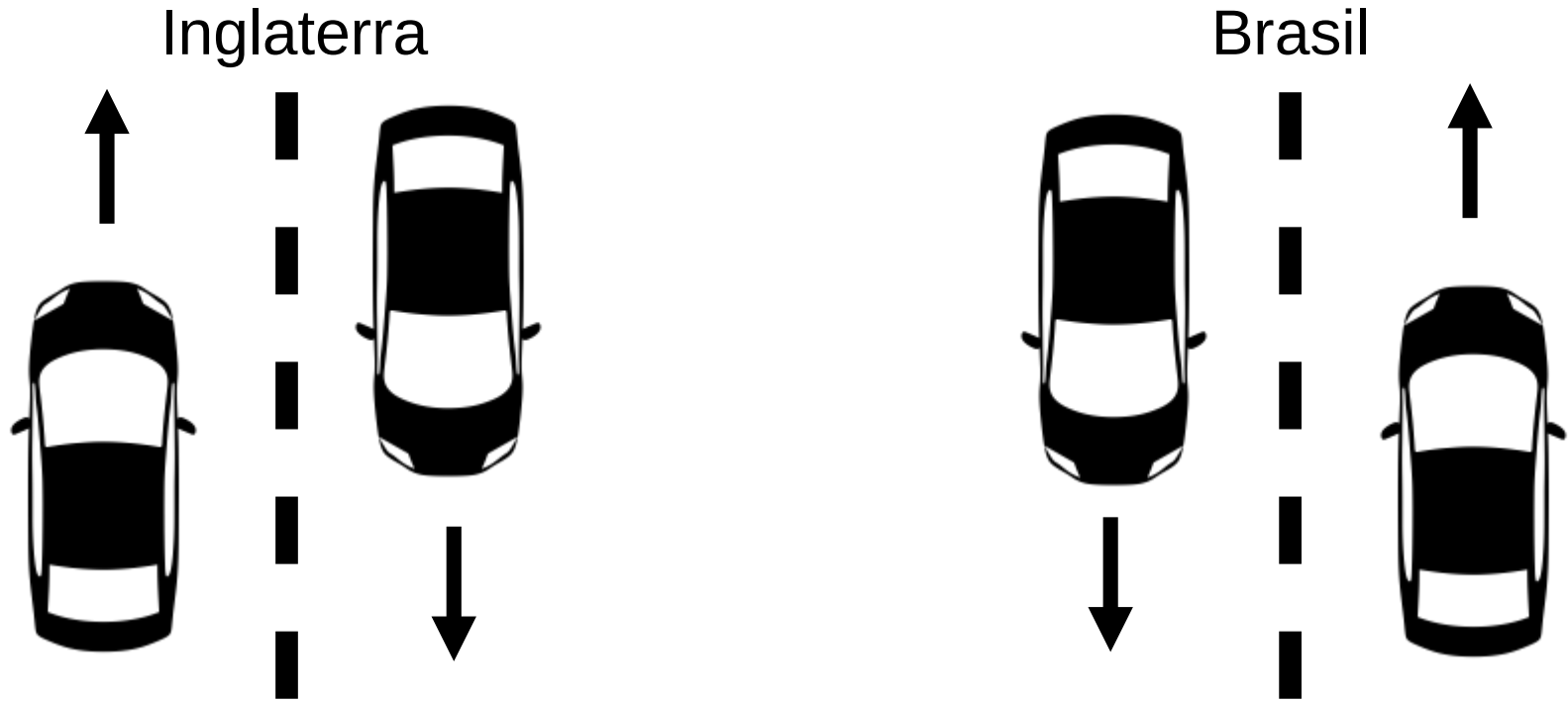
Início

```
addi sp, sp, -8  
sw   ra, 0(sp)  
sw   s0, 4(sp)
```

Final

```
lw   s0, 4(sp)  
lw   ra, 0(sp)  
addi sp, sp, 8
```

# Quem está dirigindo do lado certo?





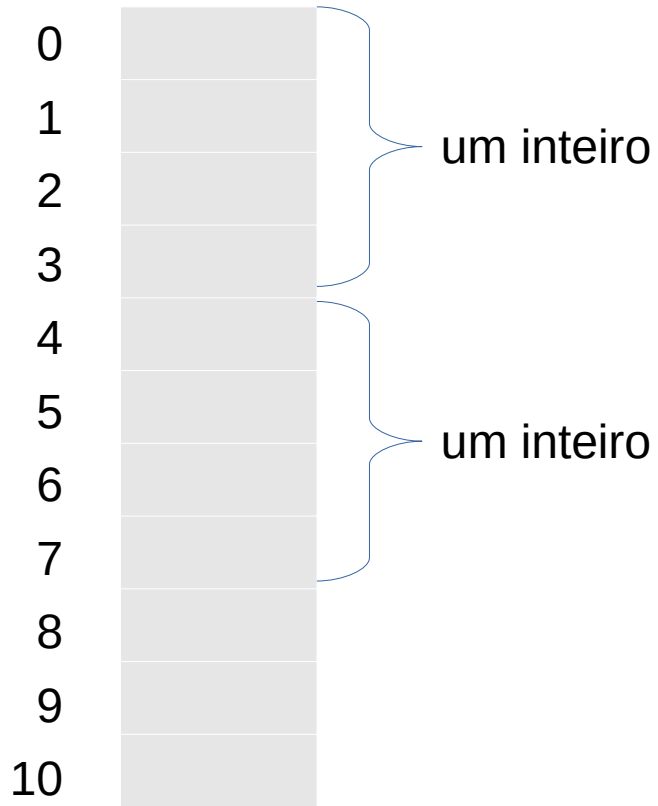
# Convenções estão em todos os lugares



# Registradores (novamente)

Register	ABI Name	Description	Saver
x0	Zero	Always zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary / alternate return address	Caller
x6–7	t1–2	Temporary	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function argument / return value	Caller
x12–17	a2–7	Function argument	Caller
x18–27	s2–11	Saved register	Callee
x28–31	t3–6	Temporary	Caller

# Endereçamento da Memória



# Organização da Memória

- Programas utilizam a memória para armazenar variáveis
- Normalmente, o compilador é responsável por organizar a memória e garantir o espaço para cada uma delas
- Em assembly, o programador é responsável por garantir o espaço das variáveis



# Pilha

- Pilha cresce para baixo
- Registrador `sp` aponta para o último elemento da pilha
- Inclua sempre a sequência simétrica nas funções

	Início		Final
<code>addi</code>	<code>sp, sp, -8</code>	<code>lw</code>	<code>s0, 4(sp)</code>
<code>sw</code>	<code>ra, 0(sp)</code>	<code>lw</code>	<code>ra, 0(sp)</code>
<code>sw</code>	<code>s0, 4(sp)</code>	<code>addi</code>	<code>sp, sp, 8</code>



# Heap

- Cresce para cima
- Armazena variáveis alocadas dinamicamente (ex.: malloc em C)
- Mais detalhes à frente



# Dados

- Armazena constantes e variáveis globais dos programas

```
.section .data
```

```
altura:
```

```
    .word 200
```

```
largura:
```

```
    .word 17
```

```
.section .text
```



# Ordem dos Bytes - Endian

- Temos duas formas de armazenar uma palavra (32 bits = 4 bytes) na memória
  - Big Endian: Byte mais significativo primeiro
  - Little Endian: Byte menos significativo primeiro
- Exemplo:
  - 0x12345678

Endereço	Big Endian	Little Endian
0	0x12	0x78
1	0x34	0x56
2	0x56	0x34
3	0x78	0x12



# Impacto

- Não importa o formato se você vai ler o que você mesmo escreveu
  - Mesmo programa
  - Mesmo processador
- RISC-V é little endian por padrão
- Cuidado quando for se comunicar com outros dispositivos que sejam Big Endian
  - Ex.: Cabeçalhos do protocolo TCP/IP são codificados como Big Endian

# Entrada e Saída

# Formatos de arquivos