



**Universidade de Brasília**

Departamento de Ciência da Computação

# **Introdução aos Sistemas Computacionais**

**Disciplina: 113468**

Prof. Marcus Vinicius Lamar



# Linguagem Assembly RISC-V

```
          addi    $t0,$a0,400    # beyond end of a[]
Loop:     beq     $a0,$t0,Exit
          lw      $t1, 0($a0)    # $t1=a[i]
          lw      $t2, 0($a1)    # $t2=b[i]
          add     $t1,$t1,$t2    # $t1=a[i] + b[i]
          sw      $t1, 0($a2)    # c[i]=a[i] + b[i]
          addi    $a0,$a0,4      # $a0++
          addi    $a1,$a1,4      # $a1++
          addi    $a2,$a2,4      # $a2++
          j       Loop
Exit:     jr      $ra
```



# Linguagem Assembly RISC-V

- Apresentada aqui como a linguagem de programação mais básica de um processador
  - Baseada na execução sequencial de instruções
  - Desvios explícitos a endereços na memória
  - Sem variáveis ou tipos
  - Sem estruturas de controle elaboradas
  - Sem chamadas automáticas a funções/ou procedimentos
  - Sem bibliotecas
- As instruções mais básicas estarão em vermelho
- Vide manual do RISC-V para o conjunto completo de instruções e detalhes de implementação

# Assembly RV32I

## ■ Instruções Aritméticas: + -

Obs.: Imm número de 12 bits em complemento de 2 com extensão de sinal

add t0, t1, t2                    # t0 = t1 + t2

sub t0, t1, t2                    # t0 = t1 - t2

addi t0, t1, Imm                # t0 = t1 + ExtensãoSinal{Imm}

# Assembly RV32I

## ■ Instruções Lógicas

*bitwise* = bit a bit

### Assembly Code

and s3, s1, s2  
or s4, s1, s2  
xor s5, s1, s2

### Source Registers

s1	1111	1111	1111	1111	0000	0000	0000	0000
s2	0100	0110	1010	0001	1111	0000	1011	0111

### Result

s3	0100	0110	1010	0001	0000	0000	0000	0000
s4	1111	1111	1111	1111	1111	0000	1011	0111
s5	1011	1001	0101	1110	1111	0000	1011	0111

### Source Values

s1	0000	0000	0000	0000	0000	0000	1111	1111
imm	1111	1111	1111	1111	1111	1010	0011	0100

← sign extended →

### Assembly Code

andi s2, s1, 0xA34  
ori s3, s1, 0xA34  
xori s4, s1, 0xA34

### Result

s2	0000	0000	0000	0000	0000	0000	0011	0100
s3	1111	1111	1111	1111	1111	1010	1111	1111
s5	1111	1111	1111	1111	1111	1010	1100	1011



# Assembly RV32I

## ■ Instruções de deslocamento de bits

### Assembly Code

```
sll s3, s1, s2
srl s4, s1, s2
sra s5, s1, s2
```

### Source Values

s1	1111	0011	0000	0100	0000	0010	1010	1000
s2	0000	0000	0000	0000	0000	0000	0000	<b>1000</b>

### Result

s3	0000	0100	0000	0010	1010	1000	0000	0000
s4	0000	0000	1111	0011	0000	0100	0000	0010
s5	1111	1111	1111	0011	0000	0100	0000	0010

### Source Values

s1	1111	0011	0000	0000	0000	0010	1010	1000
Imm	0000	0000	0000	0000	0000	0000	0000	0100

### Assembly Code

```
slli s2, s1, 4
srli s3, s1, 4
srai s4, s1, 4
```

### Result

s2	0011	0000	0000	0000	0010	1010	1000	0000
s3	0000	1111	0011	0000	0000	0000	0010	1010
s4	1111	1111	0011	0000	0000	0000	0010	1010



# Assembly RV32I

## ■ Instruções acesso à memória:

Obs.: Imm número de 12 bits em complemento de 2

lw t0,Imm(t1)

# t0 = Memória[Imm+t1]

sw t0,Imm(t1)

# Memória[Imm+t1] = t0

lh t0,Imm(t1)

# t0 = ExtensãoSinal{Memória[Imm+t1]}

lhu t0,Imm(t1)

# t0 = ExtensãoZero{Memória[Imm+t1]}

sh t0,Imm(t1)

# Memória[Imm+t1] = t0[15:0]

lb t0,Imm(t1)

# t0 = ExtensãoSinal{Memória[Imm+t1]}

lbu t0,Imm(t1)

# t0 = ExtensãoZero{Memória[Imm+t1]}

sb t0,Imm(t1)

# Memória[Imm+t1] = t0[7:0]



# Assembly RV32I

## ■ Instruções diversas

Load Upper Immediate:

lui t0,Imm                      # t0=Imm<<12

Add upper Immediate to PC:

auipc t0,Imm                  # t0 = PC +Imm<<12

## ■ Instruções de Comparação

Set on Less Than

slt t0,t1,t2                    # t1<t2 ? t0=1 : t0=0

sltu t0,t1,t2                  # t1<t2 ? t0=1 : t0=0    argumentos sem sinal

slti t0,t1,Imm                # t1<Imm ? t0=1 : t0=0

sltiu t0,t1,Imm               # t1<Imm ? t0=1 : t0=0    argumentos sem sinal



# Assembly RV32I

Obs.: Label é um endereço de 32 bits

## ■ Instruções de Salto Incondicional

<code>jal t0, Label</code>	<code># t0=PC+4 e PC=Label</code>
<code>jalr t0,t1,Imm</code>	<code># t0=PC+4 e PC=t1+Imm&amp;(!1)</code>

## ■ Instruções de Salto Condicional

<code>beq t0,t1,Label</code>	<code># t0==t1 ? PC=Label : PC=PC+4</code>
<code>bne t0,t1,Label</code>	<code># t0!=t1 ? PC=Label : PC=PC+4</code>
<code>bge t0,t1,Label</code>	<code># t0&gt;=t1 ? PC=Label : PC=PC+4</code>
<code>bgeu t0,t1,Label</code>	<code># t0&gt;=t1 ? PC=Label : PC=PC+4</code>
<code>blt t0,t1,Label</code>	<code># t0&lt;t1 ? PC=Label : PC=PC+4</code>
<code>bltu t0,t1,Label</code>	<code># t0&lt;t1 ? PC=Label : PC=PC+4</code>



# Assembly RV32IM

## ■ Instruções Aritméticas: $\times \div$

`mul t0,t1,t2`    #  $t0 = \text{Low}\{t0 \times t1\}$  32 bits menos significativos

`mulh t0,t1,t2`    #  $t0 = \text{High}\{t0 \times t1\}$  32 bits mais significativos

`mulhu t0,t1,t2`    #  $t0 = \text{High}\{t0 \times t1\}$  32 bits mais significativos, argumentos sem sinal

`div t0,t1,t2`    #  $t0 = \text{Quociente } t1/t2$

`divu t0,t1,t2`    #  $t0 = \text{Quociente } t1/t2$  argumentos sem sinal

`rem t0,t1,t2`    #  $t0 = \text{Resto } t1/t2$

`remu t0,t1,t2`    #  $t0 = \text{Resto } t1/t2$  argumentos sem sinal

