
**Universidade Federal de Pelotas**
**Centro de Desenvolvimento Tecnológico**
**Bacharelado em Ciência da Computação**
**Engenharia de Computação**

# Arquitetura e Organização de Computadores I

## Prática

## Aula 4

### Revisão, Acesso à Memória

**Prof. Guilherme Corrêa**
gcorrea@inf.ufpel.edu.br

**Prof. Bruno Zatt**

# Revisão

## MIPS: Registradores

| Registrador | Nome | Uso (convenção) |
|---|---|---|
| $0 | $zero | Zero |
| $1 | $at | *Assembler Temporary* |
| $2, $3 | $v0, $v1 | Valor de retorno de subrotina |
| $4 – $7 | $a0 – $a3 | Argumentos de subrotina |
| $8 – $15 | $t0 – $t7 | Temporários (locais à função) |
| $16 – $23 | $s0 – $s7 | Salvos (não alterados na função) |
| $24, $25 | $t8, $t9 | Temporários |
| $26, $27 | $k0, $k1 | Kernel (reservado para SO) |
| $28 | $gp | *Global Pointer* |
| $29 | $sp | *Stack Pointer* |
| $30 | $fp | *Frame Pointer* |
| $31 | $ra | Endereço de Retorno |

# Revisão

## Operações Lógicas (and, or, xor, nor)

**Tipo R**

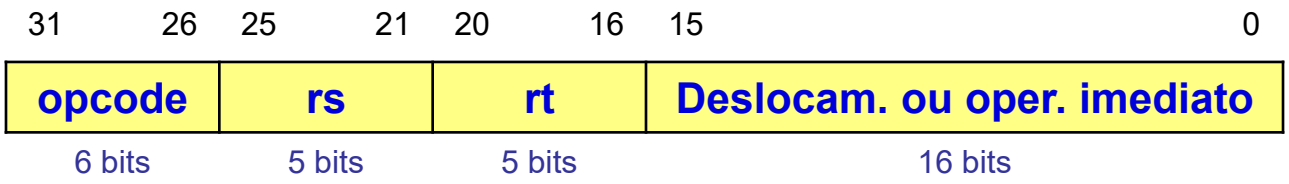
| 31 – 26 | 25 – 21 | 20 – 16 | 15 – 11 | 10 – 6 | 5 – 0 |
|---|---|---|---|---|---|
| **opcode** | **rs** | **rt** | **rd** | **shamt** | **funct** |
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

| | | | | | | |
|---|---|---|---|---|---|---|
| **and $t1, $zero, $t3** | **0H** | **0H** | **BH** | **9H** | **0H** | **24H** |
| | 000000 | 00000 | 01011 | 01001 | 00000 | 100100 |
| **or $t0, $t1, $t2** | **0H** | **9H** | **AH** | **8H** | **0H** | **25H** |
| | 000000 | 01001 | 01010 | 01000 | 00000 | 100101 |
| **xor $t1, $t2, $t3** | **0H** | **AH** | **BH** | **9H** | **0H** | **26H** |
| | 000000 | 01010 | 01011 | 01001 | 00000 | 100110 |
| **nor $t0, $t0, $t3** | **0H** | **8H** | **BH** | **8H** | **0H** | **27H** |
| | 000000 | 01011 | 01011 | 01001 | 00000 | 100111 |

# Revisão

## Operações Lógicas (andi, ori, xori)

**Tipo I**




**andi $t1, $t0, 0xA0A0**

| $C_H$ | $8_H$ | $9_H$ | $A0A0_H$ |
|---|---|---|---|
| 001100 | 01000 | 01001 | 1010000010100000 |

**ori $t0, $zero, 0xFFFF**

| $D_H$ | $0_H$ | $8_H$ | $FFFF_H$ |
|---|---|---|---|
| 001101 | 00000 | 01000 | 1111111111111111 |

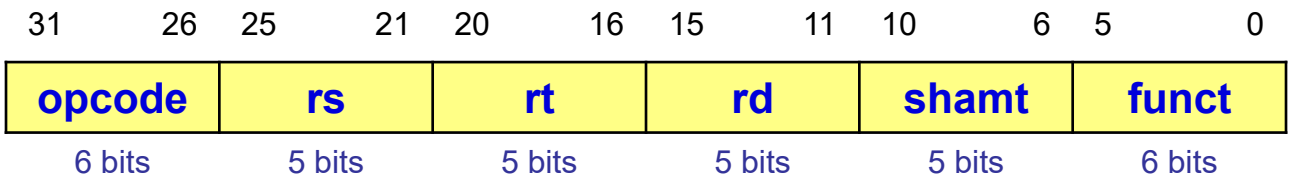
**xori $t1, $t1, 0x538**

| $E_H$ | $9_H$ | $9_H$ | $538_H$ |
|---|---|---|---|
| 001110 | 01001 | 01001 | 0000010100111000 |

# Revisão

## Operações Lógicas (sll, slr)

**Tipo R**




**sll $t0, $t1, 12**

| 0H | 0H | 9H | 8H | CH | 0H |
|---|---|---|---|---|---|
| 000000 | 00000 | 01001 | 01000 | 01100 | 000000 |

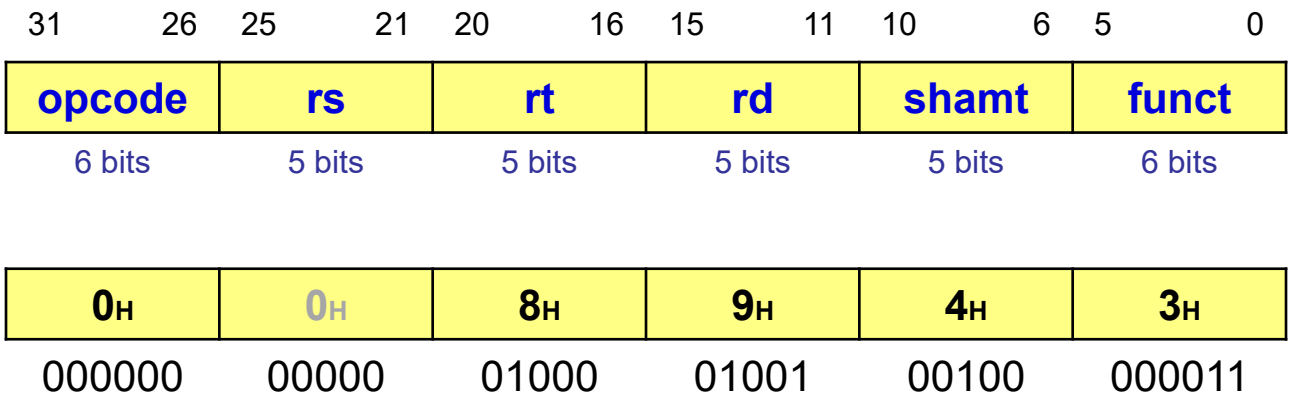
**srl $t5, $t5, 28**

| 0H | 0H | DH | DH | 1CH | 2H |
|---|---|---|---|---|---|
| 000000 | 00000 | 01101 | 01101 | 11100 | 000010 |

# Revisão

## Operações Aritméticas (sra)

**Tipo R**



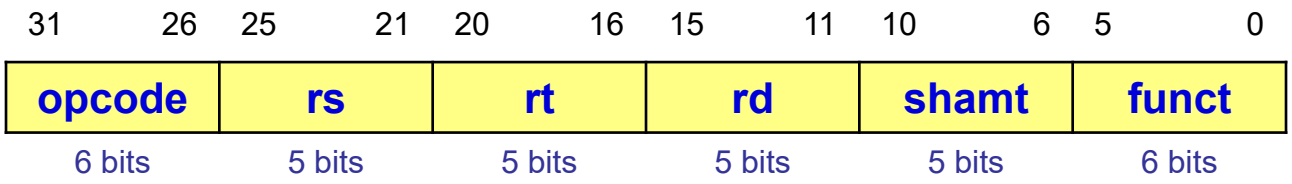

**sra $t1, $t0, 4**

| 0H | 0H | 8H | 9H | 4H | 3H |
|---|---|---|---|---|---|
| 000000 | 00000 | 01000 | 01001 | 00100 | 000011 |

# Revisão

## Operações Aritméticas (add, sub, addu, subu)

**Tipo R**




**add $t2, $t0, $t1**

| 0H | 8H | 9H | AH | 0H | 20H |
|---|---|---|---|---|---|
| 000000 | 01000 | 01001 | 01010 | 00000 | 100000 |

**sub $t2, $t0, $t1**

| 0H | 8H | 9H | AH | 0H | 22H |
|---|---|---|---|---|---|
| 000000 | 01000 | 01001 | 01010 | 00000 | 100010 |

**addu $t2, $t0, $t1**

| 0H | 8H | 9H | AH | 0H | 21H |
|---|---|---|---|---|---|
| 000000 | 01000 | 01001 | 01010 | 00000 | 100001 |

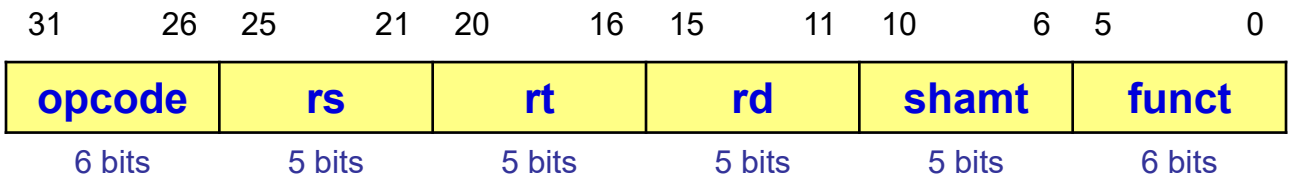
**subu $t2, $t0, $t1**

| 0H | 8H | 9H | AH | 0H | 23H |
|---|---|---|---|---|---|
| 000000 | 01000 | 01001 | 01010 | 00000 | 100011 |

# Revisão

## Operações Aritméticas (mult, multu, mfhi, mflo)

**Tipo R**

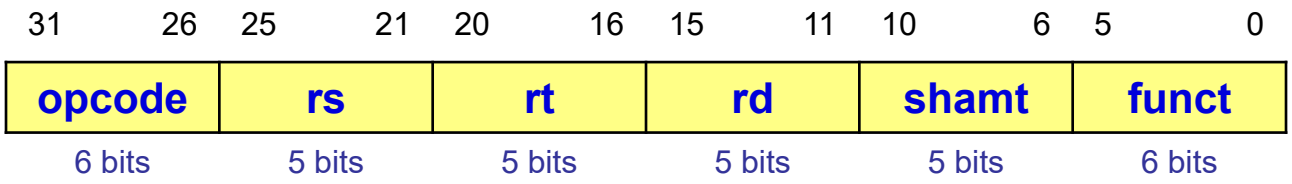



| | | | | | | |
|---|---|---|---|---|---|---|
| **mult $t0, $t1** | **0H** | **8H** | **9H** | **0H** | **0H** | **18H** |
| | 000000 | 01000 | 01001 | 00000 | 00000 | 011000 |
| **multu $t0, $t1** | **0H** | **8H** | **9H** | **0H** | **0H** | **19H** |
| | 000000 | 01000 | 01001 | 00000 | 00000 | 011001 |
| **mfhi $t2** | **0H** | **0H** | **0H** | **AH** | **0H** | **10H** |
| | 000000 | 00000 | 00000 | 01010 | 00000 | 010000 |
| **mflo $t3** | **0H** | **0H** | **0H** | **BH** | **0H** | **12H** |
| | 000000 | 00000 | 00000 | 01011 | 00000 | 010010 |

# Revisão

## Operações Aritméticas (div, divu, mfhi, mflo)

**Tipo R**

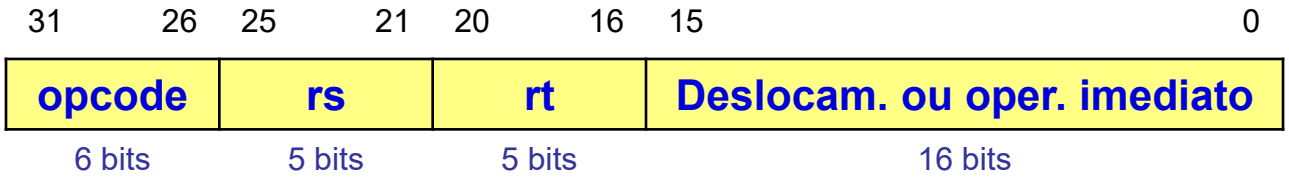



| | opcode | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|---|
| **div $t0, $t1** | $0_H$ | $8_H$ | $9_H$ | $0_H$ | $0_H$ | $1A_H$ |
| | 000000 | 01000 | 01001 | 00000 | 00000 | 011010 |
| **divu $t0, $t1** | $0_H$ | $8_H$ | $9_H$ | $0_H$ | $0_H$ | $1B_H$ |
| | 000000 | 01000 | 01001 | 00000 | 00000 | 011011 |
| **mfhi $t2** | $0_H$ | $0_H$ | $0_H$ | $A_H$ | $0_H$ | $10_H$ |
| | 000000 | 00000 | 00000 | 01010 | 00000 | 010000 |
| **mflo $t3** | $0_H$ | $0_H$ | $0_H$ | $B_H$ | $0_H$ | $12_H$ |
| | 000000 | 00000 | 00000 | 01011 | 00000 | 010010 |

# Revisão

## Operações Aritméticas (addi, addiu)

**Tipo I**




**addi $t2, $t0, 0x12**

| $8_H$ | $8_H$ | $A_H$ | $12_H$ |
|---|---|---|---|
| 001000 | 01000 | 01010 | 0000000000010010 |

**addiu $t2, $t0, 0x12**

| $9_H$ | $8_H$ | $A_H$ | $12_H$ |
|---|---|---|---|
| 001001 | 01000 | 01010 | 0000000000010010 |

# Acesso à Memória

## MIPS: estrutura básica

- Banco de registradores (32 registradores)
- Program Counter (PC)
- Memória (dados, instruções)
- Unidade Lógica e Aritmética (ULA)

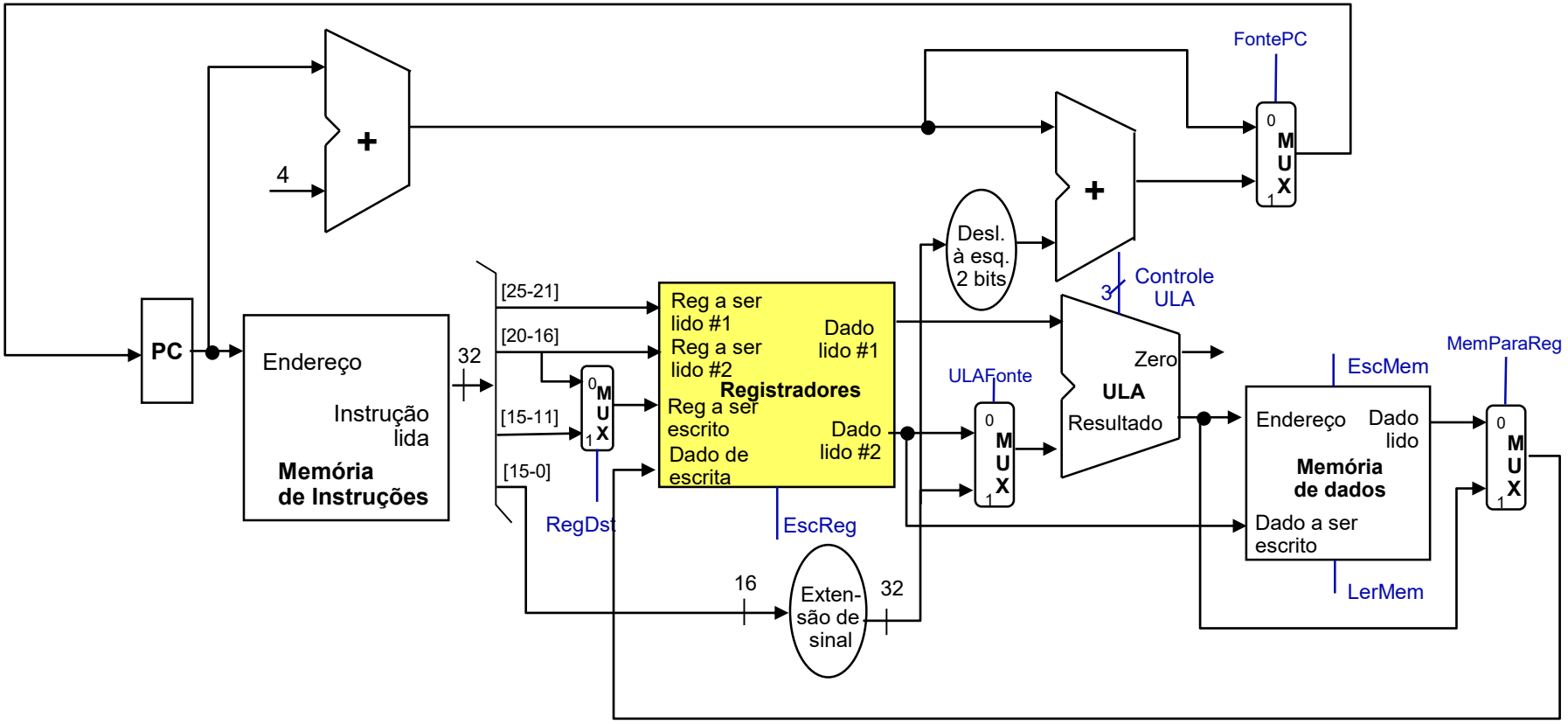
# Acesso à Memória

## MIPS: estrutura básica

- Banco de registradores (32 registradores)
- Program Counter (PC)
- Memória (dados, instruções)
- Unidade Lógica e Aritmética (ULA)

# Acesso à Memória

## MIPS (monociclo)



# Acesso à Memória

## MIPS: Memória

- Memória de Instruções
- Memória de Dados
- Acessos à memória devem ser alinhados
  - Cada endereço aponta para um *byte*
  - Dados de 32 bits (*word*) precisam iniciar em endereços múltiplos de 4
  - Outros tamanhos de dados também são suportados (*halfword*, *bytes*)

# Acesso à Memória

- Por quê?

# Acesso à Memória

## Por quê?

- Os programas podem conter estruturas de dados **muito mais complexas** que números inteiros, por exemplo;
- Apenas **32 registradores** não são suficientes;
- Os registradores mantêm pequenas quantidades de dados acessadas **frequentemente**;
- *Arrays*, estruturas, variáveis pouco usadas devem ser armazenadas na **memória**.

# Acesso à Memória

## Como funciona?

- Precisamos de **instruções de transferência de dados** para ler e escrever dados na memória;
- Os dados são transferidos da memória para os registradores e vice-versa;
- Precisamos fornecer às instruções **endereços** para acessar a memória;
- Instruções para copiar dados da memória para os registradores: **LOAD**;
- Instruções para copiar dados dos registradores para a memória: **STORE**.

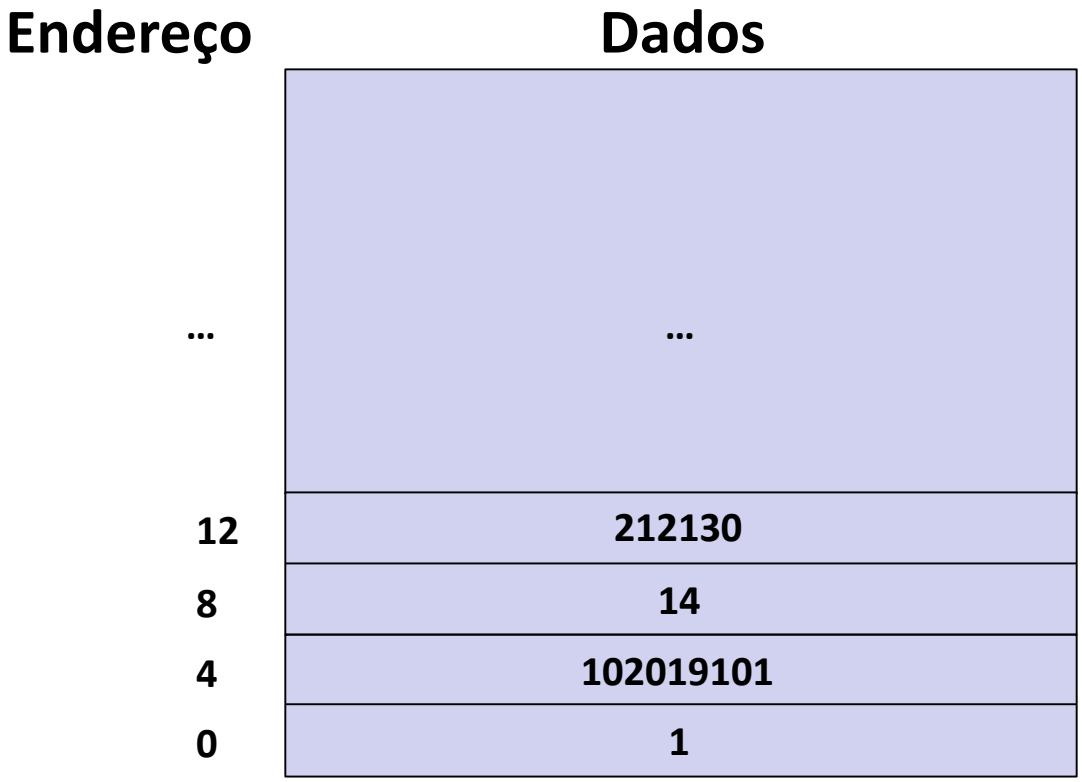
# Acesso à Memória

## Alinhamento da Memória

- No **MIPS**, os registradores têm 32 bits, de forma que a maior parte das operações é executada sobre *words* (conjunto de 4 *bytes* = 32 bits);
- Os endereçamentos são feitos a *bytes*, portanto o endereço de uma *word* deve considerar os 4 *bytes* dentro da *word*;
- Ou seja: ***words* precisam começar em endereços múltiplos de 4**!!

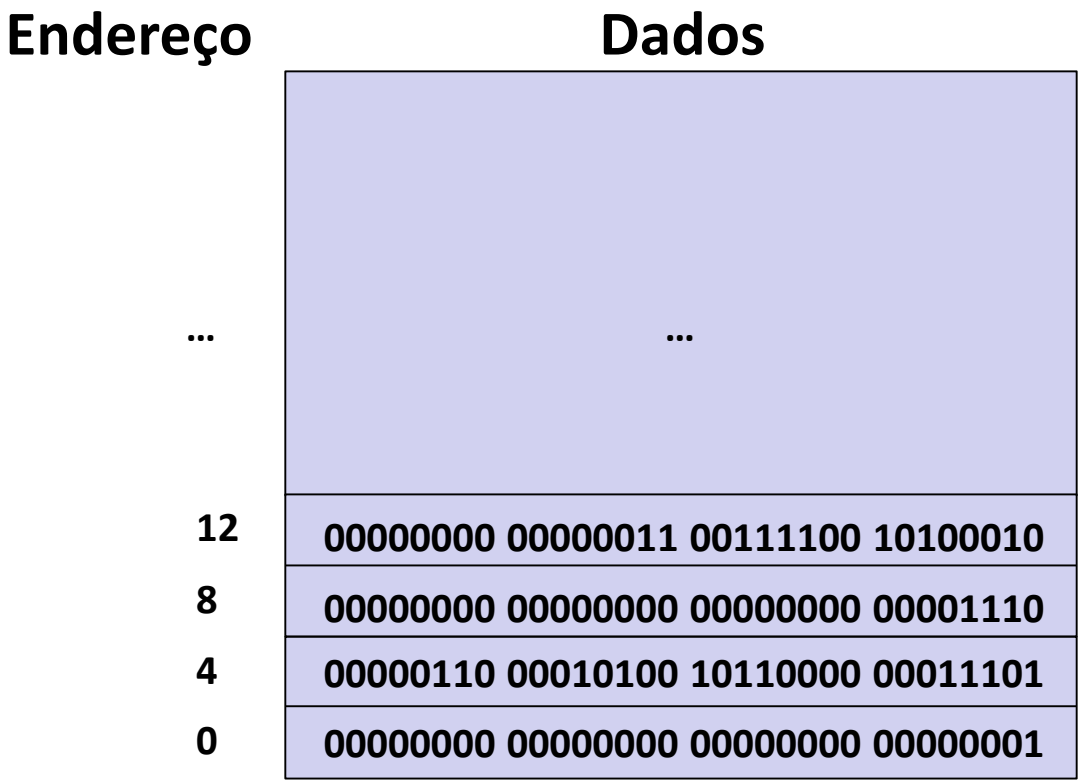
# Acesso à Memória

## Alinhamento da Memória



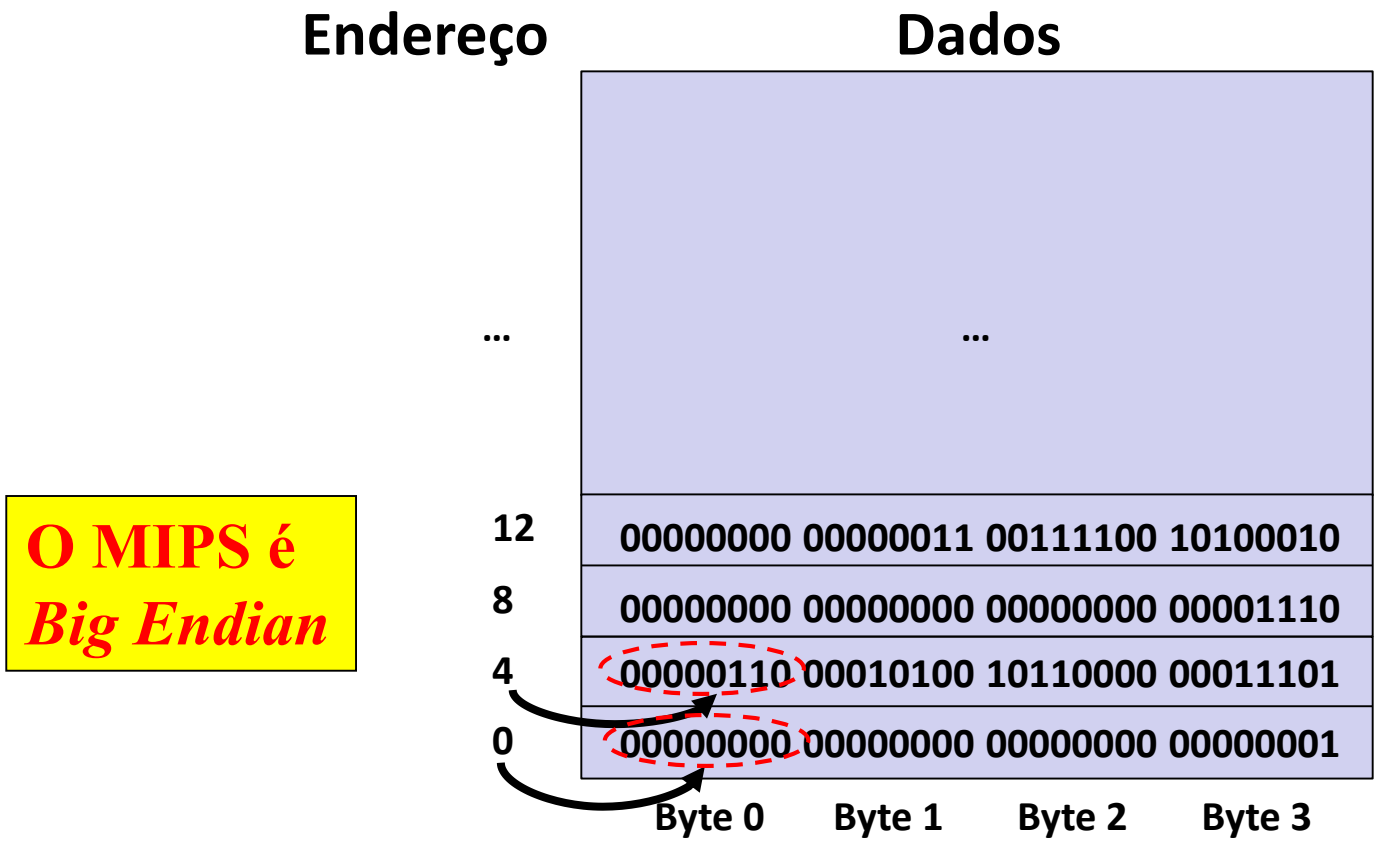
# Acesso à Memória

## Alinhamento da Memória



# Acesso à Memória

## Alinhamento da Memória




**O MIPS é *Big Endian***

# Acesso à Memória

## Endereçamento à Memória

- **Endereço = base + *offset***
  - **Base**: registrador (32 bits)
  - ***Offset***: imediato (16 bits)
- Para endereçar uma *word*, **base + *offset*** deve ser múltiplo de 4

- Por que o endereço de base fica no registrador e não no imediato (*offset*)?

# Acesso à Memória

## Endereçamento à Memória

- **Endereço = base + *offset***
  - **Base**: registrador (32 bits)
  - ***Offset***: imediato (16 bits)
- Para endereçar uma *word*, **base + *offset*** deve ser múltiplo de 4

| | |
|---|---|
| **lw $t3, 100($t2)** | **$t3 ← Memória[$t2 + 100]** |
| **lw $t1, 0($t0)** | **$t1 ← Memória[$t0]** |
| **sw $t3, 100($t2)** | **Memória[$t2 + 100] ← $t3** |
| **sw $t1, 0($t0)** | **Memória[$t0] ← $t1** |

# Acesso à Memória

## Instruções de acesso a *words*

- ***Load word* (lw)**
  - **Carrega uma *word* (32 bits) da memória para um registrador**
    **lw rd, *off*(b)**
- ***Store word* (sw)**
  - **Carrega uma *word* (32 bits) de um registrador para a memória**
    **sw rs, *off*(b)**

**Nota:** Endereços múltiplos de 4 são finalizados com **0x0**, **0x4**, **0x8**, **0xC**

***rd*: registrador destino**
***off*: offset**
***b*: base**
***rs*: registrador fonte**

# Acesso à Memória

## Instruções de acesso a *words*

- ***Load word* (lw)**
  - **Exemplo:**

    **Considere que o registrador $t1 contém 0x00400000**

    **O que faz a instrução a seguir?**

    **lw $t3, 0x60($t1)**

# Acesso à Memória

## Instruções de acesso a *words*

- ***Load word* (lw)**
  - **Exemplo:**

    **Considere que o registrador $t1 contém 0x00400000**

    **O que faz a instrução a seguir?**

    **lw $t3, 0x60($t1)**

  - **O código acima copia a *word* na posição 0x00400060 da memória para o registrador $t3.**

# Acesso à Memória

## Instruções de acesso a *words*

- *Store word* (sw)
  - **Exemplo:**

    **Considere que o registrador $t0 contém 0x00400014**

    **O que faz a instrução a seguir?**

    **sw $t3, -8($t0)**

# Acesso à Memória

## Instruções de acesso a *words*

- *Store word* (sw)
  - **Exemplo:**

    **Considere que o registrador $t0 contém 0x00400014**

    **O que faz a instrução a seguir?**

    **sw $t3, -8($t0)**

  - **O código acima copia a *word* no registrador $t3 para a posição 0x0040000C da memória.**

# Acesso à Memória

## Instruções de acesso a *halfword*

- ***Load halfword* (lh)**
  - **Carrega uma *halfword* da memória para os 16 bits menos significativos de um registrador**

    **lh rd, *off*(b)**
- ***Store halfword* (sh)**
  - **Carrega uma *halfword* (16 bits) dos 16 bits menos significativos de um registrador para a memória**

    **sh rs, *off*(b)**

**Nota:** Endereços múltiplos de 2 são finalizados com **0x0**, **0x2**, **0x4**, **0x6**, **0x8**, **0xA**, **0xC**, **0xE**

Quando **lh** é usada, os **16 bits mais significativos** do registrador são completados com **extensão do sinal**

# Acesso à Memória

## Instruções de acesso a *halfword*

- ***Load halfword unsigned*** **(lhu)**
  - **Carrega uma *halfword* da memória para os 16 bits menos significativos de um registrador**

    **lhu rd, *off*(b)**

**Nota:** Endereços múltiplos de 2 são finalizados com **0x0**, **0x2**, **0x4**, **0x6**, **0x8**, **0xA**, **0xC**, **0xE**

Quando **lhu** é usada, os **16 bits mais significativos** do registrador são completados com **zeros**

# Acesso à Memória

## Instruções de acesso a *byte*

- ***Load byte*** **(lb)**
  - **Carrega um *byte* da memória para os 8 bits menos significativos de um registrador**

    **lb rd, *off*(b)**
- ***Store byte*** **(sb)**
  - **Carrega um *byte* (8 bits) dos 8 bits menos significativos de um registrador para a memória**

    **sb rs, *off*(b)**

**Nota:** Todos endereços são válidos para bytes; não é necessário fazer alinhamento de endereços.

Quando **lb** é usada, os **24 bits mais significativos** do registrador são completados com **extensão do sinal**

# Acesso à Memória

## Instruções de acesso a *byte*

- ***Load byte unsigned* (lbu)**
  - **Carrega um *byte* da memória para os 8 bits menos significativos de um registrador**

    **lbu rd, *off*(b)**

**Nota:** Todos endereços são válidos para bytes; não é necessário fazer alinhamento de endereços.

Quando **lbu** é usada, os **24 bits mais significativos** do registrador são completados com **zeros**

# Acesso à Memória

## Instrução de *load* imediato

- ***Load upper immediate*** **(lui)**
  - **Carrega um valor imediato nos 16 bits mais significativos de um registrador**

    **lui rd,** ***immediate***

  - ***Exemplo:***

    **lui $t0, 0xABCD** **$t0 ← 0xABCD0000**

É bastante útil para inicializar o valor do registrador base.

# Acesso à Memória

## Definir seção de dados no MARS

- Utilizamos a diretiva ***.data***
- O início da área de dados é a posição **0x10010000**
- **Atenção**: ***.data*** **NÃO É uma instrução do MIPS**; é apenas uma diretiva do montador!

```
.data                    # inicia a seção de dados
    .word  1             # escreve 1 em 0x10010000
    .word  -3            # escreve -3 em 0x10010004
    .word  15            # escreve 15 em 0x10010008
```

# Acesso à Memória

## Definir seção de dados no MARS

- A partir de agora, utilizaremos:
  - ***.data*** antes da seção de dados
  - ***.text*** antes da seção de instruções

```
.data                       # inicia a seção de dados
        .word  1            # escreve 1 em 0x10010000
        .word  -3           # escreve -3 em 0x10010004
        .word  15           # escreve 15 em 0x10010008

.text                       # inicia a seção de instruções
        lui $t0, 0x1001     # carrega o reg $t0 com 0x10010000
        lw $t1, 8($t0)      # carrega $t1 com a word em 0x10010008
        add $s2, $t1, $t1   # soma $t1 com $t1 e coloca o resultado em $s2
        sw $s2, 12($t0)     # escreve a word $s2 em 0x1001000C
```

# Acesso à Memória

## Definir seção de dados no MARS

- Outras diretivas para ***.data***

  ***.byte*** reserva um *byte*

  ***.half*** reserva uma *halfword*

  ***.space x*** reserva *x bytes*