

## Aula 6

• Instruções de transferência de informação

• Organização de informação em memória:

• *little endian versus big endian*

• Resumo dos modos de endereçamento do MIPS

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira e Silva

Numa aula anterior analisámos o seguinte exemplo:

```
add    $8, $17, $18    # Soma $17 com $18 e armazena o resultado em $8
add    $9, $19, $20    # Soma $19 com $20 e armazena o resultado em $9
sub     $16, $8, $9     # Subtrai $9 a $8 e armazena o resultado em $16
```

sendo o equivalente em C

```
// a, b, c, d e z residem, respectivamente, em:
// $17, $18, $19, $20 e $16
// $8 e $9 representam variáveis temporárias utilizadas em C
```

```
int a, b, c, d, z;
z = (a + b) + (c + d);
```

Note-se que este trecho de código faz uso apenas de registos internos do CPU

## Instruções de transferência de informação (controladas)

E se pretendêssemos agora somar os elementos de um *array* composto por *n* elementos?

Se for maior do que o número de registos disponíveis na CPU seria necessário recorrer a recursos externos na memória.

Por outro lado, também vimos que a arquitectura MIPS do tipo *load-store*, pelo que não permite operar directamente sobre o conteúdo da memória externa.

Deveria existir, portanto, instruções para transferir dados entre os registos do CPU e a memória externa.

Como será então possível codificar as instruções de transferência de informação de memória externa (para escrita e leitura), sabendo que as instruções do MIPS ocupam, todas, exactamente 32 bits?

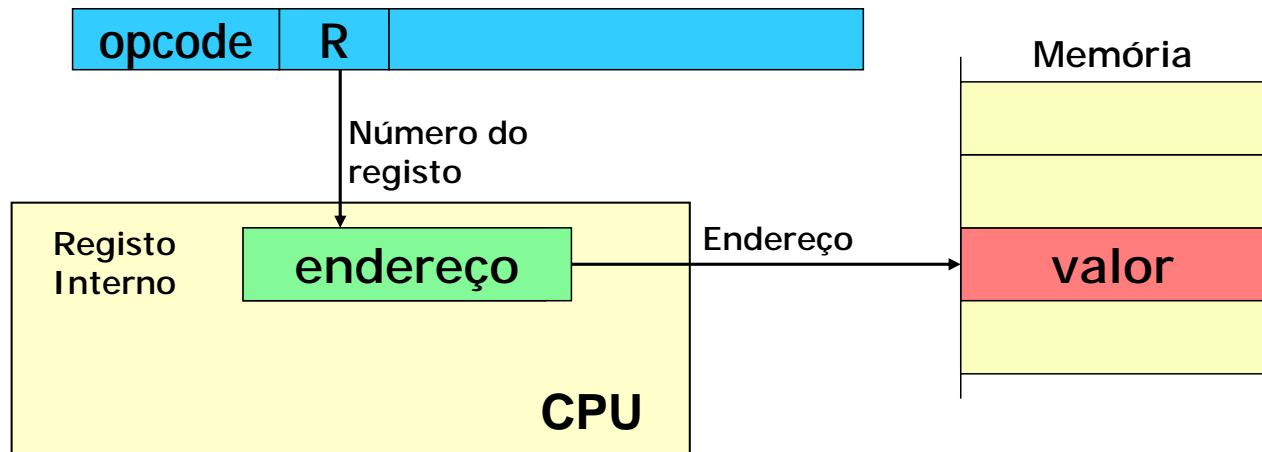
**Note-se que um endereço de memória no MIPS representa-se por 32 bits, pelo que ele sozinho ocuparia a totalidade do código máquina da instrução**

Solução: Em vez do endereço, a instrução indica um registo contendo o endereço de memória a aceder (como sabemos a dimensão do registo interno é 32 bits). Chama-se a este modo de endereçamento:

***endereçoamento indirecto por registo***

## Endereçamento indirecto por registo

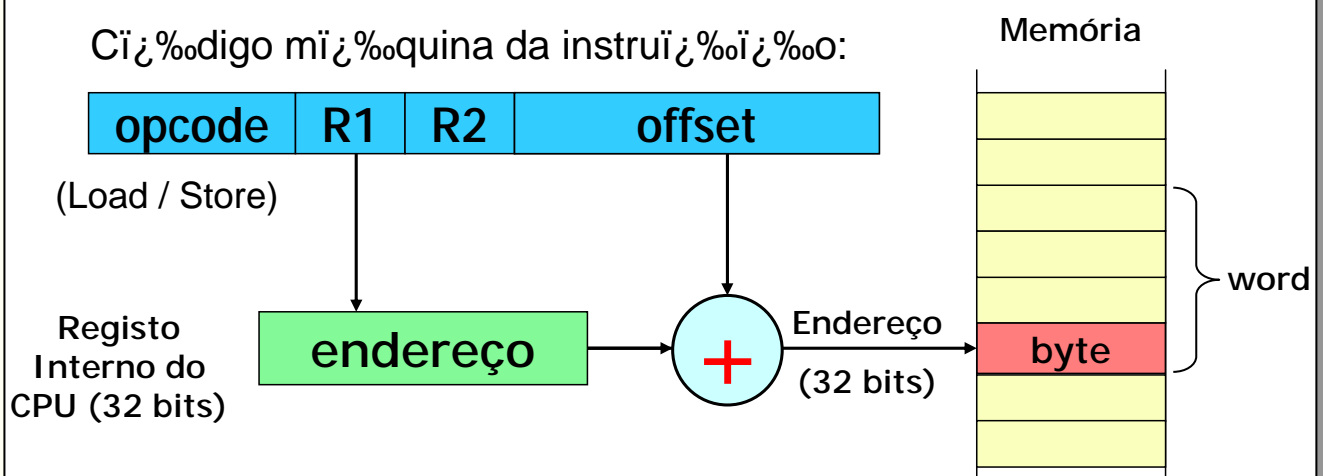
Código máquina da instrução:



## A solução do MIPS

### Endereçamento indirecto por registo com deslocamento

Código máquina da instrução:



**offset:** Deslocamento (positivo ou negativo)

**R1:** Registo de endereçamento

**R2:** Registo de dados: destino / origem

### Instrução de leitura de 1 word da memória:

**LW** - (*load word*) transfere uma palavra de 32 bits da memória para um registo interno do CPU (1 word é armazenada em 4 posições de memória consecutivas)

Formato:



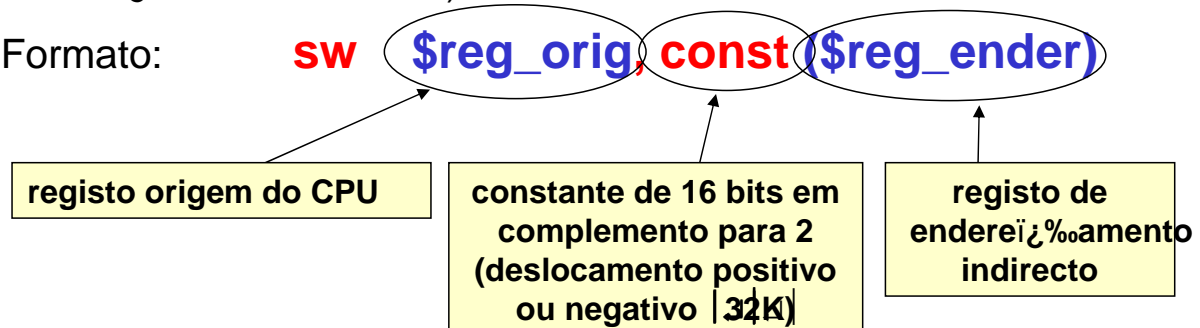
Exemplo:

**lw \$5, 4 (\$2)** # transfere para o registo \$5 a *word* armazenada  
 # no endereço de memória calculado como:  
 # **(conteúdo do registo \$2) + 4**

### Instrução de escrita de 1 word na memória:

**SW** - (*store word*) transfere uma palavra de 32 bits de um registo interno do CPU para a memória (1 word é armazenada em 4 posições de memória consecutivas)

Formato:



Exemplo:

**sw \$7, 8 (\$4)** # transfere a *word* armazenada no registo \$7  
 # para o endereço de memória calculado como:  
 # **(conteúdo do registo \$4) + 8**

Consideremos agora o seguinte exemplo:

$g = h + A[5]$

assumindo que  $g$ ,  $h$  e o endereço de início do array  $A$  residem nos registos  $\$17$ ,  $\$18$  e  $\$19$ , respectivamente

usando instrução Assembly do MIPS, a expressão anterior tomaria a seguinte forma (supondo que  $A$  é um array de words, i.e. 32 bits):

```
lw      $8, 20($19)      # Lê A[5] da memória
add     $17, $18, $8      # Calcula novo valor de g
```

Variável temporária (destino)

Não esquecer que a memória está organizada em bytes (*byte-addressable*)

Retomemos a primeira instrução:

```
lw      $8, 20($19)      # Lê A[5] da memória
```

O endereço da memória é calculado somando o conteúdo do registo indicado entre parêntesis com a constante indicada na instrução. Se o conteúdo do registo  $\$19$  for  $0x10010000$  o endereço da memória será:

```
lw      $8, 20($19)      # Lê A[5] da memória
```

$0x14 + 0x10010000 = 0x10010014$  Endereço resultante

Como cada elemento do array ocupa quatro bytes (array de words), o elemento acedido será  $A[5]$

Se pretendi mos agora obter:

$$A[5] = h + A[5]$$

assumindo mais uma vez que **h** e o endere o inicial do array residem nos registos **\$18** e **\$19**, respectivamente

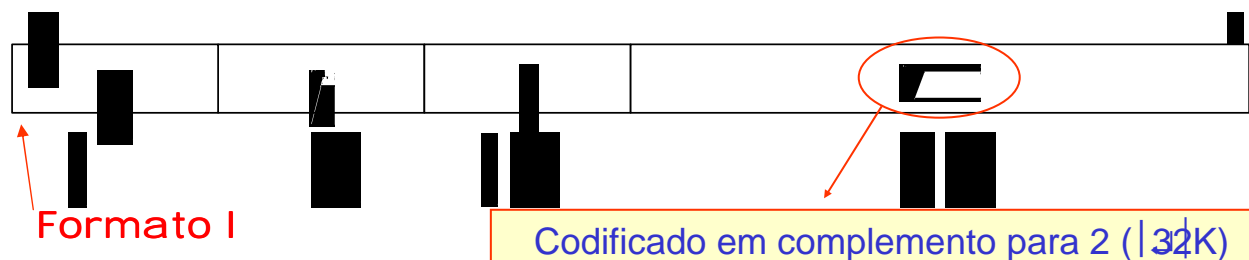
Poderi mos fazi lo com o seguinte c digo:

<b>lw</b>	\$8, 20(\$19)	#L� A[5] da mem�ria
<b>add</b>	\$8, \$18, \$8	#Calcula novo valor
<b>sw</b>	\$8, 20(\$19)	#Escreve resultado em A[5]

Arquitectura load/store: as opera  es aritm ticas e l gicas s  podem ser efectuadas sobre registos internos do CPU

### Codifica  o das instru  es de transfer ncia de registo para mem ria:

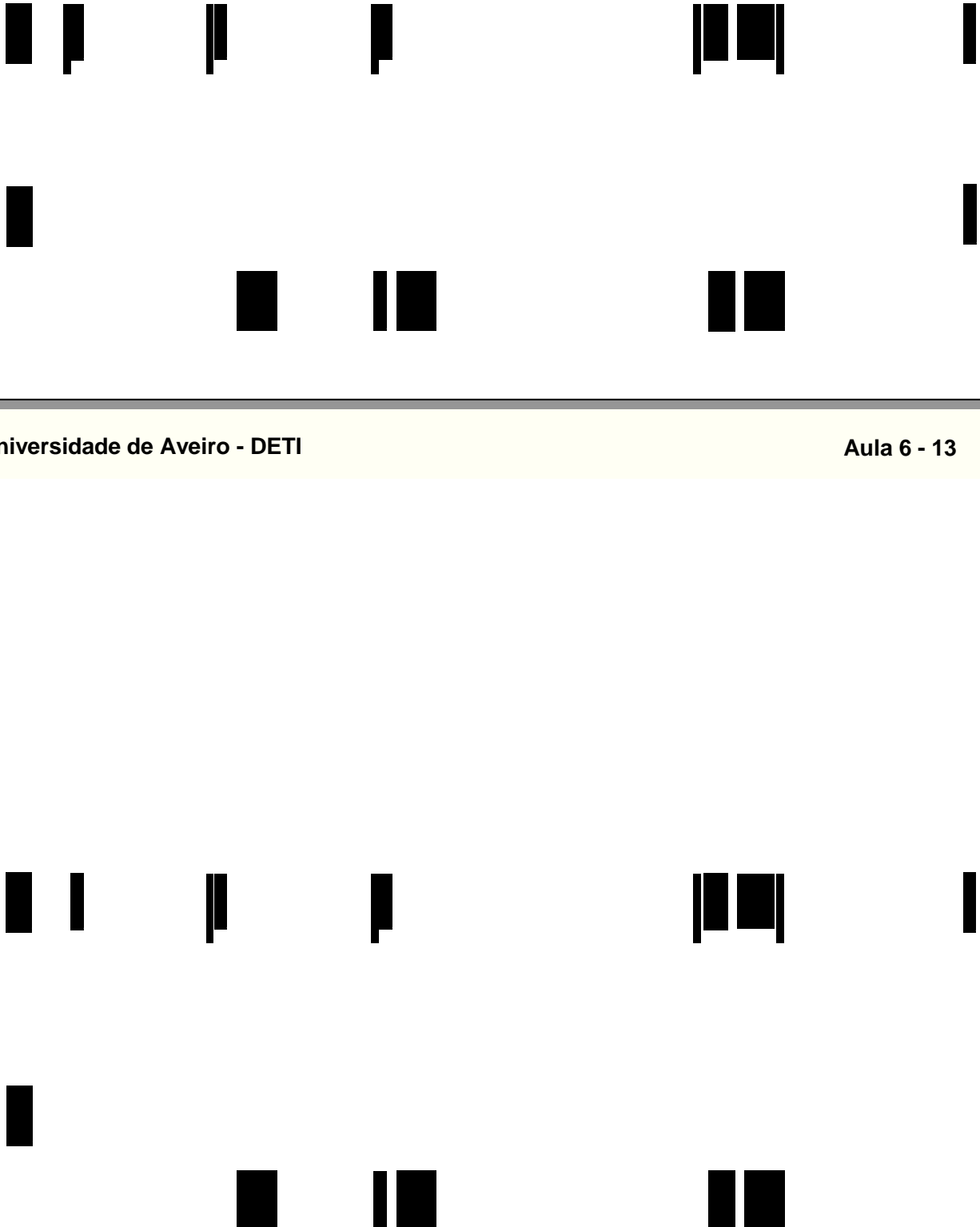
A necessidade de codifica  o de uma constante de 16 bits implica que estas instru  es sejam codificadas no **formato I**



**Codificação da instrução LW:**

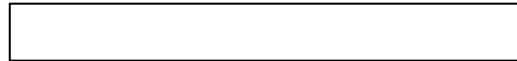
`lw $8, 20($19) #Lê o A[5] da memória`

Corresponderia à seguinte instrução máquina:



Exemplo: o seguinte trecho de código assembly:

```
lw    $8, 20($19)      # Lê A[5] da memória
add   $8, $18, $8       # Calcula novo valor
sw    $8, 20($19)      # Escreve resultado em A[5]
```



## Restrições de alinhamento nos endereços das variáveis

**Resposta 1:** Se, numa instrução de leitura/escrita de **word**, for especificado um endereço não múltiplo de 4, o MIPS a tenta executar verifica que o endereço não é válido e gera uma excepção, terminando aí a execução do programa

Como se evita o problema ?

Garantindo que as variáveis de tipo armazenadas num endereço múltiplo de 4

Directiva **align n** do *Assembler* (fora o alinhamento do endereço de uma variável num valor múltiplo de 2

## Restrições de alinhamento nos endereços das variáveis

**Questão 2:** Como é possível a leitura/escrita de 1 byte de informação uma vez que o ISA do MIPS define que a memória é organizada em bytes (byte-addressable) ?

Na leitura/escrita de **1 byte** de informação o problema do alinhamento, do ponto de vista do programador, não se coloca

Como é que o MIPS resolve o acesso?

## Restrições de alinhamento nos endereços das instruções

**Resposta:** o MIPS gera o endereço múltiplo de 4 (EM4) que, no acesso a uma word, inclui o endereço pretendido

No caso de Leitura

Executa uma instrução de **leitura** do endereço EM4, e, dos 32 bits lidos, retira os 8 bits correspondentes ao endereço pretendido

No caso de Escrita: **(Read Modify Write)**

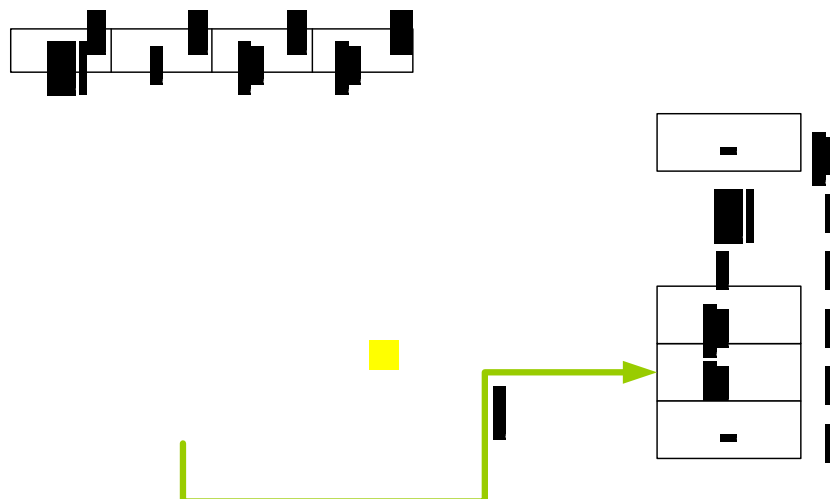
Executa uma instrução de **leitura** do endereço EM4

De entre os 32 bits lidos substitui 8 bits no endereço pretendido

Escreve **word** modificada em EM4

## Restrições de alinhamento nos endereços das instruções

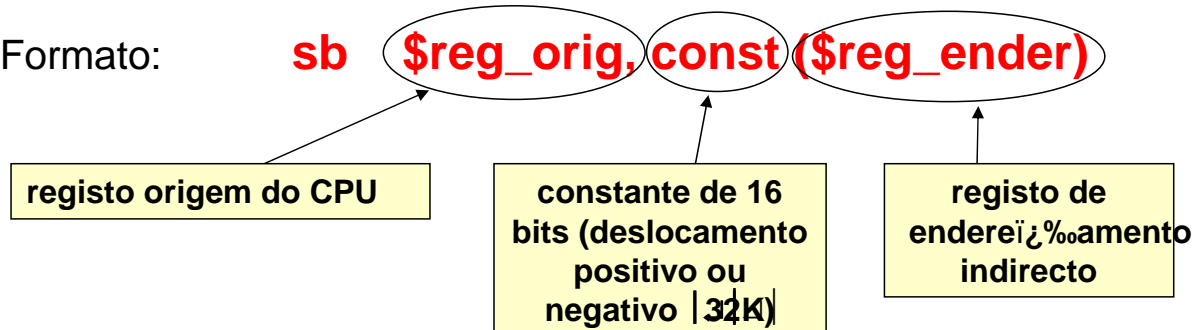
Exemplo no caso de Leitura (lbu)



### Instrução de escrita de 1 byte na memória:

**SB** - (store byte) transfere um byte de um registo interno para a memória - os 8 bits menos significativos

Formato:



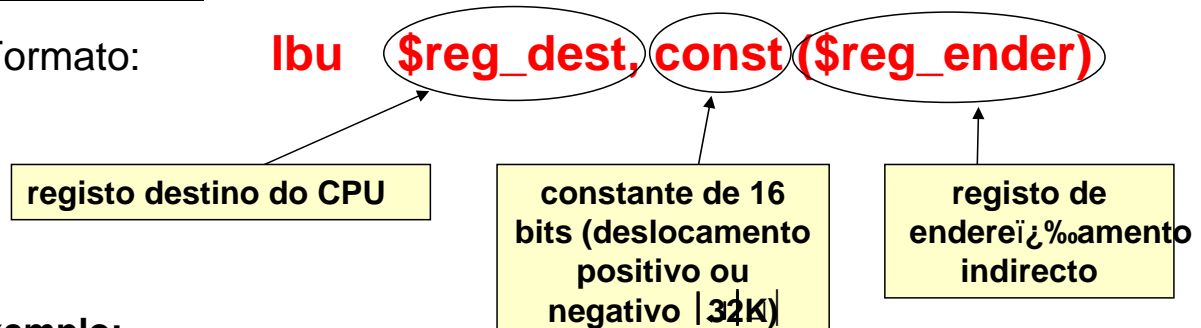
Exemplo:

**sb \$7, 8 (\$4)** # transfere o *byte* armazenado no registo \$7 (8 bits menos significativos) para o endereço de memória calculado como:  
# **(conteúdo do registo \$4) + 8**

### Instrução de leitura de 1 byte da memória (1):

**LBU** - (load byte unsigned) transfere um byte da memória para um registo interno - os 24 bits mais significativos do registo destino são colocados a 0

Formato:



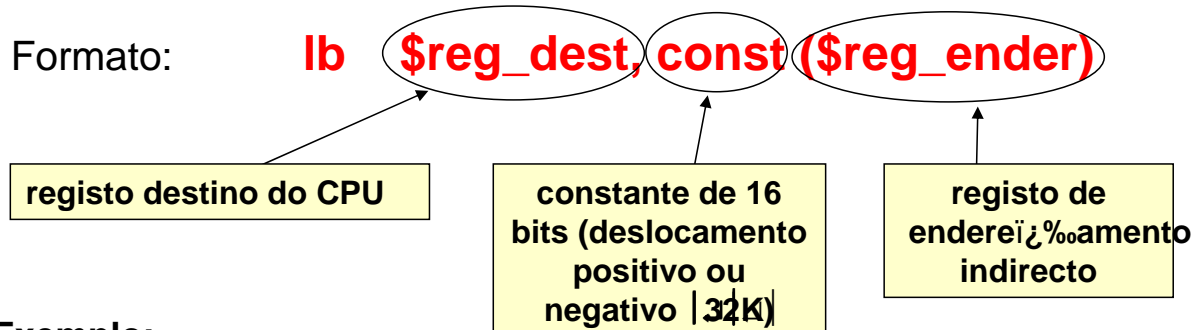
Exemplo:

**lbu \$5, 4 (\$2)** # transfere para o registo \$5 o *byte* armazenado no endereço de memória calculado como:  
# **(conteúdo do registo \$2) + 4**  
# os 24 bits mais significativos de \$5 são colocados a zero

## Instru  o de leitura de 1 byte da mem  ria (2):

**LB - (load byte)** transfere um byte da mem  ria para um registo inteiro, fazendo extens  o de sinal do valor lido de 8 para 32 bits

Formato:



Exemplo:

**lb \$5, 4 (\$2)**      # transfere para o registo \$5 o *byte* armazenado  
                          # no endere o de mem  ria calculado como:  
                          #      **(conte o do registo \$2) + 4**  
                          # o bit mais significativo do byte transferido      
                          # replicado nos 24 bits mais significativos de \$5

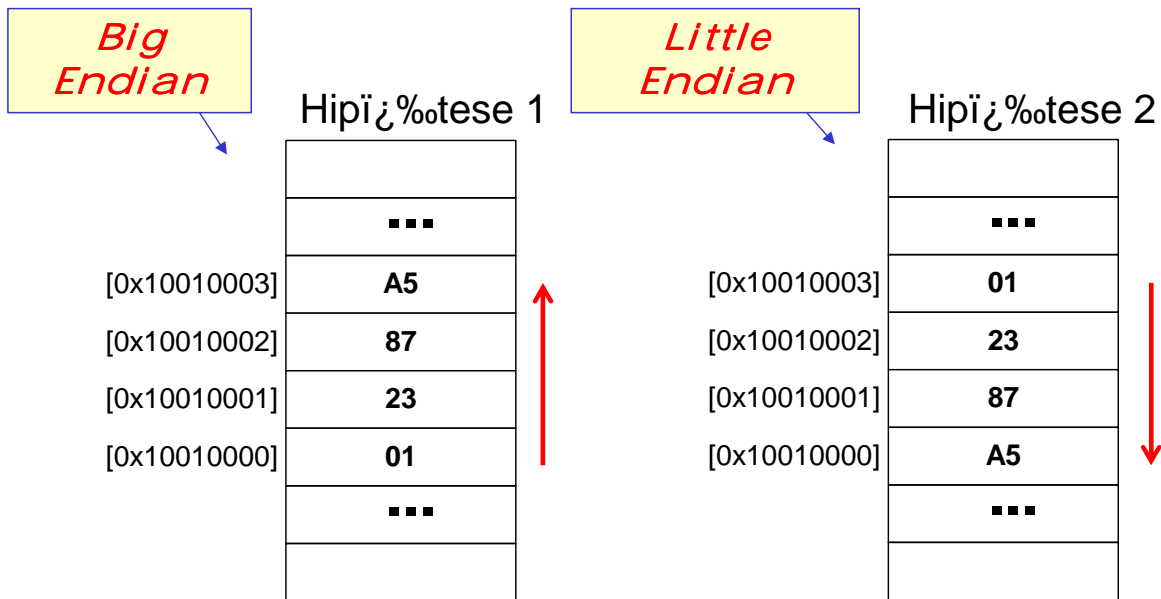
## Organiza  o da informa  o em mem  ria

Considere-se o valor:    0x012387A5

**32 bits, logo, 4 bytes**  
 numa mem  ria do tipo **byte addressable**, qual a ordem de armazenamento dos **bytes**?

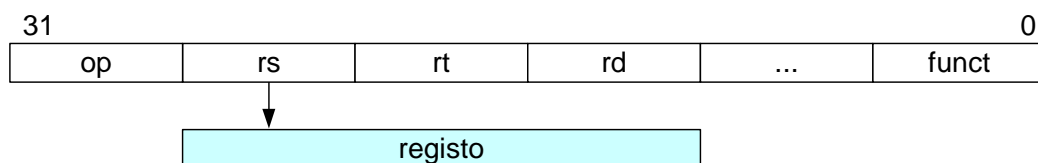
Como ser   ele armazenado na mem  ria?

**Exemplo** Valor a armazenar: 0x01 23 87 A5

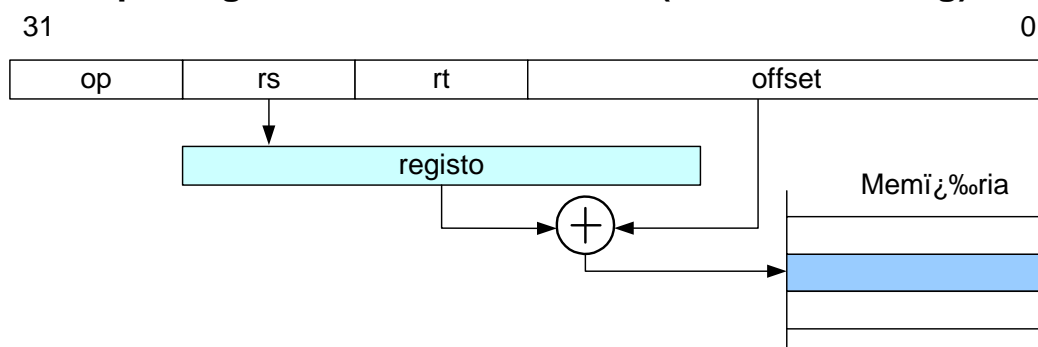


**Resumindo.** Os modos de endereçamento suportados pelo MIPS são:

### Register Addressing:



### Indirecto por registo com deslocamento (base addressing):



**Immediate Addressing:****PC-relative Addressing:****Direct Addressing:**