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, Tomás 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 não explicitadas 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 (cont.)

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

- Se ***n*** for maior do que o número de registos disponíveis no CPU seria necessário recorrer a recursos externos – a memória.
- Por outro lado, também já vimos que a arquitectura do MIPS é do tipo *load-store*, pelo que não permite operar directamente sobre o conteúdo da memória externa.
- Deverão existir, portanto, instruções para transferir informação entre os registos do CPU e a memória externa.

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

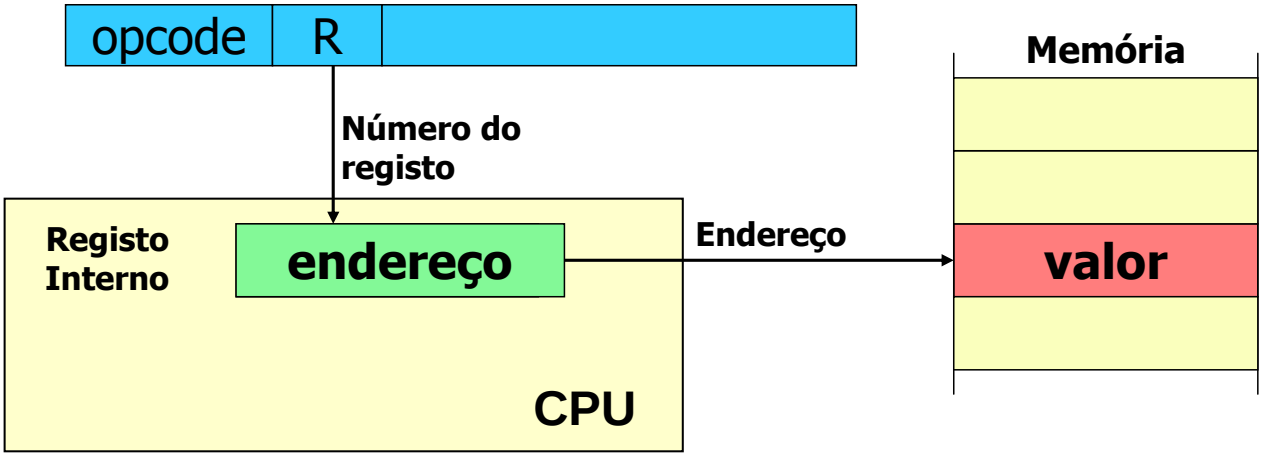
Note-se que um endereço de memória no MIPS é representado por 32 bits, pelo que ele sozinho ocuparia a totalidade do código máquina da instrução

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

endereçamento indirecto por registo

Endereçamento indirecto por registo

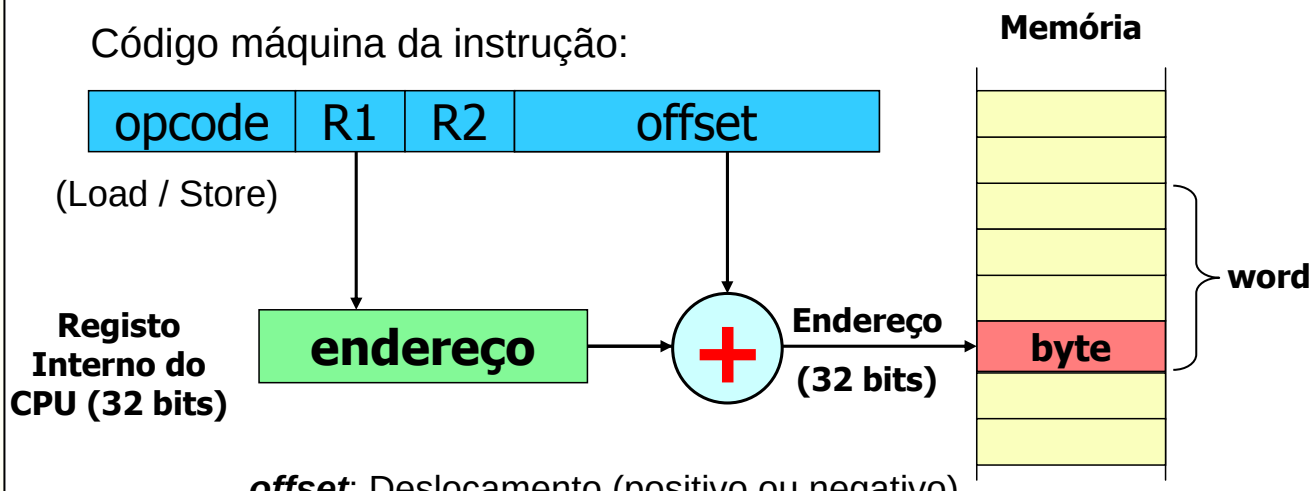
Código máquina da instrução:



A solução do MIPS:

endereçamento indirecto por registo com deslocamento

Código máquina da instrução:



offset: Deslocamento (positivo ou negativo)

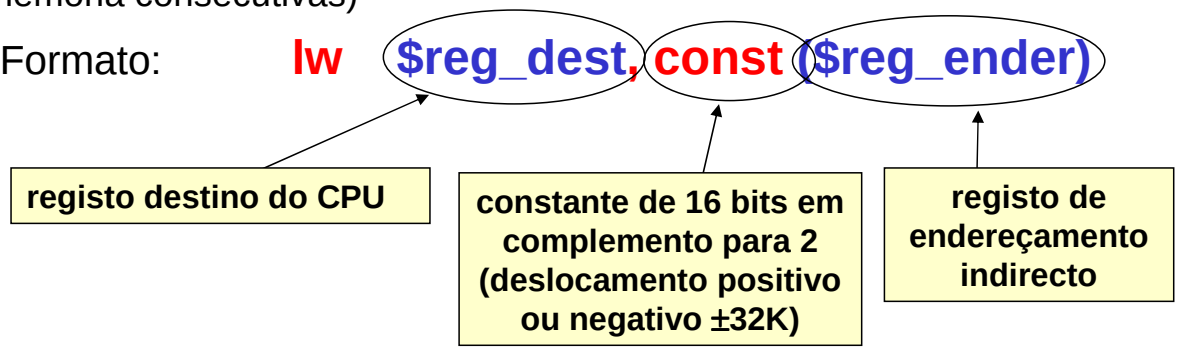
R1: Registo de endereçamento

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: **lw $reg_dest, const ($reg_ender)**



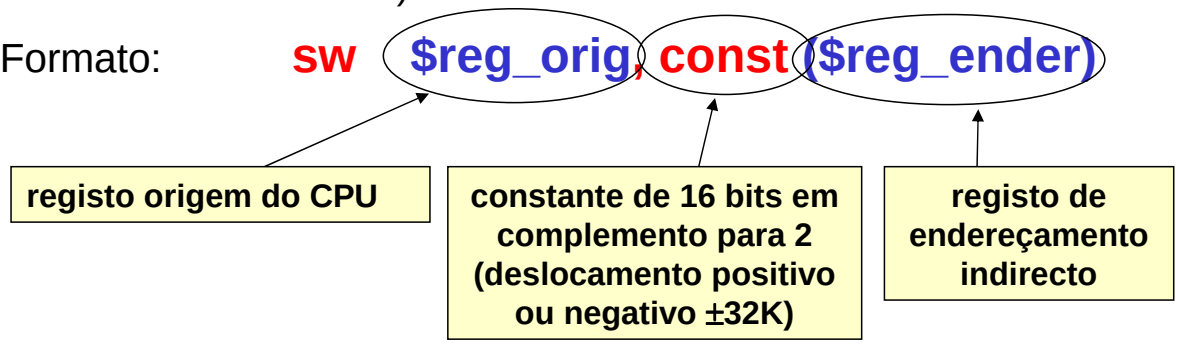
Exemplo:

```
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: **sw $reg_orig, const ($reg_ender)**



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 instruções do *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 explicitada na instrução. Se o conteúdo de **$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 pretendêssemos agora obter:

A[5] = h + A[5]

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

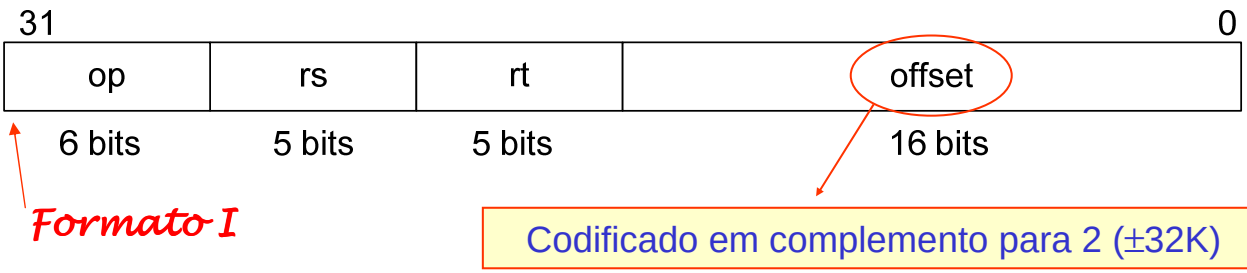
Poderíamos fazê-lo com o seguinte código:

```
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]
```

Arquitectura load/store: as operações aritméticas e lógicas só podem ser efectuadas sobre registos internos do CPU

Codificação das instruções de transferência memória/registo no MIPS:

A necessidade de codificação de uma constante de 16 bits implica que estas instruções sejam codificadas com o **formato I**



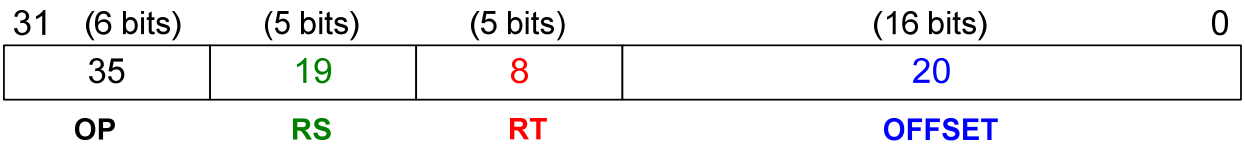
Formato I

Codificado em complemento para 2 (±32K)

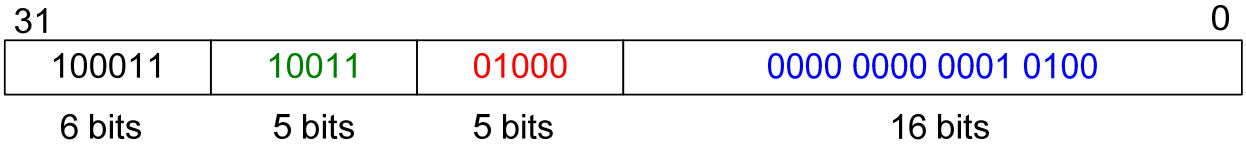
Codificação da instrução LW:

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

Corresponderia à seguinte instrução máquina:



LW RT, OFFSET(RS)

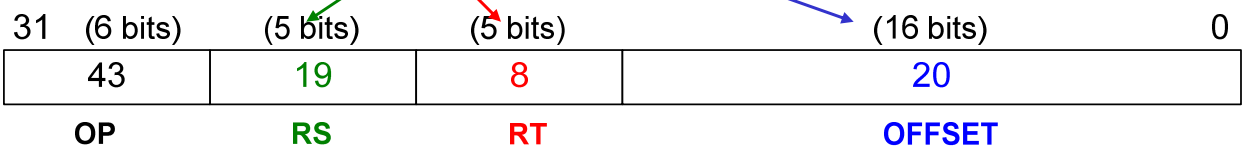


$100011\mathbf{10011}\mathbf{01000}0000000000010100_2$ = 0x8E680014

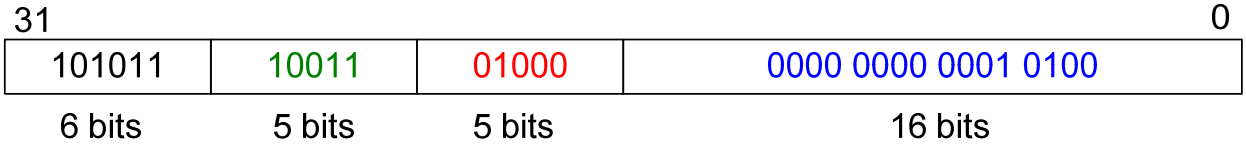
Codificação da instrução SW:

```
sw    $8, 20($19) #Escreve result. em A[5]
```

Corresponderia à seguinte instrução máquina:



SW RT, OFFSET(RS)



$101011\mathbf{10011}\mathbf{01000}0000000000010100_2$ = 0xAE680014

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]
```

Corresponde à seguinte codificação:

31 0

0x23	0x13	0x08	0x0014		
0x00	0x12	0x08	0x08	0x00	0x20
0x2B	0x13	0x08	0x0014		

Resultando no seguinte código máquina:

$10001100110100000000000000010100_2$ = 0x8E680014

$00000010010010000100000000100000_2$ = 0x02484020

$10101100110100000000000000010100_2$ = 0xAE680014

Restrições de alinhamento nos endereços das variáveis

- Externamente o barramento de endereços do MIPS só tem disponíveis 30 dos 32 bits: $A_{31}...A_2$. Ou seja, qualquer combinação nos bits A_1 e A_0 é ignorada no barramento de endereço exterior.
- Assim, do ponto de vista externo, só são gerados endereços **múltiplos de 2^2 = 4 (ex: ...0000, ...0100, , ...1000, ...1100)**

O acesso a words só é possível em endereços múltiplos de 4

- **Questão 1:** O que acontece quando o MIPS tenta executar uma instrução de leitura/escrita de uma ***word*** da memória, num endereço não múltiplo de 4 ?
- **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*) ?

Restrições de alinhamento nos endereços das variáveis (cont.)

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

- Como se evita o problema ?
  - Garantindo que as variáveis do tipo ***word*** estão armazenadas num endereço múltiplo de 4
  - Directiva `.align n` do *Assembler* (força o alinhamento do endereço de uma variável num valor múltiplo de 2^n)

Restrições de alinhamento nos endereços das variáveis (cont.)

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?

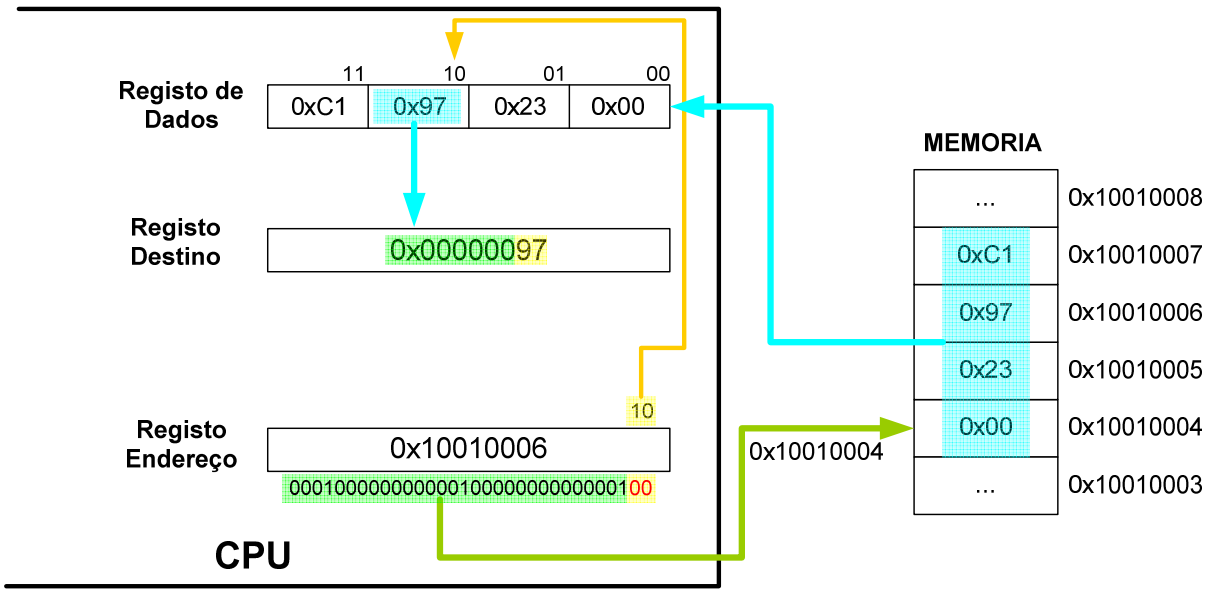
Restrições de alinhamento nos endereços das variáveis (cont.)

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 de **1 word** 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 de **1 word** do endereço EM4
  - De entre os 32 bits lidos substitui 8 bits no endereço pretendido
  - Escreve a **word** modificada em EM4

Restrições de alinhamento nos endereços das variáveis (cont.)

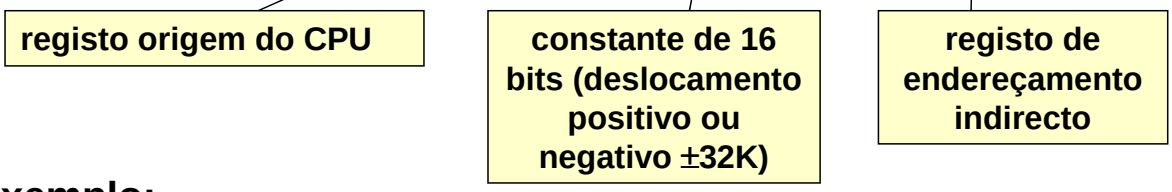
- Exemplo no caso de Leitura (lbu):



Instrução de escrita de 1 byte na memória:

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

Formato: **sb $reg_orig, const ($reg_ender)**



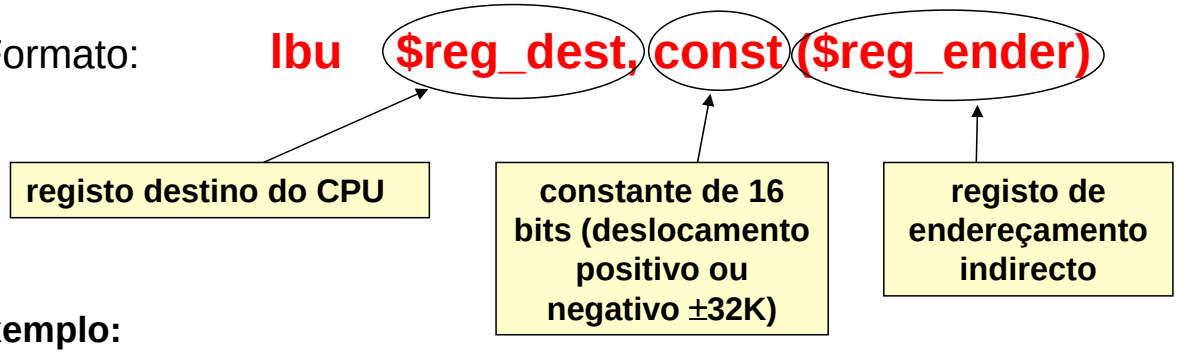
Exemplo:

```
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: **lbu $reg_dest, const ($reg_ender)**



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
```

Instrução de leitura de 1 byte da memória (2):

LB - (load byte) transfere um byte da memória para um registo interno, **fazendo extensão de sinal do valor lido de 8 para 32 bits**

Formato: **lb $reg_dest, const ($reg_ender)**

- **registo destino do CPU**
- **constante de 16 bits (deslocamento positivo ou negativo ±32K)**
- **registo de endereçamento indirecto**

Exemplo:

```
lb $5, 4 ($2)      # transfere para o registo $5 o byte armazenado
                   # no endereço de memória calculado como:
                   #       (conteúdo do registo $2) + 4
                   # o bit mais significativo do byte transferido é
                   # replicado nos 24 bits mais significativos de $5
```


Organização da informação em memória

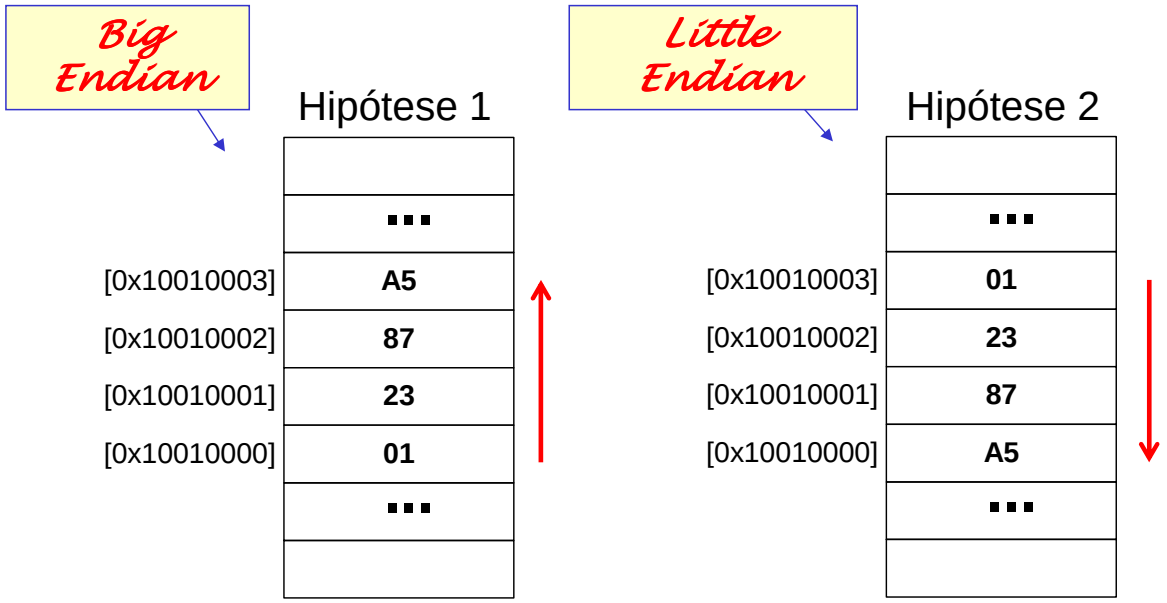
Considere-se o valor: 0x012387A5

32 bits, logo, 4 bytes

numa memória do tipo *byte addressable*, qual a ordem de armazenamento dos *bytes*?

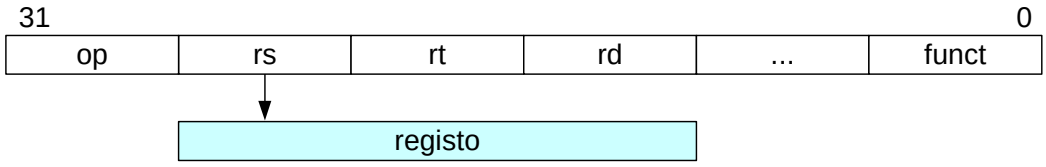
Como será ele armazenado na memória?

Exemplo – Valor a armazenar: 0x01 23 87 A5

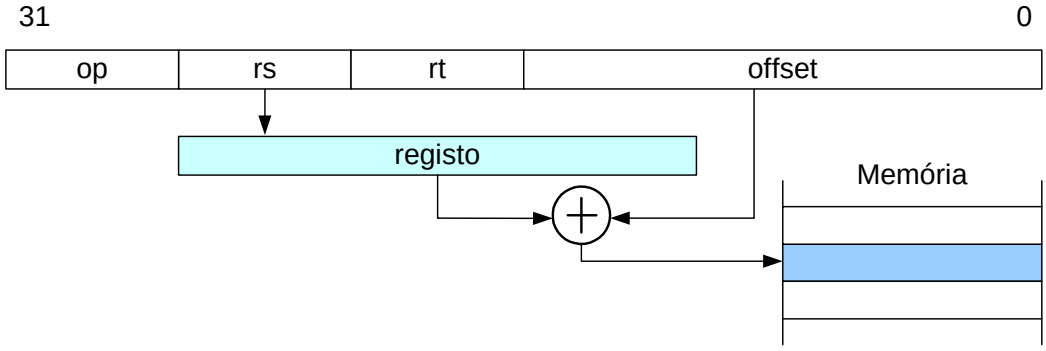


Resumindo. Os modos de endereçamento suportados pelo MIPS são:

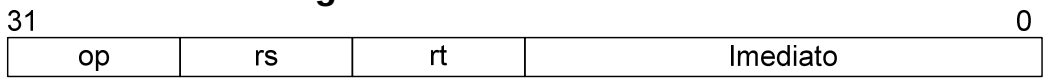
- **Register Addressing:**



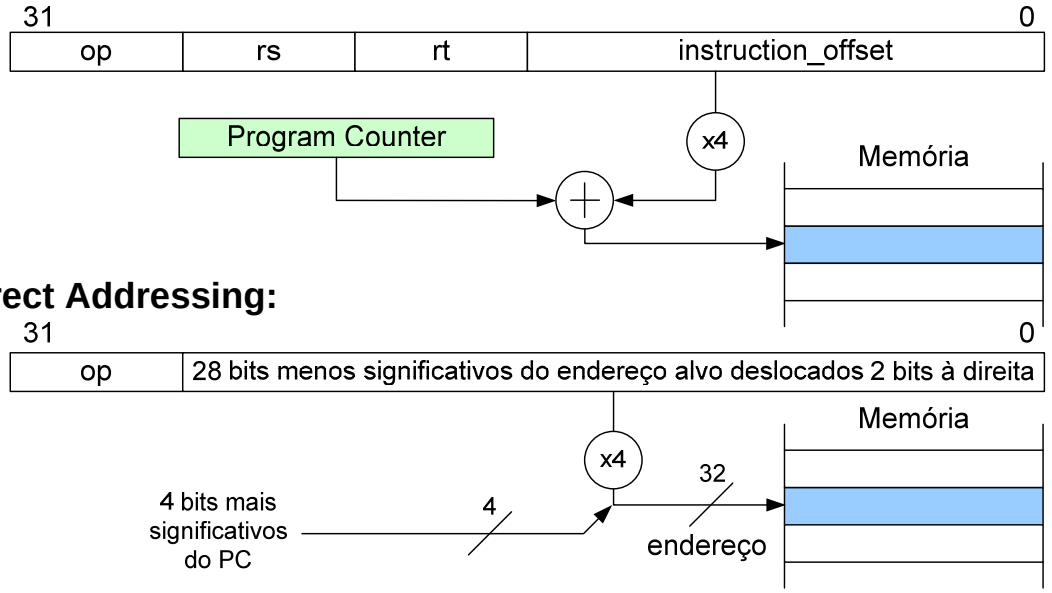
- **Indirecto por registo com deslocamento (base addressing):**



- **Immediate Addressing:**



- **PC-relative Addressing:**



- **Direct Addressing:**