

## Aula 5

• Modos de endereçamento em saltos condicionais e incondicionais

• Codificação das instruções de salto **condicional** e no MIPS

• Formatos de instruções

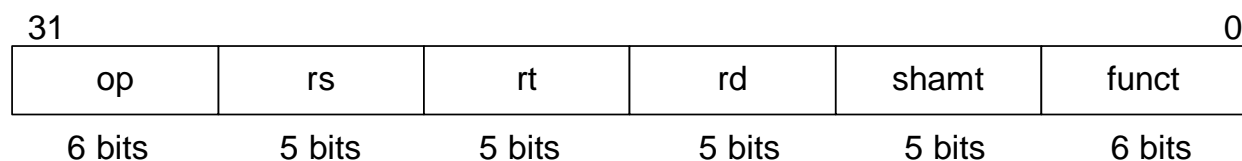
• Modos de endereçamento do MIPS:

• Modo imediato e uso de constantes

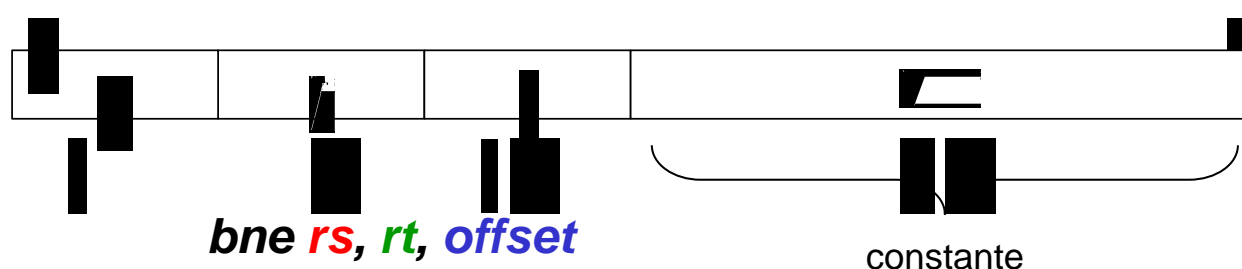
Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira e Silva

## Codificação das instruções de salto condicional no MIPS

• As instruções aritméticas e lógicas no MIPS seguem o **formato R**



• A necessidade de codificação de **endereços** da instrução de salto obriga à definição de um novo formato, o **formato I**





## Solução: Utiliza o registo PC (Program Counter)

Usando o **endereço de salto relativo**, o valor do endereço-alvo é calculado somando algebricamente o *offset* de 16 bits codificado na instrução ao valor corrente do PC

No MIPS o valor do PC corresponde ao endereço actual da próxima instrução a ser executada, uma vez que esse registo é incrementado na fase *execute* da instrução

Assim, o endereço-alvo (novo PC) passaria a ser:

$$\text{Novo PC} = \text{PC actual} + \text{offset}$$

Note-se que o *offset* de 16 bits é interpretado como um **valor em complemento para dois**, permitindo o salto para **endereços anteriores (offset negativo) ou posteriores (offset positivo)** ao PC

Tomemos o seguinte exemplo:

```
[0x00400000]    bne    $19, $20, ELSE
[0x00400004]    add    $16, $17, $18
[0x00400008]    j      END_IF
[0x0040000C]    ELSE: sub    $16, $16, $19
[0x00400010]    END_IF:
```

O endereço correspondente ao label **ELSE** é **0x0040000C**

O "branch\_offset" seria portanto:

$$\text{ELSE} - (\text{PC}) =$$

$$0x0040000C - 0x00400004 = 0x08$$

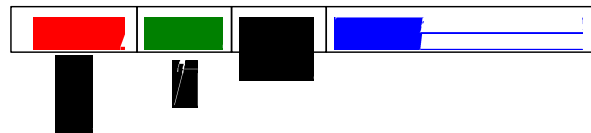
No entanto, como **cada instrução ocupa sempre 4 bytes em memória** (a partir de um endereço múltiplo de 4), o "branch\_offset" é, na realidade, **calculado em instruções**. Logo:

$$\text{"branch_offset"} = 0x08 / 4 = 0x02$$

Uma instrução de salto condicional pode, assim, **referir qualquer endereço de uma outra instrução que se situe até 32K instruções depois dela própria**.

**Exemplo!** Como interpreta o CPU uma instrução?

0x00400028	...
0x0040002C	
0x00400030	0X1509FFFD
0x00400034	...



**Exemplo:** A instrução `Label #se Label=0x001D14C8`

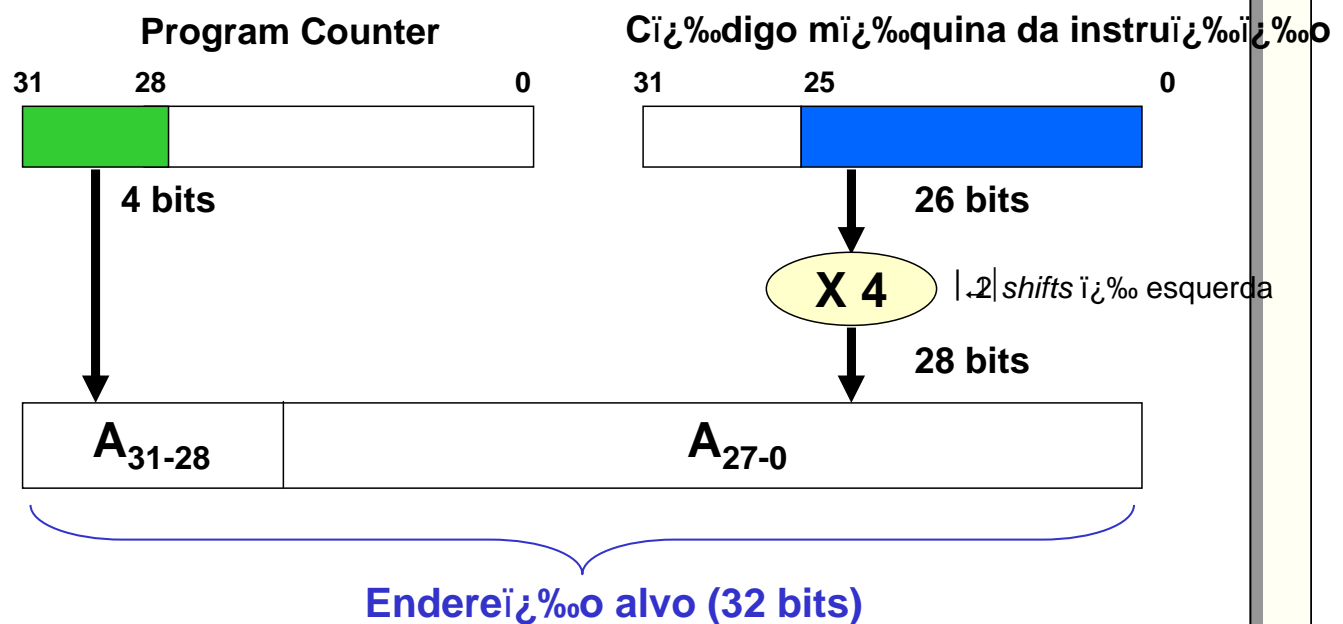
será codificada como:

$$0x001D14C8 / 4 = 0x00074532$$



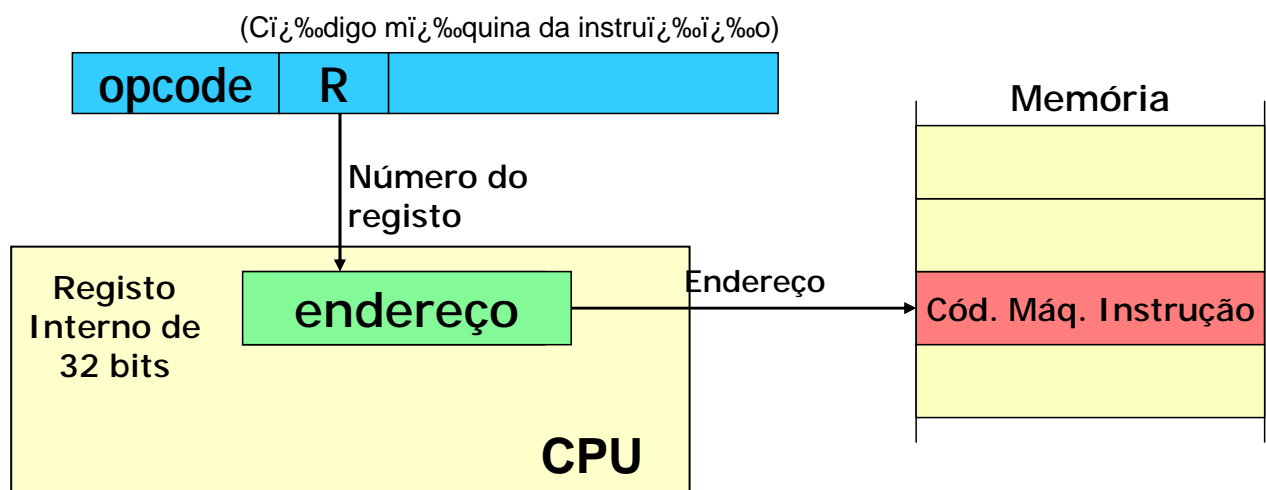
Código binário `00001000000001110100010100110010`<sub>2</sub> = 0x08074532

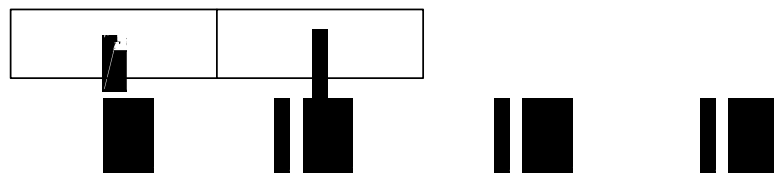
Mas se a instrução só codifica 28 bits (26 bits para o endereço),  
**como é formado o endereço final de 32 bits**



**Qual a maneira de especificar um endereço para salto,  
 usando para isso os 32 bits?**

Utiliza-se o **endereço indireto por registo**, ou seja, um registo interno (de 32 bits) armazena o endereço da instrução de salto





Para além dos modos de endereçamento que já conhecemos, MIPS suporta ainda um outro tipo de endereçamento designado por **endereçamento imediato**.

Relembremos os quatro princípios básicos no design da arquitectura

1. A simplicidade favorece a regularidade
2. Quanto mais pequeno mais rápido
3. Um bom design implica compromissos adequados
4. **O que é mais comum deve ser mais rápido**

O ponto 4. determina que a capacidade de tornar mais rápida a execução das operações que ocorrem mais vezes, em seu aumento global do desempenho!

Pode-se verificar, estatisticamente, que um número significativo de instruções em que está envolvida uma operação aritmética usam uma **constante** como um dos seus operandos

É vulgar que este número seja superior a 50% das instruções que envolvem a ALU num determinado programa.

A constante zero, por exemplo, é utilizada no MIPS para o registo permanentemente com esse valor (\$0).

A constante um, por outro lado, também é utilizada em operações de incremento ou decremento de variáveis de contagem usadas dentro de estruturas em ciclo fechado.

Chamamos **constante** a um valor determinado com antecedência (na altura em que o programa é escrito) que não se pretende que seja ou possa ser mudado durante a execução do programa

Se a constante fosse armazenada na memória externa, a sua utilização implicaria sempre o recurso a duas operações: uma para ler o valor da constante para um registo interno, e outra para operar com base nessa constante.

Ex.: #Constante na mem. externa (endereço em \$6)

```
lw    $5, 0($6)    #Ler constante p/ o registo $5
add   $8, $7, $5    #Somar $7 com a constante
```

Para aumentar a eficiência, as arquitecturas modernas, habitualmente, um conjunto de instruções e constantes se encontram armazenadas na própria instrução.

Desta forma o acesso a constantes é imediato, sem necessidade de recorrer a uma operação prévia de leitura da memória.

No caso do MIPS as instruções são identificadas pelo sufixo imediato nas seguintes:

```
addi  $3, $5, 4      # $3 = $5 + 0x0004
andi  $17, $18, 0x3AF5 # $17 = $18 & 0x3AF5
ori   $12, $10, 0x0FA2 # $12 = $10 | 0x0FA2
slti  $2, $12, 16     # $2 = 1 se $12 < 16
                        # ($2 = 0 se $12 ≥ 16)
```

Mas, se todas as instruções do MIPS ocupam um espaço de armazenamento de 32 bits, quantos desses 32 bits são dedicados a armazenar o valor imediato?

Estas instruções são codificadas no formato R, logo a resposta é 16 bits.

Este espaço é geralmente suficiente para armazenar constantes mais frequentemente utilizadas (geralmente valores pequenos).



Se h   apenas 16 bits dedicados ao armazenamento da constante, qual ser   a gama de representa  o da constante?

Depende...

No caso mais geral, a constante representa uma quantidade inteira, positiva ou negativa, codificada em **complemento para dois**. No caso das instru  es:

**addi** \$3, \$5, -4 ADD Immediate (signed)  
**slti** \$12, \$10, 0xFFFF

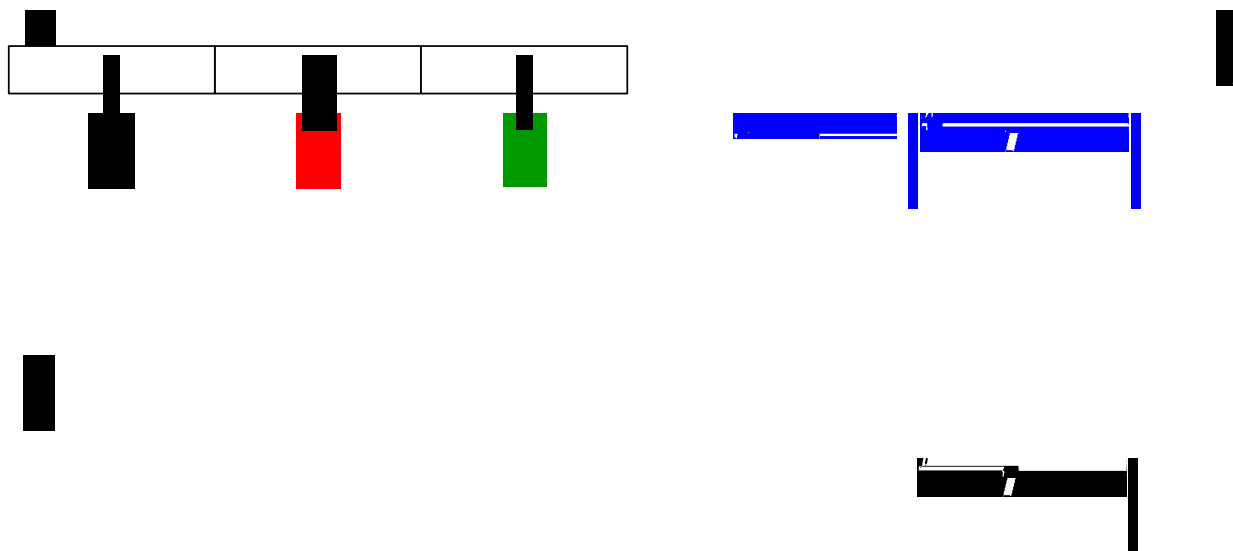
A gama de representa  o da constante ser   [-32768, +32767]

Existem tamb  m instru  es em que a constante    codificada como uma quantidade inteira sem sinal. Por exemplo todas as instru  es l  gicas:

**andi** \$3, \$5, 0xFFFF AND Immediate

A gama de representa  o da constante ser   [0, 65535]

O formato de codifica  o das instru  es que t  m s  s ent  s:

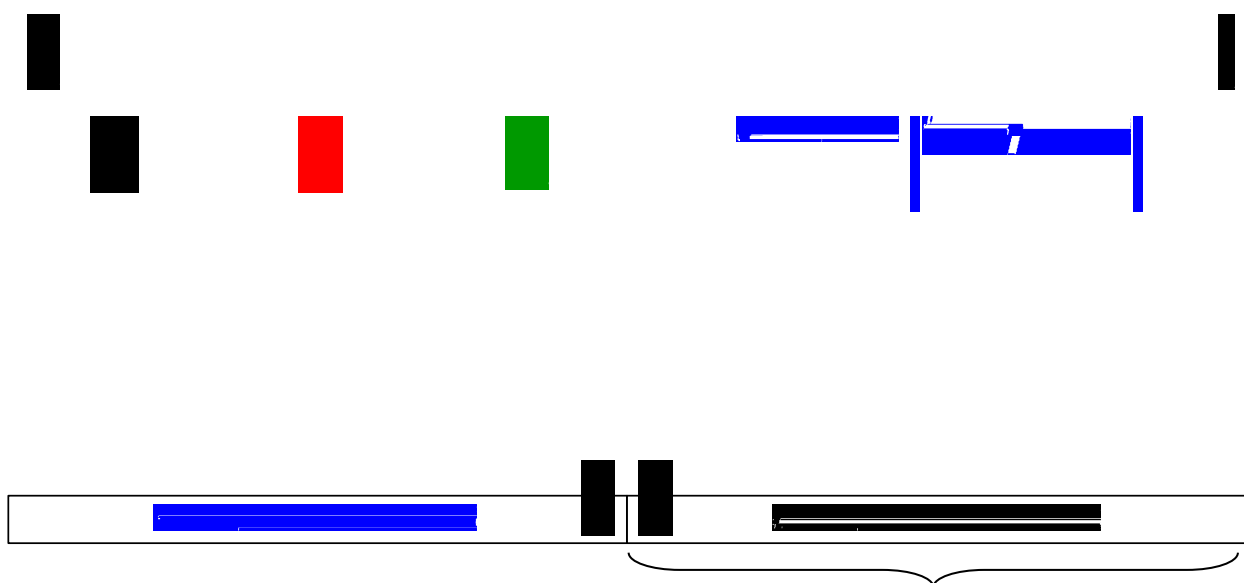


Pode ser necessário referenciar constantes que **excedem** de um espaço de armazenamento com mais do que 16 bits (como, por exemplo, a referência explícita a um endereço global com esses casos?

Para facilitar a manipulação de **immediates** com **mais de 16 bits**, o set de instruções do MIPS inclui a seguinte instrução:

`lui $reg, immediate`

**"lui Load upper immediate"**  
coloca a constante **immediate** nos **bits mais significativos do registo destino** (também uma instrução do tipo I)



Valor que fica armazenado em \$8 = 0x00FF0000

Os 16 bits menos significativos ficam com o valor 0

Desta forma, a **instrução virtual address**

```
la $16, MyData # Ex. MyData = 0x10010034
```

será executada no MIPS pela sequência de **instruções nativas**

```
lui $1, 0x1001 # $1 = 0x10010000
ori $16, $1, 0x0034 # $16 = 0x10010000 | 0x00000034
```

#### Notas:

O **registo \$1** é reservado para o *Assembler*, para permitir este tipo de decomposição de instruções virtuais em instruções nativas. A instrução **lui (\$1, *immediate*)** é decomposta em instruções nativas de forma análoga à instrução **ori**.

