ISA: Suporte para inteiros compactos, texto e constantes de 32 bits. Modos de endereçamento.

Revisão: inteiros com sinal

- **Complemento de dois**

$$-2^{n-1} \times bit_{n-1} + 2^{n-2} \times bit_{n-2} \ldots 2^{2} \times bit_{2} + 2^{1} \times bit_{1} + 2^{0} \times bit_{0}$$

- **Faixa: [- 2^{n-1} ; + 2^{n-1})**
- **Uso: variáveis e constantes inteiras**

Revisão: inteiros com sinal

- **Complemento de dois**

$$-2^{n-1} \times \text{bit}_{n-1} + 2^{n-2} \times \text{bit}_{n-2} \ldots 2^{2} \times \text{bit}_{2} + 2^{1} \times \text{bit}_{1} + 2^{0} \times \text{bit}_{0}$$

- **Faixa: [- 2^{n-1} ; + 2^{n-1})**
- **Uso: variáveis e constantes inteiras**
- **Como lidar com inteiros sinalizados com representação compacta?**
  - **Memória: 8 e 16 bits (char e short)**
  - **Registradores: 32 bits**

Revisão: extensão de sinal

- **Copiar o bit de sinal nos MSBs**
- **Exemplo:**

  $101 = -2^2 + 2^0 = -3$

  $1101 = -2^3 + 2^2 + 2^0 = -3$

  $11101 = -2^4 + 2^3 + 2^2 + 2^0 = -3$

  – **Aplicação**

    » **Conversão de 8 para 32 bits**

    » **Conversão de 16 para 32 bits**

Instruções de suporte no MIPS

- **Load sinalizado**
  - **Carrega o número nos LSBs**
  - **E copia o sinal nos MSB**
    - » **lb**
    - » **lh**
- **Load não sinalizado**
  - **Carrega o número nos LSBs**
  - **E preenche com zeros os MSB**
    - » **lbu**
    - » **lhu**

Suporte para char e string

- **Computadores**
  - **Inicialmente: inventados para fazer cálculos**
  - **Posteriormente: também processar texto**
- **Representação de caracteres em 8 bits**
  - **ASCII (Figura 2.15)**
- **Discussão:**
  - **ISA tem instruções para extrair byte de palavra**
    - » **lw e sw são suficientes para transferir bytes/palavras**
  - **Mas devido à demanda de texto em programas**
    - » **ISA inclui instruções para mover bytes: lbu e sb**

Suporte para char e string

- **Instruções para transferência de bytes**
  - **lbu $t0, 0 ($sp)**
    - » **Carrega um byte da memória nos 8 LSBs de $t0**
  - **sb $t0, 0 ($sp)**
    - » **Armazena os 8 LSBs de $t0 num byte da memória**
- **Caracteres são combinados em strings**
  - **Três alternativas de representação:**
    - » **Primeira posição do string armazena seu comprimento**
    - » **Uma variável associada armazena seu comprimento**
    - » **A última posição do string tem um caractere que marca o fim do string (adotada na linguagem C)**

Uma rotina para copiar string

```
void strcpy (char x[ ], char y[ ])
{    /* copia string y para string x */
      int i;
      i = 0;
      while  ( (x[i] = y[i]) !='\0' ) /* copia e testa byte */
      i += 1;
}
```

- **Alocação de registradores:**
  - **$a0 e $a1: endereços-base dos arranjos x e y**
  - **$s0: variável i**

Uma rotina para copiar string

```
void strcpy (char x[ ], char y[ ])
{   /* copia string y para string x */
     int i;
     i = 0;
     while  ( (x[i] = y[i]) !='\0' ) /* copia e testa byte */
     i += 1;
}
```

- **Salvamento de contexto pela rotina chamada**

```
strcpy:
            addi    $sp, $sp, -4
            sw      $s0, 0($sp)      #  push $s0
```

Uma rotina para copiar string

```
void strcpy (char x[ ], char y[ ])
{   /* copia string y para string x */
     int i;
     i = 0;
     while  ( (x[i] = y[i]) !='\0' ) /* copia e testa byte */
     i += 1;
}
```

- **Inicialização**

```
add     $s0, $zero, $zero
```

Uma rotina para copiar string

```
void strcpy (char x[ ], char y[ ])
{   /* copia string y para string x */
     int i;
     i = 0;
     while  ( (x[i] = y[i]) !='\0' ) /* copia e testa byte */
     i += 1;
}
```

- **Início do laço: acesso a y[i] e x[i]**

```
L1:     add     $t1, $s0, $a1    # $t1 = endereço de y[i]
        lbu     $t2, 0($t1)      # $t2 = y[i]
        add     $t3, $s0, $a0    # $t3 = endereço de x[i]
        sb      $t2, 0($t3)      # x[i] = y[i]
```

Uma rotina para copiar string

```
void strcpy (char x[ ], char y[ ])
{   /* copia string y para string x */
     int i;
     i = 0;
     while  ( (x[i] = y[i]) !='\0' ) /* copia e testa byte */
     i += 1;
}
```

- **Teste de saída do laço**

```
beq     $t2, $zero, L2   # se y[i] = 0, vá para L2
```

Uma rotina para copiar string

```
void strcpy (char x[ ], char y[ ])
{   /* copia string y para string x */
     int i;
     i = 0;
     while  ( (x[i] = y[i]) !='\0' ) /* copia e testa byte */
     i += 1;
}
```

- **Incremento do índice e desvio para início do laço**

```
addi    $s0, $s0, 1      # i = i + 1
j L1                     # vá para L1
```

Uma rotina para copiar string

```
void strcpy (char x[ ], char y[ ])
{   /* copia string y para string x */
     int i;
     i = 0;
     while  ( (x[i] = y[i]) !='\0' ) /* copia e testa byte */
     i += 1;
}
```

- **Saída do laço (caractere nulo atingido)**

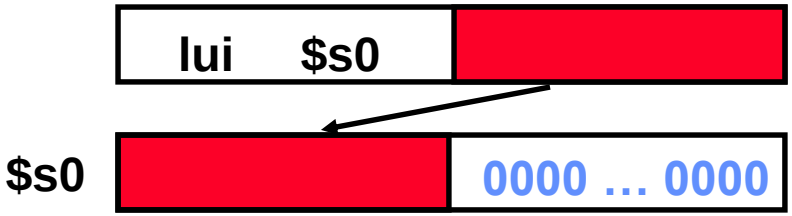
```
L2: lw      $s0, 0($sp)
    addi    $sp, $sp, 4      # pop $s0
    jr      $ra              # retorna à rotina chamadora
```

Suporte para representação de alfabeto

- **Codificação universal**
  - **Unicode (Figura 2.26)**
  - **Caractere representado em 16 bits**
    - » **Exemplo: Java**
- **Instruções para transferência de 16 bits**
  - **lhu $t0, 0 ($sp)**
    - » **Carrega meia palavra da memória nos 16 LSBs de $t0**
  - **sh $t0, 0 ($sp)**
    - » **Armazena os 16 LSBs de $t0 numa meia palavra da memória**

Suporte para constantes de 32 bits

- **Constantes pequenas**
  - **Formato I, campo de imediato 16 bits:**
    - » **Faixa de representação: [-2^{15}, $+2^{15}$)**
- **Suporte em HW para constantes grandes**
  - **lui: load upper immediate**



Suporte para constantes de 32 bits

- **Exemplo: x = 4000000;**
  - **Alocação de x em $s0**

    $s0 ← 0000 0000 0011 1101 0000 1001 0000 0000
  - **Quebra da constante em duas partes**

    0000 0000 0011 1101 = $61_{decimal}$

    0000 1001 0000 0000 = $2304_{decimal}$
- **Código em linguagem de montagem**

```
lui $s0, 61          # $s0=0000 0000 0011 1101 0000 0000 0000 0000
ori $s0, $s0, 2304   # $s0=0000 0000 0011 1101 0000 1001 0000 0000
```

Interface HW/SW

- **Compilador ou montador**
  - **Constantes grandes quebradas em pedaços**
  - **Pedaços “emendados” em um registrador**
- **Restrição de formato (campo de imediato)**
  - **Restringe acesso de “load/store” e constantes**
- **Recomposição de constantes**
  - **Pode precisar de um registrador temporário**
  - **Se recomposição feita pelo montador**
    - » **Temporário não pode ser visível ao “programador”**
  - **Registrador reservado p/ montador : $at**

Exemplo

- **Hipótese:**
  - **big é constante cuja representação exige 32 bits**
  - **big = big_H || big_L**
- **Pseudo-instrução**

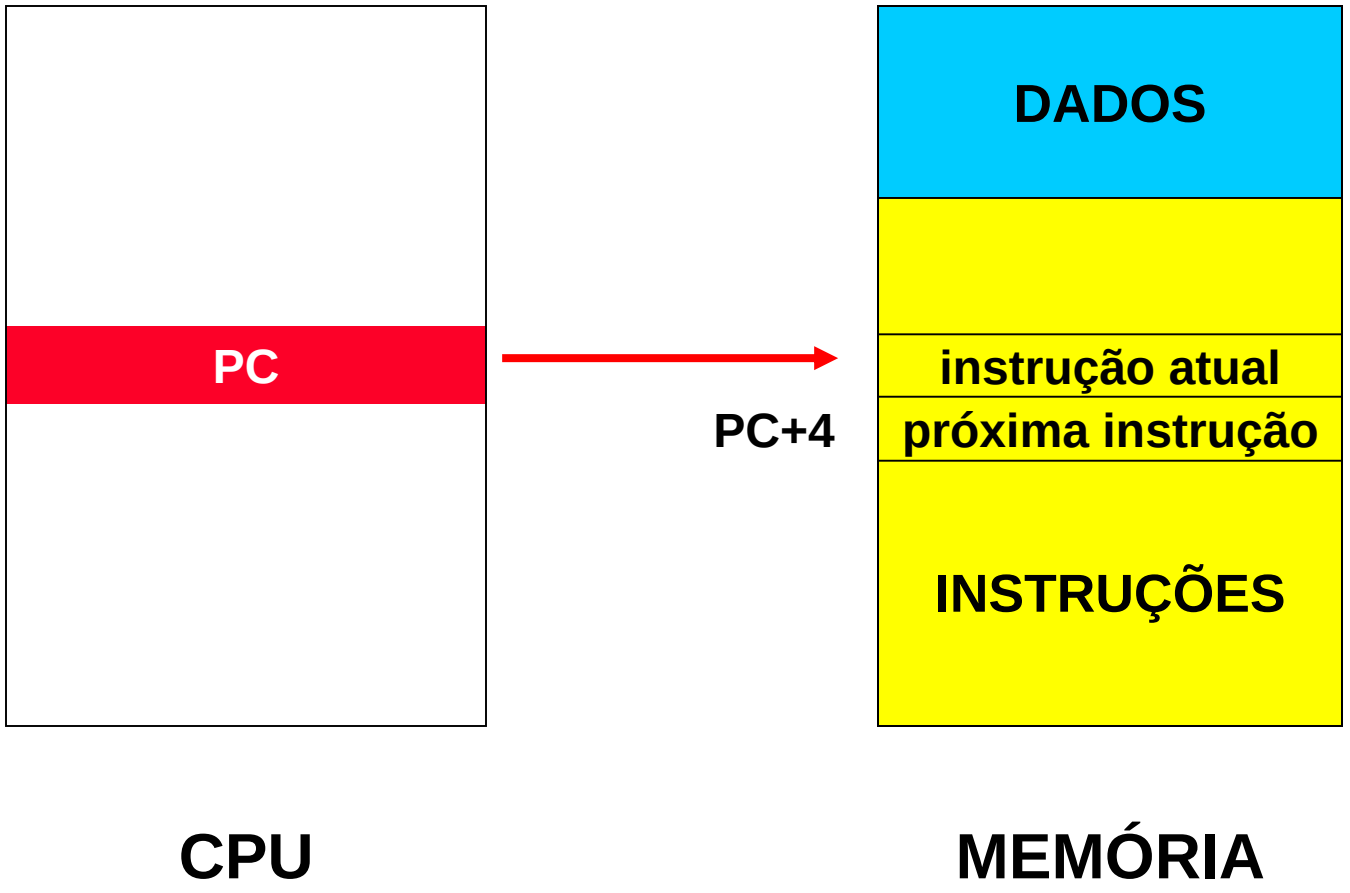
  addi $t5, $t3, big
- **Código que implementa a pseudo-instrução**

  lui $at, big_H

  ori $at, $at, big_L

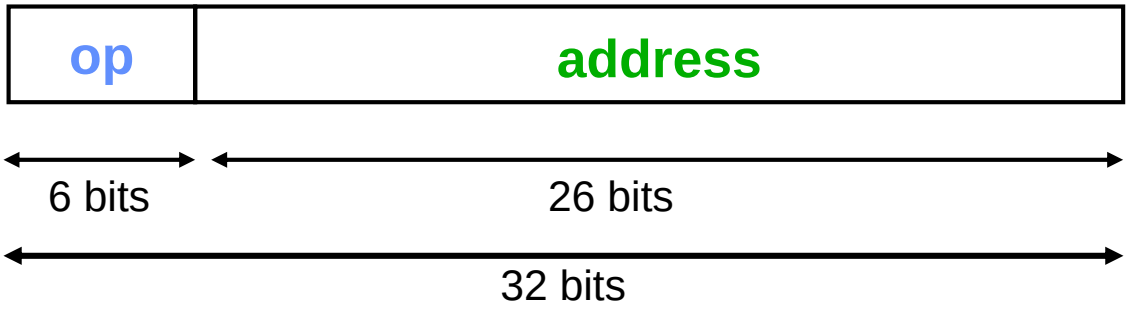
  add $t5, $t3, $at
- **É mesmo obrigatório usar $at ?**

Fluxo sequencial de controle



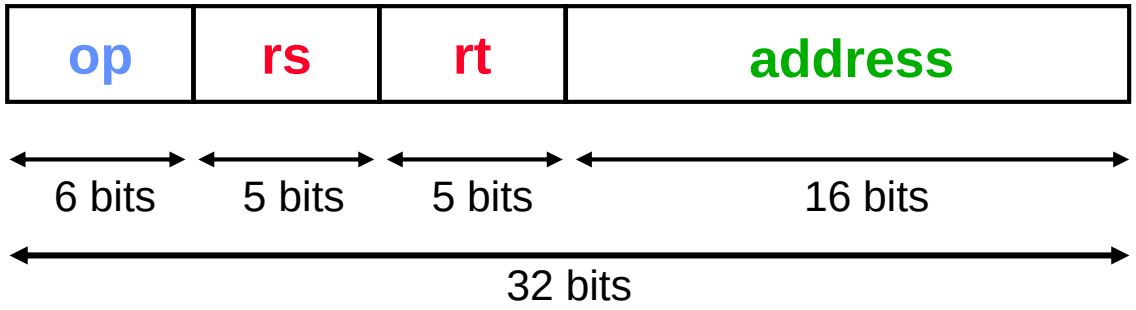
Desvios: endereçamento

- **Desvio incondicional: formato J**



j Exit

- **Desvio condicional: formato I**



bne $s0,$s1,Exit

Restrição à representação de endereços

Endereço (0x)		Código	
0000 0000	**loop:**	...	
0000 0004		...	
0000 0008		...	
0000 000C		bne $s1, $s2, **loop**	

Restrição à representação de endereços

Endereço (0x)		Código	
0000 0000	**loop:**	…	
0000 0004		…	
0000 0008		…	
0000 000C		bne $s1, $s2, **loop**	

Endereço (0x)		Código	
FFFF 0000	**loop:**	…	
FFFF 0004		…	
FFFF 0008		…	
FFFF 000C		bne $s1, $s2, **loop**	

Endereçamento relativo

- **Desvios condicionais: restrição de formato**
  - **Endereço: 16 bits**
  - **Tamanho do programa $\leq 2^{16}$???**
- **Solução:**
  - **Obter um endereço de 32 bits**
    - » **4G bytes endereçáveis**
  - **Especificar um registrador de referência**
    - » **PC = Registrador + endereço do desvio**
  - **Que registrador usar como referência ?**
    - » **Analisar propriedades de programas**

Endereçamento relativo

- **Propriedades de desvios**
  - **Instrução-alvo próxima da instrução de teste**
    - » **Cláusulas if-then-else e corpos de laços contêm poucos comandos**
  - **SPEC 2000 benchmarks**
    - » **Em 50% dos desvios, deslocamento < 16 instruções**
- **Solução**
  - **Desvio relativo à instrução corrente**
    - » **PC = Registrador + endereço do desvio**

Endereçamento relativo

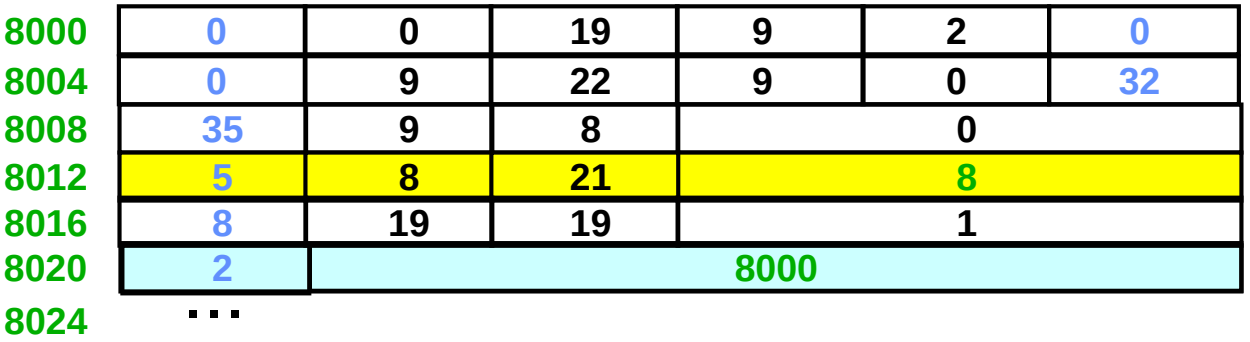
- **Propriedades de desvios**
  - **Instrução-alvo próxima da instrução de teste**
    - » **Cláusulas if-then-else e corpos de laços contêm poucos comandos**
  - **SPEC 2000 benchmarks**
    - » **Em 50% dos desvios, deslocamento < 16 instruções**
- **Solução**
  - **Desvio relativo à instrução corrente**
    - » **PC = PC + endereço do desvio**

Endereçamento relativo

- **Propriedades de desvios**
  - **Instrução-alvo próxima da instrução de teste**
    - » **Cláusulas if-then-else e corpos de laços contêm poucos comandos**
  - **SPEC 2000 benchmarks**
    - » **Em 50% dos desvios, deslocamento < 16 instruções**
- **Solução**
  - **Desvio relativo à instrução corrente**
    - » **PC = PC + deslocamento**
    - » **Deslocamento: $[-2^{15}, +2^{15})$**

Exemplo: “while”

```
Loop:   sll   $t1,$s3, 2     # $t1 = 4 × i
        add   $t1,$t1,$s6    # $t1 = end. de save[i]
        lw    $t0,0($t1)     # $t0 = save[i]
        bne   $t0,$s5, Exit  # desvie se save[i] ≠ k
        addi  $s3,$s3, 1     # i = i + 1
        j     Loop
Exit:   ...
```



Mas no MIPS, deslocamento codificado em palavras
⇒ deslocamento = # bytes / 4

Exemplo: “while”

```
Loop:   sll   $t1,$s3, 2     # $t1 = 4 × i
        add   $t1,$t1,$s6    # $t1 = end. de save[i]
        lw    $t0,0($t1)     # $t0 = save[i]
        bne   $t0,$s5, Exit  # desvie se save[i] ≠ k
        addi  $s3,$s3, 1     # i = i + 1
        j     Loop
Exit:   ...
```

8000	0	0	19	9	2	0
8004	0	9	22	9	0	32
8008	35	9	8	0		
8012	5	8	21	2		
8016	8	19	19	1		
8020	2	2000				
8024	...					

Mas no MIPS, deslocamento codificado em palavras
⇒ deslocamento = # bytes / 4

Interface HW/SW

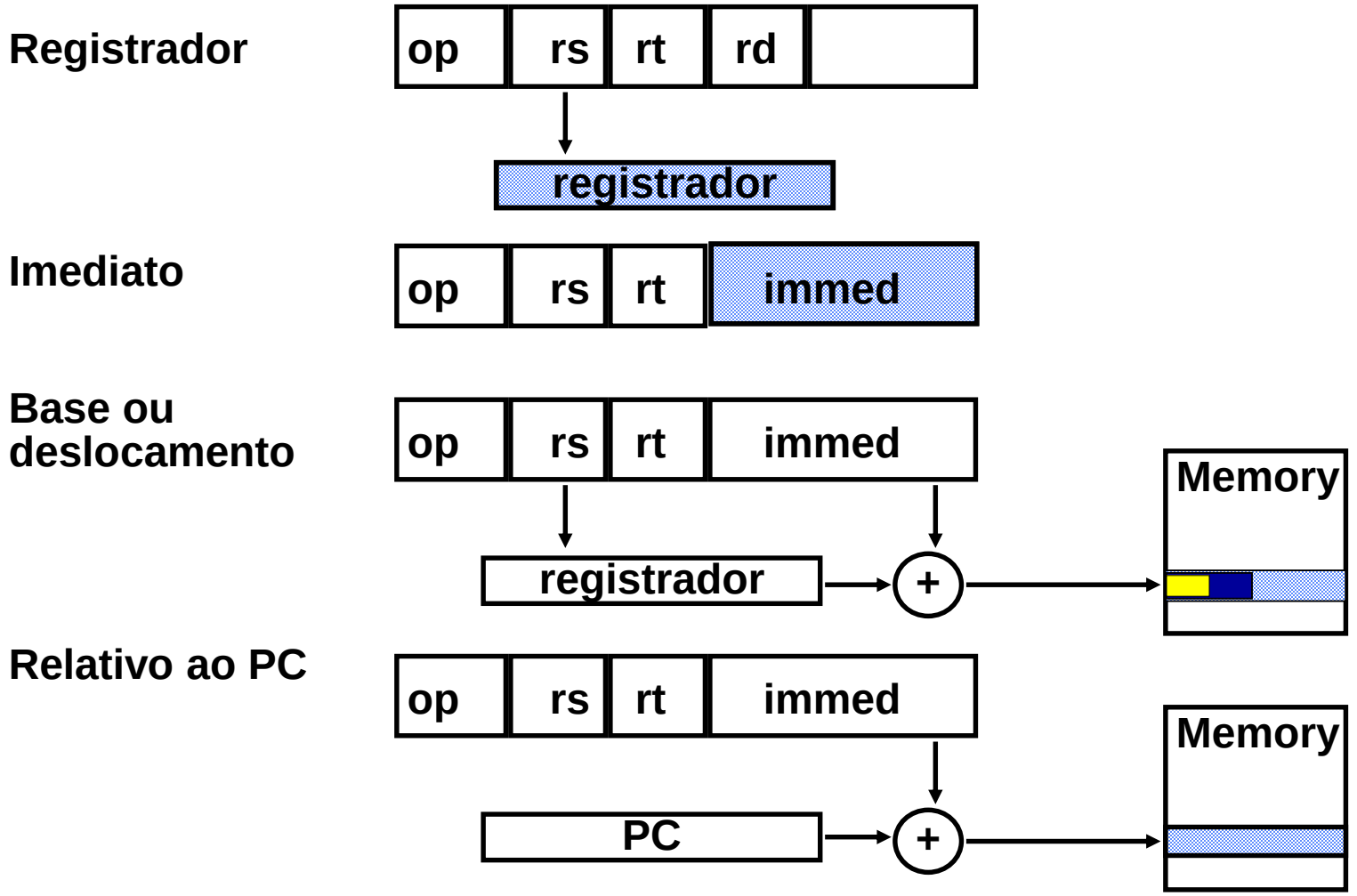
- **Desvios condicionais**
  - **A maioria tem alvo próximo**
  - **Se deslocamento não representável em 16 bits ?**

```
beq     $s0, $s1, L1
...
```

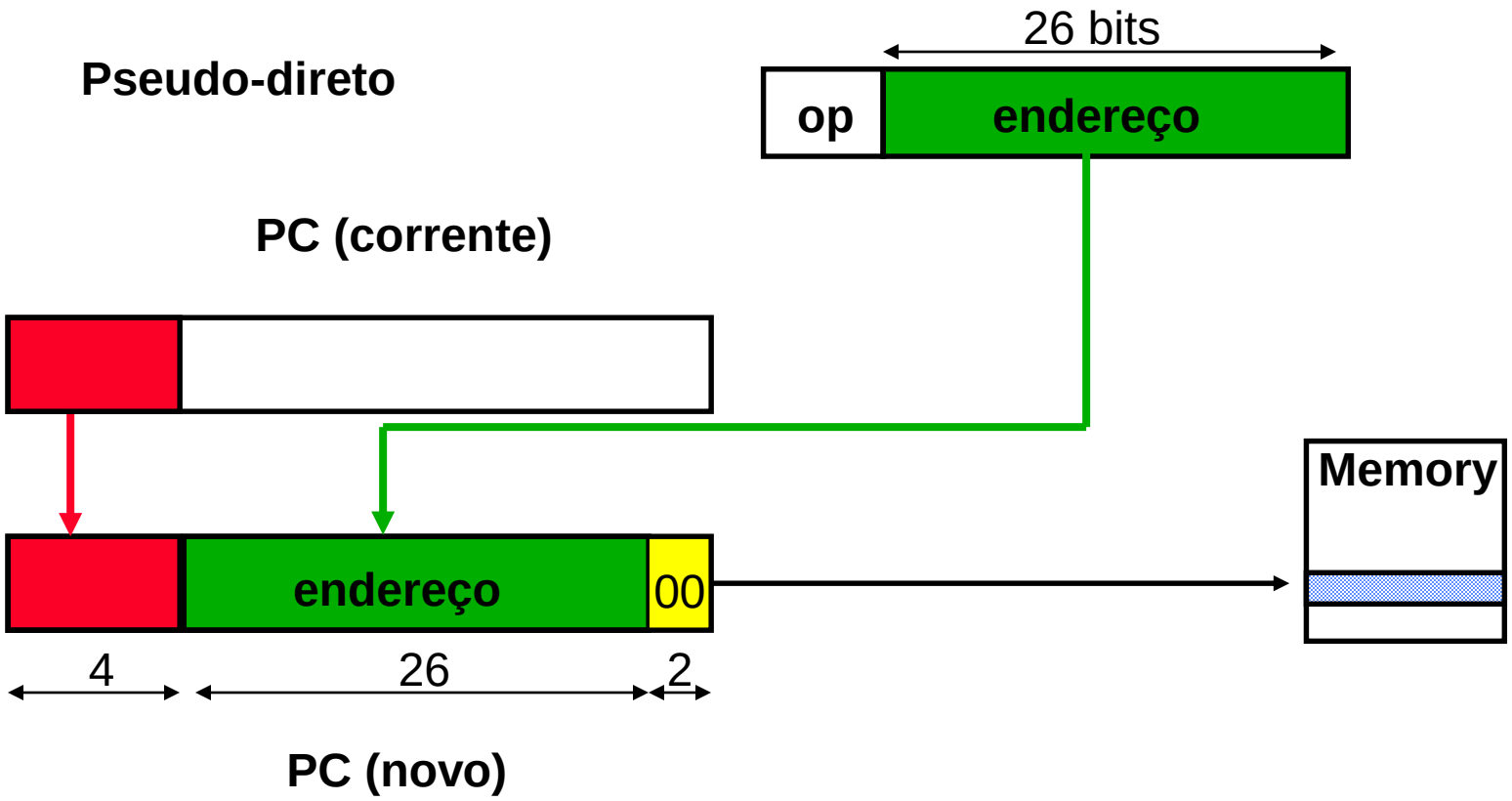
- **Solução: ajuste no montador ou ligador**
  - **Insere desvio incondicional para o alvo**
  - **Condição do desvio condicional é invertida**

```
    bne   $s0,$s1, L2
    j     L1
L2: . . .
```

Modos de endereçamento do MIPS



Modos de endereçamento do MIPS



Conclusão 1: desvios incondicionais não-relocáveis

Endereço (0x)		Código	
0000 0000	loop:	...	
0000 0004		...	
0000 0008		...	
0000 000C		j loop	

Valores diferentes!

Endereço (0x)		Código	
0FFF 0000	loop:	...	
0FFF 0004		...	
0FFF 0008		...	
0FFF 000C		j loop	

Conclusão 2: desvios condicionais automaticamente relocáveis

Endereço (0x)		Código	
0000 0000	loop:	...	
0000 0004		...	
0000 0008		...	
0000 000C		bne $s1, $s2, loop	

-16

Valores iguais!

Endereço (0x)		Código	
0FFF 0000	loop:	...	
0FFF 0004		...	
0FFF 0008		...	
0FFF 000C		bne $s1, $s2, loop	

-16

ISA: Suporte para inteiros compactos, texto e constantes de 32 bits. Modos de endereçamento.