1º Semestre de 2007/2008

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Aula 6

Instruções de controlo de fluxo de execução

Métodos de endereçamento em “saltos” condicionais e incondicionais

Instruções do tipo *J*

3. Instruções de controlo de fluxo de execução

"O que distingue um computador de uma calculadora barata é a capacidade de tomar decisões com base em valores que não são conhecidos à priori."

A capacidade de decidir e realizar uma de várias tarefas com base num critério de verdade ou falsidade determinado durante a execução (conhecido como instrução *if()* nas linguagens de alto nível) é possibilitado no assembly do MIPS pelas instruções:

```
beq    Rsrc1, Rsrc2, Label    # branch if equal
bne    Rsrc1, Rsrc2, Label    # branch if not equal
```

e são conhecidas como "*branches*" (saltos) condicionais

beq Rsrc1, Rscr2, Label # branch if equal

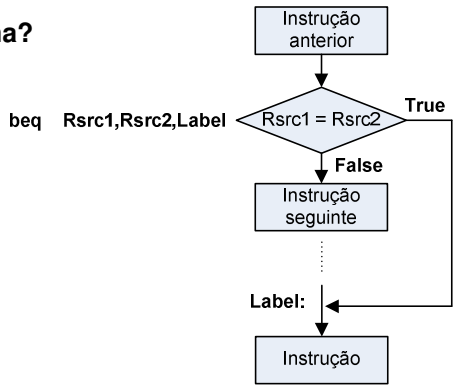
Salta para a instrução situada no endereço representado por "Label", **se** os conteúdos dos registos **Rsrc1** e **Rsrc2** forem i**guais**

Como funciona?

- Se a condição testada na instrução for verdadeira (no caso anterior Rsrc1=Rsrc2, isto é **Rsrc1 – Rsrc2 = 0**), o valor corrente do PC (Program Counter) é substituído pelo endereço a que corresponde "Label"
- Assim, a próxima instrução a ser executada é a que se situa nesse endereço

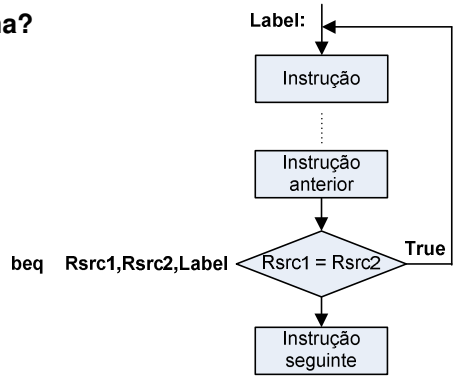
beq Rsrc1, Rscr2, Label # branch if equal

Como funciona?



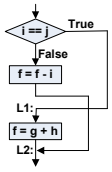
beq Rsrc1, Rscr2, Label # branch if equal

Como funciona?



Exemplo: considere o seguinte trecho de código C:

```
        if (i == j) goto L1; /* ☹ */
        f = f - i;
        goto L2;             /* ☹ */
L1:     f = g + h;
L2:     ...
```



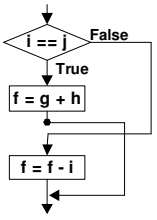
O código equivalente em *Assembly* do MIPS seria (considerando i, j, f, g e h em $19, $20, $16, $17 e $18, respectivamente):

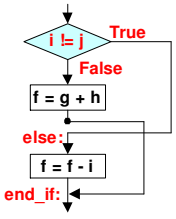
```
        beq     $19, $20, L1  # Se $19 == $20 salta para L1
        sub     $16, $16, $19 # subtrai $19 a $16 e armazena em $16
        j       L2            # goto L2
L1:     add     $16, $17, $18 # soma $17 com $18 e armazena em #16
L2:     ...
```

j significa ***jump*** e representa um salto **incondicional** para o endereço indicado

Um exemplo mais realista (sem o *Goto* ☺) seria:

```
if (i == j) {
        f = g + h;
} else {
        f = f - i;
}
```





A que corresponderia o código *Assembly*:

```
        bne     $19, $20, ELSE # if ($19 == $20) {
        add     $16, $17, $18  #       f = g + h;
        j       END_IF         #
ELSE:                          # } else {
        sub     $16, $16, $19  #       f = f - i;
END_IF:                        # }
```

São ainda suportadas um conjunto de instruções que comparam directamente com zero:

```
bltz    Rsrc, Label     # Branch if Rsrc < 0
blez    Rsrc, Label     # Branch if Rsrc ≤ 0
bgtz    Rsrc, Label     # Branch if Rsrc > 0
bgez    Rsrc, Label     # Branch if Rsrc ≥ 0
```


Para além das instruções de salto com base no critério de igualdade e desigualdade, o MIPS suporta ainda a instrução:

slt Rdst, Rsrc1, Rsrc2

```
# slt ≡ "set if less than"
# set Rdst if Rsrc1 < Rsrc2
```

Descrição: O registo "Rdst" tomará o valor "1" caso o conteúdo do registo "Rsrc1" seja inferior ao do registo "Rsrc2". Caso contrário toma o valor "0".

A utilização das instruções "**bne**", "**beq**" e "**slt**", em conjunto com o registo **$0**, permitem a implementação de todas as condições de comparação: (**A = B**), (**A ≠ B**), (**A > B**), (**A ≥ B**), (**A < B**), (**A ≤ B**)

Exemplo1: Decompor em instruções nativas a **instrução virtual** seguinte *(**b**ranch if **g**reater **t**han):*

```
        bgt     $4, $7, exit      # Goto exit IF $4 > $7
```

Resposta:

```
        slt     $1, $7, $4        # $1 = 1 if $7 < $4 (i.e. $4 > $7)
        bne     $1, $0, exit      # if $1 ≠ 0 goto exit
```

Exemplo2: Decompor em instruções nativas a **instrução virtual** seguinte *(**b**ranch if **g**reater or **e**qual than):*

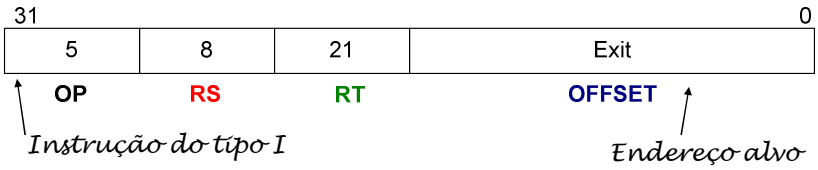
```
        bge     $4, $7, exit      # Goto exit IF $4 ≥ $7
```

Resposta:

```
        slt     $1, $4, $7        # $1 = 1 if $4 < $7 (i.e. !($4 ≥ $7))
        beq     $1, $0, exit      # if $1 = 0 goto exit
```


Representação de *branches* e *jumps* em código máquina.

Exemplo: **bne $8, $21, Exit**



bne rs, rt, offset

Mas se o endereço alvo é codificado com 16 bits, isso significa que o programa não pode ter uma dimensão superior a 2^{16} (64K). **Será essa uma opção realista?**

- Uma alternativa seria especificar um registo interno cujo conteúdo pudesse ser somado ao endereço alvo codificado na instrução, de tal modo que **PC = Reg + Branch_Offset** (desta forma a dimensão máxima do programa já poderia ser 2^{32})
- **Mas qual o registo a usar?** Comecemos por observar que:
  - A maioria das instruções de salto condicional fazem-se para a vizinhança da instrução (exemplo: no gcc, quase metade de todos os saltos condicionais são para endereços correspondentes a uma gama de ± 16 instruções)
  - Com 16 bits é possível endereçar 2^{16} endereços distintos (64K endereços)

Resposta: Utilizar o registo PC (Program Counter)

Usando **endereçamento relativo** – o valor do endereço alvo é calculado somando os 16 bits presentes na instrução ao valor corrente do PC **(no MIPS esse valor corresponde ao endereço da instrução seguinte, uma vez que o valor do PC é incrementado antes da fase *execute* da instrução)**. Isto é:

PC = PC + Branch_Offset

Note-se que a constante de 16 bits é interpretada como um valor em complemento para dois, permitindo o *branch* para endereços anteriores ou posteriores 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:
```

Durante o *instruction fetch* o PC é incrementado (i.e. PC=0x00400004)

O endereço correspondente ao label ELSE é 0x0040000C

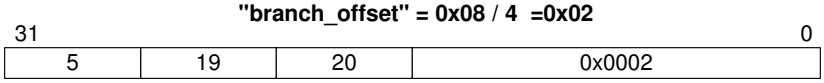
O "branch_offset" seria portanto:

ELSE - (PC) =

0x0040000C - 0x00400004 = 0x08

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

"branch_offset" = 0x08 / 4 =0x02



Uma instrução de salto condicional pode assim referenciar qualquer endereço de uma outra instrução que se situe até 32K instruções antes ou depois dela própria.

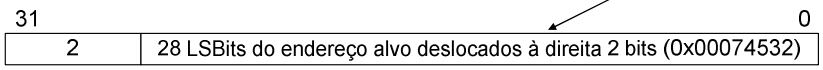
E no caso do salto incondicional (*jump*)?

Nesse caso estamos perante uma situação de **endereçamento directo** (a instrução especifica o valor do endereço do qual será lida a próxima instrução).

Exemplo: A instrução `j Label # Label = 0x001D14C8`

corresponderá a:

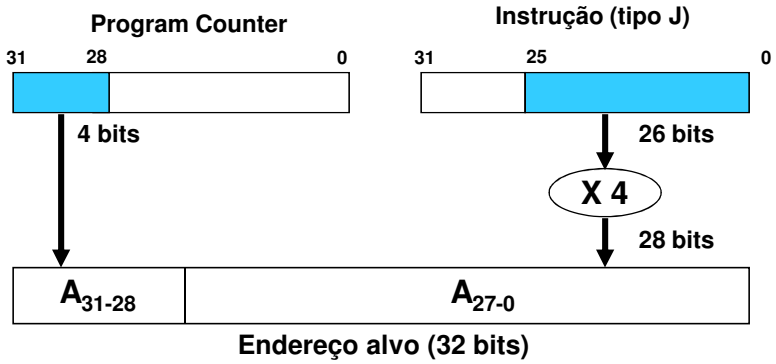
0x001D14C8 / 4 = 0x00074532



6 bits 26 bits

Instrução do tipo J

Mas se a instrução só especifica 28 bits (26 + 2), como é formado o endereço final de 32 bits?



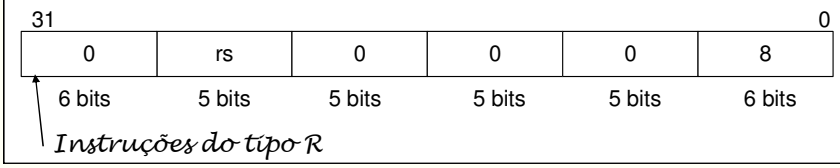
Já agora! E não haverá maneira de especificar um endereço para um *salto,* usando para isso os 32 bits?

Há! Só que para isso é necessário recorrer ao conteúdo de um registo para funcionar como ponteiro para o endereço alvo.

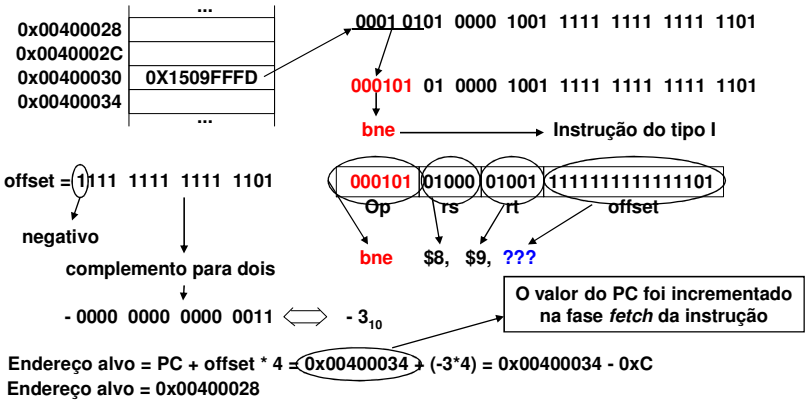
Instrução: jr Rsrc **# "Jump register" – salta para a instrução que se # encontra no endereço igual ao conteúdo do # registo Rsrc**

Exemplo: jr $t0 #

O código máquina correspondente será:



Exemplo! Como interpreta o CPU uma instrução?



instrução descodificada: **bne $8, $9, 0x00400028**