

## Aula 4

- Instruções de controlo de fluxo de execução
- Estruturas de controlo de fluxo de execução:
  - **if()...then...else**
  - Ciclos “**for()**”, “**while()**” e “**do...while()**”

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira, Tomás Oliveira e Silva

## 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:

<b>beq</b>	<b>Rsrc1, Rsrc2, Label</b>	# branch <b>if equal</b>
<b>bne</b>	<b>Rsrc1, Rsrc2, Label</b>	# branch <b>if not equal</b>

e são conhecidas como “**branches**” (saltos / *jumps*) **condicionais**

## beq Rsrc1, Rsrc2, Label # branch if equal

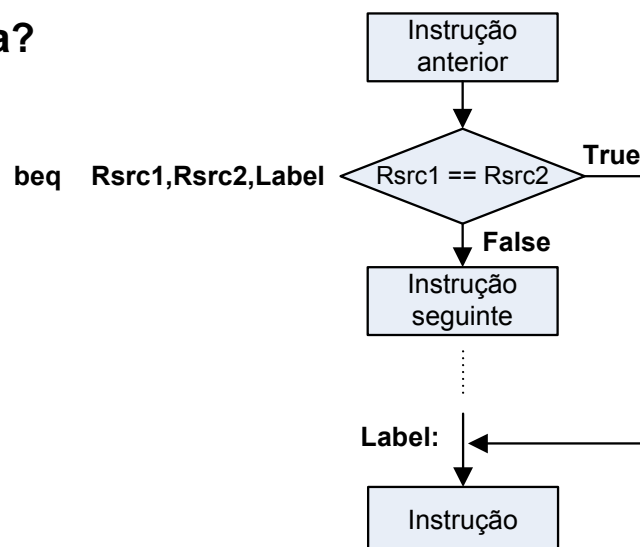
- Salta para a instrução situada no endereço representado por "Label", se os conteúdos dos registos Rsrc1 e Rsrc2 forem iguais
- O endereço para onde o salto é efectuado (no caso de a condição ser verdadeira) designa-se por **endereço-alvo**

### 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" (endereço-alvo)
  - Assim, a próxima instrução a ser executada é a que se situa nesse endereço
- Se a **condição for falsa**, a sequência de execução não é alterada, i.e., é executada a instrução que está imediatamente a seguir à instrução de *branch*

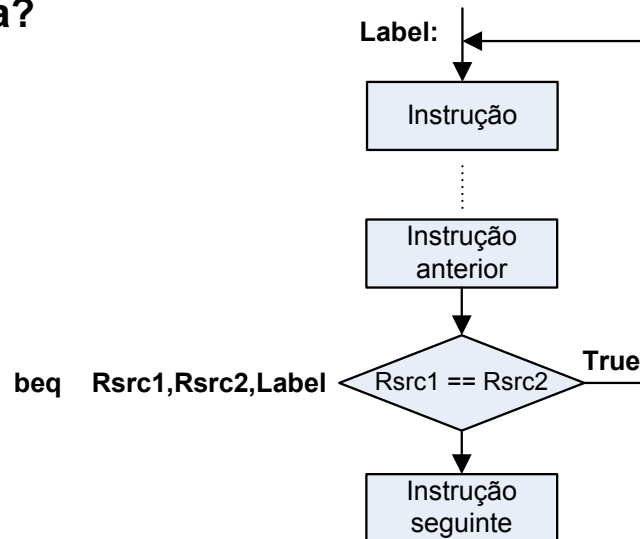
## beq Rsrc1, Rsrc2, Label # branch if equal

### Como funciona?



## beq Rsrc1, Rsrc2, Label # branch if equal

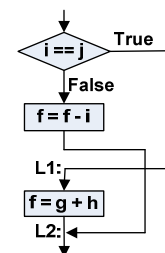
### Como funciona?



### 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* - \$19, *j* - \$20, *f* - \$16, *g* - \$17, e *h* - \$18):

```

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:    ...
  
```

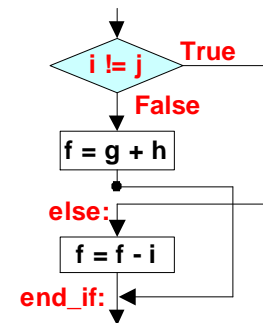
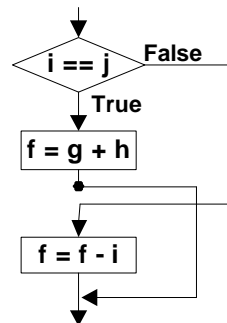
*j* significa **jump** e representa um **salto incondicional** para o endereço indicado

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

```

O ISA do MIPS suporta ainda um conjunto de instruções que comparam directamente com zero:

<b>bltz</b>	<b>Rsrc, Label</b>	<b># Branch if Rsrc &lt; 0</b>
<b>blez</b>	<b>Rsrc, Label</b>	<b># Branch if Rsrc ≤ 0</b>
<b>bgtz</b>	<b>Rsrc, Label</b>	<b># Branch if Rsrc &gt; 0</b>
<b>bgez</b>	<b>Rsrc, Label</b>	<b># Branch if Rsrc ≥ 0</b>

Pode ainda ser utilizadas, nos programas *assembly*, instruções de salto não directamente suportadas pelo MIPS (instruções virtuais), mas que são decompostas pelo *assembler* em instruções nativas do MIPS, nomeadamente:

Pode também ser uma constante

<b>blt</b>	<b>Rsrc1, Rsrc2, Label</b>	<b># Branch if Rsrc1 &lt; Rsrc2</b>
<b>ble</b>	<b>Rsrc1, Rsrc2, Label</b>	<b># Branch if Rsrc1 ≤ Rsrc2</b>
<b>bgt</b>	<b>Rsrc1, Rsrc2, Label</b>	<b># Branch if Rsrc1 &gt; Rsrc2</b>
<b>bge</b>	<b>Rsrc1, Rsrc2, Label</b>	<b># Branch if Rsrc1 ≥ Rsrc2</b>

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" toma o valor "1" se o conteúdo do registo "Rsrc1" for inferior ao do registo "Rsrc2". Caso contrário toma o valor "0".

```
slti Rdst, Rsrc1, Imm    # slt ≡ "set if less than"
                          # set Rdst if Rsrc1 < Imm
```

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

**Exemplo1:** Decompor em **instruções nativas** a **instrução virtual** seguinte (***branch if greater than***):

```
bgt      $4, $7, exit    # Goto exit IF $4 > $7
                        # (i.e. goto exit if $7 < $4)
```

**Solução:**

```
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 (***branch if greater or equal than***):

```
bge    $4, $7, exit    # Goto exit IF  $\$4 \geq \$7$ 
                        # (i.e. goto exit if  $!(\$4 < \$7)$ )
```

**Solução:**

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

## Estruturas de controlo de fluxo em *Assembly* do MIPS

Consideremos os seguintes trechos de código:

```
if (a >= n)
{
    b = c;
}
else
{
    b = d;
}...
```

```
for (n = 0; n < 100; n++)
{
    a = a + b[n];
}
...
```

```
n = 0;
do
{
    a = a + b[n];
    n++;
} while (n < 100);
...
```

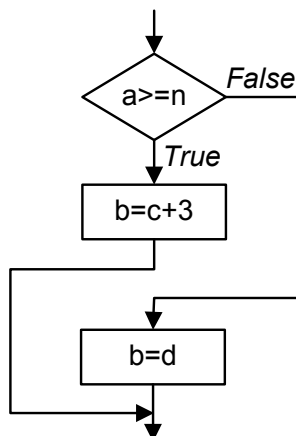
```
n = 0;
while (n < 100)
{
    a = a + b[n];
    n++;
}
...
```

### 1º Exemplo:

```
if (a >= n) {
    b = c + 3;
} else {
    b = d;
}
```

\$s1 ↔ a  
\$s2 ↔ n  
\$s3 ↔ c  
\$t0 ↔ b  
\$s4 ↔ d

Transformando o código apresentado no fluxograma equivalente:



- É possível identificar facilmente a ocorrência de um salto condicional e de um salto incondicional.

1º Exemplo:

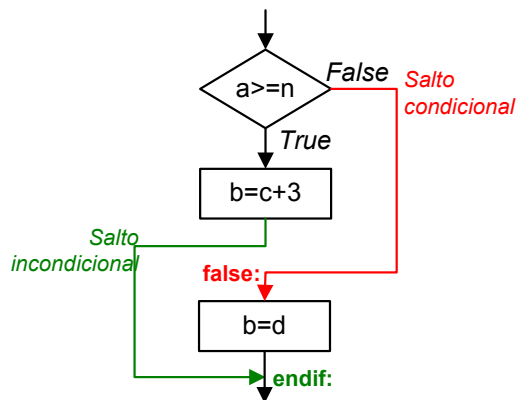
```

if (a >= n) {
    b = c + 3;
} else {
    b = d;
}

```

$\$s1 \leftrightarrow a$   
 $\$s2 \leftrightarrow n$   
 $\$s3 \leftrightarrow c$   
 $\$t0 \leftrightarrow b$   
 $\$s4 \leftrightarrow d$

Transformando o código apresentado no fluxograma equivalente:



- É possível identificar facilmente a ocorrência de um salto condicional e de um salto incondicional.
- E adaptar o salto condicional para que este se efectue quando a condição for verdadeira (tal como nos *branches*).

1º Exemplo:

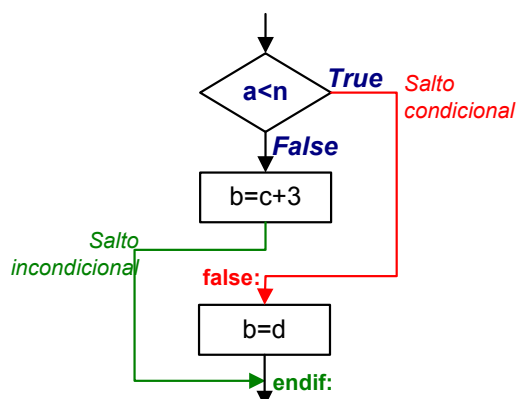
```

if (a >= n) {
    b = c + 3;
} else {
    b = d;
}

```

$\$s1 \leftrightarrow a$   
 $\$s2 \leftrightarrow n$   
 $\$s3 \leftrightarrow c$   
 $\$t0 \leftrightarrow b$   
 $\$s4 \leftrightarrow d$

Transformando o código apresentado no fluxograma equivalente:



- É possível identificar facilmente a ocorrência de um salto condicional e de um salto incondicional.
- E adaptar o salto condicional para que este se efectue quando a condição for verdadeira (tal como nos *branches*).

1º Exemplo:

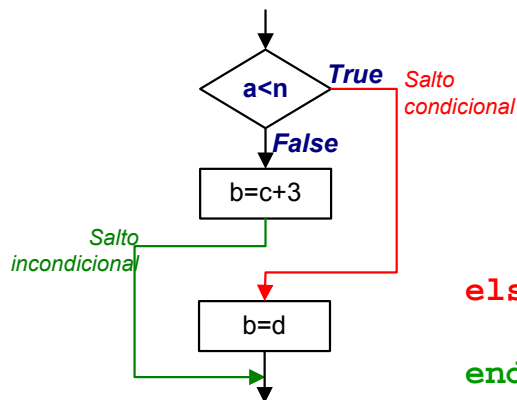
```

if (a >= n) {
    b = c + 3;
} else {
    b = d;
}

```

$\$s1 \leftrightarrow a$   
 $\$s2 \leftrightarrow n$   
 $\$s3 \leftrightarrow c$   
 $\$t0 \leftrightarrow b$   
 $\$s4 \leftrightarrow d$

Transformando o código apresentado no fluxograma equivalente:

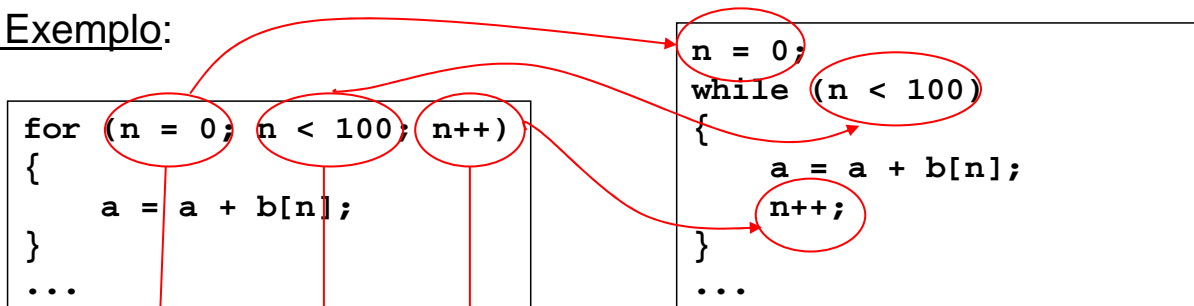


A tradução torna-se directa:

```

blt $s1,$s2,else # if (a >= n) {
addi $t0,$s3,3   #   b = c + 3;
j endif          # }
else:             # else {
    move $t0,$s4  #   b = d;
endif: ...        # }

```

2º Exemplo:

Estes dois exemplos são **funcionalmente equivalentes!**

Operações a executar **antes do corpo do ciclo** (inicializações)

Condição de continuação da execução do ciclo

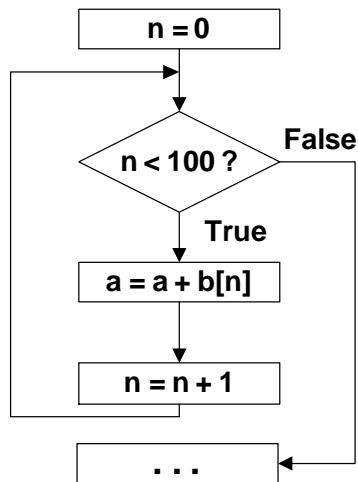
Operações a realizar no **fim do corpo do ciclo**



2º Exemplo:

```
for (n = 0; n < 100; n++)
{
    a = a + b[n];
    ...
}
```

```
n = 0;
while (n < 100)
{
    a = a + b[n];
    n++;
}
...
```

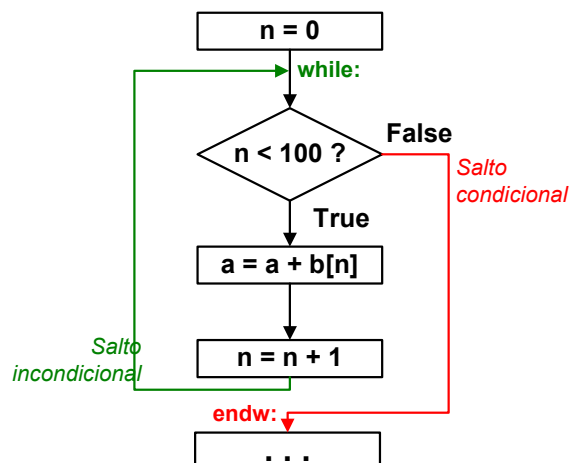


- É possível identificar a ocorrência de um salto condicional e de um salto incondicional.

2º Exemplo:

```
for (n = 0; n < 100; n++)
{
    a = a + b[n];
    ...
}
```

```
n = 0;
while (n < 100)
{
    a = a + b[n];
    n++;
}
...
```

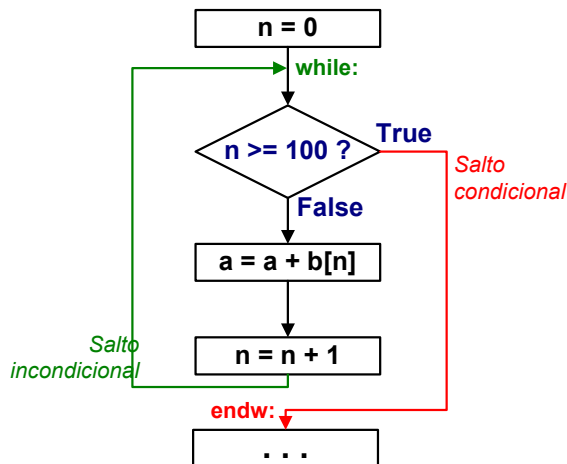


- É possível identificar a ocorrência de um salto condicional e de um salto incondicional.
- O salto condicional necessita de ser modificado de forma a ser efectuado quando a condição for verdadeira.

2º Exemplo:

```
for (n = 0; n < 100; n++)
{
    a = a + b[n];
}
...
```

```
n = 0;
while (n < 100)
{
    a = a + b[n];
    n++;
}
...
```

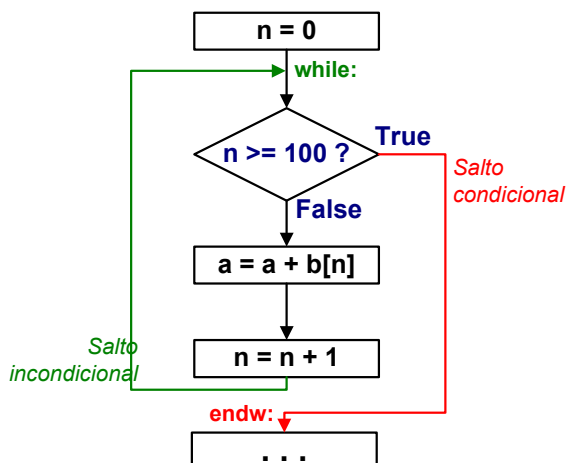


- É possível identificar a ocorrência de um salto condicional e de um salto incondicional.
- O salto condicional necessita de ser modificado de forma a ser efectuado quando a condição for verdadeira.

2º Exemplo:

```
for (n = 0; n < 100; n++)
{
    a = a + b[n];
}
...
```

```
n = 0;
while (n < 100)
{
    a = a + b[n];
    n++;
}
...
```



Traduzindo a estrutura de controlo:

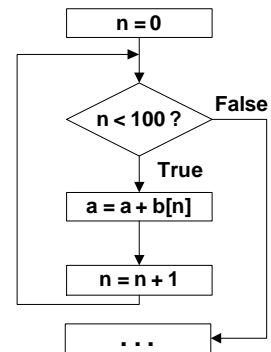
li	n,0	#n=0
while:bge	n,100,endw	#while(n<100)
		#{
.		# ...
.		# ...
addi	n,n,1	# n++;
j	while	#}
endw:		

**2º Exemplo:**

Note-se o uso do complemento lógico

```
for (n = 0; n < 100; n++)
{
    a = a + b[n];
}
...
```

```
n = 0;
while (n < 100)
{
    a = a + b[n];
    n++;
}
...
```



O código equivalente em Assembly MIPS:

(\$s1 ↔ n, \$s2 ↔ b, \$s3 ↔ a)

```

while:  li $s1, 0           # n = 0;
        bge $s1, 100, endw # while(n < 100) {
        sll $t0, $s1, 2     # temp = 4 * n;
        add $t0, $s2, $t0   # temp = temp + b;
        lw  $t0, 0($t0)     # temp = *temp;
        add $s3, $s3, $t0   # a = a + temp;
        addi $s1, $s1, 1    # n++;
        j   while          # }
endw:   ...
  
```

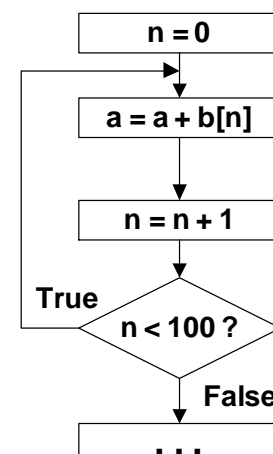
Teste condicional efectuado “à cabeça”

**3º Exemplo:**

Ao contrário do **for()** e do **while()**, o corpo do ciclo **do...while()** é executado incondicionalmente pelo menos uma vez!

```
n = 0;
do
{
    a = a + b[n];
    n++;
}while (n < 100);
...
```

```
li n, 0
do:
...
...
blt n, 100, do
```



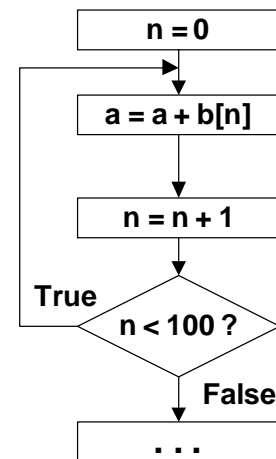
3º Exemplo:

\$s1 ↔ n  
\$s2 ↔ b  
\$s3 ↔ a

```
n = 0;
do
{
    a = a + b[n];
    n++;
}while (n < 100);
...
```

O código equivalente em *Assembly* do MIPS:

```
do:      li      $s1, 0          # n = 0;
        # do {
        sll     $t0, $s1, 2      # temp = 4 * n;
        add     $t0, $s2, $t0    # temp = temp + b;
        lw      $t0, 0($t0)      # temp = *temp;
        add     $s3, $s3, $t0    # a = a + temp;
        addi    $s1, $s1, 1      # n = n + 1;
        blt     $s1, 100, do     # } while(n < 100);
        ...
```



Desta feita,  
o teste condicional  
é efectuado “no  
fim do ciclo

**Resumindo:**

- As estruturas do tipo **ciclo** incluem, geralmente, uma ou mais instruções de inicialização de variáveis, executadas antes e fora do mesmo
- No caso do **for()** e do **while()** o teste condicional é executado no início do ciclo
- No caso do **do...while()** o teste condicional é efectuado no fim do ciclo
- Na tradução de um **for()** para *Assembly*, o terceiro campo é incluído no fim do corpo do ciclo.