

## 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()”
- Tradução das estruturas de controlo de fluxo de execução para *Assembly* do MIPS

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

# 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 *a 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 / *jumps*)  
**condicionais**

## Instruções de controlo de fluxo de execução – BEQ

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

- Se os conteúdos dos registos **Rsrc1** e **Rsrc2** forem iguais é realizado um salto, i.e., a execução continua na **instrução situada no endereço representado por "Label"** (*branch taken*)
- A execução continua na **instrução seguinte** se os conteúdos dos 2 registos forem diferentes (*branch not taken*)
- O endereço para onde o salto é efetuado (no caso de a condição ser verdadeira) designa-se por **endereço-alvo** da instrução de *branch* (*branch target address*)

## Instruções de controlo de fluxo de execução – BNE

**bne** **Rsrc1**, **Rsrc2**, **Label** # branch if not equal

- Se os conteúdos dos registos **Rsrc1** e **Rsrc2** forem diferentes é realizado um salto, i.e., a execução continua na **instrução situada no endereço representado por "Label"** (*branch taken*)
- A execução continua na instrução seguinte se os conteúdos dos 2 registos forem iguais (*branch not taken*)
- Exemplo:

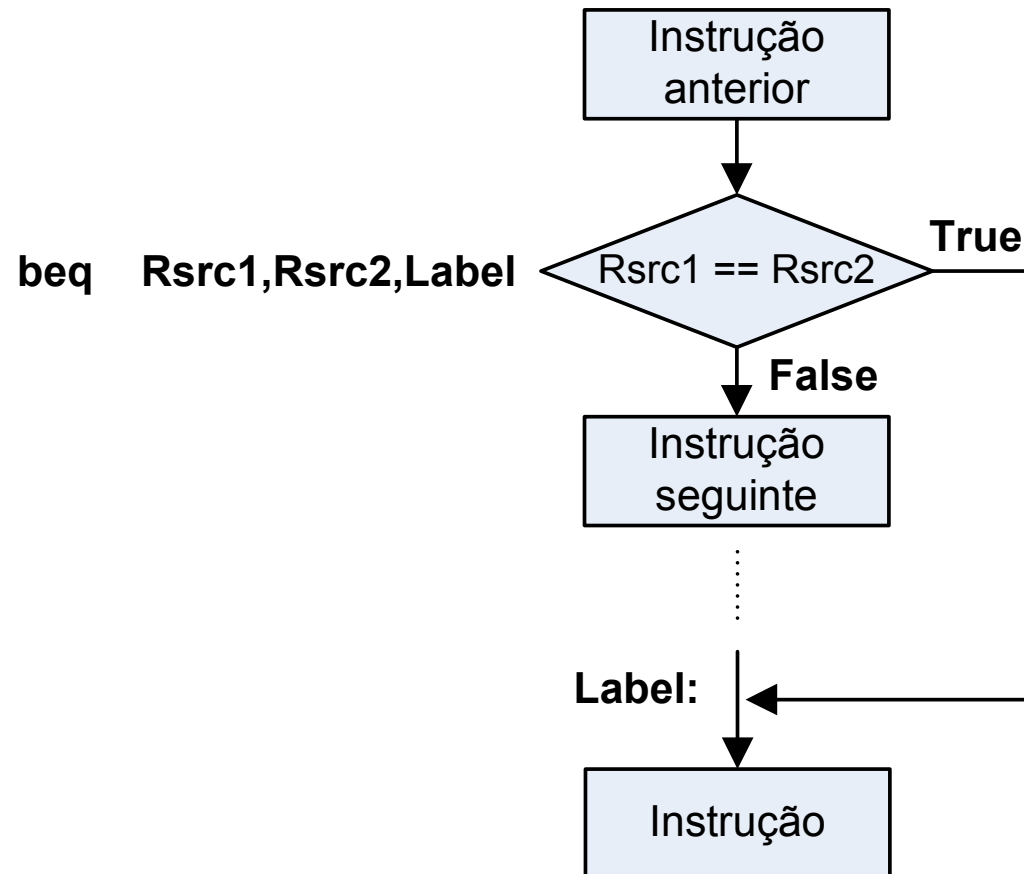
```
    bne $1, $2, L1      # se branch taken (i.e. $1 != $2)
                        # a próxima instrução está em L1
    add $3, $5, $5      # se branch not taken esta
                        # instrução não é executada
L1: add $2, $3, $4      #
```

## Instruções de *branch* – como funcionam?

- Se a **condição** testada na instrução **for verdadeira** (no caso do "beq" **Rsrc1=Rsrc2**, isto é **Rsrc1 – Rsrc2 = 0**), o valor corrente do PC (**Program Counter**) é substituído pelo endereço a que corresponde "Label" (endereço-alvo)
  - A instrução que é executada de seguida é a que se situa no endereço-alvo
- Se a **condição for falsa**, a sequência de execução não é alterada
  - Neste caso, a instrução que é executada de seguida é a que se situa imediatamente a seguir à instrução de *branch*

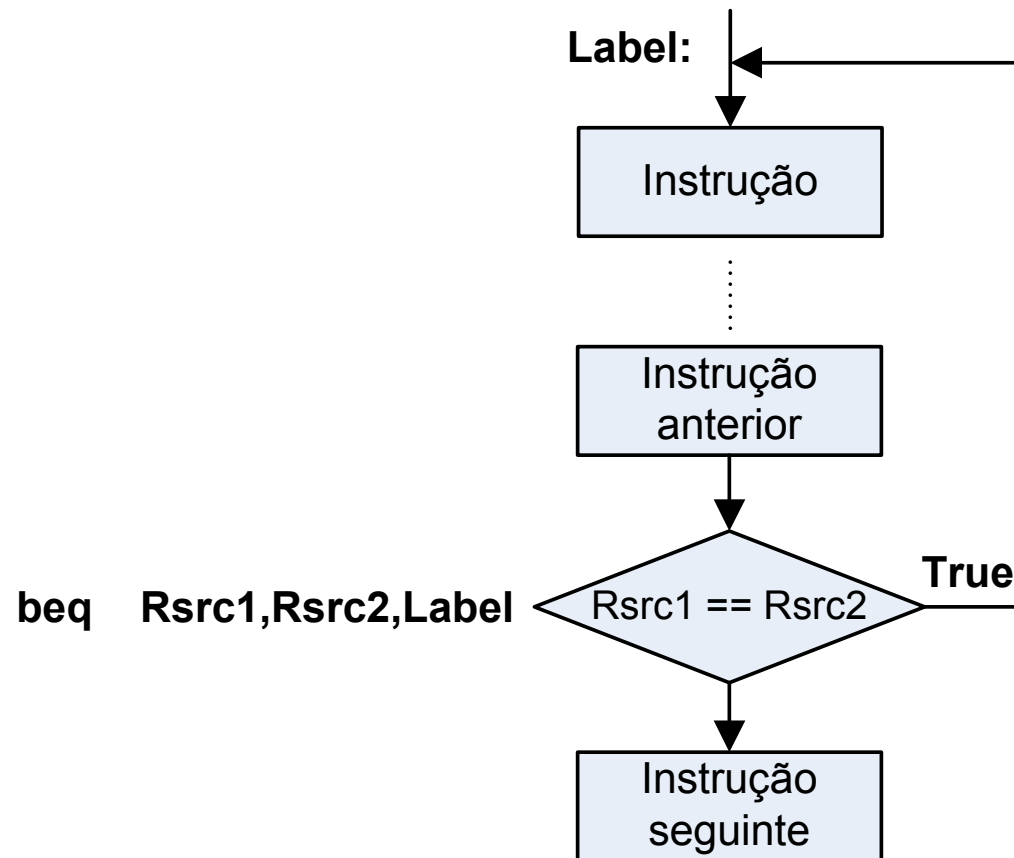
## Instruções de *branch* – como funcionam?

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



## Instruções de *branch* – como funcionam?

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



# Exemplo de tradução C / Assembly

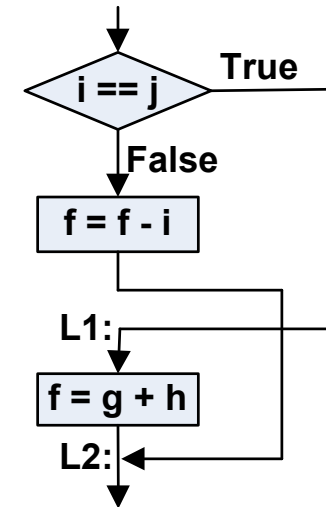
Considere-se o seguinte trecho de código C (escrito de forma inadequada):

```
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 # if(i == j) goto L1;
sub $16,$16,$19 # f = f - i;
j L2 # goto L2;
L1: add $16,$17,$18 # f = g + h;
L2: ...
```

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

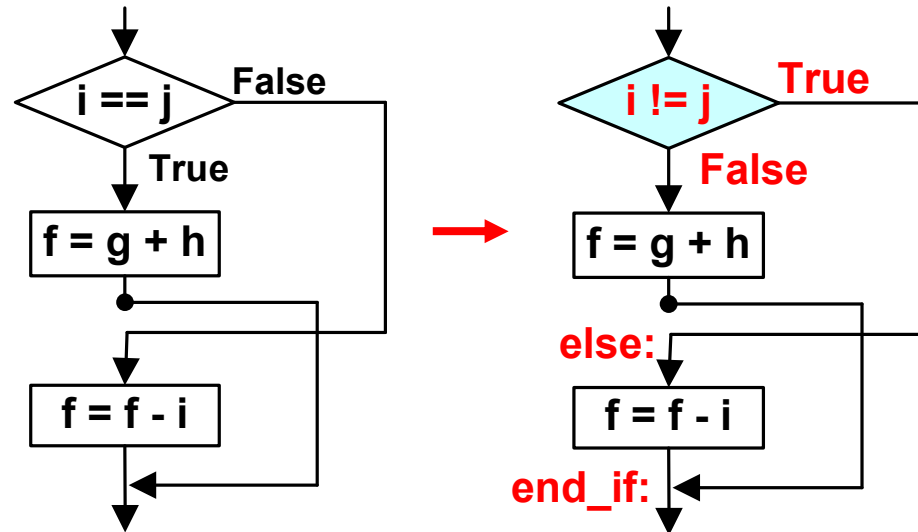




## Exemplo de tradução C / Assembly

O exemplo anterior escrito de forma correta:

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



A que corresponde o código *Assembly*:

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

## Outras instruções de *branch* do MIPS

- O ISA do MIPS suporta ainda um conjunto de instruções que **comparam diretamente 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
- Nestas instruções **o registo \$0 está implícito** como o segundo registo a comparar
- Exemplos:
  - **blez** \$1,L2      # if \$1 ≤ 0 then goto L2
  - **bgtz** \$2,L3      # if \$2 > 0 then goto L3

## Outras instruções de *branch* do MIPS

- Podem ainda ser utilizadas, nos programas *Assembly*, instruções de salto não diretamente suportadas pelo MIPS (**instruções virtuais**), mas que são **decompostas pelo assembler em instruções nativas**, nomeadamente:

- **blt**      **Rsrc1**, **Rsrc2**, **Label**      **# Branch if Rsrc1 < Rsrc2**
- **ble**      **Rsrc1**, **Rsrc2**, **Label**      **# Branch if Rsrc ≤ Rsrc2**
- **bgt**      **Rsrc1**, **Rsrc2**, **Label**      **# Branch if Rsrc > Rsrc2**
- **bge**      **Rsrc1**, **Rsrc2**, **Label**      **# Branch if Rsrc ≥ Rsrc2**

- Nestas instruções **Rsrc2** pode ser substituído por uma **constante**.

Como são decompostas estas instruções?

- Exemplos:

- **blt** \$1,\$2,L2      **# if \$1 < \$2 goto L2**
- **bgt** \$1,100,L3      **# if \$1 > 100 goto L3**

# Instrução SLT

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  $\equiv$  "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  $\equiv$  "set if less than"  
                                # set Rdst if Rsrc1 < Imm
```

A utilização das instruções "**bne**", "**beq**", "**slti**" 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  $\neq$  B**), (**A > B**), (**A  $\geq$  B**), (**A < B**), (**A  $\leq$  B**)

## Decomposição das instruções virtuais BGT e BGE

A **instrução virtual "bge"** (*branch if greater or equal than*):

```
bge    $4, $7, exit    # if $4 ≥ $7 goto exit
                        # (i.e. goto exit if !($4 < $7))
```

É decomposta nas **instruções nativas**:

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

De modo similar, a **instrução virtual "bgt"** (*branch if greater than*):

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

É decomposta nas **instruções nativas**:

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

# Estruturas de controlo de fluxo em C

- Exemplos

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

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

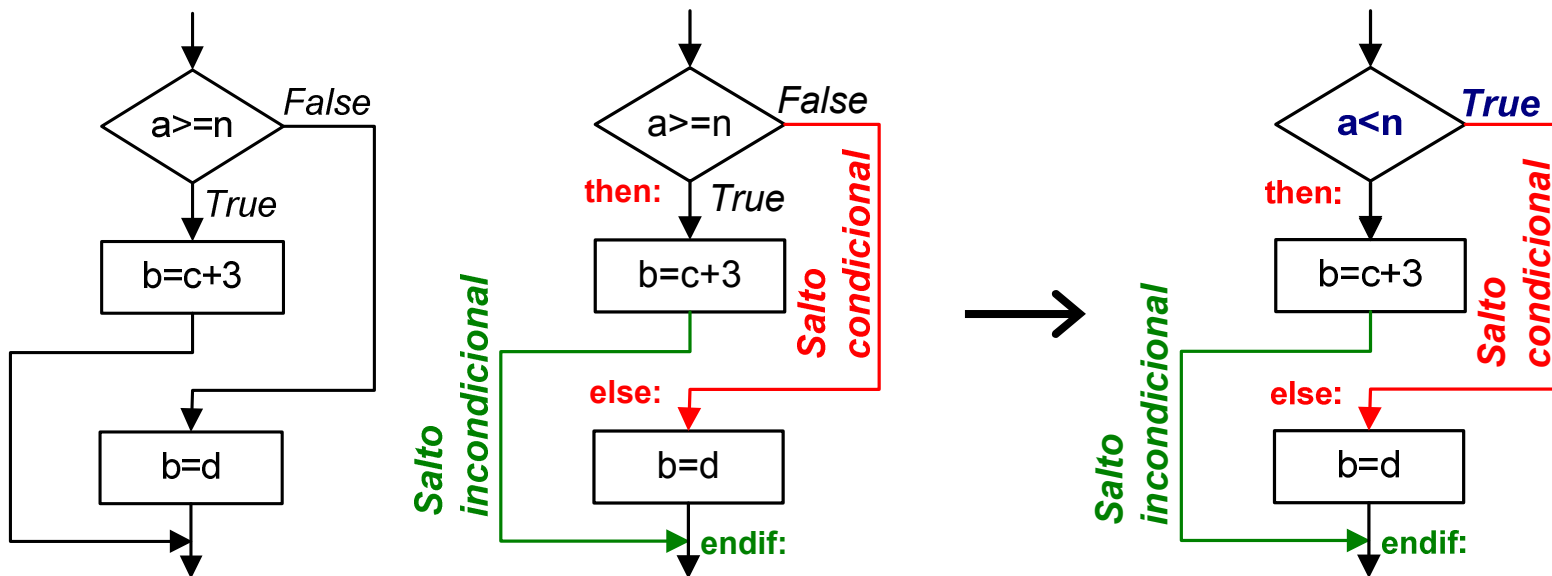
```
n = 0;  
do {  
    a = a + b[n];  
    n++;  
} while (n < 100);  
...
```

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

## Tradução para *Assembly* do MIPS (if()... then... else)

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

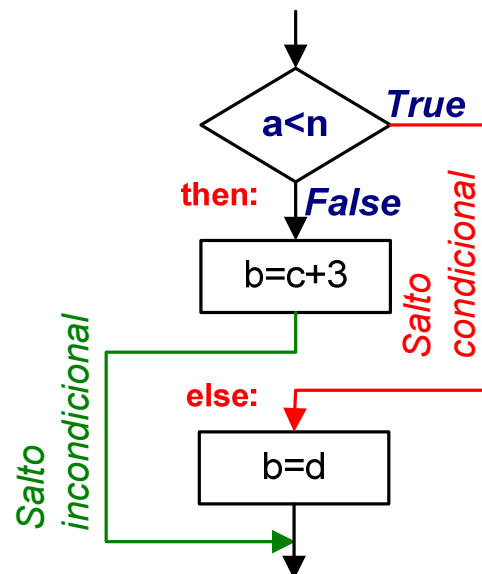
- Transformando o código apresentado no fluxograma equivalente, é possível identificar facilmente a ocorrência de um **salto condicional** e de um **salto incondicional**
- E adaptar o salto condicional para que este se efetue quando a condição for verdadeira (tal como nos *branches*).



## Tradução para *Assembly* do MIPS – *if()... then... else*

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

```
a > $t0  
n > $t1  
c > $t2  
b > $t3  
d > $t4
```



Supondo que as variáveis residem nos registos \$t0 a \$t4, a tradução para *Assembly* fica:

```
blt    $t0,$t1,else # if (a >= n) {  
addi   $t3,$t2,3    #   b = c + 3;  
j      endif        # }  
else:                                     # else {  
move   $t3,$t4      #   b = d;  
endif: ...          # }
```



## Tradução para *Assembly* do MIPS - ciclos *for()* e *while()*

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

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

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**

Os 3 campos do ciclo "**for**" são opcionais. Exemplo:

```
for( ; ; i++) {  
    ...  
}
```

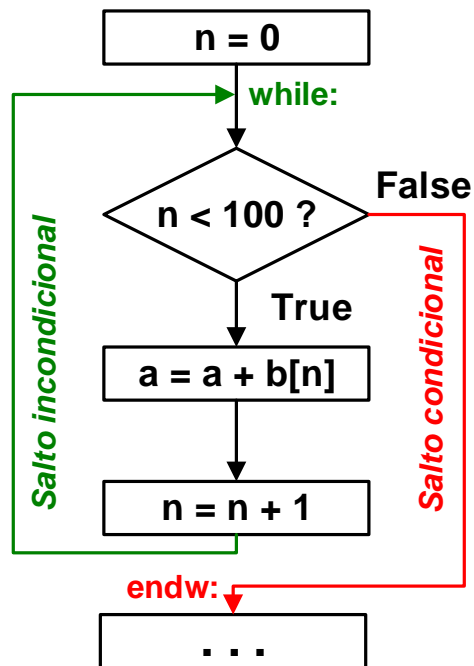
Qual o resultado deste código?

## Tradução para *Assembly* do MIPS - ciclos *for()* e *while()*

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



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



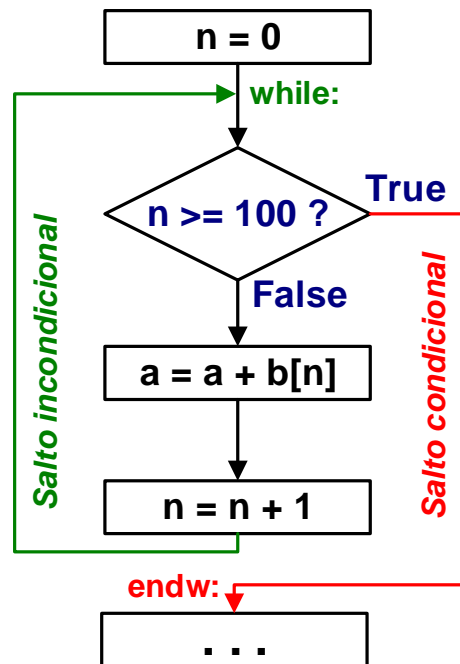
- É possível identificar a ocorrência de um **salto condicional** e de um **salto incondicional**
- O salto condicional necessita de ser modificado de forma a ser efetuado quando a condição for verdadeira
- Para isso usa-se o **complemento lógico** da condição presente no código original (para o exemplo, "<" passa a ser ">=")

## Tradução para *Assembly* do MIPS - ciclos *for()* e *while()*

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



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



A tradução da estrutura de controlo, com "n" a residir em \$t1, fica:

```
ori    $t1,$0,0      #n=0  
while:bge $t1,100,endw #while(n<100)  
      .              #  
      .              # ...  
      .              # ...  
      addi $t1,$t1,1  # n++;  
      j     while     #}  
endw:
```

## Tradução para *Assembly* do MIPS - ciclos *for()* e *while()*

Complemento lógico ( "<" passa a ">=")

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

Teste condicional feito no início do ciclo

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

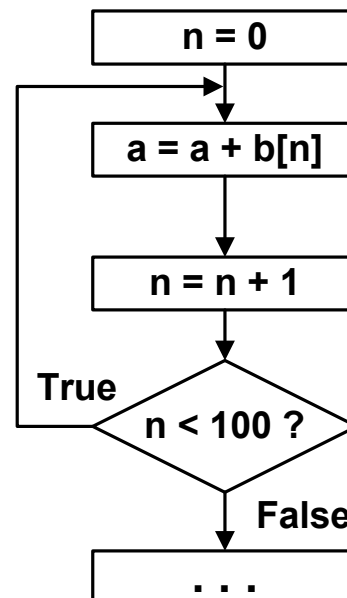
A tradução para *Assembly* MIPS (assumindo  $n > \$t1$ ,  $a > \$t2$ ,  $b > \$t3$ ) fica:

```
ori    $t1,$0,0           # n = 0;  
while: bge    $t1,100,endw  # while(n < 100) {  
    sll    $t0,$t1,2        # temp = 4 * n;  
    add    $t0,$t3,$t0      # temp = temp + b;  
    lw     $t0,0($t0)       # temp = *temp;  
    add    $t2,$t2,$t0      # a = a + temp;  
    addi   $t1,$t1,1        # n++;  
    j      while           # }  
endw:  ...
```

## Tradução para *Assembly* do MIPS - ciclo **do ... while()**

- Ao contrário do **for()** e do **while()**, o corpo do ciclo **do...while()** é executado incondicionalmente pelo menos uma vez!
- O teste da condição é efetuado no fim do ciclo

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



```
ori $t1,$0,0  
do: ...  
...  
...  
blt $t1,100,do
```

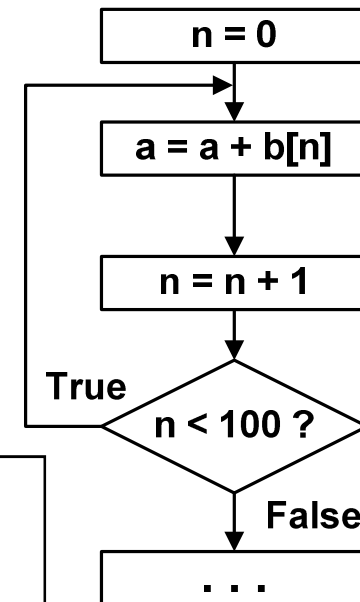
## Tradução para *Assembly* do MIPS - ciclo *do ... while()*

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

A tradução completa para *Assembly* do MIPS fica (com  $n > \$t1$ ,  $a > \$t2$ ,  $b > \$t3$ ):

```
ori    $t1,$0,0      # n = 0;  
do:    # do {  
    sll  $t0,$t1,2     # temp = 4 * n;  
    add  $t0,$t3,$t0    # temp = temp + b;  
    lw   $t0,0($t0)     # temp = *temp;  
    add  $t2,$t2,$t0    # a = a + temp;  
    addi $t1,$t1,1     # n = n + 1;  
    blt  $t1,100,do     # } while(n < 100);
```

o teste condicional efetuado no fim do ciclo



## Conclusão

- 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 efetuado no fim do ciclo
- Na tradução de um **for()** para *Assembly*, o terceiro campo é codificado no fim do corpo do ciclo.