Universidade Presbiteriana Mackenzie

Laboratório de Organização de Computadores
LAB 05
Desvios Condicionais

Prof. Jean Marcos Laine
Prof. Wilian França Costa

Faculdade de Computação e Informática

Até agora trabalhamos as instruções:

- ADD (soma)
- SUB (subtração)
- LW (transferência da memória para um registrador)
- SW (transferência de um registrador para a memória)

Desvios

- Todo programa executa instruções fora da seqüência.
- Tipos de desvios (branches):
  - Conditional branches:
    - branch if equal (`beq`)
    - branch if not equal (`bne`)
  - Unconditional branches:
    - jump (`j`)
    - jump register (`jr`)
    - jump and link (`jal`)

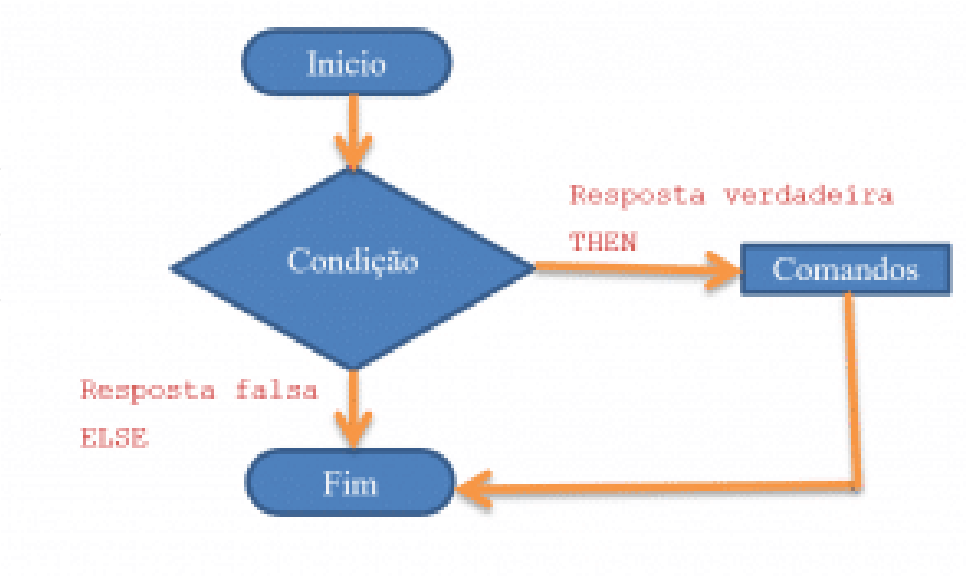
A instrução BEQ

- BEQ significa Branch On Equal, em português, Desvie se for igual. Essa instrução força um desvio para o comando com o LABEL (nome de desvio) se o valor no registrador 1 for igual ao valor no registrador 2, portanto, é uma instrução de comparação.

BEQ registrador1, registrador2, endereço de desvio

opcode	rs	rt	endereço
6 bits	5 bits	5 bits	16 bits

RS e RT são os registradores que serão comparados e o endereço de 16 bits é o desvio. A Figura ilustra o funcionamento da instrução BEQ.



- Basicamente, se r1 = r2 então, desvia para o endereço que está sendo apontado pelo LABEL e execute as instruções que ali estão. Caso contrário, o programa continuará executando as instruções seguintes. Vamos ver um exemplo:

```
if ( x == y ) go to L2;
 a = b + c;
L2 : a = b - c;
```

Considere x = $s0, y = $s1, a = $s2, b = $s3, c = $s4.
Começamos a conversão pela primeira linha if (x == y) go to L2

```
if ( x == y ) go to L2;
     a = b + c;
L2 : a = b - c;
```

Considere x = $s0, y = $s1, a = $s2, b = $s3, c = $s4.
Começamos a conversão pela primeira linha if (x == y) go to L2

BEQ $s0, $s1, L2 #desvia para L2 se x = y

Observe que o endereço fica ao fim da instrução. Devemos continuar a conversão da instrução seguindo a sequência de comandos original. A segunda linha então deve ser convertida a = b + c. Assim:

ADD $s2, $s3, $s4

Essa instrução será executada **se x não for igual a y**. Por fim, a última linha, L2 : a = b – c:

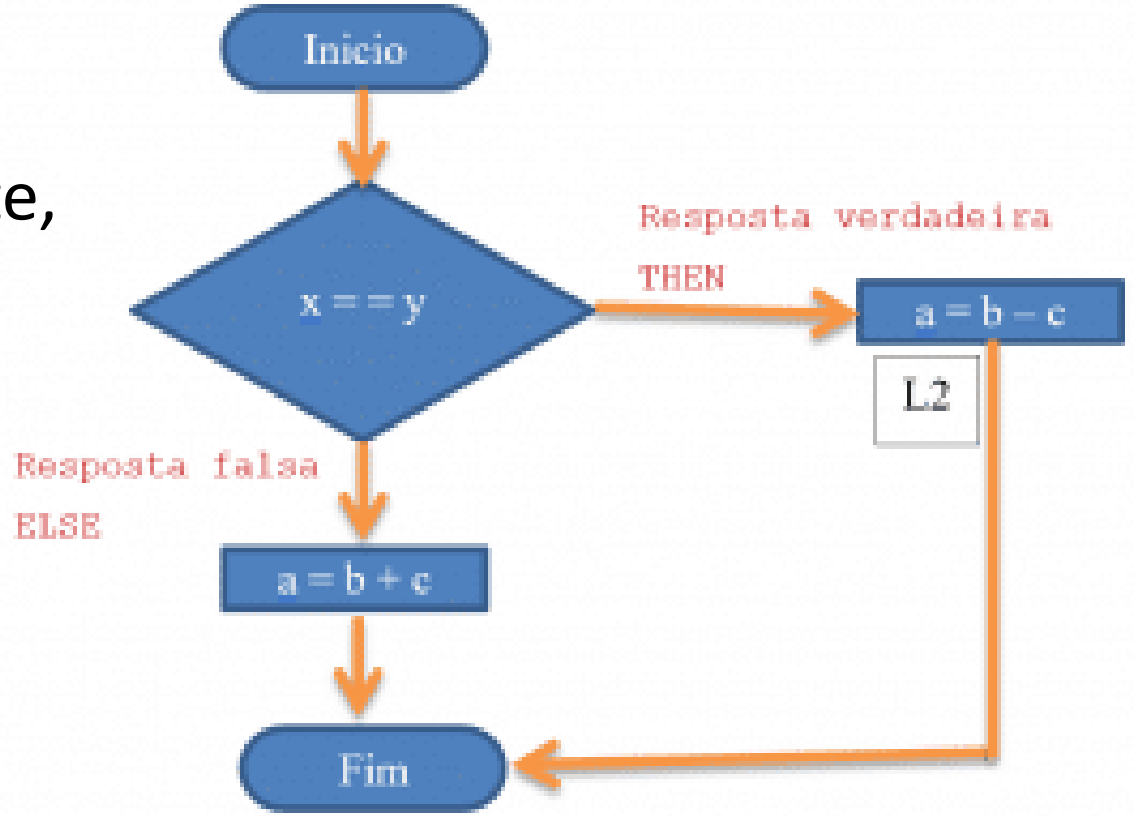
L2 : SUB $s2, $s3, $s4

- Portanto, se x = y, vai para L2,

que executará a = b - c e,

se x <> y executa a instrução seguinte,

a = b + c.



Beq: exemplo

```
# MIPS assembly
 addi  $s0, $0, 4          # $s0 = 0 + 4 = 4
 addi  $s1, $0, 1          # $s1 = 0 + 1 = 1
 sll   $s1, $s1, 2         # $s1 = 1 << 2 = 4
 beq   $s0, $s1, target    # branch is taken
 addi  $s1, $s1, 1         # not executed
 sub   $s1, $s1, $s0       # not executed

target:                    # label
 add   $s1, $s1, $s0       # $s1 = 4 + 4 = 8
```

If / Else Statement

High-level code

```
if (i == j)
  f = g + h;
else
  f = f – i;
```

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
      bne $s3, $s4, L1
      add $s0, $s1, $s2
      j   done
L1:   sub $s0, $s0, $s3
done:
```

While Loops

High-level code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

MIPS assembly code

```
# $s0 = pow, $s1 = x

        addi $s0, $0, 1
        add  $s1, $0, $0
        addi $t0, $0, 128
while:  beq  $s0, $t0, done
        sll  $s0, $s0, 1
        addi $s1, $s1, 1
        j    while
done:
```

For Loops

A forma geral de um for loop é:

```
for (inicialização; condição; loop)
   corpo do loop
```

- `inicialização:` executado antes do loop
- `condição:` testada no inicio de cada iteração
- `loop:` executa no fim de cada iteração
- `Corpodo loop:` executado para cada vez que a condição é satisfeita

For Loops

High-level code

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        add  $s0, $0, $0
        addi $t0, $0, 10
for:    beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 1
        j    for
done:
```

A Instrução SLT

- SLT significa **Set on less Than**, ao pé da letra seria algo como comparar menor que, então essa instrução será muito utilizada em comparações entre registradores, para identificar quem tem o maior ou menor valor.
- A função desta instrução é comparar dois valores de dois registradores diferentes e atribuir o valor 1 a um terceiro registrador se o valor do primeiro registrador for menor que o valor do segundo registrador. Caso contrário, atribuir zero.
- A sintaxe é: **registrador_temporário, registrador1, registrador2**

O formato da instrução é:

OpCode	RS	RT	RD	SHAMT	FUNCT
Código da Operação	Registrador Temporário	Registrador a ser comparado 2	Registrador a ser comparado 1	não usado	código da operação aritmética
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Vamos supor a seguinte instrução MIPS:

SLT $t0, $s1, $s2

Isso é o mesmo que:

$st0 = $s1 < $s2

For Loops: Usando `slt`

High-level code

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        addi $s0, $0, 1
        addi $t0, $0, 101
loop:   slt  $t1, $s0,$t0
        beq  $t1, $0, done
        add  $s1, $s1, $s0
        sll  $s0, $s0, 1
        j    loop
done:
```

$t1 = 1 if i < 101.

Exercícios

1) Faça a conversão da sequencia de instruções abaixo

```
if ( x == y) go to L2
a[1] = b - c;
b = a[2] + c;
c = b + c[3];
L2: a[4] = a[6] + a[5]
```

considere: a = $s0, b = $s1, c = $s2, x = $s3, y = $s4

- 2) Implementar em assembly uma versão para cada uma das estruturas condicionais apresentadas nesse material.