

Conjunto de Instruções do Processador MIPS

Prof. Gustavo Girão

MIPS

- MIPS (1981) – (*Microprocessor without Interlocked Pipeline Stages*)
- Desenvolvido pela MIPS Technologies (antiga MIPS Computer Systems, Inc.)
- Arquitetura RISC
- Atualmente, projetado e licenciado pela *Tallwood MIPS Inc*

Motivação

- Semelhança com outras linguagens de máquina
 - Processadores construídos a partir das mesmas tecnologias de hw
 - Algumas operações básicas são fundamentais a todos processadores
- Utilizada em sistemas embarcados como vídeo games (*Sony PlayStation 2, PlayStation Portable, Nintendo 64*)
- Outras empresas:
 - *Digital Equipment Corporation, NEC, Pyramid Technology, Siemens Nixdorf, Tandem Computers*
- **Do meio para o final dos anos 90, foi estimado que 1 em cada três processadores RISC fabricados eram implementações MIPS**

Plano de aula

- Serão apresentadas as instruções em **assembly MIPS** que podem ser utilizadas para a construção de programas
- Instruções para
 - Utilizar variáveis
 - Realizar operações
 - Criar rotinas de laços e condicionais

MIPS via exemplos

INTRODUÇÃO ÀS INSTRUÇÕES ASSEMBLY

ARITMÉTICAS/ LÓGICAS
CARREGAMENTO
DESVIOS
LOOPS

MIPS via exemplos

INTRODUÇÃO ÀS INSTRUÇÕES ASSEMBLY

ARITMÉTICAS/ LÓGICAS

CARREGAMENTO

DESVIOS

LOOPS

Instruções MIPS

Aritmética

- Soma
 - `add $t0, $t1, $t2` $\# t0 = t1 + t2$
 - `addi $t2, $t3, 50` $\# t2 = t3 + 50$
 - `addi $t2, $t3, -30` $\# t2 = t3 - 30$
 - Instrução imediata evita um carregamento de operando!
- Subtração
 - `sub $t0, $t1, $t2` $\# t0 = t1 - t2$
 - `subi` – não existe
 - Por que não existe `subi`?

Instruções MIPS

Aritmética

- Constante Zero
- Registrador 0 (\$zero) é uma constante 0
- \$zero não pode ser reescrito
- Útil para várias operações, por exemplo, mover entre dois registradores
 - `add $t1, $t2, $zero` $\#t1 = t2$

Instruções MIPS

Operações Lógicas

- Shift Left Logical

- `sll $t2, $s0, 4`

`$t2 = $s0 << 4`

- Shift Right Logical

- `srl $t2, $s0, 4`

`$t2 = $s0 >> 4`

- AND bit a bit

- `and $t0, $t1, $t2`

`$t0 = $t1 & $t2`

- OR bit a bit

- `or $t0, $t1, $t2`

`$t0 = $t1 | $t2`

- O que cada operação faz?

Instruções MIPS

Operações Lógicas

- Outros
 - andi
 - ori
 - nor

MIPS via exemplos

INTRODUÇÃO ÀS INSTRUÇÕES ASSEMBLY

ARITMÉTICAS/ LÓGICAS
CARREGAMENTO
DESVIOS
LOOPS

Instruções MIPS

Carregamento

- `addi $t1, $zero, 25` $\# t1 = 25$
- `lui $t1, 25` $\# t1 = "25"_{10} * 2^{16}$
 - ✧ *Load upper immediate* : carrega o imediato nos bits mais significativos (*upper*) – demais bits são zerados
- De acordo com o comentário, o que o **lui** faz?

Instruções MIPS

Carregamento

- Ler e Escrever na Memória
 - Ler (load) – Carrega um valor que está armazenado em uma posição de memória em um registrador

`lw $t1, offset($t0)`

\$t1 recebe valor de memória armazenado na posição
[\$t0+offset]

- lw – word (palavra)
- lb – byte

Instruções MIPS

Carregamento

- Ler e Escrever na Memória
 - Escrever (store) – Armazena um valor que está em um registrador em uma posição na memória

sw \$t1, **offset**(\$t0)

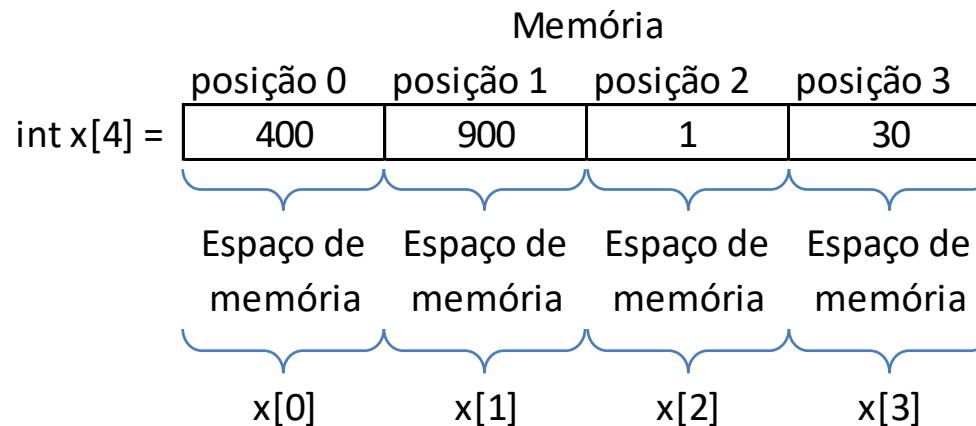
a posição de memória de endereço [\$t0+offset]
recebe o valor que está no registrador \$t1

- sw – word (palavra)
- sb – byte
- **ATENÇÃO:** apesar de terem funções opostas, lw e sw utilizam operandos na mesma ordem!

Instruções MIPS

Carregamento de vetores

- Vetor / Array (Conceito)
 - Servem para guardar vários valores do mesmo tipo de forma uniforme na memória

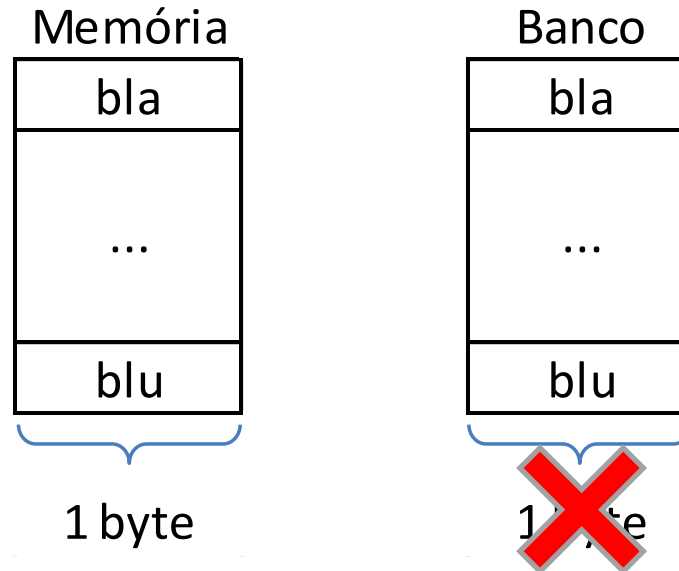


- $\text{int } a = x[0] + x[3].$ Quanto vale a ?
- $x[1] = x[2] + x[3].$ Quanto vale $x[1]$?
- $\text{int } b = x[1] + x[2].$ Quanto vale b ?

Instruções MIPS

Carregamento de vetores

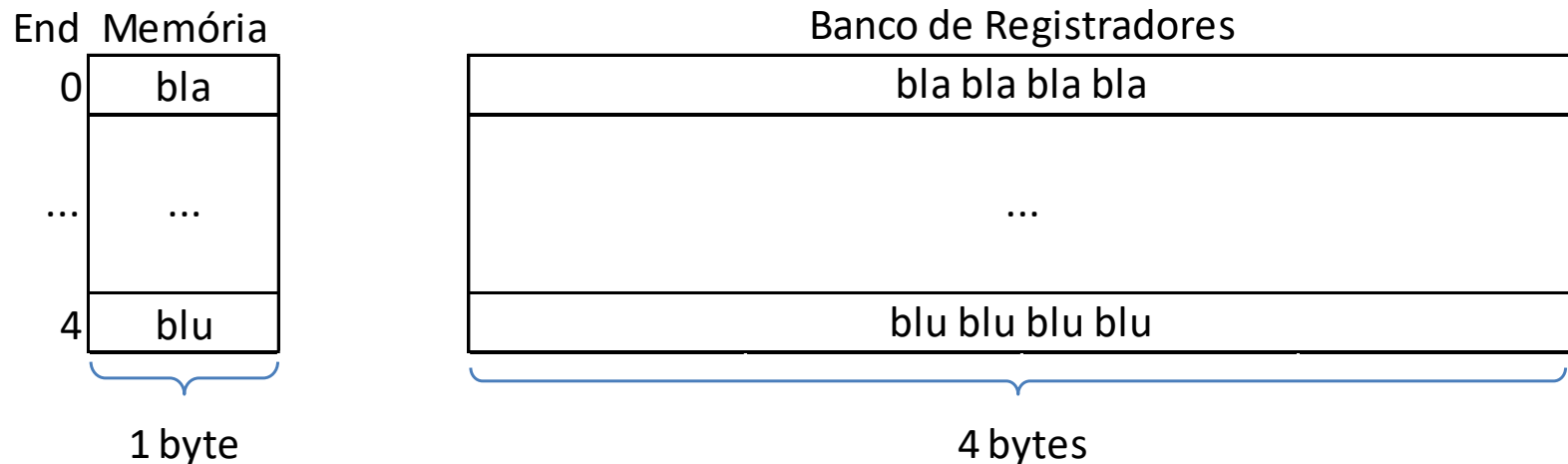
- Memória e Registrador
 - Podemos imaginar a memória e o banco de registradores como um vetor.
 - ✧ No MIPS que vamos estudar, a palavra da memória é de 1 byte (8 bits) e a palavra do processador é de 4 bytes (32 bits)



Instruções MIPS

Carregamento de vetores

- Memória e Registrador **no MIPS**

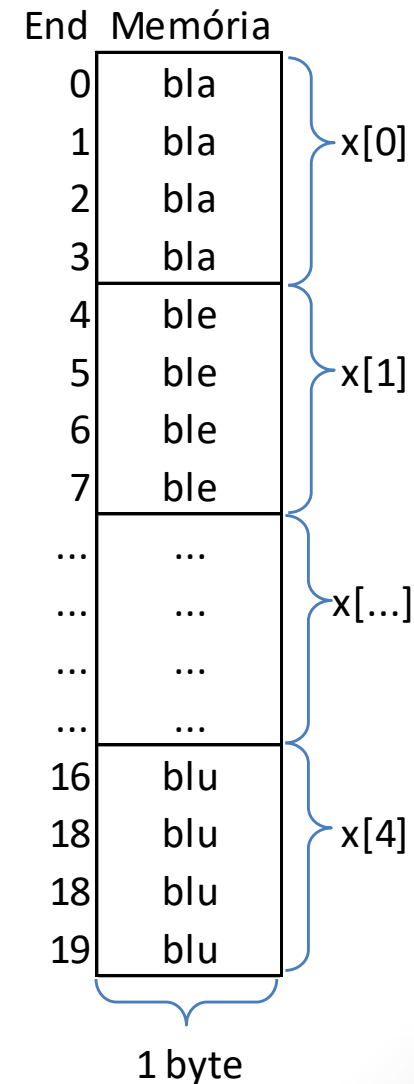


- **(1)** Quantas palavras um dado do banco ocuparia na memória?
- **(2)** Como seria um vetor de 5 posições na memória, cada posição ocupando 4 bytes?

Instruções MIPS

Carregamento de vetores

- Memória e Registrador no MIPS
- **RESPOSTA**
- **(2)** Como seria um vetor de 5 posições na memória, cada posição ocupando 4 bytes?



Instruções MIPS

Carregamento de vetores

- Seja $\$t0$ = endereço inicial do vetor na memória (**$\$t0=0$**)
- **Carregamento de Vetor**
 - $\text{lw } \$t1, \mathbf{0}(\$t0)$ # $\$t1$ recebe $\$t0[0]$
 - ✧ Carrega 4 bytes (32 bits) a partir da posição $\$t0+0$
 - $\text{lw } \$t1, \mathbf{4}(\$t0)$ # $\$t1 = \$t0[1]$
 - ✧ Carrega 4 bytes (32 bits) a partir da posição $\$t0+4$
 - $\text{lw } \$t1, \mathbf{20}(\$t0)$ # $\$t1 = \$t0[5]$
 - ✧ Carrega 4 bytes (32 bits) a partir da posição $\$t0+20$
 - $\text{sw } \$t1, \mathbf{400}(\$t0)$ # $\$t0[100] = \$t1$
 - ✧ Armazena 4 bytes (32 bits) a partir da posição $\$t0+400$
 - Generalizando: Qual a posição do vetor temos em $\mathbf{A}(\$t0)$?
 - ✧ **BASE + Deslocamento!**

MIPS via exemplos

INTRODUÇÃO ÀS INSTRUÇÕES ASSEMBLY

ARITMÉTICAS/ LÓGICAS
CARREGAMENTO
DESVIOS
LOOPS

Instruções MIPS

Desvio incondicional

- *jump to Label*

- j **LABEL**

O que é LABEL?

- Ex:

...

operação 0

j **PARA_AQUI**

operação 1

operação 2

PARA_AQUI: operação 3

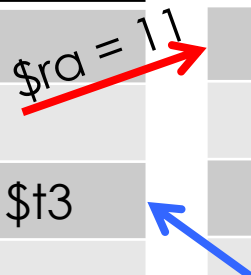
operação 1 e operação 2 não serão executadas, pois serão puladas

Instruções MIPS

Desvio incondicional

- *Jump and link*
 - jal **EndereçoProcedimento**
 - Pula para a instrução em **EndereçoProcedimento** e grava o endereço da próxima instrução em **\$ra**
- *Jump register*
 - jr **\$ra**
 - Pula de volta para seguir o fluxo de instruções anterior

Endereço	Instrução	Endereço	Instrução
...	...	35	Aqui: add \$t1, \$t2, \$t3
10	jal Aqui	36	add \$t1, \$t1, \$t1
11	add \$t1, \$t2, \$t3	37	sub \$t1, \$t1, \$t4
...	...	38	jr \$ra



Instruções MIPS

Desvio incondicional

Endereço	Instrução
...	...
10	jal Aqui
11	add \$t1, \$s4, \$t3
...	...
...	...
...	...
...	...
78	jal Aqui
79	
...	...

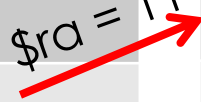
Endereço	Instrução
35	Aqui: add \$t1, \$t2, \$t3
36	add \$t1, \$t1, \$t1
37	sub \$s4, \$t1, \$t4
38	jr \$ra

Instruções MIPS

Desvio incondicional

Endereço	Instrução
...	...
10	jal Aqui
11	add \$t1, \$s4, \$t3
...	...
...	...
...	...
...	...
78	jal Aqui
79	
...	...

$\$ra = 11$




Endereço	Instrução
35	Aqui: add \$t1, \$t2, \$t3
36	add \$t1, \$t1, \$t1
37	sub \$s4, \$t1, \$t4
38	jr \$ra

Instruções MIPS

Desvio incondicional

Endereço	Instrução
...	...
10	jal Aqui
11	add \$t1, \$s4, \$t3
...	...
...	...
...	...
...	...
78	jal Aqui
79	
...	...

\$ra = 11



Endereço	Instrução
35	Aqui: add \$t1, \$t2, \$t3
36	add \$t1, \$t1, \$t1
37	sub \$s4, \$t1, \$t4
38	jr \$ra

Instruções MIPS

Desvio incondicional

Endereço	Instrução	Endereço	Instrução
...	...	35	Aqui: add \$t1, \$t2, \$t3
10	jal Aqui	36	add \$t1, \$t1, \$t1
11	add \$t1, \$s4, \$t3	37	sub \$s4, \$t1, \$t4
...	...	38	jr \$ra
...	...		
...	...		
...	...		
78	jal Aqui		
79			
...	...		

\$ra = 11

\$ra = 79

Instruções MIPS

Desvio incondicional

Endereço	Instrução	Endereço	Instrução
...	...	35	Aqui: add \$t1, \$t2, \$t3
10	jal Aqui	36	add \$t1, \$t1, \$t1
11	add \$t1, \$s4, \$t3	37	sub \$s4, \$t1, \$t4
...	...	38	jr \$ra
...	...		
...	...		
...	...		
78	jal Aqui		
79			
...	...		

Instruções MIPS

Tomada de Decisão

- *Branch if equal*
 - `beq $s3, $s4, LABEL` # Vá para **LABEL** se “\$s3 == \$s4”
 - `bne $s3, $s4, LABEL1` # Vá para **LABEL1** se “\$s3 != \$s4”

- **Exemplo**

```
beq $s1, $s2, L1
j L2
L1:
    addi $s1, $s1, 1
    j EXIT
L2:
    addi $s1, $s1, 2
EXIT: ...
```

- Qual seria o algoritmo do exemplo?

-

Instruções MIPS

Tomada de Decisão

- *Branch if equal*
 - `beq $s3, $s4, LABEL` # Vá para **LABEL** se “\$s3 == \$s4”
 - `bne $s3, $s4, LABEL1` # Vá para **LABEL1** se “\$s3 != \$s4”

- **Exemplo**

```
bne $s1, $s2, Else
addi $s1, $s1, 1
j exit
Else:
    addi $s1, $s1, 2
EXIT: ...
```

- Qual seria o algoritmo do exemplo?
- É o mesmo que o algoritmo anterior?

Instruções MIPS

Tomada de Decisão

- *Branch if equal*

- `beq $s3, $s4, LABEL` # Vá para **LABEL** se “\$s3 == \$s4”
- `bne $s3, $s4, LABEL1` # Vá para **LABEL1** se “\$s3 != \$s4”

- **Exemplo**

```
bne $s1, $s2, Else
addi $s1, $s1, 1
j exit
Else:
    addi $s1, $s1, 2
EXIT: ...
```

SIM, e está mais compacto

- Qual seria o algoritmo do exemplo?
- É o mesmo que o algoritmo anterior?

Instruções MIPS

Tomada de Decisão

- DICA:
 - De modo geral, o código será mais eficiente se testarmos a condição oposta ao desvio no lugar do código que realiza a parte “then” subsequente do “if”

OU SEJA

- De modo geral, o código será mais eficiente se testarmos primeiro o “else”.

Instruções MIPS

Tomada de Decisão

- *Branch if equal*
 - `beq $s3, $s4, LABEL` # Vá para **LABEL** se “\$s3 == \$s4”

- **Exemplo**

```
beq $s1, $s2, L1           #branch if (k == 1)
subi $s2, $s2, 1           #l--
L1: addi $s1, $s1, 1        #k++
```

- **k** está em **\$s1** e **l** está em **\$s2**
- Como poderíamos escrever o algoritmo?

Durante a execução, a sequência do código é seguida mesmo com o label

Instruções MIPS

Tomada de Decisão

- *Branch if equal*
 - `beq $s3, $s4, LABEL` # Vá para **LABEL** se “\$s3 == \$s4”

- **Exemplo**

```
beq $s1, $s2, L1      #branch if (k == l)
subi $s2, $s2, 1      #l--
L1: addi $s1, $s1, 1   #k++
```

- **k** está em **\$s1** e **l** está em **\$s2**

```
if ( k != l ) {
    l--
}
k++
```

Instruções MIPS

Teste de Igualdade

- *Set on less than*
 - `slt $t0, $s3, $s4`
 - ✧ `$t0` será “1” se “`$s3 < $s4`”
 - ✧ `$t0` será “0”, cc
- Muito utilizado juntamente com o **beq** na tomada de decisão em desigualdades

- **Exemplo** `slt $s1, $s2, $s3`
`bne $s1, $zero, Else`
`addi $s2, $s2, 1`
`j EXIT`
`Else:`
`addi $s2, $s2, 2`
`EXIT: ...`

Instruções MIPS

Teste de Igualdade

- *Set on less than*
 - `slt $t0, $s3, $s4`
 - ✧ `$t0` será "1" se "`$s3 < $s4`"
 - ✧ `$t0` será "0", cc
- Muito utilizado juntamente com o **beq** na tomada de decisão em desigualdades

- **Exemplo**

```
slt $s1, $s2, $s3
bne $s1, $zero, Else
addi $s2, $s2, 1
j EXIT
Else:
    addi $s2, $s2, 2
EXIT: ...
```

Como seria o
algoritmo?

Instruções MIPS

Teste de Igualdade

- **Exemplo**

```
slt $s1, $s2, $s3
bne $s1, $zero, Else
addi $s2, $s2, 1
j EXIT
Else:
    addi $s2, $s2, 2
EXIT: ...
```

Como seria o
algoritmo?

```
if ( s2 >= s3 ){
    s2 = s2 + 1
}
else{
    s2 = s2 + 2
}
```

Instruções MIPS

Teste de Igualdade

- Precisa existir “set on **more** than”?
- Outros
 - slti (para *imediato*)

MIPS via exemplos

INTRODUÇÃO ÀS INSTRUÇÕES ASSEMBLY

ARITMÉTICAS/ LÓGICAS
CARREGAMENTO
DESVIOS
LOOPS

Instruções MIPS

Loop

- Como poderíamos escrever o seguinte algoritmo?

```
inteiro i = 0
inteiro j = 15
enquanto ( i < 10 ) {
    j = j + 5
    i = i + 1
}
```

Instruções MIPS

Loop

- Como poderíamos escrever o seguinte algoritmo?

```
li $t0, 10      # constante 10
li $t1, 0       # contador do laço i
li $t2, 15      # variável j
loop:
    beq $t1, $t0, end    # se t1 == 10, o código acaba
    addi $t2, $t2, 5     # j = j + 5
    addi $t1, $t1, 1     # i = i + 1
    j loop
end:
...
```


Instruções MIPS

Pseudoinstruções

- Elas são reconhecidos pelo assembler, mas traduzidas em pequeno conjunto de instruções de máquina.

move \$t0, \$t1	se torna	add \$t0, \$zero, \$t1	\$t0 = \$t1
blt \$t0, \$t1, L	se torna	slt \$at, \$t0, \$t1 bne \$at, \$zero, \$L	Jump para L se \$t0 < \$t1

O HELP do simulador Mars 4.4 lista as instruções básicas e pseudoinstruções

Algumas observações

1. O código em linguagem de alto nível não tem a mesma quantidade de instruções do que o código assembly.
2. Muitas vezes, é necessário ter instruções intermediárias em assembly para poder executar a instrução em alto nível

if (s2 < s3)



```
slt $s1, $s2, $s3  
bne $s1, $zero, Else
```

Para treinar

- O que faz trecho de programa abaixo?

```
L1:  
    add $s0, $s0, $t1  
    addi $t0, $t0, 1  
    bne $t2, $t0, L1  
EXIT: ...
```

Nome	Sintaxe	Significado
<u>Add</u>	<u>add</u> \$1,\$2,\$3	\$1 = \$2 + \$3 (<u>signed</u>)
<u>Sub</u>	<u>sub</u> \$1,\$2,\$3	\$1 = \$2 - \$3 (<u>signed</u>)
<u>Add Immediate</u>	<u>addi</u> \$1,\$2,CONST	\$1 = \$2 + CONST (<u>signed</u>)
<u>Set on less than</u>	<u>slt</u> \$1,\$2,\$3	<u>if</u> (\$2 < \$3) \$1 = 1 <u>else</u> \$1 = 0
<u>Branch on not equal</u>	<u>bne</u> \$1,\$2, <u>Label</u>	<u>if</u> (\$1 != \$2) goto <u>Label</u>
<u>Branch on equal</u>	<u>beq</u> \$1,\$2, <u>Label</u>	<u>if</u> (\$1 == \$2) goto <u>Label</u>
<u>Jump</u>	<u>j</u> <u>Label</u>	goto <u>Label</u>

Para treinar

- Implementar em assembly MIPS

- $\$t0 = x$
- $\$t1 = y$

se ($y == 0$) faça
 $x = x + y - 1$;
senão
 $x = x - y + 1$;
fim se

Nome	Sintaxe	Significado
<u>Add</u>	<u>add</u> \$1,\$2,\$3	$\$1 = \$2 + \$3$ (<u>signed</u>)
<u>Sub</u>	<u>sub</u> \$1,\$2,\$3	$\$1 = \$2 - \$3$ (<u>signed</u>)
<u>Add Immediate</u>	<u>addi</u> \$1,\$2,CONST	$\$1 = \$2 + \text{CONST}$ (<u>signed</u>)
<u>Set on less than</u>	<u>slt</u> \$1,\$2,\$3	if ($\$2 < \3) $\$1 = 1$ else $\$1 = 0$
<u>Branch on not equal</u>	<u>bne</u> \$1,\$2, <u>Label</u>	if ($\$1 \neq \2) goto <u>Label</u>
<u>Branch on equal</u>	<u>beq</u> \$1,\$2, <u>Label</u>	if ($\$1 == \2) goto <u>Label</u>
<u>Jump</u>	<u>j</u> <u>Label</u>	goto <u>Label</u>

?

Bibliografia

- PATTERSON, D. A. & HENNESSY, J. L.

Organização e Projeto de Computadores –
A Interface Hardware/Software. 3ª ed. Campus,
CAPÍTULO 2

- **MIPS Assembly Language**

<http://www.inf.uni-konstanz.de/dbis/teaching/ws0304/computing-systems/download/rs-05.pdf>

Introdução Curta ao MIPS

<http://www.di.ubi.pt/~desousa/2011-2012/LFC/mips.pdf>