

Laboratório 3

Assembly MIPS

Prof. Gustavo Girão

Resumo da aula passada

- Exemplos com desvios e laços

Sobre o código

- Instruções estão no HELP
- Códigos para o Syscall estão no HELP
- Tipos de dados (.word, .ascii, etc) estão no HELP
- Registradores estão no simulador

Salto Condicional - BRANCH

- *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, $s2, L1      #salte para L1 se ($s1=$s2)
j L2                  #senão, salte L2
L1:
    addi $s1, $s1, 1   #s1++
    j EXIT              #salte EXIT
L2:
    addi $s1, $s1, 2   #s1+=2
EXIT: ...
```

Salto Condicional - BRANCH

- *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: ...
```

```
if ( s1 == s2 ) {
    s1 = s1 + 1;
}
else {
    s1 = s1 + 2;
}
```

Salto Condicional - P

Note que o jump também é necessário!

- *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: ...
```

```
if ( s1 == s2 ) {  
    s1 = s1 + 1;  
}
```

```
else {  
    s1 = s1 + 2;  
}
```

Salto Condicional - BR

15 min

- Abra o programa lab6.asm e simule a execução.
- Considerando as duas instruções abaixo, responda:
 - beq \$s3, \$s4, **LABEL** # Vá para **LABEL** se “\$s3 == \$s4”
 - bne \$s3, \$s4, **LABEL1** # Vá para **LABEL1** se “\$s3 != \$s4”

1. Qual seria o algoritmo em linguagem C do exemplo lab6.asm?

2. É o mesmo que o algoritmo do slide anterior (#6)? Justifique sua resposta.

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: ...
```


Teste de Igualdade

Perceba que o teste é utilizado para comparações de $>$ e $<$

- *Set on less than*
 - `slt $t0, $s3, $s4`
 - ✧ `$t0` será "1" se "`$s3 < $s4`"
 - ✧ `$t0` será "0", cc
- Muito utilizado juntamente com `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: ...
```

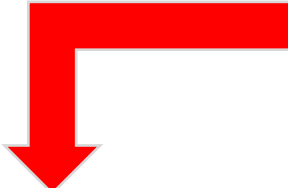
Laços

- Como poderíamos escrever o seguinte algoritmo?

```
int i = 0;  
while ( i < 10 ) {  
    i = i + 1;  
}
```

Laços

- Como poderíamos escrever o seguinte algoritmo?



```
int i = 0;
while ( i < 10 ) {
    i = i + 1;
}
```

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

Trabalhando com vetores

- Ao solicitar a reserva de espaços, o montador garante a reserva em espaços contíguos
 - Características de um vetor
- Ao utilizar um vetor em assembly, pode-se inicializar os elementos individualmente
 - .word 1, 2, 3, 4, 5
- Mas e se o vetor tiver 1000 elementos.....?
 - .word 0:1000
 - ✧ O número após os dois pontos indica quantos espaços de tamanho "word" serão reservados
 - ✧ O numero antes dos dois pontos indica o valor inicializado em cada um deles
- Verifique o programa lab7.asm

Exercício 3. Descreva em Linguagem C:

```
.data
max1: .word 40
min1: .word 20
      .text
      .globl main

main:
    li $v0, 5
    syscall
    move $t0, $v0

    lw $s0, max1
    lw $s2, min1
    li $s1, 0
    li $t1, 1
    slt $t2, $s0, $t0
    beq $t2, $t1, FORA
    slt $t3, $t0, $s2
    beq $t3, $t1, FORA

DENTRO:
    li $s1, 1
    j EXIT

FORA:
    li $s1, 0
    j EXIT

EXIT:
    li $v0, 1
    move $a0, $s1
    syscall

    li $v0, 10
    syscall
```



30 min

Exercicio 4. Implementar em assembly

```
int main() {  
    int k, f, g=5, h=7, i=56, j=9;
```

```
    printf("Digite o valor de k\n");  
    scanf("%d", &k);
```

```
    switch (k) {  
        case 0: f = i + j;    break;  
        case 1: f = g + h;    break;  
        case 2: f = g - h;    break;  
        case 3: f = i - h;    break;  
    }
```

```
    printf("f= %d", f);
```

```
    return 0;  
}
```

PRECISA TER A SAÍDA DE DADOS

PRECISA TER A ENTRADA DE DADOS

REGISTRADORES, MEMÓRIA E IMEDIATO
SÃO ESCOLHA SUA

40 min

PRECISA TER O ENCERRAMENTO DO PROGRAMA

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>