

Laboratório 2

Assembly MIPS

Prof^a Monica Magalhães Pereira

Resumo da aula passada

- Conhecendo o simulador
 - Editor
 - Memória de programa
 - Memória de dados
 - Banco de registradores
- Como usar a memória de dados
- Como fazer Entrada e saída de dados (syscall)
- Simulação da execução

Plano de aula

- Nessa aula de laboratório, você utilizará o simulador que você aprendeu a usar na aula passada para desenvolver programas com saltos e laços em Assembly MIPS
- Para fazer o download do MARS, acessar: <http://courses.missouristate.edu/KenVollmar/MARS/>

Salto Incondicional - JUMP

- *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

Salto Incondicional - JUMP

- Baixe todos os programas da aula de hoje, disponível na turma virtual numa pasta em seu computador
- Abra o programa lab4.asm.
- Simule a execução, utilizando o painel de simulação e responda as questões a seguir:



Salto Incondicional - JUMP

1. Explique o que faz cada instrução do programa
2. Demonstre, em pseudo-código ou em C, uma implementação em em alto nível que faça a mesma coisa que o código assembly.

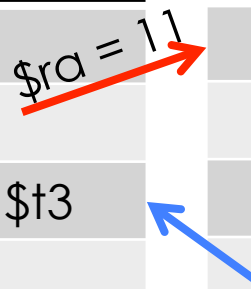


15 min

JUMP com retorno

- *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	
...	...

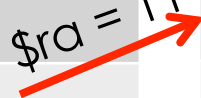
Endereço	Instrução
35	Aqui: add \$t1, \$t2, \$t3
36	add \$t1, \$t1, \$t1
37	sub \$s4, \$t1, \$t4
38	jr \$ra

Instruções MIPS

Desvio incondicional

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

$\$ra = 11$



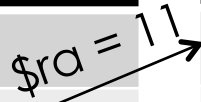
Endereço	Instrução
35	Aqui: add \$t1, \$t2, \$t3
36	add \$t1, \$t1, \$t1
37	sub \$s4, \$t1, \$t4
38	jr \$ra

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			
...	...		

Diagram illustrating unconditional branch instructions (jal) and the return address register (\$ra).

The left table shows memory addresses and instructions. The right table shows the target code block labeled "Aqui".

Annotations:

- A blue arrow points from the **jal Aqui** instruction at address 78 to the **Aqui** label at address 35.
- Handwritten notes indicate the return address register (\$ra) is updated:
 - \$ra = 11 (from the first **jal Aqui** at address 10)
 - \$ra = 79 (from the second **jal Aqui** at address 78)

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			
...	...		

JUMP com retorno

3. O que faz o programa lab5.asm?
4. Explique a diferença desse programa para o lab4.asm.



Salto Condicional - BRANCH

- Muitas vezes, precisamos mudar o fluxo de execução do programa, porém, a mudança só deve ocorrer se uma condição for satisfeita.
- Exemplo:

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

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          if ( s1 == s2 ) {
j  L2                      s1 = s1 + 1;
L1:                          }
    addi $s1, $s1, 1
    j  EXIT                else {
                             s1 = s1 + 2;
L2:                          }
    addi $s1, $s1, 2
EXIT: ...
```

Salto Condicional - BRANCH

- 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”

5. Qual seria o algoritmo do exemplo?

6. É o mesmo que o algoritmo anterior?

15 min

Salto Condicional - BRANCH

- 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”.

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: ...
```

Teste de Igualdade

7. O que faz o programa abaixo?

Reponda em pseudo-código ou em c

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

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

10 min

Teste de Igualdade

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


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:
...
```

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

- O código em linguagem de alto nível não tem a mesma quantidade de instruções do que o código assembly.
- 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
```

ÚLTIMA ATIVIDADE

8. Implemente e simule em assembly MIPS o algoritmo abaixo:

```
int num1, num2, resultado;  
printf("Digite o primeiro numero: \n");  
scanf("%d", &num1);  
printf("Digite o segundo numero: \n");  
scanf("%d", &num2);  
if (num1==0) faça  
    num1 = num1+num2-1;  
else  
    num1 = num1-num2+1;  
printf("O resultado e: %d\n", num1);
```



20 min

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](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](http://www.di.ubi.pt/~desousa/2011-2012/LFC/mips.pdf)**