Instruções do MIPS

Disciplina: Arquitetura de Computadores

Nome	Nº	Uso
$0 ou $zero	0	Registrador com valor constante igual a zero
$at	1	*Assemblador* temporário – reservado ao montador
$v0-$v1	2-3	Registradores que recebem as funções de chamada do sistema
$a0-$a3	4-7	Registradores para passagem de argumentos
$t0-$t7	8-15	Registradores temporários (não preservam os valores)
$s0-$s7	16-23	Registradores que preservam (salvam) os valores
$t8-$t9	24-25	Registradores temporários (não preservam os valores)
$k0-$k1	26-27	Registradores para *kernel* do Sistema Operacional
$gp	28	Registrador para ponteiro global
$sp	29	Registrador apontador para pilha
$fp	30	Registrador para apontador de frame
$ra	31	Registrador para guardar o endereço de retorno
$pc		Registrador especial usado para contar as execuções dos programas (*Program Counter*)
$lo		Registrador especial, guardam resultados da multiplicação e divisão. Não é acessado diretamente, precisa usar o comando mflo
$hi		Registrador especial, guardam resultados da multiplicação e divisão. Não é acessado diretamente, precisa usar o comando mfhi

$v0 e $v1 – São registradores que recebem valores das expressões e resultado de funções

Serviço	$v0	Argumento	Resposta
Imprimir inteiro	1	**$a0** deve conter o número inteiro a imprimir	Manipular o registrador **$v0** com as instruções **1**, **2**, **3** e **4**, correspondem as funções básicas de saída de dados
Imprimir *float*	2	**$f12** deve conter o número *float* a imprimir	
Imprimir *double*	3	**$f12** deve conter o número *double* a imprimir	
Imprimir *String* ou *Char*	4	**$a0** deve conter a *string* a imprimir	
Ler inteiro	5	Manipular o registrador **$v0** com as instruções **5**, **6**, **7** e **8**, correspondem as funções de entrada de dados	**$v0** armazena o número inteiro digitado pelo usuário
Ler *float*	6		**$f0** armazena o número *float* digitado pelo usuário
Ler *double*	7		**$f0** armazena o número *double* digitado pelo usuário
Ler *String* ou *Char*	8		**$a0** armazena a *string* digitada pelo usuário
Sair	10		

Diretiva .data

- É um segmento de código separado para a declaração das constantes e das variáveis

Diretiva de Tipagem	Tipo	Exemplo
.asciiz	Usado para declarar uma constante ou variável do tipo texto (*String*)	“Olá mundo””
.word	Usado para declarar uma constante ou variável do tipo inteiro (*Integer*)	22
.float	Usado para declarar uma constante ou variável do tipo decimal (*Float*)	3.14
.double	Usado para declarar uma constante ou variável do tipo decimal (*Double*)	3.14
.byte	Usado para declarar uma constante ou variável do tipo caractere (*Char*)	‘n’
.space <qtde>	Usado para declarar uma constante ou variável do tipo *String* com espaços em branco	40

Diretivas .globl e .text

- **.globl**
  - Diretiva global
  - Semelhante a função principal main() da linguagem C
  - Função principal do programa
- **.text**
  - Segmento destinado ao código do programa

```
.data                       #Diretiva de dados
.text                       #Diretiva de texto
        li $v0, 10          #Configura o fim do programa
        syscall             #Executa a instrução
```

Neste exemplo, temos uma diretiva segmentando os dados (*.data*) e temos uma diretiva segmentando a área de construção do código (*.text*)

Diretivas .globl e .text

- A diretiva **.globl** pode ser abstraída quando não há necessidade de vários blocos de código

```
.data                           #Diretiva de dados
        msg: .asciiz            "Olá mundo!"
.text                           #Diretiva de texto
.globl bloco1                   #Diretiva global
bloco1:                         #Bloco de código Bloco1
        li $v0, 4               #Impressão de String
        la $a0, msg             #$a0 reecbe msg
        syscall                 #Executa
        li $v0, 10              #Fim do programa
        syscall                 #Executa
```

Este código está dividido em diretiva de dados (*.data*), com um exemplo de criação de uma variável do tipo *.float*.
Na diretiva *.text* encontramos o código, separando a execução por uma função principal.
A configuração da função principal é definida pela diretiva *.globl*.

Registradores $a0 e $a1

- Os registradores de argumentos recebem os dados à serem direcionados para a *saída* do sistema. Por exemplo, se o registrador **$v0** for setado com as opções de serviço número **4** (imprimir uma *string*), nós precisamos carregar o registrador **$a0** com a *string* que queremos que seja exibida na saída de dados.

```
.data                       #Diretiva de dados
        #Área para dados na memória principal
        msg:    .asciiz "Olá Mundo!"
.text                       #Diretiva de texto
        #Área para instruções do programa
        li $v0, 4           #Impressão de String
        la $a0, msg         #Vai indicar o endereço em que está a mensagem
        syscall             #Executa a instrução
```

Registradores $t0 a $t9

- Os registradores temporários são as memórias de trabalho. Nosso programa usará os registradores para executar instruções na ULA e armazenar seu resultado. Qualquer número inteiro poderá ser registrado neles.

```
.data                       #Diretiva de dados
        #Área para dados na memória principal
        idade:   .word    37
.text                       #Diretiva de texto
        #Área para instruções do programa
        la $t0, idade       #$t0 = idade
        li $v0, 10          #Sair do programa
        syscall             #Executa a instrução
```

Registradores $s0 a $s7

- Os registradores *saved* conseguem preservar os valores. Uma sub-rotina que usa um desses registradores deve salvar o valor original e restaurar antes do programa terminar. Estes valores são preservados na chamada da sub-rotina.

```
.data
.text
        li $s2, 5           #carrega o valor IMEDIATO 5 no registrador s2
        li $s3, 10          #carrega o valor IMEDIATO 10 no registrador s3
        add $s1, $s2, $s3   #soma s2 com s3 e armazena o resultado em s1
```

Instruções de Leitura e Escrita

- **li**

  SINTAXE: li <registrador>, <valor imediato>

  - A instrução *load immediately* (ler imediatamente) serve para atribuir um valor inteiro diretamente a um registrador.

```
li $t0, 5 #Atribuindo o valor 5 ao registrador $t0
li $t1, 3 #Atribuindo o valor 3 ao registrador $t1
li $t2, -2 #Atribuindo o valor -2 ao registrador $t2
li $t3, 10 #Atribuindo o valor 10 ao registrador $t3
```

Instruções de Leitura e Escrita

- **la**

  SINTAXE: la <registrador>, <endereço de memória variável>

  - A instrução *load address* (ler pelo endereço de memória da variável) serve para atribuir o valor armazenado numa variável diretamente num registrador.

```
.data                   #Diretiva de dados
        #Área para dados na memória principal
        msg:    .asciiz "Olá Mundo!"
.text                   #Diretiva de texto
        #Área para instruções do programa
        li $v0, 4       #Impressão de String
        la $a0, msg     #Vai indicar o endereço em que está a mensagem
        syscall         #Executa a instrução
```

```
.data                   #Diretiva de dados
        #Área para dados na memória principal
        idade:  .word   37
.text                   #Diretiva de texto
        #Área para instruções do programa
        la $t0, idade   #$t0 = idade
        li $v0, 10      #Sair do programa
        syscall         #Executa a instrução
```

Instruções de Leitura e Escrita

- **lw**

  SINTAXE: lw <registrador>, <variável>

  - As instruções de leitura e escrita utilizam acesso direto à memória principal. As memórias possuem dois status: grava ou lê. **lw** (grava).

```
.data                           #Diretiva de dados
        idade: .word 37         #idade - 37
.text                           #Diretiva de texto
        lw $t0, idade           #$t0 = idade
        li $v0, 10
        syscall
```

Instruções de Leitura e Escrita

- **sw**

  SINTAXE: sw <registrador>, <variável>

  - A instrução **sw** (*save*) grava um dado de um registrador diretamente na memória RAM. Esta instrução é o inverso da instrução **lw** (*load*).

```
.data                       #Diretiva de dados
        #Área para dados na memória principal
        idade:  .word   0
.text                       #Diretiva de texto
#Pedindo para o usuário digitar um valor
        li $v0, 5           #Ler número inteiro
        syscall             #Executa a instrução
        move $t0, $v0       #$t0 recebe $v0
#Salvando na variável o número digitado
sw $t0, idade               #$t0 = idade
#finalizando o programa
li $v0,10                   #Sair do programa
syscall                     #Executa
```

Instruções de Movimentação

- **move**

  SINTAXE: move <registrador_destino>, <registrador_origem>

  - A instrução ***move*** copia os dados do registrador origem para o registrador destino. A instrução ***move*** é muito usada quando setamos o registrador **$v0** para receber dados de entrada do usuário, por exemplo, pedir ao usuário que ele digite um número inteiro.

```
.data                       #Diretiva de dados
.text                       #Diretiva de texto
        li $v0, 5           #Ler um número inteiro do teclado
        syscall             #Executa
        move $t0, $v0       #Salvar no registrador $t0
        li $v0, 10          #Sair do programa
        syscall
```

Instruções de Movimentação

- **mfhi**

  SINTAXE: mfhi <registrador_destino

  - A instrução **mfhi** copia a informação que está no registrador **hi** para o registrador destino. O registrador **hi** é usado na multiplicação e na divisão para ganho de funcionalidade. Por exemplo, na divisão o registrador **hi** pode conter o resto da divisão, nos possibilitando assim saber se um determinado número é par ou ímpar.

```
.data                       #Diretiva de dados
.text                       #Diretiva de texto
        li $t1, 5           #$t1 = 5
        li $t2, 2           #$t2 = 2
        div $t1, $t2        #hi = $t1/$t2
        mfhi $t0            #Registrador $t0 recebe hi
        li $v0, 10          #Sair do programa
        syscall             #Executa
```

Instruções de Movimentação

- **mflo**

  SINTAXE: mflo <registrador_destino

  - A instrução **mflo** copia a informação que está no registrador **lo** para o registrador destino. O registrador **lo** é usado na multiplicação e na divisão para o resultado do cálculo.

```
.data                           #Diretiva de dados
.text                           #Diretiva de texto
        li $t1, 18              #$t1 = 18
        li $t2, 3               #$t2 = 3
        div $t1, $t2            #lo = $t1/$t2
        mflo $t0                #Registrador $t0 recebe lo
        li $v0, 10              #Sair do programa
        syscall                 #Executa
```

Impressão

- Impressão de apenas um caractere

```
.data
        char:   .byte 'R' #Caractere que será impresso
.text
        li $v0, 4           #Imprimir uma string
        la $a0, char
        syscall
        li $v0, 10          #Encerrar o programa
        syscall
```

Impressão

- Impressão de inteiros

```
.data
        num: .word 20 #Valor inteiro da memória RAM
.text
        li $v0, 1      #Imprimir inteiro
        lw $a0, num    #Buscar na memória o valor de num e colocar em $a0
        syscall
```

Impressão

- Ler um número inteiro

```
.data
        num: .asciiz "Informe um num: "
.text
        li $v0, 4           #Imprimir a mensagem
        la $a0, num         #Carrega a mensagem
        syscall             #Executa
        li $v0, 5           #Lê um número inteiro
        syscall             #Executa
```

Operações Aritméticas

- Somar com as instruções **add** e **addi**

  SINTAXE: add<registrador_destino>, <registrador1>, <registrador2>

  - A instrução **add**, soma registradores, e a instrução **addi** pode somar números inseridos diretamente no local do registrador.

```
add $t0, $t1, $t2     #t0 = t1 + t2
addi $t3, $t0, 10     #t3 = t0 + 10
```

```
.data                                   #Diretiva de dados
.text                                   #Diretiva de texto
        li $t1, 5                       #$t1 = 5
        li $t2, 2                       #$t2 = 2
        add $t0,$t1, $t2                #$t0 = $t1 + $t2
        addi $t3, $t0, 10               #$t3 = $t0 + 10
        li $v0, 10                      #Sair do programa
        syscall                         #Executa
```

Operações Aritméticas

- Subtrair com as instruções **sub** e **subi**

  SINTAXE: sub<registrador_destino>,
  <registrador1>, <registrador2>

  - São semelhantes as instruções **add** e **addi**. **sub** trabalha com três registradores, sendo dois carregando os dados. **subi** pode carregar um valor imediatamente a própria instrução **subi**.

```
sub $t0, $t1, $t2     #t0 = t1 - t2
subi $t3, $t0, 1      #t3 = t0 - 1
```

```
.data                                   #Diretiva de dados
.text                                   #Diretiva de texto
        li $t1, 9                       #$t1 = 9
        li $t2, 2                       #$t2 = 2
        sub $t0,$t1, $t2                #$t0 = $t1 - $t2
        subi $t3, $t0, 1                #$t3 = $t0 - 1
        li $v0, 10                      #Sair do programa
        syscall                         #Executa
```

Operações Aritméticas

- Multiplicar com a instrução **mul**

  SINTAXE: mul<registrador_destino>, <registrador1>, <registrador2>

  - São usados três registradores, sendo o primeiro registrador usado para receber o resultado dos outros dois registradores usados na sequência.

mul $t2, $t0, $t1 #t2 = t0 * t1

```
.data                                   #Diretiva de dados
.text                                   #Diretiva de texto
        li $t0, 3                       #$t0 = 3
        li $t1, 2                       #$t1 = 2
        mul $t2,$t0, $t1                #$t2 = $t0 * $t1
        li $v0, 10                      #Sair do programa
        syscall                         #Executa
```

Operações Aritméticas

- Dividir com a instrução **div**

  SINTAXE: div<registrador_destino>, <registrador1>, <registrador2>

  - Se usar os três registradores para o cálculo, o primeiro registrador será usado como retorno do cálculo

```
div $t2, $t0, $t1     #t2 = t0 / t1
```

```
.data
.text
        li $t0, 8
        li $t1, 2
        div $t2, $t0, $t1
        li $v0, 10
        syscall
```