

# Guia do MIPS

## Tipos de Dados e Formatações

### Tipos de Dados:

Todas as instruções são de 32 bits  
Byte = 8 bits  
Halfword = 2 bytes  
Word = 4 bytes  
Um caractere ocupa 1 byte na memória  
Um inteiro ocupa 1 word(4 bytes) na memória

### Formatações:

Números são representados normalmente. Ex: 4  
Caracteres ficam entre aspas simples. Ex: 'a'  
Strings ficam entre aspas duplas. Ex: "palavra"

## Registadores

32 registradores

Os registradores são precedidos de \$ nas instruções

Duas formas de representação:

Numero do registrador. \$0 até \$31

Usando os nomes equivalentes (ver abaixo). Ex: \$t1, \$sp

Registadores especiais para guardar resultado das multiplicações e divisões, Lo e Hi

Eles não são acessados diretamente, mas através das instruções: mfhi ("move from Hi") e mflo ("move from Lo")

A pilha começa da parte alta da memória e cresce em direção a parte baixa da memória.

# do Reg.	Nome	Descrição
0	\$zero	retorna o valor 0
1	\$at	(assembler temporary) reservado pelo assembler
2~3	\$v0-\$v1	(values) das expressões de avaliação e resultados de função
4~7	\$a0-\$a3	(arguments) Primeiros quatro parametros para subrotinas. Não é preservado em chamadas de procedures.
8~15	\$t0-\$t7	(temporaries) Subrotinas pode usar salvando-os ou não. Não é preservado em chamadas de procedures.
16~23	\$s0-\$s7	(saved values) Uma subrotina que usa um desses deve salvar o valor original e restaurar antes de terminar. Preservados na chamada de procedures.
24~25	\$t8-\$t9	(temporaries) Usados em adição aos \$t0-\$t7
26~27	\$k0-\$k1	Reservados para uso do tratamento de interrupção/trap.
28	\$gp	(global pointer) Aponta para o meio do block de 64k de memoria no segmento de dados estaticos.
29	\$sp	(stack pointer) Aponta para o top da pilha
30	\$s8/\$fp	(saved values/ frame pointer) Preservado na chamada de procedures.
31	\$ra	(return address)
	\$f0	Recebe o retorno de floats em funções
	\$f12/\$f14	Usados para passar floats para funções
	(\$f12,\$f13)	Usados em conjunto para passar doubles para funções
	(\$f14,\$f15)	

## Estrutura do Programa

Arquivo de texto com a declaração de dados e o código do programa. O arquivo deve ter a extensão .s para ser usado com o simulador SPIM.

A declaração de dados deve vir anterior ao código do programa.

**Declaração de Dados:**

Seção do programa identificado pela diretiva **.data**

Os nomes declarados das variáveis são usados no programa. Dados guardados na memória principal (RAM)

**Código:**

Seção do programa identificado pela diretiva **.text**

Contém o código do programa (instruções).

Ponto de início do código marcado pelo label **main:**

O final da main deve usar a chamada de saída do sistema (exit system call).

Obs: Deixe uma linha vazia ao final do programa para facilitar o simulador

SPIM.

**Comentários:**

Tudo que vem após # em uma linha é considerado comentário.

**Declaração de dados****Formato das declarações:**

nome:    tipo\_de\_dados    valor(es)

cria uma variável na memória, com o tipo especificado, o nome e valores dados.

valor(es) usualmente dão o valor inicial; para reservar memória use o tipo

**.space**, dá o número de espaços a serem alocados.

Obs: Labels sempre são seguidos de dois pontos ( : )

*Exemplos:*

```
var1:            .word    3            # cria uma variável inteiro de valor 3
```

```
array1:           .byte    'a','b' # cria um vetor(array) de dois elementos  
                              já inicializados para a e b
```

```
array2:           .space 40  
# aloca 40 espaços consecutivos de bytes, não inicializados. Poderia  
ser usado como um vetor de 40 caracteres ou um vetor de 10 inteiros,  
por exemplo. (usar um comentário para informar que tipo de vetor vai  
ser usado é uma boa prática de programação).
```

**Instruções****Leitura/Escrita**

Acesso a memória RAM apenas com instruções de leitura e escrita.

Todas outras instruções usam registradores como operando.

**Leitura**

lw        registrador1, offset(registrador2)

- carrega a word que está no endereço (registrador2 + offset) para o registrador1

**Escrita**

sw        registrador1, offset(registrador2)

- copia a word no registrador1 para posição de memória de endereço dado por (registrador2+offset)

### **Movimentação**

move            registrador0, registrador1  
- copia o valor do registrador1 para o registrador0  
Ex: move      \$t2, \$t4    # t2 = t4

### **Aritiméticas**

Devem usar 3 operandos  
Todos operandos são registradores  
O tamanho do operando é word(4 bytes)

add            registrador0, registrador1, registrador2  
- salva o resultado da soma do registrador1 com o registrador2 no registrador0  
obs: soma com sinal  
Ex: add      \$t0, \$t1, \$t2    # t0 = t1 + t2

addi           registrador0, registrador1, imediato  
- salva o resultado da soma do registrador1 com o imediato no registrador0  
obs: soma com sinal  
Ex: addi     \$t0, \$t1, 5      # t0 = t1 + 5

sub            registrador0, registrador1, registrador2  
- salva o resultado da subtração do registrador1 c/ o registrador2 no registrador0  
obs: subtração com sinal  
Ex: sub      \$t0, \$t1, \$t2    # t0 = t1 - t2

mul            registrador0, registrador1, registrador2  
- multiplica o registrador1 pelo registrador2 e guarda no registrador0

div            registrador0, registrador1, registrador2  
- guarda o resultado da divisão inteira do reg1 pelo reg2 no registrador0

rem            registrador0, registrador1, registrador2  
- guarda o resto da divisão do registrador1 pelo registrador2 no registrador0

### **Lógicas**

and            registrador0, registrador1, registrador2  
- guarda o resultado da operação lógica AND entre reg1 e reg2 no registrador0

andi           registrador0, registrador1, imediato  
- guarda o resultado da operação lógica AND entre reg1 e imed no registrador0

neg            registrador0, registrador1  
- guarda o inverso do valor do registrador1 no registrador0

negu           registrador0, registrador1  
- guarda o inverso do valor do registrador1 no registrador0

obs: sem overflow

**nor**            registrador0, registrador1, registrador2

- guarda o resultado da operação lógica NOR entre reg1 e reg2 no registrador0

**not**            registrador0, registrador1

- guarda o resultado da negação binária do valor do registrador1 no registrador0

**or**             registrador0, registrador1, registrador2

- guarda o resultado da operação lógica OR entre reg1 e reg2 no registrador0

**ori**            registrador0, registrador1, imediato

- guarda o resultado da operação lógica OR entre reg1 e imed no registrador0

**rol**            registrador0, registrador1, imediato

- guarda o resultado da rotação para esquerda, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0

**ror**            registrador0, registrador1, imediato

- guarda o resultado da rotação para direita, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0

**sll**            registrador0, registrador1, imediato

- guarda o resultado do deslocamento lógico para esquerda, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0

**sla**            registrador0, registrador1, imediato

- guarda o resultado do deslocamento aritmético para esquerda, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0

**srl**            registrador0, registrador1, imediato

- guarda o resultado do deslocamento lógico para direita, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0

**sra**            registrador0, registrador1, imediato

- guarda o resultado do deslocamento aritmético para direita, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0

**xor**            registrador0, registrador1, registrador2

- guarda o resultado da operação lógica XOR entre reg1 e reg2 no registrador0

**xori**            registrador0, registrador1, imediato

- guarda o resultado da operação lógica XOR entre reg1 e imed no registrador0

## **Desvio**

### **Incondicional**

**b**             label

- muda o registrador PC(registrador que guarda o endereço da próxima instrução a ser executada) para o valor do label

j                    label  
- muda o registrador PC(registrador que guarda o endereço da próxima instrução a ser executada) para o valor do label

jr                    registrador  
- muda o registrador PC(registrador que guarda o endereço da próxima instrução a ser executada) para o endereço contido no registrador

### **Condiciona**

beq                registrador0, registrador1, label  
- desvia para o label, se: registrador0 = registrador1

blt                registrador0, registrador1, label  
- desvia para o label, se: registrador0 < registrador1

ble                registrador0, registrador1, label  
- desvia para o label, se: registrador0 <= registrador1

bgt                registrador0, registrador1, label  
- desvia para o label, se: registrador0 > registrador1

bge                registrador0, registrador1, label  
- desvia para o label, se: registrador0 >= registrador1

bne                registrador0, registrador1, label  
- desvia para o label, se: registrador0 != registrador1

### **Chamada de subrotina**

jal                label  
- desvia para o label da subrotina.  
- copia o PC para o registrador RA

jr                ra  
- desvia para o endereço contido em RA  
- usado para fazer o retorno da subrotina para o programa.

Obs: se uma subrotina for chamar outra subrotina, ou é recursiva, o endereço em RA será sobrescrito, ele deveria então ser empilhado para que ele possa ser preservado e recuperado ao termino das chamadas para dar continuidade ao programa normalmente.

**Exemplo 1: Considere o seguinte código em Python:**

A = B + C

**Convertendo o código Python para o Assembly MIPS:**

**# Definição das Variáveis. Supondo que a A tem 0, B tem 10 e C tem 20.**

.data

A: .word 0

B: .word 25

C: .word 15

**# O programa começa a iniciar a partir daqui.**

.text

.globl main **# Definição de qual Label irá começar.**

**# Label main**

main:

**# Carregando os endereços de A, B e C para os registradores t0, t1 e t2.**

la \$t0, A

la \$t1, B

la \$t2, C

**# Carregando da memória os valores contidos em B e C.**

lw \$t3, 0(\$t1)

lw \$t4, 0(\$t2)

**# Somando t3 (B) e t4 (C) e adicionando em t5.**

add \$t5, \$t3, \$t4

**# Gravando t5 (soma de B e C) em memória de t0 (A).**

sw \$t5, 0(\$t0)

**# Finalizando o programa.**

syscall

**Exemplo 2: Considere o seguinte código em Python:**

```
if B > C:  
    A = B + C
```

**Convertendo o código Python para o Assembly MIPS:**

**# Definicao das Variaveis. Supondo que a A tem 0, B tem 10 e C tem 20.**

```
.data  
A: .word 0  
B: .word 25  
C: .word 15
```

**# O programa começa a iniciar a partir daqui.**

```
.text  
.globl main # Definicao de qual Label ira comecar.
```

**# Label main**

main:

**# Carregando os enderecos de B e C para os registradores t0, e t1.**

```
la $t0, B  
la $t1, C
```

**# Carregando da memoria os valores contidos em B e C.**

```
lw $t2, 0($t0)  
lw $t3, 0($t1)
```

**# Realiza a Condicao (B > C). Se for verdadeiro o programa pula**

**# para o Label IF\_1, caso contrario continua a execucao normal.**

```
bgt $t2, $t3, IF_1
```

**# Label de Continuacao depois do bloco do if.**

continua:

**# Finalizando o programa.**

```
syscall
```

**# Bloco da Soma de B e C, e atribuicao da soma em A.**

IF\_1:

**# Somando B e C e adicionando em t4.**

```
add $t4, $t2, $t3
```

**# Carregando o endereco de A para o registrador t5.**

```
la $t5, A
```

**# Gravando t4 (soma de B e C) em memoria de t5 (A).**

```
sw $t4, 0($t5)
```

**# O programa vai para o Label continua.**

```
j continua
```

### **Exemplo 3: Considere o seguinte código em Python:**

```
def soma (a, b)
    return a + b

soma(10, 20)
```

### **Convertendo o código Python para o Assembly MIPS:**

**# O programa começa a iniciar a partir daqui.**

.text

.globl main **# Definição de qual Label irá começar.**

**# Label main**

main:

sub \$sp, \$sp, 12 **# Aloca 3 espaços (cada um de 4 bits) na pilha**

addi \$s0, \$zero, 10 **# Define 10 a s0**

addi \$s1, \$zero, 20 **# Define 20 a s1**

sw \$s0, 0(\$sp) **# Coloca na fila o primeiro parametro**

sw \$s1, 4(\$sp) **# Coloca na fila o segundo parametro**

jal soma **# Chama o Label da função soma através do jal**

lw \$s2, 0(\$sp) **# Obtem o resultado da função (o retorno)**

add \$sp, \$sp, 12 **# Desaloca 3 espaços na pilha**

syscall **# Finaliza o programa**

**# Bloco da função soma**

soma:

sw \$ra, 8(\$sp) **# Grava o endereço de retorno na pilha**

lw \$t0, 0(\$sp) **# Carrega primeiro parametro**

lw \$t1, 4(\$sp) **# Carrega segundo parametro**

add \$t0, \$t0, \$t1 **# Realiza a soma de a + b**

sw \$t0, 0(\$sp) **# Grava o resultado da soma na pilha**

lw \$ra, 8(\$sp) **# Carrega o endereço de retorno**

jr \$ra **# Retorna ao código principal (Label main)**