Guia Rápido 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"

Registradores

32 registradores
Os registradores são procedidos de $ nas instruções
Duas formas de representação:

Numero do registrador. $0 até $31
Usando os nomes equivalentes (ver abaixo). Ex: $t1, $sp

Registradores 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) ($f14,$f15)	Usados em conjunto para passar doubles para funções

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**

Contêm o código do programa (instruções).

Ponto de inicio 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/Escrita de endereçamento direto

Leitura:

lw registrador, posição_da_RAM

- copia word(4 bytes) da posição da RAM dada, para o registrador dado.

lb registrador, posição_da_RAM

- copia byte da posição da RAM dada, para a parte baixa do registrador dado.

li registrador, valor
- carrega o valor para o registrador de destino.

Escrita:

sw registrador, posição_da_RAM
- escreve a word do registrador dado na posição da RAM dada.

sb registrador, posição_da_RAM
- escreve o byte da parte mais baixa do registrador dado para a posição da RAM

Exemplo:

```
        .data
var1:   .word   23

        .text
main:
        lw      $t0, var1       # carrega o conteúdo de var1 em $t0
        li      $t1, 5          #  $t1 = 5
        sw      $t1, var1       # carrega o conteúdo de $t1 em var1
```

Leitura/Escrita de endereçamento indireto e por base

Leitura

la registrador, label
- copia o endereço do label na memória para o registrador dado

Endereçamento indireto

lw registrador1, (registrador2)
- carrega a word que está no endereço dado pelo registrador2, para o registrador1

Endereçamento por base

lw registrador1, offset(registrador2)
- carrega a word que está no endereço (registrador2 + offset) para o registrador1
obs: o offset pode ser negativo

Escrita

Endereçamento indireto

sw registrador1, (registrador2)
- copia a word no registrador1 para posição de memória de endereço dado pelo registrador2

Endereçamento por base

sw registrador1, offset(registrador2)
- copia a word no registrador1 para posição de memória de endereço dado por (registrador2+offset)
obs: o offset pode ser negativo

Exemplo:

```
                .data
array1:         .space  12              # array de 3 inteiros

                .text
main:           la      $t0, array1     # carrega o endeço do array1
```

```
                                # para t0
        li      $t1, 5          # t1 = 5
        sw $t1, ($t0)           # 1° elemento do array1 = 5
        li $t1, 13              # t1 = 13
        sw $t1, 4($t0)          # 2° elemento do array1 = 13
        li $t1, -7              # t1 = -7
        sw $t1, 8($t0)          # 3° elemento do array1 = -7
```

Movimentação

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

mfhi registrador0
- copia o valor do registrador Hi para o registrador0
Ex: mfhi $t1 # t1 = Hi

mflo registrador0
- copia o valor do registrador Lo para o registrador0
Ex: mflo $t1 # t1 = Lo

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

addu registrador0, registrador1, registrador2
- salva o resultado da soma do registrador1 com o registrador2 no registrador0
obs: soma sem sinal

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

subu registrador0, registrador1, registrador2
- salva o resultado da subtração do registrador1 c/ o registrador2 no registrador0
obs: subtração sem sinal

mul registrador0, registrador1, registrador2
- multiplica o registrador1 pelo registrador2 e guarda no registrador0
obs: multiplicação sem overflow

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

mulou registrador0, registrador1, registrador2
- multiplica o registrador1 pelo registrador2 e guarda no registrador0
obs: Multiplicação sem sinal

mult registrador1, registrador2
- multiplica o registrador1 pelo registrador2 e guarda nos registradores especiais
obs: multiplicação de 32-bits que gera um resultado de 64-bits que fica guardado em Hi e Lo.
Ex: mult $t3, $t4 # Hi, Lo = t3 * t4

multu registrador1, registrador2
- multiplica o registrador1 pelo registrador2 e guarda nos registradores especiais
obs: multiplicação de 32-bits que gera um resultado de 64-bits que fica guardado em Hi e Lo.
obs2: multiplicação sem sinal
Ex: mult $t3, $t4 # Hi, Lo = t3 * t4

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

divu registrador0, registrador1, registrador2
- guarda o resultado da divisão inteira do reg1 pelo reg2 no registrador0
obs: divisão sem sinal

div registrador1, registrador2
- divide o registrador1 pelo registrador2 e guarda nos registradores especiais
obs: O quociente fica guardado em Lo e o resto fica guardado em Hi

divu registrador1, registrador2
- divide o registrador1 pelo registrador2 e guarda nos registradores especiais
obs: O quociente fica guardado em Lo e o resto fica guardado em Hi
obs2: divisão sem sinal

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

remu registrador0, registrador1, registrador2
- guarda o resto da divisão do registrador1 pelo registrador2 no registrador0
obs: divisão sem sinal.

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

Condicional

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.

Entrada/Saída e chamadas de sistema

Usado para ler ou imprimir valores ou strings da Entrada/Saída, e indicar o fim de programa

Usar a chamada de rotina **syscall**

Deve-se informar os valores necessários nos registradores v0, a0 e a1

Se houver um retorno, ele será no registrador v0

Função	Código em v0	Argumentos	Resultado
imprime int	1	a0 = inteiro a ser impresso	
imprime float	2	f12 = float a ser impresso	
imprime double	3	f12 = double a ser impresso	
imprime string	4	a0 = endereço de memoria da string	
ler int	5		inteiro retornado em v0
ler float	6		float retornado em v0
ler double	7		double retornado em v0
ler string	8	a0 = endereço de memoria da string a1 = tamanho da string	
exit	10		endereço em v0

A função de imprimir string espera uma string terminada com caractere nulo. A diretiva .asciiz cria uma string terminada em caracter nulo.

A leitura de inteiros, floats e doubles lêem um liha inteira, inclusive o caractere de nova linha.

A função de ler strings tem a mesma semântica da rotina fgets da biblioteca UNIX:

- Lê até a posição n-1 e termina com o caractere nulo.
- Se tiver menos de n-1 caracteres na linha, todos são lidos, inclusive o caractere de nova linha, e a string é terminada com o caractere nulo.

A função exit finaliza o programa.

Exemplo: Imprime o inteiro que está em t2

```
        li      $v0, 1      # carrega o código de imprimir int.
        move    $a0, $t2    # move o inteiro para a0
        syscall             # chama o SO para realizar a operação
```

Exemplo: Lê o inteiro, guardando na memoria, na posição de memória com label int_value (declarada na sessão de data)

```
        li      $v0, 5          # carrega o código de ler inteiro
        syscall                 # chama o SO para realizar a operação
        sw      $v0, int_value  # o valor foi retornado em v0
```

Exemplo: Imprime uma string

```
                .data
string1         .asciiz   "frase para imprimir.\n"

                .text
main:           li      $v0, 4  # carrega o código de imprimir string
                la      $a0, string1 # carrega o endereço da string a
                                     ser impressa em a0
                syscall         # chama o SO para realizar a operação
```

Exemplo: Terminando o programa

```
        li      $v0, 10         # carrega o código da operação
        syscall                 # chama o SO para realizar a operação
```

Referências:

http://msdn.microsoft.com/en-us/library/ms253512%28VS.80%29.aspx
http://www.doc.ic.ac.uk/lab/secondyear/spim/node13.html
http://logos.cs.uic.edu/366/notes/MIPS%20Quick%20Tutorial.htm