MIPS Architecture and Assembly Language

Visão Geral da Arquitetura

Tipos de dados e Constantes

Tipos de dados:

- byte, meia-palavra (2 bytes), palavra (4 bytes)
- um caractere requer 1 byte para ser armazenado
- um inteiro requer 1 palavra (4 bytes) para ser armazenado

Constantes (Literais):

- números diretamente: 4
- caracteres entre aspas simples: 'b'
- strings entre aspas duplas: "Uma string"

Registradores

- 32 registradores de propósitos gerais
- Precedidos por $ nas instruções em assembly
  dois formatos de representação:
  - Usando o número do registrador ex $0 até $31
  - Usando o nome equivalente ex $t1, $sp
- Convenção de uso dos registradores
  - $t0 - $t9 (= $8 - $15, $24, $25) registradores de uso geral; não precisam ser preservados nas chamadas de procedimentos
  - $s0 - $s7 (= $16 - $23) registradores de uso geral; devem ser preservados nas chamadas de procedimentos
  - $sp (= $29) é o stack pointer
  - $fp (= $30) é o frame pointer
  - $ra (= $31) é o endereço de retorno da chamada de uma sub-rotina
  - $a0 - $a3 (= $4 - $7) são usados para passar argumentos para uma sub-rotina
  - $v0, $v1 (= $2, $3) são utilizados para o retorno de valores de uma sub-rotina
- Registradores especiais Lo e Hi usados para armazenar o resultado de divisões e multiplicações
  - Não são diretamente endereçáveis; seus conteúdos são acessados através de instruções especiais mfhi ("move from Hi") e mflo ("move from Lo")
- stack cresce dos endereços superiores para os inferiores (de cima para baixo)

MIPS Assembly Language Program Structure

- apenas um arquivo formato texto (não binário) com declaraçõpes de dados, código do programa (extensão . asm ou .s)
- seção de declaração de dados seguida da seção de código do programa

Declarações de dados

- colocados em uma seção identificada pela diretiva **.data**
- declara nomes de variáveis usadas no programa; aloca espaço de armazenamento na memória RAM

Código

- colocados em uma seção identificada pela diretiva **.text**
- o ponto inicial do programa é indicado pela label **main:**
- o final do código principal deve usar uma chamada de sistema (System Calls) para ser finalizado (veja abaixo em chamadas de sistema)

Comentários

- qualquer coisa depois de # em uma linha
- # Isto é considerado um comentário

```
ex:
Um template para a linguagem assembly do MIPS

# Comentário apresentando o nome do programa, a data, uma descrição do que
#  ele faz, de suas funções principais, etc
# Template.s
#
        .data
# declarações de variáveis aqui
# ...


        .text

main:                   # indica o início do código
# continuação do programa
# ...
# ...
```

Declarações de Dados

Formato das declarações:

```
nome:   tipo_do_armazenamento  valor(es)
```

- cria um espaço de armazenamento para uma variável de determinado tipo com um nome e valor especificados
- valor(es) usualmente possuem um valor inicial; o tipo .space informa o número de bytes a ser alocado, sem vincular a nenhuma variável específica

Nota: o rótulo deve ser sempre seguido de (:)

```
exemplo

var1:           .word   3       # cria uma variável inteira com valor inicial 3
array1:         .byte   'a','b' # cria um vetor de dois caracteres inicializado
                                #  com a e b
array2:         .space  40      # aloca 40 bytes consecutivos, sem inicialização
                                #   podem ser 40 caracteres ou 10 inteiros
```

Instruções

Instruções Load / Store

- Acesso a RAM só é permitido através destas instruções
- Todas as demais instruções utilizam registradores

load:

```
lw      registrador_de_destino, endereço_de_origem_da_RAM
```

- Copia uma palavra (4 bytes) do endereço de origem da RAM para o registrador de destino

```
lb      register_ registrador, endereço_de_origem_da_RAM
```

- Copia um (1) byte do endereço de origem da RAM para o byte de baixa ordem do registrador de destino (estende o sinal para os bytes de alta ordem)

store word:

```
sw      registrador_de_origem, endereço_de_destino_da_RAM
```

- Armazena uma palavra (4 bytes) presente no registrador de origem no endereço de destino da RAM

```
sb      registrador_de_origem, endereço_de_destino_da_RAM
```

- Armazena um (1) byte presente no registrador de origem no endereço de destino da memória RAM

load imediato:

```
li      registrador_de_destino, valor
```

- Carrega "valor" no registrador de destino

Exemplo

```
                .data
var1:           .word   23              # declara var1; inicializa com 23

                .text
__start:
                lw      $t0, var1       # lê conteúdo da RAM em $t0:  $t0 = (var1)
                li      $t1, 5          #  $t1 = 5   ("load immediate")
                sw      $t1, var1       # armazena $t1 na RAM:  (var1) = $t1
                done
```

Endereçamento Indireto e Base + DeslocamentoIndirect and Based Addressing

- Utilizado somente em instruções load e store

load address:

```
la      $t0, var1
```

- Copia o valor var1 (presumido como um rótulo definido no programa) no registrador $t0

Endereçamento indireto:

```
lw      $t2, ($t0)
```

- Carrega o valor presente no endereço de memória apontado por $t0 no registrador $t2

```
sw      $t2, ($t0)
```

- Armazena o valor presente no registrador $t2 no endereço de memória apontado pelo registrador $t0

Endereçamento Base+Deslocamento:

```
lw      $t2, 4($t0)
```

- Carrega em $t2 o conteúdo do endereço de memória obtido a partir de ($t0 + 4)
- "4" é o deslocamento em relação a posição inicial apontada por $t0

```
sw      $t2, -12($t0)
```

- Armazena o valor presente em $t2 no endereço de memória obtido a partir de ($t0 – 12)
- Deslocamentos “negativos” são permitidos

Nota: endereçamento base+deslocamento é especialmente usado em:

- arrays; acessar os elementos do array a partir do endereço base;
- pilhas; fácil de acessar elementos a partir do ponteiro para início da pilha

```
exemplo
                .data
array1:         .space  12              #  declara 12 bytes para armazenamento
                .text
main:           la      $t0, array1     #  lê o endereço base do array em $t0
                li      $t1, 5          #  $t1 = 5   ("load immediate")
                sw      $t1, ($t0)      #  armazena 5 no primeiro elemento do array;
                li      $t1, 13         #   $t1 = 13
                sw      $t1, 4($t0)     #  armazena 13 no segundo elemento do array;
                li      $t1, -7         #   $t1 = -7
                sw      $t1, 8($t0)     #  armazena -7 no terceiro elemento do array;
                done
```

Instruções Aritméticas

- devem usar 3 operandos
- todos os operandos são registradores;
- dados são do tamanho da palavra (4 bytes)

```
                add     $t0,$t1,$t2     #  $t0 = $t1 + $t2;
                sub     $t2,$t3,$t4     #  $t2 = $t3 Ð $t4
                addi    $t2,$t3, 5      #  $t2 = $t3 + 5;
                addu    $t1,$t6,$t7     #  $t1 = $t6 + $t7;
                subu    $t1,$t6,$t7     #  $t1 = $t6 + $t7;

                mult    $t3,$t4         #  multiplica valor de 32-bits de $t3 e $t4,
                                        # e armazena em 64 bits: (Hi,Lo) = $t3 * $t4
                                        # resultado é armazenado em registradores
                                        # especiais Lo e Hi:  (Hi,Lo) = $t3 * $t4
                div     $t5,$t6         #  Lo = $t5 / $t6   (quociente inteiro)
                                        #  Hi = $t5 mod $t6   (resto)
                mfhi    $t0             #  move conteúdo de Hi para $t0:   $t0 = Hi
```

```
mflo    $t1             #  move conteúdo de Lo para $t1:   $t1 = Lo
move    $t2,$t3         #  $t2 = $t3
```

Estruturas de Controle

Desvios

- comparações para desvios condicionais são construídos na própria instrução

```
b       target                  # desvio incondicional p/ rótulo target
beq     $t0,$t1,target          # desvia para target se  $t0 = $t1
blt     $t0,$t1,target          # desvia para target se $t0 < $t1
ble     $t0,$t1,target          # desvia para target se $t0 <= $t1
bgt     $t0,$t1,target          # desvia para target se $t0 > $t1
bge     $t0,$t1,target          # desvia para target se $t0 >= $t1
bne     $t0,$t1,target          # desvia para target se $t0 <> $t1
```

Saltos (desvios incondicionais)

```
j       target          # desvio incondicional p/ rótulo target
jr      $t3             # desvio incondicional para endereço
                        #  contido em $t3 ("jump register")
```

Sub-rotinas

Chamada de sub-rotina: instrução "jump and link"

```
jal     sub_label       #  "jump and link"
```

- copia PC (endereço de retorno) para registrador $ra (return address register)
- desvia o programa para o endereço marcado pelo rótulo (no caso sub_label)

Retorno de sub-rotina: instrução "jump register"

```
jr      $ra     #  "jump register"
```

- desvia o programa para o endereço contido no registrador $ra (armazenado anteriormente pela instrução jal)

Nota: o endereço de retorno é armazenado automaticamente em $ra pela instrução jal. Caso a sub-rotina chamada deseje fazer uma chamada a outra sub-rotina, esta deve primeiro salvar o conteúdo de $ra para que o mesmo não seja sobrescrito.

Chamadas de Sistema e I/O (Simulador SPIM)

- usado para ler ou imprimir valores e strings na janela de I/O e indicar o fim do programa
- para chamar a rotina de chamada do sistema (**syscall**) deve-se:
  - colocar os valores apropriados nos registradores $v0 e $a0-$a1
  - o valor resultante (se existir) será colocado em $v0

```
ex   Mostrar no console um valor inteiro contido no registrador $t2
```

```
        li      $v0, 1       # carrega código de chamada de sistema apropriado em $v0;
                             #  no caso (imprimir inteiro) o código é 1
        move    $a0, $t2     # move o inteiro a ser impresso em $a0. $a0:  $a0 = $t2
        syscall              # faz a chamada do sistema
```

ex Lê um valor inteiro e armazena na memória

```
        li      $v0, 5          # carrega a chamada de sistema apropriada em $v0;
                                # no caso (ler inteiro) o código é 5
        syscall                 # faz a chamada do sistema
        sw      $v0, int_value  # valor lido(presente em $v0) é armazenado na memória;
```

ex Mostra uma string (useful for prompts)

```
           .data
string1    .asciiz     "Print this.\n" # declaração da variável string1

           .text
main:      li   $v0, 4          # carrega a chamada de sistema aprorpiada em $v0
                                # no caso (imprimir string) o código é 4
           la   $a0, string1    # carrega endereço da string a ser mostrada em $a0
                syscall         # faz a chamada do sistema
```

Nota:

- a string deve ser implementada como um array de caracteres terminado por (\0)
- declaração do tipo de dado .asciiz coloca automaticamente o terminal (\0)
- para inficar fim do programa, utilize a chamada do sistema **exit**, por exemplo

```
                li      $v0, 10         # faz $v0 = 10 (chamada de sistema exit)
                syscall                 # faz a chamada de sistema
```

Tabela de chamadas de sistema e seus argumentos (retirada de: SPIM S20: A MIPS R2000 Simulator, James J. Larus, University of Wisconsin-Madison)

Serviço	Código da Chamada de Sistema	Argumentos	Resultado
print inteiro	1	$a0 = valor	(nenhum)
print float	2	$f12 = valor ponto flutuante	(nenhum)
print double	3	$f12 = valor double	(nenhum)
print string	4	$a0 = endereço da string	(nenhum)
lê inteiro	5	(nenhum)	$v0 = valor lido
lê float	6	(nenhum)	$f0 = valor lido
lê double	7	(nenhum)	$f0 = valor lido
lê string	8	$a0 = endereço onde a string deve ser armazenada $a1 número de caracteres lidos + 1	(nenhum)
Alocação de memória	9	$a0 = número de bytes a serem alocados	$v0 = endereço do bloco
exit (fim do programa)	10	(nenhum)	(nenhum)

Exemplo 1

```
# Implementar o valor da soma de     1*2 + 2*3 + 3*4 + ... + 10*11,
# e armazene o resultado no registrador $t1

                .data                           # seção de declaração de variáveis

out_string:     .asciiz  "O resultado é:\n"     # declara uma string

                .text

main:                                           # inicio do programa
                li      $t0, 1                  # $t0 será um contador; inicializado com 1
                li      $t1, 0                  # $t1 armazena o somatório
                li      $t2, 10                 # $t2 armazena o limite do laço

loop_top:       bgt     $t0,$t2,loop_end        # finaliza laço se  $t0 > 10
                addi    $t3,$t0,1               # $t3 = $t0 + 1
                mult    $t0,$t3                 # Lo = $t0 * $t3
                                                # (não precisa Hi pois valores são pequeno)
                mflo    $t3                     # $t3 = Lo (= $t0 * $t3)
                add     $t1,$t1,$t3             # $t1 = $t1 + $t3
                addi    $t0, 1                  # incrementa contador
                b       loop_top                # desvia para o inicio do laço

loop_end:                                       # imprime o resultado
                li      $v0, 4                  # código da chamada de sistema para
                                                #  imprimir uma string (4)
                la      $a0, out_string         # carrega posição inicial da string em $a0
                syscall                         # chamada de sistema para impressão
                                                # print out integer value in $t1
                li      $v0, 1                  # código da chamada de sistema para
                                                #  imprimir um inteiro (1)
                move    $a0, $t1                # carrega valor a ser impresso para $a0
                syscall                         # chamada de sistema para impressão

                                                # termina o programa
                li      $v0, 10                 # código da chamada do sistema para
                                                #  finalizar o programa (10)
                syscall                         # chamada de sistema

                                                # blank line at end to keep SPIM happy!
```