UNIVERSIDADE FEDERAL DE RORAIMA

Programação Assembly MIPS

Prof. Herbert Oliveira Rocha

UNIVERSIDADE FEDERAL DE RORAIMA

Programação Assembly MIPS

Baseado nas aulas do Prof. Dr. Aiman El-Maleh
King Fahd University of Petroleum and Minerals

Prof. Herbert Oliveira Rocha

Instruções

- Formato da linguagem:

```
[label:]    mnemonic    [operands]      [#comment]
```

- Label: (opcional)
  - Marca o endereço de um local de memória
  - Normalmente aparecem em dados e segmentos de texto
- Mnemonic
  - Identifica a operação (e.g. **`add`**, **`sub`**, etc.)
- Operands
  - Especifica os dados exigidos pela operação
  - Operandos podem ser registos, variáveis da memória, ou constantes
  - A maioria das instruções têm três operandos

```
L1:   addiu $t0, $t0, 1          #increment $t0
```

Programa Template

```
# Title:                                    Filename:
# Author:                                   Date:
# Description:
# Input:
# Output:
################### Data segment ########################
.data
  . . .
################### Code segment ########################
.text
.globl main
main:                                       # main program entry
  . . .
li $v0, 10                                  # Exit program
syscall
```

Diretivas

- **.DATA**
  - Define o segmento de dados de um programa que contenha dados
  - Variáveis do programa devem ser definidos no âmbito desta diretiva
  - Assembler irá alocar e inicializar o armazenamento de variáveis
- **.TEXT**
  - Define o segmento de código de um programa contendo instruções
- **.GLOBL**
  - Declara um símbolo como global
  - Símbolos globais podem ser referenciadas partir de outros arquivos
  - Usamos essa diretiva para declarar procedimento principal (main) de um programa

Declaração de Definição de Dados

- Syntax:

  [*name:*] *directive* *initializer* [, *initializer*] . . .

```
var1:  .WORD     10
```

- Todos os inicializadores tornar-se dados binários na memória

Diretivas de Dados

- **.BYTE**
  - Armazena a lista de valores como bytes de 8 bits
- **.HALF**
  - Armazena a lista como valores de 16 bits alinhados no limite de meia palavra
- **.WORD**
  - Armazena a lista como valores de 32 bits alinhados em um limite de palavra
- **.WORD w:n**
  - Armazena o valor de 32 bits **w** em **n** em palavras consecutivas alinhadas sobre o limite de uma palavra.

Diretivas de Dados

- **.FLOAT**
  - Armazena os valores listados como precisão simples de ponto flutuante
- **.DOUBLE**
  - Armazena os valores listados como de dupla precisão de ponto flutuante

Diretivas de String

- **.ASCII**
  - Aloca uma seqüência de bytes para uma cadeia de caracteres ASCII
- **.ASCIIZ**
  - Mesmo que **.ASCII**, mas acrescenta um char NULL no final da string
  - Strings são terminadas em null, como na linguagem de programação C
- **.SPACE n**
  - Aloca espaço de n bytes não inicializadas no segmento de dados
- Caracteres especiais em cadeias que segue a convenção C
  - Newline: \n Tab:\t Quote: \”

Exemplos

```
.DATA
var1:   .BYTE     'A', 'E', 127, -1, '\n'
var2:   .HALF     -10, 0xffff
var3:   .WORD     0x12345678
Var4:   .WORD     0:10
var5:   .FLOAT    12.3, -0.1
var6:   .DOUBLE   1.5e-10
str1:   .ASCII    "A String\n"
str2:   .ASCIIZ   "NULL Terminated String"
array:  .SPACE    100
```

Se o valor inicial excede o tamanho máximo, um erro é relatado pelo assembler

Tabela de Símbolo

- Assembler computa o endereço de cada label no segemento de dados

Example

```
.DATA
var1:   .BYTE   1, 2,'Z'
str1:   .ASCIIZ "My String\n"
var2:   .WORD   0x12345678
.ALIGN  3
var3:   .HALF   1000
```

Symbol Table

Label	Address
var1	0x10010000
str1	0x10010003
var2	0x10010010
var3	0x10010018

System Calls

- input/output do programa via system calls
- MIPS prover uma instrução especial **`syscall`**
  - Para obter serviços a partir do sistema operacional
  - Os simuladores SPIM e MARS têm suporte a estes serviços
- Usando **`syscall`**
  - Load o número do serviço no registrador $v0
  - Load os argumentos, se tiver, nos registradores $a0, $a1, etc.

System Calls

Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	$v0 = integer read
Read Float	6	$f0 = float read
Read Double	7	$f0 = double read
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Exit Program	10	
Print Char	11	$a0 = character to print
Read Char	12	$a0 = character read

Ler e Imprimir um inteiro

```
################## Code segment ######################
.text
.globl main
main:                                   # main program entry
   li    $v0, 5                         # Read integer
   syscall                              # $v0 = value read

   move  $a0, $v0                       # $a0 = value to print
   li    $v0, 1                         # Print integer
   syscall

   li    $v0, 10                        # Exit program
   syscall
```

Ler e imprimir String

```
################## Data segment ######################
.data
   str: .space  10          # array of 10 bytes
################## Code segment ######################
.text
.globl main
main:                       # main program entry
   la    $a0, str           # $a0 = address of str
   li    $a1, 10            # $a1 = max string length
   li    $v0, 8             # read string
   syscall
   li    $v0, 4             # Print string str
   syscall
   li    $v0, 10            # Exit program
   syscall
```

Programa 1: Soma de três inteiros

```
# Sum of three integers
#
# Objective: Computes the sum of three integers.
#     Input: Requests three numbers.
#    Output: Outputs the sum.
#################### Data segment ####################
.data
prompt:  .asciiz     "Please enter three numbers: \n"
sum_msg: .asciiz     "The sum is: "
#################### Code segment ####################
.text
.globl main
main:
      la    $a0,prompt          # display prompt string
      li    $v0,4
      syscall
      li    $v0,5               # read 1st integer into $t0
      syscall
      move  $t0,$v0
```

Programa 1: Soma de três inteiros

```
li     $v0,5                # read 2nd integer into $t1
syscall
move   $t1,$v0

li     $v0,5                # read 3rd integer into $t2
syscall
move   $t2,$v0

addu   $t0,$t0,$t1          # accumulate the sum
addu   $t0,$t0,$t2

la     $a0,sum_msg          # write sum message
li     $v0,4
syscall

move   $a0,$t0              # output sum
li     $v0,1
syscall

li     $v0,10               # exit
syscall
```

Procedimentos

- Considerando a seguinte função swap (em C)
- Traduza a função para linguagem MIPS

```
void swap(int v[], int k)
{   int temp;
    temp = v[k]
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

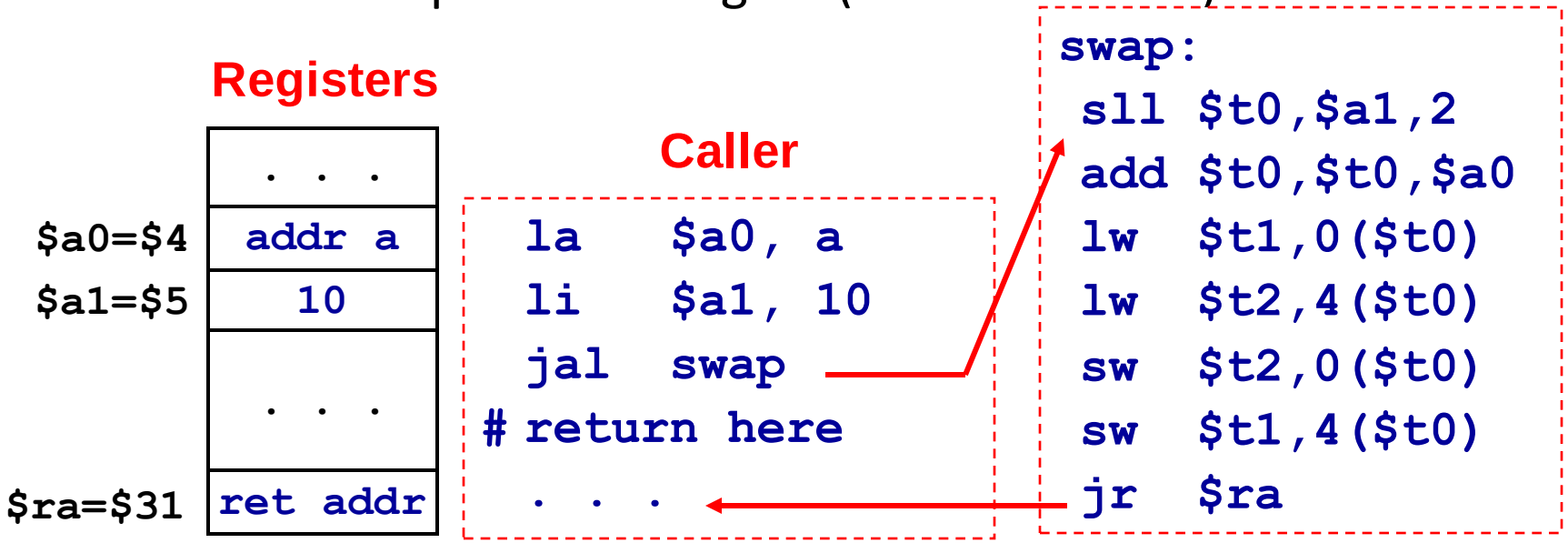
Parameters:

`$a0` = Address of `v[]`
`$a1` = `k`, and
Return address is in `$ra`

```
swap:
 sll $t0,$a1,2     # $t0=k*4
 add $t0,$t0,$a0   # $t0=v+k*4
 lw  $t1,0($t0)    # $t1=v[k]
 lw  $t2,4($t0)    # $t2=v[k+1]
 sw  $t2,0($t0)    # v[k]=$t2
 sw  $t1,4($t0)    # v[k+1]=$t1
 jr  $ra           # return
```

Call / Return

- Supondo a seguinte chamada: **`swap(a,10)`**
  - É passado o **address** do array **`a`** e **`10`** como argumentos
  - A chamada da função swap salvo o **return address** em **$31 = $ra**
  - Executa a função swap
  - Retorno ao ponto de origem (return address)



Jal e Jr

Endereço	Intruções	Assembly
00400020	lui $1, 0x1001	la $a0, a
00400024	ori $4, $1, 0	
00400028	ori $5, $0, 10	li $a1,10
0040002C	jal 0x10000f	jal swap
00400030	. . .	# return here
		swap:
0040003C	sll $8, $5, 2	sll $t0,$a1,2
00400040	add $8, $8, $4	add $t0,$t0,$a0
00400044	lw $9, 0($8)	lw $t1,0($t0)
00400048	lw $10,4($8)	lw $t2,4($t0)
0040004C	sw $10,0($8)	sw $t2,0($t0)
00400050	sw $9, 4($8)	sw $t1,4($t0)
00400054	jr $31	jr $ra

Pseudo-Direct Addressing

PC = imm26<<2

0x10000f << 2

= 0x0040003C

$31 0x00400030

Registrado $31 tem o endereço de retorno

Praticando

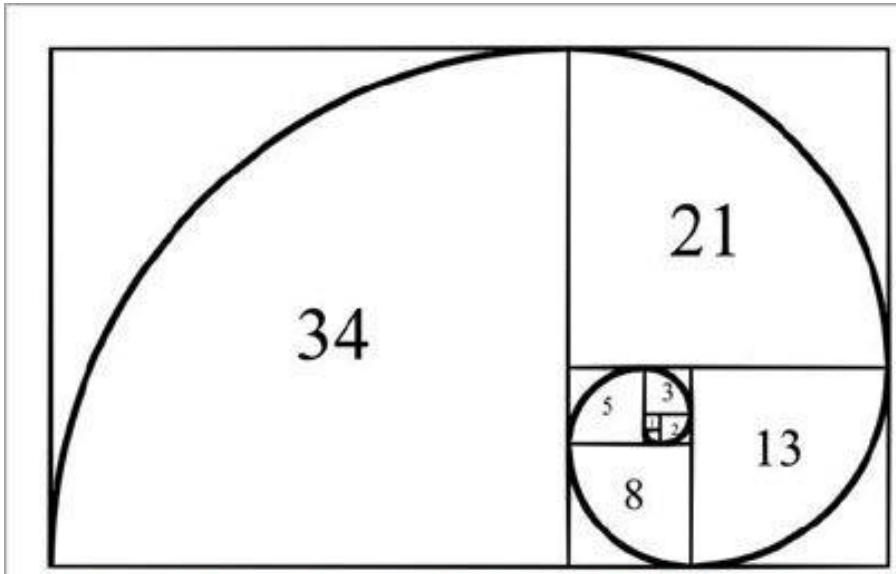
Encontre o valor máximo

Praticando

```
# Objetivo : Encontrar o elemento com maior valor
#     Input: $a0 = pointer to first, $a1 = pointer to last
#    Output: $v0 = pointer to max,   $v1 = value of max
##############################################################
max:  move  $v0, $a0         # max pointer = first pointer
      lw    $v1, 0($v0)      # $v1 = first value
      beq   $a0, $a1, ret    # if (first == last) return
      move  $t0, $a0         # $t0 = array pointer
loop: addi  $t0, $t0, 4      # point to next array element
      lw    $t1, 0($t0)      # $t1 = value of A[i]
      ble   $t1, $v1, skip   # if (A[i] <= max) then skip
      move  $v0, $t0         # found new maximum
      move  $v1, $t1
skip: bne   $t0, $a1, loop   # loop back if more elements
ret:  jr    $ra
```

Praticando

```
int fact(int n) {
      if (n<2) return 1;
      else
         return (n*fact(n-1));
}
```



0, 1, 1, 2, 3, 5, 8, 13, 21, ,34, 55, 89, 144...

$0 + 1 = 1$
$1 + 1 = 2$
$2 + 1 = 3$
$3 + 2 = 5$
$5 + 3 = 8$
$8 + 5 = 13$
$13 + 8 = 21$
$21 + 13 = 34$
$34 + 21 = 55$
$55 + 34 = 89$
$89 + 55 = 144$

Praticando

int fact(int n) { if (n<2) return 1; else return (n*fact(n-1)); }

```
fact:  slti    $t0,$a0,2      # (n<2)?
       beq     $t0,$0,else    # if false branch to else
       li      $v0,1          # $v0 = 1
       jr      $ra            # return to caller
else:  addiu   $sp,$sp,-8     # allocate 2 words on stack
       sw      $a0,4($sp)     # save argument n
       sw      $ra,0($sp)     # save return address
       addiu   $a0,$a0,-1     # argument = n-1
       jal     fact           # call fact(n-1)
       lw      $a0,4($sp)     # restore argument
       lw      $ra,0($sp)     # restore return address
       mul     $v0,$a0,$v0    # $v0 = n*fact(n-1)
       addi    $sp,$sp,8      # free stack frame
       jr      $ra            # return to caller
```