4 - MIPS: Conjunto de Instruções

sábado, 10 de abril de 2021 17:59

Assistir

Instruções de Desvio

Programas necessitam de desvios e repetições em sua execução conforme a verificação de alguma condição

BEQ: desvio se for igual
beq registrador1, registrador2, L1
BNE: desvio se não for igual
bne registrador1, registrador2, L1

Pula para a instrução com label L1

Salto incondicional(jump)
J L1 # jump to L1
Jr $reg #jump para o endereço em $reg

Label: endereço da memória onde se encontra a instrução

Estudo do MIPS: Instruções

Formato da linguagem
[Label:] mnemonic [operands] [#comand]

Programa Template

```
# Title: Calcular fatorial          Filename: fatorial.s
# Author: Karen Giovanna           Date: 07 de março de 2021
# Description: Utilizando a linguagem de programação MIPS, escreva um programa que contenha uma função para calcular o fatorial (opcional: com recursividade) de um dado número inteiro.
# Input:
# Output:

################## Data segment ####################
.data
. . .

################## Code segment ####################
.text
.globl main
main:        # main program entry
. . .


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

Declaração de Definição de Dados

Syntax:
[name:] directive inializer [.initializer] ...
Var1: .WORD 10

Diretivas de Dados	Diretivas de String
.BYTE	.ASCII
.HALF	.ASCIIZ
.WORD	.SPACE n
.WORD w:n	
.FLOAT	
.DOUBLE	

Tabela de símbolos

Assembler computa o endereço de cada label no segmento de dados

LABEL	ADRESS

var1	0x10010000
str1	0x10010003

System Calls

Faz o input/output do programa via system calls
MIPS provém uma instrução especial: syscall
Para obter serviços a partir do sistema operacional
Simuladores SPIM e MARS tem suporte a estes serviços

Usando syscall
Load o número do serviço no registrador $v0
Load dos 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 Inteiro em MIPS

```
1 .text
2 .globl main
3 main:
4     #entry point
5     li $v0, 5 # ler um inteiro
6     syscall
7
8     move $a0, $v0
9     li $v0, 1
10    syscall
11
12    li $v0, 10 # end program
13    syscall
```

Inserindo o arquivo no QTSPIM

```
                                User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)          ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4       ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4        ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2          ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2        ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]  ; 188: jal main
[00400018] 00000000  nop                    ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10         ; 191: li $v0 10
[00400020] 0000000c  syscall                ; 192: syscall # syscall 10 (exit)
[00400024] 34020005  ori $2, $0, 5          ; 5: li $v0, 5 # ler um inteiro
[00400028] 0000000c  syscall                ; 6: syscall
[0040002c] 00022021  addu $4, $0, $2        ; 8: move $a0, $v0
[00400030] 34020001  ori $2, $0, 1          ; 9: li $v0, 1
[00400034] 0000000c  syscall                ; 10: syscall
[00400038] 3402000a  ori $2, $0, 10         ; 12: li $v0, 10 # end program
[0040003c] 0000000c  syscall                ; 13: syscall
```

Ler e imprimir um inteiro

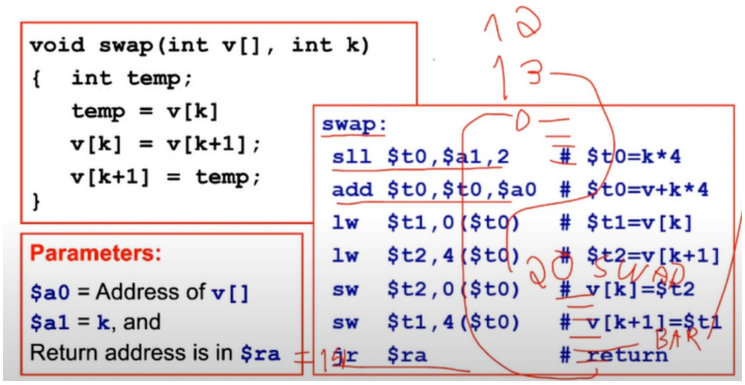
```
################## Code segment ######################
.text
.globl main
main:                          # main program entry
  li    $v0, 5                 # Read integer
  syscall                      # $v0 = value read

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

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

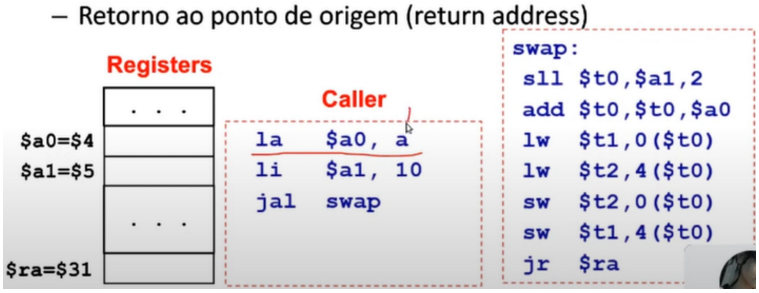
Soma de três inteiros
Procedimentos(funções)

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



Chamada de função (Call)/ Return

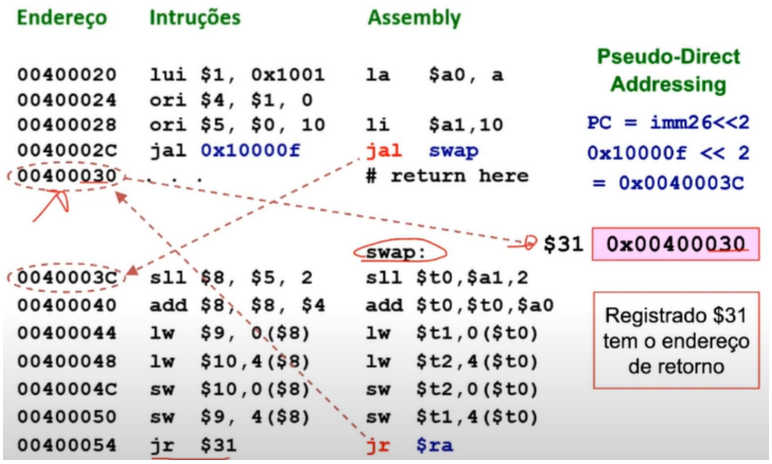
- Supondo a seguinte chamada: **swap(a,10)**
  - É passado o **address** do array **a** e **10** como argumentos
  - A chamada da função swap salvo o **return address** em **$31 = $ra**
  - Executa a função swap
  - Retorno ao ponto de origem (return address)



a é o endereço base do array

Jal é salto junto ao salvamento do $ra

JAL E JR



Jal swap: salta para o endereço de memória que contém a função swap
Executa cada uma das instruções sabendo que já salvou o endereço que deve ser retornado
(que é o com final 030, aquele após a chamada da função)
Quando realizamos chamadas recursivas em MIPS é necessário que os parâmetros e resultados da última chamada sejam armazenados.

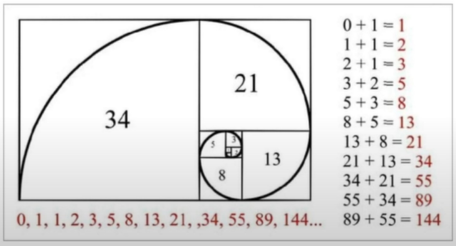
Quando trabalhamos em C, não nos preocupamos com o número de variáveis ou memória. Porém quando trabalhamos em MIPS uma hora os registradores irão acabar, pois chamadas recursivas algumas vezes não são executadas apenas 32 vezes, as vezes acontecem milhares de vezes, portanto há a necessidade de salvar esses valores na memória. Logo, no momento em que as chamadas recursivas estão desempilhando, precisamos buscar cada um desses valores na memória.

Praticando: código mips para encontrar o maior valor em um array

```
# 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
```

Código MIPS de fatorial

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




```
# 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
```