

# Organização e Arquitetura de Processadores

---

## Assembly do MIPS

### Uso da Pilha

*Principais ferramentas:*

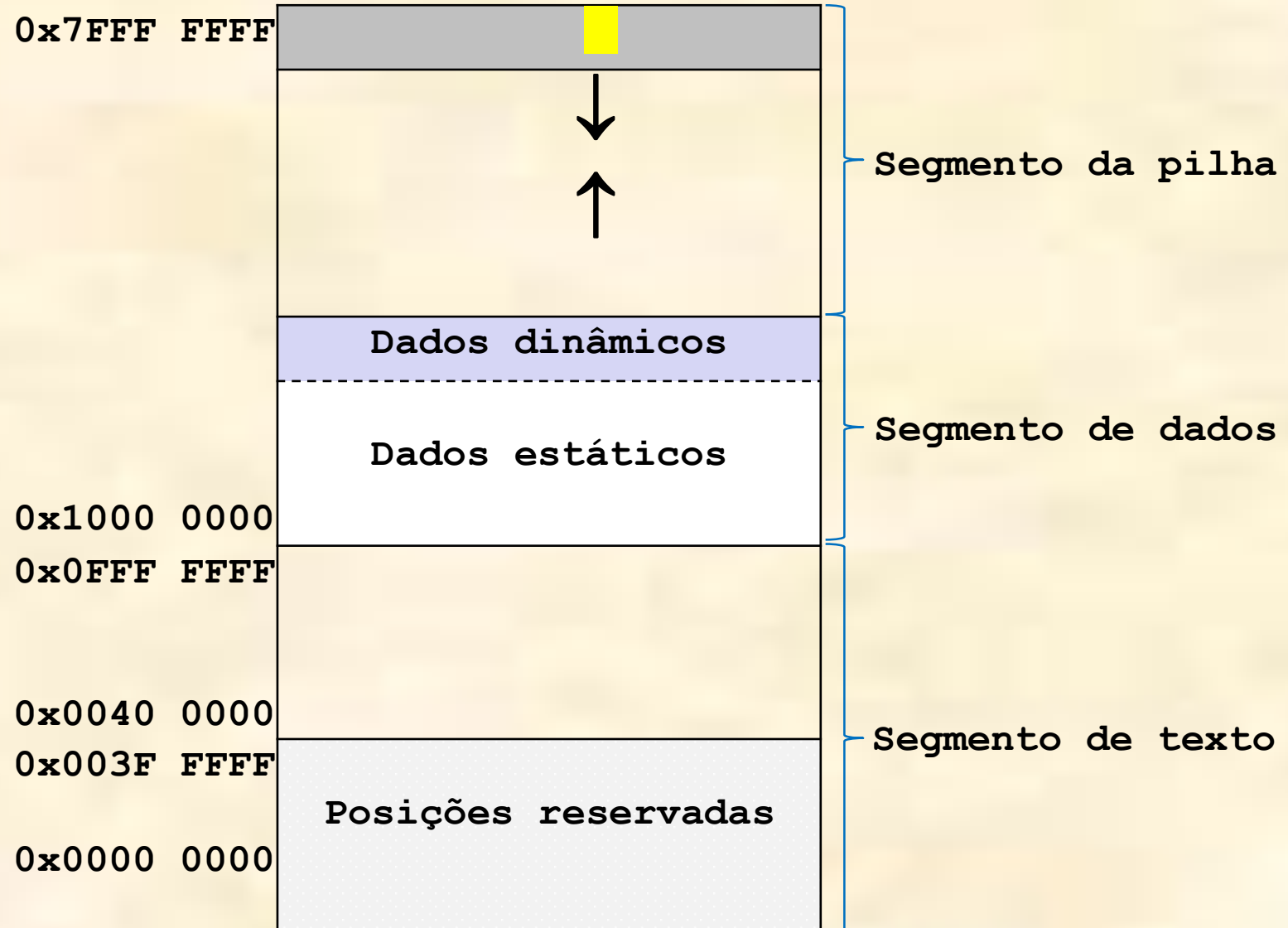
***MARS***

***Livro do Patterson e Hennessy***

***Apêndice A do livro do Patterson e Hennessy***

# Operação com a Pilha

- Pilha é uma área de memória temporária que pode ser empregada para tarefas como: passagem de parâmetros, salvamento de contexto, alocação de dados
- O gerenciamento da pilha é realizado com o auxílio do registrador \$sp
- Por convenção de software, a pilha aponta para o endereço 0x7FFFFFFF e cresce para os endereços inferiores



# Salvamento e Recuperação de Contexto com Pilha (1)

- Quando uma Função A (FA) chama uma Função B (FB), pode acontecer de FB usar parte dos registradores de FA. Desta forma, ao retornar para FA, os registradores poderão ter outro valor, implicando em erro potencial de operação
- A pilha pode ser usada para salvar o contexto de FA ao entrar em FB e recuperar este contexto ao sair de FB

```

...
outerFunction:
    ...
    li    $v0, 7
    jal   innerFunction
    move  $t1, $v0
    ...

innerFunction:
    la    $a0, String1
    li    $v0, 4
    syscall
    jr    $ra
    ...

```

(1)  $\$v0 \leftarrow 7$

(3)  $\$t1 \leftarrow 4!!!$   
**ERRO**

(2)  $\$v0 \leftarrow 4$

innerFunction:

```

    addi  $sp, $sp, -4
    sw    $v0, 0($sp)
    la    $a0, String1
    li    $v0, 4
    syscall
    lw    $v0, 0($sp)
    addi  $sp, $sp, 4
    jr    $ra
    ...

```

(1) Abre 4 bytes (1 word) na pilha

(2) Salva  $\$v0$  na pilha

(3) Recupera  $\$v0$  da pilha

(4) Esvazia a pilha

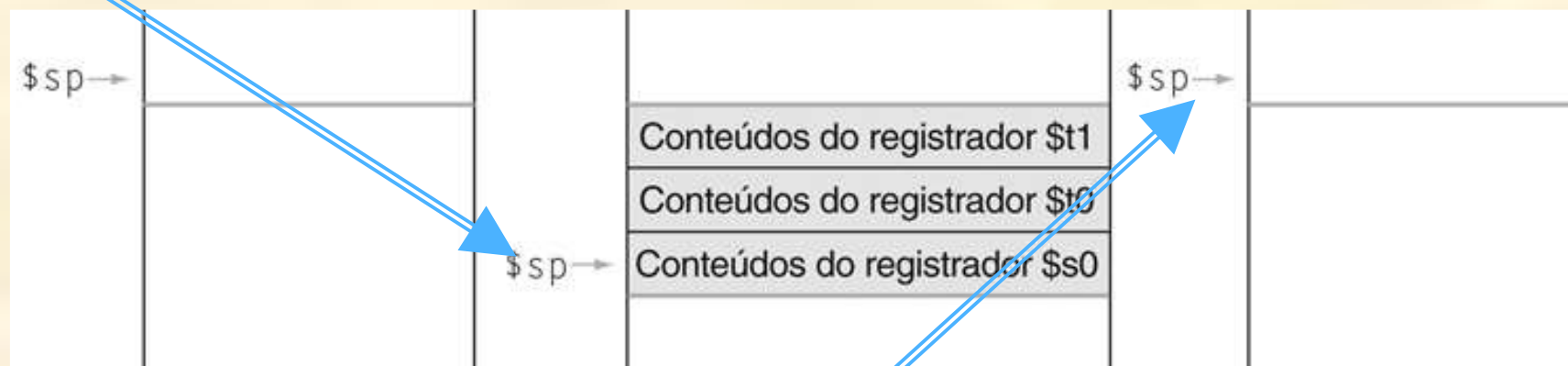
Nesta nova implementação,  
quando retornar para a  
outerFunction, o  $\$v0$  terá sido  
restaurado

# Salvamento e Recuperação de Contexto com Pilha (2)

- A operação pode envolver diversos registradores
- Exemplo com espaço para três words

```
addi $sp, $sp, -12 # ajusta pilha, criando espaço para 3 itens
sw $t1, 8($sp)    # salva reg. $t1 para usar depois
sw $t0, 4($sp)    # salva reg. $t0 para usar depois
sw $s0, 0($sp)    # salva reg. $s0 para usar depois
```

Início da função



```
lw $s0, 0($sp)    # restaura reg. $s0 para o caller
lw $t0, 4($sp)    # restaura reg. $t0 para o caller
lw $t1, 8($sp)    # restaura reg. $t1 para o caller
addi $sp, $sp, 12  # ajusta pilha para excluir 3 itens
```

Término da função

# Salvamento de Pilha com Chamada de Função

---

- **Algumas vezes é necessário salvar o contexto antes de chamar a função. Este é o caso de programas que usam aninhamento de função, devido à instrução jal**
  - A instrução jal salva o endereço de retorno em \$ra
  - Ao aninhar funções, os valores anteriores de \$ra são sobrepostos com os valores das novas chamadas
  - A solução é usar a pilha
- **Avalie o programa o programa aninhado que segue**
  - Entenda o que deve ocorrer na programação descrita
  - Analise nos próximos slides o efeito de salvar ou não o \$ra

```
void main()  
{  
    nivel1();  
    printf("N!\n");  
}
```

```
void nivel1()  
{  
    nivel2();  
    printf("N1!\n");  
}
```

```
void nivel2()  
{  
    printf("N2!\n");  
}
```

## Salvamento de Pilha com Chamada de Função (2)

```
.text
.globl main
main:      # void main() {
    jal nivel1      # nivel1();
    la $a0, strN     #
    li $v0, 4        #
    syscall         # printf("N!\n");
    li $v0, 10       #
    syscall         # }

nivel1:
    jal nivel2      # nivel2();
    la $a0, strN1    #
    li $v0, 4        #
    syscall         # printf("N1!\n");
    jr $ra

nivel2:
    la $a0, strN2    #
    li $v0, 4        #
    syscall         # printf("N2!\n");
    jr $ra
```

```
.data
strN:      .asciiz "N!\n"
strN1:     .asciiz "N1!\n"
strN2:     .asciiz "N2!\n"
```

Resultado:

N2!

N1!

N1!

N1!

N1!

... // A execução não termina

*Entenda porque acontece este comportamento e pense como resolver*

# Salvamento de Pilha com Chamada de Função (3)

```
.data
strN:      .asciiz "N!\n"
strN1:     .asciiz "N1!\n"
strN2:     .asciiz "N2!\n"

.text
    .globl main
main:      # void main() {
    jal     nivel1      # nivel1();
    la      $a0, strN    #
    li      $v0, 4        #
    syscall   # printf("N!\n");
    li      $v0, 10       #
    syscall   # }
```

```
nivel1:
    addi    $sp, $sp, -4
    sw      $ra, 0($sp)
    jal     nivel2      # nivel2();
    lw      $ra, 0($sp)
    addi    $sp, $sp, 4
    la      $a0, strN1    #
    li      $v0, 4        #
    syscall   # printf("N1!\n");
    jr      $ra

nivel2:
    la      $a0, strN2    #
    li      $v0, 4        #
    syscall   # printf("N2!\n");
    jr      $ra
```

Resultado:

N2!

N1!

N! // Resultado correto!!!

## Exercício

- Utiliza a pilha para salvar o conteúdo do registrador \$ra e permitir a chamada de uma função sem perder o retorno. *Faça um único salvamento para todo o programa!*

```
.text
.globl main
main:
    move    $t0, $zero
    move    $t1, $zero
loop: mul    $t2, $t0, $t0
    addu    $t1, $t1, $t2
    addiu   $t0, $t0, 1
    ble     $t0, 100, loop
    la      $a0, str
    jal     printString
    move    $a0, $t1
    jal     printInt
    jr      $ra
```

```
printString:
    li      $v0, 4
    syscall
    jr      $ra

printInt:
    li      $v0, 1
    syscall
    jr      $ra

.rdata
    str: .asciiz "\nSum 0..100="
```



## Resposta

- Utiliza a pilha para salvar o conteúdo do registrador \$ra e permitir a chamada de uma função sem perder o retorno. *Faça um único salvamento para todo o programa!*

```
.text
.globl main
main: subu $sp, $sp, 4 # Cria 4b na pilha
      sw  $ra, 0($sp) # Salva ra na pilha
      move $t0, $zero
      move $t1, $zero
loop: mul  $t2, $t0, $t0
      addu $t1, $t1, $t2
      addiu $t0, $t0, 1
      ble  $t0, 100, loop
      la   $a0, str
      jal  printString
      move $a0, $t1
      jal  printInt
      lw   $ra, 0($sp) # Recupera ra
      jr   $ra
```

```
printString:
      li    $v0, 4
      syscall
      jr    $ra

printInt:
      li    $v0, 1
      syscall
      jr    $ra

.rdata
      str: .asciiz "\nSum 0..100="
```

# Uso da Pilha para Passagem de Parâmetros (1)

---

- Uma forma comum de passar parâmetros para uma função é através de registradores. Contudo, o número de registradores é limitado, podendo ser necessário utilizar a pilha como recurso

```
int a=3, b=7, c=12, f;

void main() {
    f = maiorDe3(a, b, c);
    printf("Maior valor eh %d", f);
}

int maiorDe3(int a, int b, int c) {
    if(a > b && a > c)
        return a;
    if(b > c)
        return b;
    return c;
}
```

# Uso da Pilha para Passagem de Parâmetros (2)

```
.text
.globl main
main:                # void main() {
    addi    $sp, $sp, -16
    sw      $ra, 0($sp)
    lw      $t0, a
    sw      $t0, 4($sp) # maiorDe3(int a,
    lw      $t0, b
    sw      $t0, 8($sp) # maiorDe3(int a, int b
    lw      $t0, c
    sw      $t0, 12($sp) # maiorDe3(int a, int b, int c
    jal     maiorDe3    # maiorDe3(int a, int b, int c);
    sw      $v0, f      # f = maiorDe3(int a, int b, int c);
    lw      $ra, 0($sp)
    addi    $sp, $sp, 16
    la      $a0, strN    #
    li      $v0, 4       #
    syscall                # printf("Maior valor eh ")
    lw      $a0, f        #
    li      $v0, 1        #
    syscall                # printf("Maior valor eh %d", f);
    li      $v0, 10       #
    syscall                # }
```

```
maiorDe3:
    lw      $t0, 4($sp)    # $t0 = a
    lw      $t1, 8($sp)    # $t1 = b
    lw      $t2, 12($sp)   # $t2 = c
    bgt     $t0, $t1, aMb
    bgt     $t1, $t2, bMc   # b ou c maiores
cM:
    move    $v0, $t2       # c eh maior
    jr      $ra
aMb:
    bgt     $t0, $t2, aMc   # a ou c maiores
    j       cM
aMc:
    move    $v0, $t0       # a eh maior
    jr      $ra
bMc:
    move    $v0, $t1       # b eh maior
    jr      $ra

.data
a:          .word 3        # int a=3;
b:          .word 7        # int b=7;
c:          .word 12       # int c=12;
f:          .space 4       # int f;
strN:       .asciiz "Maior valor eh "
```

## Exercício

---

- **Faça a função `strlen`, que recebe o endereço de uma string e retorna o número de caracteres. Faça um programa que use duas vezes esta função. Organize o mesmo de forma modular**

## Resposta – Programa Strlen (fonte C)

---

```
int strlen(char *str) {
    int tam = 0;
    while(*str != 0) {
        tam++;
        str++;
    }
    return tam;
}


char str1[] = "Teste";
char str2[] = "Segunda string";

void main() {
    printf("Número de caracteres de %s = %d", str1, strlen(str1));
    printf("Número de caracteres de %s = %d", str2, strlen(str2));
}
```

# Resposta – Programa Strlen (1)

```
.data
str1: .asciiz "Teste"
str2: .asciiz "Segunda string"
strNumChar: .asciiz "Número de caracteres de "
strIgual: .asciiz " = "
strNewLine: .asciiz "\n"
```

*Implementados como macro no próximo slide*



```
int strlen(char *str) {
    int tam = 0;
    while(*str != 0) {
        tam++;
        str++;
    }
    return tam;
}

char str1[] = "Teste";
char str2[] = "Segunda string";

void main() {
    printf("Número de caracteres de %s = %d", str1, strlen(str1));
    printf("Número de caracteres de %s = %d", str2, strlen(str2));
}
```

```
.text
.globl main
main:
    prtStrLenInt(str1)
    prtStrLenInt(str2)
    exit()

strlen:
    lw    $s0, 0($sp)
    li    $v0, 0
laco:
    lb    $t0, 0($s0)
    beqz  $t0, return
    addi  $v0, $v0, 1
    addi  $s0, $s0, 1
    j     laco
return:
    jr    $ra
```

## Resposta – Programa Strlen (2)

```
.macro prtStr(%string)
    la    $a0, %string
    li    $v0, 4
    syscall
.end_macro

.macro prtInt(%inteiro)
    move  $a0, %inteiro
    li    $v0, 1
    syscall
.end_macro

.macro exit()
    li    $v0, 10
    syscall
.end_macro
```

```
printf("Número de caracteres de
%s = %d", str1, strlen(str1));
```

```
.macro macroStrlen(%string)
    la    $a0, %string
    addi  $sp, $sp, -8
    sw    $a0, 0($sp)
    sw    $ra, 4($sp)
    jal   strlen
    lw    $ra, 4($sp)
    lw    $a0, 0($sp)
    addi  $sp, $sp, 8
.end_macro

.macro prtStrLenInt(%string)
    prtStr(strNumChar)
    prtStr(%string)
    prtStr(strIgual)
    macroStrlen(%string)
    prtInt($v0)
    prtStr(strNewLine)
.end_macro
```