Organização e Arquitetura de Processadores

Assembly do MIPS

Recursividade

Principais ferramentas: ***MARS***
Livro do Patterson e Hennessy
Apêndice A do livro do Patterson e Hennessy

Prof. Dr. César Marcon

Recursividade – Fatorial (Versão C)

```
int val=5, result;
void main() {
    result = fatorial(val);
    printf("%d", result);
}
int fatorial(int n) {
    if(n <= 1)
       return 1;
    return n * fatorial(n - 1);
}
```

```
result   = fatorial(5)
         = 5 * fatorial(5 - 1)
         = 5 * fatorial(4)
             = 4 * fatorial(4 - 1)
             = 4 * fatorial(3)
                 = 3 * fatorial(3 - 1)
                 = 3 * fatorial(2)
                     = 2 * fatorial(2 - 1)
                     = 2 * fatorial(1)
                         = 1
                     = 2 * 1
                     = 2
                 = 3 * 2
                 = 6
             = 4 * 6
             = 24
         = 5 * 24
         = 120
```

Recursividade – Fatorial (Versão assembly)

```
int val=5, result;
void main() {
    result = fatorial(val);
    printf("%d", result);
}
int fatorial(int n) {
    if(n <= 1)
        return 1;
    return n * fatorial(n - 1);
}
```

```
.text
    .globl main             # void main()
main:                       # {
    lw      $t0, val        #   // $t0 é temp. val
    subi    $sp, $sp, 8     #   // Aloca 8b pilha
    sw      $t0, 4($sp)     #   // Coloca val pilha
    sw      $ra, 0($sp)     #   // Salva $ra pilha
    jal     fatorial        #
    sw      $v0, result     #   result=fatorial(val)
    lw      $ra, 0($sp)     #   // Recupera $ra
    addiu   $sp, $sp, 8     #   // Desaloca pilha
    move    $a0, $v0        #
    li      $v0, 1          #
    syscall                 #   printf("%d", result);
    li      $v0, 10         #
    syscall                 # }
```

```
fatorial:                   # int fatorial(int n)
                            # { // n está na pilha
    lw      $a0, 4($sp)     #    // $a0 = n
    bgt     $a0, 1, recurs  #   if(n <= 1)
    li      $v0, 1          #
    jr      $ra             #       return 1;
recurs:
    addiu   $a0, $a0, -1    #   $a0 = n - 1
    subi    $sp, $sp, 8     #   // Aloca 8 bytes na pilha
    sw      $a0, 4($sp)     #   // Salva n-1 na pilha
    sw      $ra, 0($sp)     #   // Salva o end. retorno pilha
    jal     fatorial        #   $v0 = fatorial(n - 1)
    lw      $ra, 0($sp)     #   // Recupera end. de retorno
    lw      $a0, 4($sp)     #   // Recupera n-1 na pilha
    addiu   $sp, $sp, 8     #   // Desaloca o pilha
    addiu   $a0, $a0, 1     #   // n = n-1 + 1
    mul     $v0, $v0, $a0   #   $v0 = n * fatorial(n - 1);
    jr $ra                  #   return n * fatorial(n - 1);
                            # }
```

Note que n tinha sido decrementado!

```
.data
val:        .word     5     # int val=5;
result:     .space    4     # int result;
```

Não é necessário salvar $ra, pois o main termina com exit através do syscall

Recursividade – Fatorial (Versão assembly e Execução na Pilha)

Endereço pilha	Conteúdo	Referência	Comentário
0x7FFF EFF8	5	$a0	val
0x7FFF EFF4	0x0000 0000	$ra	$ra do main
0x7FFF EFF0	4	$a0	
0x7FFF EFEC	0x0040 001C	$ra	result = fatorial(val)
0x7FFF EFE8	3	$a0	
0x7FFF EFE4	0x0040 006C	$ra	$v0 = fatorial(n - 1)
0x7FFF EFE0	2	$a0	
0x7FFF EFDC	0x0040 006C	$ra	$v0 = fatorial(n - 1)
0x7FFF EFD8	1	$a0	
0x7FFF EFD4	0x0040 006C	$ra	$v0 = fatorial(n - 1)

```
.text
    .globl main             # void main()
main:                       # {
    lw      $t0, val        #   // $t0 é temp. val
    subi    $sp, $sp, 8     #   // Aloca 8b pilha
    sw      $t0, 4($sp)     #   // Coloca val pilha
    sw      $ra, 0($sp)     #   // Salva $ra pilha
    jal     fatorial        #
    sw      $v0, result     #   result=fatorial(val)
    lw      $ra, 0($sp)     #   // Recupera $ra
    addiu   $sp, $sp, 8     #   // Desaloca pilha
    move    $a0, $v0        #
    li      $v0, 1          #
    syscall                 #   printf("%d", result);
    li      $v0, 10         #
    syscall                 # }
```

```
fatorial:                   # int fatorial(int n)
                            # { // n está na pilha
    lw      $a0, 4($sp)     #    // $a0 = n
    bgt     $a0, 1, recurs  #   if(n <= 1)
    li      $v0, 1          #
    jr      $ra             #       return 1;
recurs:
    addiu   $a0, $a0, -1    #   $a0 = n - 1
    subi    $sp, $sp, 8     #   // Aloca 8 bytes na pilha
    sw      $a0, 4($sp)     #   // Salva n-1 na pilha
    sw      $ra, 0($sp)     #   // Salva o end. retorno pilha
    jal     fatorial        #   $v0 = fatorial(n - 1)
    lw      $ra, 0($sp)     #   // Recupera end. de retorno
    lw      $a0, 4($sp)     #   // Recupera n-1 na pilha
    addiu   $sp, $sp, 8     #   // Desaloca o pilha
    addiu   $a0, $a0, 1     #   // n = n-1 + 1
    mul     $v0, $v0, $a0   #   $v0 = n * fatorial(n - 1);
    jr $ra                  #   return n * fatorial(n - 1);
                            # }

.data
val:        .word     5     # int val=5;
result:     .space    4     # int result;
```

Recursividade – Fatorial – Versão estruturada com macros (1)

```
int val, res;

void main() {
    printf("Digite número para fatorial:");
    val = getchar();
    res = fatorial(val);
    printf("\nFatorial(%d) = %d", val, res);
}
```

```
int fatorial(int n) {
    if(n <= 1)
        return 1;
    return n * fatorial(n - 1);
}
```

```
.data
val:       .space    4          #   int val;
res:       .space    4          #   int res;
strDig:    .asciiz "Digite número para calcular fatorial:"
strFat:    .asciiz "\nFatorial("
strIgual:  .asciiz ") = "

.text
    .globl main       #
main:                 # void main() {
    prtStr(strDig)    #   printf("Digite número para fatorial:")
    readInt($a0)      #
    sw      $a0, val  #   val = $a0 = getchar();
    callFatorial()    #
    sw      $v0, res  #   res = $v0 = fatorial(val);
    prtStr(strFat)    #   printf("\nFatorial(
    prtInt(val)       #   printf("\nFatorial(%d
    prtStr(strIgual)  #   printf("\nFatorial(%d) =
    prtInt(res)       #   printf("\nFatorial(%d) = %d", val, res);
    exit()            # }
```

Recursividade – Fatorial – Versão estruturada com macros (2)

```
.text
    .globl main         #
main:                   # void main() {
    prtStr(strDig)      #   printf("Digite número para fatorial:")
    readInt($a0)        #
    sw      $a0, val    #   val = $a0 = getchar();
    callFatorial()      #
    sw      $v0, res    #   res = $v0 = fatorial(val);
    prtStr(strFat)      #   printf("\nFatorial(
    prtInt(val)         #   printf("\nFatorial(%d
    prtStr(strIgual)    #   printf("\nFatorial(%d) =
    prtInt(res)         #   printf("\nFatorial(%d) = %d", val, res);
    exit()              # }
```

```
.macro prtStr(%string)
            addi    $sp, $sp, -8
            sw      $v0, 0($sp)
            sw      $a0, 4($sp)
    la      $a0, %string
    li      $v0, 4
    syscall
            lw      $a0, 4($sp)
            lw      $v0, 0($sp)
            addi    $sp, $sp, 8
.end_macro
```

```
.macro readInt(%inteiro)
            addi    $sp, $sp, -4
            sw      $v0, 0($sp)
    li      $v0, 5
    syscall
    move    %inteiro, $v0
            lw      $v0, 0($sp)
            addi    $sp, $sp, 4
.end_macro

.macro prtInt(%inteiro)
            addi    $sp, $sp, -8
            sw      $v0, 0($sp)
            sw      $a0, 4($sp)
    lw      $a0, %inteiro
    li      $v0, 1
    syscall
            lw      $a0, 4($sp)
            lw      $v0, 0($sp)
            addi    $sp, $sp, 8
.end_macro

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

Recursividade – Fatorial – Versão estruturada com macros (3)

```
.text
    .globl main         #
main:                   # void main() {
    prtStr(strDig)      #   printf("Digite número para fatorial:")
    readInt($a0)        #
    sw      $a0, val    #   val = $a0 = getchar();
    callFatorial()      #
    sw      $v0, res    #   res = $v0 = fatorial(val);
    prtStr(strFat)      #   printf("\nFatorial(
    prtInt(val)         #   printf("\nFatorial(%d
    prtStr(strIgual)    #   printf("\nFatorial(%d) =
    prtInt(res)         #   printf("\nFatorial(%d) = %d", val, res);
    exit()              # }

fatorial:                       # int fatorial(int n) {
    lw      $a0, 4($sp)         #   // $a0 = n
    bgt     $a0, 1, recurs      #  if(n <= 1)
    li      $v0, 1              #
    jr      $ra                 #      return 1;
recurs:
    addiu   $a0, $a0, -1        #   $a0 = n - 1
    callFatorial()              #
    addiu   $a0, $a0, 1         #   // n = n-1 + 1
    mul     $v0, $v0, $a0       #   $v0 = n * fatorial(n - 1);
    jr  $ra                     #   return n * fatorial(n - 1);
                                # }
```

```
.macro callFatorial()
            addi    $sp, $sp, -8
            sw      $a0, 4($sp)
            sw      $ra, 0($sp)
    jal     fatorial
            lw      $ra, 0($sp)
            lw      $a0, 4($sp)
            addi    $sp, $sp, 8
.end_macro
```

Exercício – Implementar Somatório de Inteiros Positivos Recursivo

```
int val=5, result;
void main() {
        result = somatorio(val);
        printf("%d", result);
}
int somatorio(int val) {
        if(val == 0)
                return 0;
        return val + somatorio(val-1);
}
```

Resposta - Somatório de Inteiros Positivos

```
int val=5, result;
void main() {
        result = somatorio(val);
        printf("%d", result);
}
int somatorio(int val) {
        if(val == 0)
                return 0;
        return val + somatorio(val-1);
}
```

```
.text
    .globl main             # void main()
main:                       # {
    subi    $sp, $sp, 8     #   // Aloca pilha
    lw      $t0, val        #   // $t0 = val
    sw      $t0, 4($sp)     #   // pilha recebe val
    sw      $ra, 0($sp)     #   // Salva $ra na pilha
    jal     somatorio       #
    sw      $v0, result     # result=somatorio(val);
    lw      $ra, 0($sp)     #   // Recupera $ra
    addiu   $sp, $sp, 8     #   // Desaloca pilha
    move    $a0, $v0        #
    li      $v0, 1          #
    syscall                 #   printf("%d", result);
    li      $v0, 10         #
    syscall                 # }
```

```
somatorio:                  # int somatorio(int val)
                            # {
    lw      $a0, 4($sp)     #   // $a0 = val
    bne     $a0, 0, recurs  #   if(val == 0)
    li      $v0, 0          #
    jr      $ra             #       return 0;
recurs:
    addiu   $a0, $a0, -1    #   $a0 = val - 1
    subi    $sp, $sp, 8     #   // Aloca 8 bytes na pilha
    sw      $a0, 4($sp)     #   // Salva val-1 na pilha
    sw      $ra, 0($sp)     #   // Salva retorno na pilha
    jal     somatorio       #   $v0 = somatorio(val - 1)
    lw      $ra, 0($sp)     #   // Recupera retorno
    lw      $a0, 4($sp)     #   // Recupera val-1 na pilha
    addiu   $sp, $sp, 8     #   // Desaloca pilha
    addiu   $a0, $a0, 1     #   // val = val-1 + 1
    add     $v0, $v0, $a0   #   $v0 = val + somatorio(val-1);
    jr  $ra                 #   return val+somatorio(val-1);
                            # }

.data
val:        .word     5     # int val=5;
result:     .space    4     # int result;
```

Exercício – Algoritmo Máximo Valor de um Vetor Recursivo

```
int maximo(int n, int *vet) {
    if(n == 1)
        return vet[0];
    int max = maximo(n-1, vet);
    if(max > vet[n-1])
        return max;
    return vet[n-1];
}

int vet[] = { 3, 5, 8, 12, -4, 3, 6, 1 };
int n = 8; // Número de elementos de vet

void main() {
    printf("%d", maximo(n, &vet[0]));
}
```

```c
int maximo(int n, int *vet) {
    if(n == 1)
        return vet[0];
    int max = maximo(n-1, vet);
    if(max > vet[n-1])
        return max;
    return vet[n-1];
}
int vet[] = { 3, 5, 8, 12, -4, 3, 6, 1 };
```

```c
int n = 8; // Número de elementos de vet
void main() {
    printf("%d", maximo(n, &vet[0]));
}
```

```
.data
vet: .word 3,5,8,12,-4,3,6,1 # int vet[]={3,5...
n:   .word 8                  # int n = 8;

.text
    .globl main           # void main()
main:                     # {
    addi   $sp, $sp, -12  #   // Aloca pilha
    lw     $t0, n         #
    sw     $t0, 8($sp)    #   // Coloca n na pilha
    la     $t0, vet       #
    sw     $t0, 4($sp)    #   // Coloca &vet pilha
    sw     $ra, 0($sp)    #   // Coloca $ra pilha
    jal    maximo         #   maximo(n, vet)
    lw     $ra, 0($sp)    #   // Recupera $ra
    addi   $sp, $sp, 12   #   // Libera pilha
    move   $a0, $v0       #
    li     $v0, 1         #
    syscall               #   printf("%d",maximo...
    li     $v0, 10        #
    syscall               # }
```

```
maximo:                       # int maximo(int n, int *vet) {
    lw     $a0, 8($sp)        #   // $a0 = n
    lw     $a1, 4($sp)        #   // $a1 = &vet[0]
    bne    $a0, 1, recurs     #   if(n == 1)
    lw     $v0, 0($a1)        #
    jr     $ra                #       return vet[0];
recurs:
    addiu  $a0, $a0, -1       #   $a0 = n - 1
    addi   $sp, $sp, -12      #   // Aloca 12 bytes na pilha
    sw     $a0, 8($sp)        #   // Salva n-1 na pilha
    sw     $a1, 4($sp)        #   // Coloca &vet[0] na pilha
    sw     $ra, 0($sp)        #   // Salva retorno na pilha
    jal    maximo             #   $v0 = maximo(n-1, vet);
    lw     $ra, 0($sp)        #   // Recupera retorno da pilha
    lw     $a1, 4($sp)        #   // Recupera &vet[0] da pilha
    lw     $a0, 8($sp)        #   // Recupera n-1 da pilha
    addi   $sp, $sp, 12       #   // Libera 12 bytes da pilha
    sll    $a0, $a0, 2        #    // $a0 = $a0 * 4
    add    $t0, $a0, $a1      #   // $t0 = &vet[n-1]
    lw     $t0, 0($t0)        #   // $t0 = vet[n-1]
    ble    $v0, $t0, rVn_1    #   if(max > vet[n-1])
    jr     $ra                #       return max;
rVn_1:
    move   $v0, $t0           #   // $v0 = vet[n-1]
    jr     $ra                #   return vet[n-1];
```