

Funções recursivas

ASC I 2015/2016

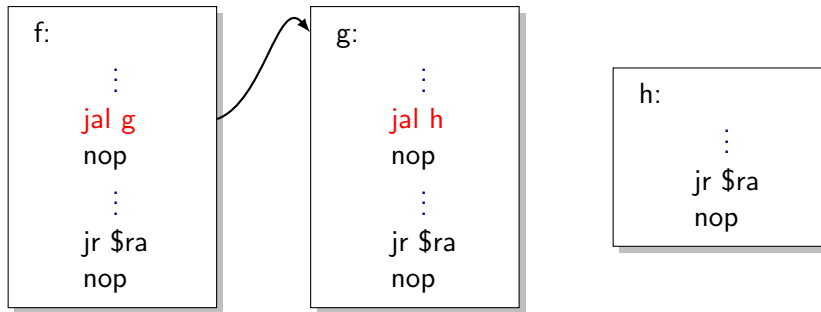
Teresa Gonçalves



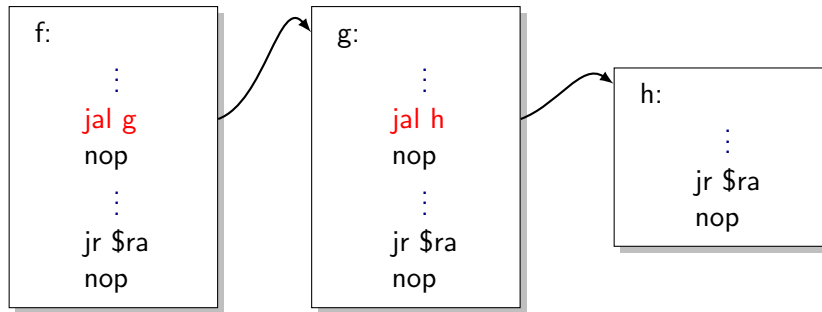
Resumo

- Funções:
 - ▶ Problema de chamadas encadeadas a funções
 - ▶ Registo \$ra: *return address*
 - ▶ Funções recursivas
 - ▶ Espaço de endereçamento (user mode)
 - ▶ Global Pointer
 - ▶ Frame Pointer

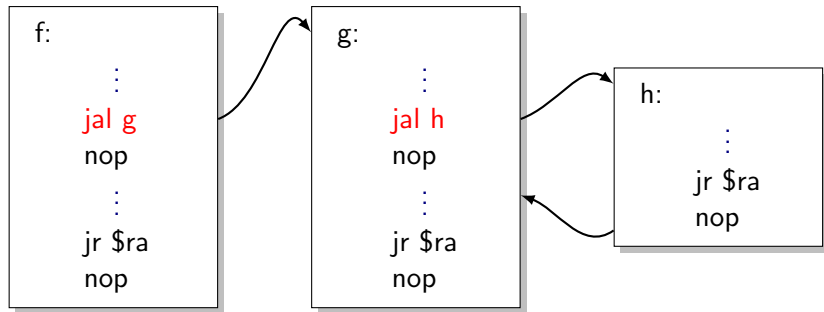
Chamadas encadeadas de funções



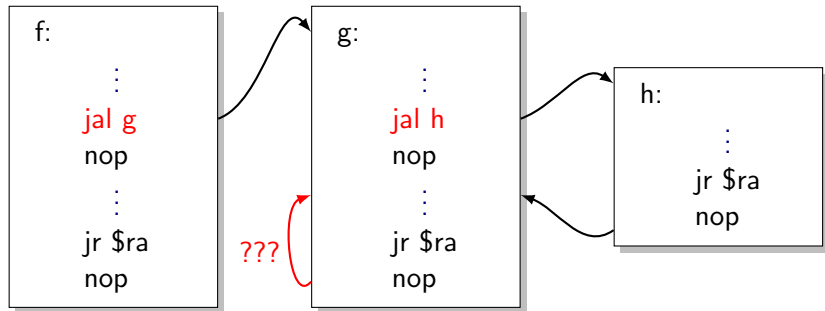
Chamadas encadeadas de funções



Chamadas encadeadas de funções

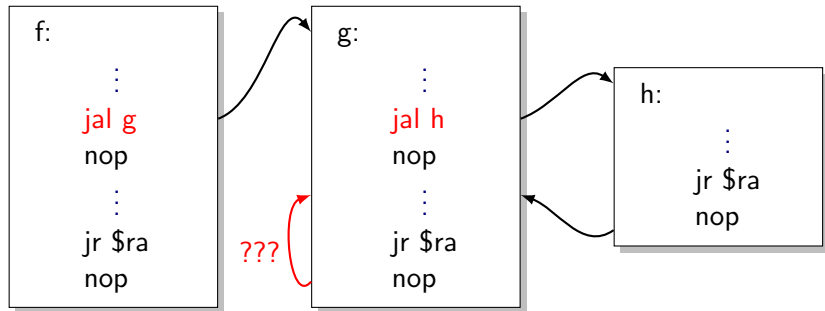


Chamadas encadeadas de funções



Como se resolve?

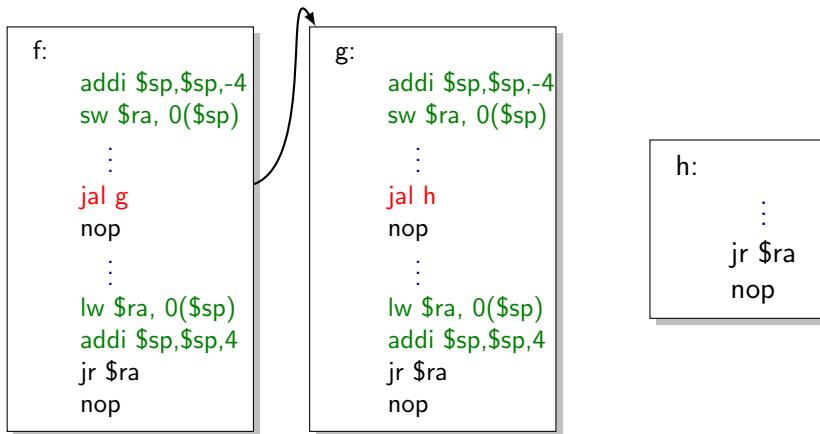
Chamadas encadeadas de funções



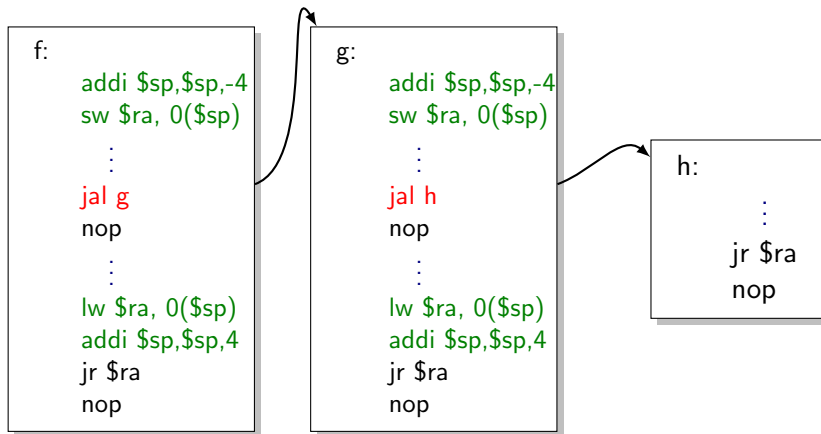
Como se resolve?

R: A função g deve guardar \$ra na pilha antes no início da execução. Assim, poderá recuperar o *return address* correcto no final. O mesmo se passa com a função f.

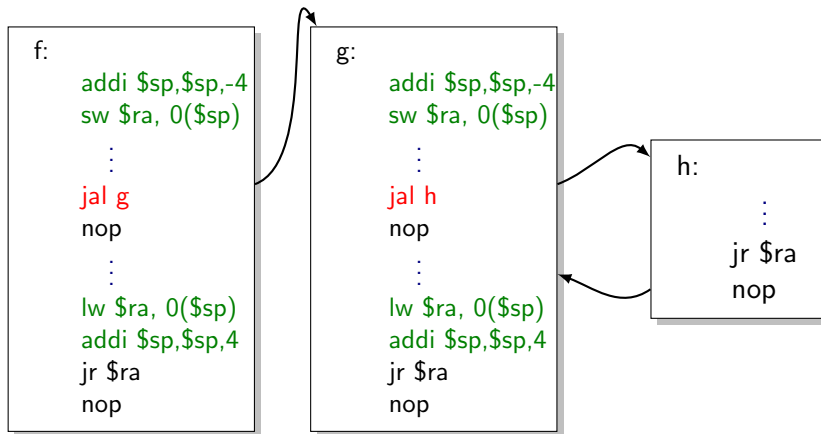
Chamadas encadeadas de funções



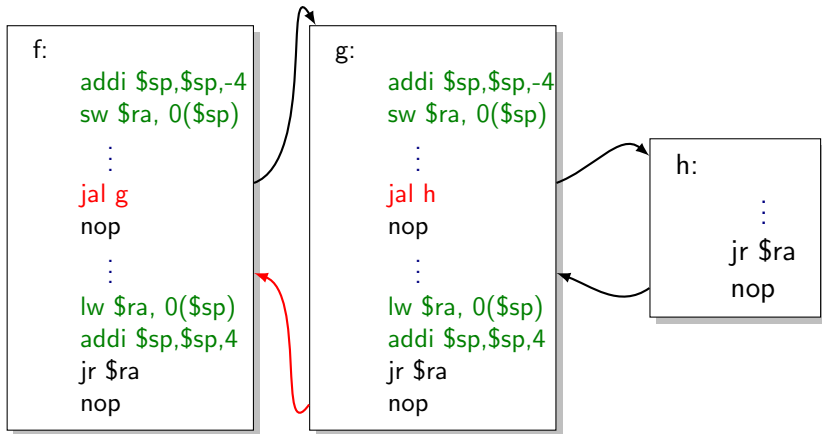
Chamadas encadeadas de funções



Chamadas encadeadas de funções



Chamadas encadeadas de funções



Definição (Recursividade)

É um método em que a solução para um problema envolve a solução de instâncias mais pequenas do mesmo problema. Normalmente, a recursividade é implementada por uma função (*função recursiva*) que se chama a ela própria.

```
1 int xpto(int a)
2 {
3     if (a == 0)
4         return 1;
5     else
6         return 2*xpto(a-1);
7 }
```

O que faz esta função?

Funções recursivas: implementação em assembly

```
1 int xpto(int a)
2 {
3     if (a == 0)
4         return 1;
5     else
6         return 2*xpto(a-1);
7 }
```

Funções recursivas: implementação em assembly

```
1 int xpto(int a)
2 {
3     if (a == 0)
4         return 1;
5     else
6         return 2*xpto(a-1);
7 }
```

xpto:

```
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    li $v0, 1
    beq $a0, $zero, RETURN
    nop
```

```
    addi $a0, $a0, -1
    jal xpto
    nop          # v0=xpto(a0-1)
    sll $v0, $v0, 1
```

RETURN:

```
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra
    nop
```

Exemplo: xpto(3)

`$sp = 0x7fffffff`

xpto:

```
addi $sp, $sp, -4
sw $ra, 0($sp)
li $v0, 1
beq $a0, $zero, RETURN
nop

addi $a0, $a0, -1
jal xpto
nop      # v0=xpto(a0-1)
sll $v0, $v0, 1
```

RETURN:

```
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
nop
```

`$sp`



`0x????????`

Espaço Livre

Exemplo: xpto(3)

`$sp = 0x7fffffff8`

xpto:

```
addi $sp, $sp, -4
sw $ra, 0($sp)
li $v0, 1
beq $a0, $zero, RETURN
nop

addi $a0, $a0, -1
jal xpto
nop      # v0=xpto(a0-1)
sll $v0, $v0, 1
```

RETURN:

```
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
nop
```

`$sp`



0x????????

ra

Espaço Livre

Exemplo: xpto(3)

`$sp = 0x7fffffff4`

xpto:

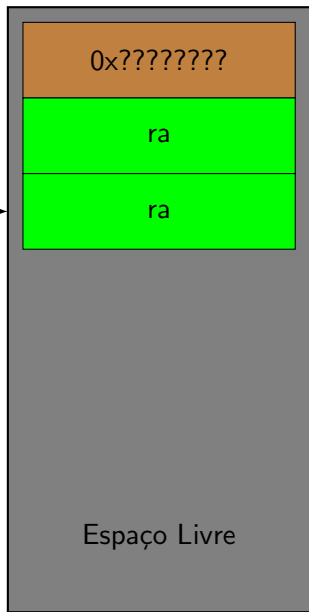
```
addi $sp, $sp, -4
sw $ra, 0($sp)
li $v0, 1
beq $a0, $zero, RETURN
nop

addi $a0, $a0, -1
jal xpto
nop      # v0=xpto(a0-1)
sll $v0, $v0, 1
```

RETURN:

```
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
nop
```

`$sp`



Exemplo: xpto(3)

\$sp = 0x7fffffff0

xpto:

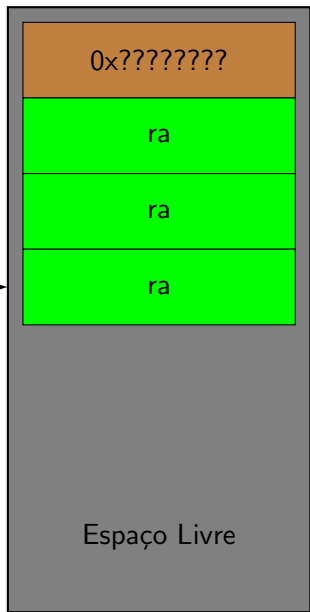
```
addi $sp, $sp, -4
sw $ra, 0($sp)
li $v0, 1
beq $a0, $zero, RETURN
nop

addi $a0, $a0, -1
jal xpto
nop      # v0=xpto(a0-1)
sll $v0, $v0, 1
```

RETURN:

```
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
nop
```

\$sp



Exemplo: xpto(3)

\$sp = 0x7ffffffc

xpto:

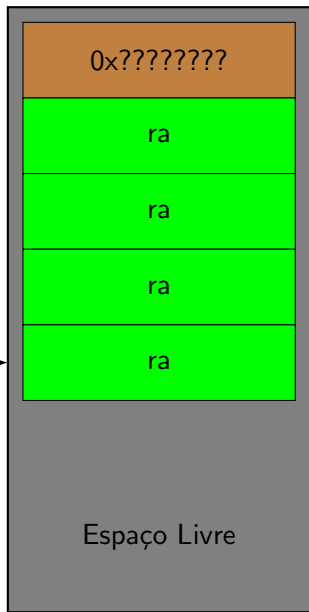
```
addi $sp, $sp, -4
sw $ra, 0($sp)
li $v0, 1
beq $a0, $zero, RETURN
nop

addi $a0, $a0, -1
jal xpto
nop      # v0=xpto(a0-1)
sll $v0, $v0, 1
```

RETURN:

```
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
nop
```

\$sp



Exemplo: xpto(3)

\$sp = 0x7ffffffc

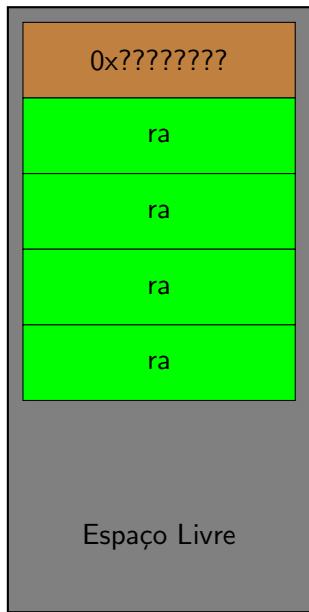
xpto:

```
addi $sp, $sp, -4
sw $ra, 0($sp)
li $v0, 1
beq $a0, $zero, RETURN
nop

addi $a0, $a0, -1
jal xpto
nop      # v0=xpto(a0-1)
sll $v0, $v0, 1
```

RETURN:

```
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
nop
```



Exemplo: somatório de 1 a n

Pretende-se implementar $s(n) \triangleq \sum_{i=0}^n i = 0 + 1 + 2 + 3 + \dots + n$.

```
1  s:  addi $sp, $sp, -8
2      sw $ra, 0($sp)
3      sw $s0, 4($sp)
4
5      li $v0, 0
6      beq $a0, $zero, RETURN
7      nop
8      move $s0, $a0
9      addi $a0, $a0, -1
10     jal s
11     nop
12     add $v0, $v0, $s0
13
14 RETURN:
15     lw $ra, 0($sp)
16     lw $s0, 4($sp)
17     addi $sp, $sp, 8
18     jr $ra
19     nop
```

As variáveis locais de uma função podem ser armazenadas em dois sítios:

- registos (e.g. variáveis do tipo int ou char)
- pilha (para todos os tipos: int, char, arrays, estruturas, ...)

As variáveis locais de uma função podem ser armazenadas em dois sítios:

- registos (e.g. variáveis do tipo int ou char)
- pilha (para todos os tipos: int, char, arrays, estruturas, ...)

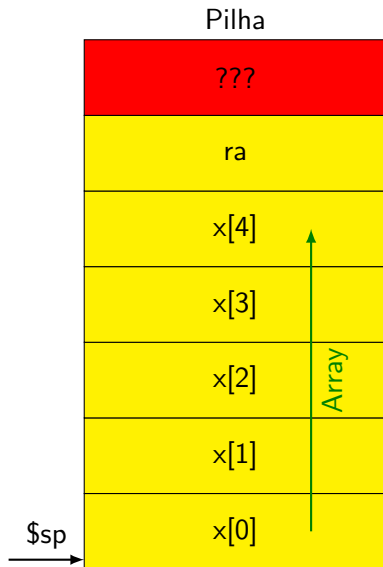
Porquê?

- Arrays, pela sua natureza, só podem residir em memória RAM.
- Para os inteiros depende do que dá mais “jeito” e da quantidade de registos disponíveis.

```
int f(int n)
{
    int a;
    int x[5];

    a = g(n);
    ...
}
```

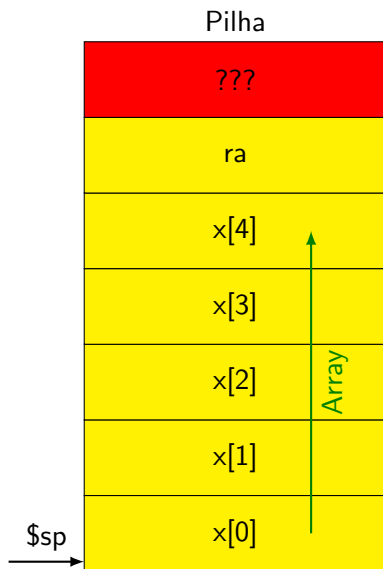
- Array `x[]` é alocado na pilha.



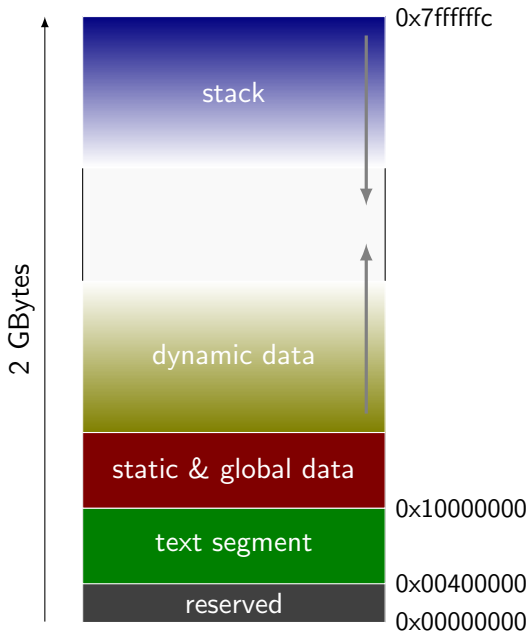

```
int f(int n)
{
    int a;
    int x[5];

    a = g(n);
    ...
}
```

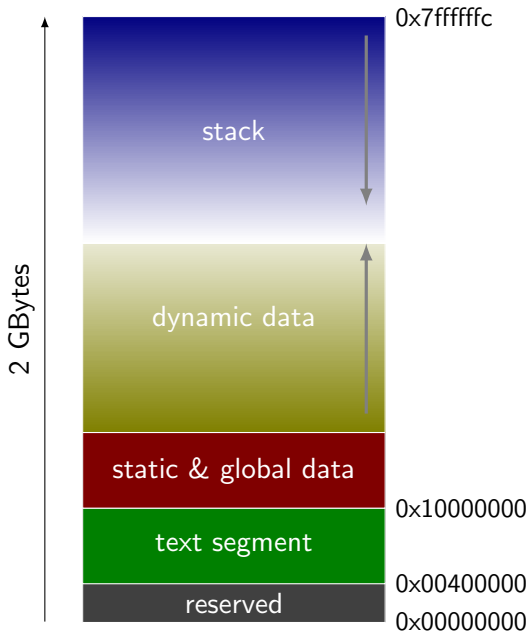
- Array `x[]` é alocado na pilha.
- Para obter `x[2]` faz-se
`lw $t0, 8($sp)`



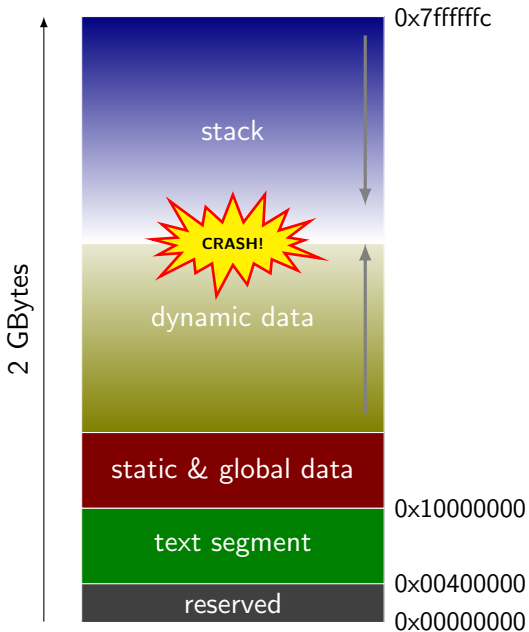
Espaço de endereçamento



Espaço de endereçamento



Espaço de endereçamento



- A *static & global data segment* é uma região de memória de tamanho 64kBytes, com início no endereço 0x10000000.
- Para facilitar o acesso a esta região, é usado o registo \$gp (Global Pointer).
- O registo \$gp aponta para o endereço 0x10008000.
- Este endereço fica a meio da região, com 32 kB para baixo e para cima.
- É usado para aceder a dados desde

```
lw $t0, -32768($gp)
```

até

```
lw $t0, 32764($gp)
```