Arquitectura de Sistemas e Computadores I

Semana 5

Miguel Barão

mjsb@di.uevora.pt

ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Resumo

- Pseudoinstruções. O registo `$at`.
- Funções:
  - Chamada de funções (instruções `jal` e `jr`)
  - Passagem de argumentos por registos (`$a0`-`$a3`)
  - Valor retornado pela função (`$v0`-`$v1`)
  - Problema de chamadas encadeadas a funções.
  - Introdução à pilha (stack)

Pseudoinstruções

- Permitem facilitar um pouco a programação.
- O assembler substitui as pseudoinstruções por instruções reais MIPS.
- Usa o registo `$at` (*assembler temporary*) como auxiliar.

Alguns exemplos:

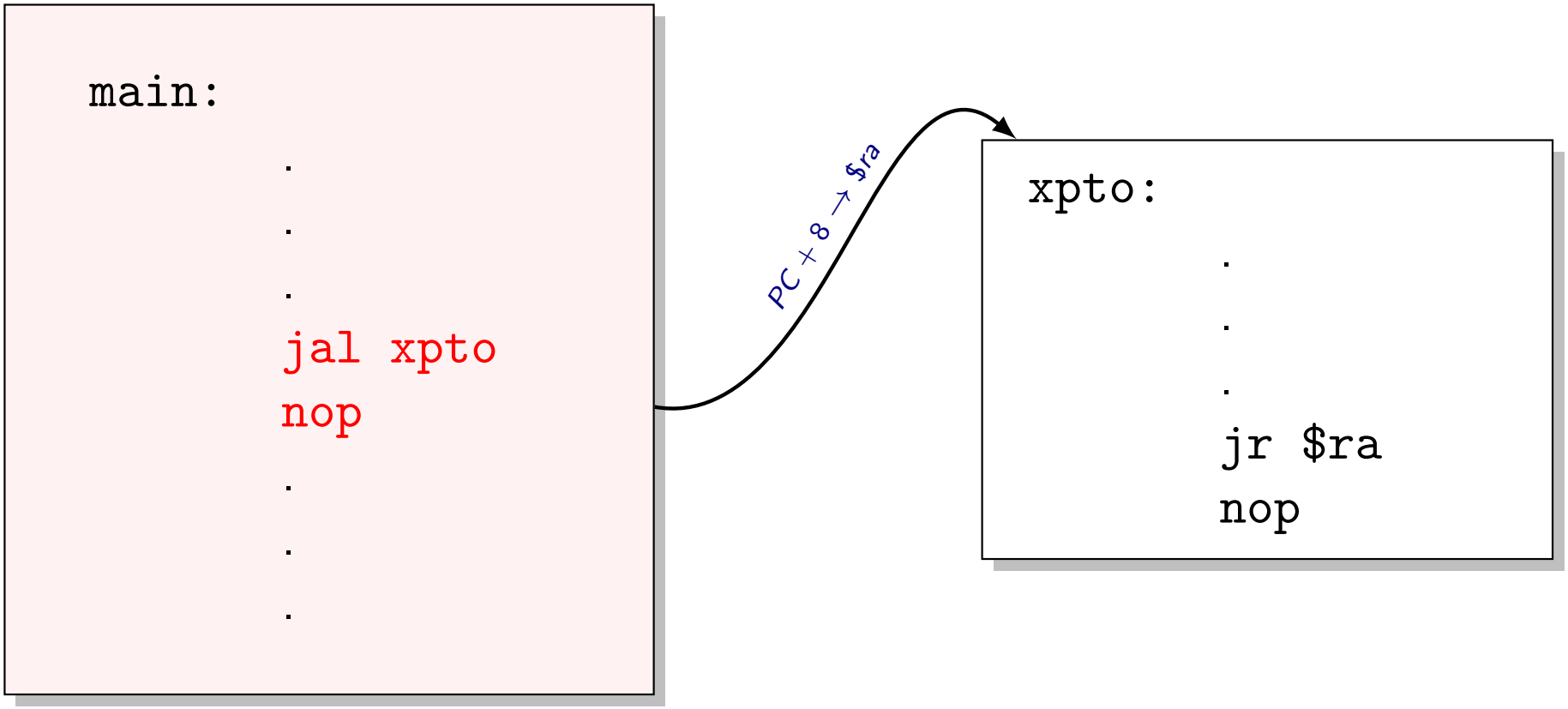
Pseudoinstruções:	Instruções reais:
`blt $t0, $t1, L`	`slt $at, $t0, $t1` `bne $at, $zero, L`
`li $t0, 0x12345678`	`lui $t0, 0x1234` `ori $t0, $t0, 0x5678`
`la $t0, LABEL`	`lui $t0, 0x????` `ori $t0, $t0, 0x????`
`move $t0, $t1`	`or $t0, $t1, $zero`

Pseudoinstruções - Exemplo

```
        .data
        # data segment
A:      .word 3,1,-2,5,0,17
B:      .asciiz "Hello!"

        .text
        # text segment
main:
        la $t0, A


        lw $t1, 0($t0)
        .
        .
        .
```

```
        .data
        # data segment
A:      .word 3,1,-2,5,0,17
B:      .asciiz "Hello!"

        .text
        # text segment
main:
        lui $t0, 0x1001
        ori $t0, $t0, 0x0014

        lw $t1, 0($t0)
        .
        .
        .
```

Chamada de funções



- Função `main` chama a função `xpto` (return address é guardado no registo `$ra`).

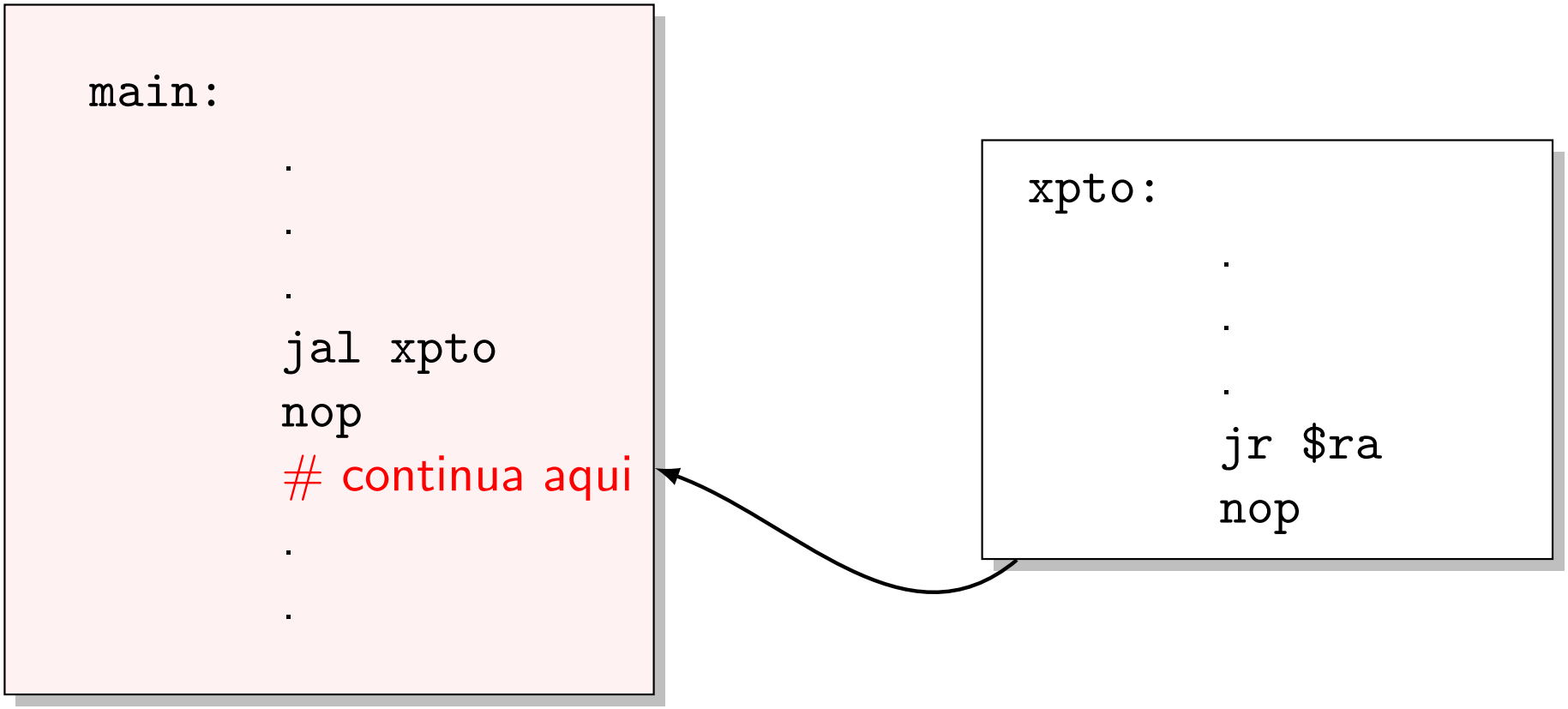
Chamada de funções

```
main:
        .
        .
        .
        jal xpto
        nop
        .
        .
        .
```

```
xpto:
        .
        .
        .
        jr $ra
        nop
```

- Função `main` chama a função `xpto` (return address é guardado no registo `$ra`).
- Função `xpto` em execução.

Chamada de funções



- Função `main` chama a função `xpto` (return address é guardado no registo `$ra`).
- Função `xpto` em execução.
- Função `xpto` termina e retorna à função inicial.

Passagem de argumentos

Os quatro primeiros argumentos são passados pelos quatro registos:

`$a0 -- $a3`

Passagem de argumentos

Os quatro primeiros argumentos são passados pelos quatro registos:

`$a0 -- $a3`

- Se uma função recebe um argumento, este é passado em `$a0`.
- Se uma função recebe dois argumentos, estes são passados em `$a0` e `$a1`.
- Se uma função recebe três argumentos, estes são passados em `$a0`, `$a1` e `$a2`.
- Se uma função recebe quatro argumentos, estes são passados em `$a0`, `$a1`, `$a2` e `$a3`.
- Se uma função recebe cinco argumentos, ...

Passagem de argumentos

Os quatro primeiros argumentos são passados pelos quatro registos:

`$a0 -- $a3`

- Se uma função recebe um argumento, este é passado em `$a0`.
- Se uma função recebe dois argumentos, estes são passados em `$a0` e `$a1`.
- Se uma função recebe três argumentos, estes são passados em `$a0`, `$a1` e `$a2`.
- Se uma função recebe quatro argumentos, estes são passados em `$a0`, `$a1`, `$a2` e `$a3`.
- Se uma função recebe cinco argumentos, ...não os consegue passar todos exclusivamente usando registos! como fazer nesse caso??

Valor de retorno da função

Cada função retorna no máximo um valor nos registos:

`$v0 -- $v1`

Valor de retorno da função

Cada função retorna no máximo um valor nos registos:

`$v0 -- $v1`

- Se retorna um valor de 32 bits, usa `$v0`.
- Se retorna um valor de 64 bits, usa `$v0` e `$v1`.
  O registo `$v1` é raramente usado.
- Se não retorna nada (*void*), então `$v0` e `$v1` são ignorados.

Exemplo de chamada a uma função

Pretende-se calcular

$$t1 = 2\ \mathrm{funcao}(t0)$$

onde funcao() é uma função existente ...

```
move $a0, $t0          # copia t0 -> a0 (argumento)
jal funcao
nop                    # v0 contem resultado
sll $t1, $v0, 1        # multiplica por 2
```

Exemplo de chamada a uma função

Pretende-se calcular

$$t1 = \text{funcao}(t0) + t0$$

```
move $a0, $t0          # copia t0 -> a0 (argumento)
jal funcao
nop                    # v0 contem resultado
add $t1, $v0, $t0      # t1 = funcao(t0) + t0 ?
```

Exemplo de chamada a uma função

Pretende-se calcular

$$t1 = \text{funcao}(t0) + t0$$

```
move $a0, $t0          # copia t0 -> a0 (argumento)
jal funcao
nop                    # v0 contem resultado
add $t1, $v0, $t0      # t1 = funcao(t0) + t0 ?
```

E se...

... a execução de `funcao` alterar o valor do registo `$t0`?

Exemplo de chamada a uma função

Pretende-se calcular

$$t1 = \text{funcao}(t0) + t0$$

```
move $a0, $t0          # copia t0 -> a0 (argumento)
jal funcao
nop                    # v0 contem resultado
add $t1, $v0, $t0      # t1 = funcao(t0) + t0 ?
```

E se...

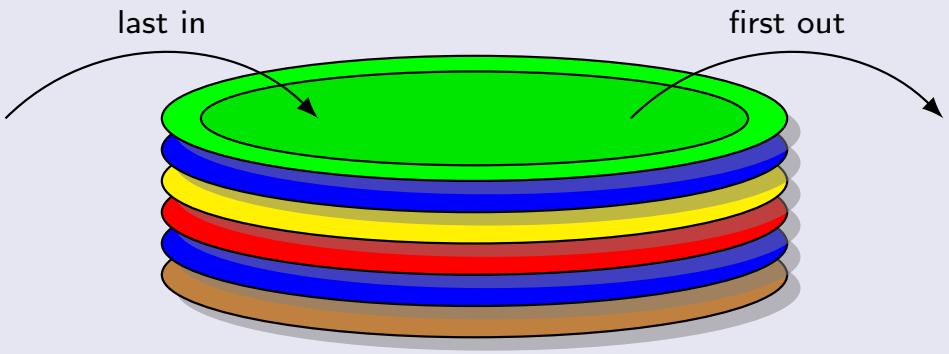
... a execução de `funcao` alterar o valor do registo `$t0`?

Nesse caso já não estamos a somar o valor original de `$t0`!

Definição (Pilha ou Stack)

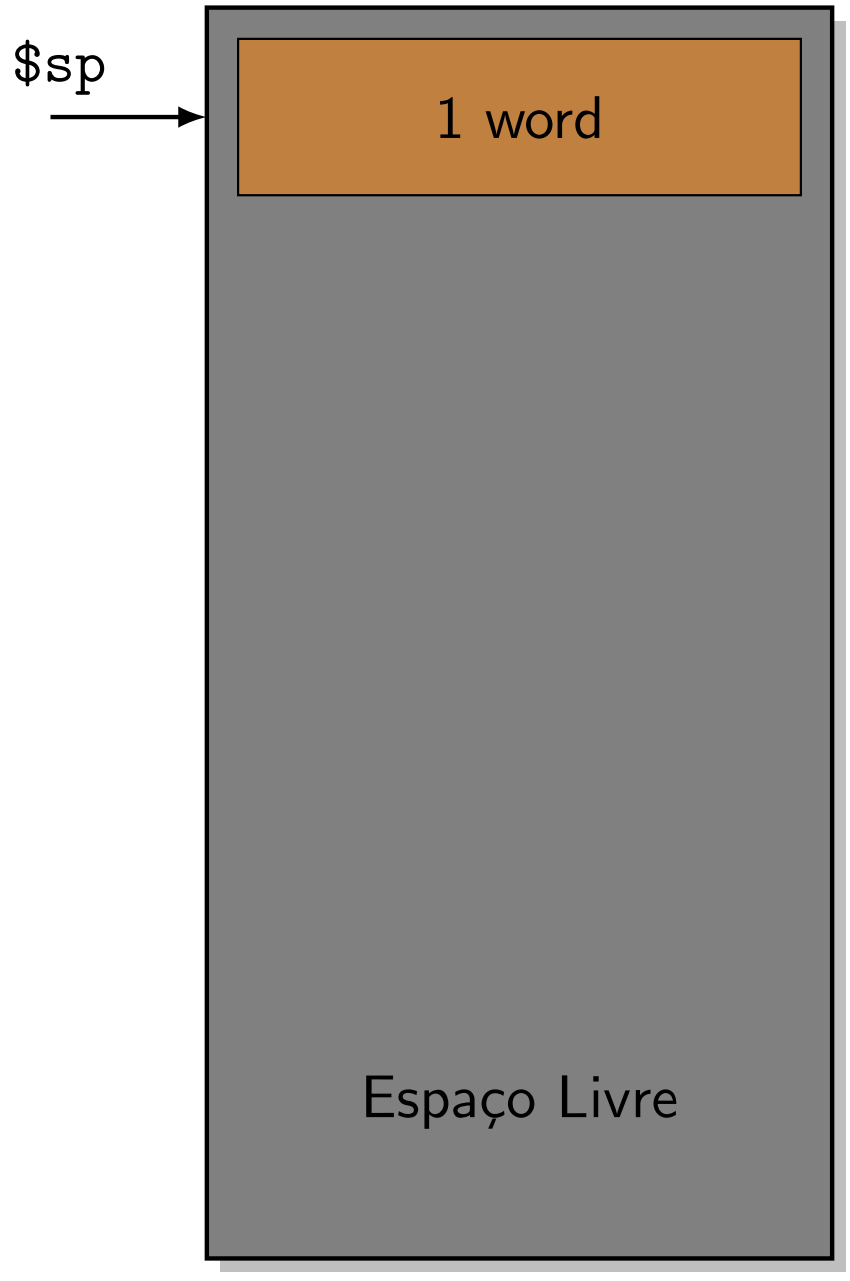
É uma estrutura do tipo LIFO (Last In First Out).

Imagine uma pilha de pratos. O primeiro a sair é o último a ter sido colocado na pilha:



Implementação da pilha em memória (Stack Pointer $sp)

```
$sp = 0x7ffffffc
```

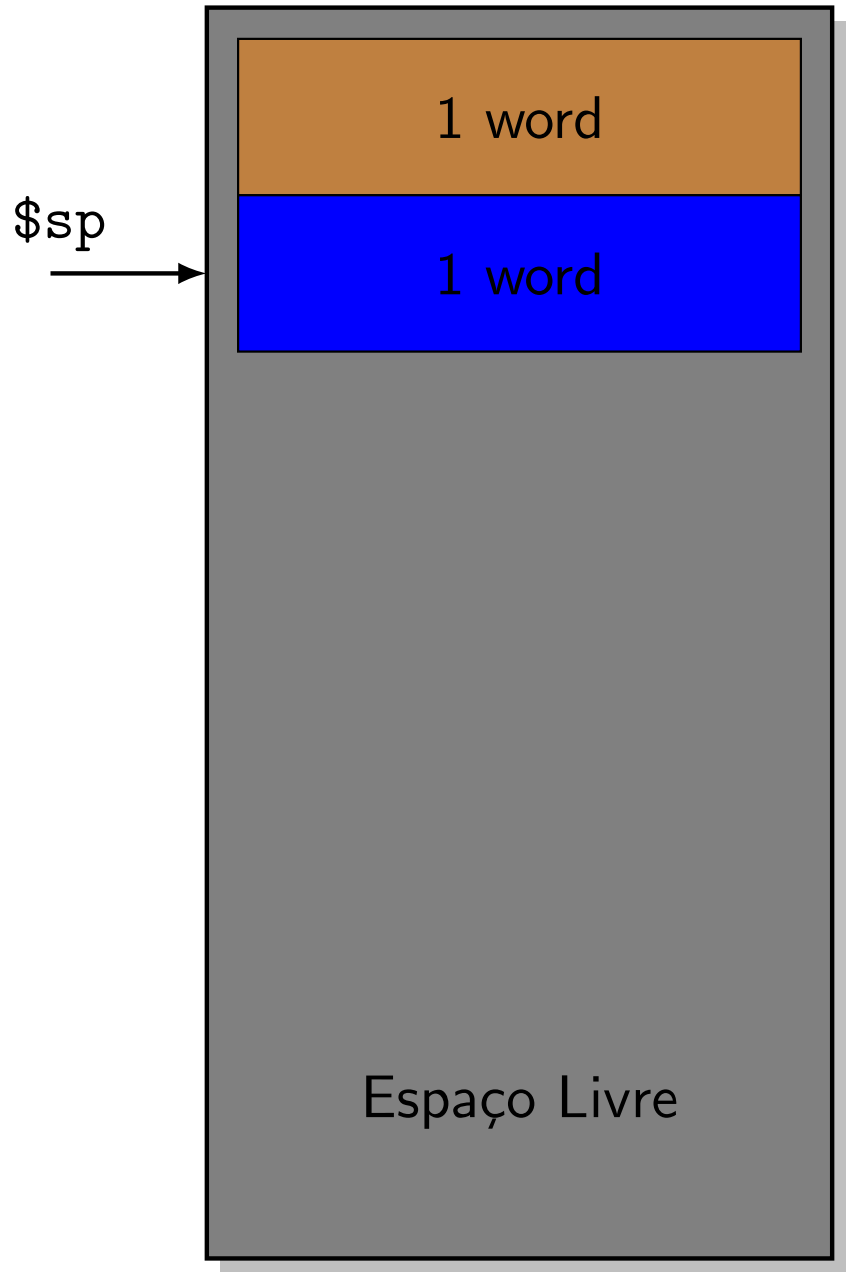


A pilha cresce de cima para baixo!

O registo $sp aponta para a última word ocupada ("topo" da pilha)

Implementação da pilha em memória (Stack Pointer $sp)

```
$sp = 0x7ffffff8
```



A pilha cresce de cima para baixo!

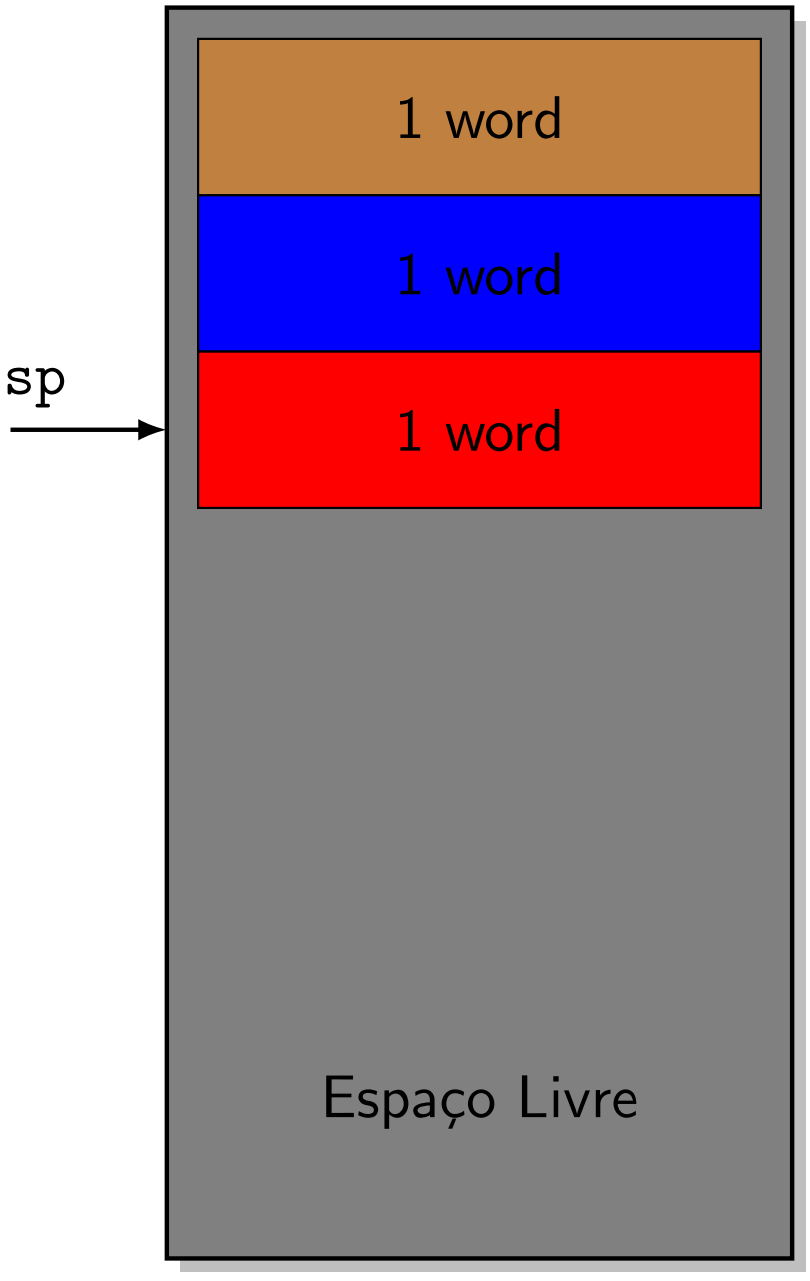
O registo $sp aponta para a última word ocupada ("topo" da pilha)

Implementação da pilha em memória (Stack Pointer $sp)

```
$sp = 0x7ffffff4
```

A pilha cresce de cima para baixo!

O registo $sp aponta para a última word ocupada ("topo" da pilha)

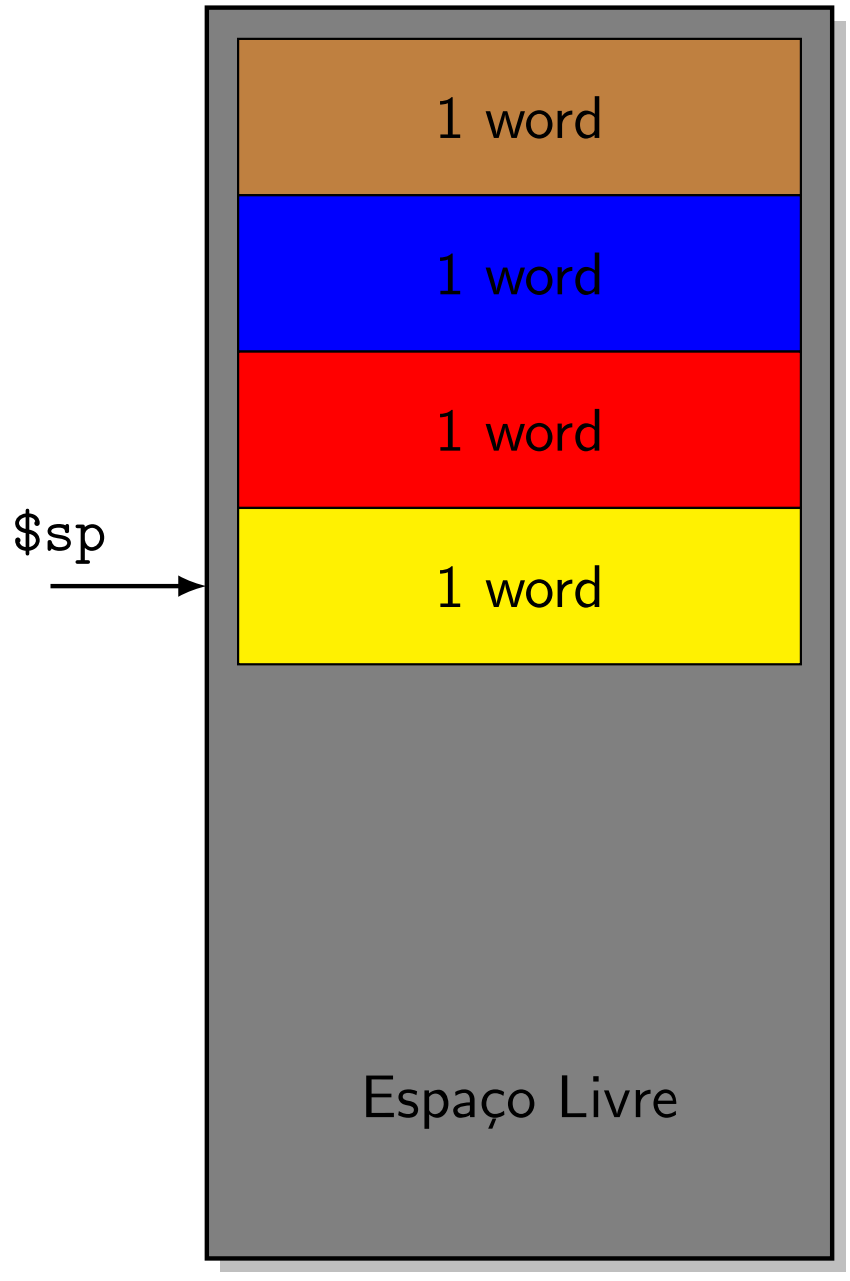


Implementação da pilha em memória (Stack Pointer $sp)

```
$sp = 0x7ffffff0
```

A pilha cresce de cima para baixo!

O registo $sp aponta para a última word ocupada ("topo" da pilha)

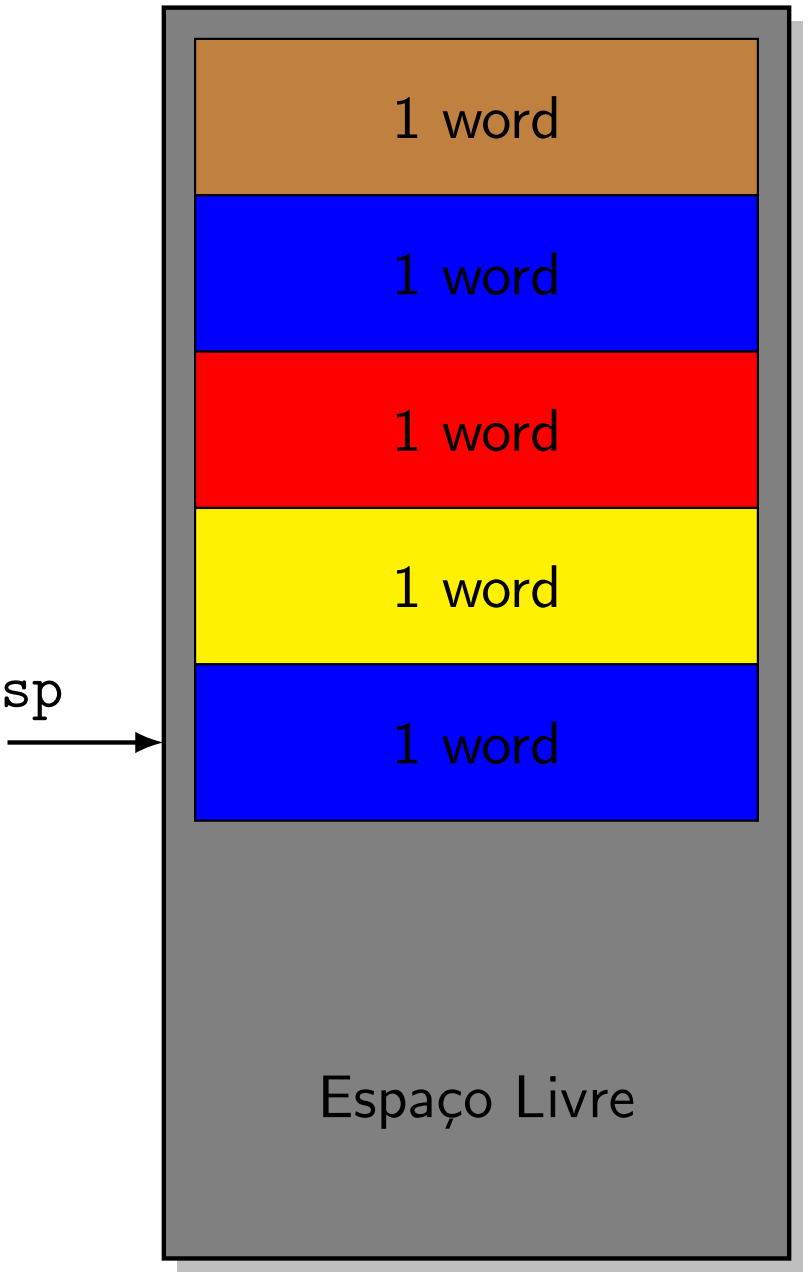


Implementação da pilha em memória (Stack Pointer $sp)

```
$sp = 0x7fffffec
```

A pilha cresce de cima para baixo!

O registo `$sp` aponta para a última word ocupada ("topo" da pilha)

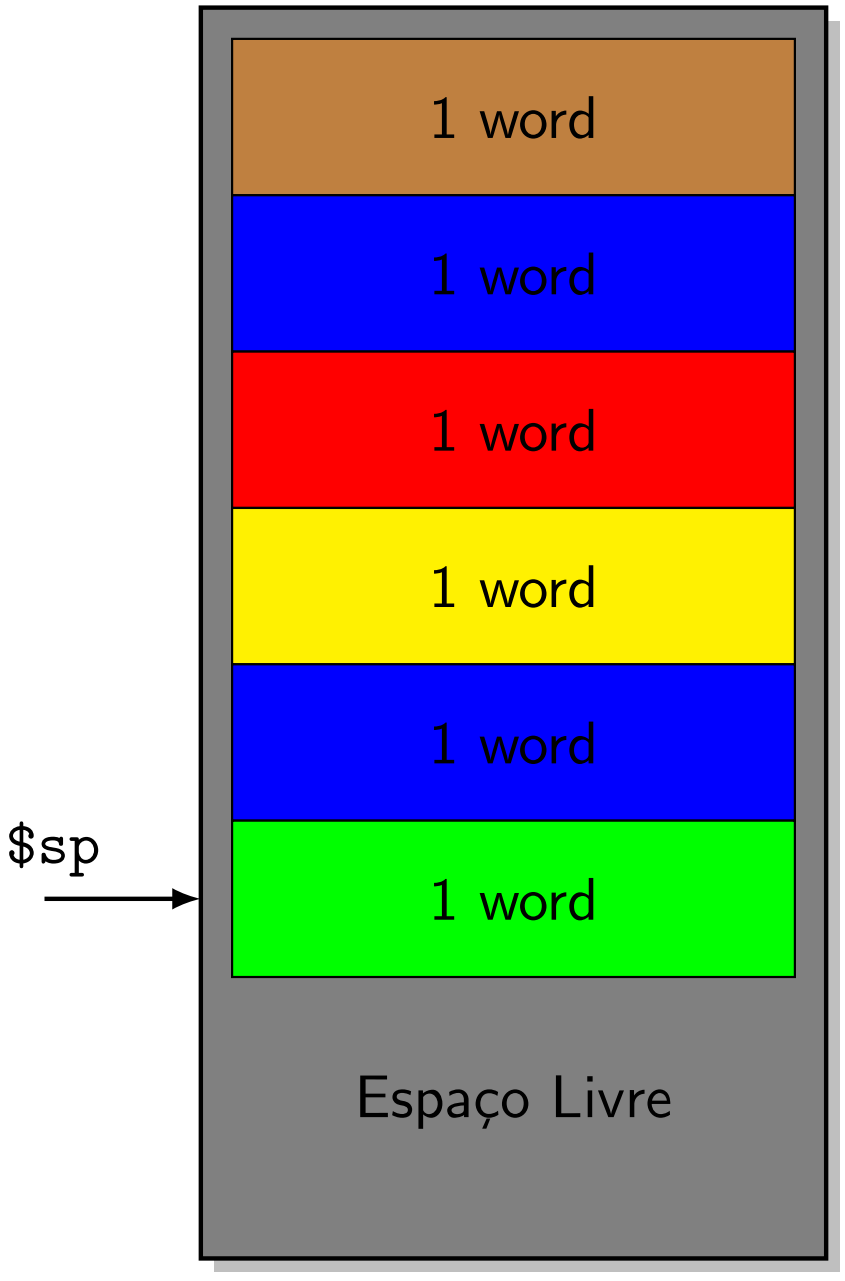


Implementação da pilha em memória (Stack Pointer $sp)

```
$sp = 0x7fffffe8
```

A pilha cresce de cima para baixo!

O registo $sp aponta para a última word ocupada ("topo" da pilha)

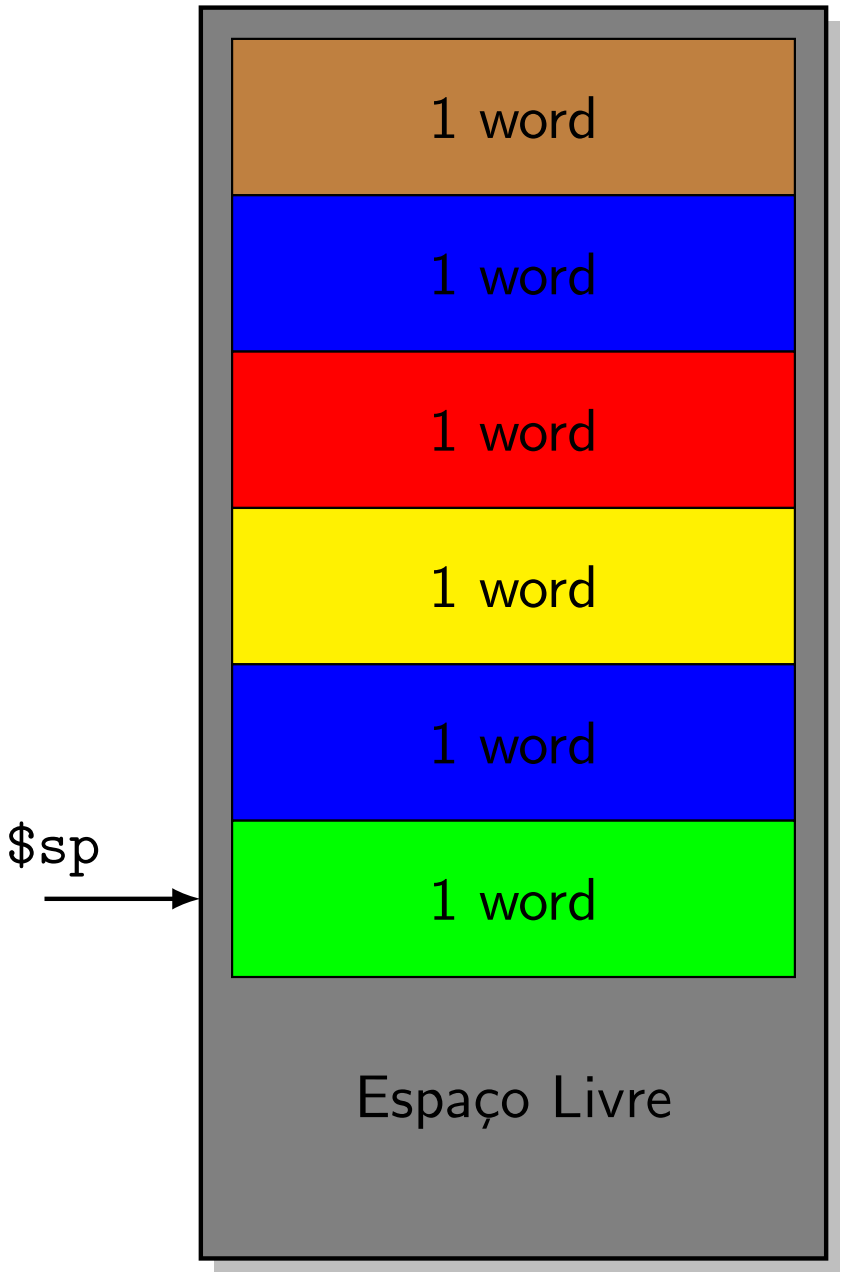


Implementação da pilha em memória (Stack Pointer $sp)

```
$sp = 0x7ffffe8
```

A pilha cresce de cima para baixo!

O registo $sp aponta para a última word ocupada ("topo" da pilha)



Exemplo

Pretende-se calcular

$$t1 = \text{funcao}(t0) + t0$$

```
addi $sp, $sp, -4       # aloca espaco na pilha
sw $t0, 0($sp)          # guarda t0 original na pilha

move $a0, $t0           # copia t0 -> a0 (argumento)
jal funcao              # e chama funcao f
nop                     # v0 contem resultado

lw $t0, 0($sp)          # recupera t0 original da pilha
add $t1, $v0, $t0       # t1 = funcao(t0) + t0

addi $sp, $sp, 4        # liberta espaco na pilha
```

Exemplo

$$t1 = \mathrm{f}(t0) + \mathrm{g}(t0)$$

```
addi $sp, $sp, -8       # aloca espaco na pilha (2 words)
sw $t0, 4($sp)          # guarda t0 original na pilha

move $a0, $t0           # copia t0 -> a0 (argumento)
jal f                   # e chama funcao f
nop                     # v0 contem resultado
sw $v0, 0($sp)          # guarda v0 = f(t0) na pilha

lw $a0, 4($sp)          # recupera t0 original da pilha
jal g                   # para usar como argumento na
nop                     # chamada da funcao g()

lw $t0, 0($sp)          # recupera resultado de f(t0)
add $t1, $v0, $t0       # t1 = f(t0) + g(t0)
addi $sp, $sp, 8        # liberta espaco na pilha
```

Convenções de utilização dos registos

Certos registos podem ser modificados livremente pelas funções. É do caso dos registos: `$t0`–`$t9`, `$a0`–`$a3`, `$v0`–`$v1` e `$ra`. Diz-se que não são preservados.

Existem outros registos onde se garante que os valores são preservados quando o programa cruza uma chamada a uma função. É o caso dos registos `$s0`–`$s7`, `$gp`, `$sp` e `$fp`.

Exemplo:

```
li $s0, 123
li $t0, 456

jal xpto
nop

# s0 == 123 ? SIM!
# t0 == 456 ? NAO!!!
```

Exemplo

Pretende-se implementar a função $f(x) \triangleq g(x) + x$.

```
# Inicio da funcao 'f'
# Argumento 'x' recebido em $a0
f:
    addi $sp, $sp, -4    # aloca espaco na pilha
    sw $s0, 0($sp)       # temos de preservar s0 (se usado)
    move $s0, $a0        # guarda a0 para ser usado + tarde

    jal g                # g(x)
    nop

    add $v0, $v0, $s0    # v0 = g(x) + x

    lw $s0, 0($sp)       # repoe s0
    addi $sp, $sp, 4     # desaloca espaco da pilha

    jr $ra               # retorna da funcao???
    nop
```