Aula 05
ISA: suporte para procedimentos

Procedimento

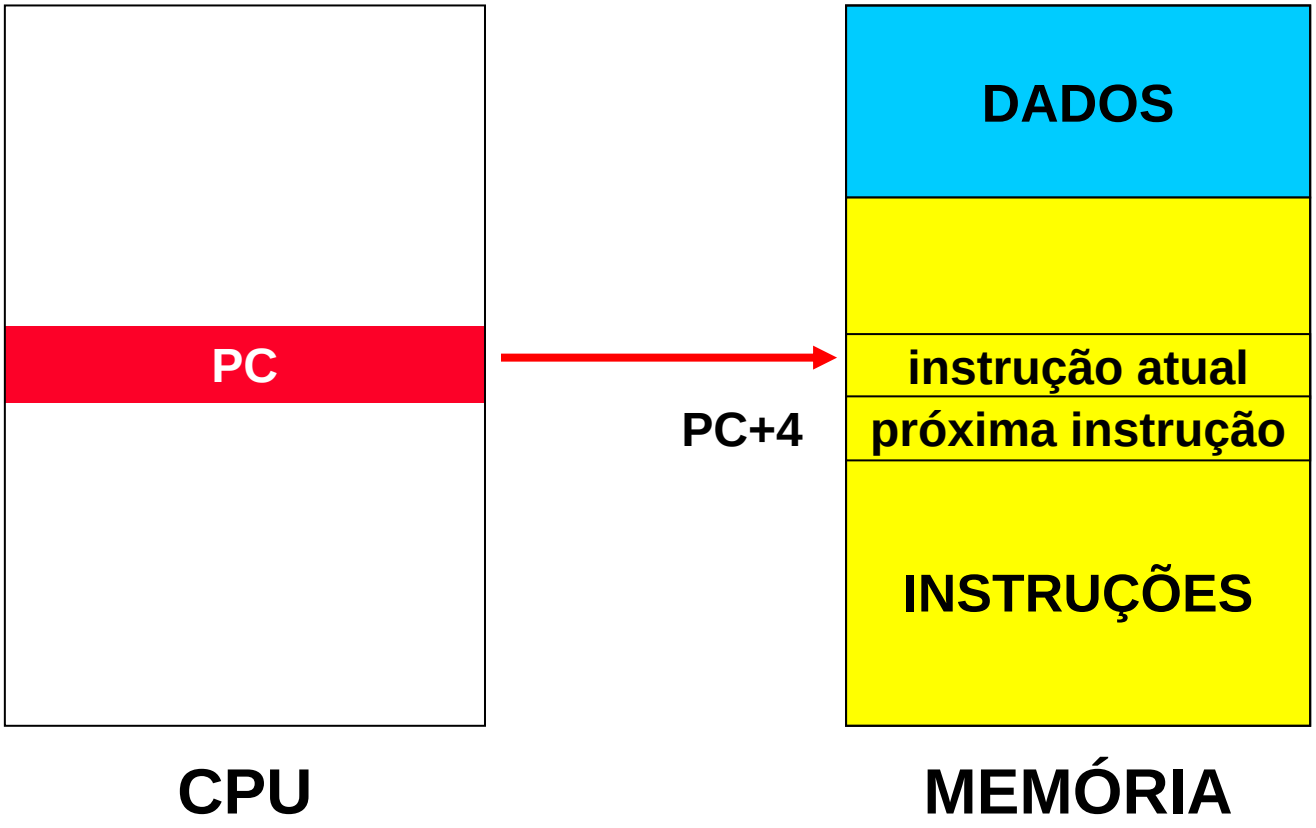
- **Uma sub-rotina armazenada**
  - **Efetua tarefa específica**
  - **Baseada nos parâmetros passados**
- **Vantagens**
  - **Estruturação do programa**
  - **Isola o programador do resto do programa**
    - » **entrada: parâmetros**
    - » **saída: valor(es) retornado(s)**

A dinâmica de um procedimento

- **Colocar os parâmetros num lugar onde o procedimento possa acessá-lo**
- **Transferir o controle para o procedimento**
- **Adquirir os recursos de armazenamento necessários para executar o procedimento**
- **Efetuar a tarefa desejada**
- **Colocar o resultado num lugar onde o programa chamador possa acessá-lo**
- **Retornar o controle ao ponto de origem**

Acesso a uma instrução

- **Registrador de endereço da instrução**
  - **PC**



MIPS: Suporte para procedimentos

- **Registradores**
  - **$a0-$a3: para armazenar parâmetros**
  - **$v0-$v1: para armazenar valores de retorno**
  - **$ra: para armazenar o endereço de retorno**
- **Instruções**
  - **jal Endereço_Procedure**
    - » **Salva em $ra o endereço da próxima instrução ($ra = PC+4)**
    - » **Desvia para o endereço (PC = Endereço_Procedure)**
  - **jr $ra**
    - » **Retorna à rotina chamadora (PC = $ra)**

MIPS: suporte para procedimentos

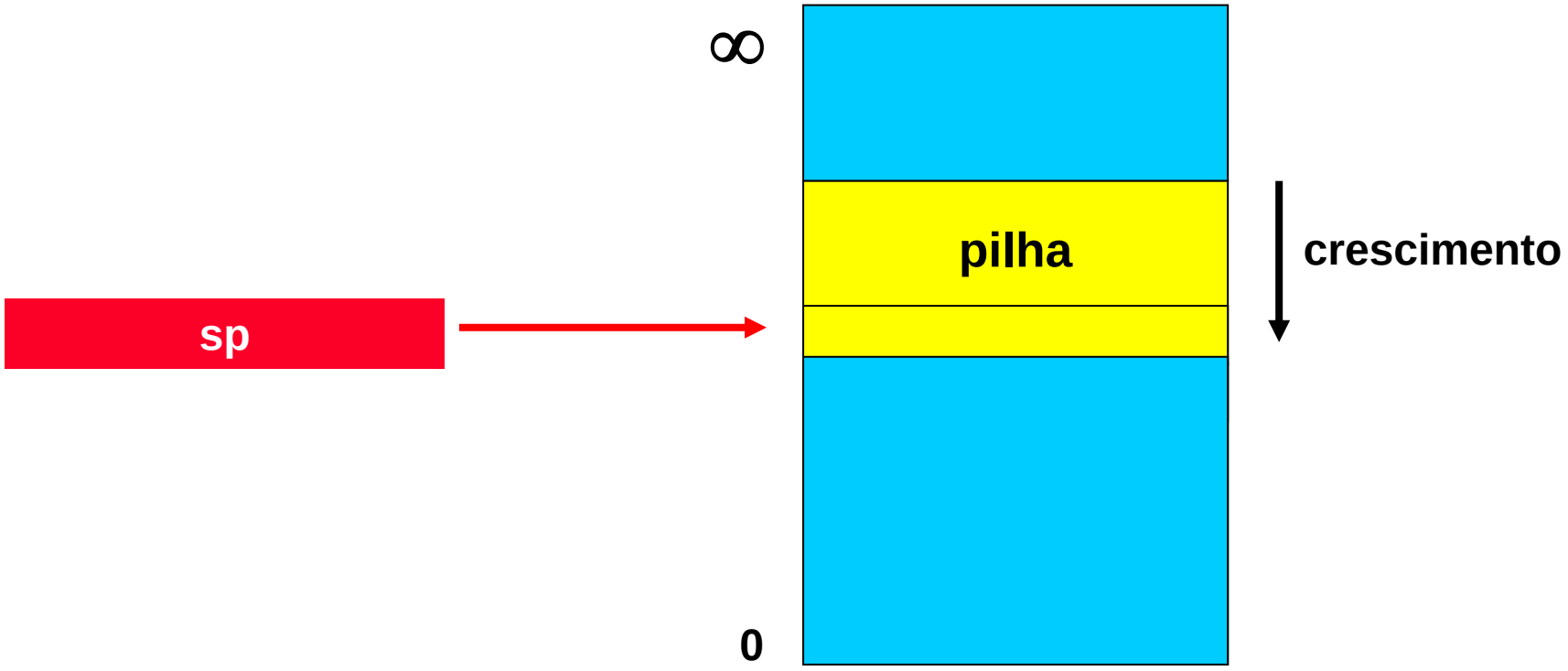
- **Antes de invocar o procedimento ...**
  - **Escrever na memória o conteúdo dos registradores a serem utilizados pelo procedimento chamado**
    - » **Salvamento do contexto do procedimento chamador**
- **Depois de executar o procedimento...**
  - **Restaurar os valores originais nos registradores**
    - » **Restauração do contexto do procedimento chamador**

MIPS: suporte para procedimentos

- **Contexto salvo numa estrutura do tipo pilha**
  - **LIFO (“last in, first out”)**
- **Precisa-se de ponteiro para o topo da pilha**
  - **$sp (“stack pointer”)**
- **Operações**
  - **“Push”: insere na pilha;**
  - **“Pop”: retira da pilha**

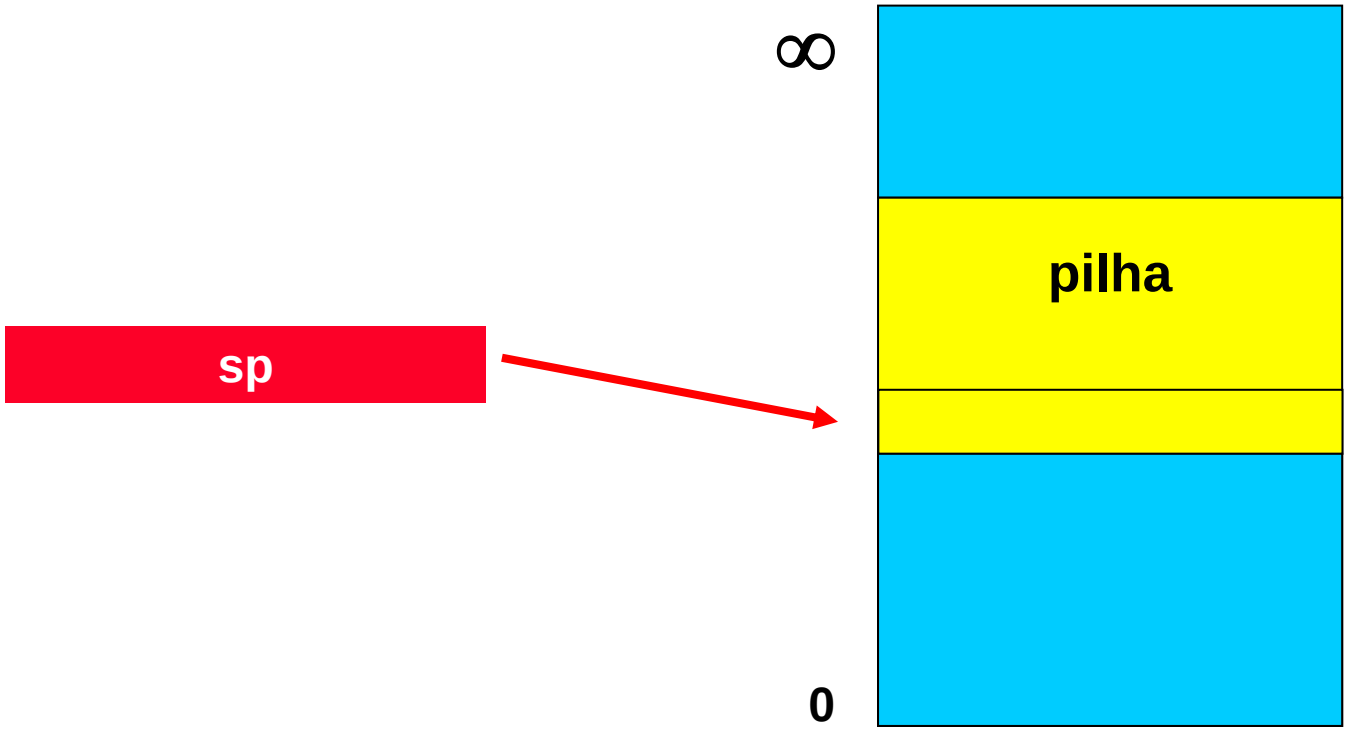
MIPS: suporte para procedimentos

- **Crescimento da pilha:**
  - **do maior para o menor endereço**



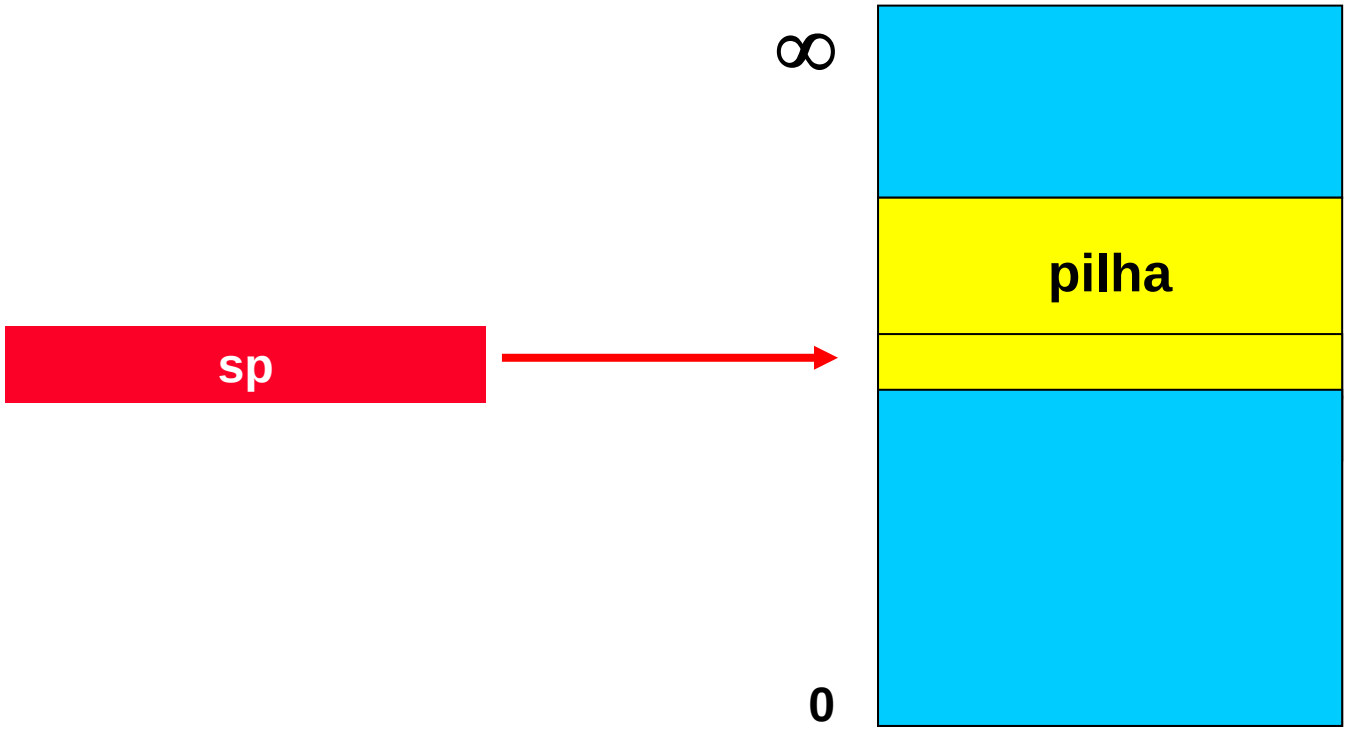
MIPS: suporte para procedimentos

- **Crescimento da pilha:**
  - **do maior para o menor endereço**
    - » **push: decrementa o “stack pointer” ($sp = $sp - 4)**



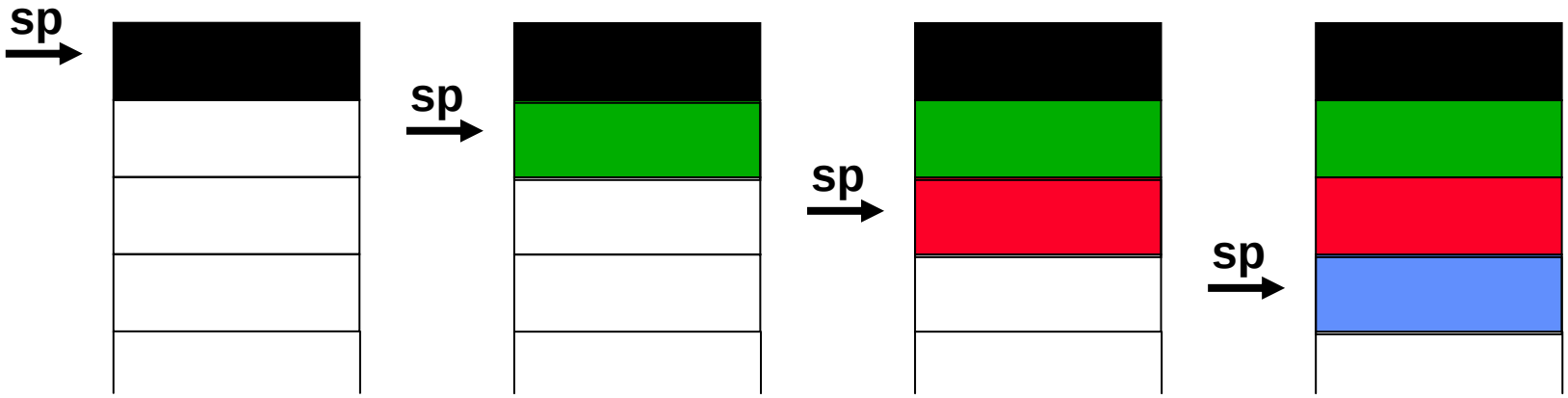
MIPS: suporte para procedimentos

- **Crescimento da pilha:**
  - **do maior para o menor endereço**
    - » **push: decrementa o “stack pointer” ($sp = $sp - 4)**
    - » **pop: incrementa o “stack pointer” ($sp = $sp + 4)**



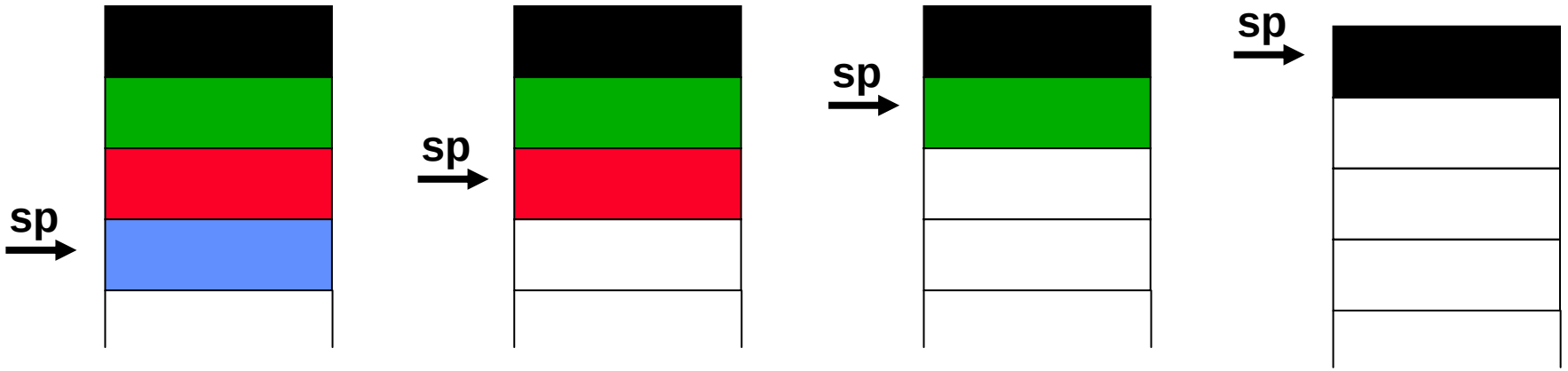
Salvamento do contexto ("pushing")

```
addi  $sp, $sp, -4
sw    $t1, 0($sp)      # salva registrador $t1
addi  $sp, $sp, -4
sw    $t0,0($sp)       # salva registrador $t0
addi  $sp, $sp, -4
sw    $s0, 0($sp)      # salva registrador $s0
```



Restauração do contexto ("popping")

```
lw   $s0, 0($sp)     # restaura registrador $s0
addi $sp, $sp, 4
lw   $t0, 0($sp)     # restaura registrador $t0
addi $sp, $sp, 4
lw   $t1, 0($sp)     # restaura registrador $t1
addi $sp, $sp, 4
```



Compilando um procedimento

```
int leaf_example (int g, int h, int i, int j)
{
    int f;
    f = (g+h) - (i+j);
    return f;
}
```

- (g,h,i,j)→($a0,$a1,$a2,$a3); f → $s0;
- Antevendo o corpo do procedimento:

```
add $t0, $a0, $a1        # $t0 = g + h
add $t1, $a2, $a3        # $t1 = i + j
sub $s0, $t0, $t1        #  f = $t0 - $t1
```

Compilando um procedimento

- **Salvando o contexto da rotina chamadora...**

```
addi $sp, $sp, -12      # amplia pilha p/ 3 itens
sw $t1, 8($sp)          # salva registrador $t1
sw $t0, 4($sp)          # salva registrador $t0
sw $s0, 0($sp)          # salva registrador $s0
```

Compilando um procedimento

- **Corpo do procedimento:**

```
add $t0, $a0, $a1        # $t0 = g + h
add $t1, $a2, $a3        # $t1 = i + j
sub$s0, $t0, $t1         #  f = $t0 - $t1
```

- **Copiando o valor de retorno f**

```
add $v0, $s0, $zero      # retorna f ($v0 = $s0 + 0)
```

Compilando um procedimento

- **Restaurando os registradores salvos**

```
lw  $s0, 0($sp)          # restaura $s0
lw  $t0, 4($sp)          # restaura $t0
lw  $t1, 8($sp)          # restaura $t1
addi $sp, $sp, 12        # deleta 3 itens
```

- **Retornando à rotina chamadora**

```
jr $ra
```

Juntando tudo

Rotina chamadora

```
. . .
jal leaf_example
. . .
```

Subrotina chamada

```
leaf_example:
addi    $sp, $sp, -12
sw      $t1, 8($sp)
sw      $t0, 4($sp)
sw      $s0, 0($sp)
add     $t0, $a0, $a1
add     $t1, $a2, $a3
sub     $s0, $t0, $t1
add     $v0, $s0, $zero
lw      $s0, 0($sp)
lw      $t0, 4($sp)
lw      $t1, 8($sp)
addi    $sp, $sp,12
jr      $ra
```

Como reduzir o “spilling”?

Rotina chamadora

. . . ← $t0 definido

jal leaf_example

. . . ← $t0 usado

Subrotina chamada

leaf_example:

```
addi    $sp, $sp, -12
sw      $t1, 8($sp)
sw      $t0, 4($sp)      ← $t0 preservado
sw      $s0, 0($sp)
add     $t0, $a0, $a1
add     $t1, $a2, $a3
sub     $s0, $t0, $t1
add     $v0, $s0, $zero
lw      $s0, 0($sp)
lw      $t0, 4($sp)      ← $t0 restaurado
lw      $t1, 8($sp)
addi    $sp, $sp,12
jr      $ra
```

Como reduzir o “spilling”?

Rotina chamadora

```
. . .   ← $t0 definido
jal leaf_example
. . .   ← $t0 não usado
```

Subrotina chamada

```
leaf_example: addi  $sp, $sp, -12
              sw    $t1, 8($sp)
              sw    $t0, 4($sp)
              sw    $s0, 0($sp)
              add   $t0, $a0, $a1
              add   $t1, $a2, $a3
              sub   $s0, $t0, $t1
              add   $v0, $s0, $zero
              lw    $s0, 0($sp)
              lw    $t0, 4($sp)
              lw    $t1, 8($sp)
              addi  $sp, $sp,12
              jr    $ra
```

Convenção de chamadas

Rotina chamadora

. . . ← $t0 definido

jal leaf_example

. . . ← $t0 não usado

Divisão de tarefas entre chamadora e chamada:

Chamadora supõe que $t0-$t8 **não são preservados** pela rotina chamada

Chamadora supõe que $s0-$s7 **são preservados** pela rotina chamada (se usados)

Subrotina chamada

```
leaf_example:  addi  $sp, $sp, -12
               sw    $t1, 8($sp)
               sw    $t0, 4($sp)
               sw    $s0, 0($sp)
               add   $t0, $a0, $a1
               add   $t1, $a2, $a3
               sub   $s0, $t0, $t1
               add   $v0, $s0, $zero
               lw    $s0, 0($sp)
               lw    $t0, 4($sp)
               lw    $t1, 8($sp)
               addi  $sp, $sp,12
               jr    $ra
```

Rotinas aninhadas

- **Rotinas-folha**
  - **Não chamam outras**
- **Rotinas aninhadas**
  - **Exemplo: “main” chama “A” que chama “B”**
  - **“main”chama “A(3)”**
    - **$a0 = 3; jal A**
  - **“A” chama “B(7)”**
    - **$a0 = 7; jal B**
  - **Conflitos**
    - **Argumento de “A” modificado por “B”**
    - **Endereço de retorno a “main” é perdido**

Rotinas aninhadas

- **Solução: preservar registradores na pilha**
  - **Rotina chamadora salva na pilha**
    - » **Registradores de argumento ($a0-$a3) necessários**
    - » **Registradores temporários ($t0-$t9) necessários**
  - **Rotina chamada salva na pilha**
    - » **Registradores usados em seu corpo ($s0-$s7)**
    - » **Registrador de endereço de retorno ($ra)**
  - **Stack pointer ($sp) é ajustado**
  - **No retorno**
    - » **Registradores são restaurados**
    - » **$sp é re-ajustado apropriadamente**

Compilando rotina recursiva

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

- **Rotina salva $a0 e $ra**

```
fact:
        addi   $sp, $sp, -8  # abre espaço para 2 itens
        sw     $ra, 4($sp)   # salva endereço de retorno
        sw     $a0, 0($sp)   #  salva argumento n
```

Compilando rotina recursiva

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

- Rotina chamada efetua o teste n < 1

```
slti $t0, $a0, 1        # t0 =1, se n < 1
beq $t0, $zero, L1      # se n >= 1, vá para L1
```

Compilando rotina recursiva

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

- Se n < 1, rotina chamada retorna

```
addi $v0, $zero, 1      # retorna valor 1
addi $sp, $sp, 8        # remove dois itens da pilha
jr $ra                  # retorna para depois de jal
```

Compilando rotina recursiva

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

- Se n >= 1, rotina invoca a si própria

```
L1:
        addi $a0, $a0, -1     # novo argumento é n-1
        jal fact              # fact é re-invocada
```

Compilando rotina recursiva

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

- **Restaurando contexto da rotina chamadora**

```
lw $a0, 0($sp)      # restaura argumento n
lw $ra, 4($sp)      # restaura endereço de retorno
addi $sp, $sp, 8    # remove dois itens da pilha
```

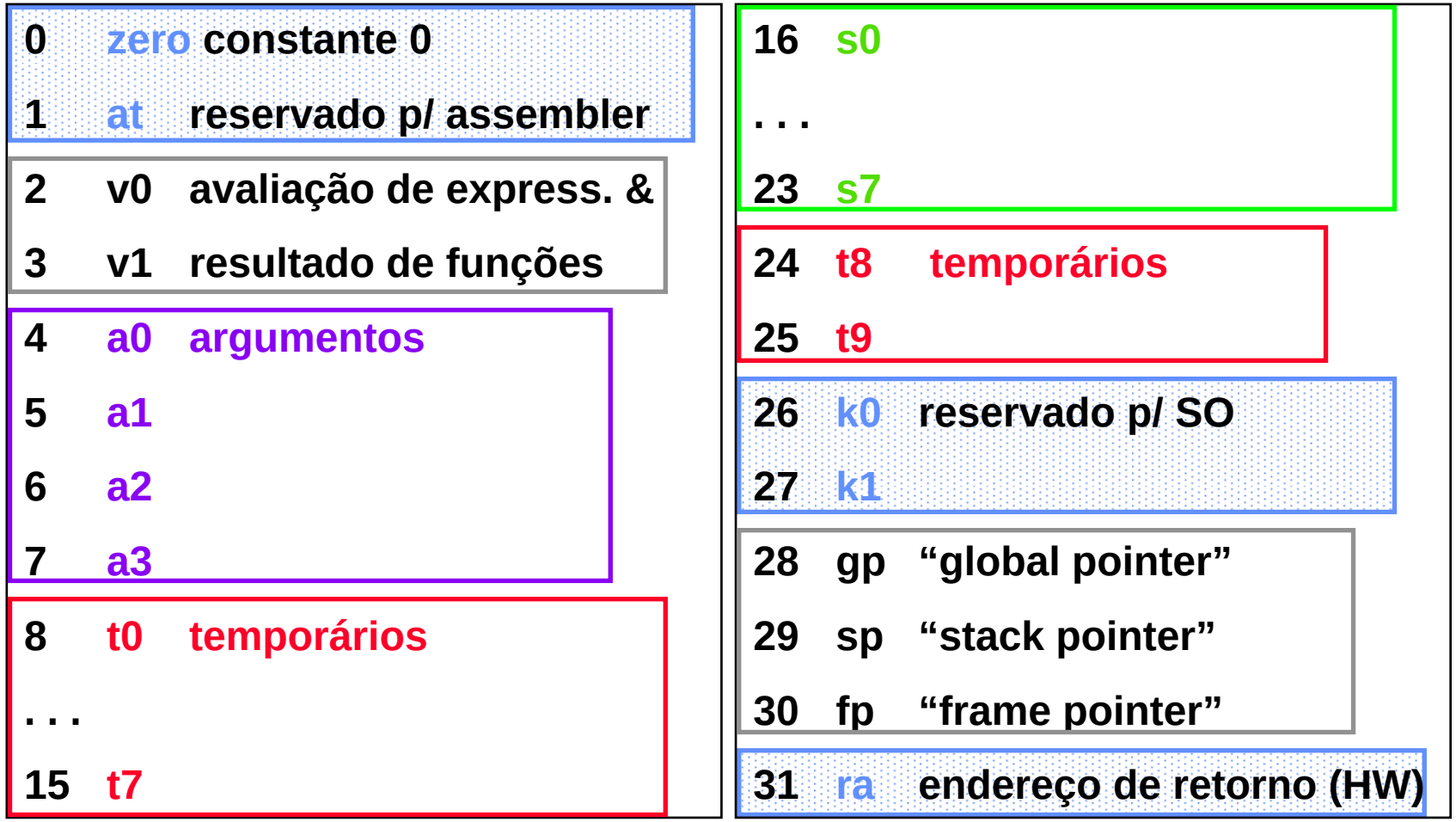
Compilando rotina recursiva

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

- Rotina chamada retorna o valor do produto

```
mul $v0, $a0, $v0        # retorna o produto
jr $ra                   # retorna à chamadora
```

MIPS:
Convenção de SW p/ Registradores



(Adapted from "Computer Organization & Design: The Hardware/Software Interface", D. Patterson and J. Hennessy, Morgan Kaufmann Publishers. Copyright 1998 UCB.)