

MIPS过程（函数）调用

本节内容

- MIPS过程调用的指令
- MIPS过程调用的规范
- MIPS过程调用的机制

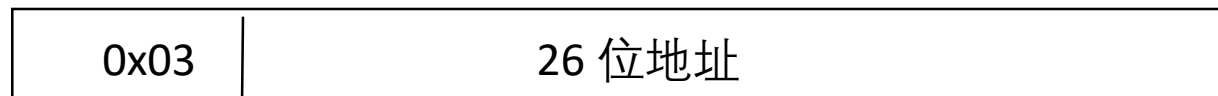
注：在本节，不区分过程与函数的区别

过程调用指令

- MIPS 过程调用指令:

`jal Procedure Address #jump and link`

- 作用: 将下一条指令的地址,即PC+4保存在寄存器 \$ra 中,从而当过程返回时可以链接到当前指令的下一条指令。
- 指令格式 (J 型):

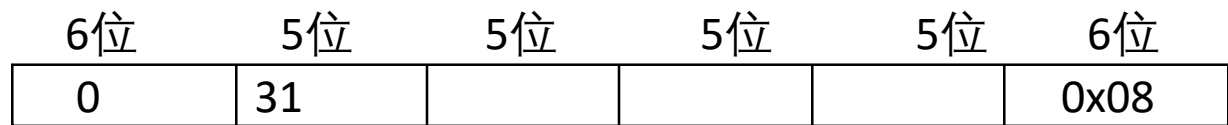


过程返回指令

- MIPS过程返回指令

`jr $ra` `#return`

- 指令格式 (R型):



R型 `$ra`

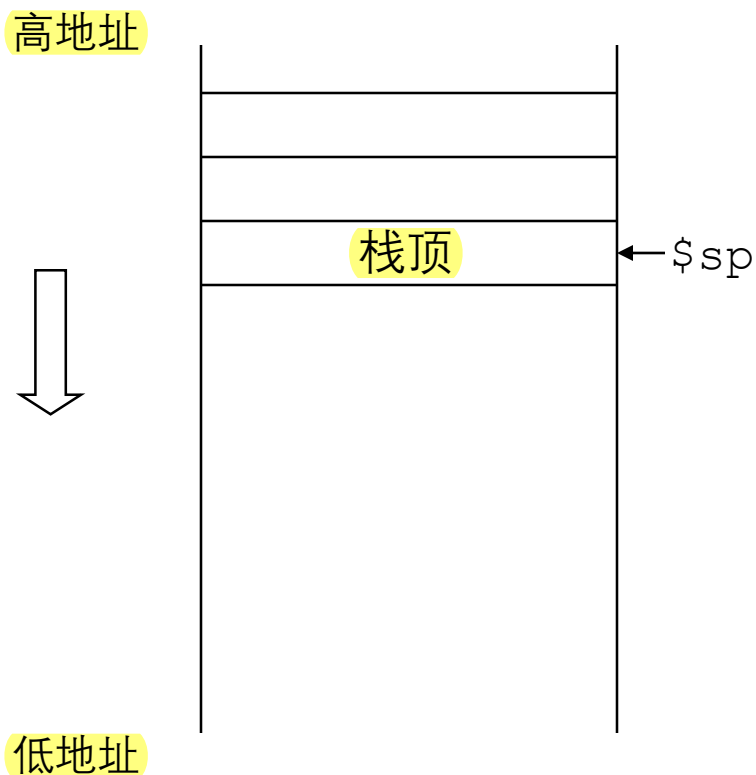
Jump register

过程执行的六个步骤

1. 主程序（调用者）将参数放置在过程（被调用者）可以访问到的位置
 - `$a0 - $a3`: 四个参数寄存器
2. 调用程序将系统控制权转移给被调用过程 (`jal`)
3. 被调用过程申请所需的存储资源
4. 被调用过程执行相应的任务
5. 被调用过程将执行结果存放在主程序可以访问的位置
 - `$v0 - $v1`: 两个结果值寄存器
6. 被调用过程将系统控制权移交给调用程序(`jr`)
 - `$ra`: 返回地址寄存器

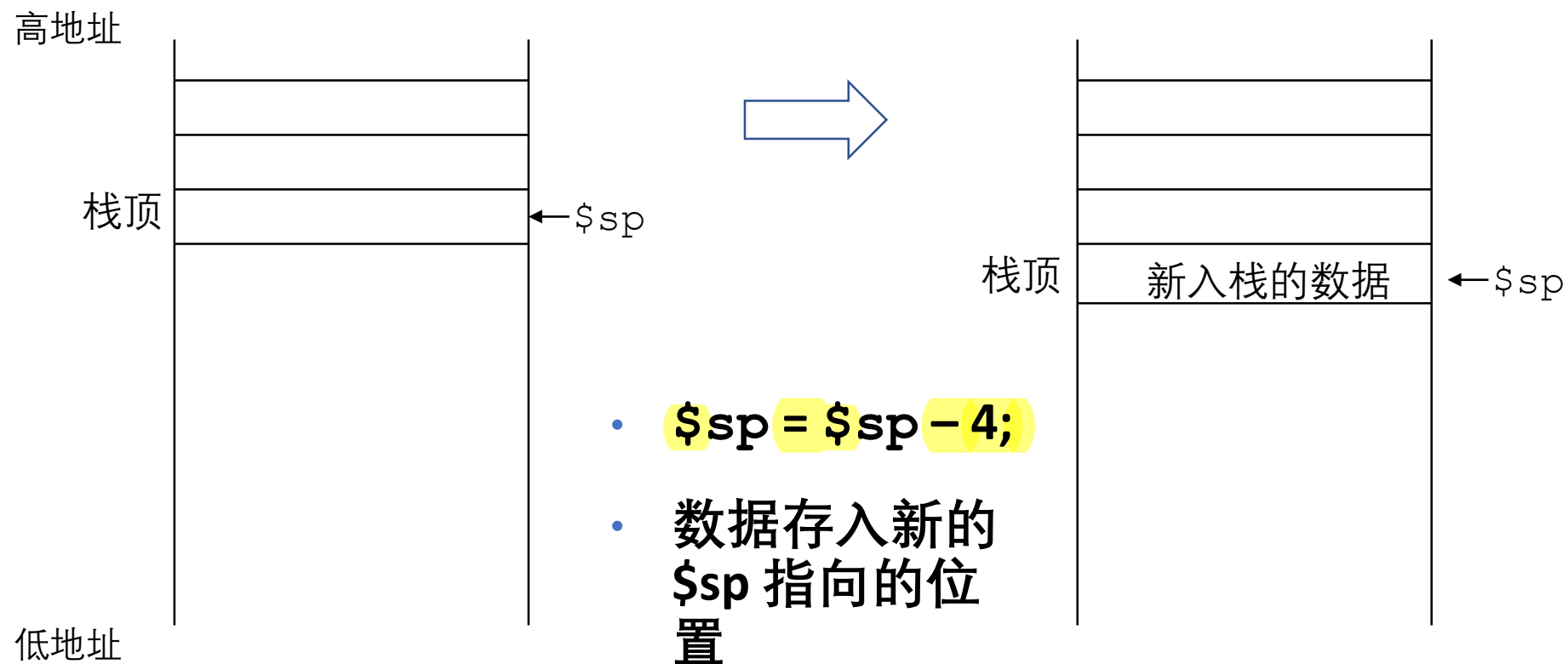
寄存器溢出

- 如果被调用过程需要使用的寄存器多于能分配到的寄存器，怎么办？
- 使用栈：一种后进先出的队列

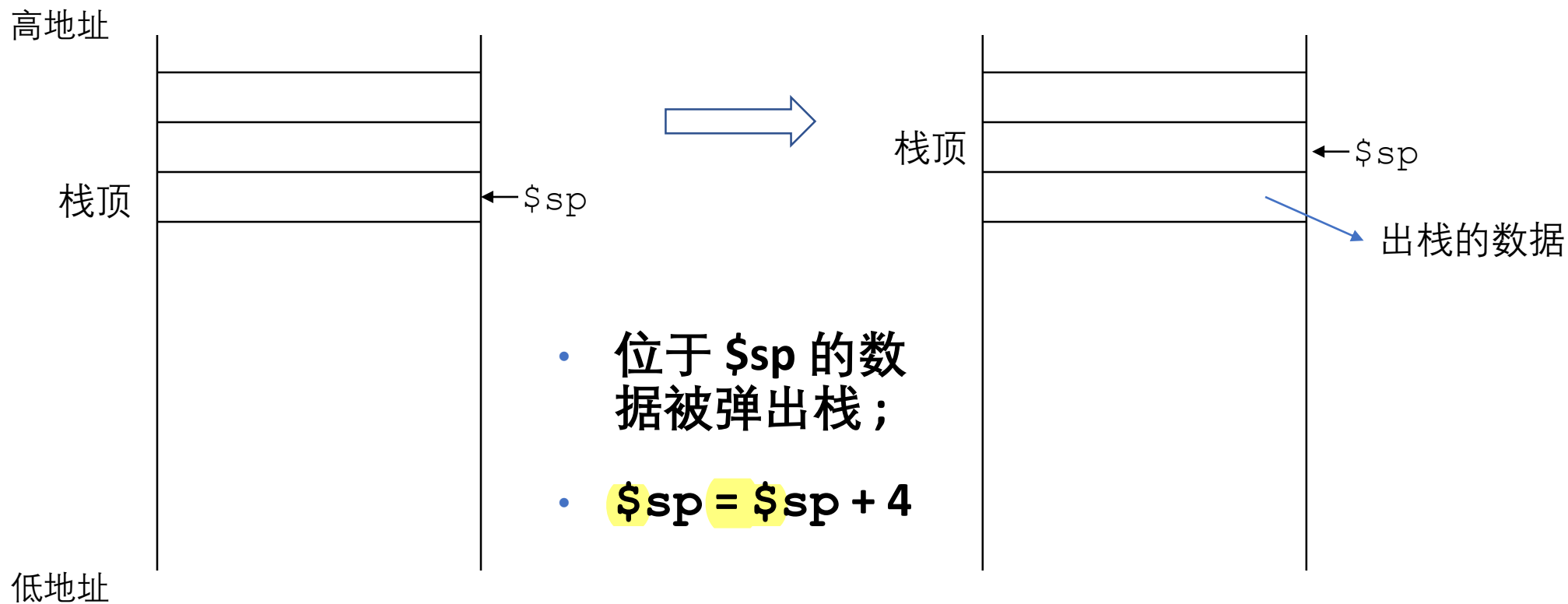


- 栈：一块从“高地址向低地址”增长“的内存空间
- 通用寄存器 `$sp` (`$29`)：栈的寻址
- Push: 数据加入栈中
- Pop: 数据从栈中弹出

Push: 数据加入栈




Pop: 数据从栈中弹出



程序示例

- C 语言代码:

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

- g, h, i, j 分别位于寄存器 \$a0, \$a1, \$a2, \$a3 中
- 函数返回的结果存储在 \$v0, \$v1
- f 在 \$s0 中 (我们需要将 \$s0 存在栈中) 

leaf_example:		
addi	\$sp, \$sp, -4	} \$s0 入栈
sw	\$s0, 0(\$sp)	
add	\$t0, \$a0, \$a1	} 函数体
add	\$t1, \$a2, \$a3	
sub	\$s0, \$t0, \$t1	
add	\$v0, \$s0, \$zero	} 结果
lw	\$s0, 0(\$sp)	} 恢复 \$s0
addi	\$sp, \$sp, 4	
jr	\$ra	} 返回

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

嵌套程序

```
int rt_1 (int i)
{
    if (i == 0)
        return 0;
    else
        return rt_2(i-1);
}
```

- i在\$a0;
- 返回结果在\$v0

```
caller:
    jal    rt_1
next: . . .
```

```
rt_1:
    bne    $a0, $zero, to_2
    add    $v0, $zero, $zero
    jr     $ra
```

```
to_2:
    addi    $a0, $a0, -1
    jal     rt_2
    jr     $ra
```

```
rt_2: . . .
```

存在的问题:

```
caller:
jal    rt_1
next: . . .

rt_1:
bne    $a0, $zero, to_2
add    $v0, $zero, $zero
jr     $ra

to_2:
addi   $a0, $a0, -1
jal    rt_2
jr     $ra

rt_2: . . .
```

问题1:

- ❑ 跳转到rt_2之前, rt_1的返回地址 (调用者的下一条指令, 即next: . . .的地址) 存储在 \$ra 寄存器中.
- ❑ 跳转到rt_2之后, \$ra寄存器中的值将会发生什么变化?
- ❑ \$ra 寄存器中存放的是新的返回地址 (即jr \$ra 的地址)
- ❑ rt_1的返回地址丢失

存在的问题:

```
caller:
jal    rt_1
next: . . .

rt_1:
bne    $a0, $zero, to_2
add    $v0, $zero, $zero
jr     $ra

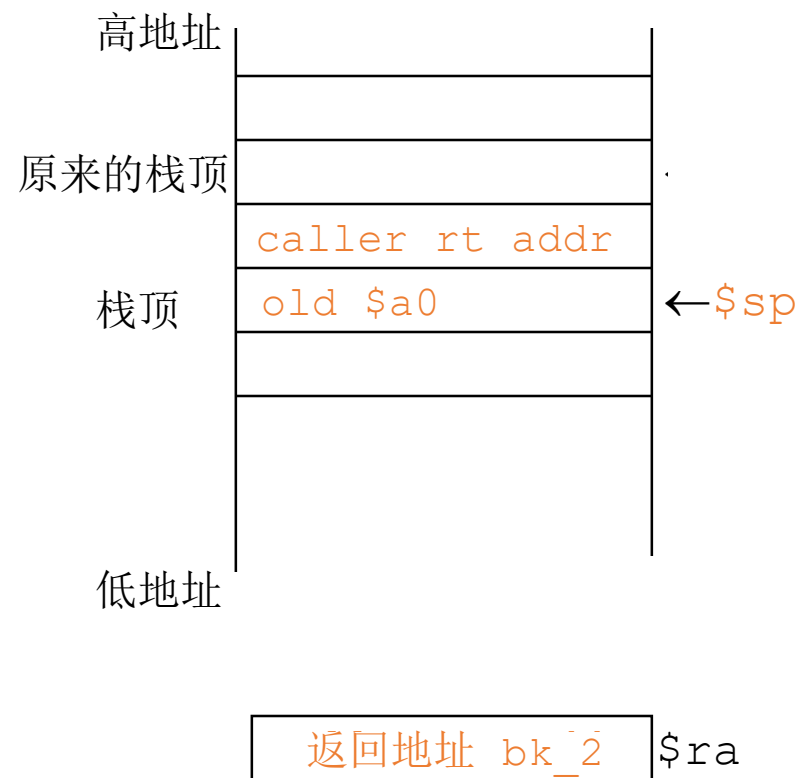
to_2:
addi   $a0, $a0, -1
jal    rt_2
jr     $ra

rt_2: . . .
```

问题2:

- ❑ 在过程rt_1中, \$a0中存放i的值
- ❑ 跳转到rt_2之后, 在\$a0中存放i-1的值
- ❑ 从rt_2返回rt_1之后, 在\$a0中不是i的值

保存返回地址 保存寄存器的内容



- i在\$a0;
- 返回结果在\$v0
- 返回地址在\$ra

```

rt_1:
    bne $a0, $zero, to_2
    add $v0, $zero, $zero
    jr  $ra
to_2:
    addi $sp, $sp, -8
    sw   $ra, 4($sp)
    sw   $a0, 0($sp)
    addi $a0, $a0, -1
    jal  rt_2
bk_2:
    lw   $a0, 0($sp)
    lw   $ra, 4($sp)
    addi $sp, $sp, 8
    jr   $ra
    
```

编译递归程序

- 计算阶乘:

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

- 递归过程(调用自身的过程)

fact (0) = 1

fact (1) = 1 * 1 = 1

fact (2) = 2 * 1 * 1 = 2

fact (3) = 3 * 2 * 1 * 1 = 6

fact (4) = 4 * 3 * 2 * 1 * 1 = 24

...

- 假定n存储在 \$a0中; 结果存储在 \$v0中

fact:

addi	\$sp, \$sp, -8	#adjust stack pointer
sw	\$ra, 4(\$sp)	#save return address
sw	\$a0, 0(\$sp)	#save argument n

用栈存储已经在
\$ra和\$a0中的数据

分支

slti	\$t0, \$a0, 1	#test for $n < 1$
beq	\$t0, \$zero, L1	#if $n \geq 1$, go to L1
addi	\$v0, \$zero, 1	#else return 1 in \$v0

addi	\$sp, \$sp, 8	#adjust stack pointer
jr	\$ra	#return to caller (1 st)

返回 Fact(0)

L1:

addi	\$a0, \$a0, -1	# $n \geq 1$, so decrement n
jal	fact	#call fact with (n-1)

#this is where fact returns

调用过程
Fact(n-1)

bk_f:

lw	\$a0, 0(\$sp)	#restore argument n
lw	\$ra, 4(\$sp)	#restore return address
addi	\$sp, \$sp, 8	#adjust stack pointer
mul	\$v0, \$a0, \$v0	#\$v0 = $n * \text{fact}(n-1)$
jr	\$ra	#return to caller (2 nd)

在过程Fact(n-1)
之后返回

过程调用寄存器使用规范

MIPS将寄存器分为了两类:

1. 过程调用时保存

- `$ra`, `$sp`, `$gp`, `$fp`,
- `$s0`-`$s7`
- 参数寄存器 `$a0`-`$a3`,

2. 过程调用时不保存

- 返回结果寄存器 `$v0`,`$v1`,
- 临时寄存器 `$t0`-`$t9`

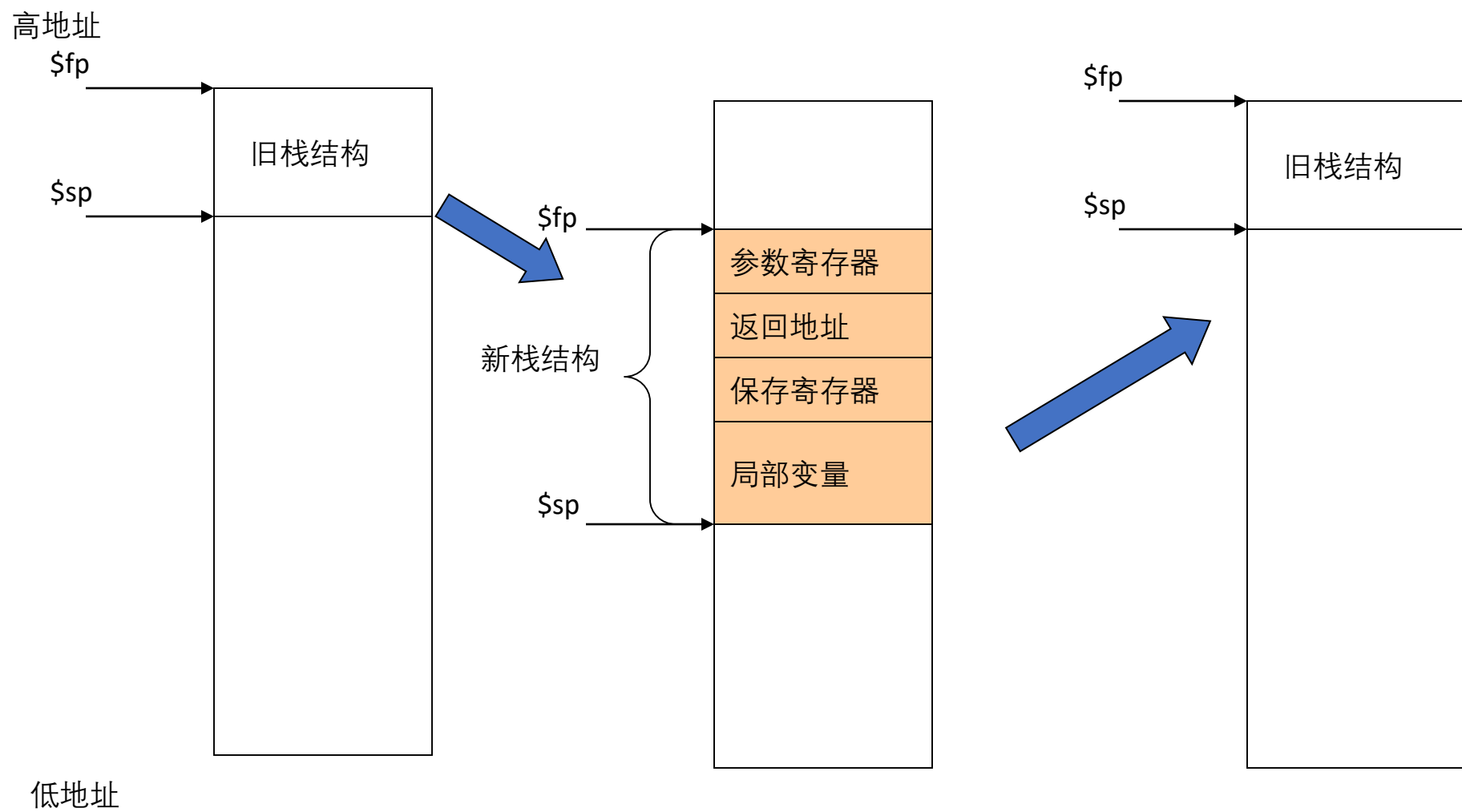
MIPS 寄存器说明

名称	寄存器号	用途	调用时是否保存?
\$zero	0	常数 0	n.a.
\$v0 - \$v1	2-3	返回值	no
\$a0 - \$a3	4-7	参数	yes
\$t0 - \$t7	8-15	临时变量	no
\$s0 - \$s7	16-23	保存值	yes
\$t8 - \$t9	24-25	临时变量	no
\$gp	28	全局指针	yes
\$sp	29	栈指针	yes
\$fp	30	结构指针	yes
\$ra	31	返回地址	yes

分配栈空间的机制

- 过程(函数) 的激活记录: 已保存寄存器和局部变量的堆栈段
- C 语言的变量有两种存储类别: 自动变量 (auto) 和静态变量(static)
 - 自动变量是函数的局部变量, 当函数调用时创建、退出时销毁
 - 静态变量在进程创建时生成, 进程终止时释放
- 栈可以用来存储寄存器无法存放的自动变量
- 栈指针(\$sp) 来指向当前栈帧的顶部
- 某些MIPS编译器使用结构指针(\$fp) 来指向当前栈帧第一个字或者结构

过程调用机制



小结

- MIPS过程调用的指令： jal, jr
- MIPS过程调用时寄存器使用规范
- MIPS过程调用的机制
 - 栈的概念
 - 过程调用期间，使用栈存储过程在活动中需要保存的局部变量、寄存器、返回地址；