计算机作曲理论第7期

-指令执行(1)-

大连理工大学立命馆大学国际信息软件学部大森孝之

讲座内容

- 指令的执行
  - 过程执行和堆栈

如何执行程序

- 程序
  - 构成程序一部分的一系列进程 在 C 语言中，一个“函数”

```
int main(){
   int result = function(100, 200);
   ...
}
```

函数调用　　实际论

返回值（类　　函数名　　形式论

```
int function(int a, int b){
   return a + b;
}
```

功能定义

* 返回值：也叫返回值

如何执行程序

- 您需要做什么来执行该程序
  - 将参数的值放在程序可访问的地方
  - 将控制权移交给程序
  - 分配程序内部处理所需的内存资源
  - 执行程序的内部处理
  - 将结果（返回值）放在调用者可以访问的地方
  - 将控制权返回到原始位置

如何执行程序

- 您需要做什么来执行该程序
  - 将参数的值放在程序可访问的地方
  - 将控制权移交给程序
  - 分配程序内部处理所需的内存资源
  - 执行程序的内部处理
  - 将结果（返回值）放在调用者可以访问的地方
  - 将控制权返回到原始位置

$a0-$a3

PC -> $ra 操作与 jal 指令

在记忆中

$v0-$v1

jr $ra

注册程序执行

$zero	0	始终为零
$at	1	汇编程序临时使用
$v0-v1	2-3	对于返回值
$a0-a3	4-7	对于参数
$t0-t9	8-15, 24-25	临时寄存器（用于临时变量）
$s0-s7	16-23	保存寄存器（用于变量）
$k0-k1	26-27	为操作系统内核保留
$gp	28	全局指针
$sp	29	堆栈指针
$fp	30	帧指针
$ra	31	退货地址

执行程序

jal Label

…jump and link

跳转到指定地址的同时，下一条指令的地址

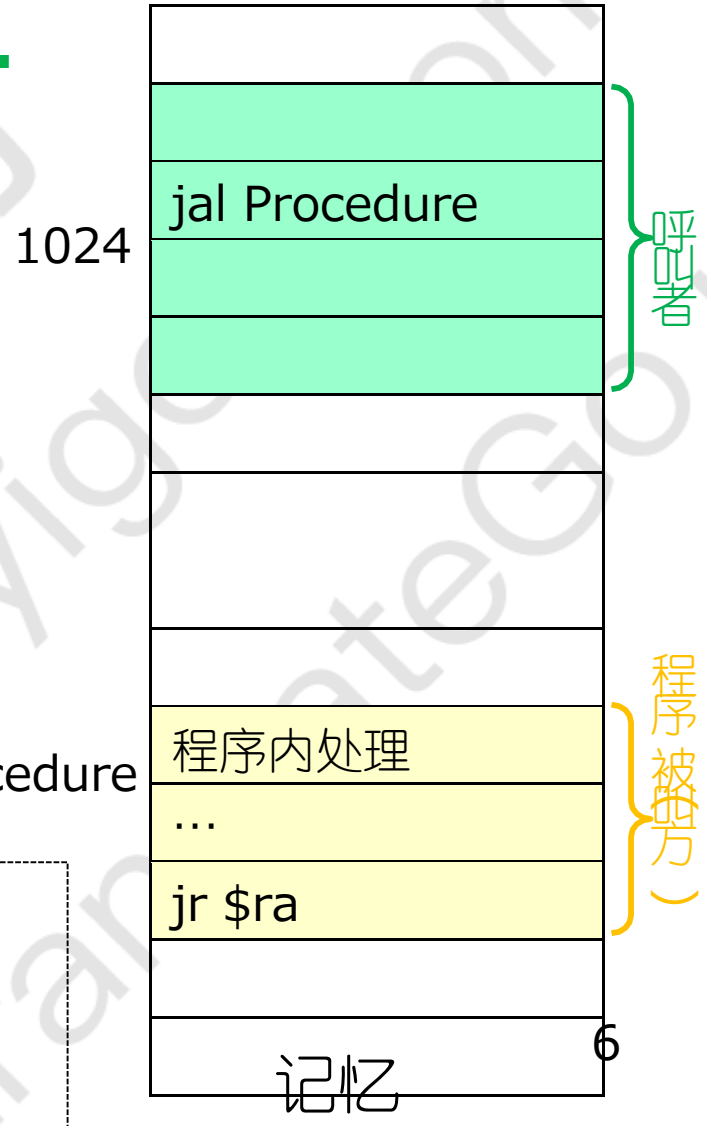
保存在寄存器 $ra（J 格式）

jr $ra

…jump register

存放在指定的寄存器中

跳转到内存地址（R格式）



在右边的例子中，地址 1024 处的 jal 指令

执行后，$ra 为 1028。

PC 将在 1024 之后成为程序。

执行程序　　如何

如果有 4 个或更多参数怎么

- 您需要做什么来执行该程序
  - 将参数的值放在程序可访问的地方
  - 将控制权移交给程序
  - 程序内　　处理所需的
    安全的

$a0-$a3

PC，$ ra 操作与 jal

如果有两个或多个返回值怎么办？

  - 执行程序的内部处理
  - 将结果（返回值）放在调用者可以访问的地方

$v0-$v1

如果我多次调用该过程怎么办？

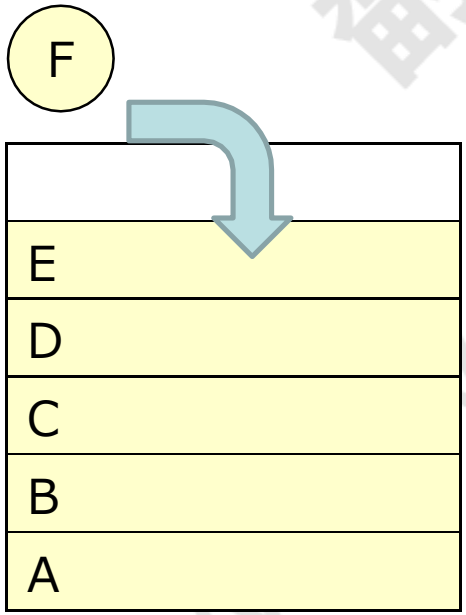
jr $ra

堆

- 检索稍后输入的数据的数据结构
  - LIFO (last in, first out)
- 您只能访问堆栈顶部的数据（=您放入的最后一个）

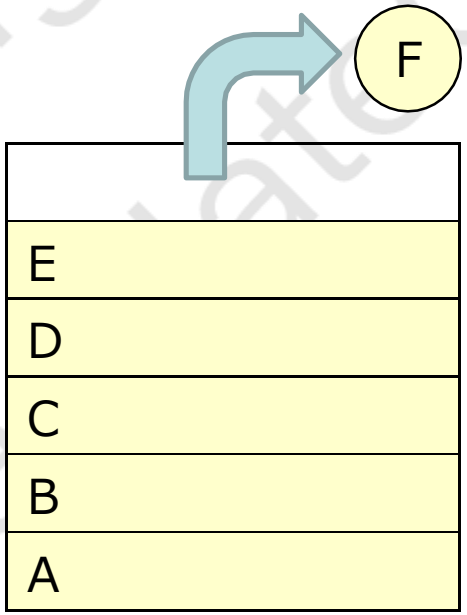
推：
保值

”栈上的F
也称为“堆栈”



流行音乐：
提取值

也称为“从堆栈中降低 F”

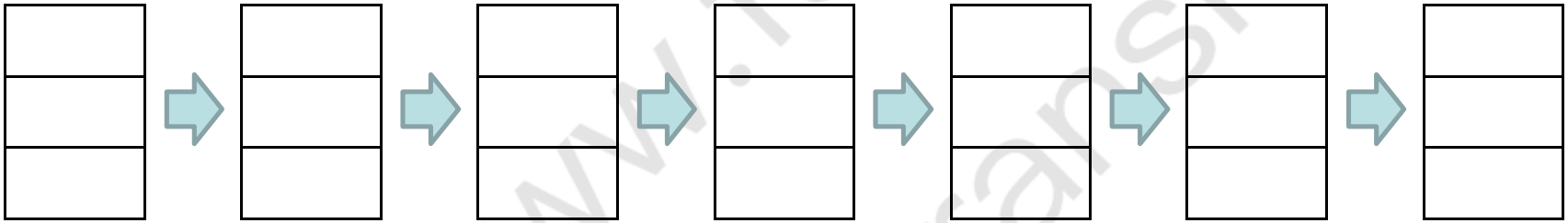


堆栈行为

- 考虑一个空栈。显示当您对此堆栈执行以下操作时堆栈如何变化。
  - (1) 推送数据A
  - (2) 推送数据 B
  - (3) 流行数据
  - (4) 推送数据 C
  - (5) 流行数据
  - (6) 流行数据

堆栈行为

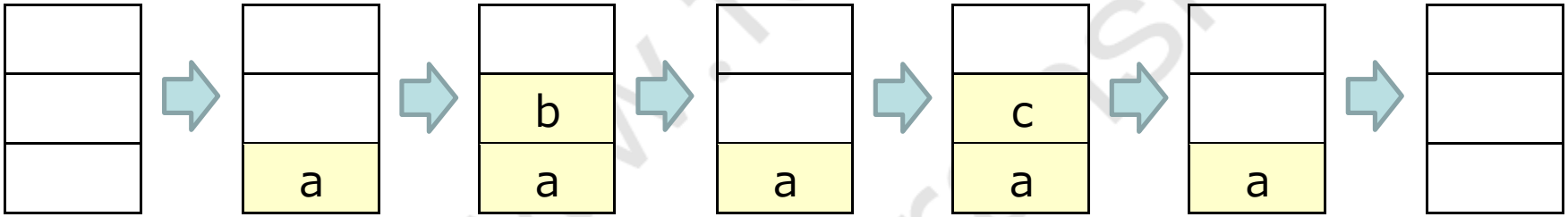
- 考虑一个空栈。显示当您对此堆栈执行以下操作时堆栈如何变化。
  - (1) 推送数据A
  - (2) 推送数据 B
  - (3) 流行数据
  - (4) 推送数据 C
  - (5) 流行数据
  - (6) 流行数据



过程调用与栈的关系

- 假设你想把当前运行的过程的名字放到栈上
  - (1) 推一个
  - (2) 推b
  - (3) 流行音乐
  - (4) 推c
  - (5) 流行音乐
  - (6) 流行音乐
  - * 省略了主要内容

```
int main(){
  a();
}

void a(){
  b();
  c();
}
```



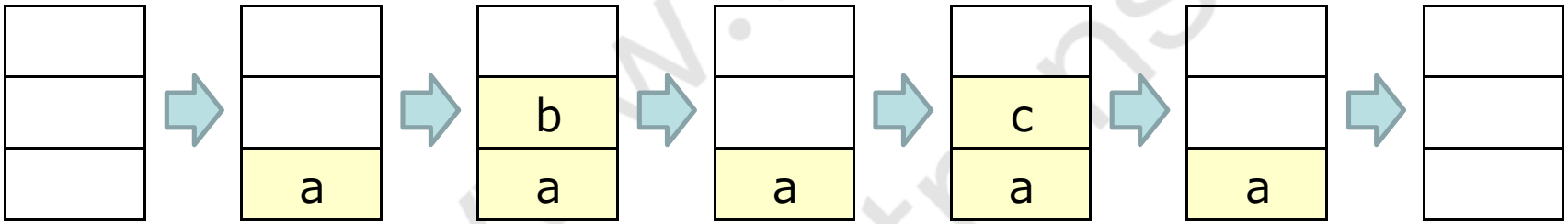
不适合寄存器的参数和返回值
堆叠在内存堆栈中（溢出）

过程调用与栈的关系

不调用其他过程的过程称为叶过程。
(在本例中，b() 和 c())

如果全部离开，则堆栈管理不是强制性的

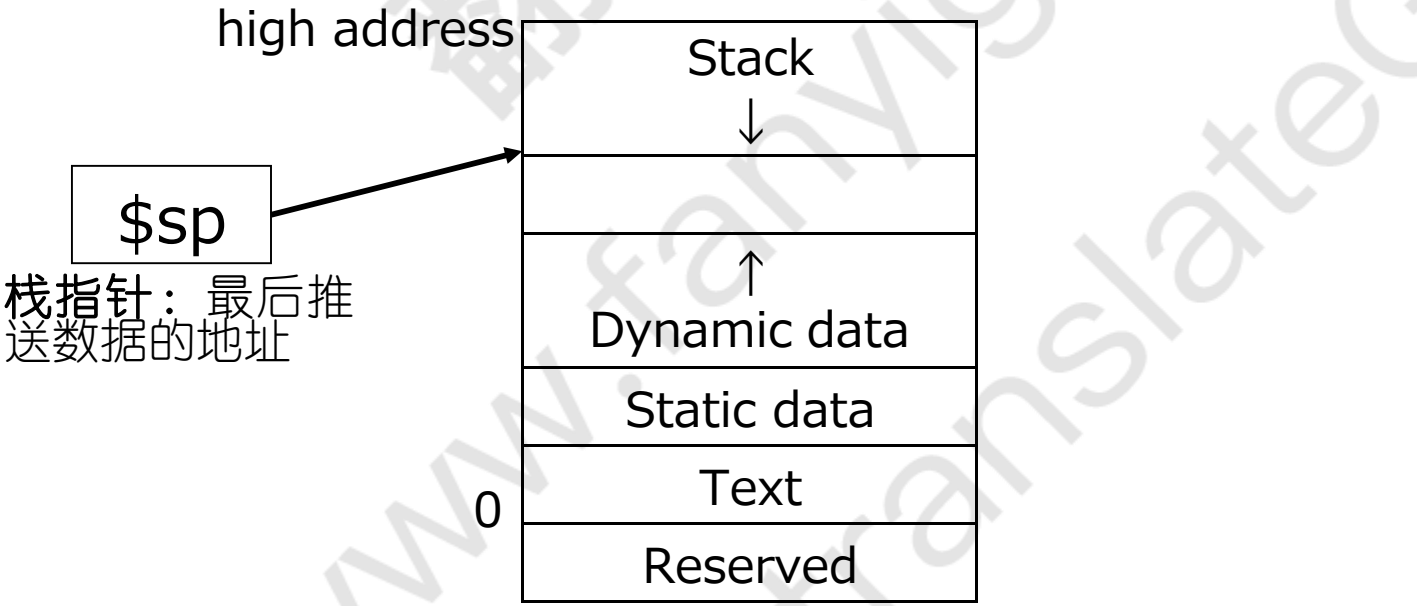
```
int main(){
  a();
}

void a(){
  b();
  c();
}
```



不适合寄存器的参数和返回值
堆叠在内存堆栈中（溢出）

栈的实现

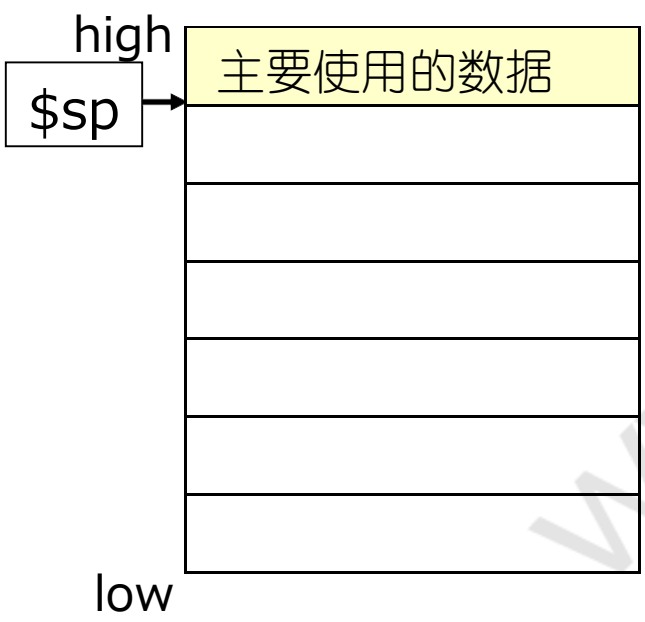
- 将栈的起始地址存放在寄存器$sp中，较大的内存地址是栈底。
- = 较小的内存地址是栈顶

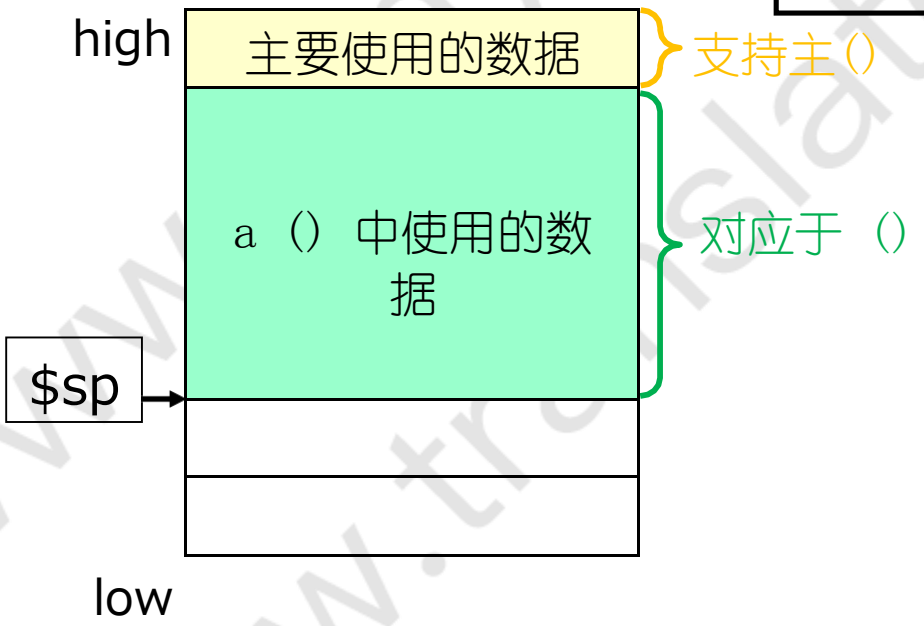


调用过程时的堆栈操作

```
①  int main(){
      a();
    }

    void
      a(){in
②     t x;
      b(x);
      c();
```

① ②

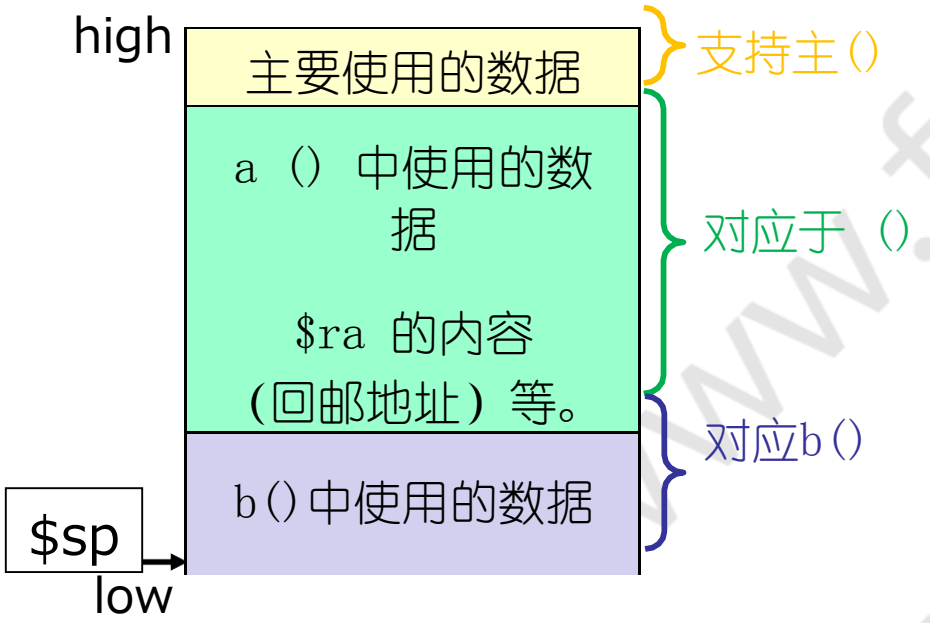




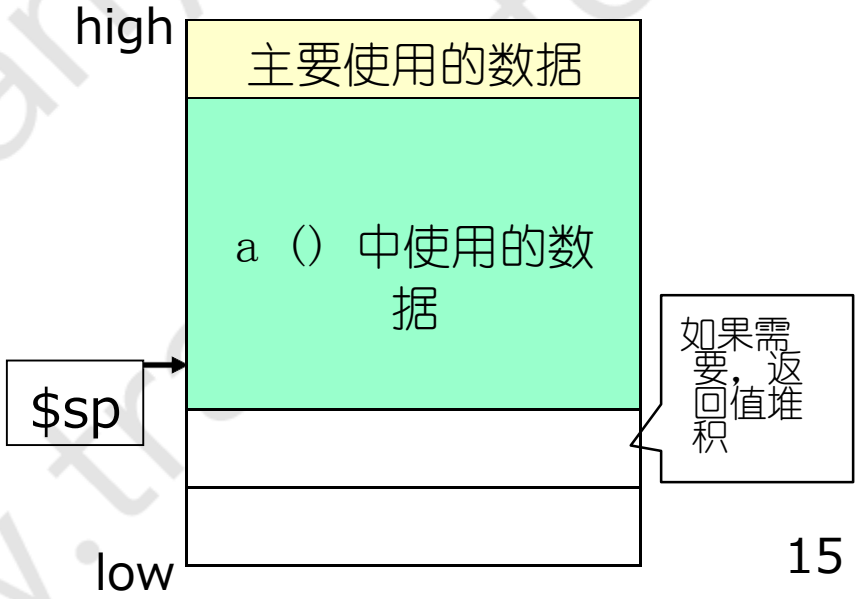
调用过程时的堆栈操作

```
int main(){
   a();
}

void
   a(){in
   t x;
   b(x);
   c();
```

③ （b() 正在内部执行）



④

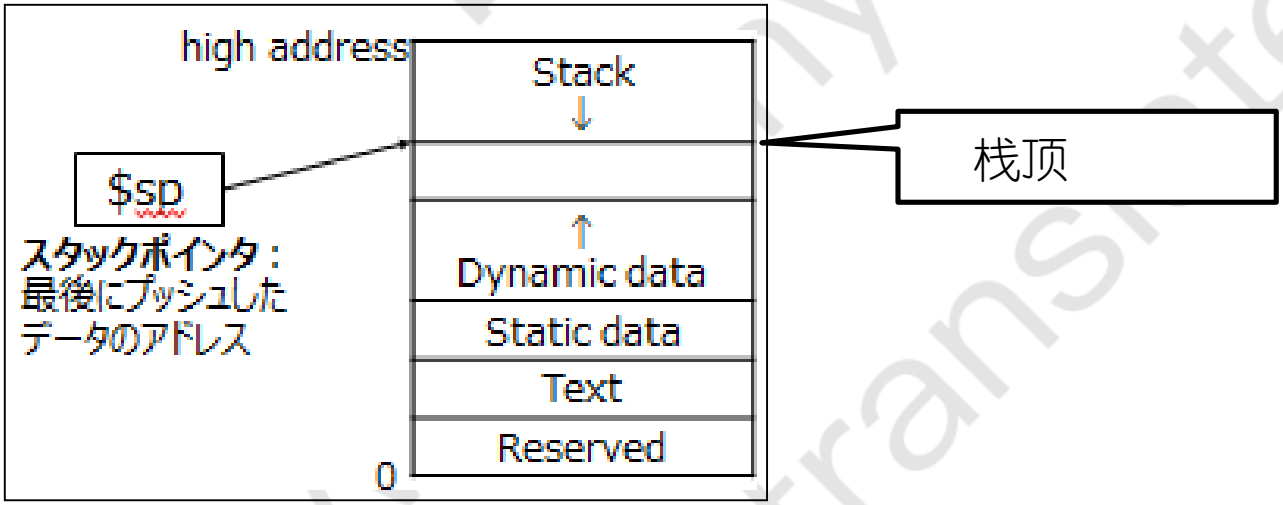


调用过程时的堆栈操作

- 当从过程 A 调用过程 B 时
  A中使用的寄存器中的数据必须保存在内存（堆栈）中。
  - 是否保存由寄存器决定
    - $ s0- $ s7（保存寄存器）、$ ra（返回地址）等。撤离
    - $t0-$t9（临时寄存器）不保存
    - 如果不覆盖就不用保存，调用B时，
  - 将上述寄存器的值压入堆栈。
  - 当 B 执行完毕时，弹出堆栈并将值返回到寄存器

调用过程时的堆栈操作

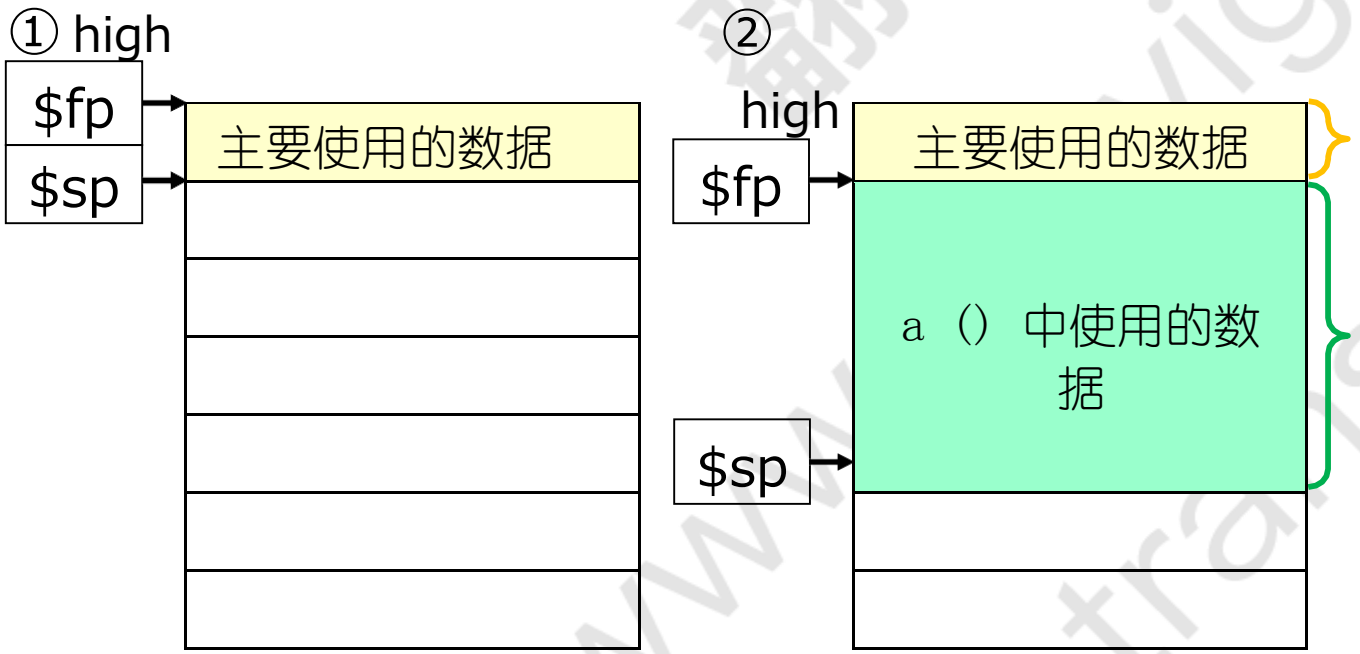
- 如何保存寄存器值
  - 先加上$sp的值（负数）
  - 将sw指令保存的寄存器内容写入内存（栈）



调用过程时的堆栈操作

转载图

- 程序框架
  - 调用过程时新分配的堆栈上的区域（也称为堆栈帧）
- 帧指针 $ fp
  - 指向帧的起始地址




```
①  int main(){
       a();
   }

   void
②     a(){in
       t x;
       b(x);
       c();
```

确认问题

编译以下 C 语言源代码后，MIPS 汇编代码会发生什么变化

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

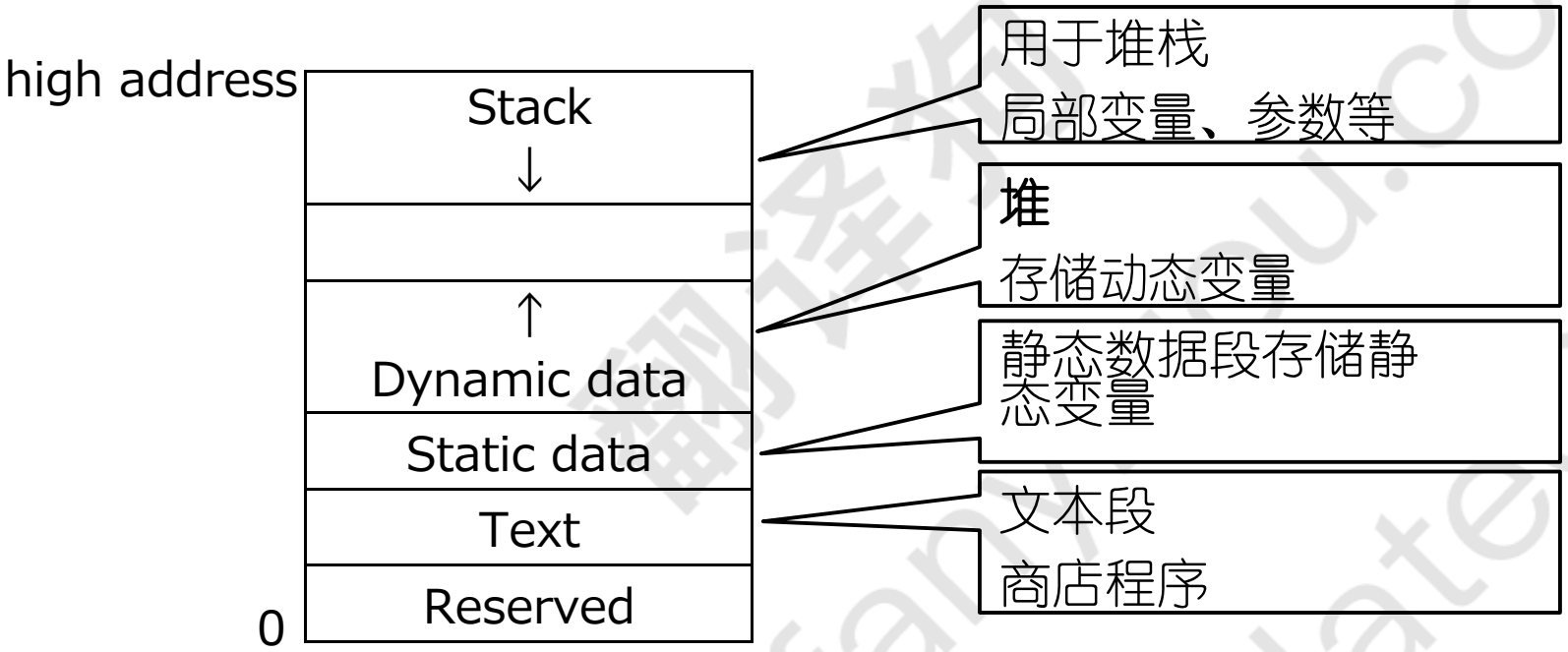
注册信件

f:$s0 g:$s1 h:$s2 i:$a0 j:$a1

```
leaf_example:
 addi $sp, $sp, (1)      # 3 字安全
 sw   $s2, 8($sp)        #疏散
 sw   $s1, 4($sp)        #疏散
 sw   $s0, 0($sp)        #疏散
 add  $s1, (2), (3)      #g=i+j
 sub  $s2, (2), (3)      #h=i-j
 add  $s0, $s1, $s2      #f=g+h
 add (4), $s0, $zero # 返回值设置
```

```
(5)   $ s0, 0 ($ sp) #注册恢复
(6)   $s1, 4($sp) #注册恢复
(7)   $ s2, 8 ($ sp) #注册恢复
(8)   $ sp, $ sp, (9) # 3 个字去掉
jr    (10)              #返回给调用者
```

变量和堆栈



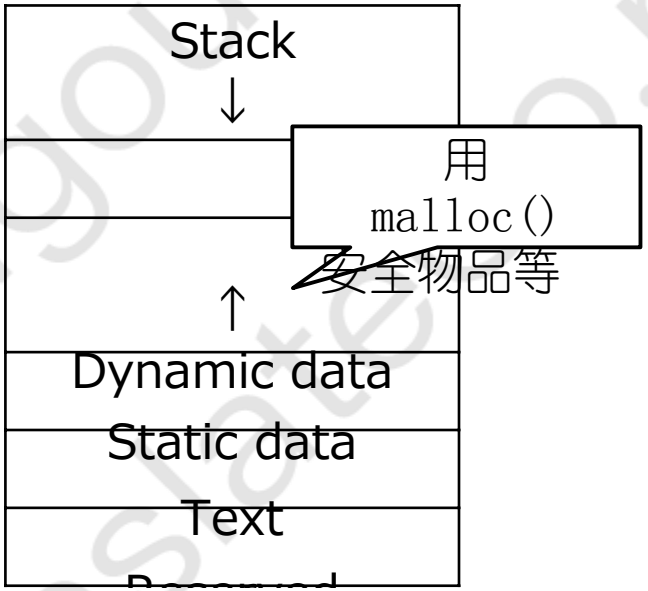
- 以下每个寄存器保存什么？
  - $ pc 当前执行指令的地址
  - $ gp 访问静态数据的参考地址
  - $ sp 当前栈上的最低地址
  - $ fp 当前程序帧的最高地址

过程调用之间保留的内容

- 举行
  - 保存寄存器 $s0-$s7 堆栈指针 $sp 帧指针 $fp
  - 全局指针 $gp 返回地址 $ra
- 未举行
  - 临时寄存器 $ t0- $ t9 参数寄存器
  - $ a0- $ a3
  - 返回寄存器 $ v0- $ v1

C语言变量存储类

- 自动变量（局部变量）
  - 只在方块内生存
  - 放置在声明范围内的可访问堆栈上
- 外部变量
  - 在整个程序中存活
  - 可从任何地方访问
  - 放置在静态数据段中
- 静态变量
  - 在整个程序中存活
  - 可从声明的范围内访问
  - 放置在静态数据段中
- * 注册变量
  - 分配给寄存器的自动变量

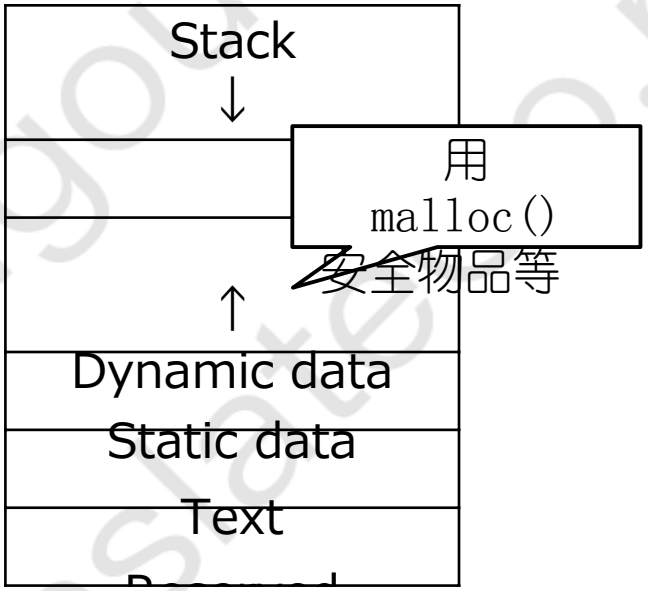


空闲内存

- 动态数据内存释放
  - C语言忘记释放 free()
  - * 内存泄漏
  - * 当内存泄漏增加时，没有可以保护的新区域
  - 过早发布
  - * 悬空指针

    （悬空指针）发生
- Java 等的垃圾收集

By（垃圾收集）



正确自动释放

确认问题

编译以下 C 语言源代码后，MIPS 汇编代码会发生什么变化

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

调用fact时，设置$a0、$ra的值
假设您需要撤离。
寄存器对应 n: $ a0

```
fact:
 addi $ sp, (1), (2) # 2 个字加密
 sw    $ra, 4($sp)        ##疏散
 sw    $a0, 0($sp)        #疏散 slti
 $ t0, $ a0, 1            # n <1 ?
 beq $ t0, $ zero, (3) #branch addi
 $ v0, $ zero, 1 #返回值设置
 addi $ sp, (4), (5) # 删除2个单词
 jr    (6)                #return
```

```
L1:
 addi $ a0, $ a0, -1 # 参数 n-1
 jal (7)                  #fact 调用
 lw    $a0, 0($sp)        #注册恢复lw
       $ra, 4($sp)        #Register
 restore addi $ sp, (8), (9) # 删
除2个字
 mul   $v0, $a0, $v0 #n*fact(n-1)
 jr    (10)               #return
```

参考

- Computer Configuration and Design 5th Edition by David A. Patterson, John L. Hennessy, 成田光明翻译, Nikkei BP
- Shigeru Yamashita"计算机组成理论1"讲义