**8.3 递归**

递归子程序(recursive subrountine) 是指直接或间接调用自身的子程序。递归，调用递归子程序的做法，在处理具有重复模式的数据结构时，它是一个强大的工具。例如链表和各种类型的连接图，这些情况下，程序都需要追踪其路径。

**无限递归** 子程序对自身的调用是递归中最而易见的类型。例如，下面的程序包含一个名为Endless的过程，它不断重复调用自身：

Endless.asm

当然，这个例子没有任何实用价值。每次过程调用自身时，它会占用4字节的堆栈空间让CALL指令将返回地址人栈。RET指令水远不会被执行，仅当堆校溢出时，程序终止。

**8.3.1递归求和**

实用的递归子程序总是包含终止条件。当终止条件为真时，随着程序执行所有挂起的RET指令，堆栈展开，举例说明，考虑一个名为CalcSum的递归过程，执行整数1到n的加法，其中n是通过ECX 传递的输人参数。CaleSum用EAX返回和数:

RecusiveSum.asm

CalcSum的开始两行检查计数值，若ECX=0则退山该过程，代码就跳过了后续的递归调用。当第一次执行RET指令时，它返回到前一次对CalcSum 的调用，而这个调用再返回到它的前一次调用，依序前推。下给出了CALL指令用压入堆栈的返回地址（用标号表示），以及与之相应的ECX (计数值) 和EAX (和数) 的值。

|  |  |  |
| --- | --- | --- |
| 入栈的返回地址 | ECX的值 | EAX的值 |
| L1 | 5 | 0 |
| L2 | 4 | 5 |
| L2 | 3 | 9 |
| L2 | 2 | 12 |
| L2 | 1 | 14 |
| L2 | 0 | 15 |

即使是一个简单的递归过程也会使用大量的堆栈空间。每次过程调用发生时最少占用4字节的堆栈空间，因为要把返回地址保存到堆栈。

**8.3.2计算阶乘**

递归子程序经常用堆栈参数来保存临时数据。当递归调用展开时，保存在堆栈中的数据就有用了。下面要查看的例子是计算整数n的阶乘。阶乘算法计算n!，其中n是无符号整数。第一次调用factorial 函数时，参数n就是初始数字。下面给出的是用C/C++/Java语法编写的代码:

int function factorial(int n)

{

if(n == 0)

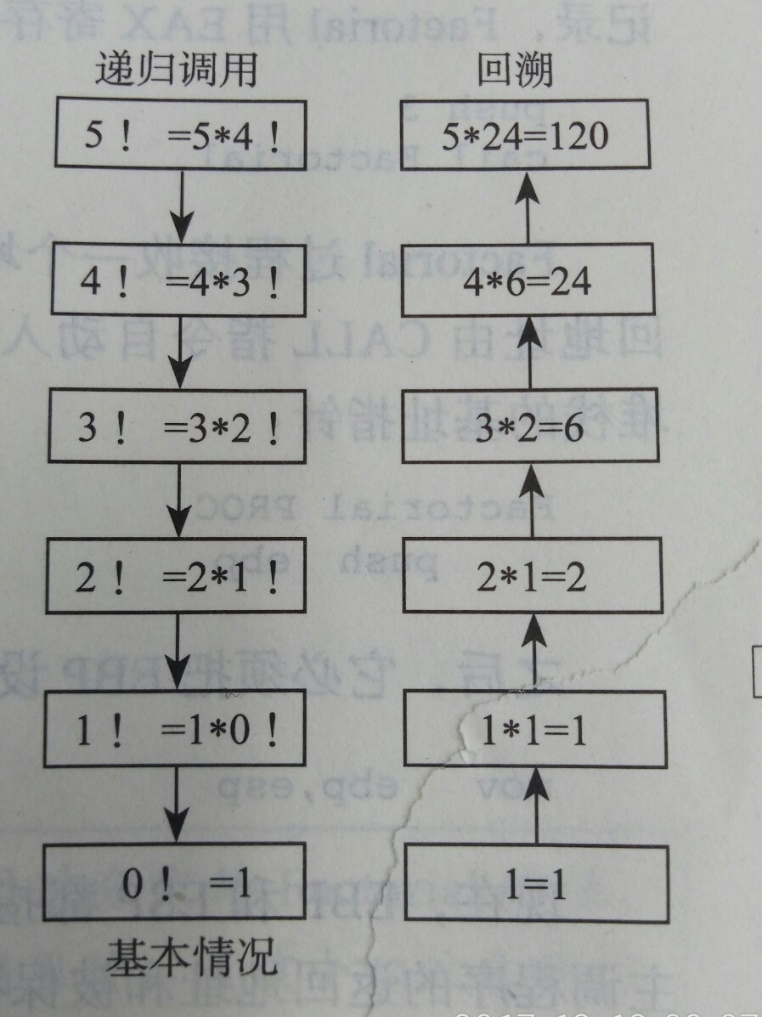
return 1;

e1se

return n\*factorial (n一1) ;

}

假设给定任意n，即可计算n-1的阶乘。这样就可以不断减少n,直到它等于0为止。根据定义，0！=1。而回溯到原始表达式n! 的过程，就会累积每次的乘积。比如计算5!的递归算法如图所示，左列为算法递进过程，右列为算法回溯过程。



**示例程序**

下面的汇编语言程序包含了过程Fatorial，递归计算阶乘。通过堆栈把n (0~ 12之间的无符号整数) 传递给过程Fatoral，返回值在EAX中。由于EAX是32位寄存器，因此，它能容纳的最大阶乘为12！(479 001 600)。

Fact.asm

**EBP+8**

**N=3**

**（返回地址）**

**（EBP）**

**ESP，EBP**

**N=3**

**（返回地址）**

**（EBP）**

**N=2**

**（返回地址）**

**（EBP）**

**N=1**

**（返回地址）**

**（EBP）**

**EBP+8**

**ESP，EBP**

**N=3**

**（返回地址）**

**（EBP）**

**N=2**

**（返回地址）**

**（EBP）**

**EBP+8**

**ESP，EBP**

**（EBP）**

**（返回地址）**

**N=1**

**（EBP）**

**（返回地址）**

**N=2**

**（EBP）**

**N=3**

**（返回地址）**

**EBP+8**

**ESP，EBP**

**（EBP）**

**（返回地址）**

**N=0**