汇编语言程序设计

第六章　子程序设计

张华平　副教授　博士

Email：kevinzhang@bit.edu.cn

Website: http://www.nlpir.org/

@ICTCLAS张华平博士

大数据搜索与挖掘验室（wSMS@BIT）

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

为了程序共享或模块化设计的需要，可以把一段公共语句序列设计成子程序或宏指令的形式。本章介绍子程序的设计方法。

6.1 子程序结构及设计方法

一、子程序结构

在汇编语言中用过程定义伪指令定义子程序。过程定义伪指令格式：

过程名　　PROC　属型

…

过程名　　ENDP

其中过程名就是子程序名，它也表示子程序入口的符号地址。

属型可以是NEAR型（缺省值）或FAR型。NEAR型子程序只可以被段内调用，而FAR型子程序可以被段间或段内调用。

1. 调用程序和子程序在同一个代码段的程序结构

```
CODE        SEGMENT
  MAIN      PROC  FAR
            …
            CALL  SUB1
            RET
  MAIN      ENDP
  SUB1      PROC        ;类型可缺省
            …
            RET
  SUB1      ENDP
CODE  ENDS          ;END后跟主程序名
            END       MAIN
```

2. 调用程序和子程序在不同段的程序结构

```
CODE1   SEGMENT
MAIN        PROC   FAR
            CALL   FAR   PTR   SUB2
            RET
MAIN        ENDP
CODE1       ENDS
CODE2       SEGMENT
SUB1        PROC   FAR
            CALL   FAR   PTR   SUB2    ;CALL SUB2
            RET
SUB1        ENDP
SUB2        PROC   FAR
            RET
SUB2        ENDP
CODE2       ENDS
            END    MAIN
```

二、设计子程序时应注意的问题

1. 子程序说明

为便于引用，子程序应在开头对其功能、调用参数和返回参数等予以说明，例如参数的类型、格式及存放位置等。

2. 寄存器的保存与恢复

为了保证调用程序的寄存器内容不被破坏，应在子程序开头保存它要用到的寄存器内容，返回前再恢复它们。

3. 注意堆栈状态

在设计含有子程序的程序时，要密切注意堆栈的变化。这包括要注意一切与堆栈有关的操作。例如CALL调用类型和子程序定义类型的一致性，PUSH和POP指令的匹配，通过堆栈传递参数时子程序返回使用RET n指令等，以确保堆栈平衡。

- 所谓堆栈，就是供程序使用的一块连续的内存空间，一般用于保存和读取临时性的数据。

6.2.1 堆栈特点

- 1. 临时性
- 2. 快速性
- 3. 动态扩展性

6.2.2 堆栈用途

- 1. 保护和恢复调用现场
- 2. 用于变量之间的数据传递
- 3. 用做临时的数据区
- 4. 子程序的调用和返回

6.2.3 子程序的返回地址

- 例. 段内调用和返回
- 设计两个子程序：第1个子程序AddProc1使用ESI和EDI作为加数，做完加法后把和放在EAX中；第2个子程序AddProc2使用X和Y作为加数，做完加法后把和放在Z中。主程序先后调用两个子程序，最后将结果显示出来。
- 在AddProc2中用到了EAX，所以要先将EAX保存在堆栈中，返回时再恢复EAX的值。否则EAX中的值会被破坏。
- 见程序PROG0603.ASM 。

6.3 子程序参数传递

可以通过给子程序传递参数使其更通用。常用的参数传递方法如下：

通过寄存器传递；

子程序直接访问模块中变量（同一模块）；

通过地址表传递参数地址；

通过堆栈传递参数或参数地址。

6.3.1 C语言函数的参数传递方式

- 在C/C++以及其他高级语言中，函数的参数是通过堆栈来传递的。C语言中的库函数，以及Windows API等也都使用堆栈方式来传递参数。

C函数常见的有5种参数传递方式（调用规则）见下表。

调用规则	参数入栈顺序	参数出栈	说　明
cdecl方式	从右至左	主程序	参数个数可动态变化
stdcall方式	从右至左	子程序	Windows API常使用
fastcall方式	用ECX、EDX传递第1、2个参数，其余的参数同stdcall，从右至左	子程序	常用于内核程序
this方式	ECX等于this，从右至左	子程序	C++成员函数使用
naked方式	从右至左	子程序	自行编写进入/退出代码

1．通过寄存器传递

这种传递方式使用方便，适用于参数较少的情况。

例．把BX中的16位二进制数转换成十进制并显示在屏幕上。

分析：采用从高到低逐个除以十进制位权的方法。（见程序6.3）

↗2．若调用程序和子程序在同模块中，子程序可以直接访问模块中的变量

例．实现数组求和功能。要求数组求和（不考虑溢出情况）由子程序实现，其数组元素及结果均为字型数据。见程序6.4。

3. 通过地址表传递参数地址

适用于参数较多的情况。具体方法是先建立一个地址表，该表由参数地址构成。然后把表的首地址通过寄存器或堆栈传递给子程序。

例. 编写一个数组求和子程序，其数组元素及结果均为字型数据。另定义两个数组，并编写一个主程序，通过调用数组求和子程序分别求出两个数组的和。

分析：虽然主、子程序在同模块中，但由于在一个程序中要分别求出两个数组的和，因此子程序不能直接引用数组变量名。

本例用数组首地址、元素个数的地址、和地址构成地址表，通过地址表传送这些参数的地址，以便子程序能够访问到所需参数。

见程序6.5。

4. 通过堆栈传递参数或参数地址

这种方式适用于参数较多，或子程序有多层嵌套、递归调用的情况。

步骤：

主程序把参数或参数地址压入堆栈；

子程序使用堆栈中的参数或通过栈中参数地址取到参数；

子程序返回时使用RET n指令调整SP指针，以便删除堆栈中已用过的参数，保持堆栈平衡，保证程序的正确返回。

例．完成数组求和功能，求和由子程序实现，要求通过堆栈传递参数地址。

（见程序6.6）

本例通过BP访问堆栈中的参数。

程序的堆栈变化情况参见图6-1，指示了程序中所有入栈操作对堆栈的影响随入栈数据的增加，SP的值不断减小，堆栈可用空间也随之减少。图6-2为已从子程序返回、而主程序RET指令执行前的堆栈状态，其中的灰色部分表示执行语句(12)~(17)时已弹出的数据。

随着弹出数据的增加，SP的值不断增大，堆栈可用空间也随之增大。子程序中语句(17)——RET 6指令，在从堆栈弹出返回地址后还要使SP值加6，这样就跳过了通过堆栈传递的三个参数，或者说删除了它们。

因此，当主程序的语句(18)——RET指令被执行时，程序控制从栈顶弹出数字0给IP，弹出PSP的段基址给CS，于是执行PSP:0处的INT 20H指令，正确返回操作系统。

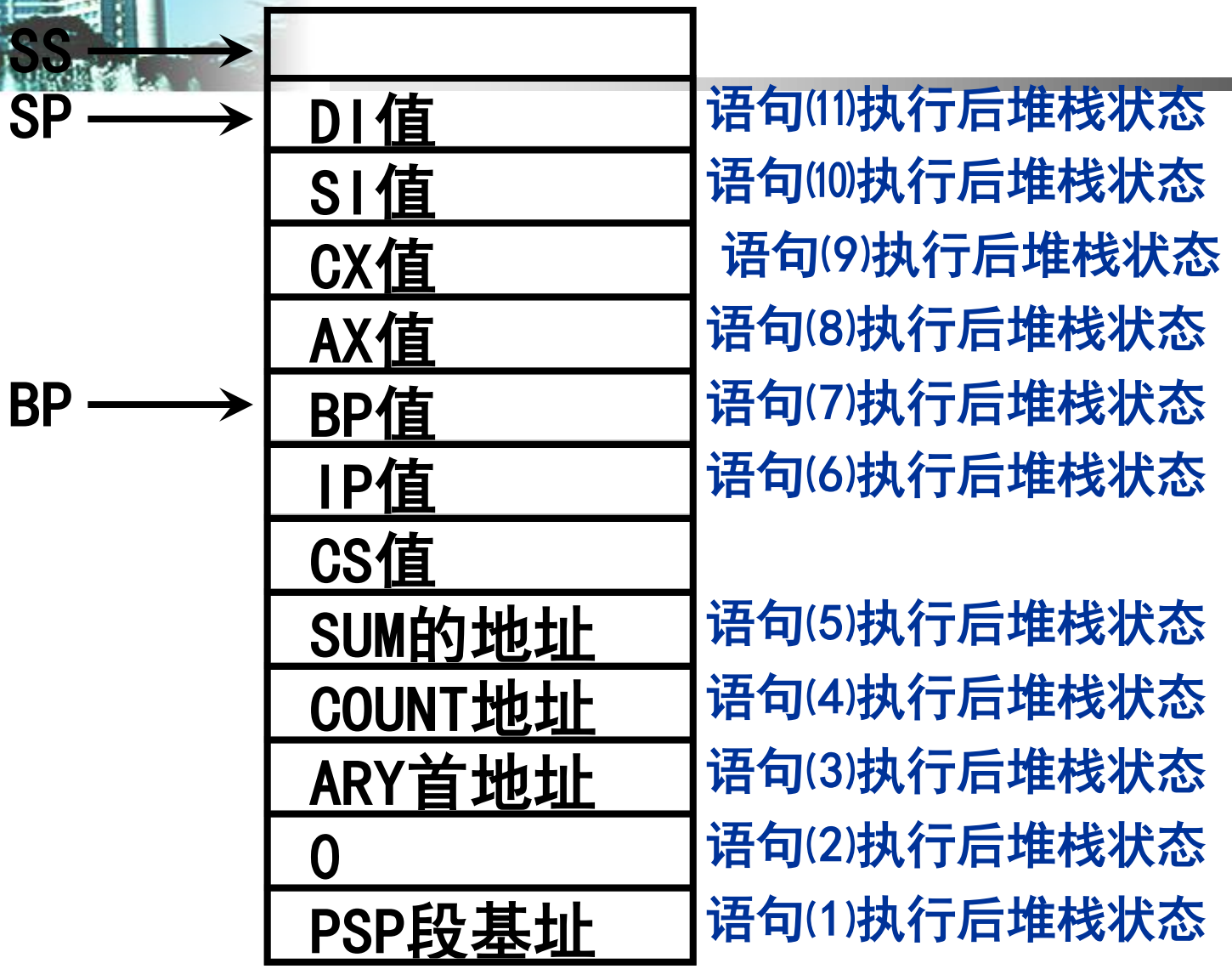


图6-1　程序6.6中所有入栈操作对堆栈的影响

返回

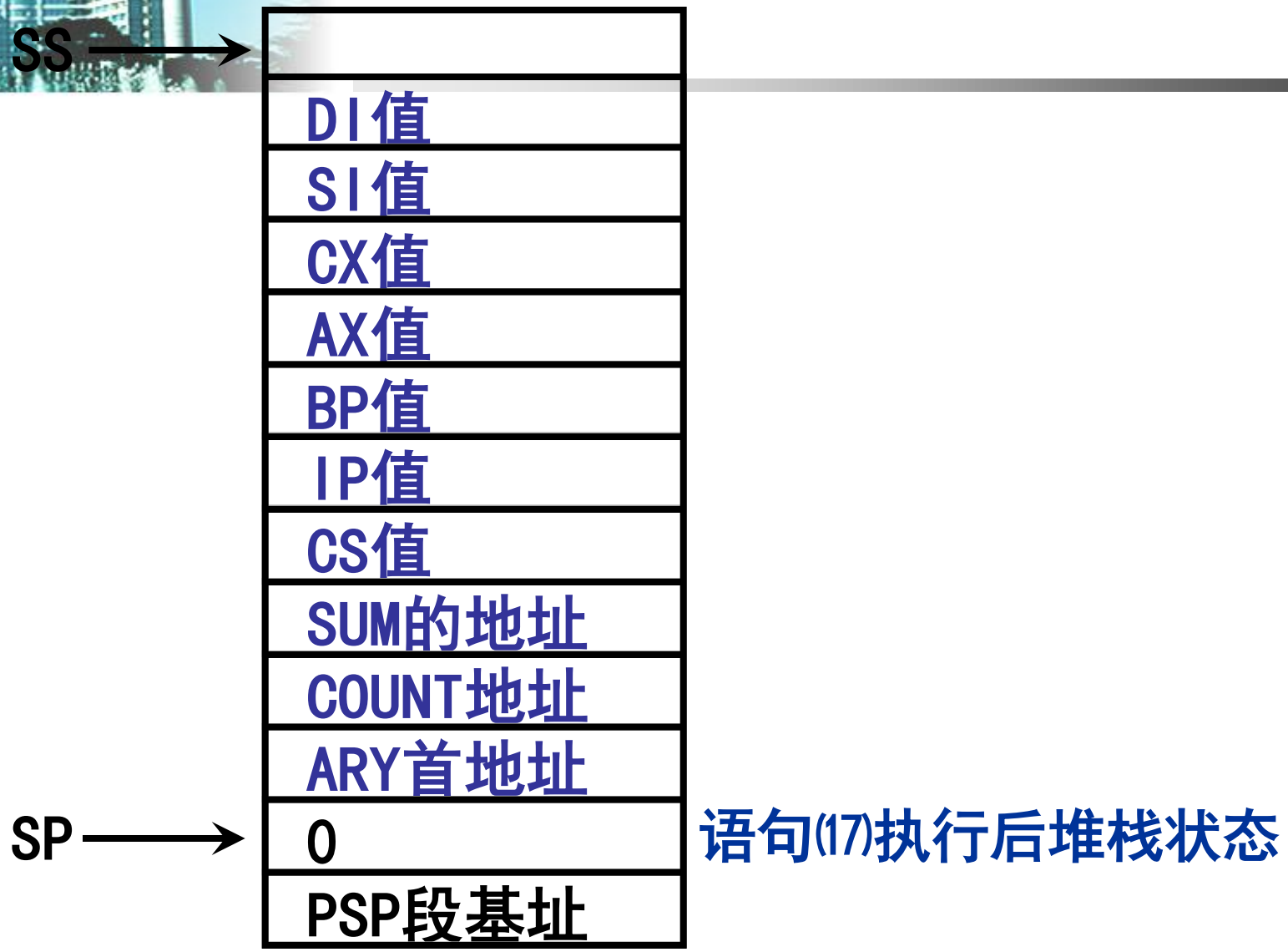


图6-2　程序6.6中主程序的RET执行前堆栈状态

返回

从以上分析可以看出，通过堆栈传递参数时子程序的返回指令必须是RET N形式，当堆栈操作是16位时N值应该是压入堆栈的参数个数的2倍，只有这样保证程序的正常运行。

用结构形式访问堆栈中的参数，这种方法更简便及规范化。

例．完成数组求和功能，其中求和由子程序实现，要求使用结构访问堆栈中的参数。

图6-3给出了堆栈及结构数据定义。注意这些结构字段的顺序为其值压入的逆序。实际上，它只是给图6-1中由主程序压入的数据、返回地址及子程序压入的BP值起了个名字而已，而字段值的预置是通过PUSH和CALL指令实现的。当子程序用到堆栈中的参数时，只需使用BP作为基地址、通过结构字段名访问就可以了。编码见程序6.7。

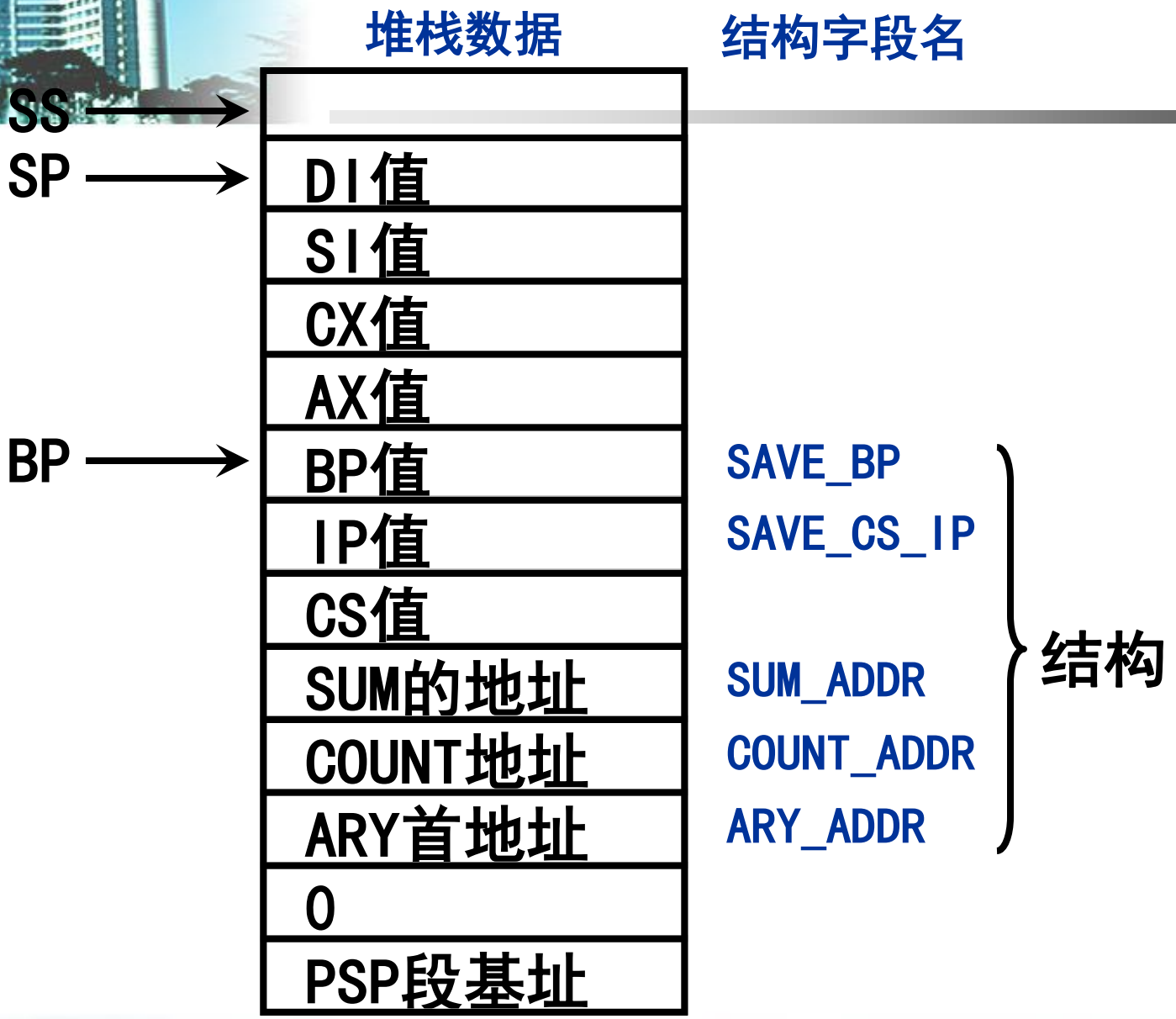


图6-3　程序6.7的堆栈及结构数据示意图

返回

6.4 嵌套与递归子程序

一、子程序嵌套

在汇编语言中，允许子程序作为调用程序去调用另一子程序，把这种关系称为子程序嵌套。

图6-4为子程序嵌套示意图。嵌套的层数没什么限制，其层数称为嵌套深度。

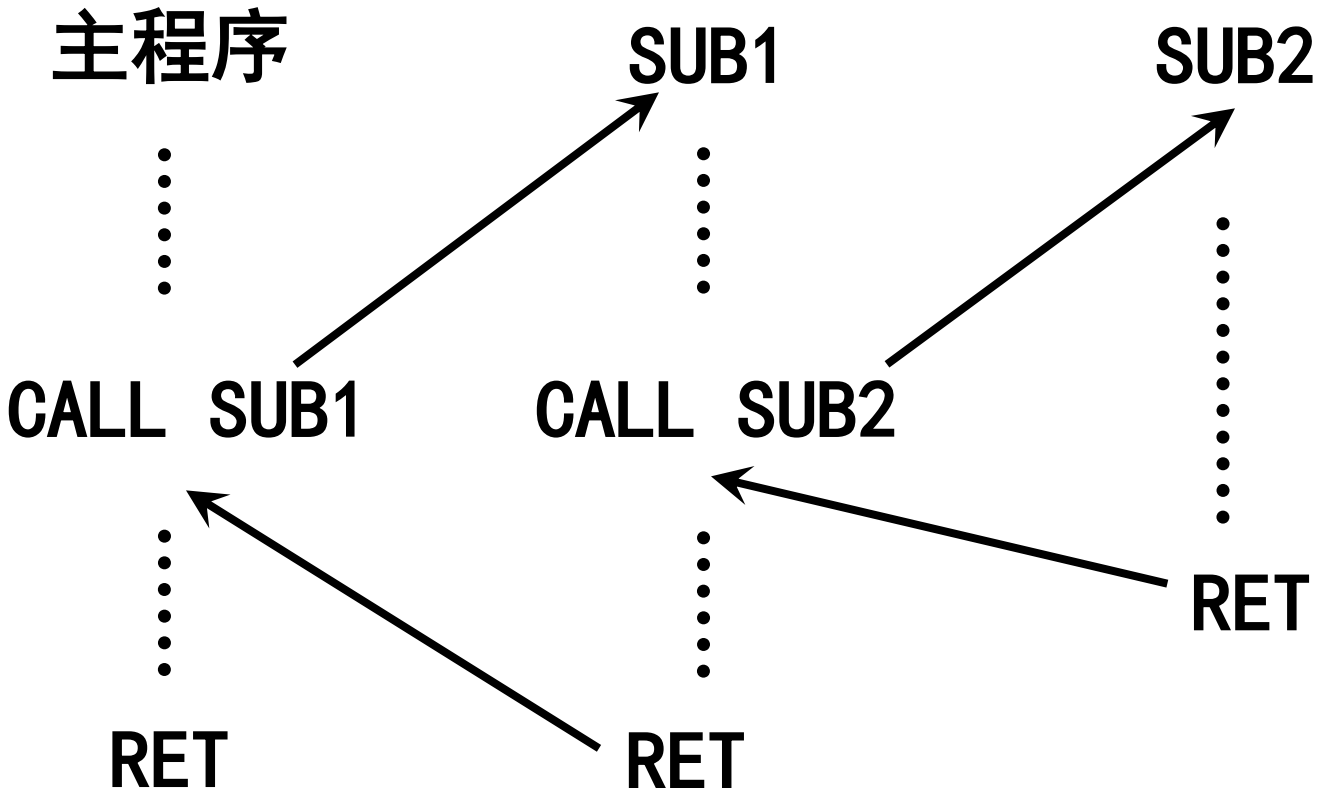


图6-4　子程序嵌套示意图

返回

由于子程序嵌套对堆栈的使用很频繁，因此还要确保堆栈有足够空间，并要注意堆栈的正确状态，这包括CALL、RET、RET N、PUSH、POP、INT、IRET等与堆栈操作有关指令的正确使用。子程序嵌套举例见程序6.3，嵌套深度2。

6.5 缓冲区溢出攻击原理

- 缓冲区溢出是目前最常见的一种安全问题，操作系统以及应用程序一般都存在缓冲区溢出漏洞。缓冲区溢出是由编程错误引起的，当程序向缓冲区内写入的数据超过了缓冲区的容量，就发生了缓冲区溢出，缓冲区之外的内存单元被程序“非法”修改。
- 一般情况下，缓冲区溢出会导致应用程序的错误或者运行中止，但是，攻击者利用程序中的漏洞，精心设计出一段入侵程序代码，覆盖缓冲区之外的内存单元，这些程序代码就可以被CPU所执行，从而获取系统的控制权。

6.5.1 堆栈溢出

在一个程序中，会声明各种变量。静态全局变量位于数据段并且在程序开始运行时被初始化，而局部变量则在堆栈中分配，只在该函数内部有效。

如果局部变量使用不当，会造成缓冲区溢出漏洞。

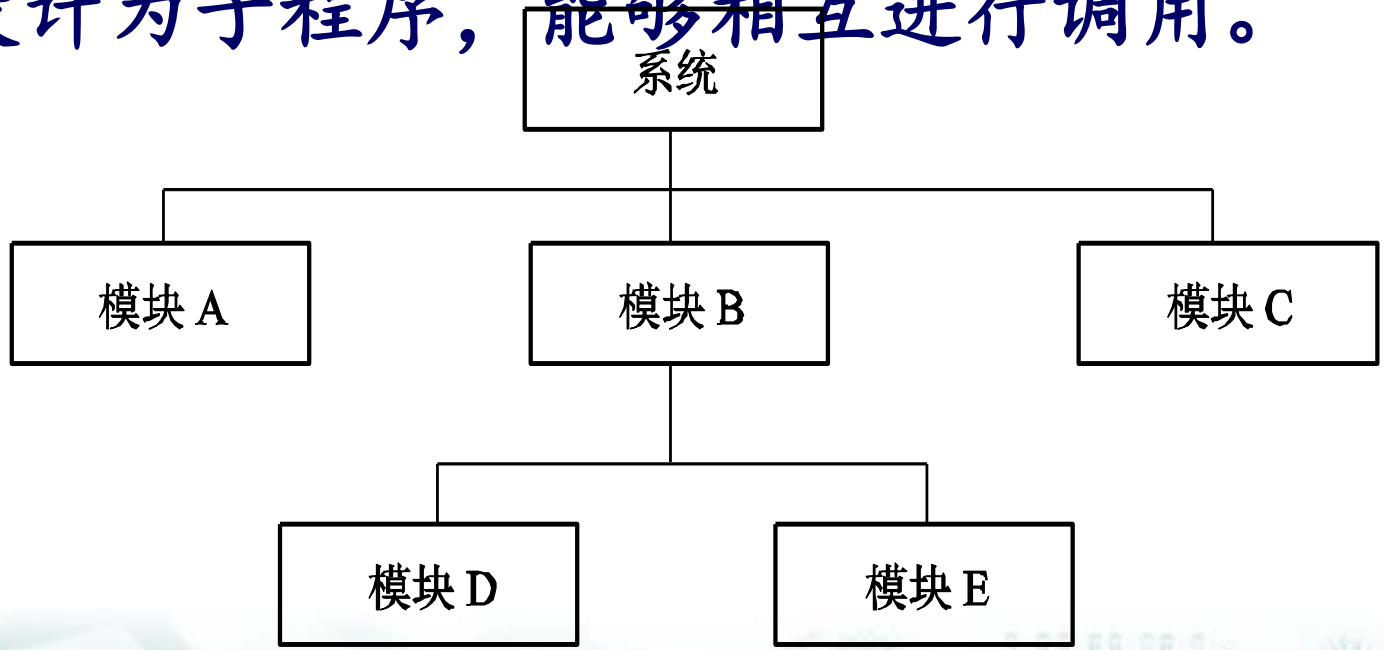
6.5.2 数据区溢出

当变量或数组位于数据区时，由于程序对变量、数组的过度使用而导致对其他数据单元的覆盖，也可能导致程序执行错误。

6.6 模块化程序设计

- 如果有多个源程序文件，或者需要使用C/C++、汇编等多种语言混合编程，就需要对这些源程序分别编译，最后连接构成一个可执行文件。

如图所示，系统由模块A、模块B、模块C组成，而模块B中的部分功能又可以进一步分解成为模块D、模块E，整个系统包括了5个模块。模块中的代码设计为子程序，能够相互进行调用。



- 在子程序设计中，主程序和子程序之间可以通过全局变量、寄存器、堆栈等方式传递数据，这种技术在模块化程序设计中同样适用。

6.6.2 模块间的通信

- 由于各个模块需要单独汇编，于是就会出现当一个模块通过名字调用另一模块中的子程序或使用其数据时，这些名字对于调用者来讲是未定义的，因此在汇编过程中就会出现符号未定义错误。可以通过伪指令EXTRN、PUBLIC等来解决。

- 1. 外部引用伪指令EXTRN
- 格式：EXTRN　变量名:类型 [,...]
- 功能：说明在本模块中用到的变量是在另一个模块中定义的，同时指出变量的类型。

- 2. 全局符号说明伪指令PUBLIC
  - 格式：PUBLIC 名字 [,...]
  - 功能：告诉汇编程序本模块中定义的名字可以被其他模块使用。这里的名字可以是变量名，也可以是子程序名。

- 3. 子程序声明伪指令PROTO
- 格式：子程序名 PROTO [C | stdcall] :[第一个参数类型] [,:后续参数类型]
- 功能：说明子程序的名字和参数类型，供主程序调用。在前面的程序中，已经多次使用这种方式调用C语言的库函数及Windows的API。

程序1a 主模块PRICEM.ASM

```
EXTRN   SUBM:FAR
PUBLIC  PRICE,QTY,TOTAL
stacksg segment stack 'stk'
        dw 32 dup('s')
stacksg ends
data    segment
PRICE   DW 60
QTY     DW 80
TOTAL   DD ?
data    ends
code    segment
main    proc far
        assume cs:code,ds:data
        mov ax,data
        mov ds,ax
        CALL FAR PTR SUBM
        mov ax,4c00h
        int 21h
main    endp
code    ends
        end main
```

程序1b 子模块PRICES.ASM

```
EXTRN   PRICE:WORD,QTY:WORD,TOTAL:DWORD
PUBLIC  SUBM
CODE    SEGMENT
SUBM    PROC FAR
        ASSUME CS:CODE
               MOV AX,PRICE
        MUL QTY
        MOV word ptr TOTAL,AX
        MOV word ptr TOTAL+2,DX
        RET
SUBM    ENDP
CODE    ENDS
          END
```

返回

感谢关注聆听！

张华平
Email: kevinzhang@bit.edu.cn
微博：@ICTCLAS张华平博士
实验室官网：
http://www.nlpir.org

大数据千人会