第七讲　堆栈与子程序设计

一、堆栈（P11）

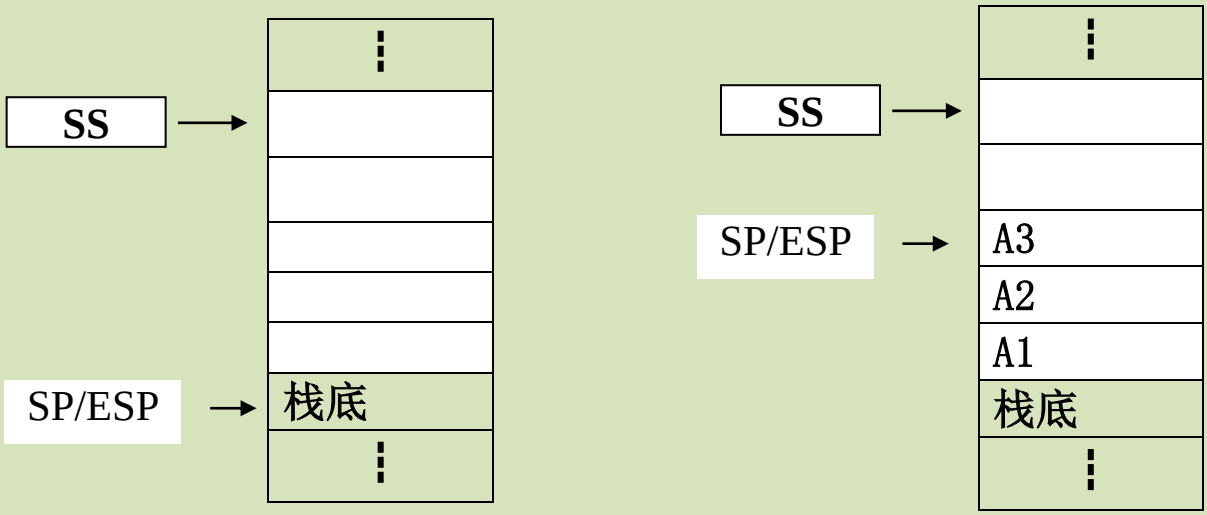
（1） 堆栈是在主存中的一片存储区，用于临时堆放一些数据。

（2） 存取方式是一端固定（固定端叫栈底），一端活动（叫栈顶）。对堆栈中数据的存取原则是“先进后出”。

（3） 堆栈中的数据也称元素。栈元素进栈称压入，出栈称弹出。

（4） 栈指针 SP/ESP 总是指向栈顶的偏移地址。

(5) 每次存取数的字长只能是一个字或双字。



空栈/堆栈溢出。

1. 进栈指令 PUSH

格式：PUSH　OPS

功能：将寄存器、立即数或存储单元中的一个字/双字数据压入堆栈。

即，（SP/ESP） － 2/4→SP/ESP；（OPS）→SS：SP/ESP 指向的单元。

例 1：PUSH　AX

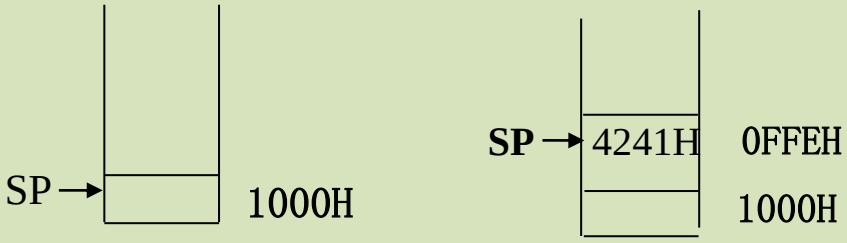
设：（AX）=4241H　（SP）=1000H

执行：①（SP）-2→SP

②（AX）→[SP]

用符号表示为：（AX）→↓（SP）。该语句执行前与执行后堆栈的状态如下图所

示：



操作结束后，堆栈段中偏移地址为 0FFEH 单元中的内容为 4241H，即（0FFEH）=4241H，也可写为：（[SP]）=4241H

2. 出栈指令 POP

格式：POP　OPD

功能：将栈顶元素弹出送至某一寄存器（除段寄存器 CS、IP/EIP 外）或字/双字存贮单元中。

即，SS：[SP/ESP]指向单元内容→OPD；

（SP/ESP） + 2/4→SP/ESP。

（POP 之后的痕迹，反跟踪）

二、子程序设计（P128）

循环程序:

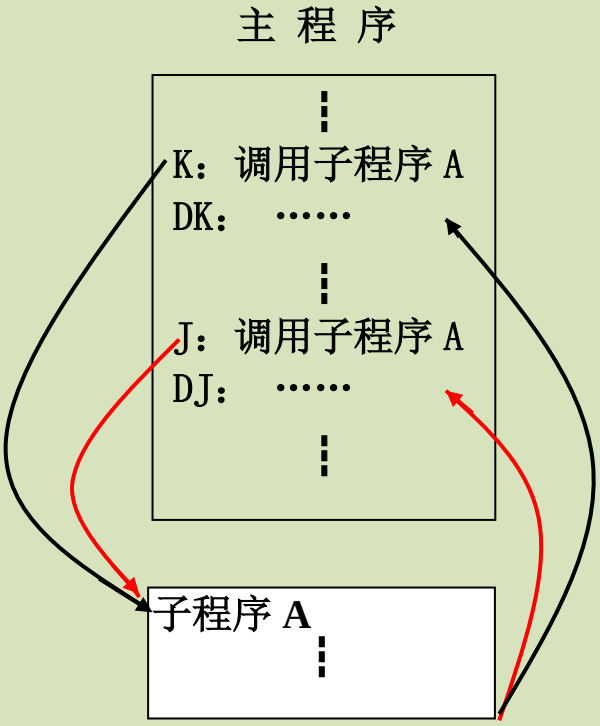
连续有规律重复的程序段.

子程序:

无规律重复的程序段.

(是模块化设计的基础)

均需要相应的控制措施.



主程序和子程序关系示意图

子程序设计中要解决的问题:

(1)如何编制子程序？(定义)

(2)主程序是怎样调用子程序的？

(3)子程序执行完之后，怎样才能返回到主程序的断点处继续往下执行？

(4) 当主程序与子程序使用相同的寄存器时，应如何处理？保护/恢复

(5)主程序每次调用子程序时提供给子程序加工的数据往往是不同的。主程序怎样把这些数据传送给子程序，而子程序又如何把加工的结果交给主程序？即主、子程序之间用什么方式传递参数？

1. 子程序的定义

格式：　子程序名　PROC　[类型]

过程体

子程序名　ENDP

子程序名也具有段属性，偏移地址属性和类型属性：FAR 和 NEAR（缺省）。

NEAR 类型为段内调用，即主程序与所调用的子程序在同一个代码段内时。

FAR 类型为段间调用,即被另外代码段调用的过程要定义为 FAR 过程(不管这两个代码段是否在同一个模块中)。

（类型是否与标号一样?也说明使用上有相似之处）

2. 子程序的调用与返回

子程序调用指令 CALL

名　　称	JMP 格式	CALL 格式
段内直接	JMP　标号	CALL　标号
段间直接	JMP　标号（FAR）	CALL　标号　（FAR）
段内间接	JMP　OPD	CALL　OPD
段间间接	JMP　OPD　（FAR）	CALL　OPD　（FAR）

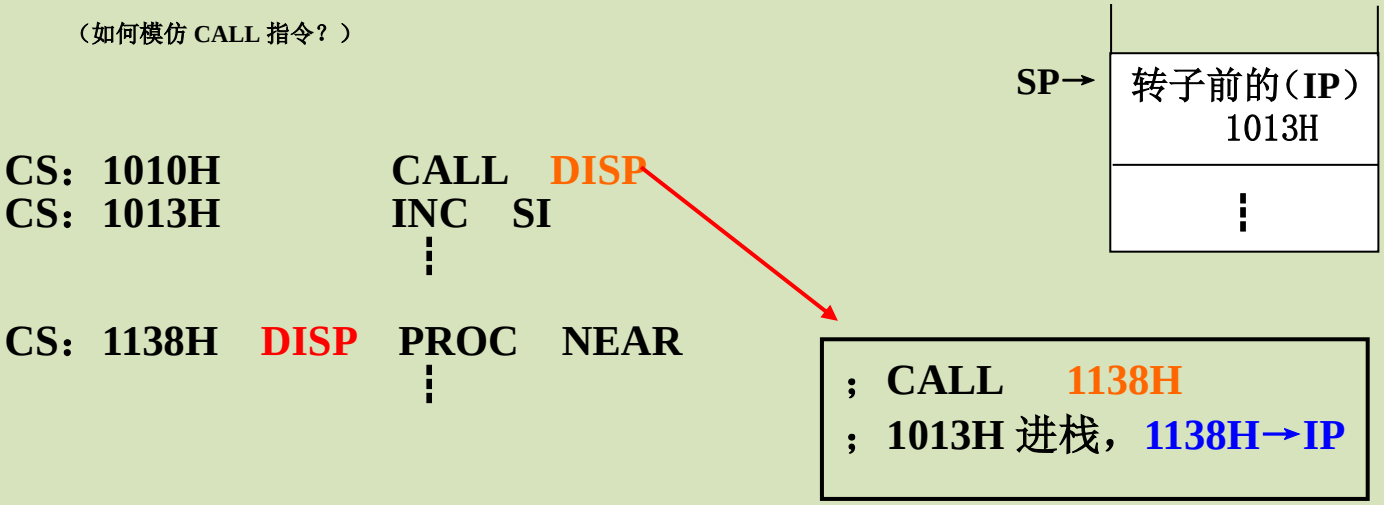
(1) 段内直接调用

格式：CALL　子程序名

功能：a. 跟在CALL后的下一条指令/断点的EA→↓(SP/ESP)　　（2/4B）

b. 子程序名的 EA→IP/EIP

（如何模仿 CALL 指令？）



(2) 段间直接调用

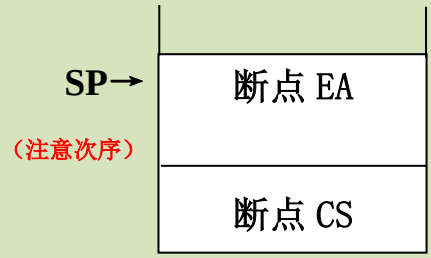
格式：CALL　FAR　PTR　子程序名

功能：　a. 断点 (CS) →↓(SP/ESP)

b. 断点(IP/EIP) →↓(SP/ESP)

c. 子程序名 所在段的段首址→CS

d. 子程序名 的 EA→IP/EIP



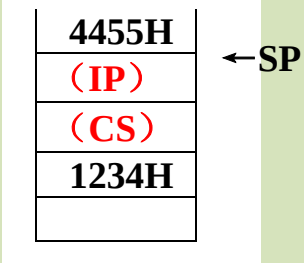
返回指令 RET　　（目标？）

格式：RET/RET　n

功能：若是段间返回，则其操作为：

① ↑（SP/ESP）→IP/EIP

② ↑（SP/ESP）→CS



若是段内返回，则只执行①。（栈顶指针要准确，栈顶数据可修改）

如果后面带常数 n（正偶数），则说明还要在堆栈中收回 n 个字节，即栈顶向下移动 n/2 个字。（SP/ESP）+ n → SP/ESP　(参数传递等中利用)

注：为了能正确返回主程序，在执行子程序过程中，不管是否使用过堆栈，必须保证进入子程序时的栈顶与执行 RET 时的栈顶一样，否则不能正确返回。

3. 调用现场的保护与恢复

调用子程序时，主程序中所使用的寄存器值不因子程序的调用而被破坏。

两种做法：

（1）该工作可在主程序中做， 即调用子程序前，将主程序所用到的寄存器的内容保存起来，到返回主程序再恢复。

```
……
BUF     DB    ‘OK’，‘$’
TEMP   DW    0，0
……
MOV   TEMP，AX
MOV   TEMP+2，DX
（PUSH   AX）
（PUSH   DX）
CALL   DISPLAY_BUF
MOV   AX，TEMP
MOV   DX，TEMP+2
（POP   DX）
（POP   AX）
```

```
；子程序 DISPLAY 定义
；显示 BUF 区的内容。 将使用 AX，DX
DISPLAY_BUF   PROC   NEAR
      ;
      LEA   DX，BUF
      MOV   AH，9
      INT   21H
      ;
      RET
DISPLAY_BUF   ENDP
```

缺点：不方便、特别麻烦，因为每调用一次子程序，就要作一次保存和恢复工作。

（2）该工作可放在子程序中做，凡是子程序用到的寄存器，在进入子程序后首先入堆栈保存，返回之前从堆栈中弹出恢复。

这样在调用子程序时，主程序不用考虑寄存器被破坏的情况（除传递参数到子程序的寄存器除外）。

```
A     PROC
      PUSH  BX
      PUSH  CX
      PUSH  DI
      PUSH  SI
        ┆          ; 子程序工作部分，可修改 BX、CX、DI、SI 的内容
      POP   SI
      POP   DI
      POP   CX
      POP   BX
      RET          ; 返回断点处
A     ENDP
```

保护现场，将 BX、CX、DI、SI 的值备份到堆栈中

恢复现场，将堆栈中的备份值送还到 BX、CX、DI、SI 中

4. 子程序与主程序之间传递参数的方式

在已经使用过的调用中（INT　21H）参数是如何传送的？

(1) 寄存器法

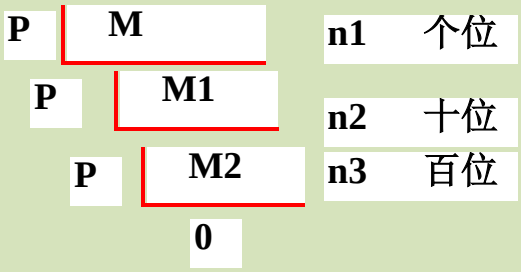
将所需参数放在寄存器中带入子程序。

优点：传递信息快，编程简单方便，节省存贮单元，但参数不能太多，要避免出错。

例：（书 P135）编写子程序 RADIX，将 EAX 中的 32 位无符号二进制数转换为（EBX）所指定进制的数字，并将其 ASCII 码送入（SI）所指定的缓冲

区中。

算法：二进制数 M 转换为 P 进制数（P=3~16）可采取“除 P 取余”法。



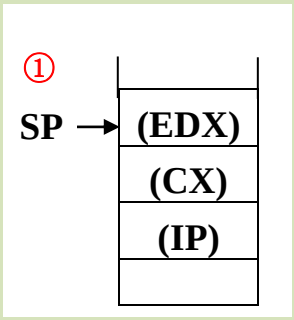
入口参数：待转换的 32 位无符号二进制数→EAX；

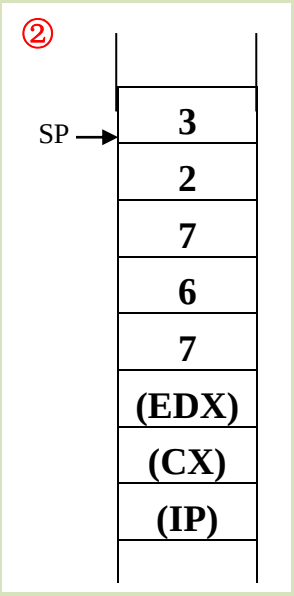
待转换的基制数→EBX；

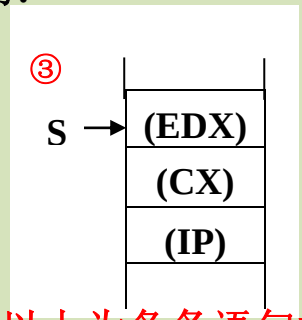
存放转换后 ASCII 码缓冲区首址→SI

出口参数：SI：存放转换结果的缓冲区末址+1

```
.386
RADIX   PROC              （设（EAX）=7FFFH=32767）
        PUSH   CX
        PUSH   EDX        } 保护现场
        MOV    CX，0      ①
LOP1:   MOV    EDX，0
        DIV    EBX
        PUSH   DX
        INC    CX
        CMP    EAX，0
        JNE    LOP1
LOP2:   POP    AX         ②
        CMP    AL，10
        JB     L1
        ADD    AL，7
L1:     ADD    AL，30H        ;OR  AL, 30H 可以吗?   x
        MOV    [SI] ，AL
        INC    SI             ;
        LOOP   LOP2       ③
        POP    EDX        } 恢复现场
        POP    CX
        RET
RADIX   ENDP
```







以上为各条语句执行之前堆栈所处的状态

例：阅读以下程序，指出它所完成的功能。

```
STACK     SEGMENT  USE16  STACK
          DB  200 DUP(0)
STACK     ENDS
DATA      SEGMENT  USE16
OUT_BUF   DB  15  DUP（0）
CRLF      DB  0AH，0DH，‘$’
ARRAY     DW  0101111011B，11011110001B，11101010110110B，……
COUNT     EQU （$-ARRAY）/2
DATA      ENDS
CODE      SEGMENT  USE16
        ASSUME  CS：CODE，DS：DATA，SS：STACK
START：   MOV  AX，DATA
          MOV  DS，AX
          LEA   DI，ARRAY
          MOV  CX，COUNT
LO:     MOV  EBX，10                    ；待转换的进制
        MOVZX  EAX，WORD  PTR[DI]   ；待转换的数
        LEA   SI，OUT_BUF               ；存 ASCII 码中首址
        CALL  RADIX                     ；出口参数为 SI
        MOV  BYTE  PTR  [SI]，20H ；在转换出的 ASCII 码后面补空格
        INC   SI
        MOV  BYTE  PTR  [SI]，’$’      ；在空格后面补’$’
        LEA   DX，OUT-BUF
        MOV  AH，9                      ；输出当前转换成的 ASCII 码
        INT   21H
        ADD  DI，2          }           ；修改指针，准备取下一待转换的数
        LOOP  LO
        LEA  DX，CRLF                   ；输出回车换行
        MOV  AH，9
        INT   21H
      MOV  AH，4CH
      INT   21H
```

```
RADIX   PROC
          ┆
RADIX   ENDP
CODE    ENDS
        END   START                *
```

该程序的功能为:将 ARRAY 字存储区中的一组二进制数分别以十进制形式显示出来，之间用空格隔开，最后以回车换行作结束。

(2) 堆栈法

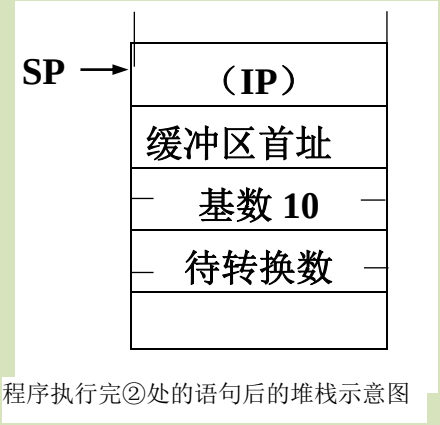
当参数个数较多时，一般用堆栈法传递参数，在使用堆栈时要注意栈顶的变化，要收回堆栈中传递参数所占用的单元。例如，将前例重写主、子程序：

```
          ┆
        LEA    DI，ARRAY
        MOV    CX，COUNT
LO:     MOVZX   EAX，WORD   PTR [DI]
        PUSH   EAX                          ；待转换的数
        MOV    EAX，10
        PUSH   EAX                          ；待转换的进制
        MOV     AX，OFFSET   OUT_BUF
        PUSH   AX                           ；存 ASSCII 码的首址
        CALL   RADIX_S                      ；出口参数为 SI (无入口参数)
        MOV   BYTE   PTR [SI]，20H   ；在转换出的 ASCII 码后面补空格
        INC     SI
        MOV   BYTE   PTR   [SI]，'$'       ；在空格后面补'$'
                ┆

RADIX_S    PROC
          PUSH   EAX                        ；保护现场
          PUSH   EBX
          PUSH   CX
          PUSH   EDX
          PUSH   BP
          MOV   BP，SP                      ；①
          MOV   SI，18[BP]                  ；获得存放转换结果的缓冲区首址
```

```
        MOV   EBX，20[BP]                ；获得基数
        MOV   EAX，24[BP]                ；获得待转换数
        MOV   CX，0
              ┇                          ；同 RADIX 对应的转换部分
        POP    BP                        ；恢复现场
        POP    EDX
        POP    CX
        POP    EBX
        POP    EAX                       ；②
        RET    10              ；返回的同时收回传递参数所占用的 10 个字节。
RADIX_S  ENDP
```

程序执行到①处时的堆栈示意图



程序执行完②处的语句后的堆栈示意图

（认真计算[BP]前和 RET 后的数字）

采用堆栈法可传递大量参数。是高级语言常采用的方法。

(3) 约定单元法

即将入口参数或出口参数放入事先规定好的存贮单元中。

例：**INT 21H　10** 号功能调用。

5. 子程序的嵌套

子程序中间再调用另外的子程序称子程序的嵌套。只要堆栈空间允许，嵌套的层次可以不限。嵌套子程序的设计方法与前面学习的方法一致。

总　结

一、程序控制指令及应用

1.　转移指令

简单：　JZ/JE　JNZ/JNE　JS　JNS　JO　JNO　JC　JNC

无条件转：　JMP ，（子程序调用）CALL，RET

有符号数转：　JGE　JG　JL　JLE

无符号数转：　JAE　JA　JB　JBE

循　　环：　LOOP，LOOPZ，LOOPNZ，JCXZ/JECXZ

2.　分支程序设计　　选择合适转移指令

3.　循环程序设计　　多重循环时，循环体不能交叉，要注意置初值的位置。

4.　子程序设计　　（1）子程序定义：PROC 和 ENDP

（2）调用与返回：CALL 和 RET

（3）参数传递：寄存器方式、堆栈方式、约定单元法。

（4）现场保护与恢复。

（5）学会使用 DEBUG 观察堆栈的变化，CS 、IP/EIP 进栈和出栈的情况，FAR、NEAR 子程序中 RET 指令翻译的情况等。

二、程序设计中注意事项

1、依操作数的个数、大小，正确分配数据存储区，不要浪费。

2、选择合适的指令和寻址方式

（1）注意字节、字、双字操作的区别。

（2）要选择合适的寻址方式，注意各寻址方式之间的区别。

```
ADD   [SI]，SI    ；SI 的内容不变。
ADD   SI，SI      ；SI 内容乘以 2。
```

（3）要选用效率高的指令。

用 INC AX；DEC AX；不用 ADD AX，1；SUB AX，1

（4）不要插入不必要的语句、表达式。

```
DEC    BX
CMP   BX，0    ；多余
JNE   LOPA
```

又如：

```
MOV AX，A                MOV AX，A
MOV BX，B      ——→       CMP AX，B
CMP AX，BX
```

（5）寄存器的使用要合理安排

（6）目的操作数不允许立即方式

（7）两个存储器操作数不能出现在同一条指令之中。

（8）当立即数与不含变量的存储器操作数一起时，一定指定其类型。

（9）两操作数类型要匹配。

3、合理安排源程序的格式

（1）每个程序的开始应简介程序的功能、所用算法、使用的寄存器及重要的符号地址、输入输出数据等。

（2）一般来说，每个语句后面都应有注释。对于较难理解的部分要给予完整的

注释。注释的内容要简洁明了，能清楚地解释出此语句在程序中所起的作用，而不是对语句作孤立的解释。

（3）语句与标号要有固定的间隔。语句中应有的空格和标志符号不可省略。

}