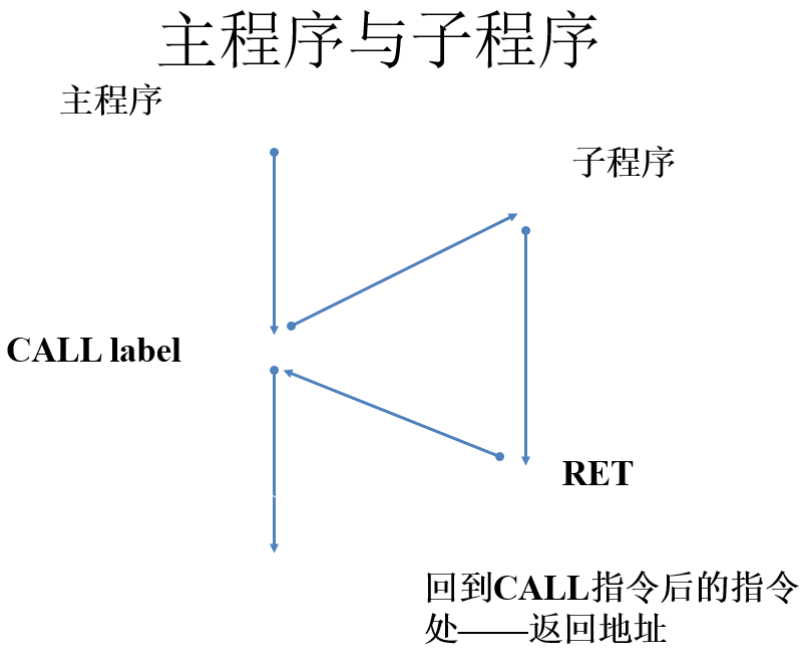
子程序

子程序是完成特定功能的一段程序，当主程序（调用程序）需要执行这个功能时，采用`call`调用指令转移到该子程序的起始处执行，当运行完子程序功能后，采用`ret`返回指令回到主程序继续执行。



call

```
CALL label          ; 段内调用、直接寻址
CALL r16/m16        ; 段内调用、间接寻址
CALL far ptr label  ; 段间调用、直接寻址
CALL far ptr mem    ; 段间调用、间接寻址
```

`call`指令需要保存返回地址：

- 段内调用——入栈偏移地址`ip`
  - SP←SP - 2，SS:[SP]←IP
- 段间调用——入栈偏移地址`ip`和段地址`cs`
  - SP←SP - 2，SS:[SP]←IP
  - SP←SP - 2，SS:[SP]←CS

ret

```
RET         ; 无参数段内返回
RET i16     ; 有参数段内返回
RETF        ; 无参数段间返回
RETF i16    ; 有参数段间返回
```

需要弹出`call`指令压入堆栈的返回地址：

- 段内返回——出栈偏移地址`ip`
  - IP←SS:[SP]， SP←SP + 2
- 段间返回——出栈偏移地址`ip`和段地址`cs`
  - IP←SS:[SP]，SP←SP + 2
  - CS←SS:[SP]，SP←SP + 2

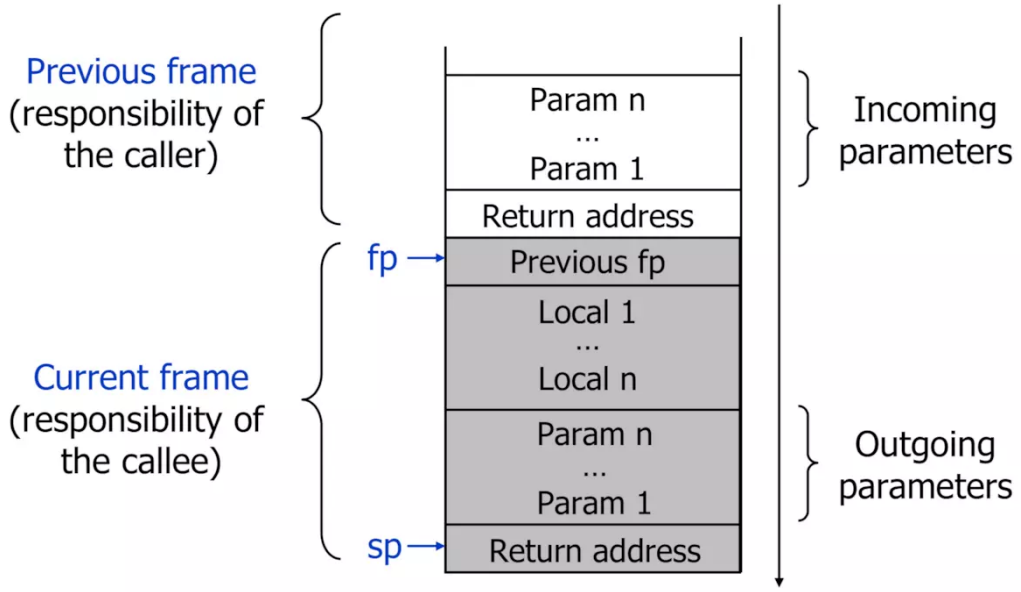
ret的参数

`ret`指令可以带有一个立即数`i16`，则堆栈指针SP将增加，即SP←SP+i16

这个特点使得程序可以方便地废除若干执行`call`指令以前入栈的参数

传参

1. 寄存器传参
2. 栈传参
   - 调用时
     - 将参数压栈
     - 将返回地址压栈
     - 将`ebp`压栈
     - 将`ebp`设置为与`esp`相等
     - 减小`esp`的地址为函数分配栈帧
   - 返回时
     - 弹出为寄存器保存的值
     - 设置`esp`等于`ebp`
     - 从栈中弹出旧`ebp`的值，并将`ebp`设置为弹出的旧值
     - 弹出返回地址



```
push bp      ; 保存基址
mov bp, sp   ; 设置栈帧底
; 保存寄存器
; push ...
; ...


; ...
; 恢复寄存器
; pop ...
mov sp, bp   ; 恢复
pop bp       ; 恢复基址
ret ; ret xxx
```

返回值

一般默认情况下使用`ax`存放返回值

栈平衡

- 调用者平栈
  - `retn (ret)`，然后在调用者空间平栈
- 被调者平栈
  - `retn imm`被调者同时完成返回和平栈

中断

中断（Interrupt ）是又一种改变程序执行顺序的方法，中断具有多种中断类型。

8086可以管理256个中断，各种中断用一个向量编号来区别，主要分成**外部中断**、**内部中断**

中断的指令有3条：

```
int i8   ; 中断调用指令：产生i8号中断
iret     ; 中断返回指令：实现中断返回
into     ; 溢出中断指令，若溢出标志OF=1，产生4号中断，否则顺序执行
```

外部中断

外部中断——来自CPU之外的原因引起的中断，又可以分成

- 可屏蔽中断：可由CPU的中断允许标志`IF`控制
- 非屏蔽中断：不受CPU的中断允许标志`IF`控制

内部中断

内部中断——CPU内部执行程序引起的中断，又可以分成：

- 除法错中断：执行除法指令，结果溢出产生的 0 号中断
- 指令中断：执行中断调用指令`INT i8`产生的`i8`号中断
- 断点中断：用于断点调试`INT 3`的 3 号中断
- 溢出中断：执行溢出中断指令，`OF=1`产生的 4 号中断
- 单步中断：`TF=1`在每条指令执行后产生的 1 号中断