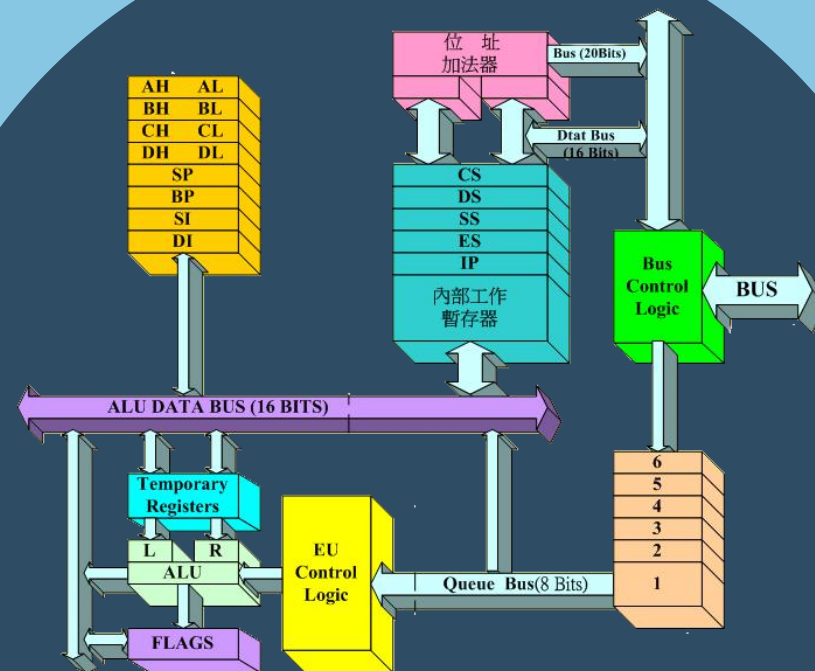


# 编制中断处理程序

贺利坚 主讲



汇编语言程序设计  
Assembly Language

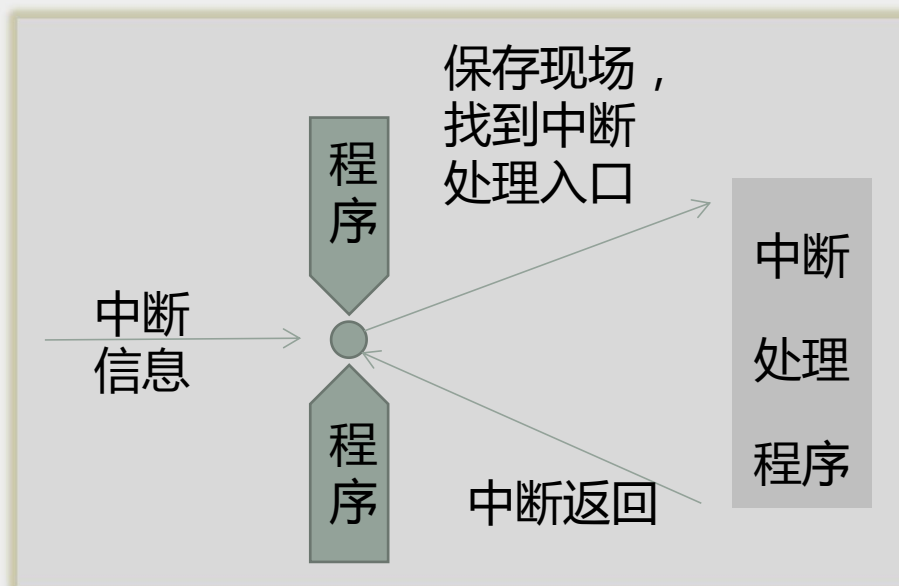
# 中断处理程序及其结构

中断过程：

- ( 1 ) 取得中断类型码N；
- ( 2 ) pushf
- ( 3 ) TF = 0 , IF = 0
- ( 4 ) push CS
- ( 5 ) push IP
- ( 6 )  $(IP) = (N*4)$  ,  $(CS) = (N*4+2)$

🖥️ CPU随时都可能检测到中断信息，所以中断处理程序必须常驻内存（一直存储在内存某段空间之中）。

🖥️ 中断处理程序的入口地址，即中断向量，必须存储在对应的中断向量表表项中(0000:0000-0000:03FF)。

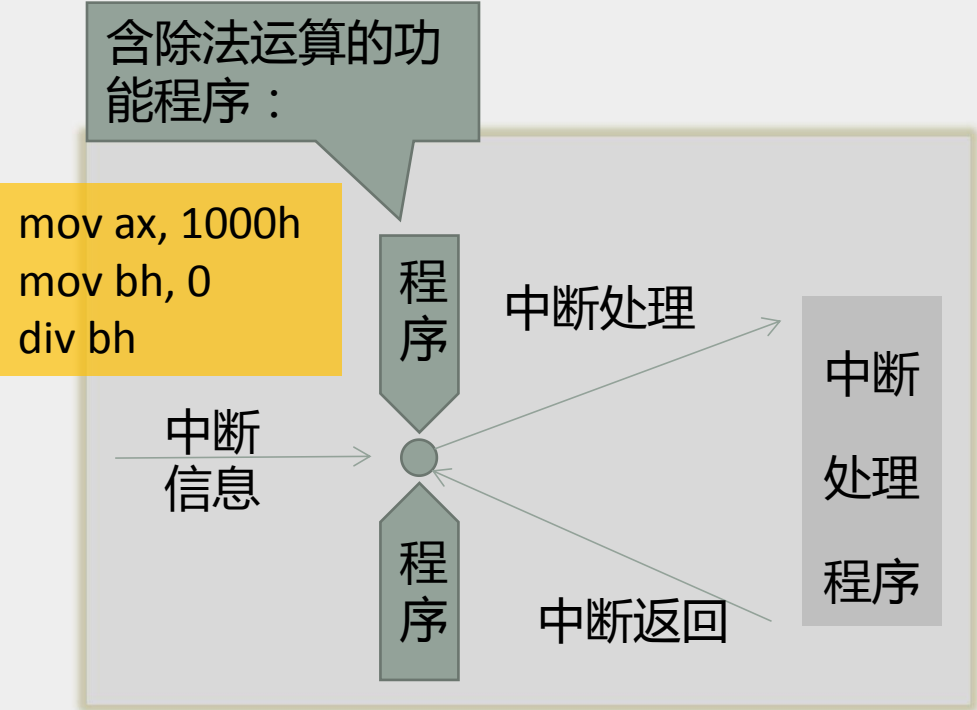


0000:0000	IP	}	0号中断处理程序 入口地址
0000:0002	CS		
0000:0004	IP	}	1号中断处理程序 入口地址
0000:0006	CS		
0000:03FC		}	255号中断处理程序 入口地址
0000:03FE			

$(IP) = (N*4)$  ,  $(CS) = (N*4+2)$  , N为中断类型码

# 编制中断处理程序——以除法错误中断为例


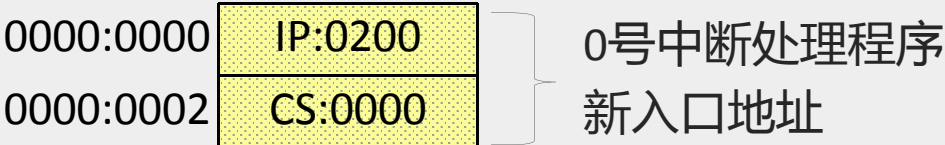
- 问题：如何编制中断处理程序？
- 方案：通过对 0 号中断，即除法错误的中断处理，体会中断处理程序处理的技术问题
- 预期效果：编写一个 0 号中断处理程序，它的功能是在屏幕中间显示 “overflow！” 后，然后返回到操作系统。



- do0:
- ① 相关处理
  - ② 向显示缓冲区送字符串“overflow！”
  - ③ 返回DOS



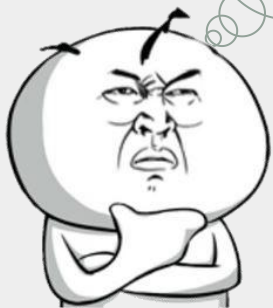
# do0子程序应该放在哪里？

要求	do0子程序应该存放在内存的确定位置 ——我们要重新找个地方，不破坏系统。		
			
事实	在操作系统之上使用计算机，所有的硬件资源都在操作系统的管理之下，应该向操作系统申请获得存放do0的内存。		——不想引入要好多技术细节，好不好？
简便方案	绕过操作系统，直接在找到一块别的程序不会用到的内存区，将do0传送到其中即可。		——不是工程化的方法，但也体现实用技巧！
找哪儿好呢？	内存0000:0000~0000:03FF，大小为1KB的空间是系统存放中断向量表，DOS系统和其他应用程序都不会随便使用这段空间。8086支持256个中断，但，实际上系统中要处理的中断事件远没有达到256个。		——找块空地给我们，好吗？
就这样做了！	利用中断向量表中的空闲单元来存放我们的程序。估计出，do0的长度不可能超过256个字节，就选用从0000:0200至0000:02FF的256个字节的空间。		
确定的方案：	将do0程序传送到内存0000:0200处。		

# 程序框架

- ( 1 ) 编写可以显示 “overflow ! ” 的中断处理程序：do0 ；
- ( 2 ) 安装程序：将do0送入内存0000:0200处；
- ( 3 ) 将do0中断处理程序的入口地址0000:0200存储在中断向量表0号表项中。

第2个问题：  
怎么写安装  
程序？



```
assume cs:code
code segment
start: do0安装程序
        设置中断向量表
        mov ax,4c00h
        int 21h

do0: 显示字符串 “overflow ! ”
        mov ax,4c00h
        int 21h

code ends
end start
```

# do0安装程序的实现

```
assume cs:code
```

```
code segment
```

```
start: do0安装程序
```

设置中断向量表

```
mov ax,4c00h
```

```
int 21h
```

```
do0: 显示字符串"overflow!"
```

```
mov ax,4c00h
```

```
int 21h
```

```
code ends
```

```
end start
```

**do0end:nop**

设置ds:si指向源地址 cs:do0

设置es:di指向目的地址 0:200h

设置cx为传输长度

设置传输方向为正

```
rep movsb
```

```
mov ax,cs
```

```
mov ds,ax
```

```
mov si,offset do0
```

```
mov ax,0
```

```
mov es,ax
```

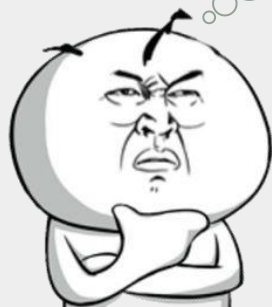
```
mov di,200h
```

```
mov cx,do0部分代码的长度
```

```
cld offset do0end-offset do0
```

```
rep movsb
```

第3个问题：  
do0怎么写？



# do0子程序的实现

```
assume cs:code
```

```
code segment
```

```
start: do0安装程序
```

```
    设置中断向量表
```

```
    mov ax,4c00h
```

```
    int 21h
```

```
do0: 显示字符串"overflow!"
```

```
    mov ax,4c00h
```

```
    int 21h
```

```
do0end:nop
```

```
code ends
```

```
end start
```

```
do0: jmp short do0start
```

```
    db "overflow!"
```

```
do0start:
```

```
    mov ax, cs
```

```
    mov ds, ax
```

```
    mov si, 202h
```

将数据写到  
代码区，保  
证与代码一  
起加载。

```
    mov ax,0b800h
```

```
    mov es,ax
```

```
    mov di,12*160+36*2
```

```
    mov cx,9
```

```
s: mov al,[si]
```

```
    mov es:[di],al
```

```
    inc si
```

```
    add di,2
```

```
    loop s
```

直接输出到  
显示缓冲区

🖥️ 设置中断向量表任务：将do0的入口地址0:200h，写到中断向量表的0号表项中

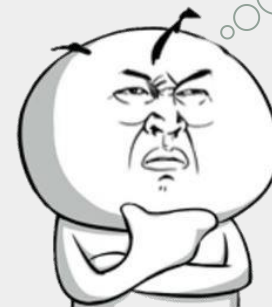
```
mov ax,0
```

```
mov es,ax
```

```
mov word ptr es:[0*4], 200h
```

```
mov word ptr es:[0*4+2], 0
```

第4个问题：设置中断向量表



# 小结

```
do0: jmp short do0start
      db "overflow!"
do0start:
  mov ax, cs
  mov ds, ax
  mov si, 202h
  mov ax, 0b800h
  mov es, ax
  mov di, 12*160+36*2
  mov cx, 9
s: mov al, [si]
   mov es:[di], al
   inc si
   add di, 2
   loop s
```

常驻内存的程序：由自己安装到内存的指定区域的代码和数据。

```
assume cs:code
```

```
code segment
```

```
start: do0安装程序
```

```
      设置中断向量表
```

```
      mov ax, 4c00h
```

```
      int 21h
```

```
do0: 显示字符串"overflow!"
```

```
      mov ax, 4c00h
```

```
      int 21h
```

```
do0end: nop
```

```
code ends
```

```
end start
```

```
mov ax, cs
mov ds, ax
mov si, offset do0
mov ax, 0
mov es, ax
mov di, 200h
mov cx, offset do0end - offset do0
cld
rep movsb
```

```
mov ax, 0
mov es, ax
mov word ptr es:[0*4], 200h
mov word ptr es:[0*4+2], 0
```

测试

🖥️ 运行程序，改变中断向量

🖥️ 触发除法错

```
mov ax, 8
mov bh, 0
div bh
```