王爽汇编第12章,内中断

1. 内中断概述

任何一个通用的CPU，比如8086，都具备一种能力，可以在执行完当前正在执行的指令之后，检测到从CPU外部发送过来的或内部产生的一种特殊信息，并且可以对立即所接收到的信息进行处理。这种特殊的信息，我们称其为：中断信息。

中断意思：CPU不再接着（刚执行完的指令）向下执行，而是转取处理这个特殊信息。（感觉和异常有点像？）

2. 内中断的产生

当CPU内部有什么事情发生的时候，将产生需要马上处理的中断信息。

产生的中断信息如下：

- 除法错误(0)
- 单步执行(1)
- into指令(4)
- int指令(int n) n:中断类型码

终端信息中包含 中断类型码 ，中断类型码为一个字节型数据，可以表示256种中断信息的来源（ 中断源 ）

3. 中断处理程序

CPU在收到中断信息后，需要对中断信息进行处理。而如何对中断信息进行处理，可以由我们编程决定，就像我们写高级语言产生异常后，可以在catch里面对抛出的异常进行处理。

- 用来处理中断信息的程序被称为中断处理程序。
- 根据CPU的设计，中断类型码的作用就是用来定位中断处理程序。比如CPU根据中断类型码4，就可以找到4号中断的处理程序

4. 中断向量表

CPU是根据中断向量表来存放 中断处理程序的入口地址 的，里面放的是所有中断处理程序入口地址的列表。

CPU用8位的中断类型码通过中断向量表找到相应的中断处理程序的入口地址

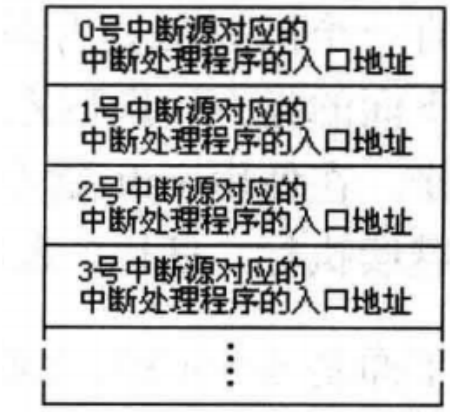

图 12.1 中断向量表

中断向量表在内存中保存，其中存放着256个中断源所对应的中断处理程序

CPU只要知道了中断类型码，就可以将中断类型码作为中断向量表的表项号，定位相应的表项，从而得到中断处理程序的入口地址。

8086PC机，中断向量表指定放在内存地址0处。内存0000:0000到0000:03FF的1024个单元中存放着中断向量表。
在中断向量表中，一个表项存放一个中断向量。这个入口地址包括段地址和偏移地址，即一个表项占两个字，高地址字存放段地址，低地址字存放偏移地址。

5. 中断过程

中断的过程一共如下步骤：

- 取得中断类型码（N）
- pushf（保存标志寄存器现场）
- TF = 0, IF = 0
- push cs（方便跳回来）
- push ip
- ip = N * 4
- CS = N * 4 +2

在最后一步完成后，CPU就会跳到由程序员编写的 中断处理程序 入口处执行代码。

6. iret指令

中断处理程序的常规编写步骤，和函数差不多只不过用了(iret)：

- 保存用到的寄存器
- 处理中断
- 恢复用到的寄存器
- 用iret指令返回

iret = pop 、 popcs 、 popf 。

7. 除法错误的中断处理

```
1 mov ax,1000h
2 mov bh,1
3 div bh ;除法溢出
```

当CPU执行 div bh 时，发生了除法溢出错误，产生了0号中断信息，从而引发中断过程

CPU跳到0号中断处理程序执行代码

0号系统中断处理程序的功能：显示 Divide overflow ，返回到DOS中。（像极了异常处理）

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=0B39  CS=0B39  IP=0100   NV UP EI PL NZ NA PO NC
0B39:0100 B80010        MOV     AX,1000
-t

AX=1000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=0B39  CS=0B39  IP=0103   NV UP EI PL NZ NA PO NC
0B39:0103 B701          MOV     BH,01
-t

AX=1000  BX=0100  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=0B39  CS=0B39  IP=0105   NV UP EI PL NZ NA PO NC
0B39:0105 F6F7          DIV     BH
-t

Divide overflow

D:\>
```

图 12.2　系统对 0 号中断的处理

7.1. 动手实践，重写除法溢出

需求：编写0号中断处理程序do0，当发生除法溢出时，在 屏幕中间显示 "overflow! "，返回DOS。

分析：

1. 发送除法溢出时，产生0号中断信息，引发中断过程，CPU将进行 中断过程小节 中的过程。
2. 按照如下步骤编写中断处理程序，当中断0发生时，即可显示 Overflow! 。

- 相关处理
- 向显示缓冲区送字符串 Overflow!
- 返回DOS　　PS:(我们可以将段程序称为: do0)

3. 现在的问题是： do0 应该存放在内存中，因为除法溢出随时可能会发生，CPU随时可能将 CS:IP 指向do0的入口,执行程序。

之前有讲过，内存 0000:0000~0000:03FF ，大小为1KB空间，存放256个中断向量表的，但是一般情况，系统中要处理中断条件肯定不会有256个那么多，所以这时候很多中断表是空着的。我们可以选 0000:0200~0000:02FF 来存中断程序。

4. 将中断处理程序 do0 放到 0000:0200 后，如果除法溢出后，CPU转去执行 do0 ，则必须将do0的 入口地址 ,存放到 0号中断码 向量表的偏移中，即 0000:0000 。
5. 经过上面的分析，总结如下:

1、0000:0200至0000:02FF的256个字节的空间所对应的中断向量表项都是空的，可以将中断处理程序 do0 传送到内存0000:0200处。

2、中断处理程序do0放到 0000:0200 ,再将 其地址登记在中断向量表 对应表项

- 0号表项的地址 0:0 。 0:0 字单元存放偏移地址， 0:2 字单元存放段地址
- 将do0的段地址0存放在 0000:0002 字单元中，将偏移地址200H存放在 0000:0000 字单元

```
1   ;**************************************
2   ; 编写0号中断处理程序do0，当发生除法溢出时，
3   ; 在`屏幕中间显示` "overflow! "，返回DOS。
4   ;**************************************
5   assume cs:code,ds:data,ss:stack
6
7   ;>>>>>>>>>>>>>>>>>>>>>>>>
8   ;数据段
9   ;>>>>>>>>>>>>>>>>>>>>>>>>
10  data segment
11  data ends
12
13  ;>>>>>>>>>>>>>>>>>>>>>>>>
14  ;代码段
15  ;>>>>>>>>>>>>>>>>>>>>>>>>
16  code segment
17   main:
```

```
_main:
    ;code
    mov ax,cs
    mov ds,ax
    mov si,offset do0 ;设置ds:si指向源地址
    ;2.
    mov ax,0
    mov es,ax
    mov di,200h ;设置es:di指向目标源地址
    ;3.
    mov cx,offset d0end - offset do0 ;设置cx为传输长度,编译时给出do0部分代码长度
    cld ;设置传输方向位正
    rep movsb ; 挨个传送字节到 0:200处.
    ;4.
    mov ax,0 ;设置中断向量表
    mov es,ax
    mov word ptr es:[0*4],200h
    mov word ptr es:[0*4+2],0
    ;return 2 dos
    mov ax,4c00h
    int 21h

;>>>>>>>>>
;do0函数
;>>>>>>>>>
d0:
     jmp short do0 start
     db "Overflow!"
do0start:
     mov ax,cs
     mov ds,ax
     mov si,202h ; 设置ds:si指向字符串
     ;2.
     mov ax,0b800h
     mov es,ax
     mov di,12*160+36*2; 设置es:di指向显存空间的中间位置
     ;3.
     mov cx,9 ;字符串长度
   s:mov al,[si]
     mov es:[di],al ;传送字符
     inc si
     add di,1
     ;
     mov al,02h ;设置颜色
     mov es:[di],al
     add di,1
     loop s

     ;return 2 dos
     mov ax,4c00h
     int 21h
do0end: nop

code ends
end _main
```

8. 单步中断

CPU在执行完一条指令后，如果检测到标志寄存器的 `TF` 位为 `1` ，则产生单步中断， `中断类型码1` 。

【思路扩展】： `原来OD和x32dbg单步调试用的应该就是这种中断办法吧，改了TF，然后产生中断，单步执行代码`

单步中断的过程：

1. 取得中断类型码1
2. pushf,TF=0、IF=0
3. push cs、push ip
4. ip=(1x4),cs=(1x4+2)

最后，CPU提供单步中断功能的原因是，为单步跟踪程序的执行过程，提供了实现机制。

9. 响应中断的特殊情况

比如我们设置 `ss段寄存器` ，然后还没开始设置 `sp栈顶` 的时候，这时候如果发生中断过程，由于要进行一系列的pushf或者push cs等操作，这样会导致我们堆栈里面有数据，导致 `ss:sp` 指向的不是正确的栈顶，将引起错误。

所以CPU提供了一个机制，在设置完 `ss段寄存器后` CPU不响应中断。（所以我们最好将ss和sp指令连续存放）

```
mov ax,1000h
mov ss,ax
mov sp,0
```

10. int中断指令

int指令的格式为： `int n` ，n为中断类型码， `它的功能是引发中断过程` 。

CPU执行int n指令，相当于引发一个n号中断的中断过程

比如：

```
int 2
int 3
```

> 参考资料：
>
> 《王爽汇编语言(第4版）第12章：内中断》
>
> https://blog.csdn.net/qq_39654127/article/details/88698911 王爽《汇编语言》笔记