一、本章的学习内容

本章学习在几种其它的计算机资源下的汇编语言程序设计技术，包括：

（1）输入输出指令的使用格式及功能；

（2）主机与外部设备之间传送数据的方式；

（3）中断的概念及中断处理程序设计；

（4）WIN32程序设计基本方法与技术。

通过本章的学习，有助于深入系统的核心，充分发掘系统的资源，有效发挥汇编语言的优势。

新的计算机资源包括：

- 外部设备；（输入/输出指令，数据传递方式）
- 中断系统；（中断机制、软/硬件中断、中断处理程序设计）
- ROM BIOS；（软中断调用）
- 协处理器；（浮点指令、运算）
- WINDOWS操作系统。（宏汇编语言功能、WIN-API、32位编程）

二、本章的学习重点

1. 输入输出指令IN、OUT的使用格式及功能；
2. 中断矢量表，中断处理程序的编制方法；
3. 段的简化定义方法；
4. 结构的定义与使用方法；
5. 基于窗口的WIN32程序的结构、功能和特点，基本的程序设计方法。

6.1 输入输出指令和数据的传送方式

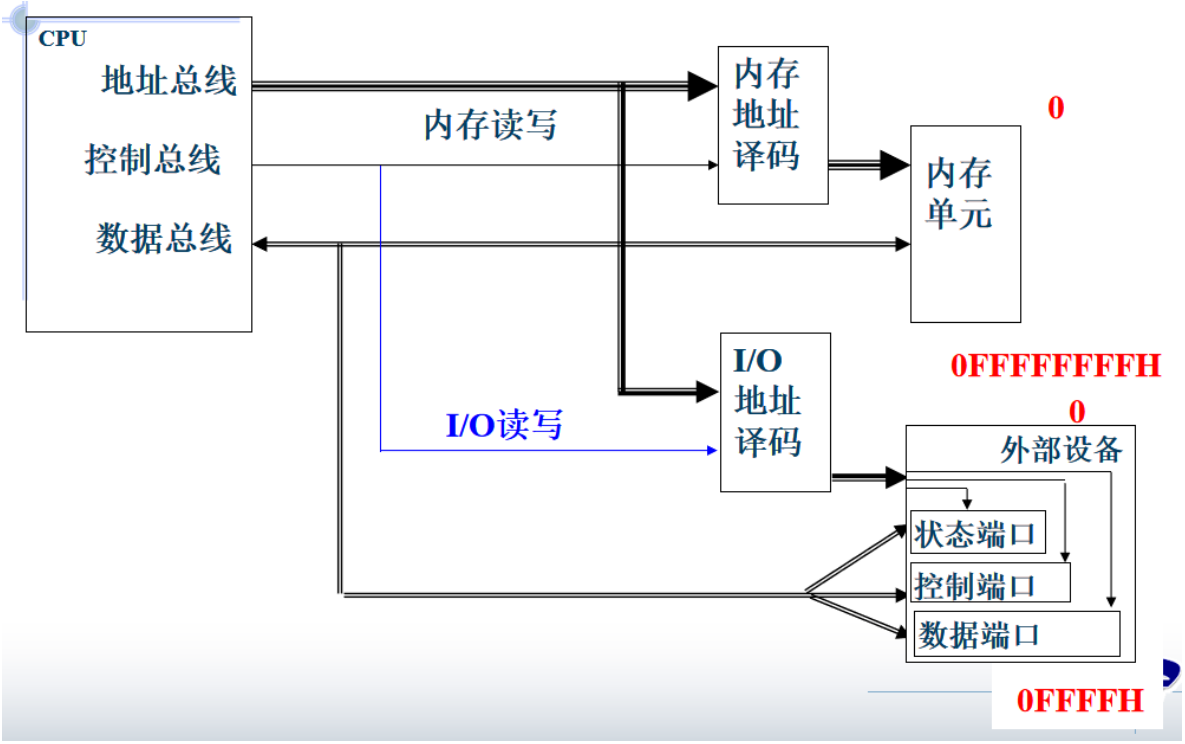
前五章所有的指令都是访问主存的指令，主存都是通过8位字节编址

CPU对外部设备的访问，通过外部设备寄存器进行

- 系统将所有的外部设备寄存器编址，通过地址去访问
- 所有的外部设备寄存器地址组成一个地址空间——I/O地址空间
- I/O地址空间与内存地址空间的差别
  编址方式与内存相同，地址范围从0000H-0FFFFH，16位，2^16共64K，不分段

  > *内存编址范围从00000000H——0FFFFFFFFH，32位，2^32共4G*

- I/O空间只能由I/O指令访问，其他指令一律无效



设备寄存器又叫端口

- 设备状态寄存器——状态端口

- 设备控制寄存器——控制端口
- 设备数据寄存器——数据端口

6.1.1输入/输出指令

1. 输入指令 IN

语句格式： IN OPD，OPS
功能： （OPS）→**累加器OPD**
即从指定的外设寄存器端口OPS中取数据送入累加器OPD中

说明：

1. 当外设寄存器的地址<=255时，OPS 可用 **立即数**或者用**DX**表示待访问的**端口地址**（也就是设备寄存器的地址）
   注意是十进制的255，不能是255H，那超过了
2. 当外设寄存器的地址 >255时，OPS**只能用DX**表示。
3. OPD只能是累加器**AL、AX或EAX。**
4. **主存空间的任何寻址方式在此均不适用**

即: IN AL/AX/EAX，OPS

```
例：
IN  AL，60H
执行前：（60H）= 11H，（AL）= 0E3H
执行后：（AL）= 11H，（60H）不变
说明：
60H是键盘将当前按键的键码输入到计算机内的端口的地址。该指令语句从60H号端口中读取一个字节的键码送到AL中，即（60H）→AL。

当（DX）=60H时，
IN  AL，DX  等价于  IN  AL，60H

例：  IN  AX，DX
执行前：（DX）＝200H，（200H）=33H  （201H）=44H（AX）＝1234H
执行后：（AX）＝4433H  （DX）、（200H）和（201H）不变
说明：
以DX中的内容200H为起始端口地址，从端口中读取一个字送到AX中，完成（[DX]）→AX的功能。
即：（200H）→AL、（201H）→AH。
```

> 注意功能是（OPS）→**累加器OPD**
>
> 它读入的是一个字节/字/双字，是看OPS的寄存器是AL/AX/EAX而定
>
> 注意DX或者立即数，都只是存放端口的地址。相当于EA。

2. 输出指令out

语句格式： OUT OPD，OPS
功能：**累加器（OPS）**→OPD
即: OUT OPD， AL/AX/EAX
将累加器OPS的数据送到外设寄存器OPD中。

说明：

1. OPD = 立即数 或者DX，同理，大于255的时候要用DX

2. OPS只能是累加器AL、AX或EAX。

```
例：   OUT    80H，EAX
执行前：
        （EAX）=11223344H，（80H）=55H，（81H）=66H，（82H）=77H，（83H）=88H
执行后：
        （80H）=44H，（81H）=33H，（82H）=22H，（83H）=11H，（EAX）
```

说明：

该指令完成（EAX）→[80]的功能。即（EAX ）中的4个字节按照从**低到高的**次序分别送到了外设寄存器地址为80H~83H的4个端口中。

> 也可以是说是从高字节到低字节放置到83H-80H的4个端口中，这样的逻辑更符合。

从上面的例子可以看出：

1. I/O空间的访问不存在分段的问题（不使用段寄存器）；
2. 在输入/输出指令中，**寻址方式的表示形式不同于第二章的格式规定。**
   - 用立即数表示的端口地址形式实际相当于第二章中的直接寻址方式；
     相当于 [地址] 这样子
   - 用寄存器表示的端口地址形式实际相当于第二章中的寄存器间接寻址方式。
     相当于 [DX] 这样子
   - 别搞混了。

3. 串输入指令 INS

语句格式： INS OPD, DX

INSB — 输入字节串

INSW — 输入字串

INSD — 输入双字串

功　能：（[DX]）→ES：[DI/EDI]，指针修改

4. 串输出指令 OUTS

语句格式： OUTS DX，OPS

OUTSB — 输出字节串

OUTSW — 输出字串

OUTSD — 输出双字串

功　能：（DS：[SI/ESI]）→[DX]，指针修改

在实方式下，I/O空间的访问没有特殊的限制，

在保护方式下，CPU对I/O功能提供保护。

6.1.2 数据的传送方式

1. 无条件传送方式（直接I/O方式）

不考虑外设的工作状态

在输入输出时都认为外设准备好了

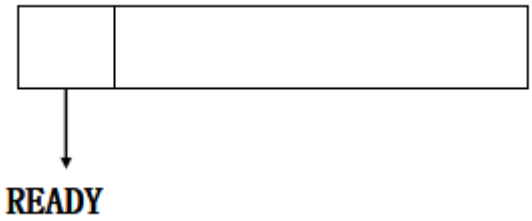
最简单，但要求外设的工作速度要能与CUP同步，否则就可能出错

也要求编程者非常清楚外部设备的工作状态，可不可用，不然很容易错，直接用IN,OUT完成

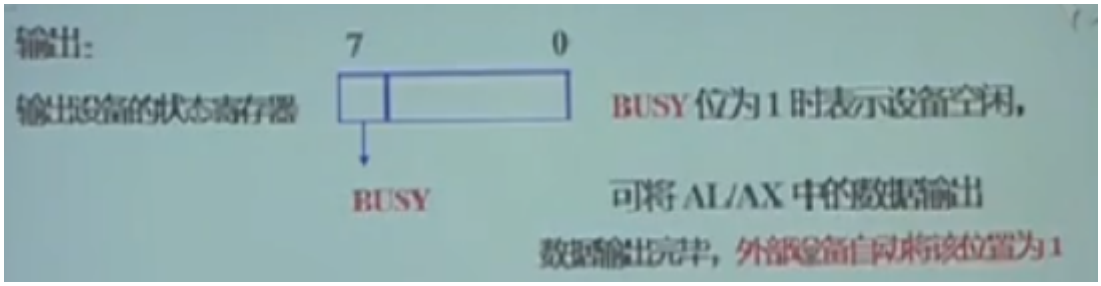
2．查询传送方式

状态寄存器的最高位：**输入状态寄存器**“READY”位为1时表示要输入的数据已准备好。

```
INPROG: IN AL,STATUS_PORT;从状态寄存器输入状态信息
        TEST AL,80H;检查"ready"位，test是逻辑乘运算
        JZ INPROG;未准备好，转回
        IN AL,DATA_PORT;准备好，把输入输入。
```



同样有输出状态检测寄存器。



但查询传送方式浪费了大量的CPU时间。CPU啥事都不干，一直在这儿做检测，浪费时间。

3．直接存储器传送方式

直接存储器传送方式也称DMA（Direct Memory Access）方式。

利用**DMA控制器**管理，适用于**高速I/O设备**。不需要经过CPU，在主存和外部设备之间开辟一条直接的传送数据通道，以硬件为代价获得传送的高速度。

4．中断传送方式

节省了CPU时间，见下节

软硬件结合的方式

特点：

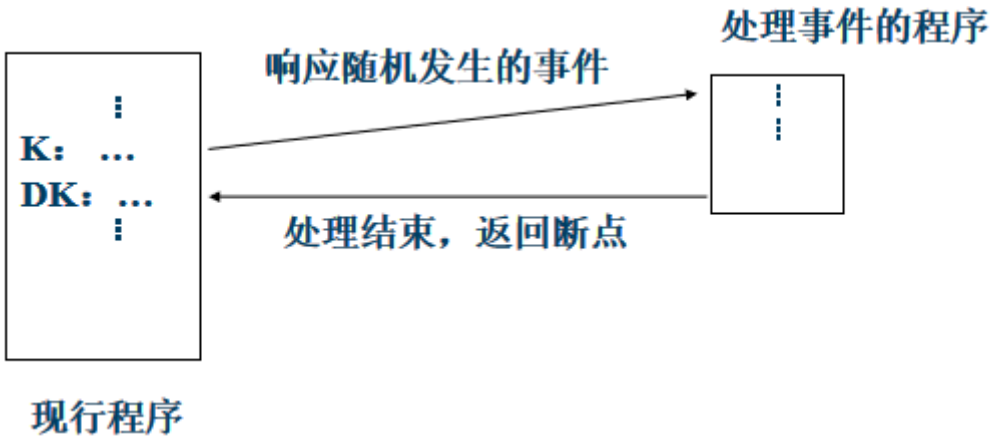
1. CPU用IN,OUT指令启动I/O设备后，不再查询I/O设备的工作状态
2. CPU去执行其他程序
3. 外部设备准备好后，向CPU发出中断请求，然后CPU相应，去执行，然后再返回
4. 如果数据传输未结束，则一直重复3，直到关闭相应的中断处理硬件和外部设备。

6.2 中断与异常

6.2.1中断的概念

- 中断：是CPU所具有的能打断当前执行的程序，转而为**临时出现的事件**服务，事后又能自动按要求恢复执行原来程序的一种功能。

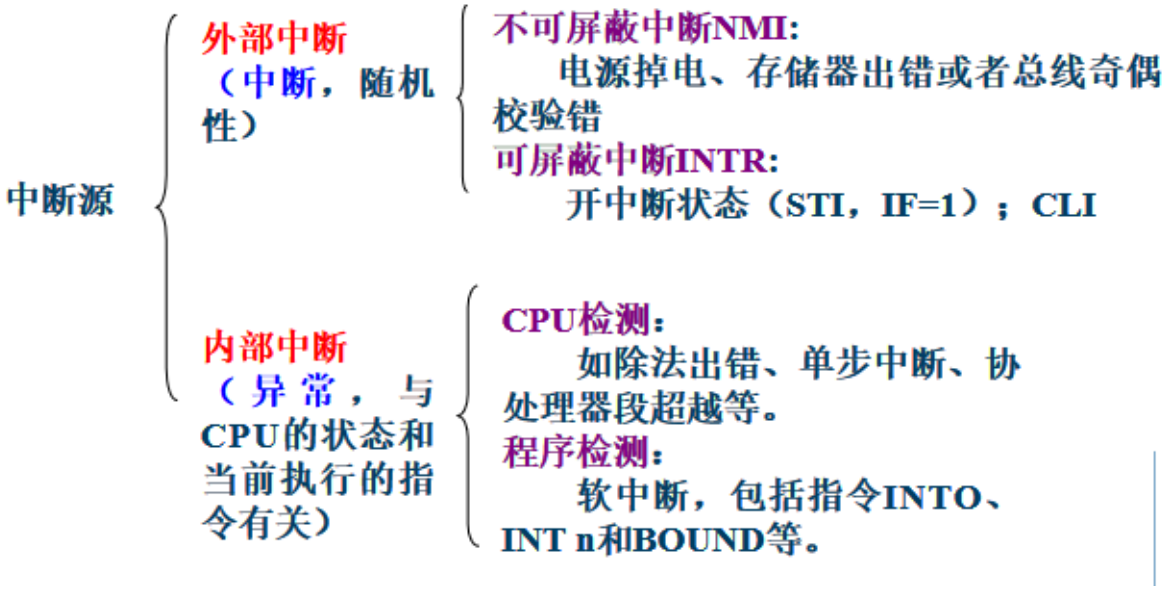
- 中断系统：实现这种功能的**软、硬件装置**。（注意是有硬件手段的）
- 中断处理程序（或中断服务程序）：临时出现事件的处理程序。
- 中断源：引起中断的事件。

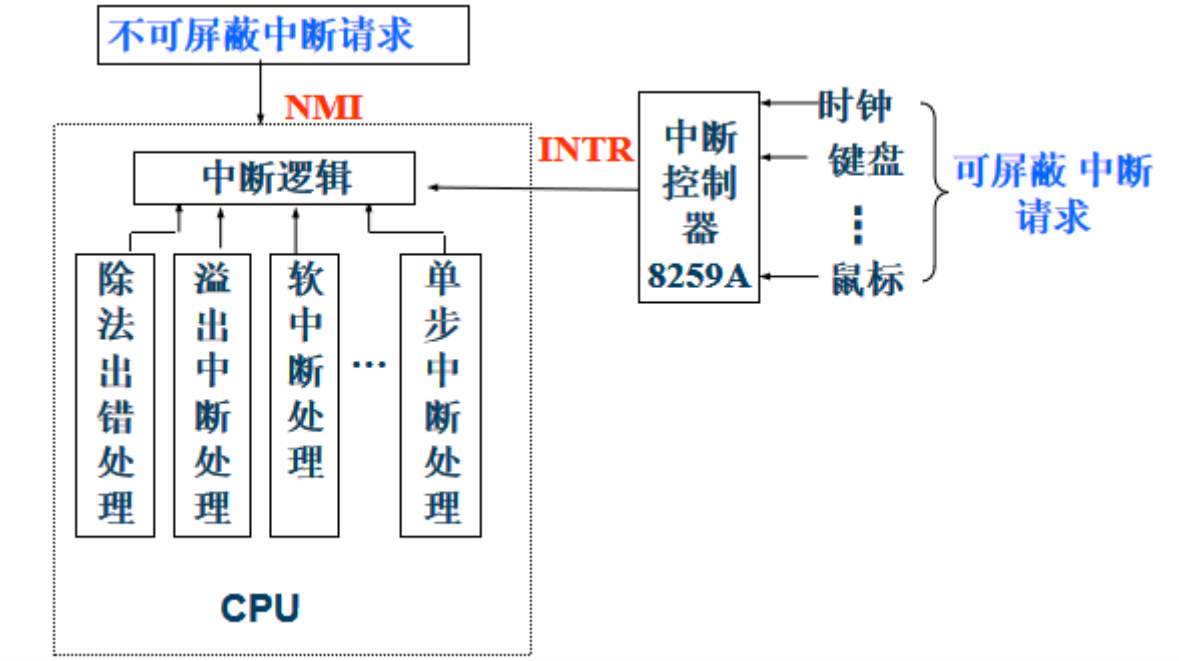


很像之前的子程序调用，但是有区别：

1. 子程序调用是主动打断，但中断是**被动**打断
2. 外部中断是**随机发生**的，不可预测
3. 引起中断的外部事件与CPU当前执行的指令之间**没有明显的约束关系**
   而主程序和子程序之间有密切的联系

80X86系统的中断源分类：





> 可屏蔽中断(INTR)优先级低，可以先不管，让它等一等，也不会出啥大问题。
> 不可屏蔽中断(NMI)优先级高，不处理会导致程序出错

中断过程中会遇到的问题

（1）几个中断同时发生时怎么办?
采用**优先级**的办法.
（2）如何得到中断处理程序的入口地址?
通过中**断号和中断矢量表**(6.2.2节介绍)实现.
（3）怎样编写中断处理程序?
通过相关指令及编写、安装方法 (在6.2.3及6.2.4节介绍).

1、优先级

中断/异常类型	优先级
除调试故障以外的异常 异常指令INTO、INT n、INT 3 对当前指令的调试异常 对下条指令的调试异常 NMI INTR	最高 ↓ 最低

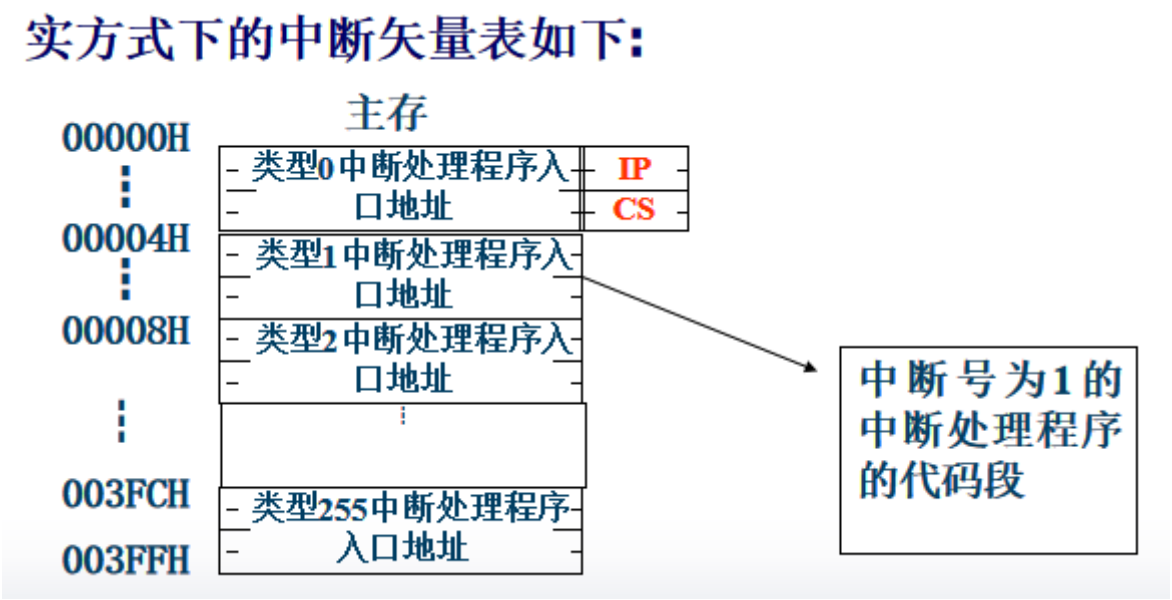
2、中断号（类型码）

0 - 255（一个字节256个），2^8

6.2.2 中断矢量表

是**中断类型码**与**对应的中断处理程序**之间的连接表，存放的是**中断处理程序的入口地址**（也称为中断矢量或中断向量）

> 实方式下1KB，起始位置固定地从物理地址0开始
>
> 256个表项，每个表项4占4B。刚好占1KB主存空间。
>
> 也就是放在内存里



CPU转到中断处理程序的重要步骤是

（1）获取中断类型码n
（2）从中断矢量表中获取入口地址
（0：[n*4]）→IP，
（0：[n*4+2]）→CS[1]
以上2步解决了如何获得入口地址的问题。

（3）返回地址的处理、标志寄存器等的处理。

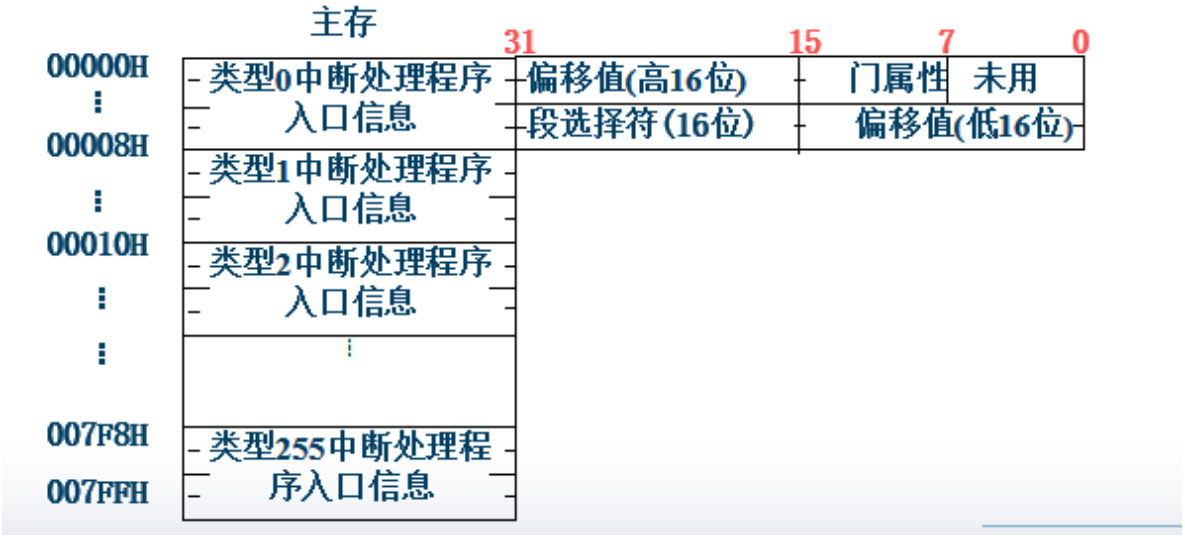
保护方式下的中断矢量表

前面介绍的是实方式下的。

在**保护方式**下，中断矢量表称作**中断描述符表(IDT)**，按照统一的描述符风格定义其中的表项；每个表项(称作门描述符)存放**中断处理程序的入口地址以及类别、权限等信息**，占**8个字节**（256X8B=2KB），共占用**2KB**的主存空间。IDTR决定IDT的起始PA。

> 保护方式，顾名思义，需要权限来保护

具体情况如下图:



- IDT在主存中的位置，是由CPU内的中断描述符寄存器IDTR的内容指示的。
- IDT中的每一个表项都可以称作一个中断门，分任务门、中断门、陷阱门
- 在IDT里显然用n*8做偏移值

6.2.3 软中断及有关的中断指令

软中断通过程序中的软中断指令实现，所以又称它为**程序自中断。**

1. 软中断指令

语句格式：INT n
　　其中，n为中断号，取值范围为0~255。
功能：

1. 实方式：（FLAGS）→↓（SP），0→IF、TF
   32位段：（EFLAGS）→↓（ESP），0→TF，中断门还要将0→IF
2. 实方式：（CS）→↓（SP），（4*n+2）→CS
   32位段：（CS）扩展成32位→↓（ESP），从门或TSS描述符中分离出的段选择符→CS
3. 实方式：（IP）→↓（SP），（4*n）→IP
   32位段：（EIP）→↓（ESP），从门或TSS描述符中分离出的偏移值→EIP

> TF跟踪标志位，IF中断标志位

“INT 21H”用来调用DOS系统功能，每当执行这条指令时，便产生类型号为21H的中断，执行事先安排好的中断处理程序。堆栈中信息的布局情况如下：

中断前的（IP）= 断点(IP) = 返回地址值

也就是顺序压入栈保存现场。

2. 中断返回指令

语句格式：`IRET`
功能：①实方式：↑（SP）→IP
32位段：↑（ESP）→EIP
②实方式：↑（SP）→CS
32位段：↑（ESP）取低16位→CS
③实方式：↑（SP）→FLAGS
32位段：↑（ESP）→EFLAGS

> 前两个恢复断点地址，第三个恢复标志寄存器的内容。

注意：在进行不能打断的操作前一定要先**关闭中断**

> （CLI指令使IF=0，不会响应INTR了，即Close IF，关中断），做完之后STI使IF=1
> **IF=0是关中断状态**，不会响应可屏蔽中断（INTR）

6.2.4 中断处理程序的设计

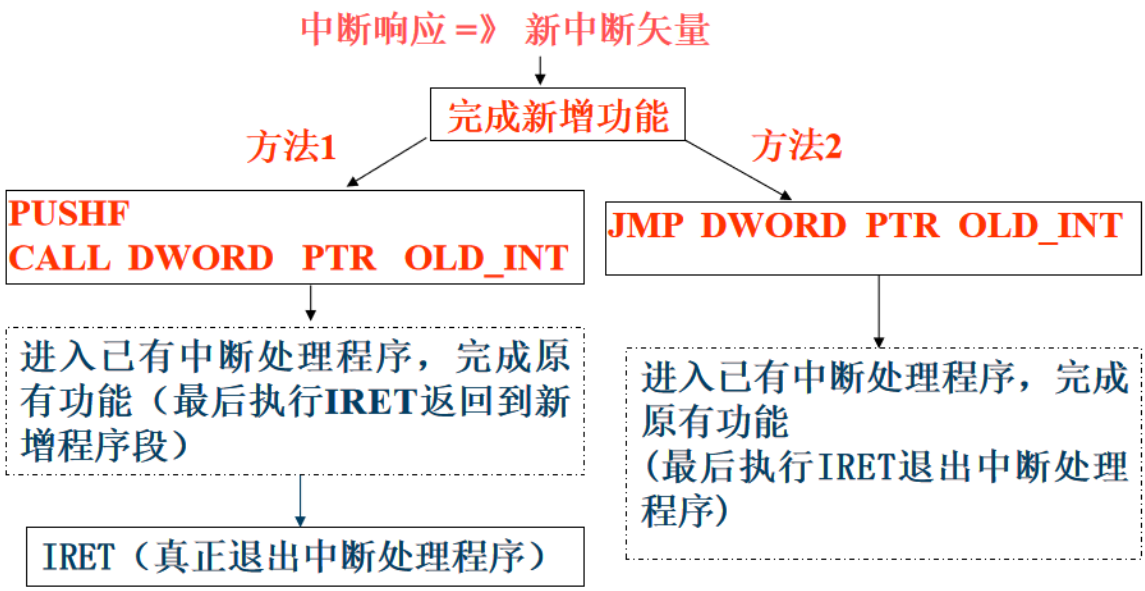
1．新增一个中断处理程序的步骤

① 根据功能编制中断处理程序。
编制方法与子程序的编制方法类似，远过程, IRET。
② 为软中断找到一个**空闲的中断号m**；或根据硬件确定中断号。
③ 将新编制的中断处理程序装入内存，将其入口地址送入中断矢量表4*m*~4m+3的**四个字节**中。
使用方法：**INT m** / 硬件中断 / CPU异常。

2．修改（接管）已有中断处理程序以扩充其功能

① 根据扩充功能的要求编制程序段。
② 将新编制的程序段装入内存，把待扩充功能的已有中断处理程序的**入口地址**复制到新编制的程序段中，用**新编制程序段的入口地址取代**中断矢量表中已有中断处理程序的入口地址。

> 直接给换个入口地址，用新入口地址换旧入口地址



采用方法1时，最后执行IRET返回新增程序段，还可以执行新增功能

采用方法2时，最后执行IRET退出中断处理程序，如果新功能在中断处理程序之前执行，那么没事，新功能可以执行。如果在中断程序之后执行，那么由于IRET直接退出中断处理程序，不会跳到新增程序段，就无法执行新增功能。

3. 装入方式

1 直接装入

实方式下，用传送指令即可

由中断号n，计算中断向量表的地址

```
;假设这个中断程序叫INTRR
OFFSET INTRR -> 0:[n*4]
SEG INTRR -> 0:[n*4+2]
;如下
MOV AX,0
MOV DS,AX
CLI;要先关中断
MOV WORD PTR DS:[n*4], OFFSET INTRR
MOV WORD PTR DS:[n*4+2], SEG INTRR
STI;开中断
```

2 用系统功能调用转入

25H号DOS功能调用来设置中断向量

功能：设置中断向量，建立由(AL)指定类型号的软中断指令。

调用格式：

```
MOV AX,SEG INTRR;中断处理程序段首址
MOV DS,AX
LEA DX,INTRR ;或OFFSET DX,INTRR
MOV AH,25H;25H号调用
MOV AL,n;n是空闲的中断号
INT 21
//好像默认装入 DS:[DX]?,约定好的?
```

6.3 浮点运算

不考

6.4 WIN32编程

WIN32程序：基于Windows运行环境的32位段程序。

在DOS下显示“How are you!”的程序

16位的

```

```

```
.386
STACK  SEGMENT  STACK  USE16
        DB  200 DUP(0)
STACK  ENDS
DATA  SEGMENT  USE16
    Hello DB 'How are you! $'
DATA  ENDS
CODE  SEGMENT  USE16
BEGIN:  MOV  AX,DATA
       MOV  DS,  AX
       LEA    DX,  hello
       MOV  AH,  9
       INT    21H
       MOV  AH, 4CH
       INT    21H
CODE  ENDS
END  BEGIN
```

如何改成在Windows下运行的32位程序?

```
.386
MessageBoxA PROTO STDCALL :DWORD,:DWORD,:DWORD,:DWORD
ExitProcess     PROTO STDCALL :DWORD
includelib  user32.lib
includelib  kernel32.lib
DATA   SEGMENT
    hello  DB "HOW ARE YOU!",0
    dir     DB "问候",0
DATA   ENDS
STACK  SEGMENT STACK
       DB  1000 DUP(0)
STACK  ENDS
CODE  SEGMENT
       ASSUME CS:CODE, DS:DATA, SS:STACK
_START: PUSH 0
               PUSH OFFSET dir
               PUSH OFFSET hello
               PUSH 0
               CALL MessageBoxA
               PUSH 0
               CALL ExitProcess
CODE   ENDS
END _START
```

> 这个没整明白，算了。

宏汇编语言对WIN32编程的支持

1．段的简化定义

① 存储模型说明伪指令 .MODEL

格式： .MODEL 存储模型 [,语言类型][,系统类型][,堆栈选项]

功能:

1. “存储模型”指定内存管理模式，

> 常用的存储类型见P240
>
> SMALL模式 16位段 代码和数据在各自的64KB段内，代码和数据的总量均不超过64KB
> HUGE 模式 32位段 代码和数据等全放在一个4GB空间内

2. “语言类型”，C、PASCAL或STDCALL等

   一般应选用STDCALL类型

   STDCALL类型：采用堆栈法传递参数，参数进栈次序为：函数原型描述的参数中最右边的参数最先入栈、最左边的最后入栈；由被调用者在返回时清除参数占用的堆栈空间

要求：该指令必须放在源文件中所有其它段定义伪指令之前且**只能使用一次。**

②定义代码段伪指令 .CODE

格式：.CODE [段名]
功能：说明一个代码段的开始，同时也表示上一个段的结束。如果指定了段名，则该段就以此名字命名。否则，在TINY、SMALL、COMPACT和FLAT模式时，段名为“_TEXT”。

③定义数据段伪指令 .DATA

格式：.DATA 或 .DATA?
功能：说明一个数据段的开始，**同时也表示上一个段的结束。**

- .DATA定义数据段，段内的变量是经过初始化的，都会占用执行文件的磁盘存储空间（即使变量的初始值为？）；其**段名被指定为_DATA。**

  _
- _.DATA？定义数据段，段内的变量是未初始化的，可以减少执行文件的磁盘存储空间并能增强与其它语言的兼容性；其段名被指定为_BSS。

④定义堆栈段伪指令.STACK

.stack 200h ; 定义堆栈段大小为200，直接用这一句就行了

源程序的最后仍需要用END伪指令结束。

2．原型说明与函数调用

已学过的伪指令PUBLIC、EXTRN、PROC和指令CALL。

新的伪指令PROTO和INVOKE，使子程序的说明/调用形式类似于高级语言。

①原型说明伪指令PROTO

格式：
函数名 PROTO `[函数类型][语言类型][[参数名]:参数类型],[[参数名]:参数类型]...`
功能：**用于说明本模块中要调用的过程或函数。**

“函数类型”:WIN32应用程序中一般为NEAR类型。
“语言类型”.MODEL语句后指定了语言类型，此处就可省略。
  参数名可以省略，但冒号和参数类型不能省略。

②完整的函数定义伪指令PROC

格式：
函数名 PROC [函数类型][语言类型][USES 寄存器表][,参数名[:类型]]...
功能：定义一个新的函数（函数体应紧跟其后）。参数名是用来传递数据的，可以在程序中直接引用，故不能省略。

3．结构

本章小结

（1）输入输出指令IN、OUT的使用格式及功能；
（2）中断的概念，中断矢量表，实方式下中断处理程序的编制方法；
（3）段的简化定义方法；
.MODEL/.CODE/.DATA/.STACK
（4）结构的定义与使用方法；
STRUCT
(5)原型定义与函数调用；
（6）基于窗口的WIN32程序的结构、功能和特点，基本的程序设计方法。

实际上有可能考的部分就是中断，以及输入输出指令

---

1. 这玩意儿起始位置固定地从物理地址0开始。 ↩