第六章
输入/输出和WIN32编程

一、本章的学习内容

本章学习在几种新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。

三、本章学习的难点

（1）输入输出指令中的地址表示方法;

（2）中断矢量表的作用、存取方法;

（3）中断处理程序的安装、驻留、调试;

（4）存储模型的理解和应用;

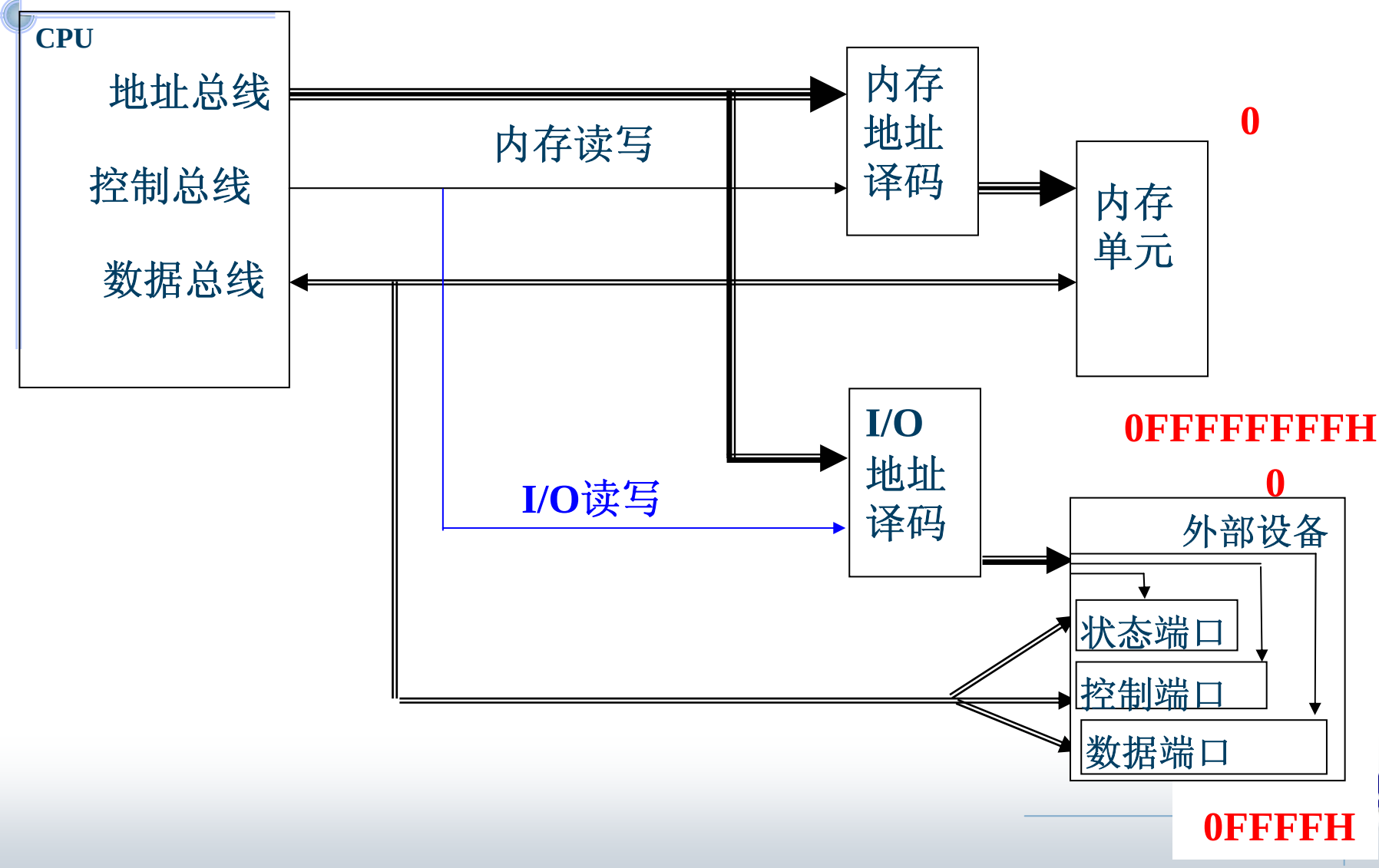
（5）不带变量名字的结构成员的访问方法;

（6）基于窗口的WIN32程序执行流程、消息驱动机制。

6.1 输入/输出指令和数据的传送方式

華中科技大學

- PC机IO端口分配情况：



6.1.1输入/输出指令

输入：

从外部设备向计算机传送信息，也即将外部设备寄存器中的数据送至累加器AL/AX/EAX或主存储器中；

输出：

从计算机向外部设备传送信息，也即将AL/AX/EAX或主存储器中的数据送至外设寄存器中。

1. 输入指令 IN

语句格式: IN OPD, OPS

功能: (OPS) → 累加器OPD

说明:

①当外设寄存器的地址<=255时, OPS = 立即数或者DX表示待访问的端口地址。

当外设寄存器的地址 >255时, OPS只能用DX表示。

② OPD只能是累加器AL、AX或EAX。

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。

2. 输出指令 OUT

语句格式： OUT OPD，OPS

功能：累加器（OPS）→OPD

说明：

① OPD = 立即数 或者DX

② OPS只能是累加器AL、AX或EAX。

即：OUT OPD， AL/AX/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个单元中。

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；

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。

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

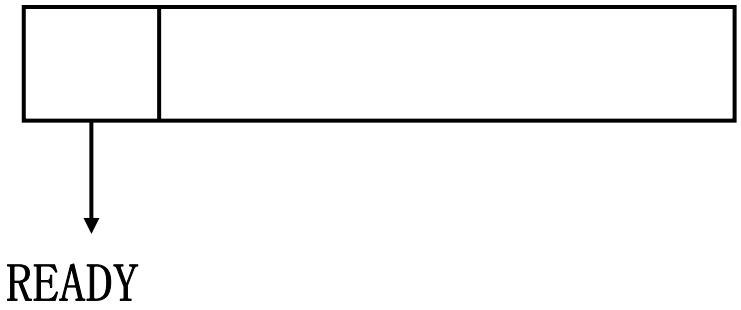
在实方式下，I/O空间的访问没有特殊的限制，

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

2、查询传送方式



输入状态寄存器“READY”位为1时表示要输入的数据已准备好。

查询式输入的程序段INPROG如下：

```
INPROG：IN   AL，STATUS_PORT  ；从状态寄存器输入状态信息
        TEST  AL，80H         ；检查“READY”位
        JZ    INPROG          ；未准备好,转INPROG
        IN    AL，DATA_PORT   ；已准备好,从数据寄存器输入数据送入AL
```

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。

4. 中断传送方式（具体见下一节）

6.2 中断与异常

6.2.1 中断的概念

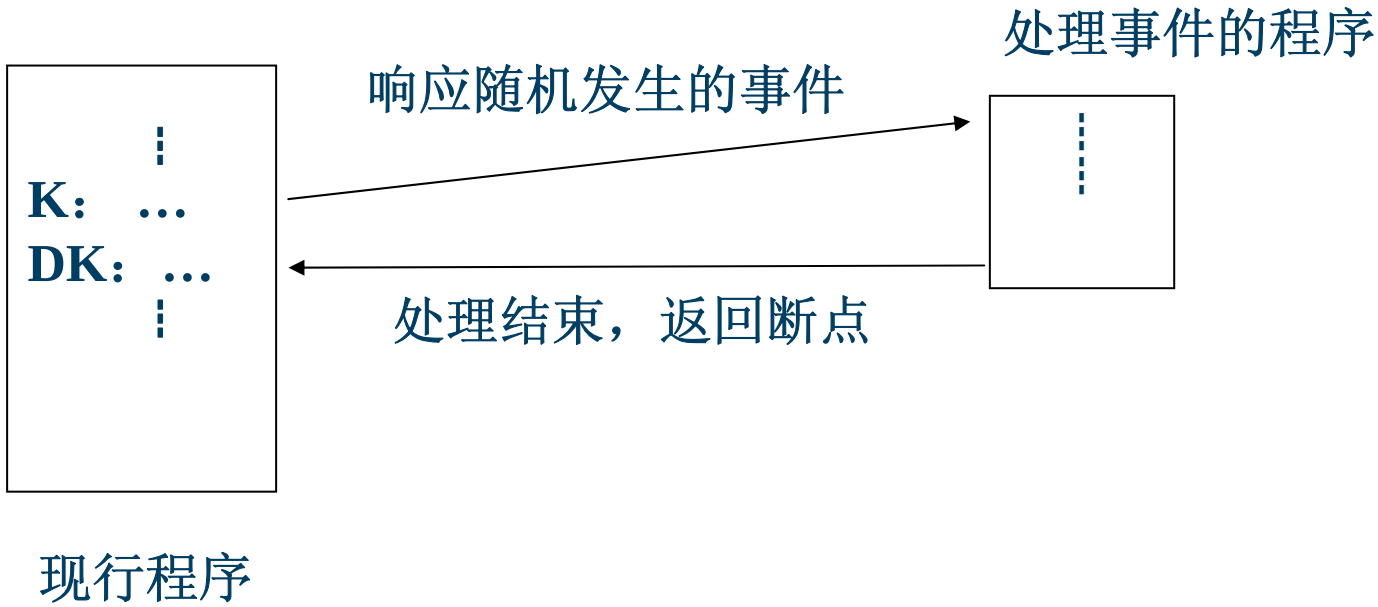
中断：是CPU所具有的能打断当前执行的程序，转而为临时出现的事件服务，事后又能自动按要求恢复执行原来程序的一种功能。

中断系统：实现这种功能的软、硬件装置 。

中断处理程序（或中断服务程序）：某事件的处理程序。

中断源：引起中断的事件。

中断处理过程如下：



注意与子程序的区别

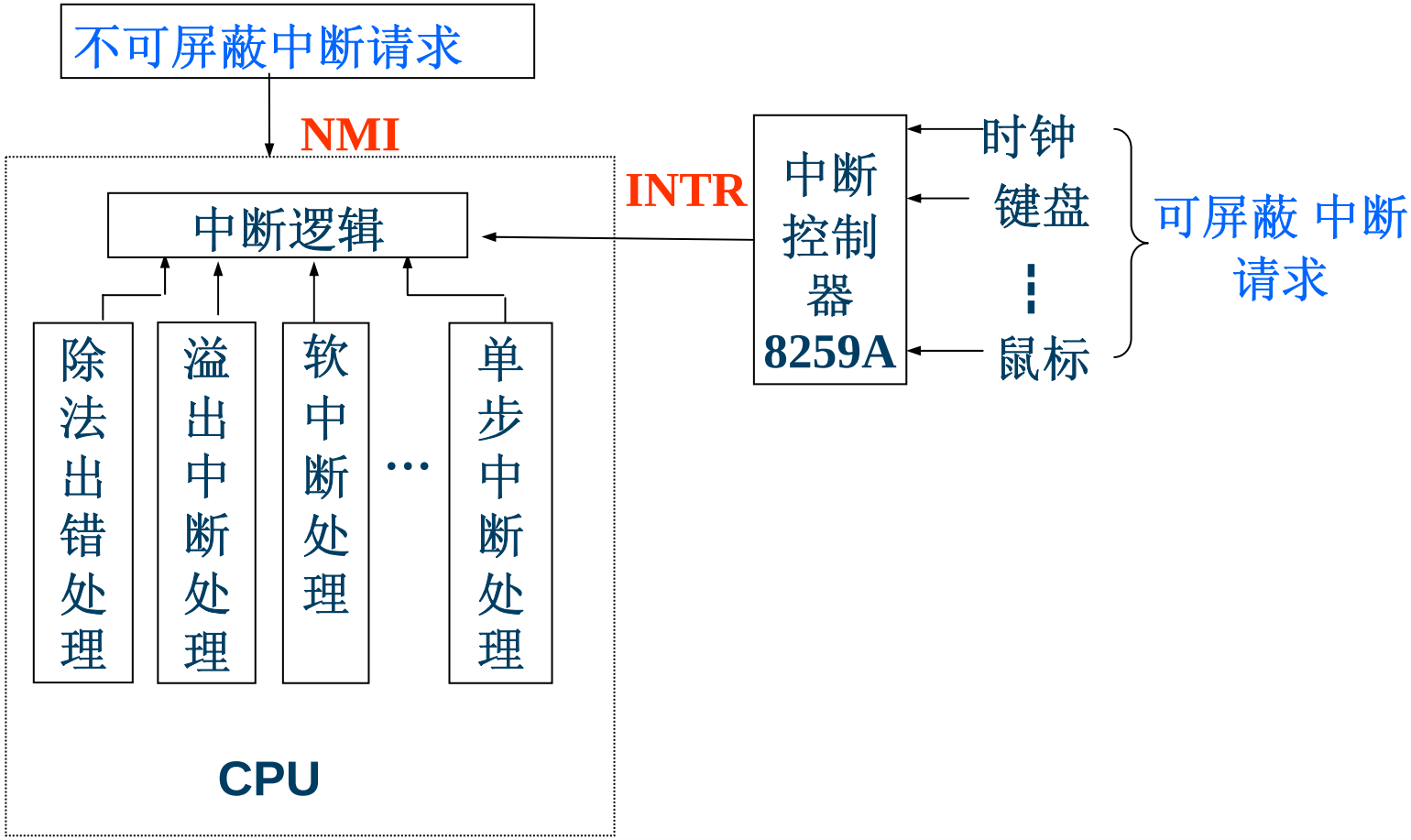
实现数据的中断传送方式时需要完成的步骤：

安装服务程序、初始化硬件、做别的事情、中断来时处理IO.

80X86系统的中断源分类：

中断源

- 外部中断（中断，随机性）
  - 不可屏蔽中断NMI:
    电源掉电、存储器出错或者总线奇偶校验错
  - 可屏蔽中断INTR:
    开中断状态（STI，IF=1）；CLI
- 内部中断（异常，与CPU的状态和当前执行的指令有关）
  - CPU检测：
    如除法出错、单步中断、协处理器段超越等。
  - 程序检测：
    软中断，包括指令INTO、INT n和BOUND等。


不可屏蔽中断请求
NMI
INTR
中断逻辑
中断控制器8259A
时钟
键盘
鼠标
可屏蔽 中断请求
除法出错处理
溢出中断处理
软中断处理
单步中断处理
CPU

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）

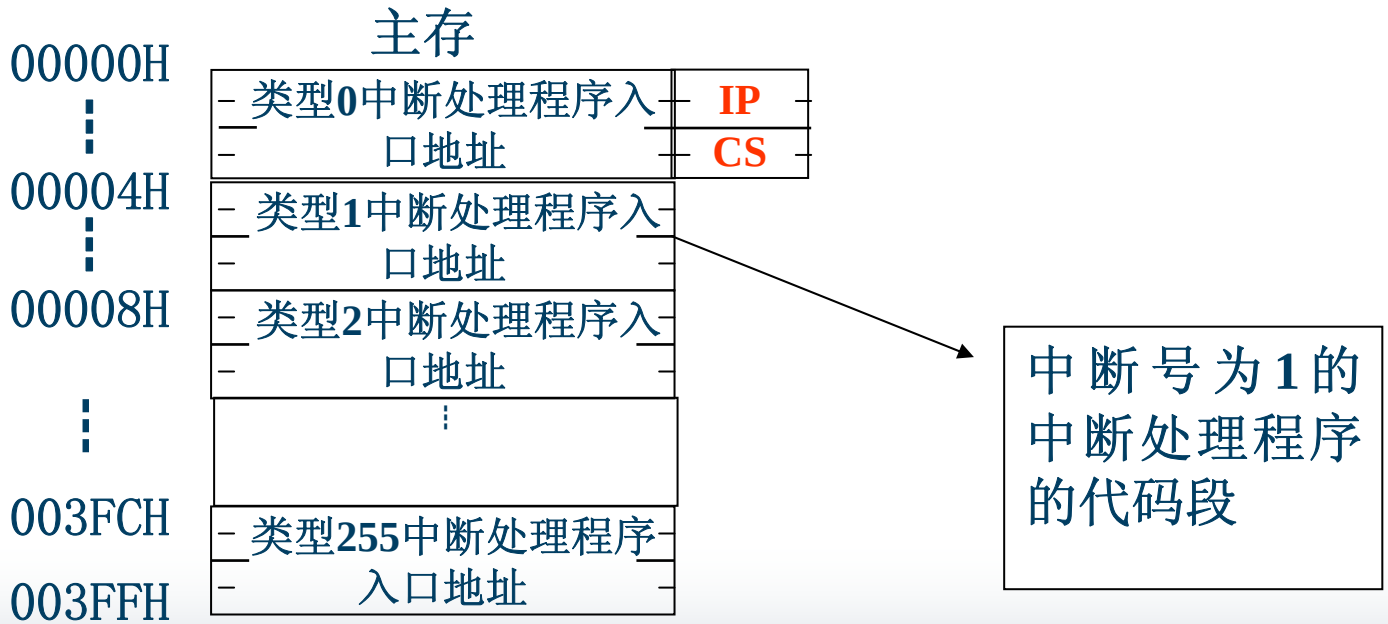
中断号	名　称	类型	相关指令	DOS下名称
0	除法出错	异常	DIV，IDIV	除法出错
1	调试异常	异常	任何指令	单步
2	非屏蔽中断	中断	-	非屏蔽中断
3	断点	异常	INT 3	断点
4	溢出	异常	INTO	溢出
5	边界检查	异常	BOUND	打印屏幕
6	非法操作码	异常	非法指令编码或操作数	保留
7	协处理器无效	异常	浮点指令或WAIT	保留

8	双重故障	异常	任何指令	时钟中断
9	协处理器段超越	异常	访问存储器的浮点指令	键盘中断
0DH	通用保护异常	异常	任何访问存储器的指令 任何特权指令	硬盘（并行口）中断
10H	协处理器出错	异常	浮点指令或WAIT	显示器驱动程序
13H	保留			软盘驱动程序
14H	保留			串口驱动程序
16H	保留			键盘驱动程序
17H	保留			打印驱动程序
19H	保留			系统自举程序
1AH	保留			时钟管理
1CH	保留			定时处理
20H~2FH	其它软/硬件中断			DOS使用
0~0FFH	软中断	异常	INT n	软中断

6.2.2 中断矢量表

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）

实方式下的中断矢量表如下：



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。

以上2步解决了如何获得入口地址的问题。

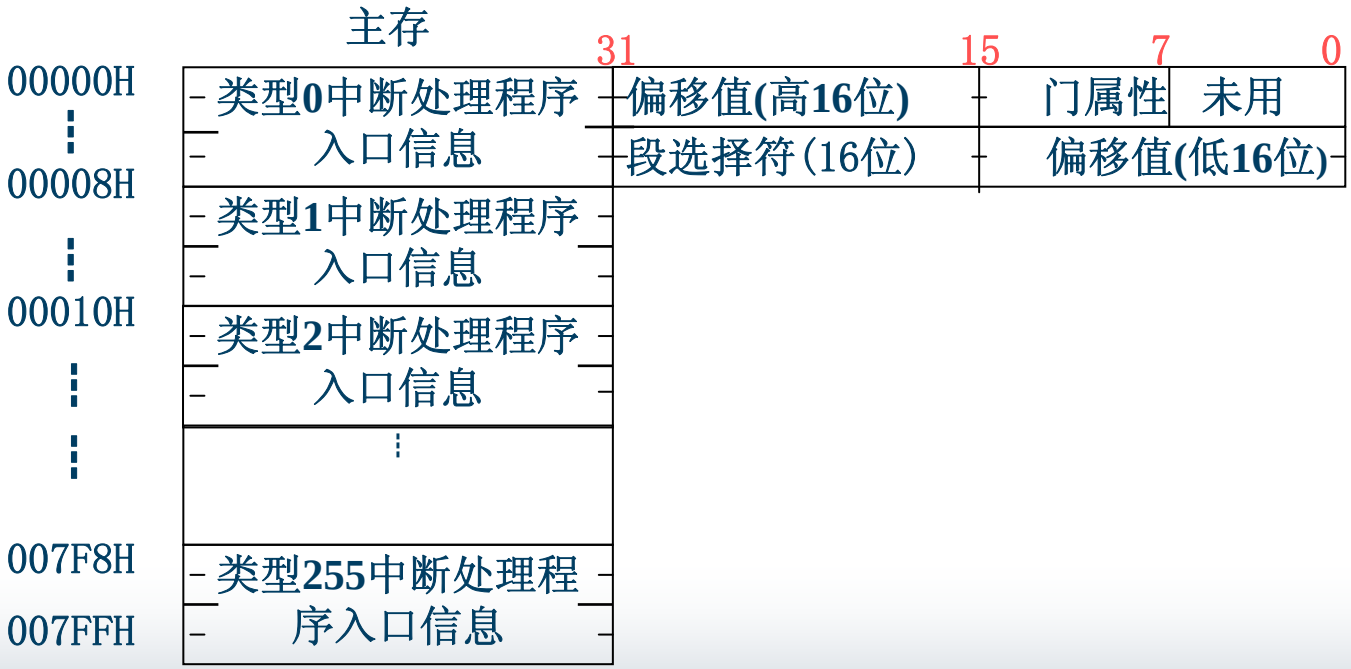
（3）返回地址的处理、标志寄存器等的处理。

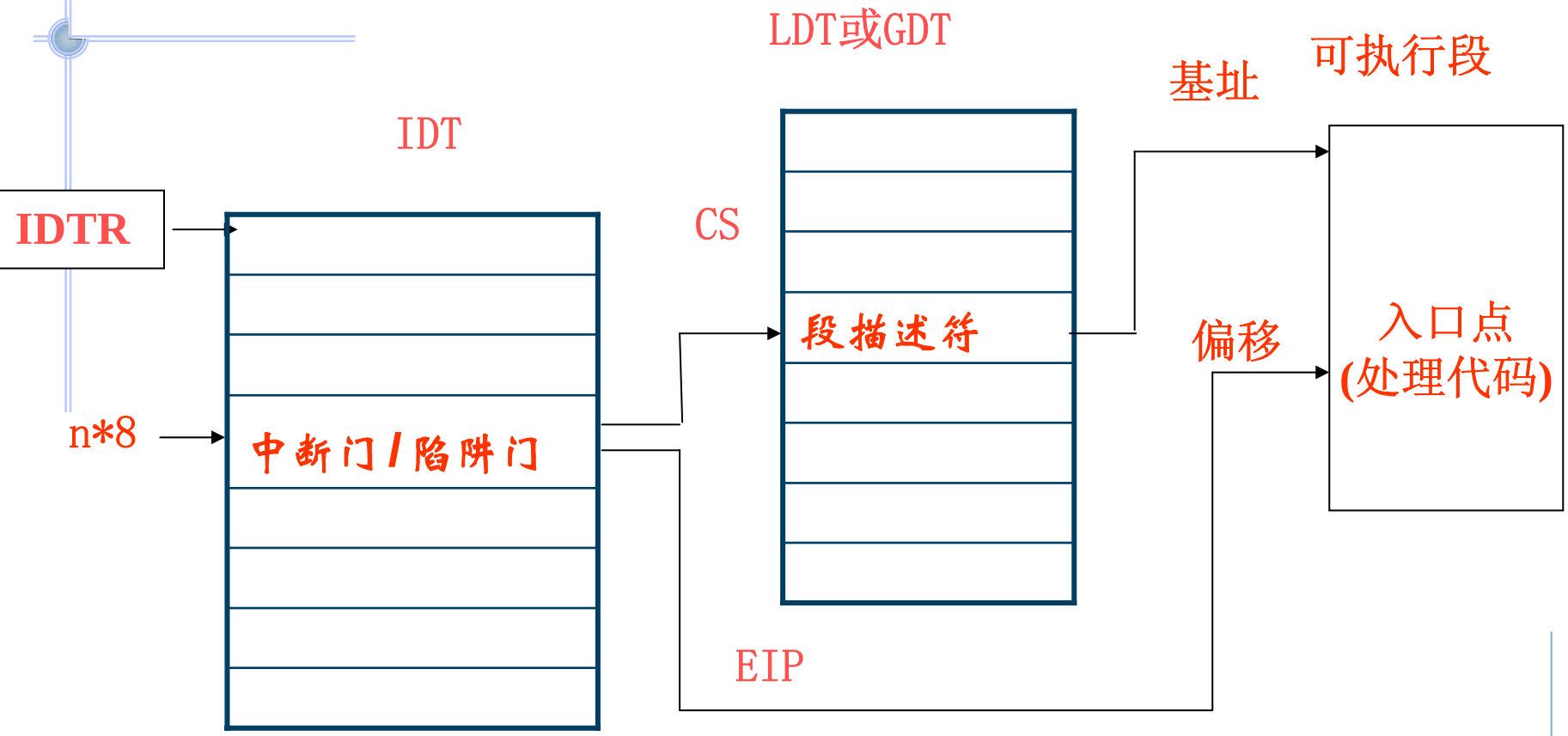
例：n=2对应的中断处理程序的入口地址值存放在物理地址范围为00008H-0000BH的内存中，如何将其读出来看看？

```
MOV  AX，0
MOV  DS，AX
MOV  AX，[0008H]    ；访问DS：[2*4]单元，
                      即0：0008H单元
MOV  BX，[000AH]
```

保护方式下的中断矢量表

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，共占用2KB的主存空间。IDTR决定IDT的起始PA。具体情况如下图：





怎样编写中断处理程序：相关指令，编写、安装方法等等，见下面内容

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），

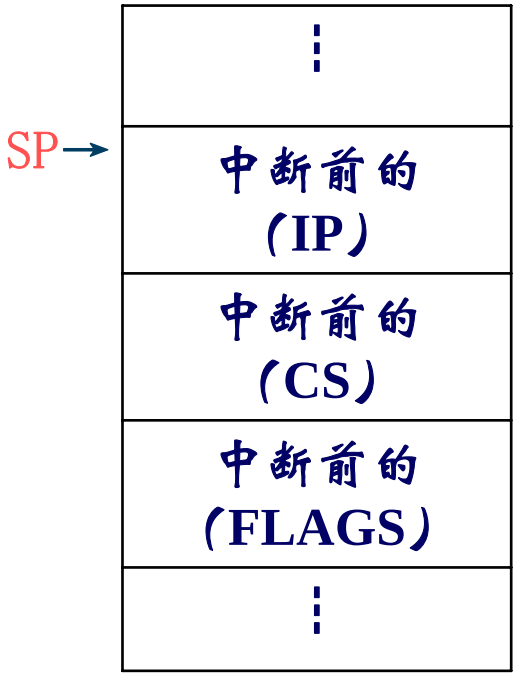
从门或TSS描述符中分离出的段选择符→CS

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

例：“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

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

恢复断点地址

恢复标志寄存器的内容

CPU对中断的响应一般是在当前指令执行完后才去响应，所以，通过分析各种中断源的特点，得出：程序设计中重点关注可屏蔽中断源INTR可能对程序执行结果的影响。

例

```
…
MOV  SS，AX
MOV  SP，BX
…
```

```
CLI
MOV  SS，AX
MOV  SP，BX
STI
```

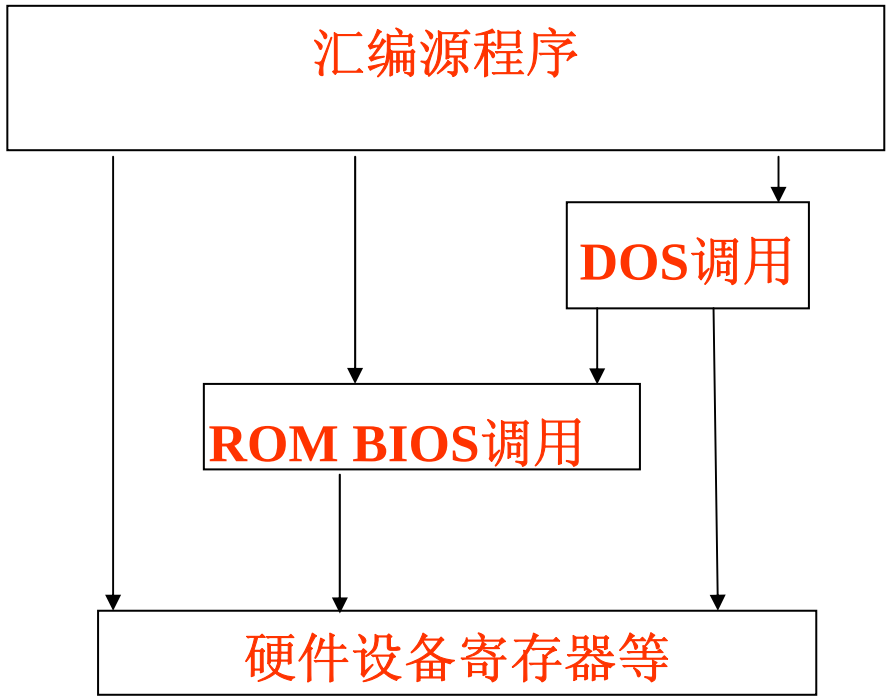
在进行不能打断的操作前一定要先关闭中断
（CLI指令使IF=0，不会响应INTR了），做完之后STI。

软中断的实例：操作系统提供了INT 21H 等软中断，完成了很多系统功能；ROM BIOS 也提供了很多系统驱动程序。BIOS软中断的功能和入口、出口参数见附录Ⅴ。

实方式下：

0000:0-03FFH	中断矢量表
0000:0400H	DOS系统区
	用户程序区
A000:0000	显示缓冲区
E000:0000-	ROM BIOS

编程时混合接口界面：



6.2.4 中断处理程序的设计

中断处理程序的设计主要包括：

为尚未分配功能的中断号设计一个中断处理程序（新增）；

修改已有的中断处理程序以扩充其功能。

下面以实方式为例进行介绍。

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~4*m+3的四个字节中。

使用方法：INT m / 硬件中断 / CPU异常。

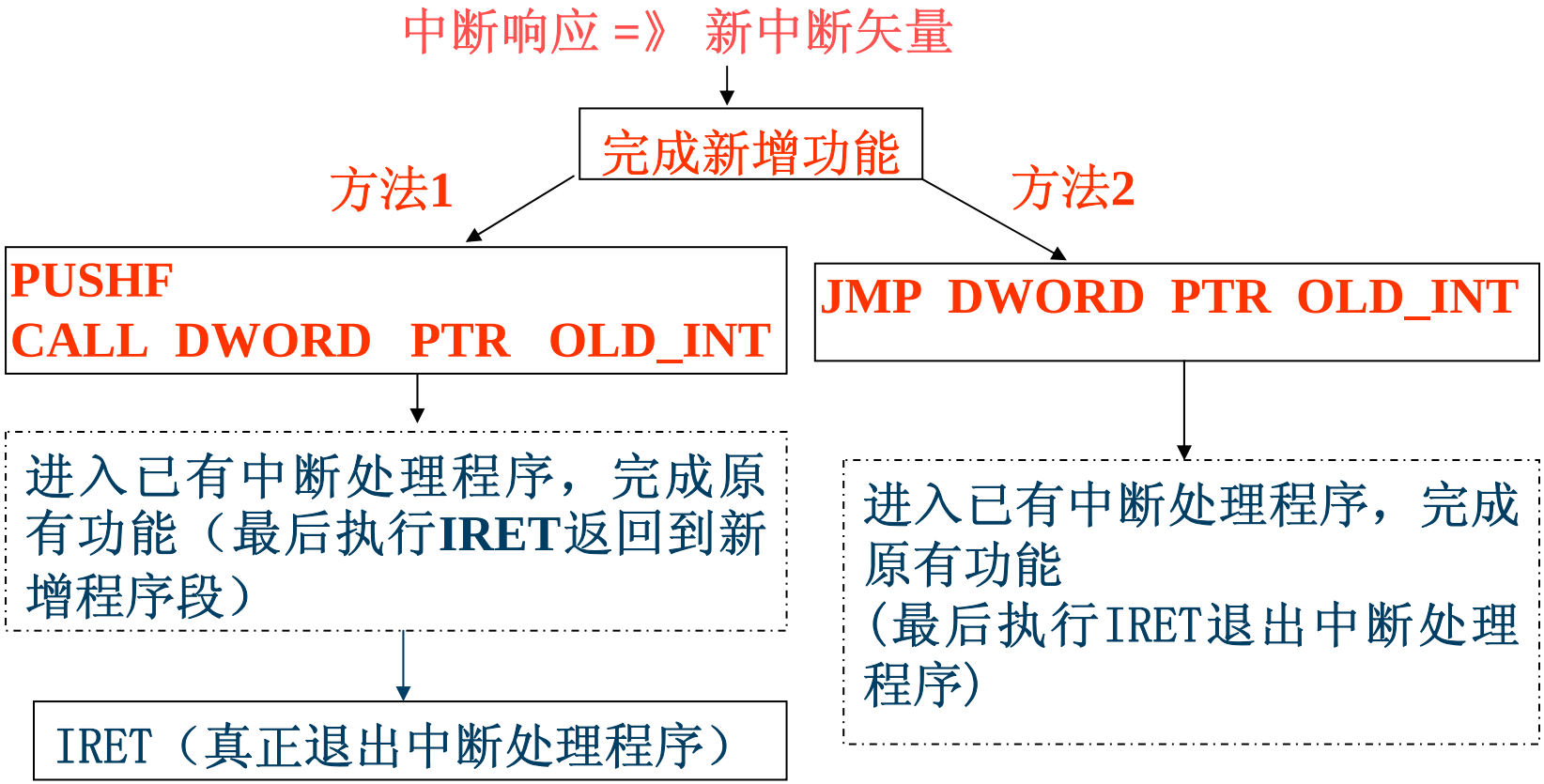
2、修改（接管）已有中断处理程序以扩充其功能

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。

有两种切换方法如下：

两种切换方法


中断响应 =》 新中断矢量
完成新增功能
方法1
方法2
PUSHF
CALL DWORD PTR OLD_INT
JMP DWORD PTR OLD_INT
进入已有中断处理程序，完成原有功能（最后执行IRET返回到新增程序段）
进入已有中断处理程序，完成原有功能
（最后执行IRET退出中断处理程序）
IRET（真正退出中断处理程序）

例：请通过修改ROM BIOS提供的磁盘驱动程序（软中断“INT 13H”）实现对磁盘的写保护。

解：在“INT 13H”的入口参数中，AH用来指定功能号，当（AH）=3或0BH时，通过软中断“INT 13H”可将主存中的内容写到指定磁盘的扇区中。

接管“INT 13H”的新源程序如下：

```
CODE     SEGMENT
         ASSUME CS:CODE,SS:STACK
OLD_INT  DW ?,?                ；存放老中断向量
```

;新中断处理程序（“INT 13H”）的代码

```
NEW13H:  CMP   AH,3          ;      判断是否调用第一种磁盘写功能
         JE    QUIT
         CMP   AH,0BH        ;      判断是否调用第二种磁盘写功能
         JE    QUIT
         JMP  DWORD PTR CS:OLD_INT  ; 允许则继续执行
                                    原中断处理程序功能
QUIT:    PUSH  BP
         MOV   BP,SP
         OR    WORD PTR [BP]+6,01H    ; 1→CF，表示出错。
                                修改了中断返回后的标志寄存器内容
         POP   BP
         MOV   AH,3          ;      错误号为写保护
         IRET                ;      弹出的标志寄存器的CF值已
                                    被修改为1
```

；初始化（安装新中断向量并常驻内存）程序

```
START:  XOR   AX,AX
        MOV   DS, AX                    ；0→DS
        MOV   AX, DS:[13H*4]            ；取原“INT 13H”的中
                                          断矢量的偏移部分
        MOV   CS:OLD_INT, AX            ；将偏移部分保存
        MOV   AX, DS:[13H*4+2]          ；取原“INT 13H”的中断矢量
                                          的段值
        MOV   CS:OLD_INT+2, AX          ；将段值保存
        CLI                             ；修改中断矢量表时必须关中断，
                                          防止中途被外部中断打断而导致出错
        MOV   WORD PTR DS:[13H*4], OFFSET NEW13H
                                        ；将新的偏移值送到中断矢量表
        MOV   DS:[13H*4+2], CS          ；将新的段值送到中断矢量表
        STI                             ；开中断
        MOV   DX, OFFSET START+15       ；计算中断处理程序占用
                                          的字节数，+15是为了在计算的时候能
                                          向上取整。
        MOV   CL, 4
```

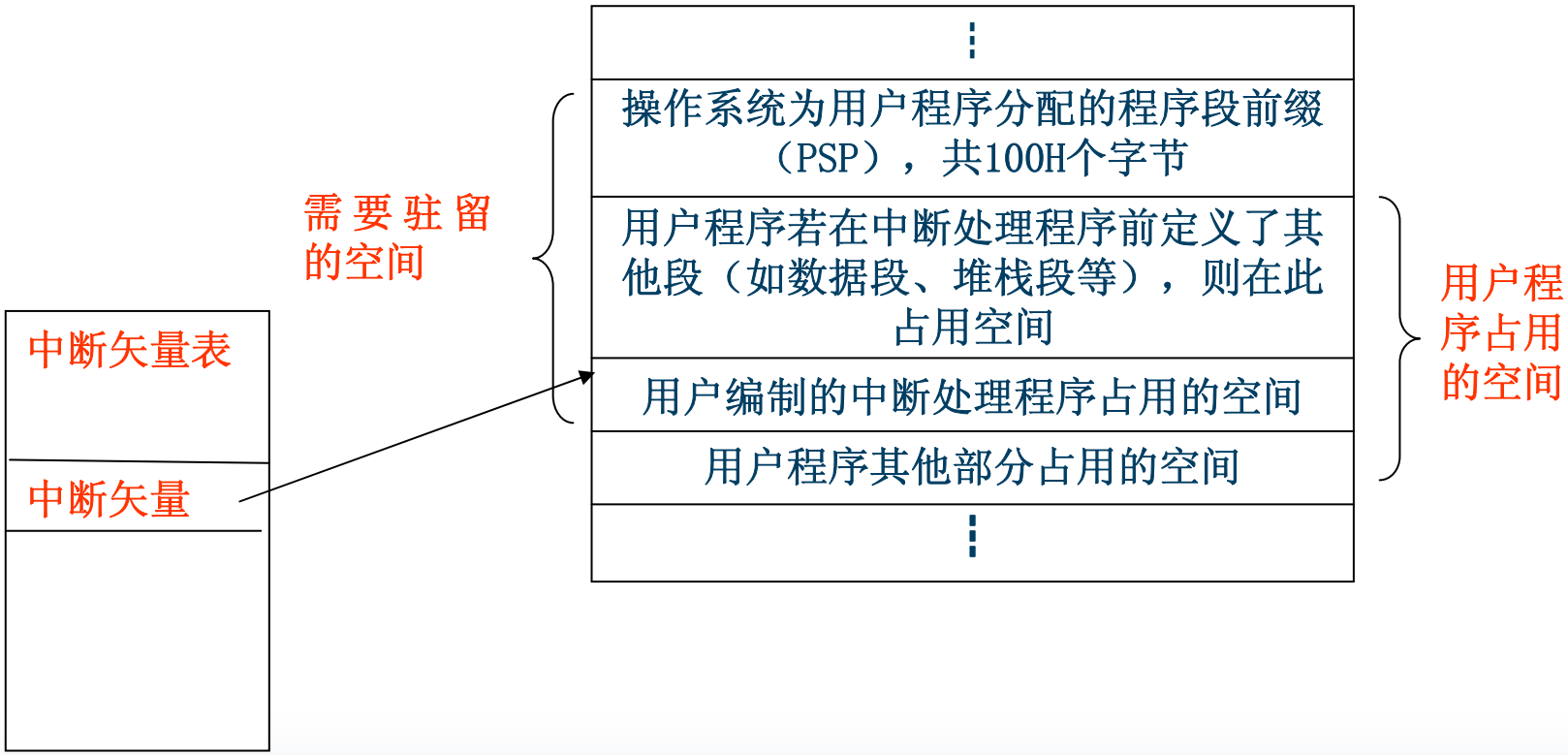
```
        SHR   DX, CL      ; 把字节数换算成节数（每节代表16个字节）
        ADD   DX, 10H     ; 驻留的长度还需包括程序段前缀的内容
                          （100H个字节）
        MOV   AL, 0       ; 退出码为0
        MOV   AH, 31H     ; 退出时,将（DX）节的主存单元驻留(不释放)
        INT   21H
CODE    ENDS

STACK   SEGMENT  STACK
        DB  200 DUP(0)
STACK   ENDS
        END   START
```

说明：

（1）中断矢量的获取与设置方法有两种。

（2）驻留与恢复中断矢量如下图所示。



（3）中断处理程序一般使用主程序的堆栈段。

例：时钟显示程序。要求每隔约1秒钟在屏幕右上角显示一次当前的时间（时：分：秒）。

分析：

1、类型码为8的中断是系统时钟中断，系统定时器被初始化成每秒产生18.2次中断。

2、I/O端口（地址为70H和71H）直接读取“实时时钟/系统配置接口芯片（RT/CMOS RAM）”内部内容的方法

(RT/CMOS RAM内部有64个字节单元。)

```
MOV  AL，4        ；4是“时”信息的偏移地址
OUT  70H，AL      ；设定将要访问的单元是偏移值为4的“时”信息
                    （完成切换）
JMP  $+2          ；延时，保证端口操作的可靠性（因端口电路
                    的响应速度较慢）
 IN  AL,71H       ；读取“时”信息
```

3、INT 10H是ROM BIOS提供的显示驱动程序。

4、INT 16H ；键盘软中断的入口和出口参数见附录Ⅴ

5、INT 21H ；获取/设置中断35/25H的入口/出口参数见附录Ⅳ

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


```
.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
```

问题思考

（1）如何改成32位段程序？

（2）如何改成在Windows下运行的32位程序？

改写后的程序

```
.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          ;起始地址的标号前带下划线
```

问题（1）：WIN-API是高级语言函数形式，放在子程序库中；参数多，取值复杂

例：

```
int MessageBoxA(
   HWND    hWnd,      //所属窗口的句柄，=0没有所属窗口
   LPCTSTR  lpText,    //待显示字符串（结束符为0）的首地址
   LPCTSTR  lpCaption, //消息框标题字符串（结束符为0）的首地 ，址指
                       针；为0时显示标题“Error”
   UINT uType          //指定消息框的样式
)
```

```
MessageBoxA  PROTO
        hWnd     :DWOR
        lpText    :DWORD,
        lpCaption :DWORD,
        uType     :DWORD
```

其中，uType指定了在消息框中要显示哪些按钮和图标，例：

=0，仅显示“确认”按钮（MB_OK EQU 0）

=1，同时显示“确认”和“取消”按钮（MB_OKCANCEL=1）

=3， 同时显示“是”、“否”和“取消”按钮（MB_YESNOCANCEL EQU 3）

解决办法

（1）将这些符号常量、函数原型的定义集中放到头文件中，用“INCLUDE 头文件”的方法使用。

windows.inc，kernel32.inc， kernel32.lib；user32.inc，user32.lib；gdi32.inc，gdi32.lib

（2）使用结构STRUCT等数据类型表达复杂的参数。

（3）提供新的子程序定义、调用等方法，使WIN-API调用过程更加直观，不易出错。

问题（2）：32位段使程序可以直接访问所有内存空间，一般不必用定义多个段的方法获得较大的空间，因此，完整的段定义方法显得复杂不必要。

解决办法：采用简化段定义的方法，定义三个基本种类的段（CODE、DATA、STACK）。

宏汇编语言对WIN32编程的支持

1. 段的简化定义

2. 原型说明与函数调用

3. 结构

1. 段的简化定义

① 存储模型说明伪指令 .MODEL

②定义代码段伪指令 .CODE

③定义数据段伪指令 .DATA

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：“存储模型”指定内存管理模式，“语言类型”指定了函数命名、调用和返回的方法，例如C、PASCAL或STDCALL等。

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。

不同“存储模型”规格表

存储 模型	段的大小	代码访问范围	数据访问范围	备注
TINY	16位	NEAR	NEAR	代码和数据全部放在同一个64K段内，常用于生成.COM程序
SMALL	16位	NEAR	NEAR	代码和数据在各自的64K段内，代码总量和数据总量均不超过64K
COMPACT	16位	NEAR	FAR	代码总量不超过64K，数据总量可以超过64K
MEDIUM	16位	FAR	NEAR	代码总量可超过64K，数据总量不超过64K
LARGE	16位	FAR	FAR	代码和数据总量均可超过64K，但单个数组不超过64K
HUGE	16位	FAR	FAR	代码和数据总量均可超过64K，单个数组可超过64K
FLAT	32位	NEAR	NEAR	代码和数据全部放在同一个4G空间内

②定义代码段伪指令

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

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。

堆栈段定义： .STACK [堆栈字节数]；（省略时为1024）

常数（只读）数据段定义： .CONST；

源程序的最后仍需要用END伪指令结束。

FLAT下的程序启动时，DS、ES、SS已被设为同一个有效值。

```
.386
. MODEL FLAT, STDCALL
 MessageBoxA PROTO STDCALL :DWORD,:DWORD, :DWORD,:DWORD
 ExitProcess    PROTO STDCALL :DWORD
 includelib  user32.lib
 includelib  kernel32.lib
. DATA
  hello  DB   "HOW ARE YOU!",0
  dir      DB    "问候",0
. STACK 200
. CODE
 START:  PUSH 0
            PUSH OFFSET dir
            PUSH OFFSET hello
            PUSH 0
            CALL MessageBoxA
            PUSH 0
            CALL ExitProcess
END START
```

2. 原型说明与函数调用

相近的新旧伪指令

原型说明伪指令PROTO

完整的函数定义伪指令PROC

函数调用伪指令INVOKE

相近的新旧伪指令

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。

例如： RADIX_S PROTO FAR C
LpResult:WORD,Radix:DWORD,:DWORD
MessageBoxA PROTO hWnd:DWORD，lpText:DWORD，
lpCaption :DWORD，uType:DWORD

“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。

USES后面所列的寄存器是需要在子程序中入栈保护的（由汇编程序自动加入入栈保护和出栈恢复的指令语句）。

局部变量的定义（紧跟在PROC语句之后）：

LOCAL 变量名[[数量]][:类型]

*括号中的“数量”用于说明重复单元的数量，类似DUP的效果。局部变量所指的单元在堆栈中。

举例说明

```
LOCAL   array[20]:BYTE   ; 汇编程序自动增加代码，完
                         ; 成在堆栈中分配以array为首
                         ; 址的20个字节空间
LOCAL   tFlag:WORD       ; 在堆栈中为变量tFlag分配
                         ; 一个字的空间

MOV    AX, tFlag
MOV    AX, n [EBP]
```

③ 函数调用伪指令INVOKE

格式：

INVOKE 函数名 [,参数]...

功能：调用函数。其中，参数可以是各种表达式。

INVOKE必须在PROTO语句或完整的PROC语句说明之后使用。

例如：

INVOKE RADIX_S,SI,10,EAX

INVOKE MessageBoxA，0，OFFSET dir，OFFSET hello MB_OK

INVOKE与CALL都完成了子程序的调用

ADDR 与 OFFSET：

INVOKE MessageBoxA，0，addr dir，ADDR hello，MB_OK

3、结构

（1）结构说明的一般格式如下：

结构名 STRUCT

数据定义语句序列

结构名 ENDS

课程的结构COURSE的说明：

```
COURSE  STRUCT
        CID             DD ?                        ; 课程编号
        CTITLE          DB  20 DUP（0）             ; 课程名
        CHOUR           DB  0                       ; 学时数
        CTEACHER    DB  'WANG ZHONGLING '  ; 主讲教师
        CTERM           DB  1，2                    ; 开课学期
COURSE  ENDS
```

结构说明应放在结构变量定义之前，不属于任何段。

(2) 结构变量定义

一般格式为：

变量名 结构名 <字段赋值表>

变量名：是当前定义的结构变量的名称，可以省略。

字段赋值表：用来给结构变量的各字段重新赋初值，其中各字段值的排列顺序及类型应与结构说明时的各字段相一致，中间用逗号分隔。

如果某个字段采用在结构说明时指定的初值，那么可简单地用逗号表示；

如果不打算对结构变量重新赋值，则可省去字段赋值表，但仍必须保留一对尖括号。

几种正确的定义形式


```
C1  COURSE  < >     ; 5个字段均用结构说明时给的初值
C2  COURSE  <2102，‘SHU XUE’，60，‘LI MING’，>
                    ; 仅CTERM字段未重新赋值
COURSE  <2103，'YU WEN'，80，，>
     ; CTEACHER和CTERM字段未重新赋值，省略了变量名
C4  COURSE  5 DUP(<2101，，40，，>)
                    ; 定义了5个相同的结构变量，
                    ; 对CID、CHOUR重新赋了值
COURSE  10 DUP(< >)  ; 定义了10个结构变量，即预留了
                    ; 相应的存储空间
```

在定义结构变量时不能对含有多个项目的字段重新赋值，若想修改其值，应通过指令语句完成。

（3）结构变量访问

a)“结构变量名.结构字段名”的形式。**例如：**

```
C2  COURSE  <2102，'SHU XUE'，60，'LI MING'，1,2>

MOV  EAX，C2.CID          ；将2102送到EAX寄存器中
MOV  AL，C2.CTITLE        ；CTITLE中的字符‘S’送到AL中
MOV  AH，C2.CTITLE+2      ；CTITLE中的字符‘U’送到AH中
MOV  C2.CTERM+1，3        ；将CTERM的第2个项目的值从“2”改
                          ；成“3”
```

EA = 结构首址（变量名的段内EA）+该字段在结构内的位移量（字段名在结构内的EA）

b)结构首址先存入某个寄存器

用“[寄存器名]”代替结构变量名。所选择的寄存器必须满足变址寻址的要求。例如：

```
MOV  BX，OFFSET C2
                              ；获得结构变量C2的结构首址→BX
MOV  AL，[BX]. COURSE .CHOUR
                              ；将C2中CHOUR字段的值→AL
MOV  AL，（COURSE PTR [EBX]）.CHOUR
                              ；将C2中CHOUR字段的值→AL
```

如果要访问省略了名字的结构变量或字段，那就需要通过其它途径获得该结构变量或字段的偏移地址。

6.4.2 WIN32程序的结构

基于窗口的应用程序结构可以简单地划分为四个部分：

- **主程序**：OS首先执行“主程序”，获得与本程序有关的基本信息后再调用“窗口主程序”，
- **窗口主程序**：创建指定窗口后，将该窗口收到的消息通过OS转发到“窗口消息处理程序”
- **窗口消息处理程序**：判断收到的消息种类，决定应该调用“用户处理程序”中的哪一个或几个函数完成相应的功能
- **用户处理程序**：完成用户实际需求的各种函数的集合

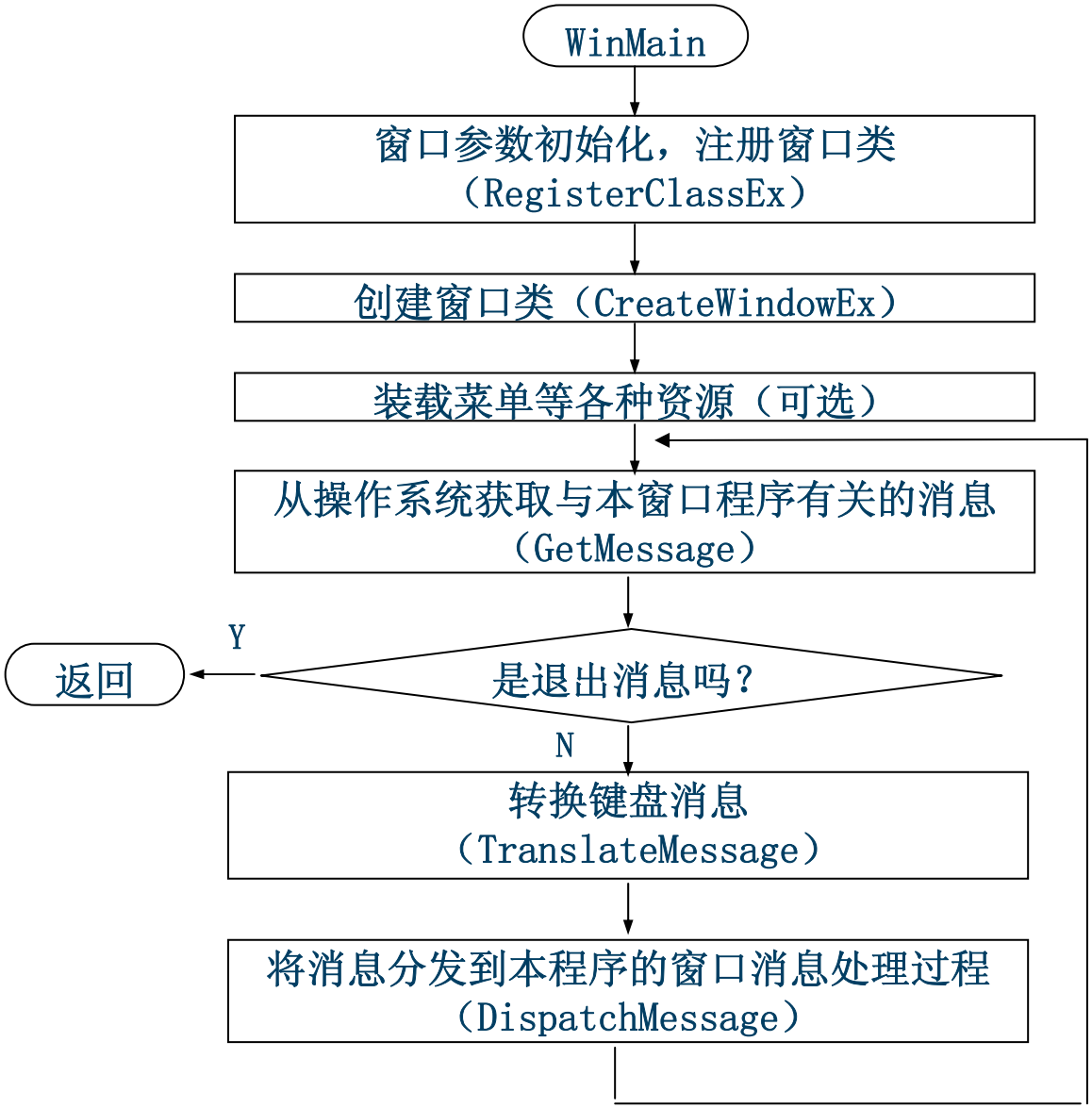
“主程序”流程图：


开始
得到本应用程序的句柄→hInstance
（GetModuleHandle）
当程序需要从命令行得到参数时，则需要获取命令行参数的地址→CommandLine（GetCommandLine）
调用窗口主程序
（WinMain）
结束（ExitProcess）

“窗口主程序”原型说明的形式


```
WinMain PROTO
        hInst：DWORD,      ；应用程序的实例句柄
    hPrevInst：DWORD,      ；前一个实例句柄（恒为NULL）
    lpCmdLine：DWORD,      ；命令行指针，指向以0结束的命令行字符串
    nCmdShow：DWORD        ；指出如何显示窗口
```

“窗口主程序”的流程图



“窗口主程序”中没有直接调用“窗口消息处理程序”

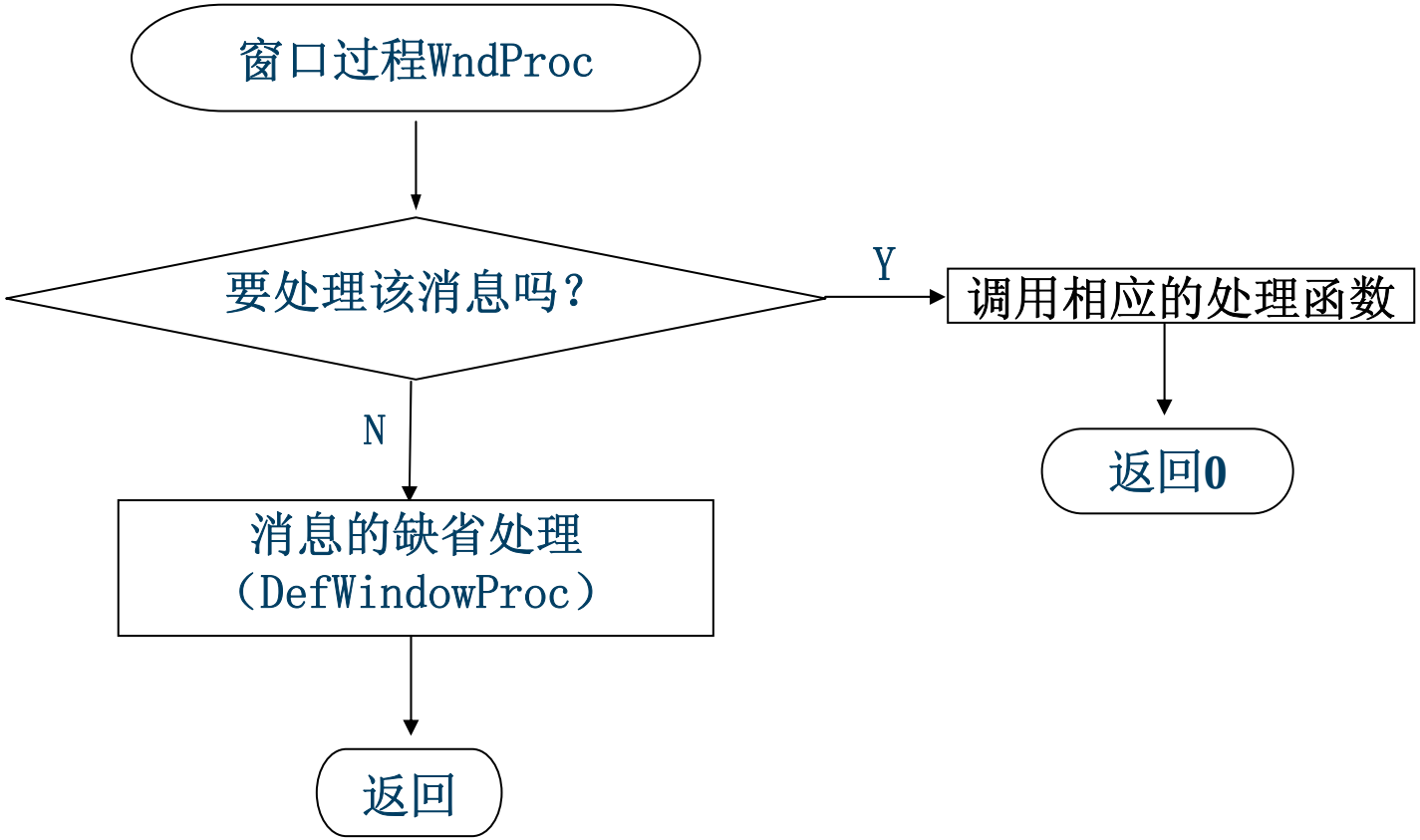
“窗口消息处理程序”

“窗口消息处理程序”（或称窗口过程，也是一个函数体形式）的主要功能是对接收到的消息进行判断，以便分类处理。其程序结构是一个典型的分支程序.

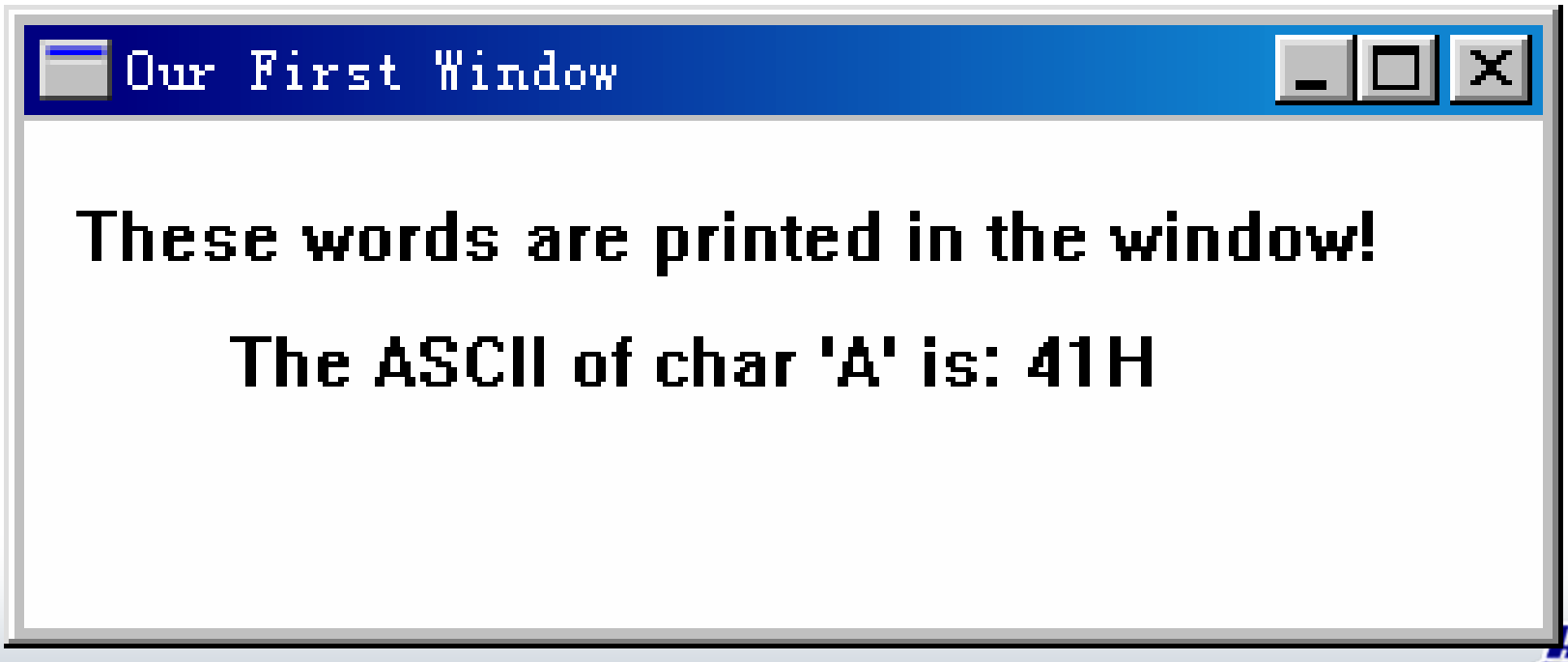
“窗口消息处理程序”的原型说明必须满足如下的形式：

```
WndProc PROTO hWin   :DWORD,      ; 窗口句柄
            uMsg    :DWORD,      ; 消息号，指明消息的种类，是分支判
                                 ; 断的主要依据
            wParam  :DWORD,      ; 该消息的附加信息。若是子消息
                                 ; 号，则是嵌套分支判断的依据
            lParam  :DWORD       ; 该消息的附加信息
```

流程图


窗口过程WndProc
要处理该消息吗？
Y
调用相应的处理函数
返回0
N
消息的缺省处理
（DefWindowProc）
返回

例：在屏幕上创建一个窗口，窗口的标题为："Our First Window"。当按下一个键时，在窗口中显示字符串"These words are printed in the window!"，并将该键的ASCII码以十六进制的形式显示出来。显示的格式为：



模块结构图

源程序

1. 头文件和数据段定义
2. 主程序
3. 窗口主程序（1）
4. 窗口主程序（2）
5. 窗口消息处理程序
6. 用户处理程序

头文件和数据段定义

```
.386
.MODEL FLAT,STDCALL                                  ; 存储模型说明
OPTION CASEMAP :NONE                                 ; 区分大小写说明
WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD            ; 原型说明
WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Convert   PROTO :BYTE, :DWORD
INCLUDE    windows.inc        ; 头文件说明
INCLUDE    gdi32.inc
INCLUDE    user32.inc
INCLUDE    kernel32.inc
INCLUDELIB gdi32.lib          ; 引入库说明
INCLUDELIB user32.lib
INCLUDELIB kernel32.lib
.DATA
ClassName   DB  "TryWinClass",0                      ; 窗口类名
AppName     DB  "Our First Window ",0                ; 窗口标题
hInstance   DD   0                                   ; 实例句柄
CommandLine DD   0
OurText     DB  "These words are printed in the window!" ; 窗口中待显示的字符串
num1        =   $-OurText
OurStr      DB  "The ASCII of char '?' is:",20H      ; 20H为空格符
szASCII     DB  "00H",20H,20H                        ; 后面加了2个空格
num2        =   $-OurStr
```

返回

主程序


```
.CODE

START:
    INVOKE  GetModuleHandle, NULL      ; 获得并保存本程序的句柄
    MOV     hInstance,EAX
    INVOKE  GetCommandLine
    MOV     CommandLine,EAX
    INVOKE  WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
                                       ; 调用窗口主程序
    INVOKE  ExitProcess,EAX            ; 退出本程序，返Windows
```

窗口主程序（1）


```
WinMain PROChInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD
LOCAL  wc   :WNDCLASSEX     ; 创建主窗口时所需要的信息由该结构说明
LOCAL  msg  :MSG            ; 消息结构变量用于存放获取的消息
LOCAL  hwnd :HWND           ; 存放窗口句柄，给WNDCLASSEX结构变量wc的各字段赋值
MOV    wc.cbSize,SIZEOF WNDCLASSEX                  ; WNDCLASSE结构类型的字节数
MOV    wc.style, CS_HREDRAW or CS_VREDRAW           ; 窗口风格(当窗口高度和宽度变
                                                    ;  化时则重画窗口)
MOV    wc.lpfnWndProc, OFFSET WndProc               ; 本窗口过程的入口地址（偏移地址）
MOV    wc.cbClsExtra,NULL                           ; 不使用自定义数据则不需OS预留空间，
MOV    wc.cbWndExtra,NULL                           ; 同上
PUSH   hInst                                        ; 本应用程序句柄→wc.hInstance
POP    wc.hInstance
MOV    wc.hbrBackground,COLOR_WINDOW+1              ; 窗口的背景颜色为白色
MOV    wc.lpszMenuName,NULL                         ; 窗口上不带菜单，置为NULL
MOV    wc.lpszClassName,OFFSET ClassName            ; 窗口类名"TryWinClass"的地址
INVOKE LoadIcon,NULL,IDI_APPLICATION                ; 装入系统默认的图标
MOV    wc.hIcon,EAX                                 ; 保存图标的句柄
 MOV   wc.hIconSm,0                                 ; 窗口不带小图标
INVOKE LoadCursor,NULL,IDC_ARROW                    ; 装入系统默认的光标
 MOV   wc.hCursor,EAX                               ; 保存光标的句柄
```

返回

窗口主程序（2）

```
    INVOKE   RegisterClassEx, ADDR wc                    ；注册窗口类
    INVOKE  CreateWindowEx,NULL,ADDR ClassName,  ；建立“TryWinClass”类窗口
            ADDR AppName,                        ；窗口标题"Our First Window"的地址
            WS_OVERLAPPEDWINDOW+ WS_VISIBLE,     ；创建可显示的窗口
            CW_USEDEFAULT,CW_USEDEFAULT,         ；窗口左上角坐标默认值
            CW_USEDEFAULT,CW_USEDEFAULT,         ；窗口宽度，高度默认值
            NULL,NULL,                           ；无父窗口，无菜单
            hInst,NULL                           ；本程序句柄，无参数传递给窗口
    MOV     hwnd,EAX                             ；保存窗口的句柄
StartLoop：                                  ；进入消息循环
    INVOKE GetMessage, ADDR msg,NULL,0,0     ；从Windows获取消息
    CMP     EAX，0                           ；如果（EAX）不为0 则要转换并分发消息
    JE      ExitLoop                         ；如果（EAX）为0 则转ExitLoop
    INVOKE TranslateMessage, ADDR msg        ；从键盘接受按键并转换为消息
    INVOKE DispatchMessage, ADDR msg         ；将消息分发到窗口的消息处理程序
   JMP    StartLoop                          ；再循环获取消息
ExitLoop：
   MOV     EAX,msg.wParam                    ；设置返回（退出）码
   RET
WinMain ENDP
```

返回

窗口消息处理程序


```
WndProc PROC hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
  LOCAL hdc:HDC                                  ; 存放设备上下文句柄
  .IF uMsg == WM_DESTROY                         ; 收到的是销毁窗口消息
     INVOKE PostQuitMessage,NULL                 ; 发退出消息
  .ELSEIF uMsg == WM_ CHAR                       ; 收到的是在窗口中按键的消息
     INVOKE GetDC,hWnd                           ; 根据窗口句柄确定设备句柄
     MOV    hdc,EAX                              ; 保存设备上下文句柄
     MOV    eax,wParam                           ; 将按键的ASCII码送到AL中
     MOV    szASCII-7,AL                         ; 用按键的ASCII码替换OurStr串中的?
     INVOKE Convert,AL,ADDR szASCII              ; 将(AL)转换成16进制形式的显示码→szASCII中
     INVOKE TextOut ,hdc,10,15,ADDR OurText,num1 ; 从窗口坐标（10，15）开始显示
                                                 ; OurText指向的串
     INVOKE TextOut ,hdc,40,40,ADDR OurStr,num2  ; 从窗口坐标（40，40）开始显示
                                                 ; OurStr指向的串
  .ELSE
    INVOKE DefWindowProc,hWnd,uMsg,wParam,lParam ; 不是本程序要处理的消息作
                                                 ; 其它缺省处理
    RET
  .ENDIF
  XOR    EAX,EAX
  RET
WndProc ENDP
```

返回

用户处理程序

```
; 子程序名：Convert
; 功能：将8位数按16进制形式转换成ASCII码的子程序
; 原型：Convert PROTO  bChar:BYTE,    ；待转换的8位二进制数
;                      lpStr:DWORD    ；转换结果的存放地址
; 受影响的寄存器：AL，ESI
Convert PROC bChar:BYTE, lpStr:DWORD
        MOV  AL,bChar
        MOV  ESI,lpStr
        SHR  AL,4
        CMP  AL,10
        JB   L1
        ADD  AL,7
L1:     ADD  AL,30H
        MOV  [ESI],AL
        MOV  AL,bChar
        AND  AL,0FH
        CMP  AL,10
        JB   L2
        ADD  AL,7
L2:     ADD  AL,30H
        MOV  [ESI+1],AL
        RET
Convert ENDP
      END START
```

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。

本章作业

- 1、6.1
- 2、6.2
- 3、6.9