80X86汇编语言程序设计

80X86 Assembly Language Programming

第6章 输入/输出和WIN32编程

金良海 教授

计算机学院 医学图像信息研究中心

Email: lianghaijin@hust.edu.cn

第6章 输入/输出和WIN32编程

本章内容

- 输入输出指令
- 主机与外部设备之间传送数据的方式
- 中断的概念及中断处理程序设计
- WIN32程序设计基本方法与技术

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

本章学习的难点

(1) 中断矢量表的作用、存取方法；

(2) 中断处理程序的安装、驻留、调试；

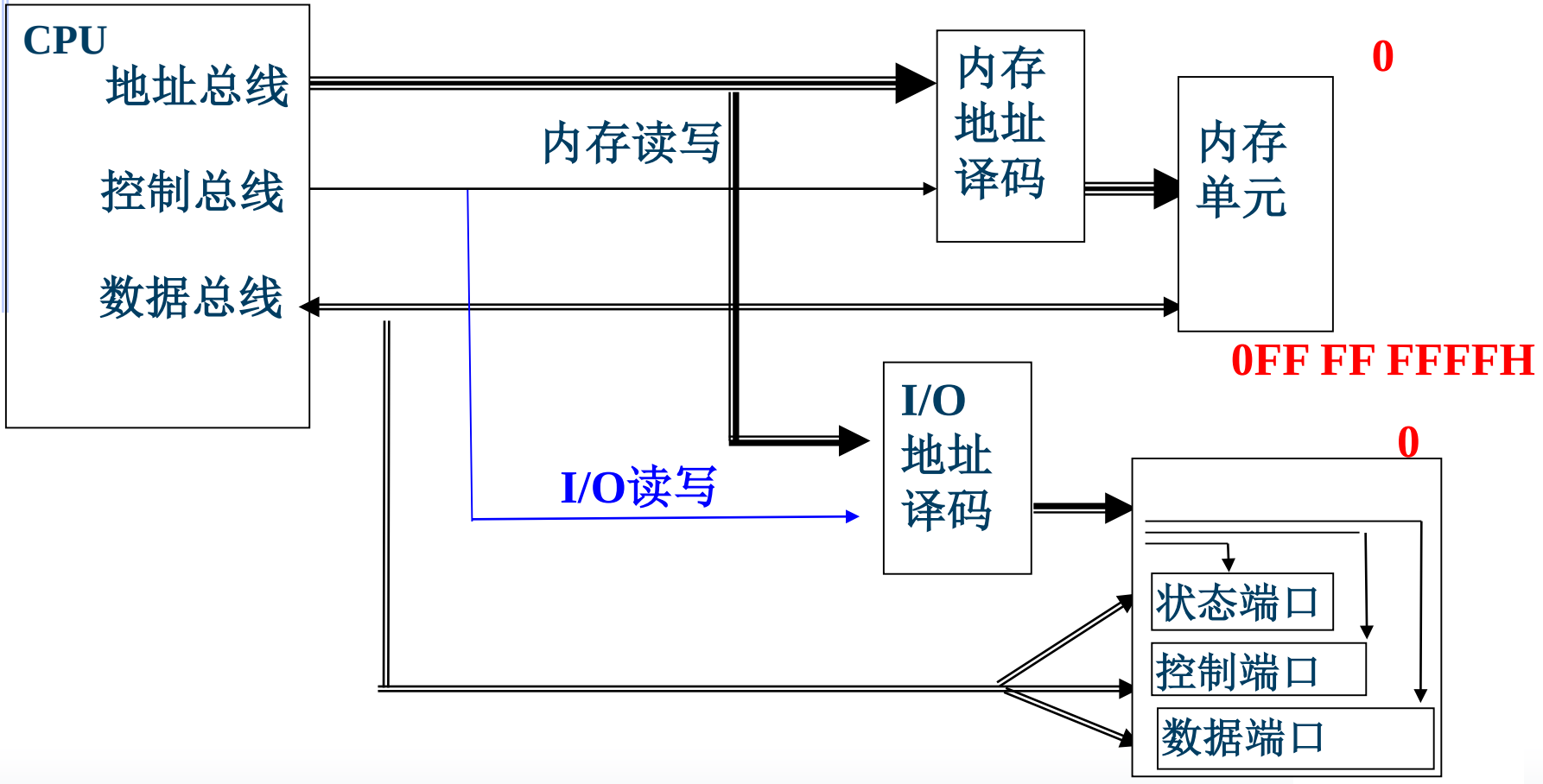
(3) 基于窗口的WIN32程序执行流程、消息驱动机制。

6.1 输入/输出指令和数据的传送方式

在80X86系统中，可以直接使用CPU的指令去访问的存贮空间有二种：主内存和I/O空间。

- 主内存一般位于计算机主板上，可以使用前面学过的各种指令去访问，如MOV、ADD指令等。
- I/O空间是外部设备与CPU通信的接口，一般位于外部设备上，它只能用专用的指令(IN/OUT)去访问。I/O空间的地址范围为0000H~0FFFFH，共64K。

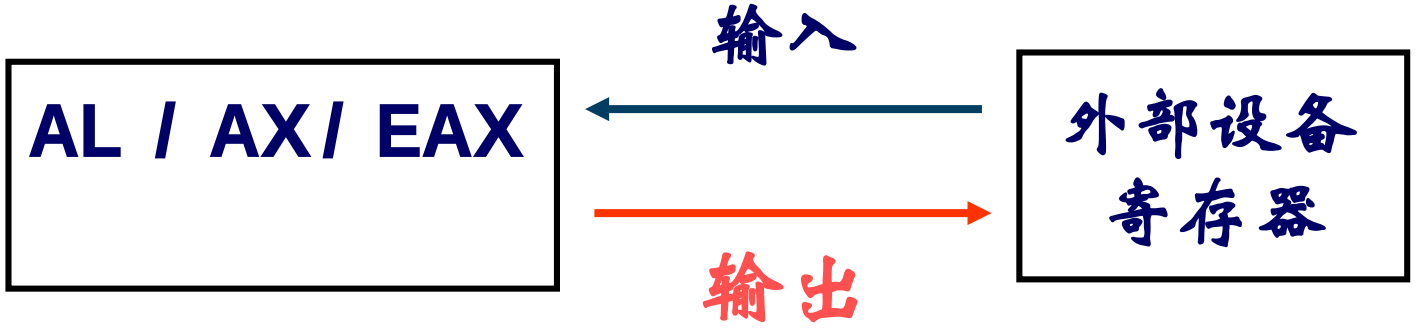
6.1 输入/输出指令和数据的传送方式



6.1.1 输入/输出指令

华中科技大学

输入/输出指令：IN / OUT



(6.1.1) 1.输入指令(1)

格式：IN OPD，OPS

功能：（OPS）→累加器OPD

说明：

①当端口地址<=255时，OPS = 立即数

或者DX表示待访问的地址。

当端口地址 >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

(6.1.1) 1.输入指令(2)

例： IN AL，60H

执行前：（60H）=11H，（AL）=0E3H

执行后：（AL）=11H，（60H）不变

```
MOV  DX,  60H
IN    AL,  DX
```

说明：

60H是8255可编程外围接口芯片的一个输入端口，存放键盘当前按键的键码。

(6.1.1) 1.输入指令(3)

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

即完成：（[DX]）→AX的功能。
（200H）→AL、（201H）→AH。

(6.1.1) 1.输出指令(1)

格式：OUT OPD，OPS

功能：累加器（OPS）→OPD

说明：

① OPD为立即数或者DX

端口地址大于255时，只能用DX

② OPS只能是累加器AL、AX或EAX。

即：OUT OPD， AL/AX/EAX

(6.1.1) 1.输出指令(2)

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

说明：该指令完成（EAX）→[80]的功能。

（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。

(6.1.1) 3.串输入指令

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]

DF=0时，DI/EDI增1/2/4

DF=1时，DI/EDI减1/2/4

说　明：此指令可以与前缀rep一起使用，其中要读写的个数存放在CX中.

(6.1.1) 3.串输出指令

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]

DF=0时，SI/ESI增1/2/4

DF=1时，SI/ESI减1/2/4

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。

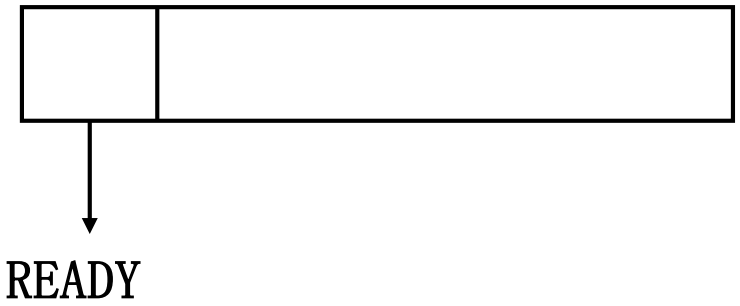
6.1.2 CPU与外设的数据传送方式(1)

计算机CPU与外部设备进行数据交换的方式：

- 无条件传送方式：不管外设的状态，直接进行数据传送。
- 查询传送方式：CPU先查询外设的状态，然后再进行数据的传送操作。
- 直接存贮器传送方式(DMA)：CPU停止工作，把控制权交给外设，进行快速、大规模的数据传送服务。磁盘与CPU的数据交换就是这一种。
- 中断传送方式：CPU运行自己的任务，当外设需要与CPU进行数据交换时，就向CPU发送一个中断信号，CPU检测到这个这个中断信号后，根据它当前的状态，决定是否进行响应。

6.1.2 CPU与外设的数据传送方式(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。

```
INPROG：IN    AL，STATUS_PORT
        TEST  AL，80H
        JZ    INPROG
        IN    AL，DATA_PORT
```

6.2 中断与异常

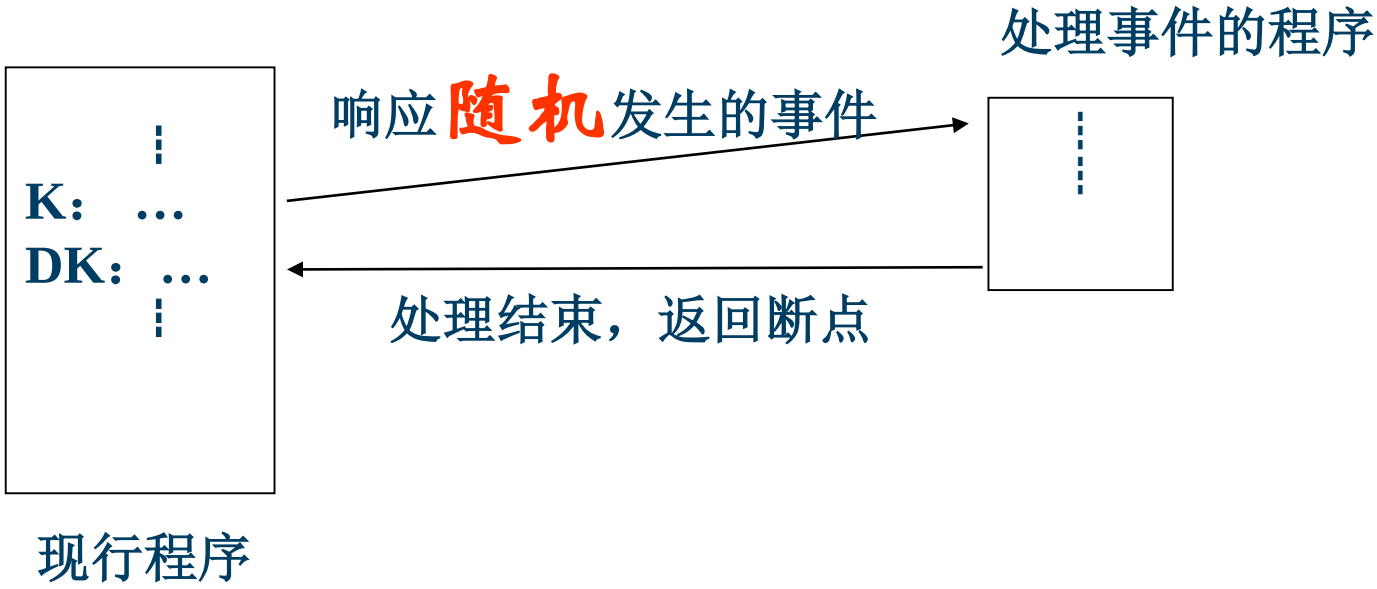
- 中断：CPU在执行正常的作业的时候，突然出现一个外部事件，CPU暂停当前的作业，转而去处理外部事件（执行外部事件处理程序），当外部事件处理完后，CPU又返回到原来的位置，继续执行原来的作业。
- 80X86能管理256种不同的中断和异常。

6.2.1 中断的概念(1)

- **中断源**：引起中断的事件（或外部设备）。
- **中断处理程序**（或中断服务程序）：对中断事件进行处理的程序。
- 中断处理程序存放在什么地方？

6.2.1 中断的概念(2)

中断处理过程：



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、中断来时的响应.

中断处理过程与子程序调用CALL的区别？

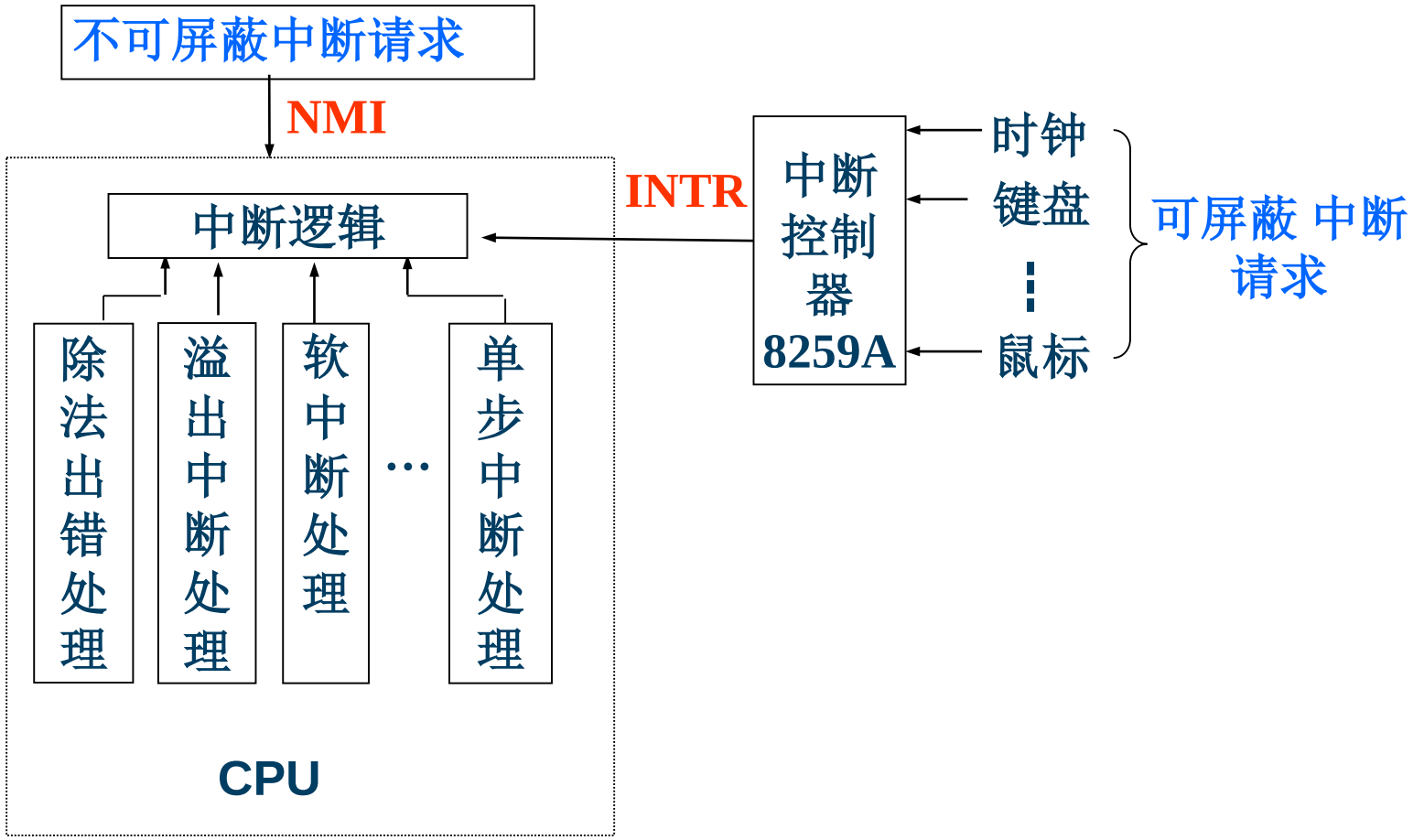
6.2.1 中断的概念(3)

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）
    - 关中断状态（CLI，IF=0）
- 内部中断（异常，与CPU的状态和当前执行的指令有关）
  - CPU检测：
    - 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。

6.2.1 中断的概念(4)


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

6.2.1 中断的概念(5)

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

6.2.1 中断的概念(6)

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

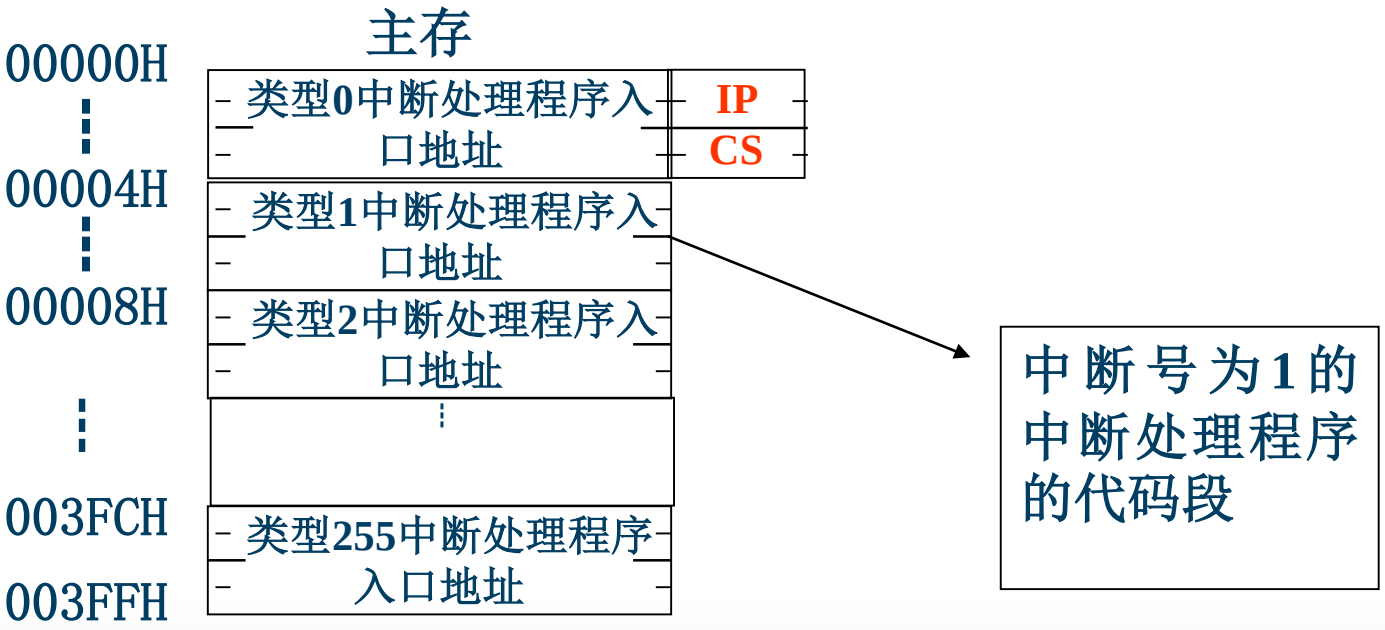
6.2.2 中断矢量表(1)

- 各种中断处理程序存放在内存的什么地方？
- 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）。
- 在实模式下，中断矢量表的大小为1k字节（每个中断服务程序的地址占4个字节，共256种中断）。其物理起始地址是内存的绝对地址00000H。因此，中断矢量表的内存空间为：00000H～003FFH。

6.2.2 中断矢量表(2)

➢实方式下的中断矢量表

大小为1KB，起始位置固定地从物理地址0开始



6.2.2 中断矢量表(3)

实方式下，中断处理程序的入口地址读取和查看：

例：读出 2 号中断处理程序的入口地址。

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

在TD中查看中断矢量表的方法

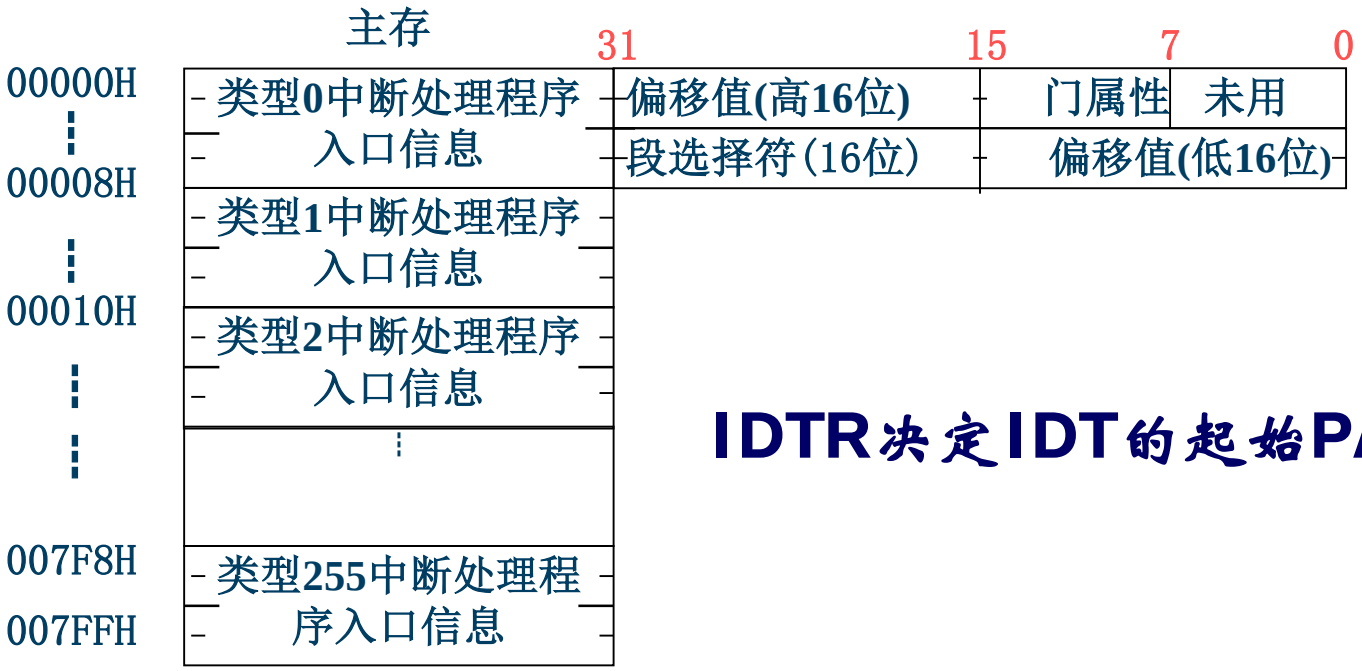
6.2.2 中断矢量表(4)

➢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。

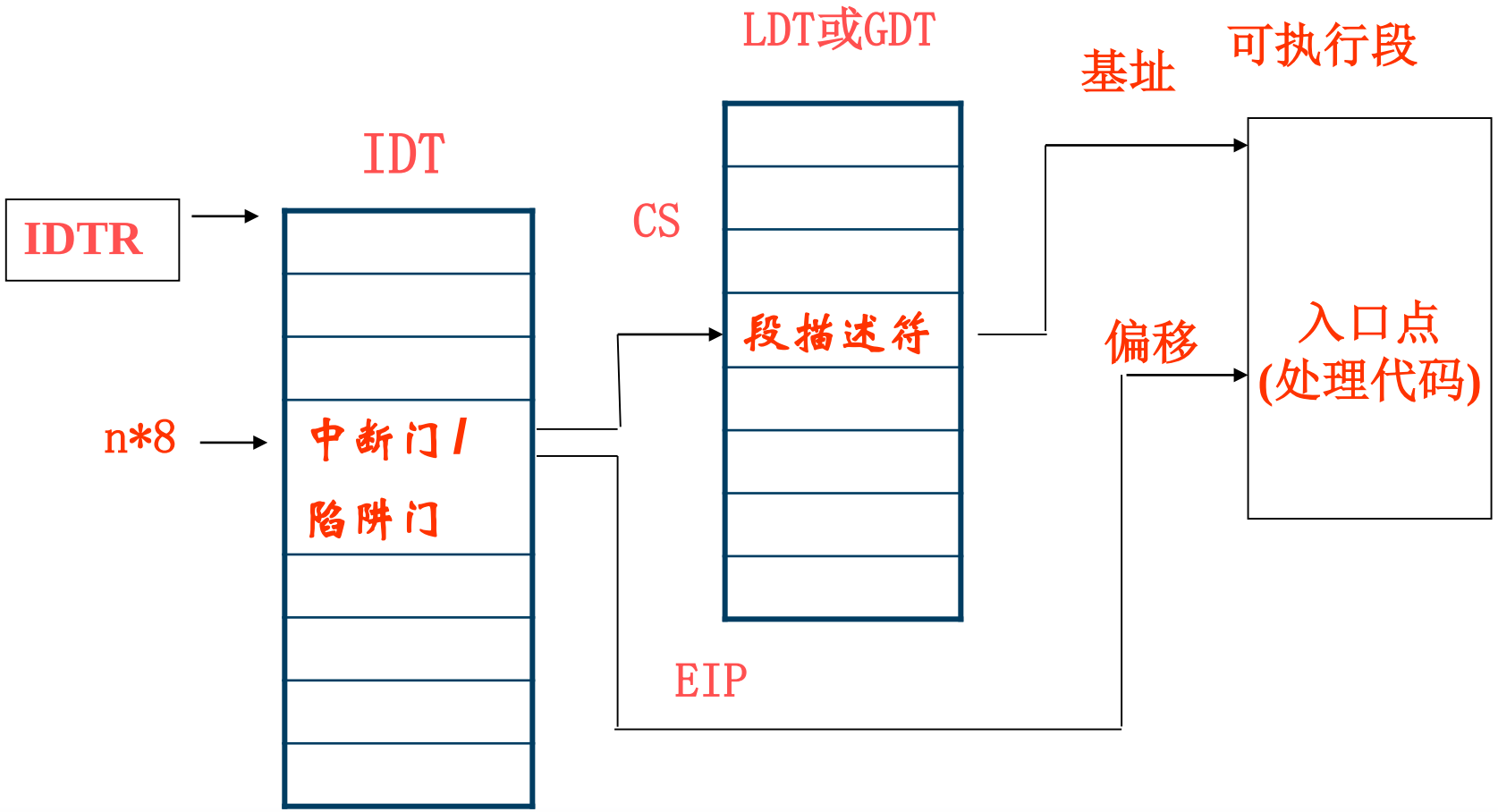
6.2.2 中断矢量表(5)

保护方式下的中断描述符表



IDTR决定IDT的起始PA

6.2.2 中断矢量表(6)



从中断号到中断处理程序的转换过程

6.2.3 软中断及有关的中断指令(1)

二个与中断相关的标志位

标志寄存器FLAGS/EFLAGS中有2个与中断相关的标志位：IF (仅用于可屏蔽中断)、TF

IF 中断允许：IF=1 允许CPU响应中断

IF=0 关中断 (CPU不能响应中断)

STI　　CLI

TF 跟踪：TF=1 CPU处于单步工作状态,每执行一条指令, CPU都会产生一个1号中断

6.2.3 软中断及有关的中断指令(2)

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

格式：INT n

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

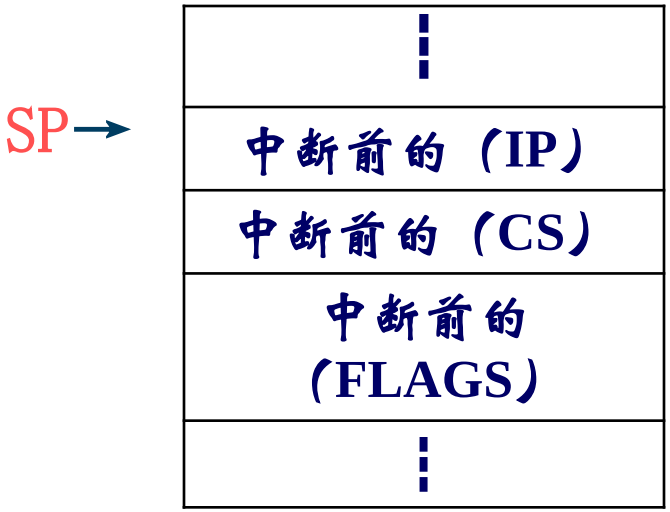
6.2.3 软中断及有关的中断指令(3)

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

6.2.3 软中断及有关的中断指令(4)

例：执行“INT 21H”时堆栈中信息的布局情况



中断前的（IP）= 断点（IP） = 返回地址值

6.2.3 软中断及有关的中断指令(5)

2. 中断返回指令

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

恢复断点地址

③ 实方式：↑(SP)→FLAGS

32位段：↑(ESP)→EFLAGS

恢复标志寄存器的内容

6.2.3 软中断及有关的中断指令(6)

```
…
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），之后再开中断，即 STI。

6.2.3 软中断及有关的中断指令(7)

★硬件中断的响应过程

(1) CPU在每条指令执行完成后，采样中断信号NMI、INTR

(2) 若中断信号无效，则执行下一条指令；
否则（NMI、INTR有效且对于INTR信号IF=1），执行如下操作：

- 关中断（IF=0）；
- 选取优先级最高的中断源n（n为其中断号）并将该中断源的中断信号置为无效；
- 执行指令 INT n 的操作；

6.2.4 中断处理程序的设计(1)

- 新增一个中断处理程序
- 修改已有的中断处理程序以扩充其功能。

6.2.4 中断处理程序的设计(2)

1.新增一个中断处理程序的步骤

① 编制中断处理程序。

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。

6.2.4 中断处理程序的设计(3)

1.新增一个中断处理程序的步骤

Q: 如何得到新中断处理程序的入口地址?

方法一: SEG 子程序的名称
OFFSET 子程序的名称

方法二: INTP_ADDRESS DD 子程序的名称

方法三: 当子程序与主程序在同一个段时,直接使用CS

6.2.4 中断处理程序的设计(4)

1.新增一个中断处理程序的步骤

Q:如何得到新中断处理程序的入口地址?

Q:如何将该入口地址写入中断矢量表?

直接写中断矢量表的相关位置；
DOS系统功能调用 P329（其他功能调用）

Q:如何调用新的中断处理程序?

6.2.4 中断处理程序的设计(5)

2. 修改（接管）已有中断处理程序以扩充其功能

- 编制程序段(根据扩充功能的要求，应注意调用原来的中断处理程序)
- 将新编制的程序段装入内存;
- 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。

Q: 如何调用老的中断处理程序呢？
直接使用 INT * ，行不行呢？**

6.2.4 中断处理程序的设计(6)

2、修改已有中断处理程序以扩充其功能

要求：

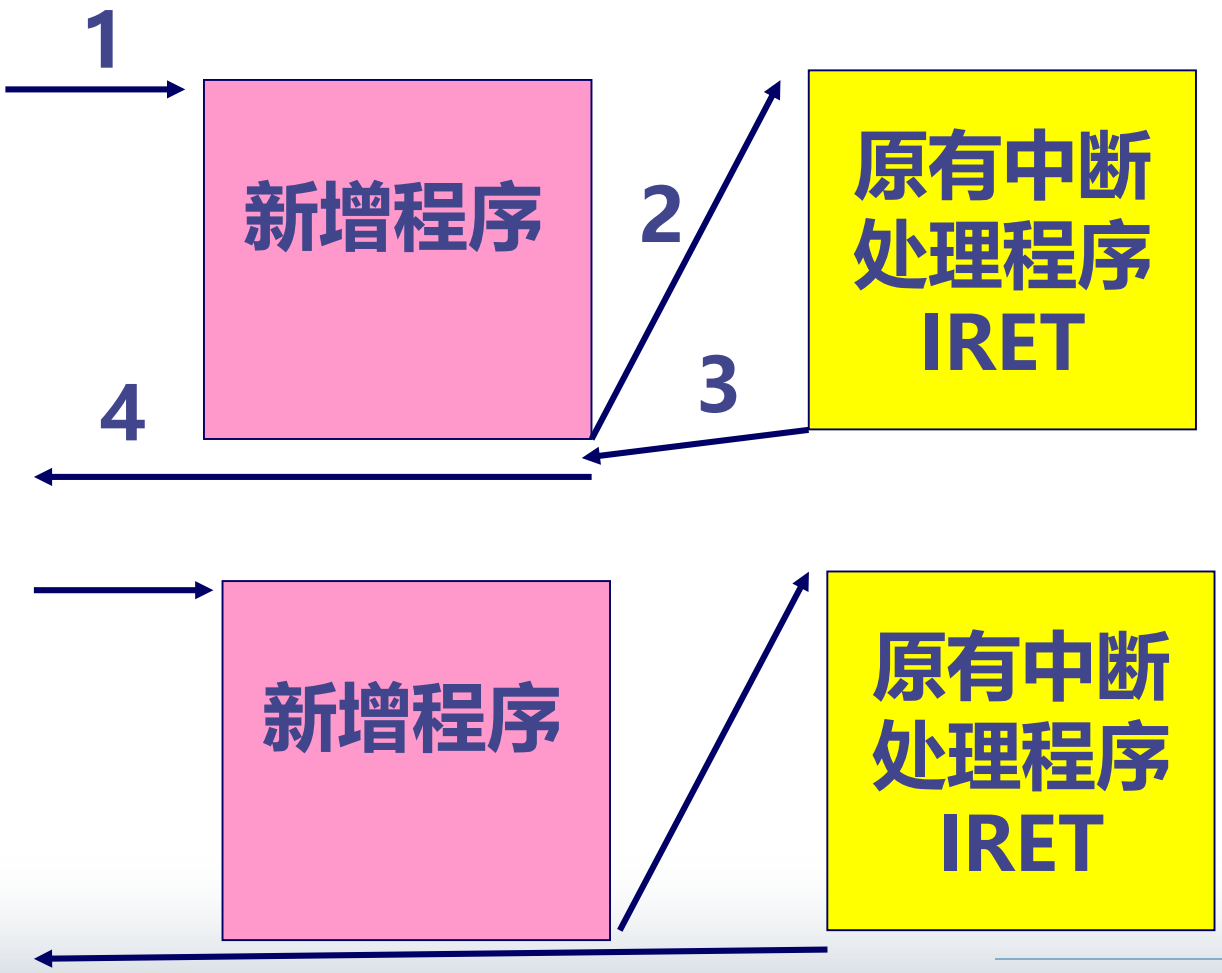
- 扩充后的程序 调用原来的中断处理程序
  即保留原有程序的功能
- 不能改原有的中断处理程序

新增程序

原有中断
处理程序
IRET

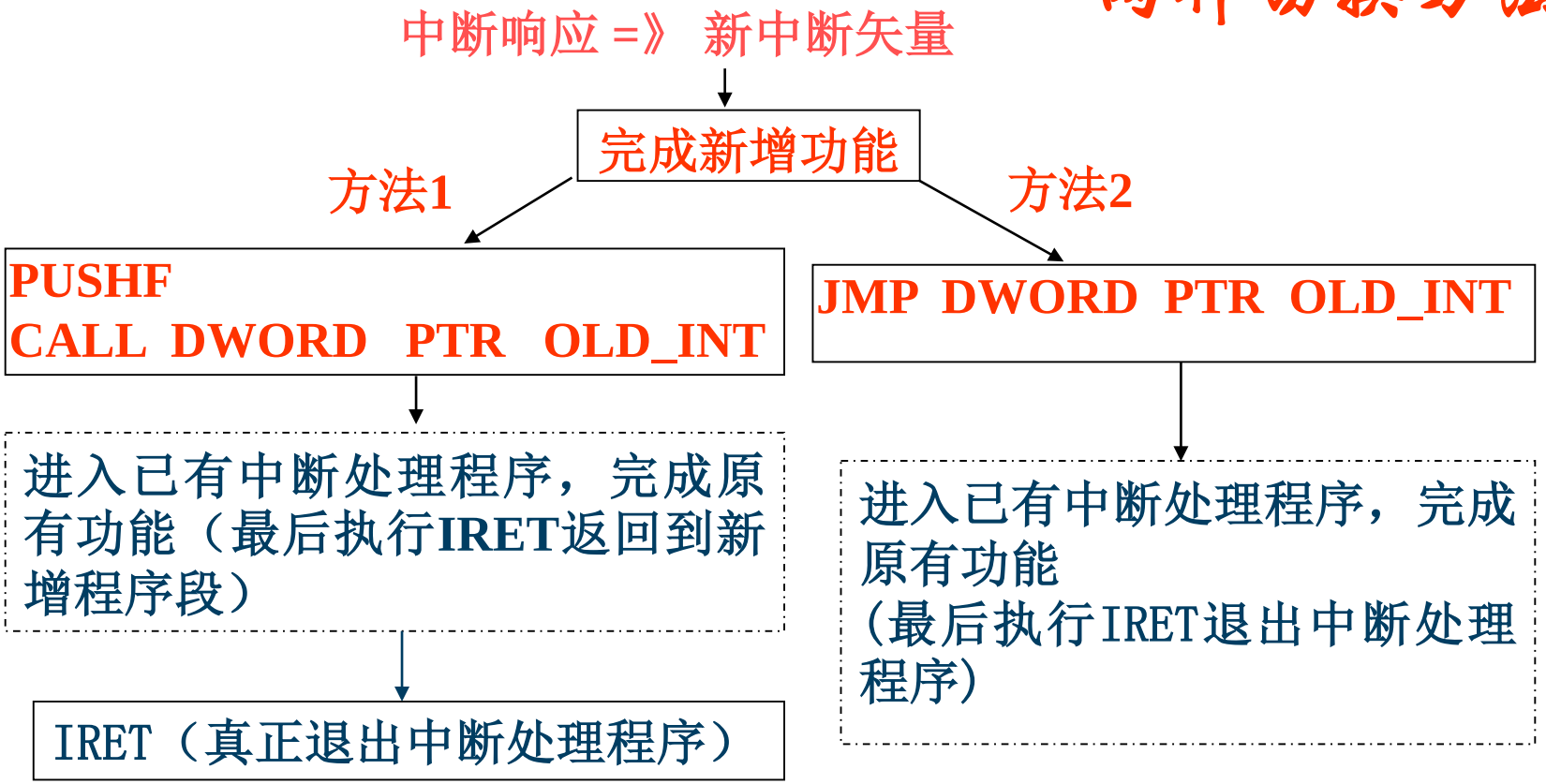
6.2.4 中断处理程序的设计(7)

2、修改已有中断处理程序以扩充其功能



6.2.4 中断处理程序的设计(8)

两种切换方法



6.2.4 中断处理程序的设计(9)

例: 编制时钟显示程序.

要求每隔1秒在屏幕的右上角显示时间。

（扩充原中断的功能）

C6_225_2.asm

在该程序运行结束后，时间显示仍然继续。

在运行其它程序时，还看得到显示的时间。

6.2.4 中断处理程序的设计(10)

程序涉及的知识要点分析:

(1)如何知道是否到达1秒？用什么中断合适?
(2)如何取当前时间？
(3)如何在指定位置显示时间？
(4)如何在显示时间后（改变了光标的位置），
　不影响其他程序的运行？
(5)如何在退出程序后，仍能显示时间？

6.2.4 中断处理程序的设计(11)

程序涉及的知识要点分析:

(1)每隔1秒

定时中断，时钟中断(P216,表6.1)

<PC中断大全 BIOS,DOS及第三方调用的程序员参考资料>

<PC中断调用大全>

由8254系统定时器的0通道每秒产生18.2次，该中断用于时钟更新。

6.2.4 中断处理程序的设计(12)

程序涉及的知识要点分析:

(1)每隔1秒

引入一个变量，记录进入中断处理程序的次数。
当达到18次时，取时间，然后显示时间。

(2)在屏幕的右上角显示时间

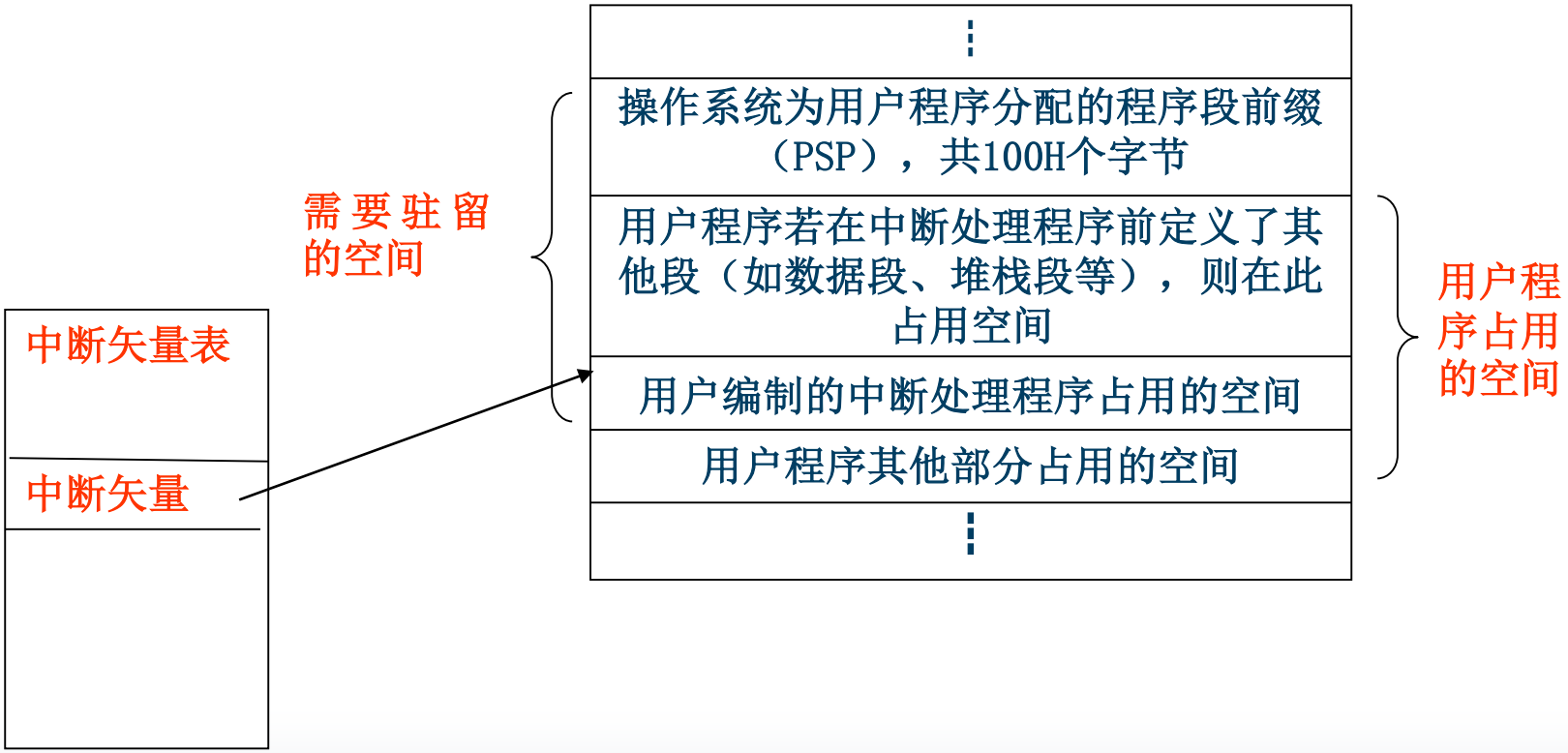
常用BIOS子程序,显示器驱动程序 P330

6.2.4 中断处理程序的设计(13)

(3) 程序驻留

一个程序所占的主存储空间，在该程序运行结束后，不被回收。

6.2.4 中断处理程序的设计(14)



6.2.4 中断处理程序的设计(15)

例1：编写一个中断处理程序，使得键盘的字母A变成字母B（按A键得到的是字母B，说明B的扫描码是30H）。

说明：所有的程序获取按键，最后都是通过执行16H号软中断的0号和10号功能调用来实现的。

资料：16H中断的0号和10号功能

功能描述：从键盘读入字符

入口参数：AH＝00H——读键盘

＝10H——读扩展键盘

出口参数：AH＝键盘的扫描码

AL＝字符的ASCII码

6.2.4 中断处理程序的设计(16)

```
.386
STACK   SEGMENT  USE16  STACK
        DB  100  DUP (0)
STACK   ENDS
CODE    SEGMENT USE16
        ASSUME CS:CODE,
                    SS:STACK
OLD_INT  DW    0, 0
NEW16H  PROC  FAR
        CMP    AH, 00H
        JZ      L1
        CMP    AH, 10H
        JZ      L1
        JMP    DWORD PTR CS:OLD_INT
L1:     PUSHF
    CALL  DWORD PTR  CS:OLD_INT
        CMP    AL, 'A'
        JNE     L2
        MOV    AL, 'B'
        MOV    AH, 30H   ;B的扫描码
        JMP     L3
L2:     CMP    AL, 'a'
        JNE     L3
        MOV    AL, 'b'
        MOV    AH, 30H   ;b的扫描码
L3:     IRET
NEW16H ENDP
```

6.2.4 中断处理程序的设计(17)

```
;;主程序开始
BEGIN:   PUSH  CS
         POP   DS
         MOV   AX, 3516H
         INT   21H
         MOV   CS:OLD_INT, BX
         MOV   CS:OLD_INT+2, ES
         ;;
;; MOV   AX, 0
;; MOV   ES, AX
;; MOV   AX, WORD PTR ES:[4*16H]
;; MOV   CS:OLD_INT, AX
;; MOV   AX, WORD PTR ES:[4*16H+2]
;; MOV   CS:OLD_INT+2, AX
```

```
;; MOV   AX,  0
;; MOV   ES,  AX
;; MOV   WORD PTR ES:[4*16H],
;;           OFFSET NEW16H
;; MOV WORD PTR ES:[4*16H+2], CS
         ;;
         MOV   DX, OFFSET NEW16H
         MOV   AX, 2516H
         INT   21H            ;DS:DX
         ;;
         ;;将新的中断处理程序驻留内存
         MOV   DX, 100H
         MOV   AH, 31H
         INT   21H
CODE     ENDS
         END   BEGIN
```

6.2.4 中断处理程序的设计(18)

例2：编写一个中断处理程序，功能如下：从键盘输入一个字符串，然后扩充16H键盘软中断，使得字符串中的字符键失效。例如，从键盘输入“Abc”后， Abc 这3个键就失效了。

说明：所有的程序获取按键，最后都是通过执行16H号软中断的0号和10号功能调用来实现的。

资料：16H中断的0号和10号功能
功能描述：从键盘读入字符
入口参数：AH＝00H——读键盘
　　＝10H——读扩展键盘
出口参数：AH＝键盘的扫描码
　　AL＝字符的ASCII码

6.2.4 中断处理程序的设计(19)

```
.386
STACK   SEGMENT  USE16  STACK
        DB  100  DUP (0)
STACK   ENDS
CODE    SEGMENT USE16
        ASSUME CS:CODE,
                    DS:CODE,SS:STACK
OLD_INTDW   0, 0
KEY_BUF DB   60, 61 DUP(0)
MSG   DB   'Strike keys which will be disabled: $'
NEW16H  PROC  FAR
        CMP   AH, 00H
        JZ      L0
        CMP   AH, 10H
        JZ      L0
        JMP   DWORD PTR CS:OLD_INT
L0:     PUSH  BP
        PUSH  BX
        MOV   BP, AX
L1:     MOV   AX, BP
        PUSHF
    CALL  DWORD PTR  CS:OLD_INT
        MOV   BX, 0
L2:     CMP   BL, CS:KEY_BUF+1
        JAE    L9
        CMP   AL, CS:KEY_BUF+2[BX]
        JE      L1
        INC    BX
        JMP   L2
        ;;
L9:     POP   BX
        POP   BP
        IRET
NEW16H ENDP
```

6.2.4 中断处理程序的设计(20)

```
;;主程序开始
BEGIN:  PUSH  CS
        POP   DS
        ;;
        MOV   DX, OFFSET MSG
        MOV   AH, 09H
        INT     21H
        ;;
        MOV   DX, OFFSET KEY_BUF
        MOV   AH, 0AH
        INT     21H
        ;;
        MOV   AX, 3516H
        INT     21H
        MOV   CS:OLD_INT, BX
        MOV   CS:OLD_INT+2, ES
;;      MOV   AX,  0
;;      MOV   ES,  AX
;;      MOV   WORD PTR ES:[4*16H],
;;            OFFSET NEW16H
;;      MOV WORD PTR ES:[4*16H+2], CS
        ;;
        MOV   DX, OFFSET NEW16H
        MOV   AX, 2516H
        INT     21H             ;DS:DX
        ;;
        ;;将新的中断处理程序驻留内存
        MOV   DX, 100H
        MOV   AH, 31H
        INT     21H
CODE    ENDS
        END   BEGIN
```

6.4 WIN32编程

華中科技大學

WIN32程序

基于Windows运行环境的32位段程序

6.4.1 WIN32编程基础 (1)

WIN32系统API函数库和头文件

WINDOWS操作系统提供了大量的用户可以使用的系统函数API，汇编语言可以直接调用API，API以动态库的形式提供给程序员：

.INC 头文件，函数原型说明
.LIB 引入库，API函数在DLL库中的位置信息
.DLL API函数的实现代码

6.4.1 WIN32编程基础 (2)

- 常用的库和头文件

KERNEL32 (.LIB, .INC) 内存和进程管理

USER32 (.LIB, .INC) 用户界面

GDI32 (.LIB, .INC) 图形操作

WINDOWS.INC 常量和结构的定义(无函数原型)

6.4.1 WIN32编程基础 (3)

应用API函数

(1) 在.ASM源文件中包含头文件

如，INCLUDE WINDOWS.INC

INCLUDE D:\SDK\USER32.INC

(2) 将相应的库连接到.EXE目标文件中

方法1：LINK时将指定相应的.LIB文件；

方法2：在.ASM源文件中将相应的.LIB文件包含进去，如

INCLUDELIB KERNEL32.LIB

INCLUDELIB D:\SDK\USER32.LIB

6.4.1 WIN32编程基础 (4)

- **段的简化定义**

  存储模型说明伪指令 .MODEL

  段定义伪指令 .CODE .DATA .STACK

- **原型说明与函数调用**

  函数原型说明伪指令 PROTO

  函数调用伪指令 INVOKE

  函数(过程)定义指令 PROC

- **结构**

  结构定义伪指令 STRUCT

6.4.1 WIN32编程基础 (5)

➢段的简化定义

```
.MODEL  存储模型        ;TINY, SMALL, FLAT等
        [,语言类型]     ;C, PASCAL, STDCALL
        [,系统类型]     ;省略
        [,堆栈选项]     ;NEAR/FARSTACK
```

在WIN32编程中，使用：

```
.MODEL  FLAT, STDCALL   ;SS=DS=ES
```

注：.MODEL伪指令必须放在所有段定义伪指令之前且只能出现一次。

6.4.1 WIN32编程基础 (6)

常用存储模型

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
FLAT	32位	NEAR	NEAR	代码和数据全部放在同一个4G空间内

6.4.1 WIN32编程基础 (7)

语言类型

STDCALL类型：

- 采用堆栈法传递参数；
- 参数进栈次序：最右边的参数最先入栈、最左边的最后入栈；
- 参数占用空间的清理：由被调用者在返回时清除相应的堆栈空间。

6.4.1 WIN32编程基础 (8)

.CODE [段名]

功能：

- 说明一个代码段的开始；
- 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；
- 若无段名，在TINY、SMALL和FLAT模式时，段名为“_TEXT”。

6.4.1 WIN32编程基础 (9)

.DATA 或 .DATA？

功能：一个数据段的开始，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。

6.4.1 WIN32编程基础 (10)

堆栈段定义：.STACK [堆栈字节数]

“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初值相同。

6.4.1 WIN32编程基础 (11)

原型说明与函数调用

函数原型说明伪指令　PROTO

函数调用伪指令　INVOKE

函数(过程)定义指令　PROC

6.4.1 WIN32编程基础 (12)

格式：

函数名 PROTO [函数类型,][语言类型,]
[[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：NEAR、FAR 等

语言类型：STDCALL 等

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

6.4.1 WIN32编程基础 (13)


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

uType指定了在消息框中要显示的按钮和图标：

=0，仅显示“确认”按钮（MB_OK=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=3）

6.4.1 WIN32编程基础 (14)

格式：INVOKE 函数名 [,参数]...

说明：INVOKE必须在PROTO语句或完整的PROC语句说明之后使用。

```
INVOKE  RADIX_S,SI,10,EAX
INVOKE  MessageBoxA , 0 , OFFSET dir ,
                  OFFSET hello, MB_OK
```

ADDR 与 OFFSET：ADDR可以处理局部变量而OFFSET不能

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

6.4.1 WIN32编程基础 (15)

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（其后为函数体）。

参数名：用来传递数据的，可以在程序中直接引用，不能省略。

USES寄存器表：需要在子程序中入栈保护的，由汇编程序自动完成。

6.4.1 WIN32编程基础 (16)

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。

```
LOCAL  array[20]:BYTE
   ；在堆栈中分配以array为首址的20个字节

LOCAL  tFlag:WORD
```

6.4.1 WIN32编程基础 (17)

其他说明(1)

(1) 在源程序中若要使用段名,则可通过预定义符来指定,如 @DATA和@CODE表示数据段和代码段的段名:

```
MOV AX, @DATA
MOV DS, AX
```

(2) 常数（只读）数据段定义伪指令 .CONST

(3) 程序开始伪指令.STARTUP
仅用于16位段（如SMALL模式）。表示程序启动时,DS、ES、SS已经被设置为同一个有效值，也不需要在程序最后的END语句中说明程序开始的位置（省掉END后面带的参数）。

6.4.1 WIN32编程基础 (18)

其他说明(2)

(4) 函数返回值

若函数具有返回值，则按如下约定：

1、2、4字节的返回值存放在AL、AX、EAX中

若需要多个返回值，怎么办？

6.4.1 WIN32编程基础 (19)


```
.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    “Hello", 0
. STACK 200
. CODE
 START:  PUSH 0                      ; START:
         PUSH OFFSET dir             ;  invoke MessageBoxA, 0,
         PUSH OFFSET hello           ;            OFFSET hello,
         PUSH 0                      ;            OFFSET dir,0
         CALL MessageBoxA            ;
         PUSH 0                      ; invoke ExitProcess, 0
         CALL ExitProcess            ;
END START                            ; END START
```

6.4.1 WIN32编程基础 (20)

结构

结构名 STRUCT
　　数据定义语句序列
结构名 ENDS

```
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 ZHONG ’ ;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。

6.4.1 WIN32编程基础 (21)

定义结构变量：

变量名 结构名 <字段赋值表 >

字段赋值表：

- 给结构变量的各字段重新赋初值;
- 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。

6.4.1 WIN32编程基础 (22)

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, CID、CHOUR重新赋了值

COURSE 10 DUP(<>)

定义结构变量时，不能对含有多个项目的字段(如CTERM)重新赋值。若想修改其值，则需通过访问结构成员的指令语句来完成。

6.4.1 WIN32编程基础 (23)

访问结构成员

方法1：“结构变量名.结构字段名”

```
C2  COURSE  <2102，'SHU XUE'，60，'LI MING'，>
MOV  EAX，C2.CID      ；将2102送到EAX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    ；修改CTERM的第2个项的值
```

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）

如何将上述4条语句翻译成汇编指令？

6.4.1 WIN32编程基础 (24)

访问结构成员

方法2：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  EBX，OFFSET C2
MOV  AL，[EBX].COURSE.CHOUR
MOV  AL，(COURSE PTR [EBX]).CHOUR
```

如何将上述条语句翻译成汇编指令？

6.4.1 WIN32编程基础 (25)

- WIN32汇编上机（MASM32软件包）

汇编：ML

连接：LINK

调试：TD32（机器码调试）、
VC调试软件（源码符号级调试）

其他：QEDITOR 一个简单的集成开发环境
NMAKE 工具软件
RC CVTRES .RC => .OBJ 转换
(见7.3.3)

6.4.1 WIN32编程基础 (26)

➢WIN32汇编上机（操作过程-1）

(1) 进入命令行状态

(2) 设置环境（如运行var.bat）

(3) 建立源文件（demo.asm）

(4) 编译：ml /Fl /c /coff demo.asm

(5) 连接：link /subsystem:windows demo.obj

(6) 调试：td32 demo.exe （机器码调试）

6.4.1 WIN32编程基础 (27)

➢WIN32汇编上机（操作过程-2）

说明：(1) ml /help 和 link /? 可显示相应的帮助

(2) 若有资源脚本文件（如menu.rc），
则需转换为.obj文件，再连接到一起：

```
rc  menu.rc                                  ;menu.res
cvtres  /machine:ix86  menu.res  ;menu.obj
link  /subsystem:windows demo.obj menu.obj
```

6.4.1 WIN32编程基础 (28)

➢WIN32汇编上机（操作过程-3）

设置环境变量

```
set  Masm32Dir=d:\Masm32
set  include=%Masm32Dir%\Include;%include%
set  lib=%Masm32Dir%\lib;%lib%
set  path=%Masm32Dir%\Bin;%Masm32Dir%;%PATH%
set  Masm32Dir=
```

6.4.2 WIN32程序的结构 (1)

WIN32应用程序结构

- 主程序
- 窗口主程序 WinMain
- 窗口消息处理程序
- 用户处理程序

6.4.2 WIN32程序的结构 (2)

(1) 主程序

6.4.2 WIN32程序的结构 (3)

句柄是一个代号。

用来唯一标识某一个程序或窗口等，一般是它们的地址。

应用程序的句柄、窗口的句柄、图标的句柄、文件的句柄等。

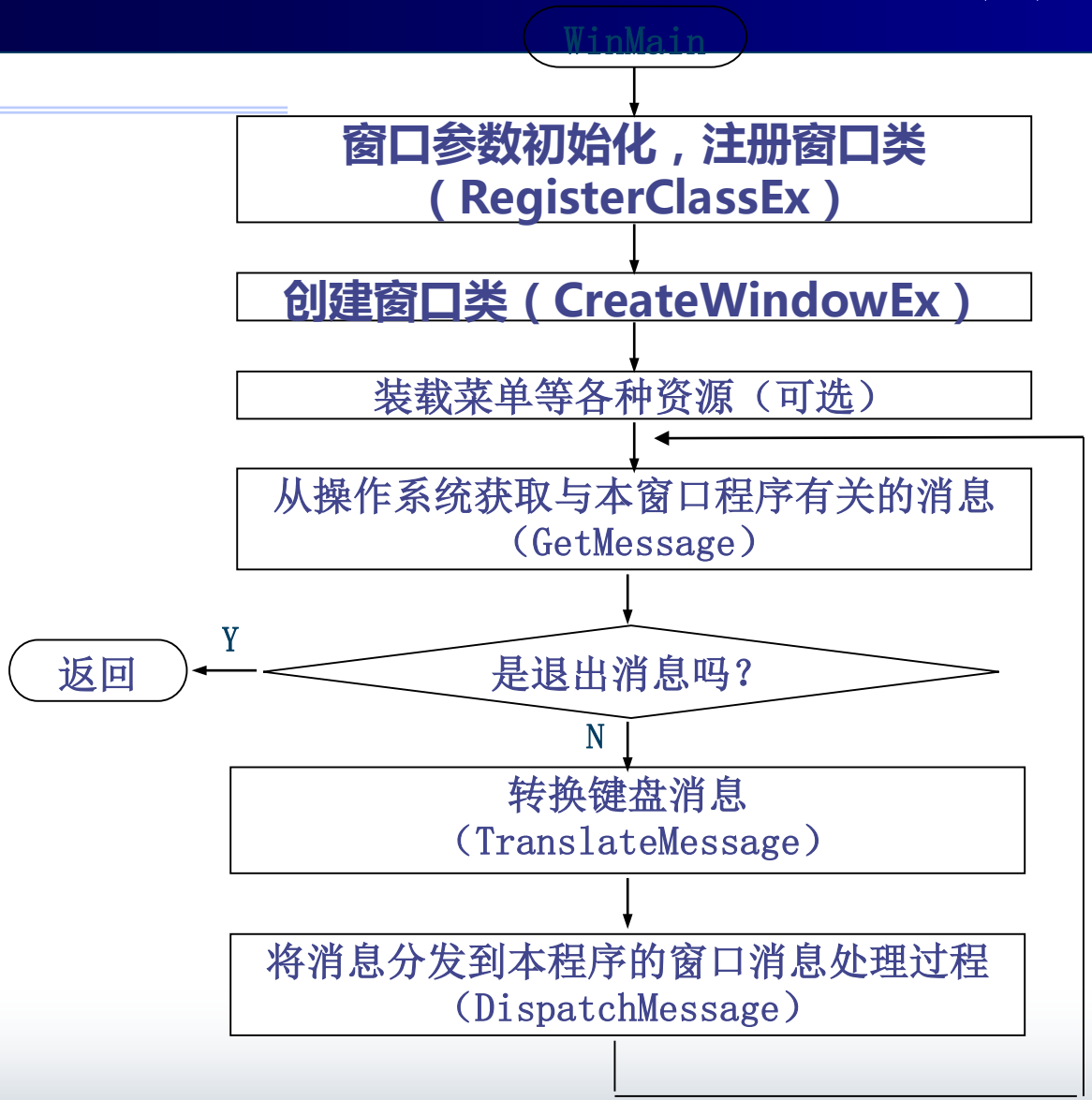
用户程序必须先通过Windows API来获得指定程序或窗口的句柄，然后才能通过该句柄对所代表的程序或窗口进行操作，如发送消息等。

6.4.2 WIN32程序的结构 (4)

(2) 窗口主程序

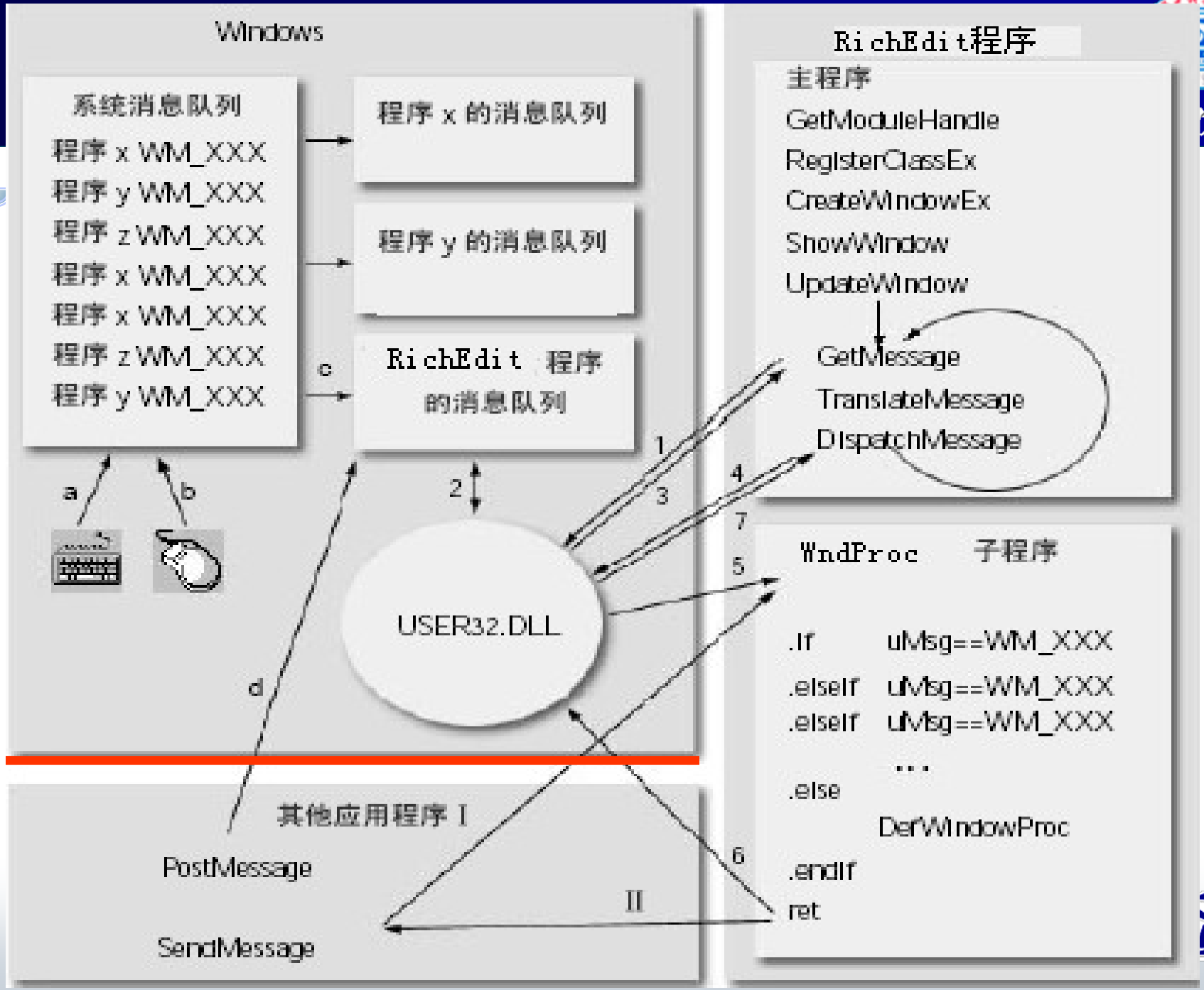
```
WinMain PROTO
        hInst :    DWORD ,  ;应用程序的实例句柄
        hPrevInst : DWORD ,  ;前一个实例句柄,NULL
        lpCmdLine : DWORD ,  ;命令行指针
        nCmdShow :  DWORD    ;指出如何显示窗口
```

6.4.2 WIN32程序的结构 (5)



“窗口主程序”中没有直接调用“窗口消息处理程序”

```
WinMain PROC hInst :DWORD, hPrevInst :DWORD,
            CmdLine :DWORD, CmdShow :DWORD
    LOCAL wc  :WNDCLASSEX
    LOCAL msg :MSG
    invoke RegisterClassEx, ADDR wc       ; 注册窗口
    invoke CreateWindowEx,NULL,           ; 创建窗口
                ADDR szClassName, ADDR AppName,
                WS_OVERLAPPEDWINDOW + WS_VISIBLE,
                CW_USEDEFAULT, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT,
                NULL,NULL,  hInst, NULL
StartLoop:
    invoke GetMessage, ADDR msg, NULL, 0, 0
    cmp    eax, 0
    je     ExitLoop
    invoke TranslateMessage, ADDR msg
    invoke DispatchMessage , ADDR msg
    jmp    StartLoop
ExitLoop:
    mov    eax, msg.wParam
    ret
WinMain endp
```


Windows
系统消息队列
程序 x WM_XXX
程序 y WM_XXX
程序 z WM_XXX
程序 x WM_XXX
程序 x WM_XXX
程序 z WM_XXX
程序 y WM_XXX
程序 x 的消息队列
程序 y 的消息队列
RichEdit 程序
的消息队列
a
b
c
d
2
USER32.DLL
PostMessage
SendMessage
其他应用程序 I
II
RichEdit程序
主程序
GetModuleHandle
RegisterClassEx
CreateWindowEx
ShowWindow
UpdateWindow
GetMessage
TranslateMessage
DispatchMessage
1
3
4
5
6
7
WndProc 子程序
.If uMsg==WM_XXX
.elseIf uMsg==WM_XXX
.elseIf uMsg==WM_XXX
...
.else
DefWindowProc
.endIf
ret

6.4.2 WIN32程序的结构 (6)

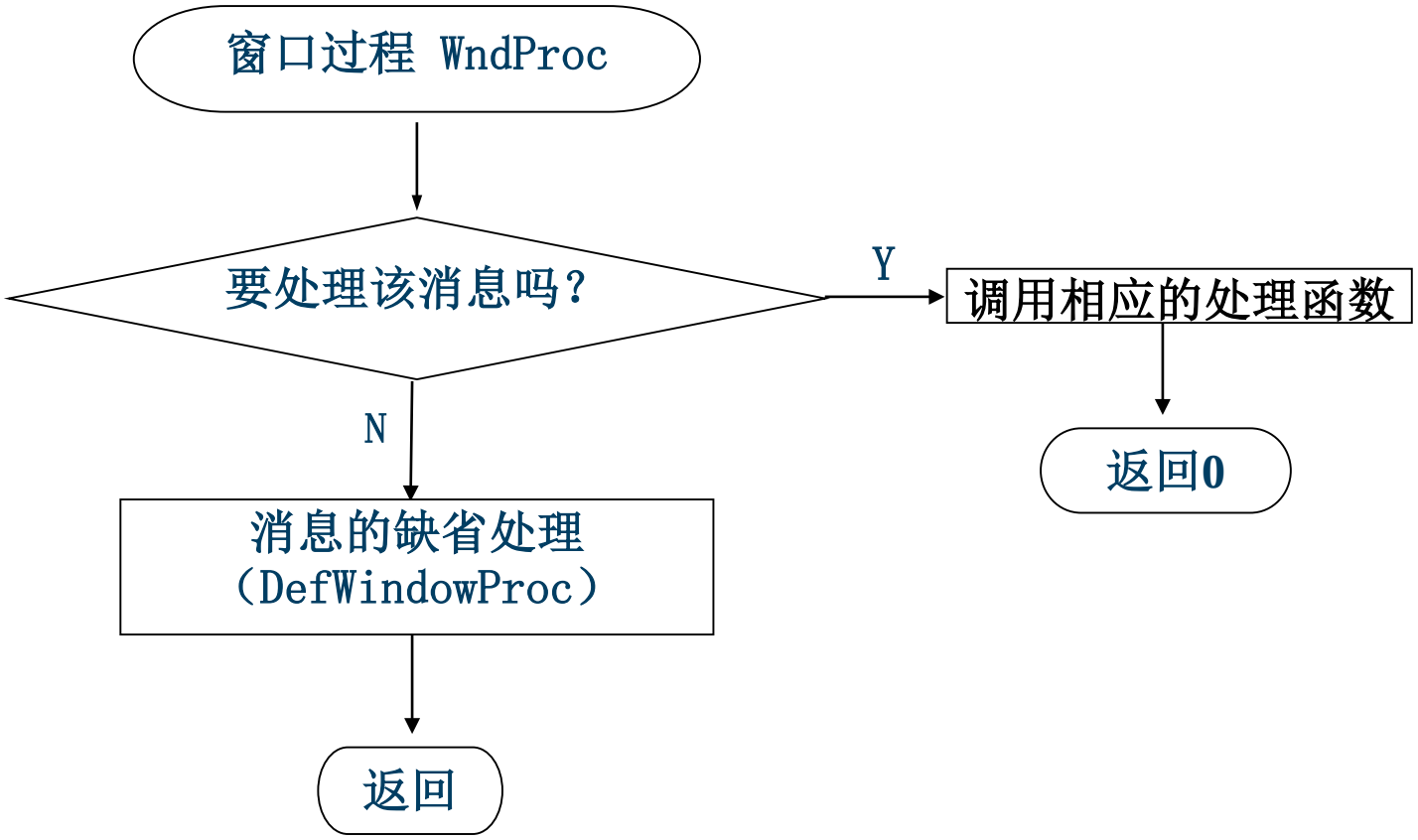
(3) 窗口消息处理程序（窗口过程）

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：

```
WndProc PROTO   hWin   :DWORD,
                uMsg   :DWORD,
                wParam :DWORD,
                lParam :DWORD
```

6.4.2 WIN32程序的结构 (7)



```
WndProc   proc hWnd   :DWORD,   uMsg   :DWORD,
          wParam :DWORD,        lParam :DWORD
  .IF uMsg == WM_DESTROY
          INVOKE  PostQuitMessage, NULL
  .ELSEIF uMsg == WM_COMMAND
         MOV EAX, wParam
        .if  AX == IDM_TEST
            invoke MessageBox, NULL,ADDR Test_string,
                                  OFFSET AppName, MB_OK
        .elseif   AX == IDM_HELLO  ……
        .elseif   AX == IDM_GOODBYE
           invoke MessageBox,NULL,ADDR Goodbye_string,
                                  OFFSET AppName,MB_OK
        .else
             INVOKE  DestroyWindow,hWnd
        .ENDIF
  .ELSE
         INVOKE  DefWindowProc,hWnd,uMsg,wParam,lParam
         RET
  .ENDIF
   XOR  EAX, EAX
   RET
WndProc  ENDP
```

较高版本的汇编程序提供了条件控制流(.IF .ELSE) 伪指令和关系操作符(如==, >=, !=,等 p253)

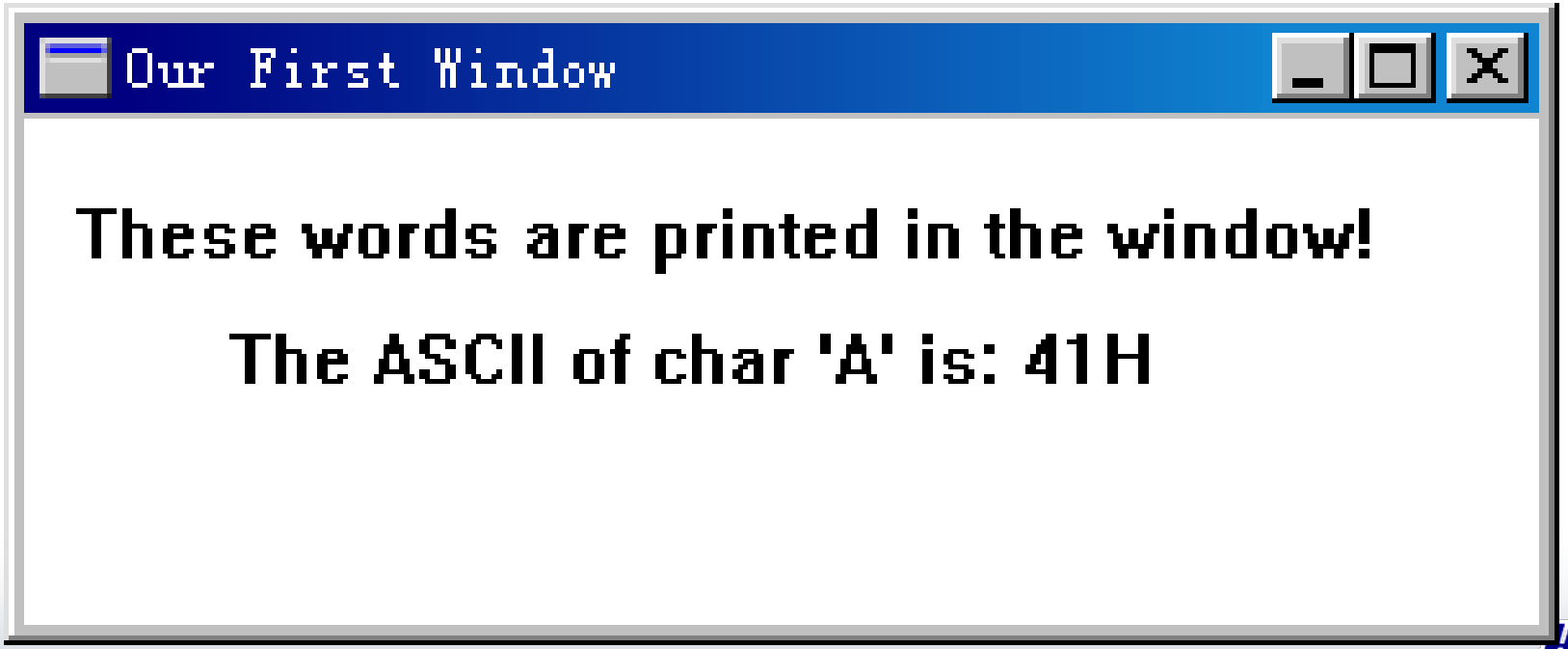
6.4.3 WINDOWS API函数简介

華中科技大學

参考 WINDOWS API 相关手册

6.4.4 WIN32编程实例 (1)

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：



6.4.4 WIN32编程实例 (2)

c6_267.ASM

6.4.4 WIN32编程实例 (3)

1. 头文件和数据段定义
2. 主程序
3. 窗口主程序
4. 窗口消息处理程序
5. 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:  ",
szASCII     DB  "00H  "
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，返回Windows
```

窗口主程序（1）


```
WinMain PROC hInst: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
```

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

華中科技大學

```
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2进制数按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
```

WIN32程序源码符号级调试

(1) 进入命令行状态

(2) 设置环境（如运行var.bat）

(3) 建立源文件（demo.asm）

(4) 编译：ml /Fl /c /coff /Zi demo.asm

(5) 连接：link /subsystem:windows /debug
/debugtype:cv demo.obj

(6) 调试：运行VC2008或VC6，执行：

“文件” ->“打开” ->“项目”，选 demo.exe

“文件” ->“打开” ->“文件”，选 demo.asm

然后就可以进行源码级调试了。

第6章 小结

(1) 输入输出指令IN、OUT的使用格式及功能；

(2) CPU与外设的数据传送方式；

(3) 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；

(4) 段的简化定义方法；

.MODEL/.CODE/.DATA/.STACK

(5) 结构的定义与使用方法STRUCT ；

(6) 原型定义与函数调用；

(7) 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。