



华中科技大学

# 计算机系统基础

许向阳

xuxy@hust.edu.cn

华中科技大学计算机科学与技术学院



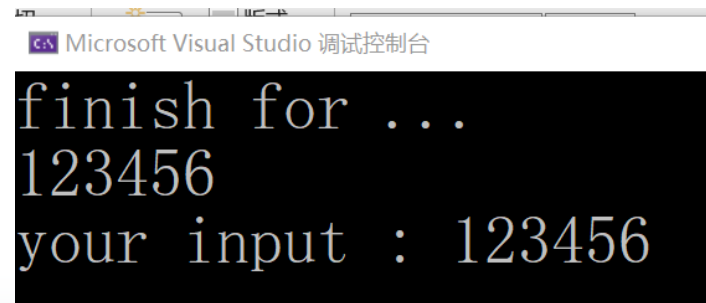


# 第12章 中断和异常处理

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
    int i, j;
    int r = 0;
    char msg[20];
    for (i=0; i<20000; i++) {
        r = 0;
        for (j = 0; j < 100000; j++)
            r = r + 1;
    }
    printf("finish for ... \n");
    scanf("%s", msg);
    printf("your input : %s \n", msg);
    return 0;
}
```

循环语句的执行时间较长，  
循环执行后显示 finish  
for ...

Q: 在出现finish...之前，  
输入一行 123456，  
运行结果结果是什么？



Microsoft Visual Studio 调试控制台

```
finish for ...
123456
your input : 123456
```

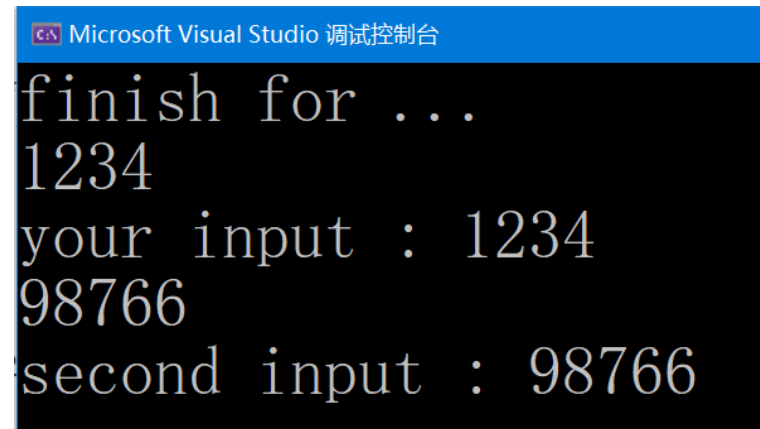




# 第12章 中断和异常处理

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
    int i, j;
    int r = 0;
    char msg[20];
    for (i=0; i<20000; i++) {
        .....
    }
    printf("finish for ... \n");
    scanf("%s", msg);
    printf("your input : %s \n", msg);
    scanf("%s", msg);
    printf("second input : %s \n", msg);
    return 0;
}
```

Q: 在出现finish...之前,  
输入一行 1234,  
再输入一行 : 98766  
运行结果结果是什么?



Microsoft Visual Studio 调试控制台

```
finish for ...
1234
your input : 1234
98766
second input : 98766
```

Q: 如何解释看到的现象?





# 第12章 中断和异常处理

## 12.1 中断与异常的基础知识

中断和异常的概念、中断描述符表

中断和异常的响应过程、软中断指令

## 12.2 Windows中的结构化异常处理

编写异常处理函数

异常处理程序的注册

全局异常处理程序的注册

## 12.3 C异常处理程序反汇编分析





# 12.1 中断与异常的基础知识

## 12.1.1 中断和异常的概念

日常生活当中的“中断”

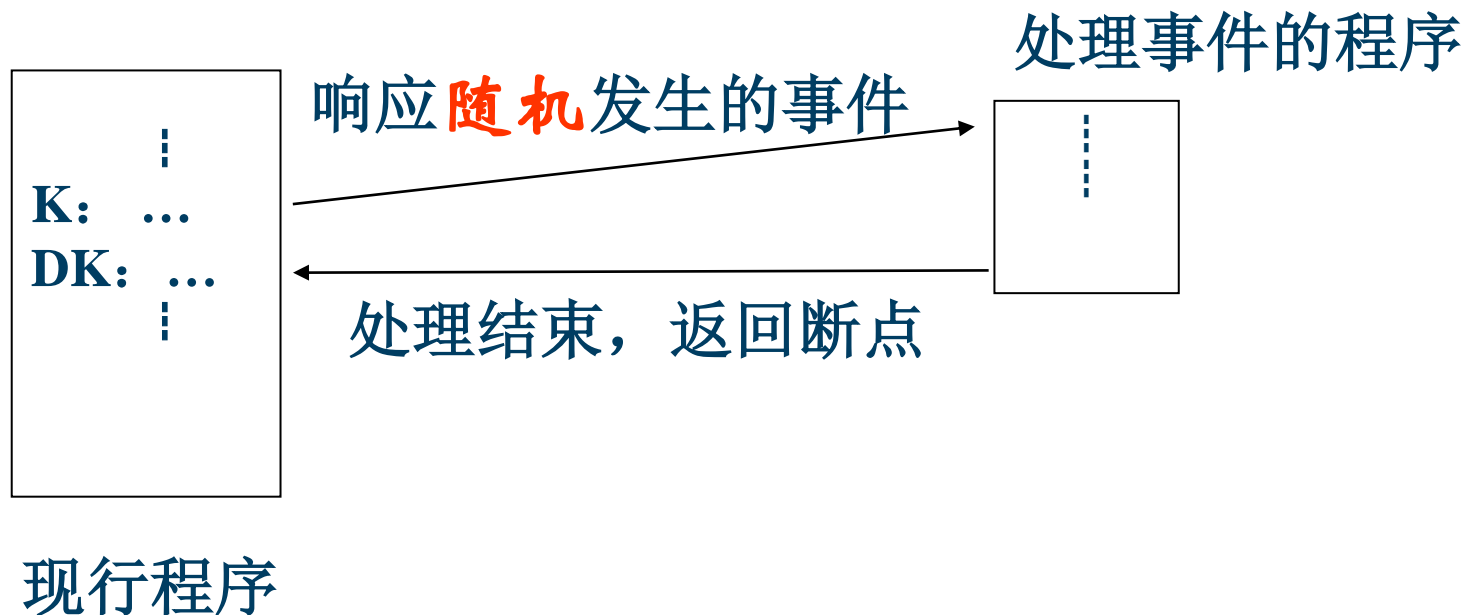
计算机世界中的“中断”

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。



# 12.1 中断与异常的基础知识

## 中断处理过程：



Q: 与子程序调用有什么差别？

事先安排好的      VS      随机发生的  
有转移指令          VS      无转移指令

# 12.1 中断与异常的基础知识



华中科技大学

## 12.1.1 中断和异常的概念

- (1) 为什么要引入中断机制?
- (2) 有哪些事情会引起中断? 中断源
- (3) CPU为什么能感知中断? 中断系统
- (4) 在何处去找中断处理程序? 中断描述符表  
中断矢量表
- (5) 如何从中断处理程序返回?
- (6) 如何使用中断?





# 12.1 中断与异常的基础知识

## 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:

键盘、鼠标、时钟.....

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。

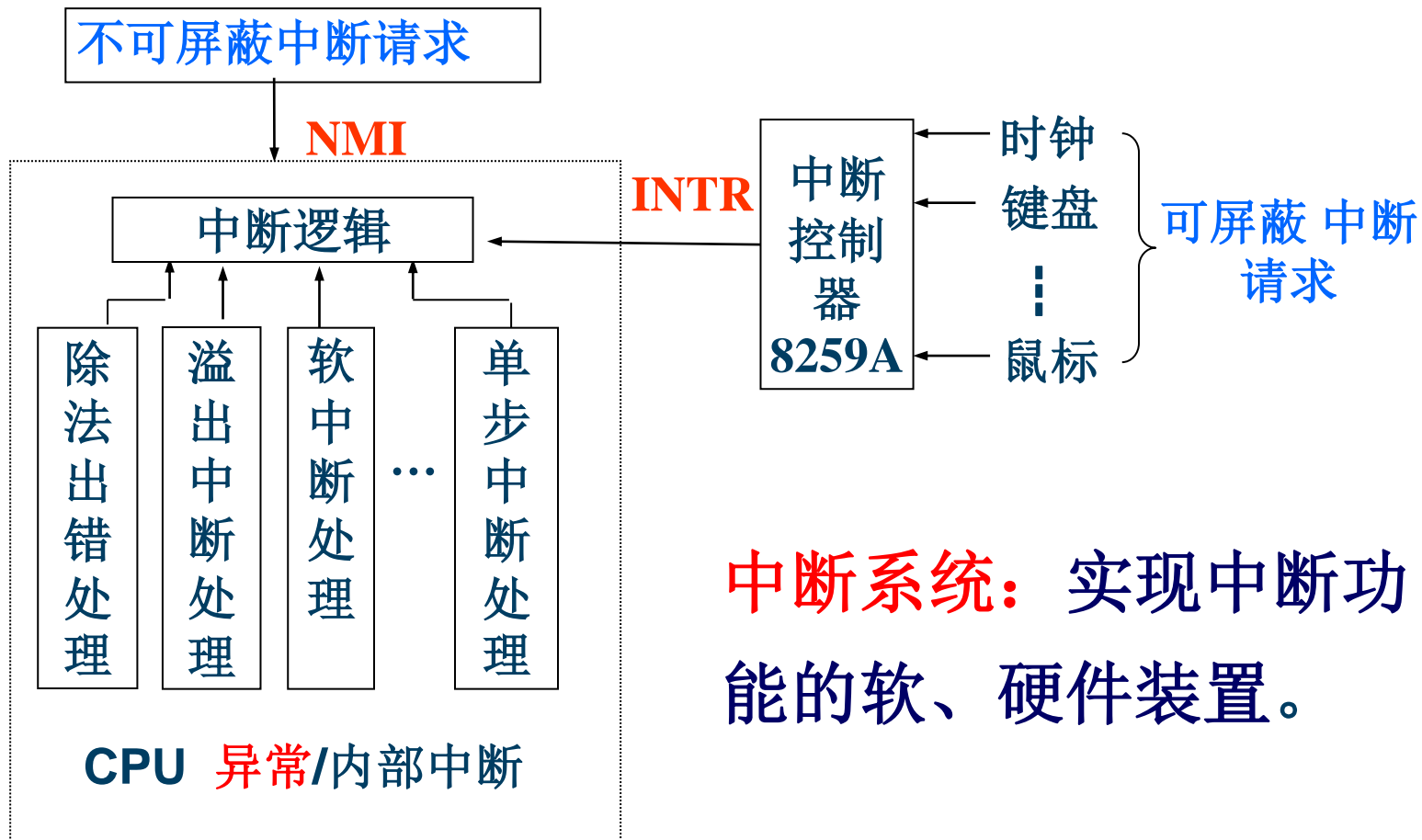
程序检测:

软中断, 包括指令INTO、  
INT n 和 BOUND等。





# 12.1 中断与异常的基础知识



**中断系统：**实现中断功能的软、硬件装置。



# 12.1 中断与异常的基础知识

## 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 <b>NMI</b> <b>INTR</b>	最高 ↓ 最低





# 12.1 中断与异常的基础知识

Intel 8086/8088 称外中断、内中断  
从 80286 开始，称为中断、异常

➤ **中断**：由外部设备触发、与正在执行的指令无关、异步事件

➤ **异常**：与正在执行的指令相关的 **同步事件**。

CPU内部出现的中断，也称为**同步中断**。

一条指令的执行过程中，CPU检测到了某种预先定义条件，产生的一个异常信号，进而调用**异常处理程序**对该异常进行处理。





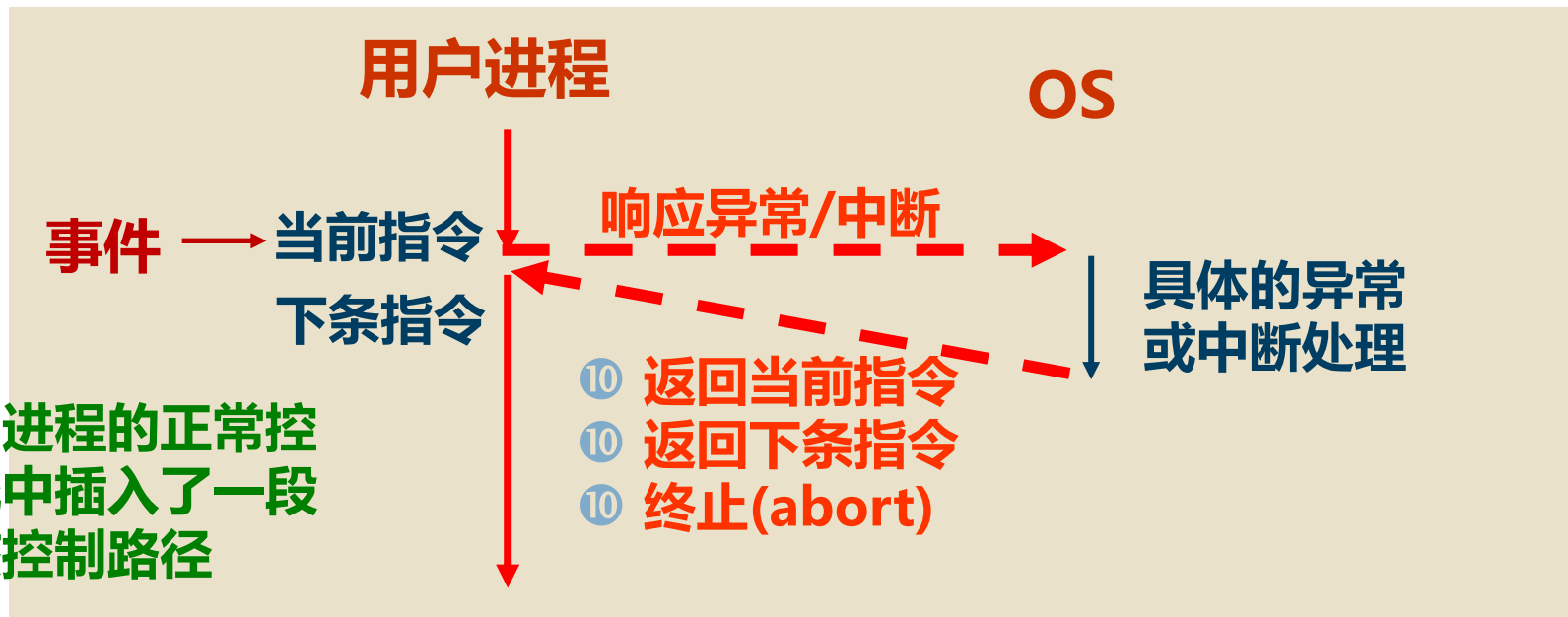
# 12.1 中断与异常的基础知识

- 在Intel CPU中，异常分为三类：  
故障（faults）、陷阱（traps）、中止（aborts）。
- 在异常处理程序执行后，后续操作取决于异常的类型：
  - 重新执行引起异常的指令 —— 故障
  - 执行引起异常指令之下的指令 —— 陷阱
  - 终止程序运行 —— 中止
- 中断处理程序执行后会返回到被中断处继续执行



# 12.1 中断与异常的基础知识

用户进程的正常控制流中插入了一段内核控制路径





# 12.1 中断与异常的基础知识

## ➤ 异常 —— 故障

- 故障异常是在引起异常的指令之前或者指令执行期间，在检测到故障或者预先定义的条件不能满足时产生。
- 常见的故障异常
  - 除法出错（除数为0；除数很小被除数很大，商溢出）
  - 数据访问越界（访问一个不准本程序访问的内存单元）
  - 缺页
- 故障异常通常可以纠正，处理完异常时，引起故障的指令被重新执行。





# 12.1 中断与异常的基础知识

## ➤ 异常 —— 陷阱

- 在执行引起异常的指令之后，把异常情况通知给系统。
- 执行异常处理程序后，回到产生异常信号指令之下的一条语句。
- **软中断**是一种常见的**陷阱**。所谓软中断就是在程序中写了中断指令，执行该语句就会去调用中断处理程序，中断处理完后又继续运行下面的程序。
- 软中断调用与调用一般的子程序非常类似。借助于中断处理这一模式，可以调用操作系统提供的服务程序。
- 另外一种常见的陷阱异常是**单步异常**，用于**防止一步步跟踪程序**。





# 12.1 中断与异常的基础知识

## ➤ 异常 ——中止

- 中止是在系统出现严重问题时通知系统的一种异常；
- 引起中止的指令是无法确定的；
- 产生中止时，正执行的程序不能被恢复执行。系统接收中止信号后，处理程序要重新建立各种系统表格，并可能重新启动操作系统；
- 中止的例子包括硬件故障和系统表中出现非法值或不一致的值；







## 12.1.2 中断描述符表

- 每一个中断或异常处理程序都有一个入口地址；
- 将中断和异常处理程序的入口地址等信息称为门 (gate)，就像一栋楼房的门代表了该楼房的入口一样；
- 根据中断和异常处理程序的类别，将与之连接的中断描述符划分为三种门：
  - 任务门（执行中断处理程序时将发生任务转移）
  - 中断门（主要用于处理外部中断）
  - 陷阱门（主要用于处理异常）
- CPU根据门提供的信息（由IDT中的门属性字节提供）进行切换，对不同的门，处理过程是有些差异的。





## 12.1.2 中断描述符表

中断号	名称	类型	相关指令	DOS下名称
0	除法出错	异常	<b>DIV, IDIV</b>	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	<b>INT 3</b>	断点
4	溢出	异常	<b>INTO</b>	溢出
5	边界检查	异常	<b>BOUND</b>	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 <b>WAIT</b>	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



## 12.1.2 中断描述符表

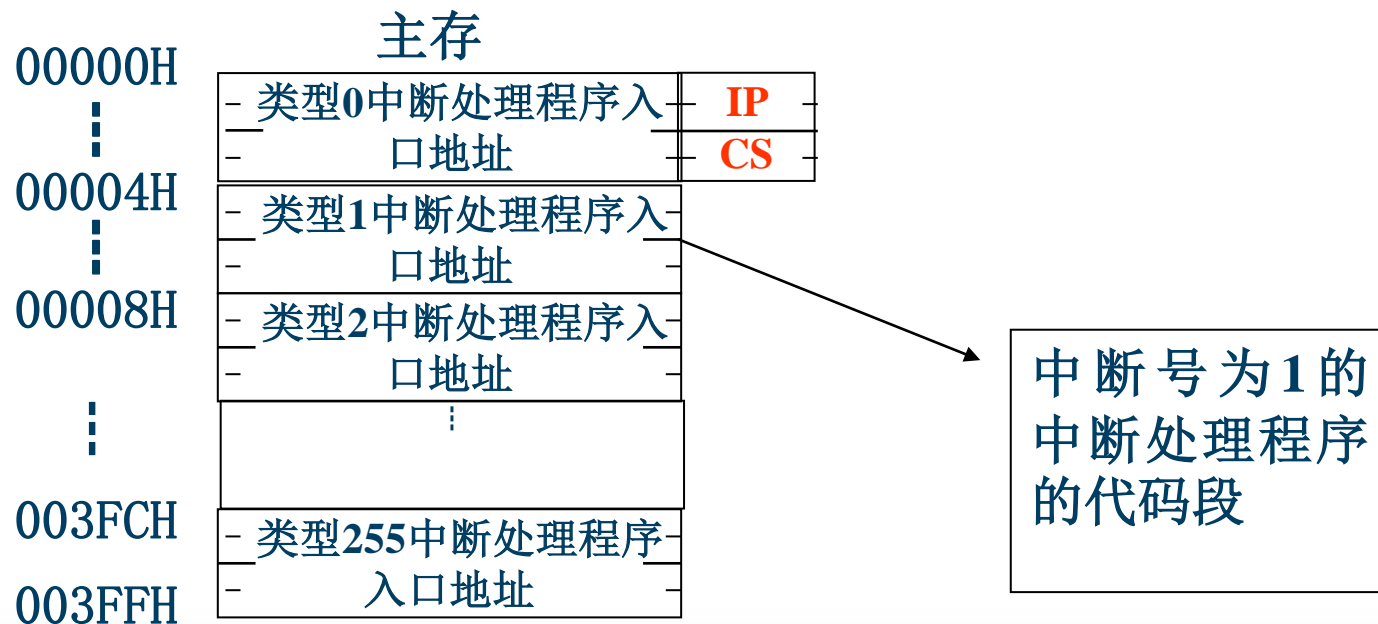
中断类型码与对应的中断处理程序之间的连接表，存放的是中断处理程序的入口地址（也称为中断矢量或中断向量）。



## 12.1.2 中断描述符表

### ➤ 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





## 12.1.2 中断描述符表

### ➤ 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, Interrupt Descriptor Table)
- 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。



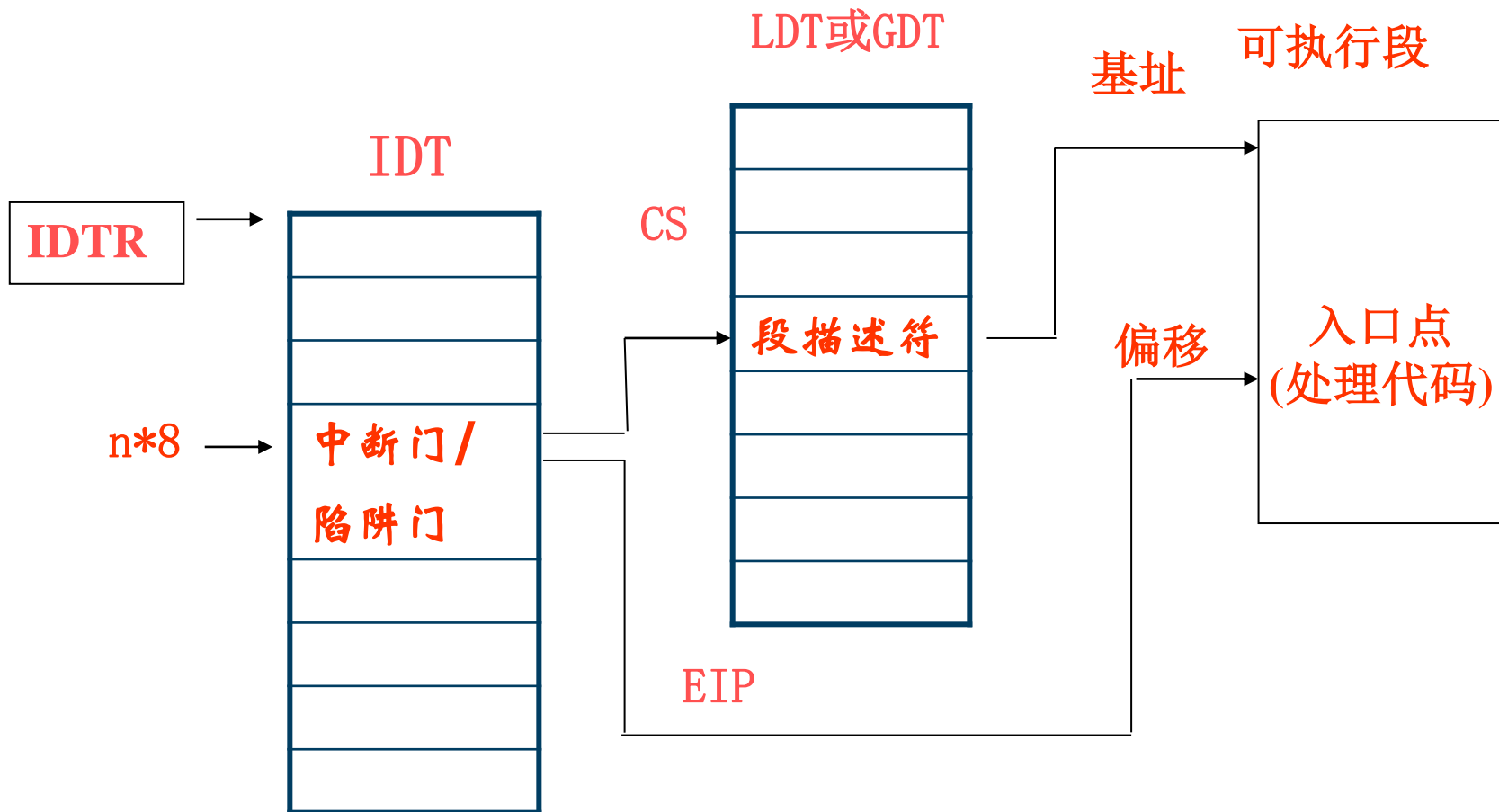
# 12.1.2 中断描述符表

## 保护方式下的中断描述符表

主存	31	15	7	0
00000H ⋮ 00008H	- 类型0中断处理程序入口信息	偏移值(高16位)	门属性	未用
⋮ 00010H ⋮ ⋮	- 类型1中断处理程序入口信息	段选择符(16位)	偏移值(低16位)	
	- 类型2中断处理程序入口信息			
	⋮			
007F8H 007FFH	- 类型255中断处理程序入口信息			

**IDTR 决定 IDT 的起始 PA。**

## 12.1.2 中断描述符表



从中断号到中断处理程序的转换过程





## 12.1.4 软中断指令

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

格式：INT n

功能：

$(EFLAGS) \rightarrow \downarrow (ESP)$  ,

$0 \rightarrow TF$ , 中断门还要将  $0 \rightarrow IF$





## 12.1.4 软中断指令

② (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 (EIP) → ↓ (ESP)

从门或TSS描述符中分离出的偏移值 → EIP



## 12.1.4 软中断指令

### 2. 中断返回指令

格式: IRET

功能: ①  $\uparrow (\text{ESP}) \rightarrow \text{EIP}$   
②  $\uparrow (\text{ESP})$  取低16位  $\rightarrow \text{CS}$

恢复断  
点地址

③  $\uparrow (\text{ESP}) \rightarrow \text{EFLAGS}$

恢复标志寄  
存器的内容



# 实方式下中断处理程序示例

## ■ 上机演示

- ◆ 查看 INT 21H 中断处理程序的入口地址  
内存的最低端，84H处  
数据区 显示 0 : 84 处的内容
- ◆ 跟踪进入 中断处理程序 (Alt+F7)  
查看 CS、IP是否为 上一步看到的程序入口地址  
查看 堆栈是否存放的INT 21H 之下的断点地址





華中科技大學

# 实方式下中断处理程序示例

- 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





# 实方式下中断处理程序示例

## 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。





# 实方式下中断处理程序示例

## 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: 直接使用CS,当子程序与主程序在  
          同一个段时





# 实方式下中断处理程序示例

## 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?**

**方法1 : 直接写中断矢量表的相关位置;**

**方法2 : DOS系统功能调用**







# 实方式下中断处理程序示例

**例：读出 08H 号中断处理程序的入口地址。**

**方法1：直接读中断矢量表的相关位置**

```
MOV  AX, 0
MOV  DS, AX
MOV  AX, ds:[0020H]      ; 访问DS: [08H*4]单元
                           ; 即0: 0084H单元

MOV  BX, ds:[0034]
```

**方法2：DOS系统功能调用**

```
MOV  AX, 3508H
INT  21H
      (ES):(BX) 为 中断 08H 服务程序的入口地址
```





# 实方式下中断处理程序示例

## 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?

Q: 如何调用新的中断处理程序呢?





# 实方式下中断处理程序示例

## 2. 修改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?**





# 实方式下中断处理程序示例

## 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

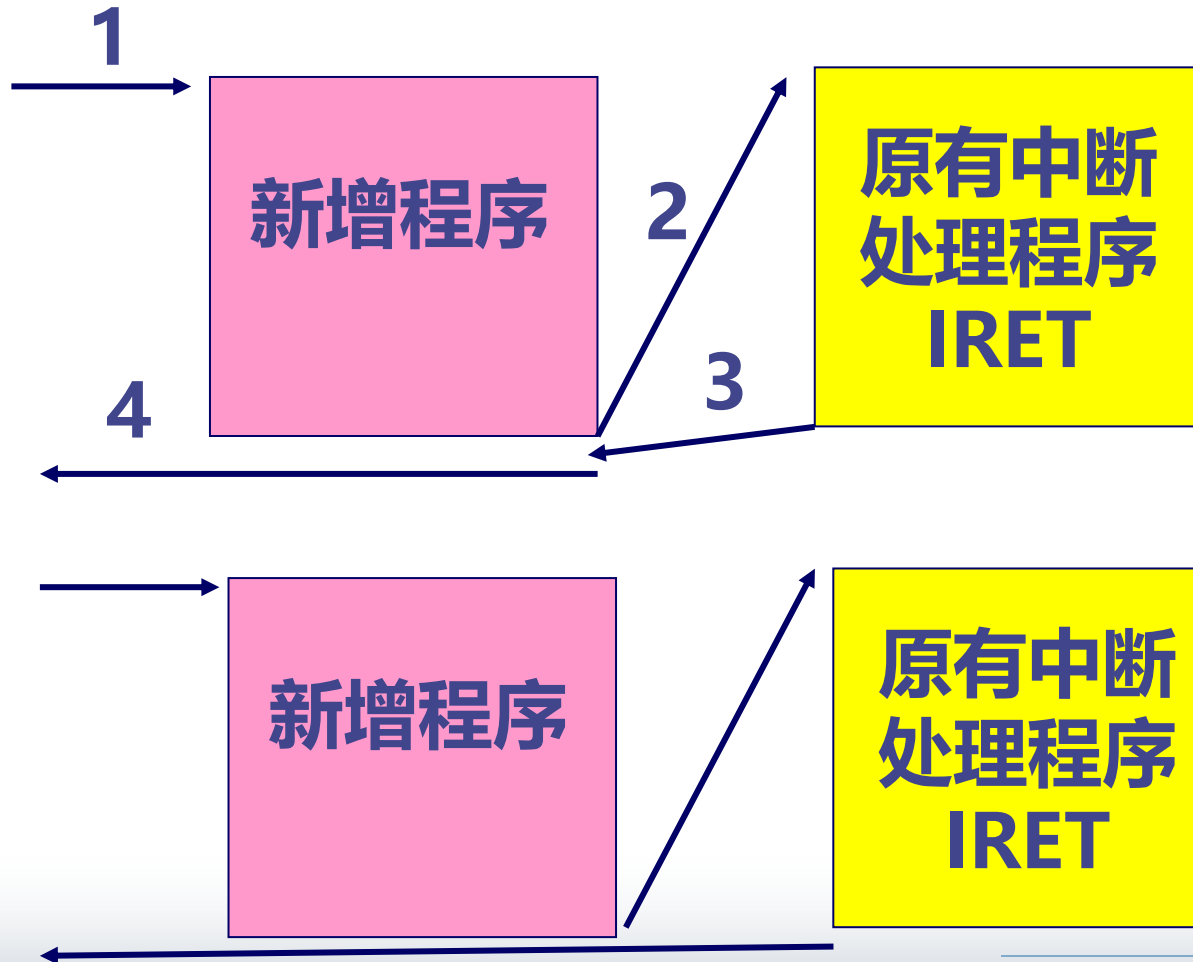
新增程序

原有中断  
处理程序  
IRET



# 实方式下中断处理程序示例

## 2. 修改已有中断处理程序以扩充其功能





# 实方式下中断处理程序示例

## 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）



# 实方式下中断处理程序示例

**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)**

**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.**





# 实方式下中断处理程序示例

## 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?







# 实方式下中断处理程序示例

## 程序涉及的知识要点分析:

### (1)每隔1秒

定时中断，时钟中断

<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。





# 实方式下中断处理程序示例

## 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





# 实方式下中断处理程序示例

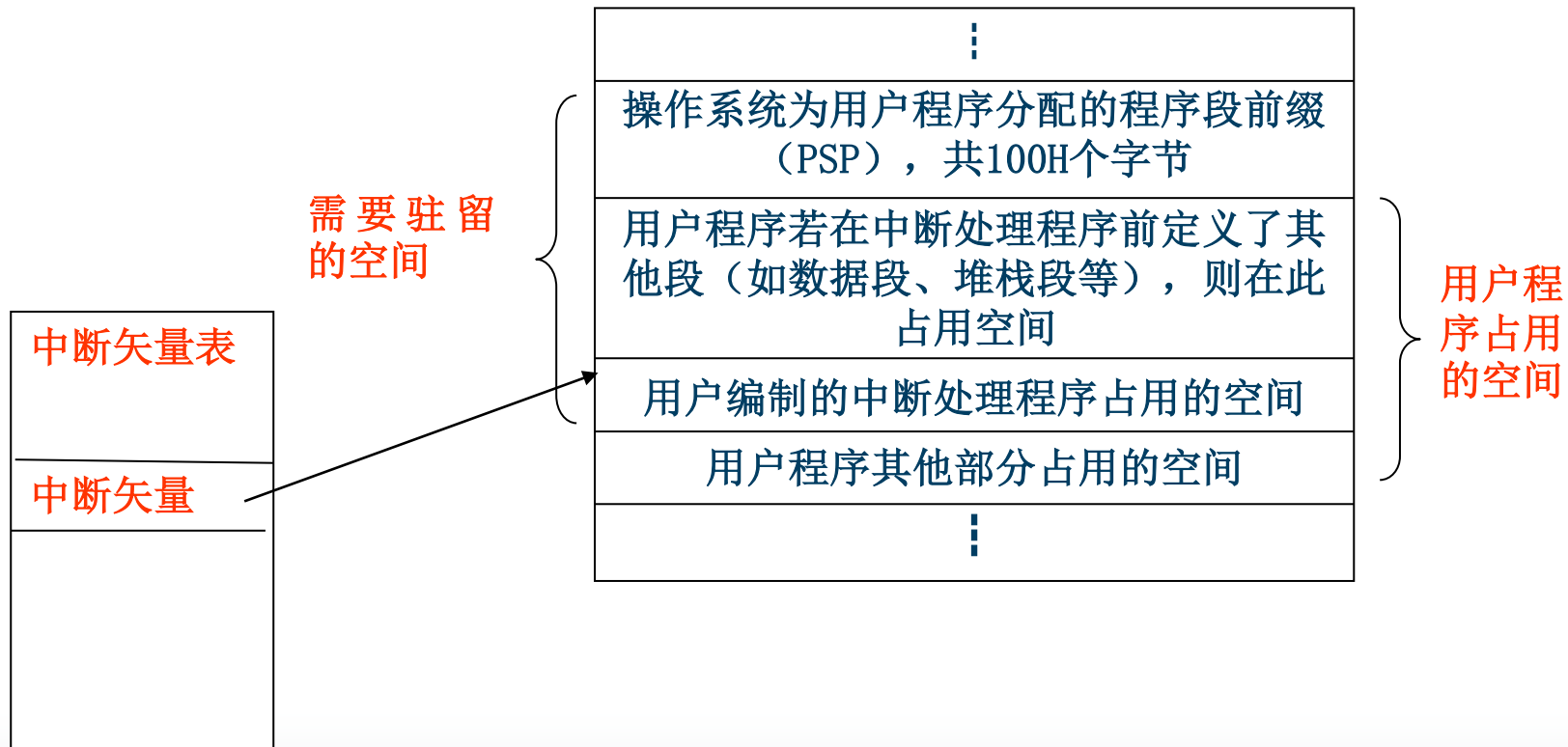
## (3) 程序驻留

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。





# 实方式下中断处理程序示例



# 12.2 Windows 中的结构化异常处理



华中科技大学



# 12.2 Windows 中的结构化异常处理



华中科技大学

Windows 系统：结构化异常处理

Structured Exception Handling, SHE

Windows XP：向量化异常处理

Vectored Exception Handling, VEH

C++ 异常处理

C++ Exception Handler, C++EH





## 12.2 Windows 中的结构化异常处理

- 如何感知到出现了异常？
- 异常发生后如何处理？
- 如何去找到异常处理程序？
- 如何编写异常处理程序？
- 如何让系统找到自己的编写的异常处理程序？
- 异常处理程序执行后，再做什么？



## 12.2 Windows 中的结构化异常处理

```
#include <signal.h>           #include <setjmp.h>
#include <stdio.h>             #include <Windows.h>
jmp_buf buf;
LONG WINAPI Exceptionhandler(EXCEPTION_POINTERS* ExceptionInfo)
{
    printf("error \n");
    longjmp(buf, 1);
}

int main()
{
    if (!setjmp(buf)) {
        printf("then begin ... \n");
        Exceptionhandler(NULL);
        printf("then over ... \n");
    }
    else { printf("else branch ... \n"); }
    printf("finish .... \n");
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
then begin ...
error
else branch ...
finish ....
```





## 12.2 Windows 中的结构化异常处理

```
typedef struct __JUMP_BUFFER
{
    unsigned long Ebp;
    unsigned long Ebx;
    unsigned long Edi;
    unsigned long Esi;
    unsigned long Esp;
    unsigned long Eip;
    unsigned long Registration;
    unsigned long TryLevel;
    unsigned long Cookie;
    unsigned long UnwindFunc;
    unsigned long UnwindData[6];
} __JUMP_BUFFER;
```





# 12.2 Windows 中的结构化异常处理

```
if (!setjmp(buf)) {
```

006818E6	6A 00	push	0
006818E8	68 C0 A5 68 00	push	offset buf (068A5C0h)
006818ED	E8 61 F9 FF FF	call	__setjmp3 (0681253h)
006818F2	83 C4 08	add	esp, 8
006818F5	85 C0	test	eax, eax
006818F7	75 23	jne	main+4Ch (068191Ch)

```
printf("then begin ... \n");
```

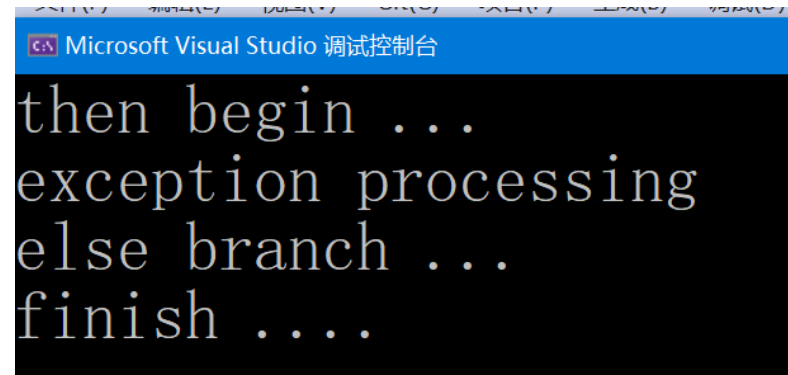
100 %	
监视 1	
搜索(Ctrl+E) 搜索深度: 3	
名称	值
buf	0x0068a5c0 {4128236, 4861952, 6819875, 6819875, 4128020, 6822130, 4128344, 0, 1447244336, 0, 0, 0, 0, ...}
eip	6821818
ebp	4128236
ebx	4861952



## 12.2 Windows 中的结构化异常处理


```
jmp_buf buf;
LONG WINAPI Exceptionhandler(EXCEPTION_POINTERS* ExceptionInfo)
{
    printf("exception processing \n");
    longjmp(buf, 1);
}

int main()
{
    int a, t;
    SetUnhandledExceptionFilter(Exceptionhandler);
    if (!setjmp(buf)) {
        printf("then begin ... \n");
        a = 100;          t = 0;
        a = a / t;
        printf("then over ... \n");
    }
    else printf("else branch ... \n");
    printf("finish .... \n");
    return 0;
}
```





## 12.2 Windows 中的结构化异常处理

```
14 int main()  
15 { int a, t;  
16 // SetUnhandledExceptionFilter(Exceptionhandler);  
17 if (!setjmp(buf))  
18     printf("then over ... \n");  
19     a = 100;  
20     t = 0;  
21     a = a / t;   
22     printf("then over ... \n");  
}
```

未经处理的异常

0x00531918 处有未经处理的异常(在 c\_divide\_zero.exe 中):  
0xC0000094: Integer division by zero.

[复制详细信息](#) | [启动 Live Share 会话...](#)

[异常设置](#)



## 12.2.1 编写异常处理函数

### 回调函数：

当执行一条指令出现异常时，操作系统可以自动调用程序开发者编写的一个异常处理函数。

- 程序员编写出接口符合规范的异常处理函数；
- 向操作系统注册该函数；
- 回调函数可以显示错误信息、修复错误或者完成其他任务；
- 回调函数最后都要返回一个值来告诉操作系统下一步做什么。





## 12.2.1 编写异常处理函数

异常处理回调函数的格式:

```
EXCEPTION_DISPOSITION __cdecl _except_handler(  
    struct _EXCEPTION_RECORD *ExceptionRecord,  
    void * EstablisherFrame,  
    struct _CONTEXT *ContextRecord,  
    void * DispatcherContext);
```





## 12.2.1 编写异常处理函数

EXCEPTION\_RECORD定义在winnt.h中，含有异常的编码、异常标志、异常发生的地址、参数个数、异常信息。

```
typedef struct _EXCEPTION_RECORD {  
    DWORD      ExceptionCode;  
    DWORD      ExceptionFlags;  
    struct _EXCEPTION_RECORD *ExceptionRecord;  
    PVOID      ExceptionAddress;  
    DWORD      NumberParameters;  
    ULONG_PTR  ExceptionInformation  
                [EXCEPTION_MAXIMUM_PARAMETERS];  
} EXCEPTION_RECORD;
```





## 12.2.1 编写异常处理函数

```
typedef enum _EXCEPTION_DISPOSITION {  
    ExceptionContinueExecution,  
        //已经处理了异常，回到异常触发点继续执行  
    ExceptionContinueSearch,  
        //继续遍历异常链表，寻找其他的异常处理方法  
    ExceptionNestedException,  
        // 在处理过程中再次触发异常  
    ExceptionCollidedUnwind  
} EXCEPTION_DISPOSITION;
```

下一步准备采取的动作







## 12.2.2 异常处理程序的注册

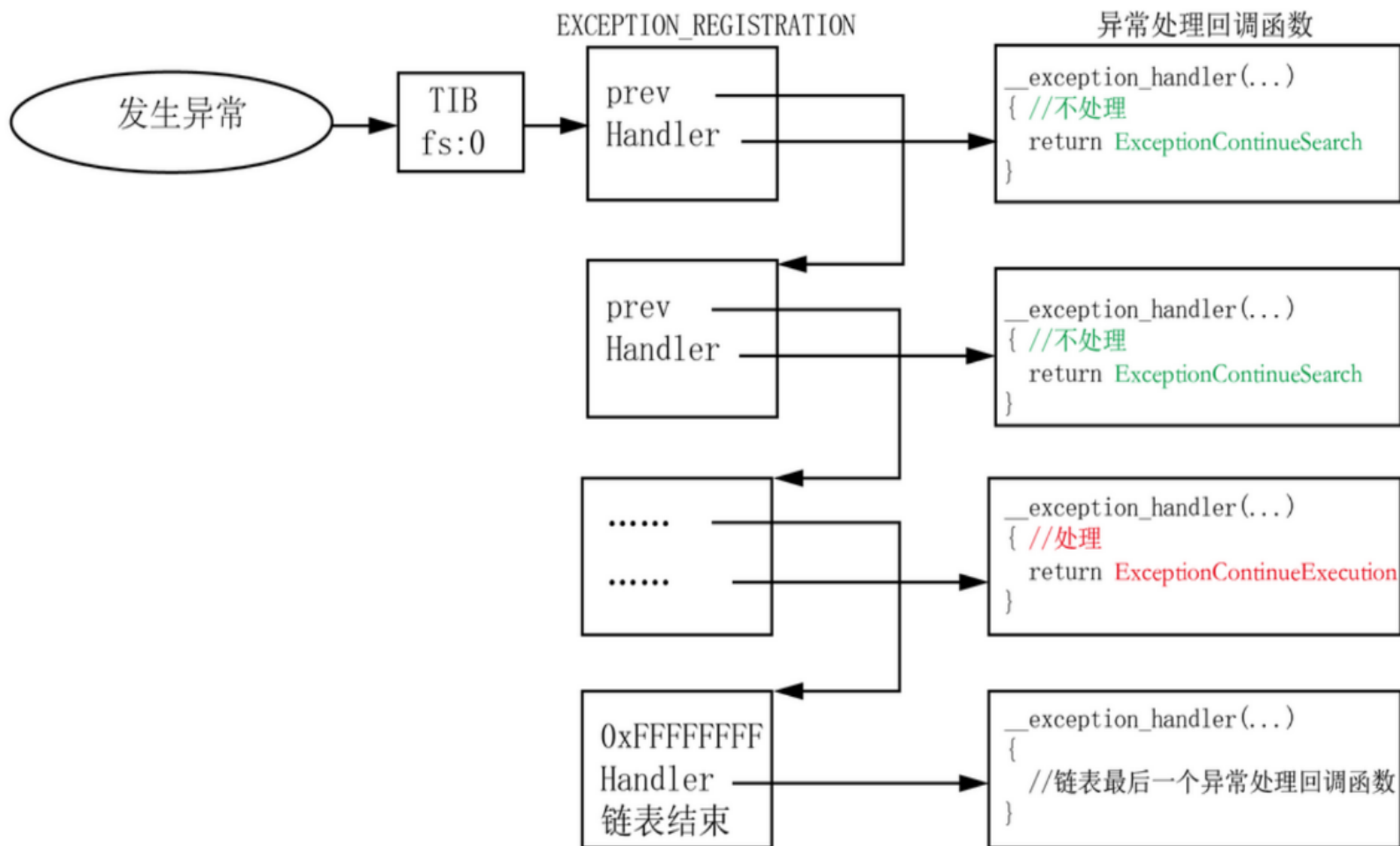
在windows系统中，对每一个进程都创建一个线程信息块TIB（Thread Information Block）来保存与线程有关的信息。线程信息块的数据结构NT\_TIB的定义在winnt.h中。

TIB 的第一个字段：

```
struct _EXCEPTION_REGISTRATION_RECORD *ExceptionList;
```



## 12.2.2 异常处理程序的注册





## 12.2.2 异常处理程序的注册

在程序执行过程中遇到异常事件时，操作系统中的函数 `RtlDispatchException` 会从 `FS:[0]` 指向的链表表头开始，依次调用每个节点指明的异常处理回调函数，直到某个异常处理回调函数的返回值为0为止（即函数的返回值为 `ExceptionContinueExecution`）。

异常处理回调函数的返回值为0，表示已经处理了该异常，该线程可以恢复执行。链表最末一项是操作系统在装入线程时设置的指向 `UnhandledExceptionFilter` 函数，该函数总是向用户显示 “Application error” 对话框。





## 12.3 C异常处理程序反汇编分析

对C语言程序编译时，编译器会生成一些语句：

- 建立和恢复异常处理注册记录链表
- 使用FS:[0]指向链表表头
- 在异常处理所需信息的数据区 `__sehtable$_main` 中，会设置异常处理程序的入口地址。



- 中断和异常的概念

中断源、中断描述符表

- 中断和异常响应的过程

- 中断和异常处理程序的调用与返回

**INT    IRET**

- 中断服务程序调用与一般的子程序调用的异同点