实验报告：重写溢出中断服务程序

一、 实验目的

1. **深入理解标志寄存器：** 区分进位标志（CF）与溢出标志（OF）在无符号数与有符号数运算中的不同作用。
2. **掌握中断机制：** 理解 x86 架构下的中断向量表（IVT）结构，掌握挂接（Hook）自定义中断服务程序的方法。
3. **应用 INTO 指令：** 学习使用 `INTO` 指令替代传统的条件跳转指令（`JO`/`JNO`）来处理算术溢出异常。
4. **混合编程实践：** 熟练掌握 C 语言与内联汇编（Inline Assembly）的混合编程技巧。

二、 实验环境

- **宿主系统：** macOS (使用 DOSBox 模拟器)
- **运行环境：** MS-DOS 16位实模式
- **开发工具：** Borland Turbo C 2.0
- **硬件架构模拟：** Intel 8086/8088 兼容架构

三、 实验原理

1. 溢出标志 (OF) 与 C 语言的局限性

在 16 位有符号二进制补码运算中，数据范围为 `-32768` 到 `+32767`。当运算结果超出此范围（例如 `32767 + 1`），会发生“溢出”，导致最高位符号位突变（正数变负数），此时 CPU 会将标志寄存器的 **OF (Overflow Flag)** 置为 1。 C 语言标准规定有符号数溢出属于“未定义行为”（Undefined Behavior），编译器通常不自动检测，这会给系统带来隐患。因此需要利用底层汇编机制进行捕获。

2. INTO 指令的工作机制

本实验不使用 `JO` (Jump if Overflow) 指令进行软件逻辑判断，而是使用 `INTO` (Interrupt on Overflow) 指令。`INTO` 是一条单字节指令，其微操作逻辑如下：

Plaintext

```
IF (OF == 1) THEN
    PUSH Flags
    PUSH CS
    PUSH IP
    OF = 0, TF = 0
    CALL Far Pointer [0000:0010] (即执行4号中断)
END IF
```

它利用硬件电路自动检测 OF 标志，一旦溢出直接触发异常处理，无需程序显式进行比较跳转。

3. 中断向量表的修改

系统默认的 4 号中断处理程序通常是 `IRET`（直接返回）。为了捕获错误，我们需要修改中断向量表（IVT），将 4 号中断的入口地址指向我们自定义的函数 `new_overflow_handler`。

四、 实验内容与步骤

1. 实验流程设计

1. **保存现场：** 使用 `getvect(4)` 获取并保存系统原有的 4 号中断向量。
2. **安装中断：** 编写自定义中断服务程序（打印报错信息），并使用 `setvect(4, new_handler)` 将其挂载。
3. **触发溢出：**
    - 定义 `int a = 32767; int b = 1;`
    - 使用内联汇编 `ADD AX, BX` 执行加法，导致 `OF=1`。
    - 紧接着执行 `INTO` 指令。
4. **观察结果：** 验证程序是否跳入自定义函数并输出警告。
5. **恢复现场：** 程序结束前，务必使用 `setvect` 将 4 号中断恢复为原值，防止系统不稳定。

2. 核心代码实现

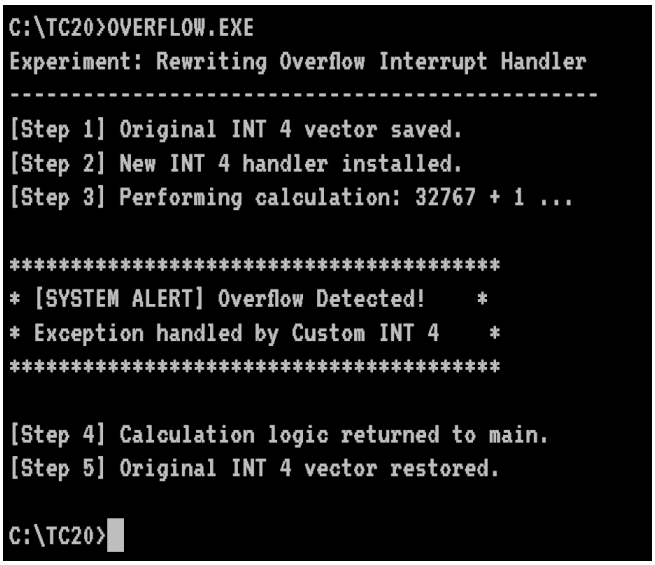
```
/* * 文件名：OVERFLOW.C
 * 描述：通过重写INT 4中断向量，捕获INTO指令触发的溢出异常
 */
#include <stdio.h>
#include <dos.h>
#include <conio.h>

/* 全局变量保存旧中断向量 */
```

```
void interrupt (*old_handler)(void);

/* * 自定义中断服务程序 (ISR)
 * 关键字 interrupt 指示编译器自动添加 IRET 指令及寄存器保护
 */
void interrupt new_overflow_handler(void) {
    printf("\n\n**************************************\n");
    printf("* [SYSTEM ALERT] Overflow Detected!   *\n");
    printf("* Exception handled by Custom INT 4   *\n");
    printf("**************************************\n\n");
}

int main() {
    int a = 32767; /* 0x7FFF */
    int b = 1;     /* 0x0001 */

    clrscr();
    printf("Experiment: Rewriting Overflow Interrupt Handler\n");
    printf("------------------------------------------------\n");

    /* 1. 保存旧中断 */
    old_handler = getvect(4);
    printf("[Step 1] Original INT 4 vector saved.\n");

    /* 2. 安装新中断 */
    setvect(4, new_overflow_handler);
    printf("[Step 2] New INT 4 handler installed.\n");

    printf("[Step 3] Performing calculation: %d + %d ...\n", a, b);

    /* 3. 内联汇编制造溢出并触发中断 */
    asm {
        mov ax, a
        add ax, b   /* AX = 0x8000 (-32768), OF Flag set to 1 */

        /* * 关键点：不使用 JO 跳转，使用 INTO 触发硬件中断
         * 如果 OF=1，CPU自动执行 INT 4
         */
        into
    }

    printf("[Step 4] Calculation logic returned to main.\n");

    /* 4. 恢复旧中断 */
    setvect(4, old_handler);
    printf("[Step 5] Original INT 4 vector restored.\n");

    getch();
    return 0;
}
```

五、 运行结果与分析

屏幕输出结果：

结果分析：



1. 程序运行到 `add ax, b` 时，寄存器 AX 变为 `0x8000`（即十进制 -32768），此时符号位与操作数符号相反，CPU 内部标志寄存器的 OF 位被置 1。
2. 执行到 `into` 指令时，CPU 检测到 OF=1，随即挂起当前流程，查询中断向量表第 4 项。
3. 屏幕成功输出了被星号包围的 Alert 信息，证明控制流成功跳转到了 `new_overflow_handler` 函数。
4. 中断函数执行完毕后（`IRET`），控制流正确回到了 `main` 函数继续执行后续的打印语句。

六、 实验总结

1. **中断的优越性：** 相比于在每一处运算后都编写 `JO error_label` 这样的跳转逻辑，利用 `INTO` 配合中断服务程序可以实现错误处理与业务逻辑的分离，代码更加简洁且具有通用性。
2. **系统安全性：** 实验过程中深刻体会到了“恢复中断向量”的重要性。如果在程序退出前没有恢复旧的 INT 4，后续运行其他程序如果发生溢出，可能会跳转到已经释放的内存地址，导致 DOSBox 崩溃或死机。
3. **内联汇编威力：** 通过 C 语言内联汇编，既保留了 C 语言构建程序框架的便利性，又利用了汇编语言直接操作 CPU 标志位和特殊指令（INTO）的能力，解决了 C 语言无法直接感知处理器状态（如 OF 标志）的问题。