

作业题目 6：重写溢出中断服务程序

背景：

- 1、了解标志寄存器；
- 2、理解一次计算，硬件会同时根据运算结果置进位/借位标志 CF（无符号数）和溢出标志（有符号数），当 OF 被置 1 表示运算结果错误；
- 3、C 语言对溢出是 UNB 行为，程序会有一定风险；

基本要求：

- 1、C 语言中用内联汇编解决溢出报错；
 - 2、不用 JO 或 JNO 逻辑（上节课课堂作业）；
 - 3、要求重写 INTO（4 号中断）的中断服务程序；
- 目前 INTO 中断服务程序没实际内容，如下图所示：

```
0070:0008 FE3B      ???      [BX+SI]
0070:000A 2C00      SUB      AL,00
0070:000C CF          IRET
```

程序总体思路

1. 需要解决的问题

CPU 执行运算时会同时设置 CF（无符号溢出）和 OF（有符号溢出）标志，C 语言默认忽略溢出，可能导致安全隐患，需要重写 INTO 中断服务程序，在溢出时进行专门处理。

2. 解决方案架构

运算发生 → 检测 OF 标志 → 触发中断 → 执行自定义服务程序
→ 恢复执行

3. 具体实现方法

首先检测溢出。有符号数溢出发生在两个正数相加得到负数，两个负数相加得到正数。

```
int check_signed_overflow(int a, int b) {
    int sum = a + b;
    return ((a > 0 && b > 0 && sum < 0) || (a < 0 && b < 0 && sum > 0));
}
```

然后进行中断服务程序模拟：

```

void simulate_into(int a, int b) {
    if (check_signed_overflow(a, b)) {
        printf("OF=1, Triggering simulated INTO interrupt service routine\n");
        printf("Custom interrupt service routine started...\n");
        printf("Handling overflow situation...\n");
        printf("Interrupt service routine completed\n");
    }
}

```

最后内联汇编集成，执行实际的加法运算，让硬件自动设置 OF 标志，获取运算结果：

```

__asm__(
    "movl %1, %%eax\n"
    "addl %2, %%eax\n"
    "movl %%eax, %0"
    : "=r" (result)
    : "r" (a), "r" (b)
    : "%eax"
);

```

演示案例 1：明显溢出

2147483647 + 1 = -2147483648 // OF=1

演示案例 2：正常情况

100 + 200 = 300 // OF=0

程序运行结果：

```

INTO Interrupt Service Routine Simulation - 64-bit Version

Test 1: Signed overflow case
a = 2147483647, b = 1
Calculation result: -2147483648
OF=1, Triggering simulated INTO interrupt service routine
Custom interrupt service routine started...
Handling overflow situation...
Interrupt service routine completed

Test 2: No overflow case
a = 100, b = 200
Calculation result: 300
OF=0, No interrupt triggered

Program demonstrates:
1. Overflow detection for signed integers
2. Simulation of INTO interrupt behavior
3. Custom interrupt service routine execution
PS C:\Users\要成为老太太以后再死\Desktop\汇编语言> |

```

现在 INTO 部分的反汇编代码如下：

```

00000001400013cf <simulate_into>:
1400013cf: 55                push    %rbp
1400013d0: 48 89 e5          mov     %rsp,%rbp
1400013d3: 48 83 ec 20       sub     $0x20,%rsp
1400013d7: 89 4d 10          mov     %ecx,0x10(%rbp)
1400013da: 89 55 18          mov     %edx,0x18(%rbp)
1400013dd: 8b 55 18          mov     0x18(%rbp),%edx
1400013e0: 8b 45 10          mov     0x10(%rbp),%eax
1400013e3: 89 c1            mov     %eax,%ecx
1400013e5: e8 96 ff ff ff    call    140001380 <check_signed_overflow>
1400013ea: 85 c0            test    %eax,%eax
1400013ec: 74 3e            je      14000142c <simulate_into+0x5d>
1400013ee: 48 8d 05 0b 5c 00 00 lea     0x5c0b(%rip),%rax      # 140007000 <.rdata>
1400013f5: 48 89 c1          mov     %rax,%rcx
1400013f8: e8 ab 47 00 00    call    140005ba8 <puts>
1400013fd: 48 8d 05 3c 5c 00 00 lea     0x5c3c(%rip),%rax      # 140007040 <.rdata+0x40>
140001404: 48 89 c1          mov     %rax,%rcx
140001407: e8 9c 47 00 00    call    140005ba8 <puts>
14000140c: 48 8d 05 5d 5c 00 00 lea     0x5c5d(%rip),%rax      # 140007070 <.rdata+0x70>
140001413: 48 89 c1          mov     %rax,%rcx
140001416: e8 8d 47 00 00    call    140005ba8 <puts>
14000141b: 48 8d 05 6e 5c 00 00 lea     0x5c6e(%rip),%rax      # 140007090 <.rdata+0x90>
140001422: 48 89 c1          mov     %rax,%rcx
140001425: e8 7e 47 00 00    call    140005ba8 <puts>
14000142a: eb 0f            jmp     14000143b <simulate_into+0x6c>
14000142c: 48 8d 05 81 5c 00 00 lea     0x5c81(%rip),%rax      # 1400070b4 <.rdata+0xb4>
140001433: 48 89 c1          mov     %rax,%rcx
140001436: e8 6d 47 00 00    call    140005ba8 <puts>
14000143b: 90                nop
14000143c: 48 83 c4 20       add     $0x20,%rsp
140001440: 5d                pop     %rbp
140001441: c3                ret

```