作业题目 3：求和

基本要求：求 1+2+……+100，并将结果“5050”打印到屏幕。

```
D:\>sum
5050
```

注意和的结果数据表示范围，结果的进制转换等问题。

1）›尝试结果分别放在寄存器中、放在数据段中，放在栈中等不同位置的操作；

2）›用户输入 1~100 内的任何一个数，完成十进制结果输出。（查找 21 号中断的功能表，找到输入数据的功能调用）

3）›用 C 语言实现后察看反汇编代码并加注释；

(1) 结果如图所示：

```
C:\>sum1.exe
Sum (stored in register): 5050

C:\>
```

```
C:\>sum2.exe
Sum (stored in data segment): 5050

C:\>_
```

```
C:\>sum3.exe
Sum (stored in stack): 5050

C:\>_
```

(2) 程序功能说明用户输入处理

首先使用 DOS 21h 中断的 0Ah 功能读取字符串输入。

输入缓冲区结构：

第一个字节：缓冲区大小

第二个字节：实际输入的字符数

后续字节：输入的字符串内容

字符串转数字时，逐个字符处理，将 ASCII 数字转换为数值，使用公式：result = result * 10 + new_digit。

求和计算从 1 到用户输入数字 n 的和，使用公式：sum = 1 + 2 + 3 + ... + n。

最后结果输出显示完整的算式：1+2+...+n = sum，使用数字转字符串算法输出结果。

21h 中断功能表（相关功能）

功能号	描述	输入参数	输出参数
01h	读取单个字符	-	AL = 字符
02h	显示单个字符	DL = 字符	-
09h	显示字符串	DS:DX = 字符串地址	-
0Ah	读取字符串	DS:DX = 输入缓冲区地址	缓冲区填充

结果如图：

```
C:\>sum4.exe
Enter a number n(1-100): 2
The sum of 1+2+...+n is: 3

C:\>sum4.exe
Enter a number n(1-100): 25
The sum of 1+2+...+n is: 325

C:\>sum4.exe
Enter a number n(1-100): 100
The sum of 1+2+...+n is: 5050

C:\>
```

(3) C 语言反汇编

```
c:\MASM>objdump -d -S sum.exe | grep -A 50 "<main>:"
0000000140001380 <main>:
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>

int main()
{
   140001380:   55                      push   %rbp
   140001381:   48 89 e5                mov    %rsp,%rbp
   140001384:   48 83 ec 30             sub    $0x30,%rsp
   140001388:   e8 cf 00 00 00          call   14000145c <__main>
        int n;
        int i=1;
   14000138d:   c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
        int sum = 0;
   140001394:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
        printf("Enter a num:");
   14000139b:   48 8d 05 5e ac 00 00    lea    0xac5e(%rip),%rax        # 14000c000 <.rdata>
   1400013a2:   48 89 c1                mov    %rax,%rcx
   1400013a5:   e8 56 0c 00 00          call   140002000 <__mingw_printf>
        scanf("%d", &n);
   1400013aa:   48 8d 45 f4             lea    -0xc(%rbp),%rax
   1400013ae:   48 8d 0d 58 ac 00 00    lea    0xac58(%rip),%rcx        # 14000c00d <.rdata+0xd>
   1400013b5:   48 89 c2                mov    %rax,%rdx
   1400013b8:   e8 c3 0c 00 00          call   140002080 <__mingw_scanf>
        while (i <= n) {
   1400013bd:   eb 0a                   jmp    1400013c9 <main+0x49>
            sum = sum + i;
   1400013bf:   8b 45 fc                mov    -0x4(%rbp),%eax
   1400013c2:   01 45 f8                add    %eax,-0x8(%rbp)
            i++;
   1400013c5:   83 45 fc 01             addl   $0x1,-0x4(%rbp)
        while (i <= n) {
   1400013c9:   8b 45 f4                mov    -0xc(%rbp),%eax
   1400013cc:   39 45 fc                cmp    %eax,-0x4(%rbp)
   1400013cf:   7e ee                   jle    1400013bf <main+0x3f>
        }
        printf("%d", sum);
   1400013d1:   8b 45 f8                mov    -0x8(%rbp),%eax
   1400013d4:   48 8d 0d 32 ac 00 00    lea    0xac32(%rip),%rcx        # 14000c00d <.rdata+0xd>
   1400013db:   89 c2                   mov    %eax,%edx
   1400013dd:   e8 1e 0c 00 00          call   140002000 <__mingw_printf>

        return 0;
   1400013e2:   b8 00 00 00 00          mov    $0x0,%eax
   1400013e7:   48 83 c4 30             add    $0x30,%rsp
   1400013eb:   5d                      pop    %rbp
   1400013ec:   c3                      ret
   1400013ed:   90                      nop
   1400013ee:   90                      nop
   1400013ef:   90                      nop
```

反汇编注释

0000000140001380 <main>:

#define _CRT_SECURE_NO_WARNINGS

#include<stdio.h>

int main()

{

;;; 函数序言 建立栈帧

```
140001380: 55                    push   %rbp               ; 保存旧的基址指针
140001381: 48 89 e5              mov    %rsp,%rbp          ; 设置新的基址指针
140001384: 48 83 ec 30           sub    $0x30,%rsp         ; 在栈上分配 48 字节空间给局部变量
140001388: e8 cf 00 00 00        call   14000145c <__main>  ; 调用 MinGW 的初始化代码

    ;;; 局部变量初始化
    int n;
    int i=1;
14000138d: c7 45 fc 01 00 00 00  movl   $0x1,-0x4(%rbp)     ; i = 1 (存储在栈位置 -0x4)
    int sum = 0;
140001394: c7 45 f8 00 00 00 00  movl   $0x0,-0x8(%rbp)     ; sum = 0 (存储在栈位置 -0x8)

    ;;; 调用 printf 输出提示信息
    printf("Enter a num:");
14000139b: 48 8d 05 5e ac 00 00  lea    0xac5e(%rip),%rax  ; 加载字符串地址 "Enter a num:"
1400013a2: 48 89 c1              mov    %rax,%rcx          ; 第一个参数放入 rcx (Windows x64 调用约定)
1400013a5: e8 56 0c 00 00        call   140002000 <__mingw_printf>  ; 调用 printf

    ;;; 调用 scanf 读取输入
    scanf("%d", &n);
1400013aa: 48 8d 45 f4           lea    -0xc(%rbp),%rax    ; 加载 n 的地址 (栈位置 -0xc)
1400013ae: 48 8d 0d 58 ac 00 00  lea    0xac58(%rip),%rcx  ; 加载格式字符串 "%d" 的地址
1400013b5: 48 89 c2              mov    %rax,%rdx          ; 第二个参数放入 rdx (n 的地址)
1400013b8: e8 c3 0c 00 00        call   140002080 <__mingw_scanf>  ; 调用 scanf

    ;;; while 循环开始
    while (i <= n) {
1400013bd: eb 0a                 jmp    1400013c9 <main+0x49>  ; 跳转到循环条件检查

    ;;; 循环体
    sum = sum + i;
1400013bf: 8b 45 fc              mov    -0x4(%rbp),%eax    ; 将 i 的值加载到 eax
1400013c2: 01 45 f8              add    %eax,-0x8(%rbp)    ; sum += i
```

```
    i++;
    1400013c5: 83 45 fc 01          addl    $0x1,-0x4(%rbp)         ; i = i + 1

    ;;; 循环条件检查
    while (i <= n) {
    1400013c9: 8b 45 f4             mov     -0xc(%rbp),%eax         ; 将 n 的值加载到 eax
    1400013cc: 39 45 fc             cmp     %eax,-0x4(%rbp)         ; 比较 i 和 n
    1400013cf: 7e ee                jle     1400013bf <main+0x3f> ; 如果 i <= n，跳回循环体开始

    ;;; 循环结束，输出结果
    printf("%d", sum);
    1400013d1: 8b 45 f8             mov     -0x8(%rbp),%eax         ; 将 sum 的值加载到 eax
    1400013d4: 48 8d 0d 32 ac 00 00 lea     0xac32(%rip),%rcx       ; 加载格式字符串 "%d" 的地址
    1400013db: 89 c2                mov     %eax,%edx               ; 第二个参数放入 edx (sum 的值)
    1400013dd: e8 1e 0c 00 00       call    140002000 <__mingw_printf>  ; 调用 printf

    ;;; 函数收尾 (Function Epilogue) - 清理栈帧并返回
    return 0;
    1400013e2: b8 00 00 00 00       mov     $0x0,%eax               ; 返回值 0
    1400013e7: 48 83 c4 30          add     $0x30,%rsp              ; 释放栈空间
    1400013eb: 5d                   pop     %rbp                    ; 恢复旧的基址指针
    1400013ec: c3                   ret                             ; 返回

    ;;; 以下是指令对齐用的 nop 指令（无操作）
    1400013ed: 90                   nop
    1400013ee: 90                   nop
    1400013ef: 90                   nop
```