汇编语言学习笔记：求和程序设计

一、实验目的

1. 掌握汇编语言基本程序设计方法
2. 理解不同存储位置（寄存器、数据段、栈）的数据操作
3. 学习DOS系统功能调用的使用方法
4. 掌握数字与字符串的转换算法
5. 通过C语言反汇编理解高级语言与汇编语言的对应关系

二、实验环境

- **操作系统**: Windows 11
- **汇编环境**: DOSBox + MASM 5.0
- **调试工具**: DEBUG.EXE
- **C语言环境**: GCC编译器

三、实验内容与实现

3.1 基础求和：1+2+...+100

算法设计

最初采用“从100递减到1”的LOOP方案，但在后续实验中发现递减循环在引入用户输入后不够直观。 最终定稿采用“从1递增到n”的思路：BX从1开始累加到用户输入的n，AX存放累加结果，CX可用来记录目标上限。

```
mov ax, 0           ; AX存放和
mov bx, 1           ; 当前被加数
mov cx, n           ; 用户输入
sum_loop:
    add ax, bx
    inc bx
    cmp bx, cx
    jle sum_loop
```

关键技术点

- **循环控制**: 通过CMP/JLE组合实现可读性更高的上升序列
- **结果范围**: 5050 < 65535，使用16位寄存器安全
- **代码鲁棒性**: 递增写法在处理1~n求和时更贴合自然语言描述

3.2 不同存储位置的实现比较

3.2.1 寄存器存储

```
; 结果保存在AX寄存器中
mov ax, 0
add ax, cx
```

特点:

- 访问速度最快
- 寄存器数量有限
- 临时存储，程序结束即丢失

3.2.2 数据段存储

```
data segment
    result dw 0     ; 在数据段定义变量
data ends

mov result, ax      ; 将结果存入数据段
```

特点:

- 永久存储
- 需要设置DS寄存器
- 占用内存空间

3.2.3 栈存储

```
push ax             ; 结果压栈
pop ax              ; 结果出栈
```

特点:

- 临时存储，LIFO结构
- 需要平衡push/pop操作
- 适合函数参数传递

3.3 用户输入功能的实现（sum_input.asm）

3.3.1 DOS 21H中断输入功能

```
buffer db 6, ?, 6 dup(0)  ; buffer[0]=最大长度, buffer[1]=实际长度
mov dx, offset buffer
mov ah, 0ah               ; DOS 缓冲输入
int 21h
```

缓冲区结构:

```
buffer db maxLen, actualLen, data0, data1, ...
```

关键点在于必须读取 `buffer+1` 得到实际输入长度，而不是等待读入的 0Dh。

3.3.2 字符串到数字的转换算法

最终版本对子程序全面重写：

```
string_to_number proc
    mov cl, [buffer + 1]       ; 实际字符数
    mov si, offset buffer + 2
    mov ax, 0
convert_loop:
    mov bl, [si]
    cmp bl, 0dh
    je convert_exit
    cmp bl, '0'
    jb convert_exit
    cmp bl, '9'
    ja convert_exit
    sub bl, '0'
    mov bh, 0
    push bx
    mov bx, 10
    mul bx                     ; DX:AX = AX * 10
    pop bx
    add ax, bx
    inc si
    loop convert_loop
convert_exit:
    ret
string_to_number endp
```

改进要点：

- 使用 `buffer+1` 的实际长度控制 loop，解决旧版本依赖 0Dh 导致的残留字符问题。
- `mul` 前后显式管控 DX，避免 `DX:AX` 被除法误用。
- 子程序内部 `push`/`pop` 保护所有使用过的寄存器，防止回到主流程后寄存器被污染。

3.4 数字显示的实现

3.4.1 数字到字符串的转换

使用"除10取余"算法：

```
print_number proc
    mov bx, 10
    mov cx, 0
digit_loop:
    mov dx, 0
    div bx              ; AX/10, 余数在DX
    push dx             ; 保存余数
    inc cx              ; 位数计数
    cmp ax, 0
    jne digit_loop
```

3.4.2 字符串显示

```
display_loop:
    pop dx
    add dl, '0'         ; 数字转ASCII
    mov ah, 02h         ; 字符显示功能
    int 21h
    loop display_loop
```

四、C语言实现与反汇编分析

4.1 C语言源代码

```c
#include <stdio.h>
int main() {
    int sum = 0;
    for (int i = 1; i <= 100; i++) {
        sum += i;
    }
    printf("Sum: %d\n", sum);
    return 0;
}
```

4.2 反汇编代码分析

```
_main:
    push    rbp
    mov     rbp, rsp
    sub     rsp, 48         ; 栈空间分配

    mov     DWORD PTR [rbp-4], 0    ; sum = 0
    mov     DWORD PTR [rbp-8], 1    ; i = 1
    jmp     L2
```

```
L3:
    mov     eax, DWORD PTR [rbp-8]  ; eax = i
    add     DWORD PTR [rbp-4], eax  ; sum += i
    add     DWORD PTR [rbp-8], 1    ; i++

L2:
    cmp     DWORD PTR [rbp-8], 100  ; i <= 100
    jle     L3

    mov     eax, DWORD PTR [rbp-4]  ; 参数准备
    mov     edx, eax
    lea     rcx, LC0[rip]
    call    printf
```

4.3 对比分析

特性	汇编语言	C语言
循环控制	LOOP指令或条件跳转	for语句
变量存储	显式指定存储位置	编译器自动分配
函数调用	INT 21H系统调用	标准库函数
类型转换	手动ASCII转换	自动类型转换

五、遇到的问题与解决方案

5.1 输入转换导致结果异常

问题: 旧版本依赖回车符检测结束，DX 未清零导致 `div` 取值错误，输入“2”会输出 2307 等异常。 **解决方案**: 全面重写 `string_to_number`，严格按照 DOS 缓冲区格式取实际长度，所有子程序调用前手动清理 DX。

5.2 求和循环与寄存器冲突

问题: 循环计数寄存器与子程序共享，push/pop 不完整导致CX被破坏，出现死循环或结果偏差。 **解决方案**: 输入转换与打印子程序全部保存、恢复所用寄存器；主程采用 BX 自增方式避免与 LOOP 冲突。

5.3 数字显示

问题: 直接输出AX得到字符；当AX=0时旧算法会陷入空输出。 **解决方案**: `print_number` 中增加 `AX=0` 特判，通用情况下使用“除10取余 + 栈倒序输出”并保护寄存器。

六、实验总结

6.1 主要收获

1. **掌握了汇编语言的基本编程方法**
    - 理解了寄存器、内存、栈的操作
    - 掌握了循环和条件跳转的实现

2. **深入理解了系统调用机制**
    - 学会了使用DOS 21H中断的各种功能
    - 理解了输入输出的底层实现
3. **提升了调试和问题解决能力**
    - 使用DEBUG工具进行程序调试
    - 学会了分析程序执行流程
4. **理解了高级语言与汇编语言的对应关系**
    - 通过反汇编理解了C语言的底层实现
    - 认识了编译器的优化策略

6.2 不足与改进

1. **错误处理不足**: 仍缺乏对范围(>100)与非数字输入的友好提示，可进一步完善。
2. **代码复用性**: `string_to_number` 和 `print_number` 可整理成公用库，方便其他实验复用。
3. **界面体验**: 可考虑增加再次输入或清屏逻辑，使DOS窗口交互更自然。

6.3 未来学习方向

1. 学习更复杂的数据结构和算法
2. 研究保护模式编程
3. 探索操作系统底层机制
4. 学习现代x86-64架构汇编

七、源代码文件清单

- `sum_reg.asm` - 寄存器版本
- `sum_data.asm` - 数据段版本
- `sum_stack.asm` - 栈版本
- `sum_input.asm` - 用户输入版本
- `sum.c` - C语言版本