Assignment 2 SUM

1. **计算1-100累加和**

1.1 汇编代码

.MODEL SMALL

.STACK 100h

.DATA

    SUM\_RESULT DW ?        ; 定义一个16位变量来存储结果

.CODE

PRINT\_DEC\_NUM PROC

    PUSH CX              ; 保存 CX 和 DX 寄存器

    PUSH DX

    XOR CX, CX           ; CX = 0，用作计数器，计算压入堆栈的数字个数

    MOV BX, 10           ; 除数 BCD 10

    ; *--- 循环：重复将 AX 除以 10，将余数（即数字）压入堆栈 ---*

DIV\_LOOP:

    XOR DX, DX           ; 每次除法前必须清零 DX

    DIV BX               ; AX = AX / 10, DX = AX % 10 (余数)

    PUSH DX              ; 将余数（一个十进制数字）压入堆栈

    INC CX               ; 计数器加 1

    CMP AX, 0            ; 商是否为 0？

    JNE DIV\_LOOP         ; 如果不为 0，继续循环

    ; *--- 循环：从堆栈中弹出数字并打印 ---*

PRINT\_LOOP:

    POP DX               ; 弹出数字

    ADD DL, '0'          ; 将数字 (0-9) 转换为 ASCII 字符 ('0'-'9')

    MOV AH, 2            ; DOS 功能：显示字符

    INT 21H              ; 调用 DOS 中断

    LOOP PRINT\_LOOP      ; 循环 CX 次

    POP DX               ; 恢复 DX 和 CX 寄存器

    POP CX

    RET

PRINT\_DEC\_NUM ENDP

START:

    MOV AX, @DATA        ; 初始化数据段寄存器

    MOV DS, AX

    ; *--- 计算 1+...+100 ---*

    MOV CX, 100          ; CX = 100 (循环计数器)

    XOR AX, AX           ; AX = 0 (累加器)

SUM\_LOOP:

    ADD AX, CX           ; AX = AX + CX

    LOOP SUM\_LOOP        ; CX 减 1，如果 CX != 0 则跳转到 SUM\_LOOP

    MOV SUM\_RESULT, AX   ; 将最终结果 (5050) 存入变量

    ; *--- 打印结果 ---*

    ; AX 中已经有我们的结果 (5050)

    CALL PRINT\_DEC\_NUM   ; 调用打印过程

    MOV AH, 4CH          ; DOS 功能：退出程序

    INT 21H              ; 调用 DOS 中断

END START

* 1. 汇编结果

1. **计算1-N的累加和**

2.1 汇编代码

.MODEL SMALL

.STACK 100h

.DATA

    PROMPT\_N  DB 'Please Enter A Number Between 1 And 100: $'

    PROMPT\_SUM DB 0DH, 0AH, 'The sum is: $'

    NEWLINE    DB 0DH, 0AH, '$'

    N\_VALUE   DW ?            ; 存储用户输入的 N

    SUM\_RESULT DW ?          ; 存储计算结果

.CODE

PRINT\_DEC\_NUM PROC

    PUSH CX                    ; 保存 CX 和 DX 寄存器

    PUSH DX

    PUSH BX

    XOR CX, CX

    MOV BX, 10

; *--- 循环：重复将 AX 除以 10，将余数（即数字）压入堆栈 ---*

DIV\_LOOP:

    XOR DX, DX               ; 每次除法前必须清零 DX

    DIV BX                   ; AX = AX / 10, DX = AX % 10 (余数)

    PUSH DX                  ; 将余数（一个十进制数字）压入堆栈

    INC CX                   ; 计数器加 1

    CMP AX, 0                ; 商是否为 0？

    JNE DIV\_LOOP            ; 如果不为 0，继续循环

PRINT\_LOOP:

    POP DX               ; 弹出数字

    ADD DL, '0'          ; 将数字(0-9)转换为ASCII字符('0'-'9')

    MOV AH, 2            ; DOS 功能：显示字符

    INT 21H              ; 调用 DOS 中断

    LOOP PRINT\_LOOP      ; 循环 CX 次

    POP BX

    POP DX

    POP CX

    RET

PRINT\_DEC\_NUM ENDP

READ\_DEC\_NUM PROC

    PUSH AX

    PUSH CX

    PUSH DX

    XOR BX, BX           ; BX = 0，用于累积结果

    MOV CX, 10           ; CX = 10，用于乘法

READ\_CHAR\_LOOP:

    MOV AH, 1            ; DOS 功能：读取一个字符 (带回显)

    INT 21H

    CMP AL, 0DH          ; 是回车键吗？

    JE READ\_DONE         ; 是，则读取结束

    CMP AL, '0'          ; < '0'?

    JL READ\_CHAR\_LOOP    ; 不是数字，忽略

    CMP AL, '9'          ; > '9'?

    JG READ\_CHAR\_LOOP    ; 不是数字，忽略

    ; 是数字字符

    SUB AL, '0'          ; AL = ASCII 转换为 数字

    XOR AH, AH           ; AH = 0, AX = AL (数字值)

    PUSH AX              ; 暂时保存这个数字

    MOV AX, BX           ; AX = 当前累积的值

    MUL CX               ; AX = AX \* 10 (DX:AX)

    MOV BX, AX           ; BX = BX \* 10

    POP AX               ; 恢复刚才输入的数字

    ADD BX, AX           ; BX = (BX \* 10) + 新数字

    JMP READ\_CHAR\_LOOP

READ\_DONE:

    POP DX

    POP CX

    POP AX

    RET

READ\_DEC\_NUM ENDP

START:

    MOV AX, @DATA        ; 初始化数据段寄存器

    MOV DS, AX

    ; *--- 1. 提示用户输入 ---*

    MOV AH, 9

    LEA DX, PROMPT\_N

    INT 21H

    ; *--- 2. 读取用户输入的 N ---*

    CALL READ\_DEC\_NUM    ; 调用读取过程，结果在 BX 中

    MOV N\_VALUE, BX      ; 保存 N 的值

    ; *--- 3. 计算 1+...+N ---*

    MOV CX, BX           ; CX = N (循环计数器)

    XOR AX, AX           ; AX = 0 (累加器)

    CMP CX, 0            ; 检查 N 是否为 0

    JE CALC\_DONE         ; 如果 N=0，总和为 0，跳过计算

SUM\_LOOP:

    ADD AX, CX           ; AX = AX + CX

    LOOP SUM\_LOOP

CALC\_DONE:

    MOV SUM\_RESULT, AX   ; 保存总和

    ; *--- 4. 打印 "The sum is: " ---*

    MOV AH, 9

    LEA DX, PROMPT\_SUM

    INT 21H

    ; *--- 5. 打印总和 ---*

    MOV AX, SUM\_RESULT   ; 将总和加载到 AX

    CALL PRINT\_DEC\_NUM   ; 调用打印过程

    ; *--- 打印换行 ---*

    MOV AH, 9

    LEA DX, NEWLINE

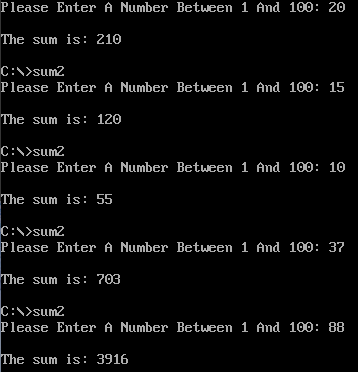
    INT 21H

    ; *--- 6. 退出程序 ---*

    MOV AH, 4CH          ; DOS 功能：退出程序

    INT 21H              ; 调用 DOS 中断

END START

* 1. 汇编结果

1. **1-100累加和反汇编注释**

3.1 C代码

#include <stdio.h>

void main() {

    int i;

    int sum = 0;

    for (i = 1; i <= 100; i++) {

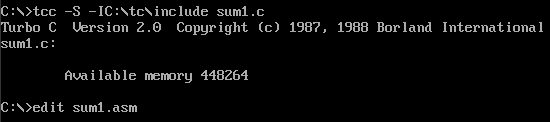
        sum = sum + i;

    }

    printf("Sum is: %d\n", sum);

}

* 1. 反汇编

在Dosbox中生成汇编文件sum1.asm后使用edit sum1.asm打开并注释

\_main   proc    near

        push    si              ; 保存 si 寄存器的原始值

        push    di              ; 保存 di 寄存器的原始值

        xor     di,di           ; 'di = 0'。这是将寄存器清零的最快指令 (di ^ di = 0)

        mov     si,1            ; 'si = 1' (i = 1)

        jmp     short @5        ; 跳转到循环条件检查 (cmp si, 100)

@4:     ; <*--- 循环体标签*

        mov     ax,di           ; 'ax = di' (ax = sum)

        add     ax,si           ; 'ax = ax + si' (ax = sum + i)

        mov     di,ax           ; 'di = ax' (sum = ax)，'sum' (在di) 更新完毕

@3:     ; <*--- 循环增量标签*

        inc     si              ; 'si = si + 1' (i++)

@5:     ; <*--- 循环条件检查标签*

        cmp     si,100          ; 比较 'si' (i) 和 100

        jle     @4              ; "Jump if Less or Equal" (如果 i <= 100，跳回 @4 执行循环体)

@2:     ; <*--- 循环结束，准备打印*

        push    di              ; 将 'di' (sum 的值, 5050) 压入堆栈

        mov     ax,offset DGROUP:s@ ; 'ax = 字符串 s@ 的地址'

        push    ax              ; 将字符串地址压入堆栈

        call    near ptr \_printf

        pop     cx              ; 清理 2 字节 (ax)

        pop     cx              ; 清理 2 字节 (di)

@1:     ; <*--- 函数即将结束*

        pop     di              ; 恢复 di 寄存器的原始值

        pop     si              ; 恢复 si 寄存器的原始值

        ret                     ; 返回 (return)

\_main   endp                    ; \_main 过程(procedure)结束

\_TEXT   ends                    ; 代码段结束

; *--- 数据段定义 ---*

\_DATA   segment word public 'DATA'

s@      label   byte

        ; 这是 printf 的格式字符串: "%d\n"

        db      37              ; 37 是 '%' 的 ASCII 码

        db      100             ; 100 是 'd' 的 ASCII 码

        db      10              ; 10 是 '\n' (换行符) 的 ASCII 码

        db      0               ; 0 是 '\0' (字符串空终止符)

\_DATA   ends

1. **1-n累加和反汇编注释**

4.1 C代码

#include <stdio.h>

void main() {

    int i;

    int n;

    int sum = 0;

    printf("Please Enter A Number Between 1 And 100 ");

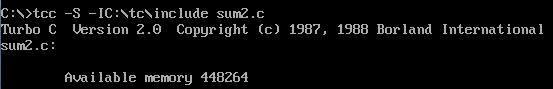
    scanf("%d", &n);

    for (i = 1; i <= n; i++)

        sum = sum + i

    printf("The Sum is: %d\n", sum);

}

* 1. 在Dosbox中生成汇编文件sum2.asm后使用edit sum2.asm打开并注释

\_main   proc    near

        push    bp              ; 保存调用者的基址指针

        mov     bp,sp           ; 设置当前函数的基址指针 (栈帧)

        sub     sp,2            ; 在堆栈上分配 2 字节空间，用于存储 'n'

                               'n' 将通过 [bp-2] 访问

        push    si              ; 保存 'si' (因为本函数要用它)

        push    di              ; 保存 'di' (因为本函数要用它)

;       ?debug  L 6             ; 调试信息：C 代码第 6 行 (int sum = 0;)

        xor     di,di           ; 'di = 0'编译器选择'di'寄存器来存储'sum'

;       ?debug  L 8             ; 调试信息：C 代码第 8 行 (printf(...);)

        ; 对应 C 代码: printf("Please Enter A Number...");

        mov     ax,offset DGROUP:s@ ; 1. 'ax = ' s@ 字符串的地址

        push    ax              ; 将字符串地址压栈 (printf 的参数)

        call    near ptr \_printf ; 调用 C 库函数

        pop     cx              ; 堆栈清理，弹出 2 字节 (ax) 到 cx 中

        lea     ax,word ptr [bp-2] ; 'ax = '[bp-2]的有效地址，就是 C 的 &n

        push    ax              ; 将'n'的地址(&n)压栈(scanf 的第 2 个参数)

        mov     ax,offset DGROUP:s@+41;'ax='s@+41("%d")格式字符串的地址

        push    ax              ; 将格式字符串地址压栈(scanf 的第 1 个参数)

        call    near ptr \_scanf; 调用 C 库函数。输入的值被存入 [bp-2]

        pop     cx              ; 弹出 2 字节 (ax)

        pop     cx              ; 弹出 2 字节 (lea ax,...)

        mov     si,1           ; 'si = 1'，编译器选择'si'寄存器来存储 'i'

        jmp     short @5        ; 跳转到循环条件检查

@4:     ; 循环体标签

        mov     ax,di           ; 'ax = di' (ax = sum)

        add     ax,si           ; 'ax = ax + si' (ax = sum + i)

        mov     di,ax           ; 'di = ax'(sum = ax)，'sum'(在di)更新完毕

@3:     ; 循环增量标签 (... i++)

        inc     si               ; 'si = si + 1' (i++)

@5:     ; 循环条件检查标签 (... i <= n;)

        cmp     si,word ptr [bp-2]; 比较 'si' (i) 和 存储在堆栈上的 'n'

jle     @4              ;如果 i <= n，跳回 @4

@2:     ; 循环结束

        push    di              ; 将'di'(最终的sum)压栈

        mov     ax,offset DGROUP:s@+44; 'ax='s@+44("The Sum is...") 地址

        push    ax              ; 将字符串地址压栈 (printf 的第 1 个参数)

        call    near ptr \_printf; 调用 C 库函数

        pop     cx              ; 弹出 2 字节 (ax)

        pop     cx              ; 弹出 2 字节 (di)

@1:

        pop     di              ; 恢复 'di' 寄存器的原始值

        pop     si              ; 恢复 'si' 寄存器的原始值

        mov     sp,bp           ; 释放局部变量 'n' (恢复堆栈指针)

        pop     bp              ; 恢复调用者的基址指针

        ret                      ; 返回

\_main   endp                     ; 过程 \_main 结束

\_TEXT   ends

\_DATA   segment word public 'DATA'

s@      label   byte

        ; C 字符串: "Please Enter A Number Between 1 And 100 \0"

        db      80, 108, 101, 97, 115, 101, 32, 69, 110, 116, 101, 114, 32, 65, 32

        db      78, 117, 109, 98, 101, 114, 32, 66, 101, 116, 119, 101, 101, 110

        db      32, 49, 32, 65, 110, 100, 32, 49, 48, 48, 32, 0

        ; C 字符串: "%d\0" (这是 s@+41)

        db      37, 100, 0

        ; C 字符串: "The Sum is: %d\n\0" (这是 s@+44)

        db      84, 104, 101, 32, 83, 117, 109, 32, 105, 115, 58, 32

        db      37, 100, 10, 0

1. **总结**

寄存器的优势在于对诸如加法这样的简单操作，寄存器的速度非常快。因此，我选择将中间计算结果暂时存放在AX寄存器中，这样在每次循环中都能够快速地更新总和值。在计算循环结束之后，把 AX 寄存器中的值复制一份，保存到 SUM\_RESULT 这个变量里，SUM\_RESULT 变量就在数据段中。PRINT\_DEC\_NUM 程序利用了堆栈计算得到余数0，5，0，5，执行PUSH DX，把它们分别压入堆栈，输出时执行 POP DX分别取出5，0，5，0。

通过这个程序的学习，我清晰地理解了如何在寄存器、数据段和栈中分别存储和处理数据。在寄存器中进行高效运算，在数据段中存储较大或持久化的数据，并通过栈在程序运行中保存和恢复状态。每种方式都有其独特的优势，合理运用这些存储机制可以极大提高汇编程序的效率和灵活性。

在 DOS 系统中，中断 21h 是处理输入和输出操作操作的关键工具。通过中断 21h 实现输出也让我学到了如何精确控制输出字符到屏幕，通过使用中断进行字符输入，我学习到了如何通过设置不同的功能号来实现各种操作。