

一. 求和

1. 结果存在寄存器中

```
.MODEL SMALL
.STACK 100H
.DATA
    numStr DB 6 DUP(0)    ; 存储数字字符串

.CODE
MAIN PROC
    MOV AX, @DATA        ; 初始化数据段寄存器
    MOV DS, AX            ; 设置DS指向数据段

    MOV CX, 100           ; 设置循环次数为100
    MOV AX, 0             ; 初始化AX寄存器为0

SumToRegister:
    ADD AX, CX            ; 将CX寄存器的值加到AX寄存器
    LOOP SumToRegister    ; 循环CX次，CX减1直到0

    CALL ConvertToStr      ; 结果转化为字符串

    ; 输出结果
    MOV DX, OFFSET numStr ; 将结果字符串的地址放入DX
    MOV AH, 09H           ; DOS 功能：输出字符串
    INT 21H               ; 调用DOS中断

    MOV AX, 4C00H         ; 正常退出
    INT 21H               ; 中断退出
MAIN ENDP

ConvertToStr PROC
    MOV BX, 10             ; 基数为 10
    MOV CX, 0              ; 字符计数器

convert_loop:
    XOR DX, DX             ; 清零 DX
    DIV BX                 ; AX / 10, 商在 AX, 余数在 DX
    PUSH DX                ; 保存余数
    INC CX                 ; 计数
    TEST AX, AX            ; 检查 AX 是否为 0
    JNZ convert_loop       ; 如果不为 0, 继续循环

    MOV DI, 0              ; 初始化DI 用于字符索引

    ; 从栈中弹出余数并转换为字符
print_loop:
    POP DX                 ; 弹出余数
    ADD DL, '0'            ; 转换为字符
    MOV numStr[DI], DL     ; 存储字符到字符串
    INC DI                 ; 增加字符索引
    DEC CX                 ; 减少计数
    TEST CX, CX            ; 检查 CX 是否为 0
    JNZ print_loop         ; 如果CX不为0, 说明还有字符, 继续

    MOV numStr[DI], '$'    ; 添加字符串结束符
    RET
ConvertToStr ENDP

END MAIN
```

这段代码首先初始化数据段寄存器，使用 CX 寄存器控制循环，从 100 递减至 1，并将每次循环的值累加到 AX 寄存器中。累加完成后，通过调用 ConvertToStr 函数，将计算结果转换为字符串。该函数利用除法操作将数字按位分解，并通过栈操作逐位存入字符串。最后，程序调用 DOS 中断 21H 的功能号 09H，将存储的字符串输出到屏幕。程序执行完毕后，使用 DOS 中断正常退出。

2. 结果存在内存中

这种方法实现过程与第一种方法几乎相同，只是会把 AX 寄存器中累加的结果存在内存中。

3. 结果存在栈中

这种方法实现过程与第一种方法也几乎相同，只是会把 AX 寄存器中累加的结果压入栈中保存。

二 . 求和（输入数字）

```
.MODEL SMALL
.STACK 100H
.DATA
    errMsg DB 'Error: Invalid input! Please enter a number.$'
    numStr DB 6 DUP(0)    ; 存储数字字符串
.CODE
MAIN PROC
    MOV AX, @DATA        ; 初始化数据段寄存器
    MOV DS, AX           ; 设置DS指向数据段

    CALL INPUT            ; 调用输入过程，读取用户输入的数字
    MOV AX, BX
    CALL SUM              ;
    CALL OUTPUT           ; 调用输出过程，输出结果

    MOV AH, 4CH           ; 正常退出
    INT 21H              ; 中断退出

INPUT PROC
    MOV BL, 0 ; 当前输入位之前的结果
    MOV CL, 10 ; 乘数10
L:
    MOV AH, 1; 输入一个字符
    INT 21H

    CMP AL, 0Dh        ; 判断是否为回车键
    JE OVER            ; 如果是回车键，跳转到 OVER

    SUB AL, 48
    MOV DL, AL
    MOV DH, 0; dl即dx, 当前位
    MOV AL, BL
    MUL CL ; a1*c1->ax
    ADD AX, DX
    MOV BX, AX ; 结果保存在BX中
    JMP L

OVER:
    RET
INPUT ENDP

SUM PROC
    MOV CX, AX          ; 设置循环次数为输入的数字
    MOV AX, 0           ; 初始化AX寄存器为0
```

```

S:
    ADD AX, CX          ; 将CX寄存器的值加到AX寄存器
    LOOP S              ; 循环CX次, CX减1直到0

    RET
SUM ENDP

OUTPUT PROC
    MOV BX, 10          ; 设置除数为10, 用于十进制输出
    MOV CX, 0

L1:
    XOR DX, DX          ; 清零DX以便进行除法
    DIV BX
    PUSH DX             ; AX / 10, 商在AX, 余数在DX
    INC CX              ; 输出字符计数加1
    TEST AX, AX         ; 检查商是否为0
    JNZ L1              ; 如果AX > 0, 继续循环

    MOV DI, 0

L2:
    POP DX
    ADD DL, '0'         ; 将数字转换为ASCII字符 (加上48)
    MOV numStr[DI], DL  ; 存储字符到字符串
    INC DI
    DEC CX              ; 输出字符计数减1
    TEST CX, CX         ; 检查是否所有字符已输出
    JNZ L2              ; 如果还有字符未输出, 继续循环

    MOV numStr[DI], '$' ; 添加字符串结束符
    MOV DX, OFFSET numStr ; 将结果字符串的地址放入DX
    MOV AH, 09H         ; DOS 功能: 输出字符串
    INT 21H             ; 调用DOS中断

    RET
OUTPUT ENDP

MAIN ENDP
END MAIN

```

程序首先初始化数据段寄存器，并通过调用 INPUT 过程来获取用户输入的数字。在 INPUT 过程中，程序逐个读取字符，直到检测到回车键（Enter），将输入的每个字符转换为相应的数字，并根据当前位置累加到结果中，最终将结果保存在 BX 寄存器中。然后回调用 SUM 过程进行求和，并把结果存在 AX 寄存器中。最后程序调用 OUTPUT 过程输出结果。在 OUTPUT 过程中，程序使用除法将数字逐位转换为字符形式，通过推入栈来保

存每位数字，然后再逐位弹出并转换为字符串，最终通过 DOS 中断将字符逐个输出到控制台。程序最后通过调用 DOS 中断正常退出。

三． 求和（C 语言实现）

```
#include <stdio.h>

int main() {
    int result = 0;
    for (int i = 1; i <= 100; i++)
    {
        result += i;
    }
    printf("%d", result);
    return 0; // 程序正常退出
}
```

反汇编结果：

```
00007FF77F481890  push        rbp
```

;将 rbp（基址指针）压入栈中，保存调用者的栈帧基址

```
00007FF77F481892  push        rdi
```

;将 rdi 寄存器压入栈中，保存它的值（一般用于函数调用）

```
00007FF77F481893  sub         rsp,128h
```

;将栈指针 rsp 减少 128h（即 304 字节），为局部变量分配栈空间

```
00007FF77F48189A  lea         rbp,[rsp+20h]
```

;将 rbp 设置为 rsp+20h，建立栈帧的基址

```
00007FF77F48189F  lea         rcx,[_0FE406C9_print_sum@c (07FF77F492008h)]
```

```
00007FF77F4818A6  call        __CheckForDebuggerJustMyCode (07FF77F48136Bh)
```

```
00007FF77F4818AB  nop
```

```
00007FF77F4818AC  mov         dword ptr [result],0
```

;将局部变量 result 初始化为 0

```
00007FF77F4818B3  mov         dword ptr [rbp+24h],1
```

; 将局部变量 i 初始化为 1

```
00007FF77F4818BA  jmp         main+34h (07FF77F4818C4h)
```

; 跳转到循环条件判断的位置

00007FF77F4818BC mov eax,dword ptr [rbp+24h]

；将变量 i 的值加载到 eax 寄存器中

00007FF77F4818BF inc eax

；将 eax 的值加 1

00007FF77F4818C1 mov dword ptr [rbp+24h],eax

；更新变量 i 的值

00007FF77F4818C4 cmp dword ptr [rbp+24h],64h

；比较 i 的值是否大于 100

00007FF77F4818C8 jg main+49h (07FF77F4818D9h)

；如果 i > 100，则跳出循环

00007FF77F4818CA mov eax,dword ptr [rbp+24h]

；将 i 的值加载到 eax 寄存器中

00007FF77F4818CD mov ecx,dword ptr [result]

；将 result 的值加载到 ecx 寄存器中

00007FF77F4818D0 add ecx,eax

；将 i 的值加到 result 中

00007FF77F4818D2 mov eax,ecx

；更新 eax 为新的 result 值

00007FF77F4818D4 mov dword ptr [result],eax

；将 result 的值保存回内存

00007FF77F4818D7 jmp main+2Ch (07FF77F4818BCh)

；跳回循环起始处

00007FF77F4818D9 mov edx,dword ptr [result]

；将 result 的值加载到 edx 寄存器中

00007FF77F4818DC lea rcx,[string "%d" (07FF77F48AC24h)]

；加载格式化字符串到 rcx 中

00007FF77F4818E3 call printf (07FF77F481195h)

；调用 printf 函数打印 result 的值

```
00007FF77F4818E8  nop
```

```
00007FF77F4818E9  xor          eax,eax
```

；将 eax 设置为 0，表示返回值为 0

```
00007FF77F4818EB  lea          rsp,[rbp+108h]
```

；恢复栈指针 rsp 的值

```
00007FF77F4818F2  pop          rdi
```

；恢复 rdi 寄存器的值

```
00007FF77F4818F3  pop          rbp
```

；恢复之前的栈帧基址

```
00007FF77F4818F4  ret
```

；返回，结束 main 函数

程序开始时，通过压入基址指针（rbp）和 rdi 寄存器的值来保存当前栈帧的状态，并为局部变量分配栈空间。

接着，局部变量 result 被初始化为 0，而循环变量 i 则初始化为 1。

然后程序进入一个循环，通过将 i 的值加 1 并与 100 进行比较来判断是否继续循环。在循环内部，i 的值被累加到 result 中，直至 i 的值超过 100，循环结束。此时，程序将累加结果加载到 edx 寄存器，并调用 printf 函数打印输出。

最后，程序通过设置返回值为 0 来标识正常退出，并恢复栈指针和寄存器的状态，完成主函数的返回。

四． 总结

1. 直接用汇编语言完成的程序，变量的管理大多是通过寄存器和内存地址直接进行，二反汇编则会涉及到更为复杂的数据管理，例如使用栈帧管理局部变量，所以反汇编生成的代码往往会有更多的入栈和出栈操作，同时也涉及函数调用时的参数传递和返回值处理。
2. 直接用汇编语言完成需要调用 DOS 中断来实现输出，所以涉及到多位数字的输出很不方便，需要逐位转化为字符并压入栈中，再逐位弹出转化为字符串，最后输出字符串。

3. 通过这次实验，我学会了使用 DOS 中断（INT 21H）实现输入、输出字符、输出字符串以及结束程序等功能。这些功能在编写汇编程序时是非常重要的，为程序的用户交互提供了基础。