Assignment 3 SUM

计算 1~100 的累加和

汇编代码

程序执行结果:



汇编代码如下:

; 采用小内存模型 .MODEL SMALL .STACK 100h ; 分配256字节堆栈空间 .DATA SumResult DB ? ; 存储1-100的累加和(低8位) OutputResult DB ? ; 预留输出变量 (未使用) .CODE START: MOV AX, @DATA ; 获取数据段地址(为初始化DS做准备) ; 初始化数据段寄存器, 使程序能访问数据段 MOV DS, AX ;循环计数器设为100(控制累加范围1-100) MOV CX, 100 MOV AX, 0 ; 累加器清零,准备存储总和 SumLoop: ; 累加当前CX的值(实现1+2+...+100) ADD AX, CX LOOP SumLoop ; CX自动减1, 非0则继续循环, 直到CX=0 MOV SumResult, AL ; 保存累加结果的低8位 PUSH AX ; 将完整结果(AX=5050)压栈暂存 MOV CX, 10 ; 十进制基数,用于后续取位 MOV BX, 10000 ; 从万位开始解析数字(除数初始值) PrintLoop: XOR DX, DX MOV AX, BX ; 计算下一位除数(如10000→1000) DIV CX ; 若除数为0, 说明所有位已处理完 CMP AX, 0 JL ExitProgram JE ExitProgram ; 更新除数为下一位(如千位、百位等) MOV BX, AX XOR DX, DX ; 恢复累加结果到AX POP AX DIV BX ; 取出当前位的数字(商存AL) PUSH DX ; 余数入栈, 作为下次计算的被除数

```
; 数字转ASCII码(如5→'5')
   ADD AL, 30H
   MOV DL, AL
                  ; DOS功能号:字符输出
   MOV AH, 2
   INT 21H
                  ; 调用中断显示当前位
   JMP PrintLoop
                  ; 继续处理下一位
ExitProgram:
  MOV AH, 4CH
                  ; DOS功能号:程序退出
                  ; 结束程序
  INT 21H
                   ;程序入口为START
END START
```

在编写本次汇编程序时,我对寄存器、数据段和栈的用法有了更深入的认识。

程序中,我将 CX 初始化为 100,通过 AX 寄存器存储累加结果来实现快速计算。由于寄存器位于 CPU 内部,像加法这类简单操作依赖它能显著提升速度,因此中间计算结果暂存在 AX 中,便于循环中快速更新总和。

但寄存器空间有限,需要保存结果或进行复杂数据管理时,数据段便发挥作用。我将最终求和结果存入数据段的 SumResult 变量,通过设置 DS 寄存器指向数据段,能轻松将 AX 中的值写入内存,确保程序可随时访问和操作这些数据。

栈的使用同样关键。输出部分,我利用栈的先进后出特性管理寄存器和中间结果。比如输出前需保存 AX 的值时,用 PUSH AX 压入栈,后续再通过 POP AX 恢复,保证了数据一致性,让多步骤处理中能灵活管理寄存器内容,避免关键数据丢失。

综上,我明确了三者的分工:寄存器用于高效运算,数据段存储较大或需持久化的数据,栈则负责程序运行中的状态保存与恢复。合理运用这些机制,能大幅提升汇编程序的效率与灵活性。

通过 C 语言实现并查看反汇编代码

C 语言代码:

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

return 0;
}</pre>
```

反汇编代码:

07C A . 0000	DOC BOOT	MOLL	AU ATER
076A:0000		MOV	AX,076D
076A:0003	8ED8	MOV	DS,AX
076A:0005	B96400	MOV	CX,0064
076A:0008	B80000	MOV	AX,0000
076A:000B	03C1	ADD	AX,CX
076A:000D	EZFC	LOOP	000B
076A:000F	A20000	MOV	[000C],AL
076A:001Z	50	PUSH	AX
076A:0013	B90A00	MOV	CX,000A
076A:0016	BB1027	MOV	BX,2710
076A:0019	33DZ	XOR	DX,DX
076A:001B	8BC3	MOV	AX,BX
076A:001D	F7F1	DIV	CX
076A:001F	3D0000	CMP	AX,0000
0764 : 0022	7014	TT	0038

076A:0022	7014	JL	0038
076A:0024	7412	JZ	0038
076A:0026	8BD8	MOV	BX,AX
076A:0028	33D2	XOR	DX,DX
076A:00ZA	58	POP	AX
076A:002B	F7F3	DIU	BX
076A:00ZD	52	PUSH	DX
076A:00ZE	0430	ADD	AL,30
076A:0030	8AD0	MOV	DL,AL
076A:0032	B402	MOV	AH,02
076A:0034	CD21	INT	21
076A:0036	EBE1	JMP	0019
076A:0038	B44C	MOV	AH,4C
076A:003A	CD21	INT	21

解释如下:

076A:0000 B86D07	MOV AX,076D	;将数据段地址076D加载至AX寄存器,用于初始化数据段
076A:0000 B86D07 076A:0003 8ED8	MOV AX,076D	
	·	
076A:0005 B96400	MOV CX,0064	,
076A:0008 B80000	MOV AX,0000	
076A:000B 03C1	ADD AX,CX	; 执行累加操作: AX = AX + CX (从100递减累加)
076A:000D E2FC	LOOP 000B	; 循环控制: CX减1后若不为零则返回000B继续累加
076A:000F A20000	MOV [0000],AL	;将累加结果的低字节存储到数据段偏移地址0000处
076A:0012 50	PUSH AX	;将完整的累加结果(5050)压入堆栈保存
076A:0013 B90A00	MOV CX,000A	; 设置基数为10(十六进制0A),用于十进制位分解
076A:0016 BB1027	MOV BX,2710	;初始化除数为10000(十六进制2710),从最高位开始处理
076A:0019 33D2	XOR DX, DX	; 清零DX寄存器,为32位除法做准备(DX:AX作为被除数)
076A:001B F7F3	DIV BX	;执行DX:AX除以BX的运算,商在AX,余数在DX
076A:001D 8BC3	MOV AX, BX	;将当前除数转移到AX寄存器
076A:001F F7F1	DIV CX	;将除数除以10得到下一位的权值(如10000→1000)
076A:0022 7C14	JL 0038	; 若结果为负则跳转至程序结束(异常处理)
076A:0024 7412	JZ 0038	;若商为零则跳转至程序结束(所有位数处理完毕)
076A:0026 8BD8	MOV BX,AX	; 更新除数为新的位权值(如1000)
076A:0028 33D2	XOR DX, DX	;再次清零DX为下一次除法做准备
076A:002A 58	POP AX	; 从堆栈恢复累加结果到AX寄存器
076A:002B F7F3	DIV BX	; 用当前位权值分解出当前位的数字
076A:002D 52	PUSH DX	; 将余数压栈保存,用于下一位的计算
076A:002E 0430	ADD AL,30	; 将数字(0-9)转换为对应的ASCII字符
076A:0030 BA0200	MOV DX,0002	; 设置DOS功能参数(应为字符输出准备)
076A:0033 CD21	INT 21H	;调用DOS中断21H显示当前数字字符

计算累加和

汇编代码

程序执行结果:

```
NUS<
Input (1-100): 1
Output: 1
D:\>SUM
Input (1-100): 2
Output: 3
D:\>SUM
Input (1-100): 10
Output: 55
D:\>SUM
Input (1-100): 50
Output: 1275
D:\>SUM
Input (1-100): 60
Output: 1830
D:N>SUM
Input (1-100): 70
Jutput: 2485
```

汇编代码如下:

```
.MODEL SMALL
                                ; 小内存模型(代码+数据≤64KB,适用于DOS小程序)
                                ; 分配256字节堆栈空间, 存临时数据和返回地址
.STACK 100h
.DATA
   prompt_msg DB 'Input (1-100): $' ; 输入提示串,以'$'结尾(符合DOS 09h输出格式)
   sum_msg DB 'Output: $'
                              ; 结果提示串,以'$'结尾
                                ; 存用户输入的1-100整数
   number
           DW ?
           DW 0
                                ; 存1到输入数的累加和, 初始为0
   sum
.CODE
START:
  MOV AX, @DATA
                                ; 取数据段地址到AX(DS不可直接赋值)
  MOV DS, AX
                                ; 初始化DS, 使程序能访问数据段变量
                                ;初始化ES,与DS同段(备用)
  MOV ES, AX
                                ; 加载提示串地址到DX (DOS 09h要求地址在DX)
   LEA DX, prompt_msg
  MOV AH, 09h
                                ; DOS功能号09h: 输出'$'结尾字符串
   INT 21h
                                ; 调用中断, 显示输入提示
                                ; 清空CX, 作为输入数字的累加寄存器
   XOR CX, CX
                                ; 读取键盘输入, 转成十进制整数
READ INPUT:
                                ; DOS功能号01h: 读单个字符(带回显)
  MOV AH, 01h
   INT 21h
                                ; 读取字符ASCII码存入AL
   CMP AL, ODh
                                ; 判断是否按回车键 (ASCII 0Dh)
                                ; 是则跳至求和
   JE CALCULATE SUM
```

```
SUB AL, '0'
                                 ; 字符转数字(如'5'→5)
   MOV BL, AL
                                 ; 数字暂存BL
                                 ; 已存数字载入AX
   MOV AX, CX
   MOV CX, 10
                                 ; CX设为10(处理高位)
   MUL CX
                                ; AX = AX*10 (如1→10, 准备拼下一位)
                                ; 拼接当前数字(如10+2=12)
   ADD AX, BX
   MOV CX, AX
                                 ; 结果存回cx, 继续累加
   JMP READ INPUT
                                 ; 循环读下一个字符
                                 ; 计算1到输入数的累加和
CALCULATE SUM:
   MOV AX, CX
                                 ; 输入数载入AX
   MOV number, AX
                                ; 存入number变量
   MOV CX, AX
                                ; 输入数作为循环计数器
   MOV BX, 1
                                ; 累加起始值设为1
   XOR AX, AX
                                 ; 清空AX, 作为累加器
SUM LOOP:
                                 ; 累加循环(1+2+...+输入数)
  ADD AX, BX
                                 ; AX += BX (累加当前值)
   INC BX
                                 ; BX自增1 (下一个累加值)
                                 ; CX自减1, 非0则继续循环
   LOOP SUM LOOP
                                ; 结果存入sum变量
   MOV sum, AX
   LEA DX, sum_msg
                                ; 加载结果提示串地址到DX
                                ; DOS功能号09h: 输出提示串
   MOV AH, 09h
                                ; 显示"Output: "
   INT 21h
   MOV AX, sum
                                ; 结果载入AX(作为子程序参数)
   CALL PRINT DECIMAL
                                ; 调用子程序, 十进制输出结果
   MOV AH, 4Ch
                                 ; DOS功能号4Ch: 终止程序
   INT 21h
                                 ; 结束程序
PRINT DECIMAL PROC
                                 ; 子程序: AX中整数转十进制并输出
                                 ; 保存寄存器现场 (避免影响主程序)
  PUSH AX BX CX DX
   XOR CX, CX
                                 ; 清空CX, 作为位数计数器
   MOV BX, 10
                                 ; BX=10 (十进制基数,用于取位)
                                 ; 数字转十进制(逆序存栈)
CONVERT LOOP:
   XOR DX, DX
                                 ; 清空DX (除法需DX:AX为被除数)
                                 ; AX÷10, 商存AX, 余数(当前位)存DX
   DIV BX
   PUSH DX
                                 ; 余数压栈(栈特性实现正序输出)
   INC CX
                                ; 位数计数+1
                                 ; 商是否为0 (是否取完所有位)
   CMP AX, 0
   JNE CONVERT LOOP
                                 ; 非0则继续取位
                                 ; 从栈取数, 转ASCII并输出
PRINT LOOP:
                                 ; 弹出一位数字(DL存数)
   POP DX
   ADD DL, '0'
                                ; 数字转ASCII (如5→'5')
                                ; DOS功能号02h: 输出单个字符
   MOV AH, 02h
                                ; 显示当前位
   INT 21h
                                ; 循环输出所有位
   LOOP PRINT LOOP
   POP DX CX BX AX
                                ; 恢复寄存器现场
   RET
                                 ; 子程序返回主程序
PRINT DECIMAL ENDP
                                 ; 子程序结束
```

;程序入口为START

汇编程序缺乏高级语言的库函数支持,输入输出操作需依赖底层系统调用,而 DOS 系统中,**中断 21h**是处理这类操作的核心工具。

在字符输入方面,通过为中断 21h 设置不同功能号可实现多样操作。例如,使用功能号 01h 能读取单个字符输入,并将其存入寄存器以便后续处理。这种直接操控硬件的方式,让我对计算机的输入机制有了更深入的理解。

输出操作的实现同样依赖中断 21h,这让我学会了如何精确控制字符在屏幕的显示。程序中,我用功能号 09h 输出以 \$ 结尾的字符串。相较于现代编程语言的 print 或 console.log,这种方式更底层,能清晰看到数据从内存传输到显示设备的全过程。此外,使用功能号 02h 可将单个字符从寄存器取出并显示,这让我对字符编码与 ASCII 码的转换建立了更直观的认知。

最重要的是,这些中断调用让我明白,汇编中的所有操作都需手动控制。每一次中断调用、每一次寄存器使用,都会直接决定程序的运行方式。这种学习不仅帮助我理解计算机底层工作原理,还锻炼了逻辑思维与程序设计能力。通过中断实现输入输出,既让我体验到与硬件的直接交互,也让我对程序执行流程有了更细致的把握。

通过 C 语言实现并查看反汇编代码

C 语言代码:

```
#include <stdio.h>

int main() {
    int sum = 0;
    int n = 0;
    printf("Input (1-100): ");
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    printf("Output: %d\n", sum);

return 0;
}</pre>
```

反汇编代码:

```
076A:0001 1F
                               PNP
                                         DS
076A:000Z BA0E00
                                         DX,000E
                               MOV
076A:0005 B409
076A:0007 CD21
076A:0009 B8014C
                               MOV
                                         AH,09
                               INT
                                         21
                               MOV
                                         AX,4C01
076A:000C CD21
076A:000E 54
                               INT
                               PUSH
                                         SP
076A:000F 68
                               DB
976A:0010 69
                                         69
                               DΒ
076A:0011 7320
                               JNB
                                         0033
076A:0013 7072
                               JO
                                         0087
076A:0015 6F
                               DB
                                         6F
076A:0016 67
076A:0017 7261
                                         67
                               DB
                               JB
                                         007A
076A:0019 6D
                               DB
                                         6D
076A:001A 206361
                                         [BP+DI+61],AH
                               AND
076A:001D 6E
                                         6E
                               DB
976A:001E 6E
                                         6E
                               \mathbf{DR}
976A:001F 6F
                               \mathbf{DR}
```

```
976A:0020 7420
                                     0042
076A:0022 62
                            DΒ
                                     62
076A:0023 65
                                     65
                            DB
076A:0024 207275
                            AND
                                     [BP+SI+75],DH
076A:0027 6E
                            DB
                                     6E
076A:0028 20696E
                           AND
                                     [BX+DI+6E],CH
076A:002B 20444F
                                     [SI+4F],AL
                           AND
076A:002E 53
                           PUSH
                                     BX
076A:002F 206D6F
076A:0032 64
                                     [DI+6F1,CH
                           AND
                           DB
                                     64
                                     65
076A:0033 65
                           DB
076A:0034 ZE
                           CS:
076A:0035 0D0D0A
                           OR
                                     AX,0AOD
076A:0038 2400
                           AND
                                     AL,00
076A:003A 0000
076A:003C 0000
                                     [BX+SI],AL
                           ADD
                                     [BX+SI],AL
[BX+SI],AL
                            ADD
076A:003E 0000
                            ADD
```

解释如下:

```
076A:0000 B86D07 MOV AX,076D; 将程序数据段的段地址076D加载到AX寄存器,为后续数据段初始化做准备
076A:0003 8ED8 MOV DS,AX; 将AX中的段地址传送到数据段寄存器DS, 建立数据段的寻址能力
076A:0005 B96400 MOV CX,0064 ; 设置循环计数器CX为100(十六进制64),控制从100到1的递减循环
076A:0008 B80000 MOV AX,0000 ; 清零AX寄存器, 作为累加结果的初始值
076A:000B 03C1 ADD AX,CX ; 将CX的当前值累加到AX中, 实现1到100的求和计算
076A:000D E2FC LOOP 000B ; 循环控制: CX减1后若不为零则跳回000B继续累加
076A:000F A20000 MOV [0000],AL; 将累加结果的低8位存储到数据段偏移地址0000处
076A:0012 50 PUSH AX ; 将完整的累加结果(5050)压入堆栈保存
076A:0013 B90A00 MOV CX,000A;设置基数为10(十六进制0A),用于后续的十进制位分解
076A:0016 BB1027 MOV BX,2710 ; 初始化除数为10000(十六进制2710), 从最高位开始处理
076A:0019 33D2 XOR DX, DX; 清零DX寄存器, 为32位除法做准备
076A:001B F7F3 DIV BX ; 执行DX:AX除以BX的除法运算, 商在AX, 余数在DX
076A:001D 8BC3 MOV AX,BX ; 将当前除数转移到AX寄存器
076A:001F F7F1 DIV CX ; 将除数除以10得到下一位的除数(如10000→1000)
076A:0022 7C14 JL 0038 ; 若结果为负则跳转到结束处理(实际不会触发)
076A:0024 7412 JZ 0038 ; 若除数为零则跳转到结束处理
076A:0026 8BD8 MOV BX,AX ; 更新除数为新的位权值
076A:0028 33D2 XOR DX,DX; 再次清零DX为下一次除法做准备
076A:002A 58 POP AX ; 从堆栈恢复累加结果到AX寄存器
076A:002B F7F3 DIV BX ; 用当前位权值分解出当前位的数字
076A:002D 52 PUSH DX ; 将余数压栈保存, 用于下一位的计算
076A:002E 0430 ADD AL,30 ; 将数字转换为对应的ASCII字符
076A:0030 BA0200 MOV DX,0002 ; 设置DOS功能参数(此处应为字符输出准备)
076A:0033 CD21 INT 21H ; 调用DOS中断显示当前数字字符
```