Assignment 2 ASCII

1. **用loop指令实现**

1.1 汇编代码

.MODEL SMALL                   ; 定义内存模型为 small 模式

.STACK 100h                    ; 定义堆栈大小为 256 字节

.DATA

    newline DB 0Dh, 0Ah, '$' ; 定义换行符，0Dh 是回车符，0Ah 是换行符，$ 是 DOS 输出字符串的结束符

.CODE

MAIN PROC

    MOV AX, @DATA              ; 将数据段基地址加载到 AX

    MOV DS, AX                 ; 将 AX 的值加载到数据段寄存器 DS

    MOV CX, 26                 ; 初始化 CX 为 26，表示a-z的总数

    MOV AL, 'a'                ; 初始化 AL 为 'a'，即第一个小写字母

outer\_loop: ; 外层循环

    PUSH CX                    ; 保存外层循环的 CX，准备内层循环

    MOV CX, 13                 ; 初始化 CX 为 13，表示每行输出 13 个字符

    CMP AL, 'z'                ; 检查 AL 中的值是否大于 'z'

    JA end\_loop                ; 如果 AL 已经超过 'z'，跳转到结束标签

inter\_loop: ; 内层循环

    MOV AH, 02h                ; 设置 AH 为 2，调用 DOS 打印单个字符

    MOV DL, AL                 ; 将当前的字母存入 DL 准备输出

    INT 21h                    ; 调用 DOS 中断 21h，输出字符

    INC AL                     ; 递增 AL，指向下一个字母

    LOOP inter\_loop            ; 内层循环，用 LOOP 指令控制循环 13 次

    MOV AH, 09h               ; 设置 AH 为 9，调用 DOS 输出字符串功能

    LEA DX, newline            ; 将换行符的地址加载到 DX

    INT 21h                    ; 调用 DOS 中断 21h，输出换行符

    POP CX                     ; 恢复外层循环的 CX

    SUB CX, 13                 ; 减去已经打印的 13 个字符

    JNZ outer\_loop             ; 如果还有剩余(CX不为0)，跳回outer\_loop

end\_loop:

    MOV AH, 4Ch                ; 设置 AH 为 4Ch，调用 DOS 的程序退出功能

    INT 21h                    ; 调用 DOS 中断 21h，程序结束

MAIN ENDP

END MAIN                       ; 程序结束

* 1. 汇编结果

1. **用条件跳转指令（jump）实现**

2.1 汇编代码

.MODEL SMALL ; 定义内存模型为 small 模式

.STACK 100h ; 定义堆栈大小为 256 字节

.DATA

    newline DB 0Dh, 0Ah, '$' ; 定义换行符，0Dh 是回车符，0Ah 是换行符，$ 是 DOS 输出字符串的结束符

    counter DB 0              ; 字符计数器

.CODE

MAIN PROC

    MOV AX, @DATA ; 将数据段基地址加载到 AX

    MOV DS, AX ; 将 AX 的值加载到数据段寄存器 DS

    MOV AL, 'a'               ; 初始化 AL 为 'a'，即第一个小写字母

    MOV counter, 0            ; 初始化计数器

print\_loop: ; 打印循环

    CMP AL, 'z' ; 检查是否已经打印完所有字母

    JA end\_program             ; 如果超过 'z',结束程序

    MOV AH, 02h ; 设置 AH 为 2，调用 DOS 打印单个字符

    MOV DL, AL ; 将当前的字母存入 DL 准备输出

    INT 21h ; 调用 DOS 中断 21h，输出字符

    INC AL ; 递增 AL，指向下一个字母

    INC counter ; 递增计数器

    CMP counter, 13 ; 检查是否需要换行(每13个字符)

    JNE print\_loop             ; 如果不等于13,继续打印

    MOV AH, 09h ; 设置 AH 为 9，调用 DOS 输出字符串功能

    LEA DX, newline ; 将换行符的地址加载到 DX

    INT 21h ; 调用 DOS 中断 21h，输出换行符

    MOV counter, 0 ; 重置计数器

    JMP print\_loop             ; 继续下一行

end\_program:

    MOV AH, 4Ch ; 设置 AH 为 4Ch，调用 DOS 的程序退出功能

    INT 21h ; 调用 DOS 中断 21h，程序结束

MAIN ENDP

END MAIN ; 程序结束

* 1. 汇编结果

1. **用C语言实现后查看反汇编代码并加注释**

3.1 C语言代码

#include <stdio.h>

int main() {

    int count = 0;  *// 计数器*

    for (char ch = 'a'; ch <= 'z'; ch++) {

        printf("%c", ch);

        count++;

        if (count == 13) {

            printf("\n");

            count = 0;

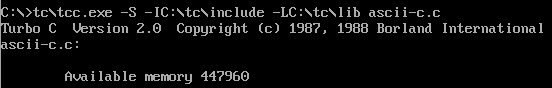
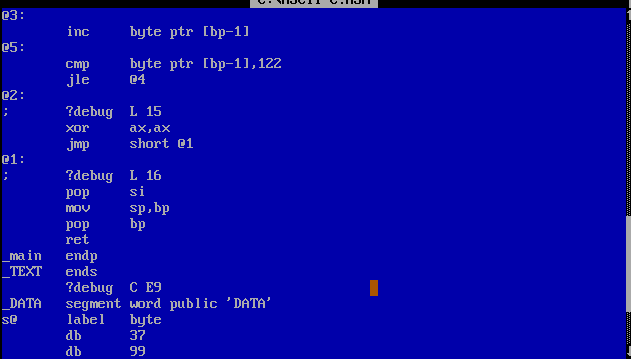
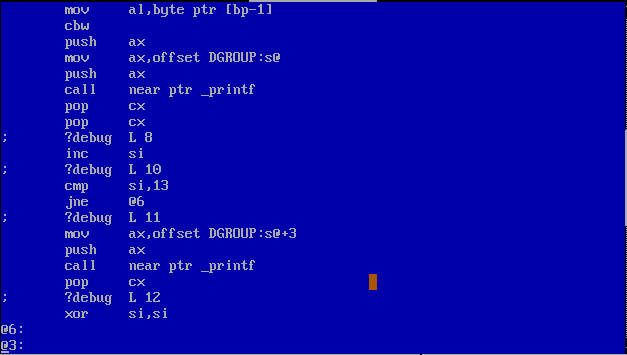
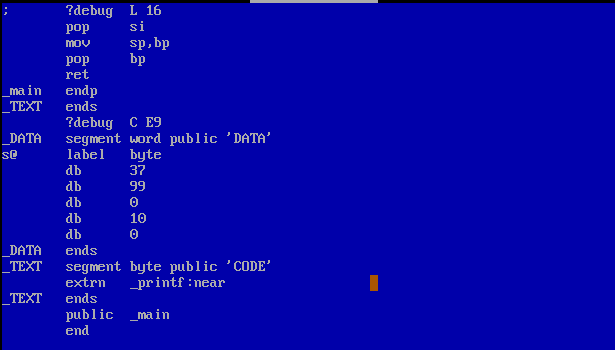
        }

    }

    return 0;

}

* 1. 编译运行

1. 首先下载TurboC\_v2.0，将里面的tcc.exe复制到目录下面
2. 接下来生成汇编文件ascii-c.asm，-S 参数表示“生成汇编文件，不执行汇编和链接”
3. 使用edit ascii-c.asm在编辑器里打开
4. 添加注释

\_main   proc    near             ;函数的开始

        push    bp               ;保存堆栈基址

        mov     bp,sp

        sub     sp,2             ;在堆栈上为局部变量 'ch' 预留空间

        push    si               ;保存si寄存器 (因为它要被代码使用)

        xor     si,si            ;把'count 变量存到了 SI 寄存器里

        mov byte ptr [bp-1],97  ; 这是ch = 'a';

                                ; 97 是 'a' 的 ASCII 码

                                ; [bp-1] 是 'ch' 变量在堆栈上的地址

        jmp     short @5         ; for循环, 跳到 @5 去做条件检查

@4:

        mov     al,byte ptr [bp-1]  ;把 ch ([bp-1]) 加载到 AL

        cbw                        ;把 AL (字节) 扩展到 AX (字)

        push    ax                   ;把 ch 作为参数压栈

        mov     ax,offset DGROUP:s@ ;把格式字符串 "%c" 的地址加载到 AX

        push    ax                   ;把 "%c" 地址作为参数压栈

        call    near ptr \_printf     ;调用 printf 函数

        pop     cx                   ;清理堆栈

        pop     cx                   ;清理堆栈

        inc     si    ;count++ (因为 count 在 SI 寄存器里)

        cmp     si,13                ;比较 count (si) 和 13

        jne     @6                   ;如果不等于, 则 JNE (Jump Not Equal)

                                    ;跳到 @6 (跳过换行代码)

        mov     ax,offset DGROUP:s@+3;把换行符 "\n" 的地址加载到 AX

        push    ax                   ;把 "\n" 地址作为参数压栈

        call    near ptr \_printf     ;调用printf 函数

        pop     cx                   ;清理堆栈

        xor     si,si                ;count = 0 (再次把 si 清零)

@6:                                  ;if (count != 13) 跳到的地方

@3:

        inc     byte ptr [bp-1]     ;ch++

@5:                                  ;循环条件检查的标签

        cmp     byte ptr [bp-11],122 ;比较ch和'z'(122 是 'z' 的 ASCII)

        jle     @4                   ;JLE = Jump if Less or Equal

                                    ;如果 ch <= 'z', 跳回到 @4 (循环体)

@2:                                  ;循环结束后的代码

@1:                                  ;函数返回

        pop     si                  ;恢复 si 寄存器

        mov     sp,bp                ;恢复堆栈

        pop     bp                   ;恢复堆栈

        ret                          ;return; (从 main 函数返回)

\_main   endp                         ;函数结束

s@      label   byte

        db      37       ;ASCII '%'

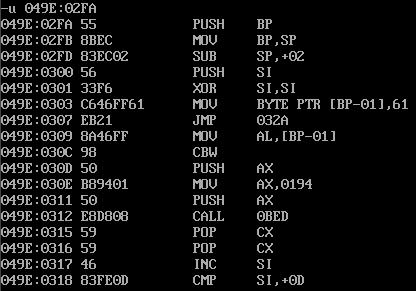
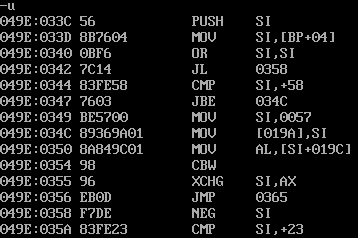
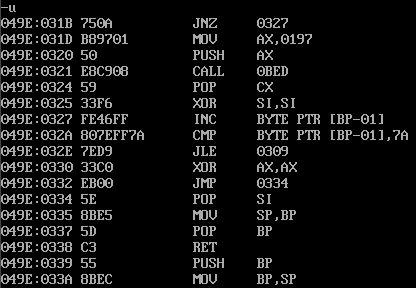
        db      99       ;ASCII 'c'

        db      0        ;字符串结束符 '\0'

        db      10       ; ASCII '\n' (换行符)

        db      0        ; 字符串结束符 '\0'

部分反汇编代码：

1. **总结**

通过LOOP指令和条件跳转指令的对比，我认识到不同的循环方式在性能和灵活性上的差异。例如，LOOP 依赖于CX寄存器的递减操作，而条件跳转指令则通过手动调整寄存器和标志位进行循环控制，更具灵活性。

在代码中使用了汇编语言中的基本指令如 MOV, INT, LOOP, PUSH, POP, CMP, JMP 等让我熟悉了汇编指令如何操作寄存器、内存、栈以及如何调用中断服务。通过这些指令，我加深了对数据传递、循环控制和条件跳转的理解。