


Assignment 2 ASCII

LOOP 指令实现小写英文字母的输出

程序结果：



汇编代码如下：

```
.MODEL SMALL                ; 指定使用小内存模型（代码和数据各不超过 64KB）
.STACK 100h                 ; 分配 256 字节的堆栈空间

.DATA
    newline DB 0Dh, 0Ah, '$' ; 定义回车换行符序列，用于 DOS 环境下的换行输出，'$' 为字符串结束标记

.CODE
MAIN PROC
    MOV AX, @DATA            ; 获取数据段的段地址
    MOV DS, AX              ; 将数据段地址装入 DS 寄存器，建立数据段访问
    MOV CX, 26               ; 设置总字母数：26 个小写英文字母
    MOV AL, 'a'              ; 从字符 'a' 开始作为初始输出字符

outer_loop:
    PUSH CX                  ; 保存外层循环计数器（剩余字母总数）
    MOV CX, 13               ; 内层循环次数设为 13，每行输出 13 个字符
    CMP AL, 'z'              ; 判断当前字符是否已超过 'z'
    JA end_loop              ; 若已超出字母范围，则跳转至程序结束

inter_loop:
    MOV AH, 02h              ; 设置 DOS 功能号：02h — 输出单个字符
    MOV DL, AL                ; 待输出字符送入 DL 寄存器
    INT 21h                  ; 调用 DOS 中断，显示字符
    INC AL                    ; 字符递增，例如从 'a' 变为 'b'
    LOOP inter_loop           ; 循环执行内层输出，直到 CX 减为 0（共 13 次）

    MOV AH, 09h              ; 设置 DOS 功能号：09h — 输出字符串
    LEA DX, newline           ; 取换行符字符串的偏移地址送入 DX
    INT 21h                  ; 调用 DOS 中断，输出换行符
    POP CX                    ; 恢复外层循环的原始计数值
    SUB CX, 13                ; 减去本次已打印的 13 个字符
    JNZ outer_loop            ; 若仍有未打印字符（CX ≠ 0），继续外层循环

end_loop:
    MOV AH, 4Ch              ; 设置 DOS 功能号：4Ch — 终止程序
    INT 21h                  ; 调用 DOS 中断，正常退出程序并返回操作系统
```

```
MAIN ENDP
END MAIN ; 汇编结束，指定程序入口点为 MAIN
```

在完成本次汇编程序的编写过程中，我深刻体会到了汇编语言在底层系统操作中所具备的精细控制能力。通过实际编程实践，我更加熟悉了寄存器在数据传递中的核心作用，也掌握了如何通过循环结构与条件跳转指令精确控制程序的执行流程。

我对 `LOOP` 指令的机制有了更深入的认识。该指令通过自动递减 `CX` 寄存器的值来判断是否继续循环，这一特性体现了汇编语言在实现循环时的简洁与高效。特别是在实现嵌套循环时，我意识到必须使用 `PUSH` 和 `POP` 指令保存和恢复外层循环的计数器，以防止内层循环对 `CX` 的修改破坏外层循环逻辑，这凸显了在汇编中手动管理寄存器状态的重要性。

同时，我对 DOS 中断 `INT 21h` 的应用有了更扎实的掌握。通过设置 `AH=02h` 实现单个字符的输出，以及使用 `AH=09h` 输出以 `$` 结尾的字符串，我理解了不同功能号对应的具体服务机制。在逐个输出小写字母的过程中，我切实感受到了汇编语言对执行过程的直接掌控力，能够精确控制每一个输出动作和格式。

此次编程经历也让我更加明白，汇编语言要求程序员对每一步操作进行显式管理，虽然编程复杂度较高，但正因如此，它能让人更清晰地理解程序在硬件层面的运行机制。这段代码不仅提升了我对汇编语法的熟练度，也加深了我对 CPU 指令执行顺序、循环控制、条件判断以及中断调用等底层工作原理的理解，进一步增强了对计算机体系结构的整体认知。

条件跳转指令实现小写英文字母的输出

程序执行结果：

```
D:\>ASCII
abcdefghijklmnopqrstuvwxyz
```

汇编代码如下：

```
.MODEL SMALL ; 采用小模式内存模型：代码段和数据段均不超过 64KB
.STACK 100h ; 声明大小为 256 字节的运行时堆栈

.DATA
    newline DB 0Dh, 0Ah, '$' ; 定义回车换行符序列，用于在 DOS 中实现换行输出；'$' 作为字符串输出终止标志

.CODE
MAIN PROC
    MOV AX, @DATA ; 将当前程序的数据段地址送入 AX 寄存器
    MOV DS, AX ; 将 AX 中的数据段地址加载到 DS，完成数据段初始化
    MOV CX, 26 ; 设置外层循环总次数：共 26 个小写英文字母需要打印
    MOV AL, 'a' ; 初始化起始字符为 'a'，存储于 AL 中

outer_loop:
    PUSH CX ; 保存外层循环剩余字符数（CX 值），以便内层使用
    MOV CX, 13 ; 设置内层循环计数为 13，控制每行输出 13 个字符
    CMP AL, 'z' ; 比较当前字符是否已超过最后一个字母 'z'
    JA end_loop ; 若已超出范围，则跳转至程序结束处

inter_loop:
```

```

    CMP CX, 0                ; 判断内层循环计数器是否已减至零
    JE end_inner_loop        ; 若等于零，表示本行已输出完毕，跳出内层循环
    MOV AH, 02h              ; 设置 DOS 功能号：02h — 输出单个字符
    MOV DL, AL                ; 将待显示的字符从 AL 传送到 DL
    INT 21h                  ; 调用 DOS 中断，执行字符输出
    INC AL                    ; 将 AL 中的字符递增（如 'a' → 'b'）
    DEC CX                    ; 内层循环计数器减一
    JMP inter_loop            ; 无条件跳转回循环开头，继续执行

end_inner_loop:
    MOV AH, 09h              ; 设置 DOS 功能号：09h — 显示以 '$' 结尾的字符串
    LEA DX, newline           ; 获取换行符字符串的偏移地址并送入 DX
    INT 21h                  ; 调用 DOS 中断，输出换行符以换行
    POP CX                    ; 恢复外层循环的原始计数值
    SUB CX, 13                ; 减去本次已打印的 13 个字符数量
    JG outer_loop             ; 若剩余字符数大于 0，继续执行外层循环

end_loop:
    MOV AH, 4Ch               ; 设置 DOS 功能号：4Ch — 终止程序并返回操作系统
    INT 21h                  ; 调用 DOS 中断，正常退出程序

MAIN ENDP
END MAIN                      ; 汇编源文件结束，指定程序入口为 MAIN 过程

```

在通过条件跳转指令实现小写英文字母逐个输出的过程中，我对汇编语言中程序控制流的底层机制有了更为深入的认识。与使用 `LOOP` 指令自动递减 `CX` 的方式不同，本次实现依赖于显式的条件判断和跳转指令（如 `CMP`、`JE`、`JG` 和 `JMP`），完全由程序员手动管理循环的执行与退出。这让我深刻体会到，尽管汇编语言缺乏高级语言中诸如 `for` 或 `while` 这样的结构化语法，但通过组合基本的比较与跳转指令，依然能够构建出复杂而精确的控制逻辑。

我掌握了如何利用 `CMP` 指令比较寄存器值，并结合 `JE` 等条件跳转指令来判断循环终止条件。在内层循环中，通过 `CMP CX, 0` 判断计数器是否归零，若为零则跳转至循环结束标签，从而实现了对每行13个字符输出的精准控制。这种手动递减 `CX` 并进行条件判断的方式，不仅帮助我理解了嵌套循环的实现机制，也强化了对寄存器生命周期和状态管理的操作能力。

此外，本次实践进一步加深了我对 DOS 功能调用 `INT 21h` 的理解。通过设置 `AH=02h` 实现单字符输出，以及使用 `AH=09h` 输出以 `$` 结尾的字符串（如换行符），我更加熟悉了 DOS 中断服务的调用规范。在每行输出13个字母后插入换行操作，不仅体现了对字符串输出功能的灵活运用，也提升了我在底层环境下对字符流格式控制的实践能力。

整个编程过程让我切实体会到汇编语言在控制精度上的优势。虽然需要手动维护循环变量和跳转逻辑，增加了编程的复杂性，但这也正是其强大之处——它让我们能够清晰地看到程序每一步的执行路径。这种“从零构建”的方式，使我对 CPU 如何依据指令逐条执行、如何通过标志位决定跳转方向等底层机制有了更直观和深刻的理解，进一步拉近了我与计算机硬件之间的距离。

C 语言实现并查看反汇编代码

C 语言代码：

```
#include <stdio.h>

int main() {
    int i = 0;
    while (i < 26) {
        printf("%c", 'a' + i);
        if ((i + 1) % 13 == 0) {
            printf("\n");
        }
        i++;
    }
    return 0;
}
```

一部分反汇编代码：

076A:0000	0E	PUSH	CS
076A:0001	1F	POP	DS
076A:0002	BA0E00	MOV	DX,000E
076A:0005	B409	MOV	AH,09
076A:0007	CD21	INT	21
076A:0009	B8014C	MOV	AX,4C01
076A:000C	CD21	INT	21
076A:000E	54	PUSH	SP
076A:000F	68	DB	68
076A:0010	69	DB	69
076A:0011	7320	JNB	0033
076A:0013	7072	JO	0087
076A:0015	6F	DB	6F
076A:0016	67	DB	67
076A:0017	7261	JB	007A
076A:0019	6D	DB	6D
076A:001A	206361	AND	[BP+DI+61],AH
076A:001D	6E	DB	6E
076A:001E	6E	DB	6E
076A:001F	6F	DB	6F
076A:0020	7420	JZ	0042
076A:0022	62	DB	62
076A:0023	65	DB	65
076A:0024	207275	AND	[BP+SI+75],DH
076A:0027	6E	DB	6E
076A:0028	20696E	AND	[BX+DI+6E],CH
076A:002B	20444F	AND	[SI+4F],AL
076A:002E	53	PUSH	BX
076A:002F	206D6F	AND	[DI+6F],CH
076A:0032	64	DB	64
076A:0033	65	DB	65
076A:0034	2E	CS:	
076A:0035	0D0D0A	OR	AX,0A0D
076A:0038	2400	AND	AL,00
076A:003A	0000	ADD	[BX+SI],AL
076A:003C	0000	ADD	[BX+SI],AL
076A:003E	0000	ADD	[BX+SI],AL

一部分反汇编代码的解释：

076A:0000 0E	PUSH CS	; 将代码段寄存器 (CS) 的值压入堆栈
076A:0001 1F	POP DS	; 弹出堆栈顶的数据到数据段寄存器 (DS)
076A:0002 BA 0E00 移地址	MOV DX,000E	; 将 0x000E 移动到 DX, 设置 DX 为数据段的某个偏
076A:0005 B4 09 串)	MOV AH,09	; 设置 DOS 中断 21h 的功能号为 09h (输出字符
076A:0007 CD 21	INT 21	; 调用 DOS 中断 21h, 执行字符串输出
076A:0009 B8 014C (返回码为 1)	MOV AX,4C01	; 将 4C01 移动到 AX, 调用 DOS 的程序结束功能
076A:000C CD 21	INT 21	; 调用 DOS 中断 21h, 结束程序
076A:000E 54	PUSH SP	; 将堆栈指针 (SP) 的值压入堆栈
076A:000F 68 69	DB 68, 69	; 定义两个字节的常量数据, ASCII 字符 'hi'
076A:0011 7320	JNB 0033	; 无符号比较下跳转, 如果没有进位则跳转到 0033
076A:0013 7072	JO 00787	; 如果溢出标志设置, 则跳转到 00787
076A:0015 6F	DB 6F	; 定义字节 'o'
076A:0016 7261	JB 007A7	; 如果有进位, 则跳转到 007A7
076A:0018 6D	DB 6D	; 定义字节 'm'
076A:0019 2063 61	AND [BP+DI+611],AH	; 对内存地址中的值与寄存器 AH 做 AND 操作
076A:001C 6E	DB 6E	; 定义字节 'n'
076A:001D 6E	DB 6E	; 定义字节 'n'
076A:001F 6F	DB 6F	; 定义字节 'o'