# 汇编语言课程作业

**————报告2**

2251334 倪朗恩

## 基本要求

输出ASCII表中的小写字母部分，要求每行13个字符

1. 用loop指令实现（注意双重循环中CX值的保存与恢复）
2. 用条件跳转指令实现
3. 用C语言编写后查看返汇编代码并加注释

## 汇编编写

使用NEW\_LINE作为换行符常量，COUNT作为输出剩余总数，LOW\_ASCII作为输出的开始字符“a“。

划分出四个标记字段：

1. L0

L0:

    CMP [COUNT],13

    JGE L1

    JL L2

比较剩余字符数与13的大小，大于等于进入L1段，小于进入L2段

1. L1

L1:

    MOV CX,13

    JMP L3

为CX赋值13，进入内层循环，表示接下来输出13个字符

1. L2

L2:

    MOV AL,[COUNT]

    MOV CX,0

    MOV CL,AL

为CX赋值COUNT，即剩余总数，进入最后一次循环

1. L3

L3:

    MOV AL,[LOW\_ASCII]

    MOV DL,AL

    INC AL

    MOV [LOW\_ASCII],AL

    INT 21H

LOOP L3

通过LOW\_ASCII作为中间存储变量，循环递增输出字符

1. 循环后操作

    MOV DL,[NEW\_LINE]

    INT 21H

    MOV AL,[COUNT]

    SUB AL,13

    MOV [COUNT],AL

    CMP [COUNT],0

    JG L0

输出换行符，并判断外层循环是否结束，减少COUNT/剩余总数

## C语言编写

    int j=1;

    char i='a';

    while(i<='z'){

        printf("%c",i);

        if(j==13){

            printf("\n");

            j=0;

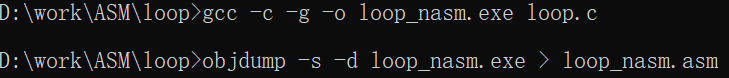
        }

        ++i;

        ++j;

    }

输出风格略有不同，原因是为了方便汇编的编写，每次内层循环结束都输出换行符



在cmd窗口输入以上命令得到汇编代码（使用了Mingw工具），并加以注释

## 实验结果

1. 汇编编写



1. C语言编写

Disassembly of section .text:

0000000000000000 <main>:

   0: 55                      push   %rbp             ;将当前的%rbp(基址寄存器)的值压入栈中，以保存其原始值

   1: 48 89 e5                mov    %rsp,%rbp        ;将%rsp(栈指针寄存器)的值复制到rbp中，从而更新rbp为新的栈帧基址

   4: 48 83 ec 30             sub    $0x30,%rsp       ;从%rsp中减去30H/48个字节(在栈上分配48B的空间，用于存放局部变量或其他临时数据)

   8: e8 00 00 00 00          callq  d <main+0xd>     ;调用函数

   d: c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)  ;将立即数（$0x1）存储到相对于%rbp指向位置偏移-0x4处，j

  14: c6 45 fb 61             movb   $0x61,-0x5(%rbp) ;同上，即存储了两个临时变量，i

  18: eb 30                   jmp    4a <main+0x4a>   ;无条件跳转

  1a: 0f be 45 fb             movsbl -0x5(%rbp),%eax  ;将从栈上相对于%rbp偏移-0x5处读取一个字节的数据扩展为一个字/32位存储到%eax中，%eax=i

  1e: 89 c1                   mov    %eax,%ecx        ;%ecx=%eax

  20: e8 00 00 00 00          callq  25 <main+0x25>   ;调用函数

  25: 83 7d fc 0d             cmpl   $0xd,-0x4(%rbp)  ;比较立即数0xd与相对于%rbp指向位置偏移-0x4处的值，j

  29: 75 11                   jne    3c <main+0x3c>   ;不相等跳转

  2b: b9 0a 00 00 00          mov    $0xa,%ecx        ;%ecx=0xa

  30: e8 00 00 00 00          callq  35 <main+0x35>   ;调用函数

  35: c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)  ;j=0

  3c: 0f b6 45 fb             movzbl -0x5(%rbp),%eax  ;%eax=i

  40: 83 c0 01                add    $0x1,%eax        ;%eax+=1

  43: 88 45 fb                mov    %al,-0x5(%rbp)   ;i=%al

  46: 83 45 fc 01             addl   $0x1,-0x4(%rbp)  ;j+=1

  4a: 80 7d fb 7a             cmpb   $0x7a,-0x5(%rbp) ;比较立即数0x7a与i（cmpb比较1Byte，cmpl比较4Byte）

  4e: 7e ca                   jle    1a <main+0x1a>   ;小于等于跳转

  50: b8 00 00 00 00          mov    $0x0,%eax        ;%eax=0

  55: 48 83 c4 30             add    $0x30,%rsp       ;%rsp+=48

  59: 5d                      pop    %rbp             ;从栈顶弹出值，将其存储%rbp中

  5a: c3                      retq                    ;从当前函数返回到调用者，并从栈中弹出返回地址，恢复%rip（指令指针寄存器）

  5b: 90                      nop                     ;空操作/占位符

  5c: 90                      nop

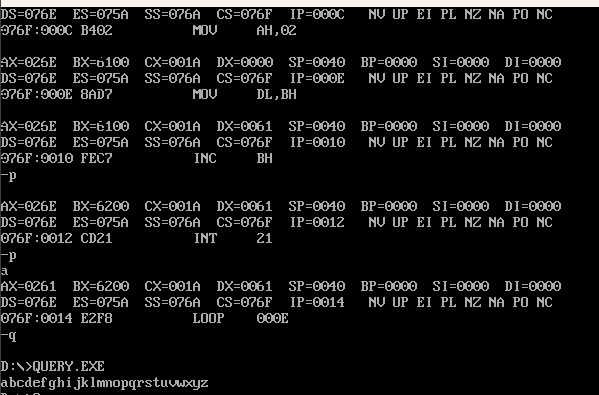
  5d: 90                      nop

  5e: 90                      nop

  5f: 90                      nop

## 问题与心得

1. INT 21H指令会使用AL寄存器，使其变为61H



心得：可以使用无关的寄存器保存数据，如本例中可以使用BX（以后案例要注意数据冲突）

1. MOV指令必须保证寄存器、数据位数相等

不可以将8位数据/寄存器与16位数据/寄存器进行比较或赋值，否则编译报错

1. 使用数据段进行常量、变量的存储，避免寄存器冲突

寄存器是有限的，而且可能会有函数进行未知的调用且不恢复，使用数据段作为中转，可以有效避免数据冲突，解决寄存器不足问题，而这是基于速度的牺牲