****

# 中南大学课程实践报告

实践名称：汇编实验

指导教师：贺建飚

专 业：计算机与科学

班 级：2202

姓 名：陈康坤

学 号：8208221512

2023年 12月 23日

目录

[中南大学课程实践报告 1](#_Toc6638)

[第二次实验 试验2.2 查找匹配字符串 2](#_Toc541)

[1. 设计说明 2](#_Toc12567)

[2. 调试说明 3](#_Toc14150)

[3. 程序框图 4](#_Toc20054)

[4. 程序清单 4](#_Toc9090)

[第三次试验 分类统计字符个数 7](#_Toc20313)

[1. 设计说明 7](#_Toc6767)

[2. 调试说明 8](#_Toc6990)

[3. 程序框图 8](#_Toc18561)

[4. 程序清单 9](#_Toc28390)

[第四次试验 查找电话号码 11](#_Toc31085)

[1. 设计说明 11](#_Toc2470)

[2. 调试说明 12](#_Toc3774)

[3. 程序框图 12](#_Toc20129)

[4. 程序清单 13](#_Toc4131)

[第五次试验 求 Fibonacci 数 20](#_Toc15369)

[1. 设计说明 20](#_Toc19269)

[2. 调试说明 20](#_Toc30777)

[3. 程序框图 21](#_Toc26395)

[4. 程序清单 21](#_Toc26662)

## 第二次实验 试验2.2 查找匹配字符串

### 1. 设计说明

#### 1.1 程序名

关键字匹配程序

#### 1.2 功能

该程序用于在输入的句子中查找特定的关键字，如果找到，则输出关键字的位置；如果未找到，则输出“NOT MATCH”。

#### 1.3 原理及算法说明

程序通过接收用户输入的关键字和句子，然后在句子中查找关键字。它使用简单的比较算法，逐字符比较关键字和句子中的字符，如果找到相匹配的子串，则输出匹配位置；否则，输出“NOT MATCH”。

#### 1.4 程序及数据结构

`mess1`, `mess2`, `mess3`, `mess4`, `mess5`: 存储输出消息的字符串

`change`: 存储回车换行的字符

`stoknin1`: 标签，用于存储关键字相关的信息

`max1`: 存储关键字的最大长度

`act1`: 存储实际输入的关键字长度

`stokn1`: 存储关键字的字符数组

`stoknin2`: 标签，用于存储句子相关的信息

`max2`: 存储句子的最大长度

`act2`: 存储实际输入的句子长度

`stokn2`: 存储句子的字符数组

#### 1.5 主要符号的说明

`mess1`, `mess2`, `mess3`, `mess4`, `mess5`: 输出消息的字符串 `change`: 回车换行的字符

`stoknin1`: 关键字的标签

`max1`: 关键字的最大长度

`act1`: 实际输入的关键字长度

`stokn1`: 关键字的字符数组

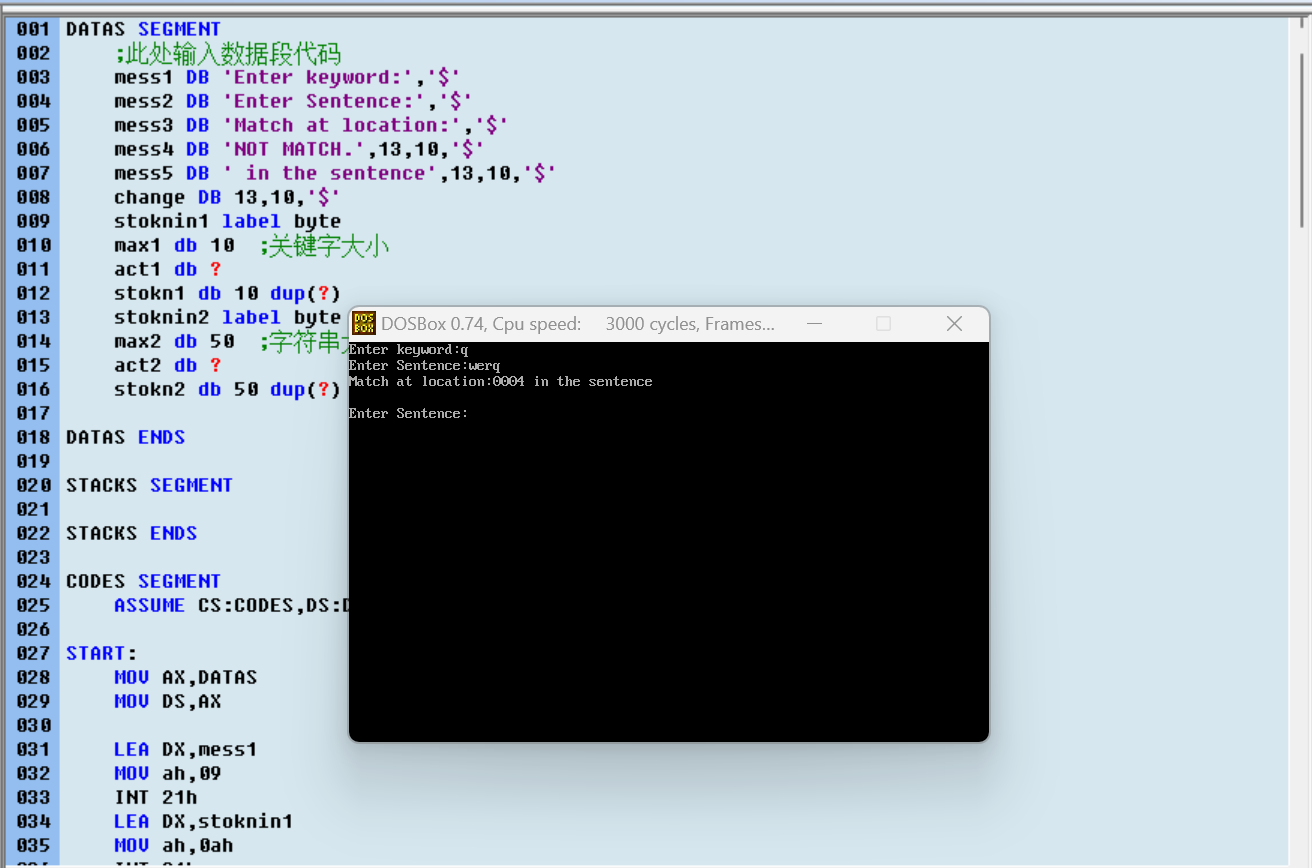
`stoknin2`: 句子的标签

`max2`: 句子的最大长度

`act2`: 实际输入的句子长度

`stokn2`: 句子的字符数组

1. 调试说明



#### 2.1 调试情况

在输入关键字和句子时，需要注意长度不要超过最大限制。

检查比较循环的逻辑，确保正确比较字符并找到匹配。

#### 2.2 解决方法

确保输入的数据不超过规定长度，防止数组溢出。

仔细检查比较逻辑，确保正确处理各种情况。

#### 2.3 观察到的现象及其分析

在输入较长的关键字或句子时，可能会导致数组越界。

比较循环中的跳转逻辑可能需要调整，以确保正确的比较。

#### 2.4 对程序设计技巧的总结分析

需要小心处理输入边界，确保不会发生数组溢出。

在比较循环中，确保逻辑正确，特别是在循环结束的条件上。

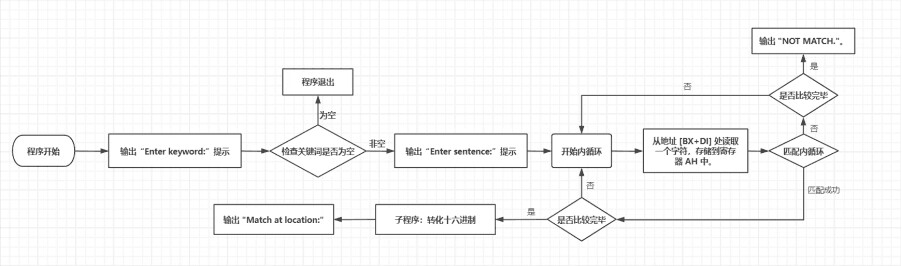
#### 2.5 程序的输出结果及分析

输入关键字和句子后，程序会输出匹配位置或“NOT MATCH”信息。

#### 2.6 实验的心得体会

通过此实验，我学到了如何在汇编语言中实现简单的字符串匹配算法。调试时要仔细检查数组的边界和循环逻辑，以确保程序的正确性。

### 3. 程序框图



### 4. 程序清单

DATAS SEGMENT

    ;此处输入数据段代码

    mess1 DB 'Enter keyword:','$'

    mess2 DB 'Enter Sentence:','$'

    mess3 DB 'Match at location:','$'

    mess4 DB 'NOT MATCH.',13,10,'$'

    mess5 DB ' in the sentence',13,10,'$'

    change DB 13,10,'$'

    stoknin1 label byte

    max1 db 10  ;关键字大小

    act1 db ?

    stokn1 db 10 dup(?)

    stoknin2 label byte

    max2 db 50  ;字符串大小

    act2 db ?

    stokn2 db 50 dup(?)

DATAS ENDS

STACKS SEGMENT

STACKS ENDS

CODES SEGMENT

    ASSUME CS:CODES,DS:DATAS,SS:STACKS

START:

    MOV AX,DATAS

    MOV DS,AX

    LEA DX,mess1

    MOV ah,09

    INT 21h

    LEA DX,stoknin1

    MOV ah,0ah

    INT 21h

    cmp act1,0

    je exit     ;如果为空直接退出程序

a10:

    ;输入Sentence并判断

    LEA DX,change

    MOV ah,09

    INT 21h

    LEA DX,mess2

    MOV ah,09

    INT 21h

            ;输出Enter Sentence:

    LEA DX,stoknin2

    MOV ah,0ah

    INT 21h

    MOV AL,act1

    CBW;拓展

    MOV CX,AX

            ;保存关键字长度到cx

    PUSH CX

    MOV AL,act2

    cmp AL,0

    je a50

            ;保存句子长度到al，若句子为空则跳转显示not match

    SUB AL,act1

    js a50

            ;若句子长度小于关键字长度，则跳转显示not match

    INC AL

    CBW

    LEA BX,stokn2

            ;将句子的首地址放进BX

    MOV DI,0

    MOV SI,0

a20:

            ;比较，内循环

    MOV AH,[BX+DI]

    CMP AH,stokn1[SI]

            ;遇见字符不相等就跳转到a30

    jne a30

    INC DI

    INC SI

    DEC CX

            ;每遇到一个相等的字符,cx 1,若cx不为0则比较下一个字符,当cx为0是说明关键字比较完

    CMP CX,0

    je a40

    jmp a20

a30:

  ;外循环，BX+1,清空si，di继续内循环比较

  INC BX

  DEC AL

  cmp AL,0

  je a50

  MOV DI,0

  MOV SI,0

  POP CX

  push CX

  jmp a20

a40:

  ;match,将bx减去句子的首地址加一得到关键字所在位置，调用二进制转十六进制子函数将位置输出

  SUB BX,offset stokn2

  INC BX

  LEA DX,change

  MOV ah,09

  INT 21h

  LEA DX,mess3

  MOV ah,09

  INT 21h

  CALL btoh

  LEA DX,mess5

  MOV ah,09

  INT 21h

  jmp a10

  ;二进制转换十六进制

  btoh PROC NEAR

  MOV CH,4

  rotate: MOV CL,4

  ROL BX,CL;左循环

  MOV AL,BL

  and AL,0fh

  add AL,30h

  cmp al,3ah

  jl printit

  add al,7h

  printit:

  MOV dl,al

  MOV ah,2

  int 21h

  dec ch

  jnz rotate

  ret

btoh endp

a50:

  ;显示not match

  LEA DX,change

  MOV ah,09

  INT 21h

  LEA DX,mess4

  MOV ah,09

  INT 21h

  jmp a10

exit:

  ret

CODES ENDS

    END START

## 第三次试验 分类统计字符个数

### 1. 设计说明

#### 1.1 程序名

**分类统计字符个数**

#### 1.2 功能

该程序接受用户输入的字符串，然后统计并输出其中字母、数字和其他字符的个数。

#### 1.3 原理及算法说明

程序通过逐个检查输入字符串中的字符，并根据其ASCII值的范围来判断是字母、数字还是其他字符。然后分别计数，并在最后输出统计结果。

#### 1.4 程序及数据结构

`INPUT\_SIZE`: 定义输入缓冲区的大小。

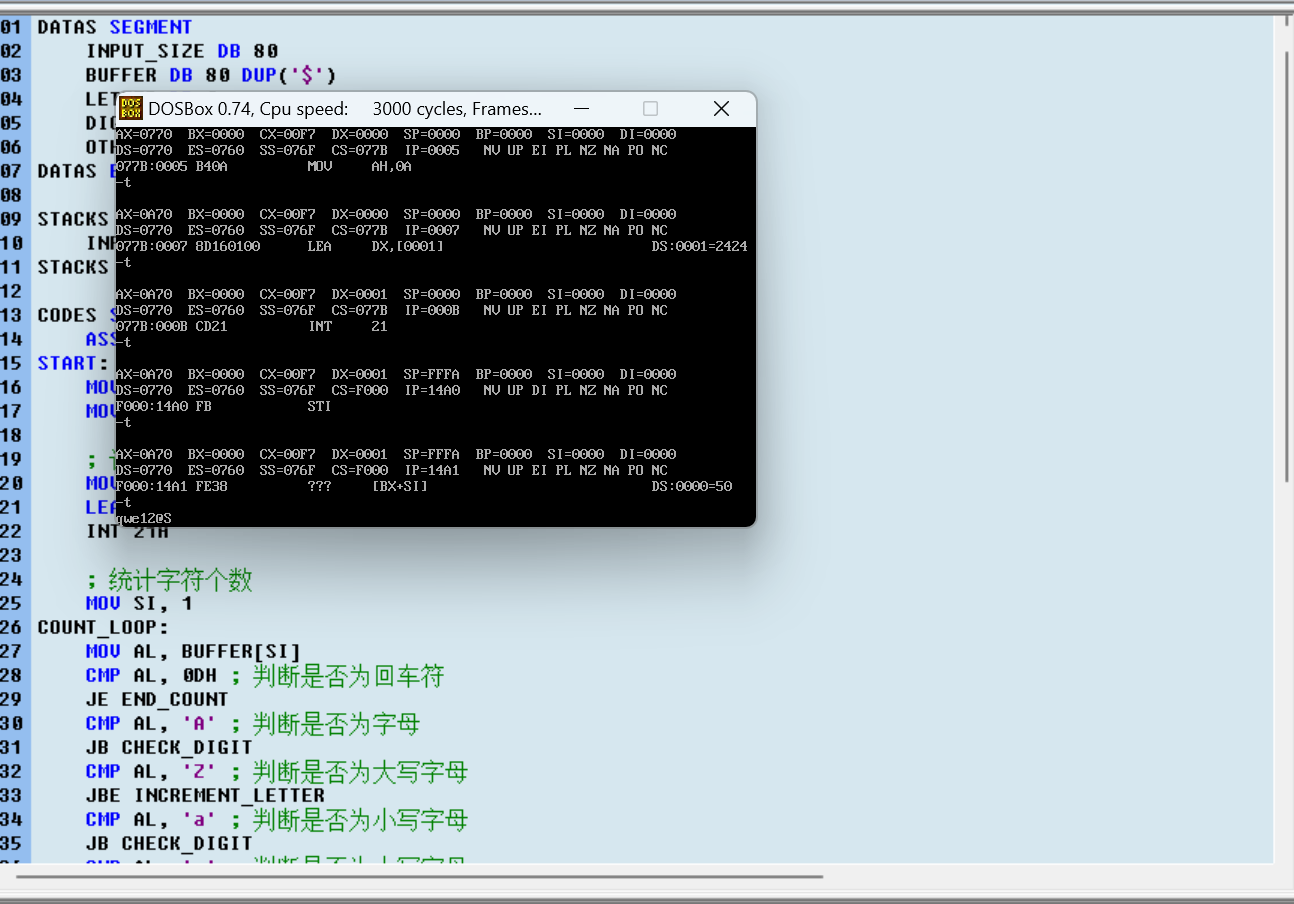
`BUFFER`: 存储用户输入的字符串的缓冲区。

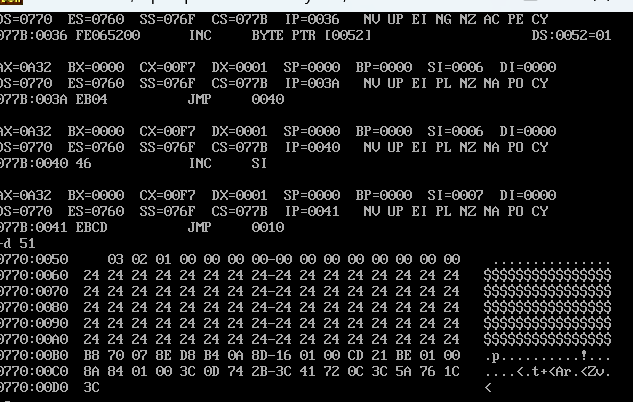
`LETTER`: 存储字母的计数。

`DIGIT`: 存储数字的计数。

`OTHER`: 存储其他字符的计数。

### 调试说明





#### 2.1 调试情况

确保输入字符串不超过缓冲区大小。

检查字符统计逻辑，确保每个字符被正确分类。

#### 2.2 解决方法

确保输入数据不超过规定大小。

逐步检查字符统计的逻辑，确保每个字符都被正确分类。

#### 2.3 观察到的现象及其分析

在输入过长的字符串时可能导致缓冲区溢出。

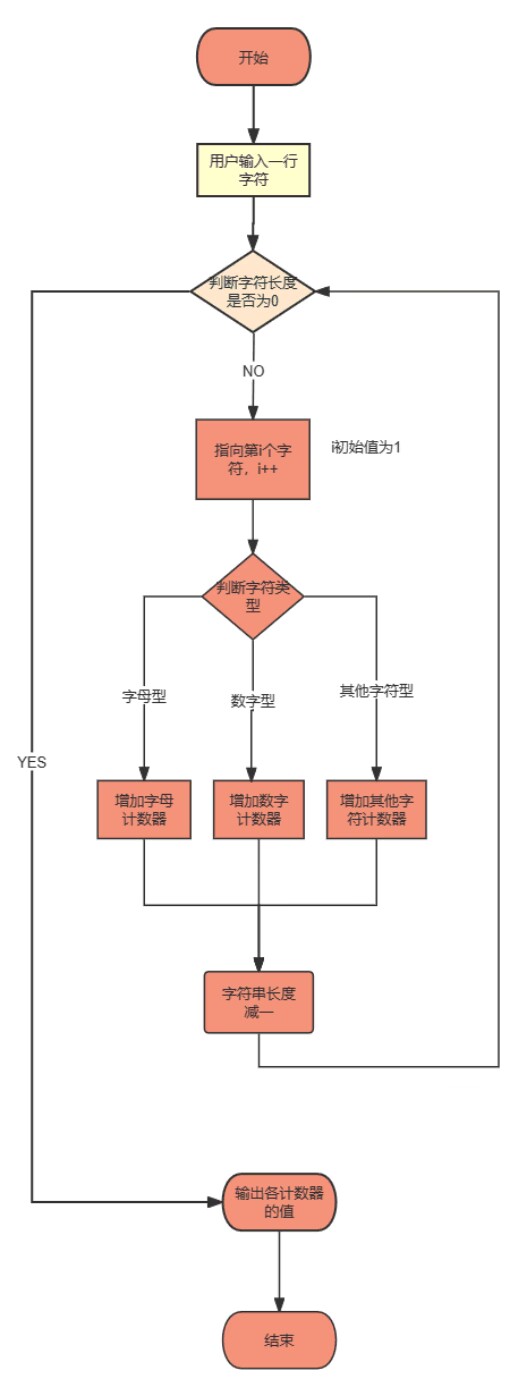
需要确保字符统计的逻辑正确，不会漏掉任何一种字符。

#### 2.4 对程序设计技巧的总结分析

缓冲区大小需要足够大，以容纳用户可能输入的字符串。

字符分类逻辑需要仔细检查，确保每个字符都被正确地分类。

### 3. 程序框图



### 4. 程序清单

DATAS SEGMENT

    INPUT\_SIZE DB 80

    BUFFER DB 80 DUP('$')

    LETTER DB 0

    DIGIT DB 0

    OTHER DB 0

DATAS ENDS

STACKS SEGMENT

    INPUT\_STACK DB 80 DUP('$')

STACKS ENDS

CODES SEGMENT

    ASSUME CS:CODES,DS:DATAS,SS:STACKS

START:

    MOV AX,DATAS

    MOV DS,AX

    ; 读取输入至缓冲区

    MOV AH, 0AH

    LEA DX, BUFFER

    INT 21H

    ; 统计字符个数

    MOV SI, 1

COUNT\_LOOP:

    MOV AL, BUFFER[SI]

    CMP AL, 0DH ; 判断是否为回车符

    JE END\_COUNT

    CMP AL, 'A' ; 判断是否为字母

    JB CHECK\_DIGIT

    CMP AL, 'Z' ; 判断是否为大写字母

    JBE INCREMENT\_LETTER

    CMP AL, 'a' ; 判断是否为小写字母

    JB CHECK\_DIGIT

    CMP AL, 'z' ; 判断是否为小写字母

    JBE INCREMENT\_LETTER

CHECK\_DIGIT:

    CMP AL, '0' ; 判断是否为数字

    JB INCREMENT\_OTHER ; 若字符小于 '0'，则不是数字

    CMP AL, '9' ; 判断是否为数字

    JBE INCREMENT\_DIGIT ; 若小于等于 '9'，则是数字

INCREMENT\_OTHER:

    INC OTHER ; 增加其他字符计数

    JMP CONTINUE\_COUNT

INCREMENT\_DIGIT:

    INC DIGIT ; 增加数字计数

    JMP CONTINUE\_COUNT

INCREMENT\_LETTER:

    INC LETTER ; 增加字母计数

CONTINUE\_COUNT:

    INC SI ; 移至下一个字符

    JMP COUNT\_LOOP

END\_COUNT:

    ; 输出统计结果

    ; 在此处输出统计结果的代码

    MOV AH, 4CH

    INT 21H

CODES ENDS

    END START

## 第四次试验 查找电话号码

### 1. 设计说明

#### 1.1 程序名

电话簿管理程序

#### 1.2 功能

该程序实现了一个简单的电话簿管理系统，用户可以输入姓名和电话号码，并且可以查询已存储的信息。

#### 1.3 原理及算法说明

程序通过在数组中存储姓名和电话号码，实现了添加和查询的功能。添加信息时，根据姓名的字母顺序插入表中，查询时通过比较姓名找到对应的电话号码。

#### 1.4 程序及数据结构

mess1, mess2, mess3, mess4, mess5, mess6, mess7: 存储输出消息的字符串。

num: 存储已存储信息的数量。

phonein: 标签，用于存储电话号码的相关信息。

pmax: 电话号码的最大长度。

pact: 存储实际输入的电话号码长度。

phone: 存储电话号码的字符数组。

namin: 标签，用于存储姓名的相关信息。

nmax: 姓名的最大长度。

nact: 存储实际输入的姓名长度。

nam: 存储姓名的字符数组。

temp\_nam, temp\_tab: 临时存储姓名和表的字符数组。

tel\_tab: 存储电话簿信息的表。

### 2. 调试说明

#### 

#### 2.1 调试情况

确保输入数据不超过规定大小，防止数组溢出。

检查排序和插入逻辑，确保正确插入和查询。

#### 2.2 解决方法

确保输入数据不超过规定大小。

仔细检查排序和插入逻辑，确保正确处理各种情况。

#### 2.3 观察到的现象及其分析

在输入过长的姓名或电话号码时可能导致数组越界。

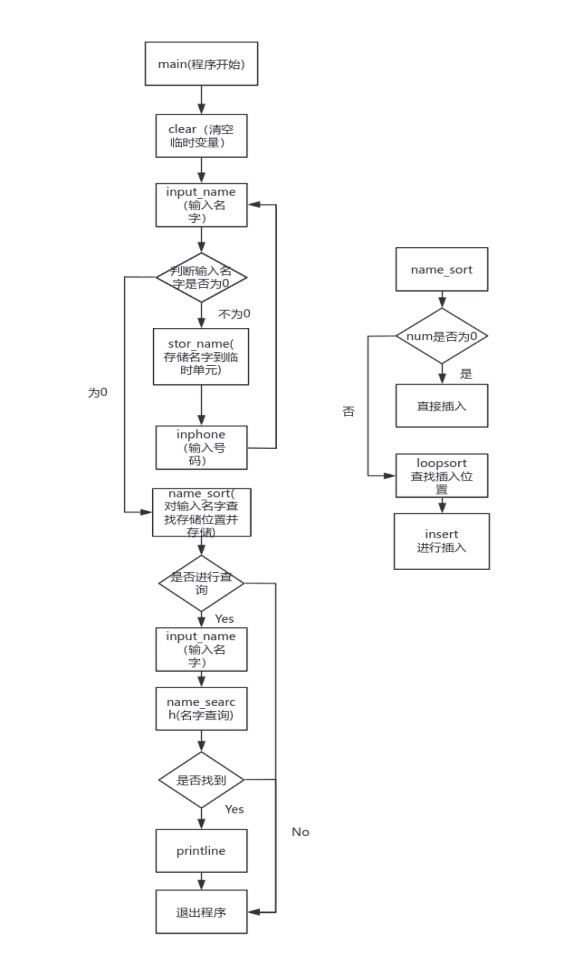
需要确保排序和插入逻辑正确，以确保表中的数据是有序的。

#### 2.4 对程序设计技巧的总结分析

缓冲区大小需要足够大，以容纳用户可能输入的姓名和电话号码。

排序和插入逻辑需要仔细检查，确保正确处理各种情况。

### 程序框图



### 程序清单

DATAS SEGMENT

  mess1 db 'Input name:',13,10,'$'

  mess2 db 'Input a telephone number:',13,10,'$'

  mess3 db 'Do you want a telephone number?(Y/N)',13,10,'$'

  mess4 db 'name?',13,10,'$'

 mess5  db    'name',19 dup(0),'tel.',13,10,'$'

  mess6   db    'Not find!',13,10,'$'

  mess7 db 'wrong',13,10,'$'

  num dw 0;

  phonein label   byte

    pmax  db    9

    pact  db    ?

    phone db    9 dup(?)

  namin   label   byte

    nmax  db    20

    nact  db    ?

    nam   db      20 dup(?)

temp\_nam db 20 dup(?)

temp\_tab db 20 dup(?),4 dup(?),9 dup(?),13,10,'$'

tel\_tab db 50 dup(20 dup (?),4 dup(?),9 dup(?),13,10,'$')

    ;此处输入数据段代码

DATAS ENDS

STACKS SEGMENT

    ;此处输入堆栈段代码

STACKS ENDS

CODES SEGMENT

  main proc far

    ASSUME CS:CODES,DS:DATAS,SS:STACKS

START::

    push ds

    sub ax,ax

    push ax

    MOV AX,DATAS

    MOV DS,AX

    mov es,ax

begin:

     mov ah,09h

     lea dx,mess1

     int 21h

     call clear

     call input\_name;输入名字

     cmp nact,0

     je search;未输入

     call stor\_name

     mov ah,09h

     lea dx,mess2

     int 21h

     call inphone

     call name\_sort

     jmp begin

search:

      mov ah,09h

      lea dx,mess3

      int 21h

      mov ah,01h

      int 21h

      cmp al,'N'

      je exit

      call crlf

      mov ah ,09h

      lea dx, mess4

      int 21h

      call  input\_name

      cmp nact,0

      je exit

      call name\_search

      call printline

      jmp search

exit:

     ret

main endp

input\_name proc  near

    mov   ah,0ah

  lea   dx,namin

  int   21h

  call  crlf

  ret

input\_name    endp

stor\_name    proc    near

    cmp   nact,0

    je    exit1

    lea   si,nam

    lea   di,temp\_tab

    sub   ch,ch

    mov   cl,nact

    cld

    rep   movsb

exit1:

    ret

stor\_name    endp

inphone    proc    near

    mov   ah,0ah

    lea   dx,phonein

    int   21h

    cmp   pact, 8       ; 检查电话号码是否为8位

    je      valid\_phone  ; 如果是8位，跳转到 valid\_phone 标签

    ; 如果不是8位，显示错误消息并跳转到 exit2

    mov ah, 09h

    lea dx, mess7  ; 显示错误消息

    int 21h

    jmp exit2      ; 跳转到退出

valid\_phone:

    call  crlf

    lea   si,phone

    lea   di,temp\_tab

    add     di,23

    sub   ch,ch

    mov   cl,pact

    cld

    rep   movsb

 exit2:

  ret

inphone    endp

name\_sort    proc    near

  cmp   num,0

  jnz   sort

  lea   si,temp\_tab   ;如果表项中没有内容，直接插入

  lea   di,tel\_tab

  mov   cx,35

  cld

  rep   movsb

  jmp   exit3

sort:             ;否则，进行排序

  mov   cx,num

  lea   di,tel\_tab

  lea   si,temp\_tab

loopsort:

  push  di

  push  cx

  mov   cx,20

  repz  cmpsb

  ja    move

  pop   cx

  pop   di

  call  insert

  jmp   exit3   ;插入结束，退出

move:

  pop   cx

  pop   di

  add   di,35

  lea   si,temp\_tab

  loop  loopsort

  mov   cx,35   ;正常退出循环，说明需要插入在最后

  rep   movsb

exit3:

  inc   num     ;表项个数加1

  ret

name\_sort endp

insert proc near

 mov ax,num

loopinsert:

    push ax

    mov bx,35

    mul bx

    lea di,tel\_tab

    add di,ax

    mov si,di

    sub si,35

    push cx

    mov cx,35

    rep movsb

    pop cx

    pop ax

    dec ax

    loop loopinsert

    lea si,temp\_tab

    lea di,tel\_tab

    mov bx,35

    mul bx

    add di,ax

    mov cx,35

    rep movsb

    ret

insert endp

name\_search proc near

    call clear

    mov ch,0

    mov cl,nact

    lea si,nam

    lea di,temp\_nam

    rep movsb

    mov cx,num

    lea di,tel\_tab

    lea si,temp\_nam

loopfind:

    push di

    push cx

    mov ch,0

    mov cl,20

    repz cmpsb

    je find

    pop cx

    pop di

    add di,35

    lea si,temp\_nam

    loop loopfind

    mov cx,0

    jmp exit4

find:

   pop cx

   pop di

exit4:

  ret

name\_search endp

printline proc near

    cmp cx,0

    jnz next

    mov ah,09h

    lea dx,mess6;not find

    int 21h

    jmp exit5

next:

    lea dx,mess5

    mov ah,09h

    int 21h

    mov ax,num

    sub ax,cx

    mov bx,35

    mul  bx

    lea dx,tel\_tab

    add dx,ax

    mov ah,09h

    int 21h

exit5:

  ret

printline endp

crlf  proc  near

  mov   dl,0ah

  mov   ah,02h;换行

  int   21h

  mov   dl,0ah

  mov   ah,02h

  int   21h

  ret

crlf endp

clear   proc  near

  lea   di,temp\_tab

  xor   al,al;设为0

    mov   cx,32

  rep   stosb;清空

  lea   di,temp\_nam

  xor   al,al

    mov   cx,20

  rep   stosb

  ret

clear   endp

    ;此处输入代码段代码

    INT 21H

CODES ENDS

    END START

## 第五次试验 求 Fibonacci 数

### 1. 设计说明

#### 1.1 程序名

斐波那契数列

#### 1.2 功能

该程序通过用户输入一个数字 n，计算并显示斐波那契数列的前 n 项。

#### 1.3 原理及算法说明

程序通过迭代计算斐波那契数列的前 n 项，并在每次计算时考虑到可能产生进位的情况。在计算过程中，采用了两个数组 `n1` 和 `n2` 来保存当前项和前一项的值，通过不断循环迭代，计算得到结果。

#### 1.4 程序及数据结构

`prompt`: 提示用户输入 n 的字符串。

`output`: 显示斐波那契数列的标签的字符串。

`newline`: 换行字符串。

`error\_prompt`: 错误提示字符串，当输入不在 0 100 范围内时使用。

`number`: 用户输入的数字的相关信息。

`max`: 最大长度。

`act`: 实际输入的长度。

`num`: 存储输入数字的字符数组。

`n1`: 保存当前项的数组。

`n2`: 保存前一项的数组。

`len`: 保存数组长度。

`add\_time`: 保存进位的计数。

### 2. 调试说明

#### 

#### 2.1 调试情况

确保输入数据不超过规定大小。

检查斐波那契数列计算逻辑，确保正确计算结果。

处理进位的逻辑是否正确。

#### 2.2 解决方法

确保输入数据不超过规定大小。

仔细检查斐波那契数列计算的逻辑，确保正确处理各种情况。

#### 2.3 观察到的现象及其分析

在输入过长的数字时可能导致数组越界。

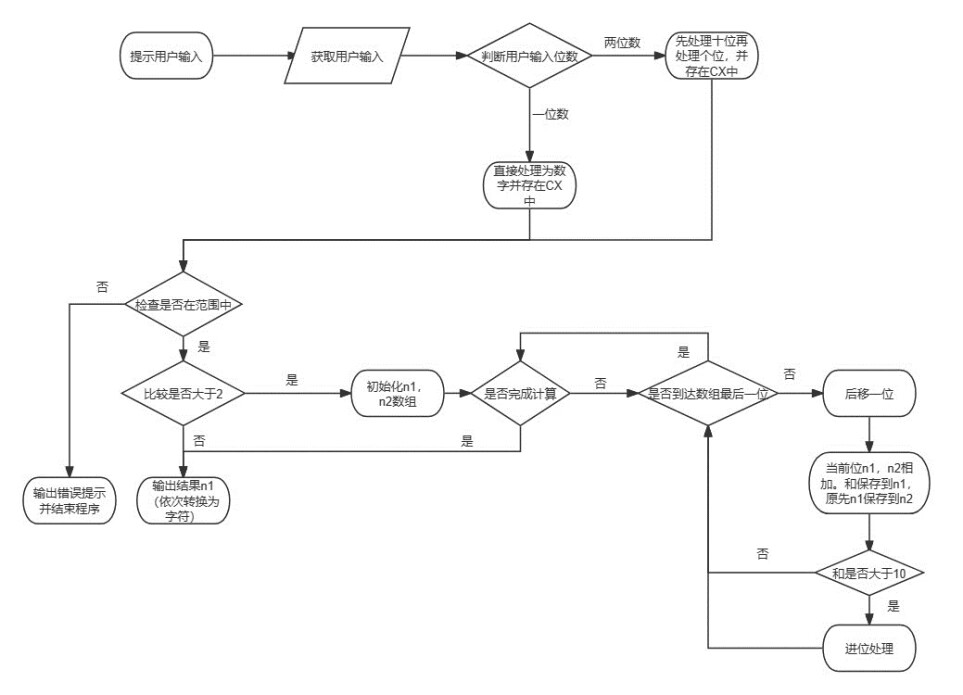
需要确保斐波那契数列计算的逻辑正确，特别是在涉及进位的情况。

#### 2.4 对程序设计技巧的总结分析

缓冲区大小需要足够大，以容纳用户可能输入的数字。

斐波那契数列计算逻辑需要仔细检查，确保正确处理各种情况。

### 3. 程序框图



### 4. 程序清单

data segment

    prompt db 'Please input number n: ', 13, 10 ,'$' ; 提示用户输入n

    output db 'fibonacci(n): ', 13, 10, '$' ; 显示斐波那契数列的标签

    newline db 13, 10, '$'

    error\_prompt db 'Error input out of 0 100','$'

    number label byte

        max db 3

        act db ?

        num db 3 dup(?)

    n1  dw 22 dup(0)

    n2  dw 22 dup(0)

    len dw 1

    add\_time dw 1 dup(0)

data ends

code segment

main    proc    far

    assume cs:code, ds:data

start::

    push    ds

    xor     ax, ax

    push    ax

    mov     ax, data

    mov     ds, ax

    lea     dx, prompt

    mov     ah, 09

    int     21h

    lea     dx, number

    mov     ah, 0ah

    int     21h

    mov ah, 9

    lea dx, newline

    int 21h

    lea     si, num

    xor     cx, cx

    xor     ax, ax

    xor     bx, bx

    cmp     act, 2

    je      two

one:

    mov     ah, 0

    mov     al, ds:[si]

    sub     al, 48          ; 将 ASCII 字符转换为数字

    mov     cx, ax

    jmp     check

two:

    mov     al, 10

    mov     ah, ds:[si]

    sub     ah, 48

    mul     ah              ; 将十位乘以 10(al)保存在al中

    mov     ah, 0

    add     al, ds:[si+1]   ; 将个位的数字（ASCII形式）加到十位数上(10\*十位数+个位数+‘0’)

    sub     al, 48          ; 将 ASCII 字符转换为数字

    mov     cx, ax          ; 将数字保存在 CX 寄存器中

check:

    cmp     cx, 1

    jb      error\_exit

    cmp     cx, 99

    jbe     compare

error\_exit:

    mov     ah, 9

    lea     dx, newline

    int     21H

    mov     ah, 9

    lea     dx, error\_prompt

    int     21H

    jmp     exit

compare:

    xor     ax, ax

    xor     bx, bx

    lea     ax, n1

    lea     bx, n2

    mov     si, ax

    mov     di, bx

    push    ax

    xor     ax, ax

    mov     ax, 1

    mov     ds:[si], ax

    mov     ds:[di], ax

    pop     ax              ; 将栈中的值弹出到 AX 寄存器(即此前保存的AX）

    ; 判断n>1&&n>2

    cmp     cx, 2

    jbe     print

    sub     cx, 2           ; cx是进行斐波那契数列计算的次数

fibonacci:

    ; 已经确定n>2

    push    cx

    mov     si, ax

    mov     di, bx

    push    ax

    ; 计算下一项的斐波那契数列值

    mov     ax, len

    shr     ax, 1        ; 右移一位，相当于除以 2

    mov     cx, ax

    add     cx, add\_time ; 加上上次迭代时的进位值

    xor     ax, ax

    mov     add\_time, ax ; 清零上次迭代的进位值

    pop     ax

\_add:

    push    ax

    push    bx

    mov     ax, ds:[si] ; 获取当前项的值

    mov     bx, ds:[di] ; 获取前一项的值

    add     bx, ax

    mov     ds:[si], bx

    mov     ds:[di], ax ; 将计算前的当前项保存到前一项

    mov     bx, 10

    cmp     ds:[si], bx

    pop     bx

    pop     ax

    jb      less\_than\_10 ; 如果小于 10，跳转到标签 less\_than\_10

bigger\_than\_10:

    mov     dx, ds:[si] ; 获取当前项的值

    push    ax

    mov     ax, 10

    sub     ds:[si], ax ; 当前项减去 10

    xor     ax, ax

    mov     ax,1

    add     ds:[di+2], ax ; 将进位加到下一位

    pop     ax

    add     si, 2

    add     di, 2

    add     add\_time, 1   ; 递增进位计数

    mov     len, si

    sub     len, ax

    loop    \_add         ; 继续下一次迭代

    jmp     addr\_change   ; 跳转到地址调整处

less\_than\_10:

    add     si, 2

    add     di, 2

    mov     len, si

    sub     len, ax

    cmp     len,44

    jne     \_add

addr\_change:

    ; 调整地址，准备下一次迭代

    pop     cx

    loop    fibonacci   ; 继续下一次迭代

print:

    xor     dx, dx

    lea     dx, output ; 将输出字符串的地址加载到 dx 寄存器

    mov     ah, 09

    int     21h

    ; 输出计算结果

    xor     dx, dx

    xor     bx, bx

    mov     cx, 44

    shr     cx, 1

    sub     si, 2    ; 将 si 寄存器移动到最后一个有效数字

    print\_num:

        ; 将数字转化为 ASCII 码，并显示

        mov     dx, ds:[si] ; 获取当前项的值

        remove\_head\_zero:

        cmp     bx, 0

        jne     convert\_num

        cmp     dx, 0

        je      move\_forward

        convert\_num:

        mov     bx, 1

        add     dx, 48 ; 重新转化为ASCII十进制

        mov     ah, 02

        int     21h

        move\_forward:

        sub     si, 2        ; 移动到前一位

        loop    print\_num   ; 继续下一次迭代

    mov     dl, 13 ; 回车

    mov     ah, 02

    int     21h

exit:

    mov     ax, 4c00h

    int     21h

main endp

code ends

end start