汇编程序设计期末报告

文本查看器的改进

1 分析程序

1.1 源程序分析

具体分析直接写为代码注释

1.1.1 show.asm

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 文件:    show.asm                                          ;
; 作者:    xuhongxu.com 改编自汇编程序设计大作业              ;
; 日期:    2016.12.21                                        ;
; 修改:    2016.12.21                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PAGE
TITLE       SHOW

DOSSEG
.MODEL      SMALL

INCLUDE     DOS.INC
INCLUDE     BIOS.INC

.STACK      100h

.DATA

; 状态栏

PUBLIC      statline, linenum, statline_l
statline    DB      " Line:      "                    ; 状态栏行号提示
statfile    DB      " File:              "           ; 状态栏文件名提示
stathelp    DB      " Quit:Esc  Etr:Fnd  Move:  PGUP PGDN HOME END"   ; 状态栏帮助提示
statline_l  DW      statfile - statline - 7           ; 状态栏行号空位长
linenum     DW      1                                 ; 行号

; 搜索栏
```

```
PUBLIC      findline, findstat, findstr, findlen, matchn, findmatch
findstat    DB      0                                   ; 搜索栏是否可见
findline    DB      " Find Text:", 50 dup(" "), "<< "   ; 搜索栏提示
findmatch   DB      "0      matched "                   ; 搜索匹配
findstr     DW      60 dup (0)                          ; 搜索缓冲区
findlen     DW      0                                   ; 搜索缓冲大小
matchn      DW      0

; 屏幕显示控制变量

PUBLIC      cell, rows, columns, vidadr, statatr, scrnatr, cga, hltatr, hltline, hltpos
cell        LABEL   WORD                    ; 屏幕显示单元（字符和配色）
char        DB      " "                     ; | 字符：初始为空格
attr        DB      ?                       ; | 属性

columns     EQU     80                      ; 列数
rows        DW      24                      ; 行数
mode        DB      ?                       ; 显示模式
pag         DB      ?                       ; 显示页
newvid      DB      0                       ; 显示交换标识
cga         DB      1                       ; CGA 标识（默认真）

vidadr      DW      0B800h                  ; 显示缓冲地址（默认 CGA）
mono        EQU     0B000h                  ; 单色地址

hltatr      DB      071h                    ; 高亮配色：红底白字
statatr     DB      030h                    ; 状态栏默认配色：蓝绿底黑字
bwstat      EQU     070h                    ; 白底黑字配色
scrnatr     DB      017h                    ; 屏幕默认配色：蓝底白字
bwscrn      EQU     007h                    ; 黑底白字配色

hltline     DW      0
hltpos      DW      0

; 缓冲和文件控制变量

PUBLIC      buffer, pbuffer, sbuffer, fsize, namebuf
buffer      LABEL   DWORD
pbuffer     DW      0                       ; 缓冲偏移量
sbuffer     DW      ?                       ; 缓冲段基址
lbuffer     DW      ?                       ; 缓冲大小
fhandle     DW      ?                       ; 文件句柄
fsize       DW      ?                       ; 文件大小
```

```
prompt      DB      13, 10, 13, 10, "Enter filename: $"
prompt2     DB      13, 10, "File problem. Try again? $"
namebuf     DB      66, ?
filename    DB      66 dup(0)               ; 文件名缓冲

err1        DB      13, 10, "Must have DOS 2.0 or higher", 13, 10, "$"
err2        DB      13, 10, "File too big", 13, 10, "$"

; 函数调用表

exkeys      DB      71, 72, 73, 79, 80, 81  ; 扩展键盘扫描码
lexkeys     EQU     $ - exkeys              ; 键表长
extable     DW      homek
            DW      upk
            DW      pgupk
            DW      endk
            DW      downk
            DW      pgdnk
            DW      nonek

.CODE

EXTRN       pager:PROC, isEGA:PROC, ShowFind:PROC, ShowKey:PROC

start:      mov     ax, @DATA                ; 初始化数据段为 DATA
            mov     ds, ax

            cli                              ; 关闭中断
            mov     ss, ax                   ; 初始化堆栈段为 DATA
            mov     sp, OFFSET STACK         ; 初始化堆栈指针
            sti                              ; 开启中断

; 调整内存分配

            mov     bx, sp
            mov     cl, 4
            shr     bx, cl
            add     ax, bx
            mov     bx, es
            sub     ax, bx
            @ModBlok    ax

; 为文件缓冲分配动态内存空间
```

```
            @GetBlok    0FFFh                   ; 尝试分配 64K
            mov     sbuffer, ax                 ; 存储缓冲的段基址
            mov     lbuffer, bx                 ; 存储实际分配大小

; 检查 DOS 版本

            @GetVer
            cmp     al, 2                       ; 比较 2.0
            jge     video                       ; >= 2.0，可以
            @DispStr    err1                    ; 否则报错
            int     20h

; 设置显示模式

; 调用子程序：isEGA -> al
; 判断 EGA 是否激活，未激活（非 EGA）返回屏幕行数，否则返回 0

video:      call    isEGA                       ; 判断 EGA 还是 VGA
            or      ax, ax                      ; 若为 0，则是 CGA（或 MA）
            je      modechk                     ; 为 0，跳到 modechk
            mov     rows, ax                    ; 否则加载行数
            dec     cga                         ; 不是 CGA

modechk:    @GetMode                            ; 获取显示模式
            mov     mode, al                    ; 存储显示模式
            mov     pag, bh                     ; 存储页
            mov     dl, al                      ; 复制
            cmp     dl, 7                       ; 7 模式（单色）
            je      loadmono                    ; 是，跳转到 loadmono
            cmp     dl, 15                      ; 15 模式（单色）
            jne     graphchk                    ; 不是，跳转到 graphchk
loadmono:   mov     vidadr, mono                ; 加载单色地址
            mov     statatr, bwstat             ; 更换配色为单色方案
            mov     scrnatr, bwscrn
            dec     cga                         ; 不是 CGA
            cmp     al, 15                      ; 15 模式
            jne     cmdchk                      ; 不是 15 而是 7，结束，跳到 cmdchk
            mov     dl, 7                       ; 否则，设模式为 7
            jmp     SHORT chmod
graphchk:   cmp     dl, 7                       ; 与 7 比
            jg      color                       ; >7 则跳到 color
            cmp     dl, 4                       ; 与 4 比
            jg      bnw                         ; 5，6 一般为黑白
            je      color                       ; 4 是彩色
```

```
            test    dl, 1                       ; 奇数测试
            jz      bnw                         ; 0, 2 是黑白
color:      cmp     dl, 3                       ; 与 3 比
            je      cmdchk                      ; =3，结束，跳到 cmdchk
            mov     dl, 3                       ; 否则，设模式为 3
            jmp     SHORT chmod
bnw:        mov     statatr, bwstat             ; 更换配色为单色方案
            mov     scrnatr, bwscrn             ; 屏幕黑底白字配色
            cmp     dl, 2                       ; 与 2 比
            je      cmdchk                      ; =2，结束，跳到 cmdchk
            mov     dl, 2                       ; 否则设置模式为 2
chmod:      @SetMode    dl                      ; 设置显示模式
            @SetPage    0                       ; 设置页
            mov     newvid, 1                   ; 设置标识

; 打开命令行文件

cmdchk:     mov     bl, es:[80h]                ; 获取长度
            sub     bh, bh
            mov     WORD PTR es:[bx + 81h], 0   ; 添加字符串终止符'\0'
            push    ds
            @OpenFil    82h, 0, es              ; 打开文件
            pop     ds
            jc      getname                     ; 如果错误，重新获取文件名
            mov     fhandle, ax                 ; 否则，保存文件句柄
            push    ds
            @GetFirst   82h, , es               ; 找到文件名
            pop     ds
            jnc     opened                      ; 文件成功打开

; 获取文件名

getname:    @DispStr    prompt                  ; 显示提示
            @GetStr     namebuf, 0              ; 获取输入文本
            @OpenFil    filename, 0             ; 打开文件
            jc      badfile                     ; 如果错误，转到 badfile
            mov     fhandle, ax                 ; 保存文件句柄
            @GetFirst   filename                ; 找到文件名
            jnc     opened                      ; 文件成功打开

badfile:    @DispStr    prompt2                 ; 打开错误，提示是否再试
            @GetKey     0, 1, 0
            and     al, 11011111b               ; 转换输入为大写
            cmp     al, "Y"                     ; 判断输入是否为 Y
```

```
            je      getname                    ; 是 Y，重试
            jmp     quit                       ; 否则，退出

; 复制文件名到状态栏

opened:     mov     si, 9Eh                    ; 加载 FCB
            mov     di, OFFSET statfile[7]     ; 加载状态栏 文件名
            mov     al, es:[si]                ; 加载第一个字节
            inc     si
copy:       mov     [di], al                   ; 存储并加载字节，直到 0
            inc     di
            mov     al, es:[si]
            inc     si
            or      al, al                     ; 检查是否为 0
            loopne  copy                       ; 不是 0 继续复制

; 检查文件大小

            @GetFilSz   fhandle                ; 获得文件大小
            or      dx, dx                     ; 比 64K 大?
            jne     big                        ; 是的，那就太大了
            mov     fsize, ax                  ; 存储文件大小
            mov     cx, 4                      ; 右移四位
            shr     ax, cl
            cmp     ax, lbuffer                ; 是否比缓冲区大
            jle     fileread                   ; 不大，跳到 fileread
big:        @DispStr    err2                   ; 文件过大错误
            @Exit       2
fileread:   push    ds
            @Read   buffer, fsize, fhandle     ; 读取文件
            pop     ds
            jnc     readok                     ; 没有读取错误
            jmp     getname                    ; 否则重新获取文件

; 存储文件大小

readok:     mov     di, ax                     ; 加载文件大小
            push    es
            mov     es, sbuffer                ; 加载缓冲区段
            std                                ; set di = 1
            mov     cx, 0FFh                   ; 查找 0FFh（最大长度）个字符
            mov     al, 1Ah                    ; 查找 EOF（1Ah）
            repne   scasb
            cld
```

```
            jcxz    noeof                   ; 如果没有 EOF，跳到 noeof
            inc     di                      ; 否则存储文件大小
            mov     fsize, di
noeof:      pop     es
            @SetCurPos  0, 43               ; 移除光标

; 调用子程序：Pager (0)
; 显示第一页

            xor     ax, ax                  ; ax = 0
            push    ax                      ; 参数 1
firstpg:    call    Pager

; 处理键盘事件

nextkey:    @GetKey     0, 0, 0             ; 获取按键
nextkey2:   cmp     al, 0                   ; 是否为空
            je      extended                ; 如果是，则为扩展键，跳到 extended
            cmp     al, 13                  ; 是否为 Enter
            jne     nextkey3                ; 不是，跳到 Esc 判断
            call    findk                   ; 是 Enter，调用 findk
            jmp     nextkey
nextkey3:   cmp     al, 27                  ; 是否为 Esc
            je      quit                    ; 是 Esc，跳到 quit
            call    ShowKey                 ; 调用显示 key 子程序，为搜索栏显示
            jmp     nextkey
quit:       @ClosFil    fhandle             ; 是 Esc，关闭文件
            @FreeBlok   sbuffer             ; 释放缓冲区
            cmp     newvid, 1               ; 显示模式是否改变
            jne     thatsall                ; 没改，跳到 thatsall
            @SetMode    mode                ; 还原显示模式
            @SetPage    pag                 ; 还原页
thatsall:   mov     dx, rows                ; 加载行数
            xchg    dl, dh                  ; dh = 行数
            mov     cx, dx                  ; cx = dx
            mov     dl, 79                  ; dl = 79
            @Scroll     0                   ; 滚动到新行
            sub     dl, dl
            @SetCurPos                      ; 设置光标
            @Exit       0
extended:   @GetKey     0, 0, 0             ; 获取扩展码
            push    es
            push    ds
            pop     es
```

```
            mov     di, OFFSET exkeys           ; 加载键表的地址和长度
            mov     cx, lexkeys + 1
            repne   scasb                       ; 找到位置
            pop     es
            sub     di, (OFFSET exkeys) + 1     ; 指向键码
            shl     di, 1                       ; 调整指针为字地址
            call    extable[di]
            jmp     nextkey

; 键处理

findk:      call    ShowFind
            retn

homek:      mov     pbuffer, 0
            push    pbuffer
            mov     linenum, 1
            call    Pager
            retn

upk:        mov     ax, -1
            push    ax
            call    Pager
            retn

pgupk:      mov     ax, rows
            neg     ax
            push    ax
            call    Pager
            retn

endk:       mov     ax, fsize
            mov     pbuffer, ax
            mov     linenum, -1
            mov     ax, rows
            neg     ax
            push    ax
            call    Pager
            retn

downk:      mov     ax, 1
            push    ax
            call    Pager
            retn
```

```
pgdnk:      push    rows
            call    Pager
            retn

nonek:      retn

            END     start
```

1.1.2 pager.asm

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 文件:    pager.asm                                          ;
; 作者:    xuhongxu.com 改编自汇编程序设计大作业               ;
; 日期:    2016.12.20                                         ;
; 修改:    2016.12.21                                         ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PAGE        60, 132
.MODEL      SMALL

INCLUDE     DOS.INC
INCLUDE     BIOS.INC

.DATA

EXTRN       statatr:BYTE,   scrnatr:BYTE,   sbuffer:WORD,   pbuffer:WORD
EXTRN       fsize:WORD,     cell:WORD,      statline:BYTE,  linenum:WORD
EXTRN       rows:WORD,      vidadr:WORD,    cga:BYTE,       findline:BYTE
EXTRN       findstat:BYTE,  statline_l:WORD,findstr:BYTE,   findlen:WORD
EXTRN       matchn:WORD,    findmatch:BYTE, hltatr:BYTE,    hltline:WORD
EXTRN       hltpos:WORD

.CODE
PUBLIC      Pager, isEGA, ShowFind, ShowKey

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序:    Pager
; 功能:    显示状态栏和文本行
; 参数:    栈 - 滚动的行数 n
; 输出:    无

Pager       PROC
            push    bp
            mov     bp, sp
```

```
            mov     es, sbuffer             ; es = sbuffer 缓冲段基址
            mov     di, pbuffer             ; di = pbuffer 缓冲偏移量

            mov     cx, [bp + 4]            ; 栈参数 n


            mov     ax, 10                  ; 搜索换行符 Ascii 10 (LF: Linefeed)
                                            ; GoBack/GoForwd 滚动时会涉及换行符搜索

            or      cx, cx                  ; 判断滚动方向
            jg      forward
            jl      backward
            jmp     SHORT show              ; 不滚动，直接显示

; 调用子程序：GoBack (cx - 滚动行数，es:di - 缓冲位置)

backward:   call    GoBack                  ; 向上滚
            jmp     SHORT show

; 调用子程序：GoForwd (cx - 滚动行数，es:di - 缓冲位置)

forward:    call    GoForwd                 ; 向下滚

; 显示

show:       cld
            push    di
            push    es
            push    ds
            pop     es                      ; DS->ES

; 调用子程序：BinToStr (linenum, OFFSET statline[7]) -> ax
; 将行号转换为文本，长度存入 ax

            push    linenum                 ; 参数 1
            mov     ax, OFFSET statline[7]  ; 参数 2
            push    ax
            call    BinToStr                ; 二进制到文本

; 初始化状态栏填充文本

            mov     cx, statline_l          ; 行号的空位数
            sub     cx, ax                  ; 填充空格数 cx = 空位数 - 行号文本长度
```

```
            mov     al, " "
            rep     stosb                   ; 填充空格
            pop     es

            mov     bl, statatr             ; 加载状态栏配色
            mov     BYTE PTR cell[1], bl

; 调用子程序：CellWrt (ds, OFFSET statline, 0, cell)
; 指定配色写入第 0 行，即状态栏

            push    ds                      ; 参数 1
            mov     ax, OFFSET statline     ; 参数 2
            push    ax
            sub     ax, ax                  ; 参数 3
            push    ax
            push    cell                    ; 参数 4
            call    CellWrt

; 初始化内容填充

            pop     di
            mov     bl, scrnatr             ; 加载内容配色
            mov     BYTE PTR cell[1], bl

            mov     si, di                  ; 缓冲偏移量
            mov     cx, rows                ; cx = 行数

show1:      mov     bx, rows                ; bx = 行数
            inc     bx                      ; bx++
            sub     bx, cx                  ; bx -= cx 即当前行
            push    cx                      ; 保存行数

; 调用子程序：CellWrt (sbuffer, pbuffer, line, cell) -> ax
; 指定配色，将缓冲区内容写入当前行，返回写入后的缓冲偏移量

            push    sbuffer                 ; 参数 1
            push    si                      ; 参数 2
            push    bx                      ; 参数 3
            push    cell                    ; 参数 4
            call    CellWrt

            pop     cx                      ; 还原行数
            mov     si, ax                  ; 更新缓冲偏移量
```

```
            cmp     ax, fsize               ; 是否读到文件尾
            jae     fillout                 ; 是，跳到 fillout，用空格填充剩余
            loop    show1                   ; 否则，继续写入下一行，cx--
            jmp     SHORT pagedone          ; 结束则跳到 pagedone

; 空格填充剩余

fillout:    dec     cx                      ; 计算剩余行数
            jcxz    pagedone                ; 为 0，结束，跳到 pagedone

            ; 列数 * 剩余行数

            mov     al, 80
            mul     cl

; 调用子程序：CellFil (sbuffer, count, cell)
; 指定配色，缓冲区填充指定数目空格

            push    sbuffer                 ; 参数 1
            push    ax                      ; 参数 2
            push    cell                    ; 参数 3
            call    CellFil

pagedone:   @SetCurPos  0, 43               ; 移除光标
            mov     findstat, 0             ; 搜索栏不可见
            pop     bp
            ret     2
Pager       ENDP



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    CellWrt (segment, offset, line, cell)
; 功能：    写入一行
; 参数：    栈 - 1. 缓冲所在段；2. 缓冲偏移量；3. 行号；4. 配色
; 输出：    ax - 写入后的缓冲偏移量

CellWrt     PROC
            push    bp
            mov     bp, sp

            push    ds
            sub     dx, dx
            cmp     cga, 1                  ; CGA?
            jne     noscan                  ; 不是 CGA，noscan
```

```
            mov     dx, 03DAh               ; 加载端口

noscan:     mov     es, vidadr              ; 加载屏幕缓冲段
            mov     ds, [bp + 10]           ; ds = 缓冲所在段
            mov     si, [bp + 8]            ; si = 缓冲偏移量
            mov     cx, 80                  ; 列数
            mov     ax, [bp + 6]            ; 行号
            mov     bx, 80 * 2              ; 每行字节数
            mul     bl                      ; 行号 * 每行字节数 = 起始字节偏移
            mov     di, ax                  ; di = 起始字节偏移
            mov     bx, di                  ; bx = 起始字节偏移
            mov     ax, [bp + 4]            ; 配色
movechar:   lodsb                           ; 从文件缓冲区（ds:si）取字符到 al
            cmp     al, 13                  ; 是否为回车符
            je      fillspc                 ; 是回车符，该行内容结束，跳到 fillspc 填充空格
            cmp     al, 9                   ; 是否为 Tab
            jne     notab                   ; 不是 Tab，跳到 notab

; 调用子程序：FillTab
; 填充 Tab 空格

            call    FillTab

            jcxz    nextline                ; 如果超过列数，结束本行，显示下一行
            jmp     SHORT movechar          ; 否则，继续显示本行下一字符

notab:      or      dx, dx                  ; CGA?
            je      notab2                  ; 不是 CGA，跳到 notab2

; 调用子程序：Retrace
; 是 CGA 时，使用 Retrace

            call    Retrace

            loop    movechar                ; 继续显示本行下一字符
            jmp     SHORT nextline          ; 显示下一行

notab2:     stosw                           ; 将字符 ax 存到屏幕缓冲区（es:di）
            loop    movechar                ; 继续显示本行下一字符
            jmp     SHORT nextline          ; 显示下一行

fillspc:    mov     al, " "                 ; 该行结束，为剩余列填充空格
            or      dx, dx                  ; CGA?
            je      space2                  ; 不是 CGA，跳到 space2
```

```
; 调用子程序：Retrace
; 是 CGA 时，使用 Retrace

space1:     call    Retrace
            loop    space1                  ; 继续填充空格
            inc     si                      ; 缓冲偏移量++
            jmp     SHORT exit              ; 结束

space2:     rep     stosw                   ; 写空格
            inc     si                      ; 下一位置
            jmp     SHORT exit              ; 结束

nextline:   mov     ah, 10                  ; 搜索换行符
chklf:      lodsb                           ; 文件缓冲区（ds:si）读取字符到 al
            cmp     al, ah                  ; 比较是否是换行符
            loopne  chklf                   ; 不是就继续找
exit:       mov     ax, si                  ; 返回 ax = 写入后的偏移量
            pop     ds
            pop     bp
            ret     8
CellWrt     ENDP



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    CellFil (segment, count, cell)
; 功能：    向缓冲区填充指定指定数目字符
; 参数：    栈 - 1. 缓冲所在段；2. 数目；3. 配色和字符（高位和低位）
; 输出：    ax - 写入后的缓冲偏移量

CellFil     PROC
            push    bp
            mov     bp, sp

            push    ds
            sub     dx, dx
            cmp     cga, 1                  ; CGA?
            jne     noscan2                 ; 不是 CGA，noscan
            mov     dx, 03DAh               ; 加载端口

noscan2:    mov     es, vidadr              ; 加载屏幕缓冲段
            mov     ds, [bp + 8]            ; 缓冲所在段
            mov     cx, [bp + 6]            ; 填充的数目
            mov     ax, [bp + 4]            ; 配色和字符
```

```
            or      dx, dx                  ; CGA?
            je      fillem2                 ; 不是，跳到 fillem2

; 调用子程序：Retrace
; 是 CGA 时，使用 Retrace

fillem1:    call    Retrace

            loop    fillem1                 ; 重复填充
            jmp     SHORT filled            ; 填充完毕，结束跳到 filled
fillem2:    rep     stosw                   ; 写 ax 到屏幕缓冲区（es:di）
filled:     pop     ds
            pop     bp
            ret     6
CellFil     ENDP



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    FillTab
; 功能：    填充 Tab
; 参数：    bx - 指向行首位置，di - 指向当前位置，cx - 剩余列数
; 输出：    cx - 剩余列数

FillTab     PROC
            push    bx
            push    cx

            ; 计算行内偏移（除以 2 是因为一个显示的字符占两个字节）存入 bx
            sub     bx, di                  ; bx = bx - di
            neg     bx                      ; bx = -bx
            shr     bx, 1                   ; bx /= 2

            mov     cx, 4                   ; 默认 Tab 为 4 个空格
            and     bx, 3                   ; bx = bx mod 4，即 Tab 占用位数
            sub     cx, bx                  ; cx -= bx，即 Tab 剩余空位
            mov     bx, cx                  ; 存储 cx 到 bx

            mov     al, " "                 ; 空格字符
            or      dx, dx                  ; CGA?
            je      tabem2                  ; 不是，跳到 tabem2

; 调用子程序：Retrace
; 是 CGA 时，使用 Retrace
```

```
tabem1:     call    Retrace
            loop    tabem1                  ; 重复填充
            jmp     SHORT tabbed            ; 填充完毕，结束跳到 tabbed
tabem2:     rep     stosw                   ; 写 ax 到屏幕缓冲区（es:di）
tabbed:     pop     cx                      ; cx 此时是列数
            sub     cx, bx                  ; cx -= bx
            jns     nomore                  ; 如果无符号，说明 cx > bx，跳到 nomore
            sub     cx, cx                  ; 否则，cx 为负，该列已经到头了，清空 cx
nomore:     pop     bx
            ret
FillTab     ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    GoBack
; 功能：    向上搜索缓冲
; 参数：    cx - 行数（向上所以是负数），es:di - 缓冲位置
; 输出：    无

GoBack      PROC
            std                         ; set df = 1，向上
            neg     cx                  ; cx 变正
            mov     dx, cx              ; 保存 cx 到 dx
            inc     cx                  ; 加一步
            or      di, di              ; 判断是否为文件头（di=0）
            je      exback              ; di = 0，是文件头，跳到 exback
findb:      push    cx                  ; 不是文件头，保存 cx
            mov     cx, 0FFh            ; 查找 0FFh（最大文本长度）个字符
            cmp     cx, di              ; 是否超出缓冲位置
            jl      notnear             ; 没有，那继续
            mov     cx, di              ; 超出了，修改加载字符数为
notnear:    repne   scasb               ; 找到前一个 LF
            jcxz    atstart             ; 如果没找到，就去开头
            pop     cx                  ; 还原行数
            loop    findb               ; 再上一行
            cmp     linenum, 0FFFFh     ; EOF?
            jne     notend              ; 没有，继续
            add     di, 2               ; 跳过 CR/LF
            mov     pbuffer, di         ; 存储位置
            call    EndCount            ; 计算行号
            ret
notend:     sub     linenum, dx         ; 计算行号
            jg      positive
            mov     linenum, 1          ; 行号为负数，设为 1
positive:   add     di, 2               ; 跳过 CR/LF
```

```
            mov     pbuffer, di             ; 存储位置
            ret
atstart:    pop     cx
            sub     di, di                  ; di = 0，到文件开头
            mov     linenum, 1              ; 设行为 1
            mov     pbuffer, di             ; 存储位置
exback:     ret
GoBack      ENDP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    GoForwd
; 功能：    向下搜索缓冲
; 参数：    cx - 行数（向上所以是负数），es:di - 缓冲位置
; 输出：    无

GoForwd     PROC
            cld                             ; set df = 0，向下
            mov     dx, cx                  ; 保存 cx 到 dx
findf:      push    cx                      ; 保存 cx
            mov     cx, 0FFh                ; 查找 0FFh（最大文本长度）个字符
            repne   scasb                   ; 找到下一个 LF
            jcxz    atend                   ; 没找到，已经在文件尾
            cmp     di, fsize               ; 是否超出结尾
            jae     atend                   ; 超出，已经在文件尾
            pop     cx
            loop    findf                   ; 继续下一行
            add     linenum, dx             ; 计算行号
            mov     pbuffer, di             ; 存储位置
            ret
atend:      pop     cx                      ; 已在文件尾，不做事情
            mov     di, pbuffer             ; 还原位置
            ret
GoForwd     ENDP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    EndCount
; 功能：    向上计算行数
; 参数：    es:di - 缓冲位置
; 输出：    无

EndCount    PROC
            push    di
```

```
            mov     al, 13                  ; 搜索 CR
            mov     linenum, 0              ; 从 0 行开始

findstrt:   inc     linenum                 ; 下一行
            mov     cx, 0FFh                ; 查找 0FFh（最大文本长度）个字符
            cmp     cx, di                  ; 超出缓冲位置
            jl      notnear2                ; 没超出，继续
            mov     cx, di                  ; 超出，只搜到缓冲位置
notnear2:   repne   scasb                   ; 向上找 CR
            jcxz    found                   ; 没找到 CR，那就一直到开头
            jmp     SHORT findstrt          ; 找到，继续上一行
found:      pop     di
            ret
EndCount    ENDP



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    isEGA
; 功能：    判断 EGA 是否激活
; 参数：    无
; 输出：    al - 0 为激活（是 EGA），未激活则返回屏幕行数

isEGA       PROC
            push    bp
            push    es
            mov     ah, 12h                 ; 获取 EGA 状态
            mov     bl, 10h
            sub     cx, cx                  ; 清除状态位
            int     10h
            sub     ax, ax                  ; 清空返回值
            jcxz    noega                   ; 如果状态位仍为 0，没有 EGA

            mov     es, ax                  ; es = 0
            test    BYTE PTR es:[487h], 1000b   ; 测试激活位
            jnz     noega                   ; 如果不为 0，未激活
            mov     ax, 1130h               ; 获取 EGA 信息
            int     10h
            mov     al, dl                  ; 返回行数
            cbw
noega:      pop     es
            pop     bp
            ret
isEGA       ENDP
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序:    BinToStr
; 功能:    转换整数到文本
; 参数:    栈 - 1. 欲转换数值, 2. 写入地址
; 输出:    ax - 字符数

BinToStr    PROC
           push    bp
           mov     bp, sp
           mov     ax, [bp + 6]           ; 参数 1
           mov     di, [bp + 4]           ; 参数 2

           sub     cx, cx                 ; 清除 cx
           mov     bx, 10                 ; bx = 10
getdigit:   sub     dx, dx                 ; dx = 0
           div     bx                     ; 除以 10
           add     dl, "0"                ; 余数转换为 Ascii
           push    dx                     ; 字符存到栈
           or      ax, ax                 ; 商是否为 0
           loopnz  getdigit               ; 不为 0, 继续转换

           neg     cx                     ; cx = -cx, 即长度
           mov     dx, cx
putdigit:   pop     ax                     ; 得到字符
           stosb                          ; 存储字符到指定地址
           loop    putdigit
           mov     ax, dx                 ; 返回字符数

           pop     bp
           ret     4
BinToStr    ENDP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序:    Retrace
; 功能:    CGA 模式下写字符
; 参数:    es:di - 屏幕缓冲偏移, ax - 配色和字符
; 输出:    无

Retrace     PROC
           push    bx
           mov     bx, ax                 ; 保存字符
```

```
lscan2:     in      al, dx                ; 找到端口
            shr     al, 1                 ; 直到低位
            jc      lscan2
            cli
hscan2:     in      al, dx                ; 保存字符
            shr     al, 1                 ; 直到低位
            jnc     hscan2
            mov     ax, bx                ; 还原并写入
            stosw
            sti
            pop     bx
            ret
Retrace     ENDP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    ShowFind
; 功能：    切换搜索栏
; 参数：    无
; 输出：    无

ShowFind    PROC
            push    bp
            mov     bp, sp

            push    dx

            mov     dl, findstat          ; 搜索栏是否可见
            or      dl, dl
            je      tofind                ; 为0不可见，则切换为可见
            mov     findstat, 0           ; 否则，置为0

; 调用子程序：Pager (0)
; 刷新显示，以隐藏搜索栏

            sub     dx, dx
            push    dx
            call    Pager

            @SetCurPos  0, 43             ; 隐藏光标

            pop     dx
            pop     bp
            ret
```

```
tofind:
            mov     findlen, 0              ; 重置搜索缓冲长度
            mov     matchn, 0               ; 匹配数设为0

; 调用子程序：ShowMatch
; 在搜索栏显示匹配数目

            call    ShowMatch               ; 显示匹配数

            ; 初始化搜索栏填充文本

            push    es
            push    ds
            pop     es
            mov     di, OFFSET findline + 12
            mov     cx, 45                  ; 行号的空位数
            mov     al, " "
            rep     stosb                   ; 填充空格
            pop     es

; 调用子程序：CellWrt (ds, OFFSET findline, rows, cell)
; 指定配色，填充搜索栏内容

            push    ds                     ; 参数1
            mov     dx, OFFSET findline     ; 参数2
            push    dx
            mov     dx, rows                ; 参数3
            push    dx
            mov     dl, statatr             ; 参数4
            mov     BYTE PTR cell[1], dl
            push    cell
            call    CellWrt

            mov     dh, BYTE PTR rows
            @SetCurPos      12              ; 显示光标
            mov     findstat, 1             ; 设置搜索栏可见

            pop     dx
            pop     bp
            ret
ShowFind    ENDP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; 程序：    ShowKey
; 功能：    显示输入的字符
; 参数：    al - Ascii
; 输出：    无

ShowKey     PROC
            push    cx
            push    dx

            mov     dl, findstat           ; 搜索栏是否可见
            or      dl, dl
            je      endshow                ; 为0不可见，返回

            cmp     al, 8                  ; 退格键?
            jne     normalkey              ; 不是，跳到正常按键
            mov     si, findlen            ; 退格，判断长度
            or      si, si
            je      endshow                ; 长度=0，结束
            dec     findlen                ; 文本长度--
            dec     si
            jmp     showk
normalkey:
            cmp     al, 32
            jl      endshow
            mov     si, findlen            ; 搜索文本长度
            cmp     si, 44
            jg      endshow
            mov     findstr[si], al        ; 加入新字符
            inc     si                     ; 长度++
            mov     findlen, si            ; 更新长度
showk:
            mov     dh, BYTE PTR rows
            mov     dl, BYTE PTR findlen
            add     dl, 12
            @SetCurPos                     ; 更新显示光标

            push    es
            push    ds
            pop     es                     ; ds->es

            push    si                     ; 保存长度
            mov     cx, si                 ; cx = 长度
            mov     si, OFFSET findstr     ; si为搜索文本起始
            mov     di, OFFSET findline + 12    ; di为搜索栏的填空起始
```

```
            rep     movsb                   ; 循环复制

            ; 初始化搜索栏填充文本

            pop     di                      ; di 取出长度
            mov     ax, di                  ; 存入 ax
            add     di, OFFSET findline + 12    ; di 为空格起始
            mov     cx, 45                  ; 行号的空位数
            sub     cx, ax                  ; 填充空格数 cx = 空位数 - 行号文本长度
            mov     al, " "
            rep     stosb                   ; 填充空格

            pop     es                      ; 恢复 es


; 调用子程序：FindString

            call    FindString


; 调用子程序：CellWrt (ds, OFFSET findline, rows, cell)
; 指定配色，填充搜索栏内容

            push    ds                      ; 参数 1
            mov     dx, OFFSET findline     ; 参数 2
            push    dx
            mov     dx, rows                ; 参数 3
            push    dx
            mov     dl, statatr             ; 参数 4
            mov     BYTE PTR cell[1], dl
            push    cell
            call    CellWrt
endshow:
            pop     dx
            pop     cx
            ret
ShowKey     ENDP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    FindString
; 功能：    搜索文本
; 参数：    无
```

```
; 输出：    无

FindString  PROC

            mov     matchn, 0
            mov     hltline, 1              ; 当前高亮行为 1
            mov     hltpos, -1               ; 当前高亮偏移为 0

            mov     ax, fsize               ; 原文长度 len0
            mov     bx, findlen             ; 搜索长度 len1
            or      bx, bx
            je      zeroend                 ; 搜索长度为 0，结束
            sub     ax, bx                  ; len0 - len1
            jb      findend                 ; 搜索长度超过原文，结束

            mov     dx, 0FFFFh              ; 初始化计次

findnext:   inc     dx                      ; 计次++
            cmp     dx, ax
            jg      findend                 ; dx > len0，到头，结束

            cld                             ; 正向
            push    es                      ; 加载原文段
            mov     es, sbuffer
            mov     di, dx                  ; 加载计次（原文偏移）
            mov     si, OFFSET findstr      ; 搜索偏移

            mov     cx, bx                  ; 匹配 len1 次
            repe    cmpsb

            pop     es                      ; 恢复 es

            jz      matched                 ; 匹配到，跳到 matched

; 调用子程序：ShowMatch
; 未匹配到，更新背景为原配色

            call    RefreshBack

            jmp     findnext

matched:    inc     matchn                  ; 匹配到，数目++
            mov     dx, di
            sub     dx, bx
```

```
; 调用子程序：Highlight (bx)
; 匹配到，更新背景为高亮配色，高亮 bx 个字符

            push    bx
            call    Highlight

            mov     dx, di
            dec     dx

            jmp     findnext                ; 继续

; 调用子程序：ClearHlt
; 还原所有配色

zeroend:    call    ClearHlt

; 调用子程序：ShowMatch
; 在搜索栏显示匹配数目

findend:    call    ShowMatch               ; 显示匹配数目
            ret
FindString  ENDP



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    ShowMatch
; 功能：    显示匹配数目
; 参数：    无
; 输出：    无

ShowMatch   PROC

; 调用子程序：BinToStr (matchn, OFFSET findmatch) -> ax
; 将匹配数目转换为文本，长度存入 ax
            push    es
            push    ds
            pop     es
            push    matchn                  ; 参数 1
            mov     ax, OFFSET findmatch    ; 参数 2
            push    ax
            call    BinToStr                ; 二进制到文本

            mov     cx, 7                   ; 匹配数目的空位数
```

```
            sub     cx, ax                  ; 填充空格数 cx = 空位数 - 行号文本长度
            mov     al, " "
            rep     stosb                   ; 填充空格
            pop     es
            ret

ShowMatch   ENDP



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    Highlight (len)
; 功能：    高亮显示
; 参数：    dx - 位置，栈 - 1.长度
; 输出：    无

Highlight   PROC
            push    bp
            mov     bp, sp

            push    es
            push    di
            push    ax
            push    bx
            push    cx
            push    dx

            mov     es, sbuffer
            mov     di, dx                  ; 载入位置
            dec     di                      ; 回退一个（因为搜索时递增了）
            mov     al, 13
            cmp     es:[di], al             ; 当前字符是否为换行
            jne     nonewline2              ; 不是，nonewline
            inc     hltline                 ; 是，当前高亮行++
            mov     hltpos, di              ; 记录行首
            inc     hltpos
            jmp     skipfill2               ; 跳出
nonewline2:
            mov     ax, hltline             ; 当前高亮行
            cmp     ax, linenum             ; 显示的行号
            jl      skipfill2               ; 没到，跳过
            sub     ax, linenum             ; 否则，作差
            inc     ax                      ; 补 1
            cmp     ax, rows                ; 是否超出
            jge     skipfill2               ; 超出，跳过
```

```
            sub     di, hltpos              ; 行内偏移
            add     di, di                  ; 偏移*2，变成字节

            mov     bx, 80 * 2              ; 加上行偏移
            mul     bl
            add     di, ax

            mov     cx, [bp + 4]            ; 参数1，重复高亮次数
nexthlt:
            mov     es, vidadr              ; 屏幕缓冲区
            mov     ax, es:[di]
            mov     ah, hltatr              ; 修改配色
            mov     es:[di], ax

            add     di, 2                   ; 加俩字节
            loop    nexthlt

skipfill2:  pop     dx
            pop     cx
            pop     bx
            pop     ax
            pop     di
            pop     es
            pop     bp
            ret     2
Highlight   ENDP


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    RefreshBack
; 功能：    高亮显示
; 参数：    dx - 位置
; 输出：    无

RefreshBack PROC

            push    es
            push    di
            push    ax
            push    bx
            push    cx
            push    dx
```

```
            mov     es, sbuffer
            mov     di, dx                  ; 载入位置
            dec     di                      ; 回退一个（因为搜索时递增了）
            mov     al, 13
            cmp     es:[di], al             ; 当前字符是否为换行
            jne     nonewline               ; 不是，nonewline
            inc     hltline                 ; 是，当前高亮行++
            mov     hltpos, di              ; 记录行首
            inc     hltpos
            jmp     skipfill                ; 跳出
nonewline:
            mov     ax, hltline             ; 当前高亮行
            cmp     ax, linenum             ; 显示的行号
            jl      skipfill                ; 没到，跳过
            sub     ax, linenum             ; 否则，作差
            inc     ax                      ; 补 1
            cmp     ax, rows                ; 是否超出
            jge     skipfill                ; 超出，跳过

            sub     di, hltpos              ; 行内偏移
            add     di, di                  ; 偏移*2，变成字节

            mov     bx, 80 * 2              ; 加上行偏移
            mul     bl
            add     di, ax

            mov     es, vidadr              ; 屏幕缓冲区
            mov     ax, es:[di]
            mov     ah, scrnatr             ; 修改配色
            mov     es:[di], ax



skipfill:   pop     dx
            pop     cx
            pop     bx
            pop     ax
            pop     di
            pop     es
            ret
RefreshBack ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    ClearHlt
; 功能：    取消高亮显示
```

```
; 参数：    无
; 输出：    无

ClearHlt PROC

          push    es
          push    di
          push    ax
          push    bx
          push    cx
          push    dx



          mov     di, 80 * 2            ; 从第二行开始
          mov     ax, rows              ; 计算总行数
          sub     ax, 2
          mov     bx, 80 * 2
          mul     bl
          mov     cx, ax                ; 放在 cx，作为循环次数

nextclear:  mov     es, vidadr            ; 屏幕缓冲区
          mov     ax, es:[di]
          mov     ah, scrnatr           ; 修改配色
          mov     es:[di], ax
          add     di, 2                 ; 加俩字节
          loop    nextclear

          pop     dx
          pop     cx
          pop     bx
          pop     ax
          pop     di
          pop     es
          ret
ClearHlt ENDP

          END
```

1.2 如何高亮显示

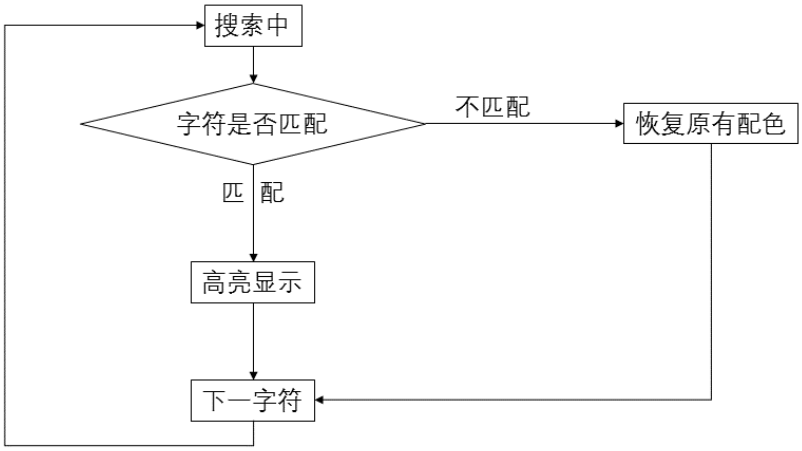
可以想到，搜索过程中，需要对全文遍历，并可以获取到搜索结果的位置。

我们知道，屏幕显示内容的每个单元由两个字节构成：高位为配色，低位为字符。

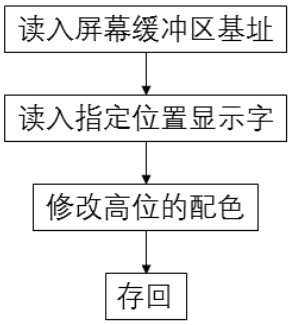
因此，在遍历过程中，修改所有未匹配的字符的配色为屏幕默认配色，修改匹配到的字符为高亮配色即可。

1.2.1 流程图

1.2.1.1 高亮（还原配色）的调用



1.2.1.2 高亮（还原配色）的实现



1.2.2 程序中的体现

```
FindString  PROC
            …
findnext:   …
            jz      matched                 ; 匹配到，跳到 matched
; 调用子程序：ShowMatch
; 未匹配到，更新背景为原配色
            call    RefreshBack
            jmp     findnext
matched:    …
; 调用子程序：Highlight (bx)
; 匹配到，更新背景为高亮配色，高亮 bx 个字符
            push    bx
            call    Highlight
            …
            jmp     findnext                ; 继续
```

```
; 调用子程序：ClearHlt
; 还原所有配色
zeroend:    call    ClearHlt
; 调用子程序：ShowMatch
; 在搜索栏显示匹配数目
findend:    call    ShowMatch              ; 显示匹配数目
            ret
FindString  ENDP
```

可以看到，在搜索过程中，就是按照上述方法进行处理的。同时，还对空搜索文本进行特殊处理，即清空所有高亮。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 程序：    Highlight (len)
; 功能：    高亮显示
; 参数：    dx - 位置，栈 - 1.长度
; 输出：    无

Highlight   PROC
            …

nonewline2:
            …
            mov     cx, [bp + 4]           ; 参数 1，重复高亮次数
nexthlt:
            mov     es, vidadr             ; 屏幕缓冲区
            mov     ax, es:[di]
            mov     ah, hltatr             ; 修改配色
            mov     es:[di], ax

            add     di, 2                  ; 加俩字节
            loop    nexthlt
skipfill2:  …
            ret     2
Highlight   ENDP
```

可见，在 Highlight 子程序中，就将 FindString 中指定的字符的配色修改了。RefreshBack 和 ClearHlt 同理。

其中，省略的大部分代码是用于计算偏移位置的，具体参见前文详细注释。

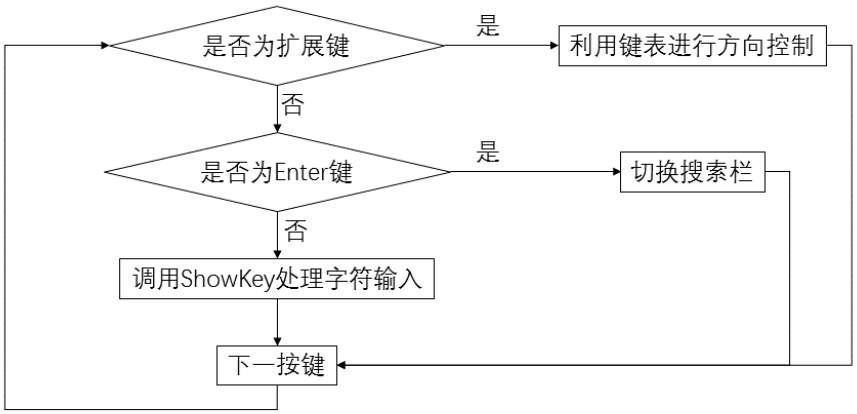
1.3 如何增加输入功能

show.asm 最后的部分，是有关键盘输入的。针对一般 Ascll 输入和扩展码输入分别做了判断。我的程序中，设定 Enter 键激活搜索功能，然后可以输入字符进行搜索。

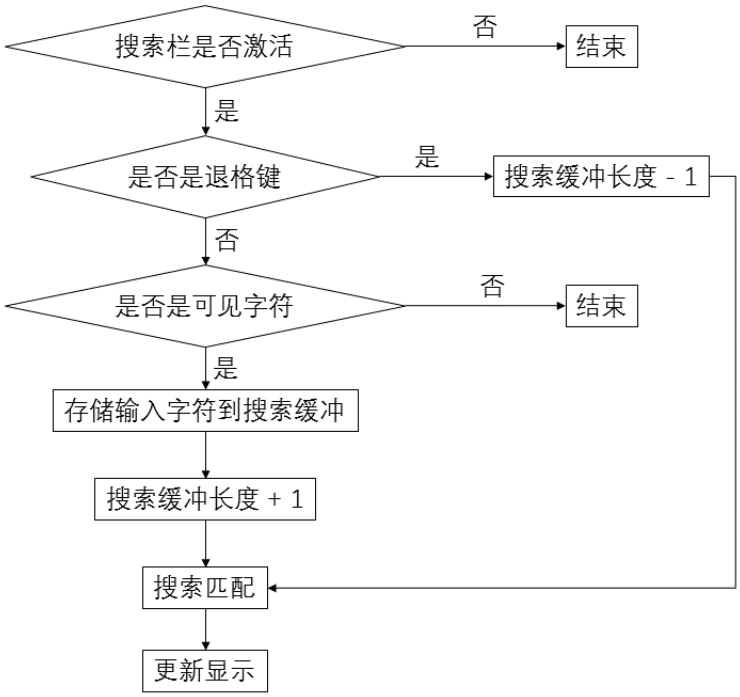
因此，我需要增加 Ascll 处理的部分。

1.3.1 流程图

1.3.1.1 按键判断



1.3.1.2 搜索栏显示字符



1.3.2 程序中的体现

```
nextkey2:   cmp     al, 0                    ; 是否为空
            je      extended                 ; 如果是，则为扩展键，跳到 extended
            cmp     al, 13                   ; 是否为 Enter
            jne     nextkey3                 ; 不是，跳到 Esc 判断
            call    findk                    ; 是 Enter，调用 findk
            jmp     nextkey
nextkey3:   cmp     al, 27                   ; 是否为 Esc
            je      quit                     ; 是 Esc，跳到 quit
            call    ShowKey                  ; 调用显示 key 子程序，为搜索栏显示
            jmp     nextkey
```

```
quit:      …
```

代码中，我增加了对 Enter 键的判断以及在输入其他字符时，调用 ShowKey 来在搜索栏显示字符的部分。

```
ShowKey     PROC
          …

            mov     dl, findstat          ; 搜索栏是否可见
            or      dl, dl
            je      endshow               ; 为 0 不可见，返回

            cmp     al, 8                 ; 退格键?
            jne     normalkey             ; 不是，跳到正常按键
            …
            jmp     showk
normalkey:  cmp     al, 32
            jl      endshow
            …
showk:      …
endshow:    …
…
            ret
ShowKey     ENDP
```

首先判断搜索栏是否显示，如果没有显示，则不处理字符输入。

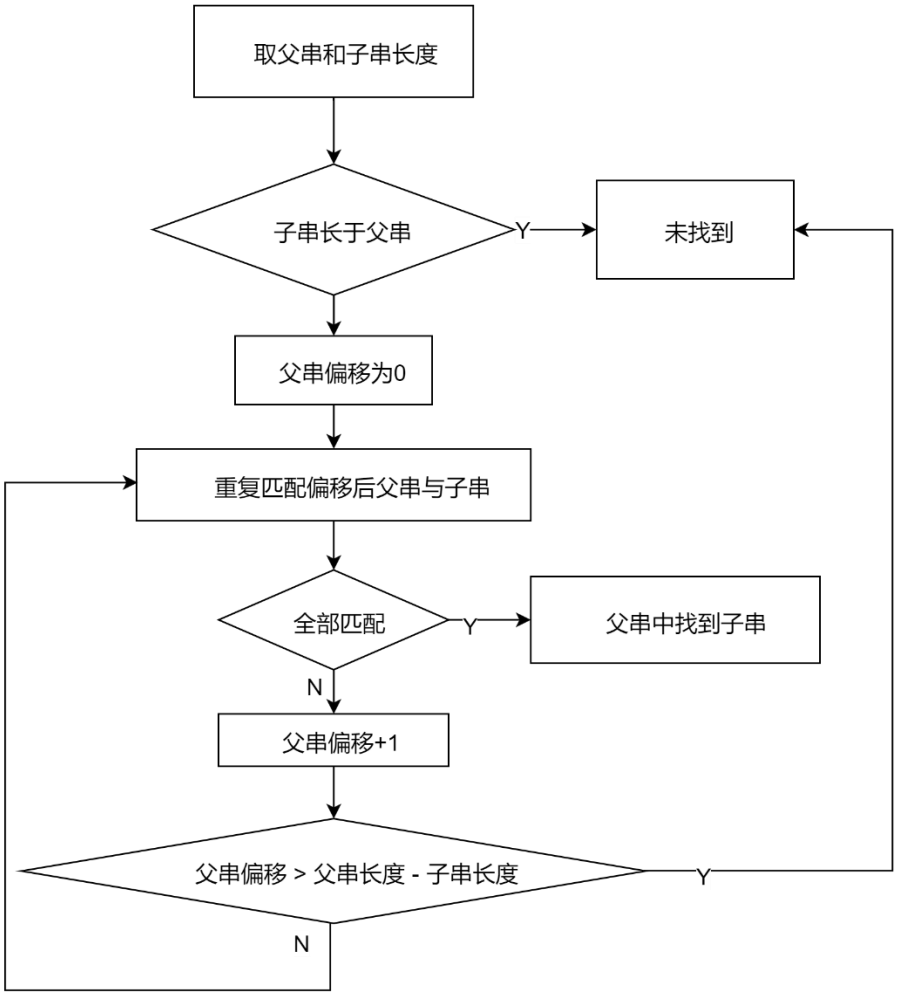
然后，对退格键进行特殊处理。如果不是退格键，排除 Ascll 小于 32 的非可见字符，记录到搜索缓冲区中。

1.4 如何实现搜索

从原文第一个字符开始，依次与搜索文本匹配。

1.4.1 流程图

图示展示了匹配一次的算法，实际还要不断重复这一过程，得到所有匹配。



1.4.2 程序中的体现

```
FindString  PROC
            …
            mov     ax, fsize               ; 原文长度 len0
            mov     bx, findlen             ; 搜索长度 len1
            or      bx, bx
            je      zeroend                 ; 搜索长度为 0，结束

            sub     ax, bx                  ; len0 - len1
            jb      findend                 ; 搜索长度超过原文，结束

            mov     dx, 0FFFFh              ; 初始化计次
findnext:   inc     dx                      ; 计次++
            cmp     dx, ax
            jg      findend                 ; dx > len0，到头，结束
            cld                            ; 正向
            push    es                      ; 加载原文段
            mov     es, sbuffer
            mov     di, dx                  ; 加载计次（原文偏移）
            mov     si, OFFSET findstr      ; 搜索偏移
```

```
            mov     cx, bx                  ; 匹配 len1 次
            repe    cmpsb

            pop     es                      ; 恢复 es

            jz      matched                 ; 匹配到，跳到 matched
            …
            jmp     findnext
matched:    inc     matchn                  ; 匹配到，数目++
            …
            jmp     findnext                ; 继续
            …
findend:    …
            ret
FindString  ENDP
```

程序利用了串操作指令 repe cmpsb，循环对原文每个字符开头时的文本和搜索文本匹配。

1.5 如何定位匹配字符

这是整个程序的难点。搜索过程中，我们可以得到匹配字符串相对原文的位置。

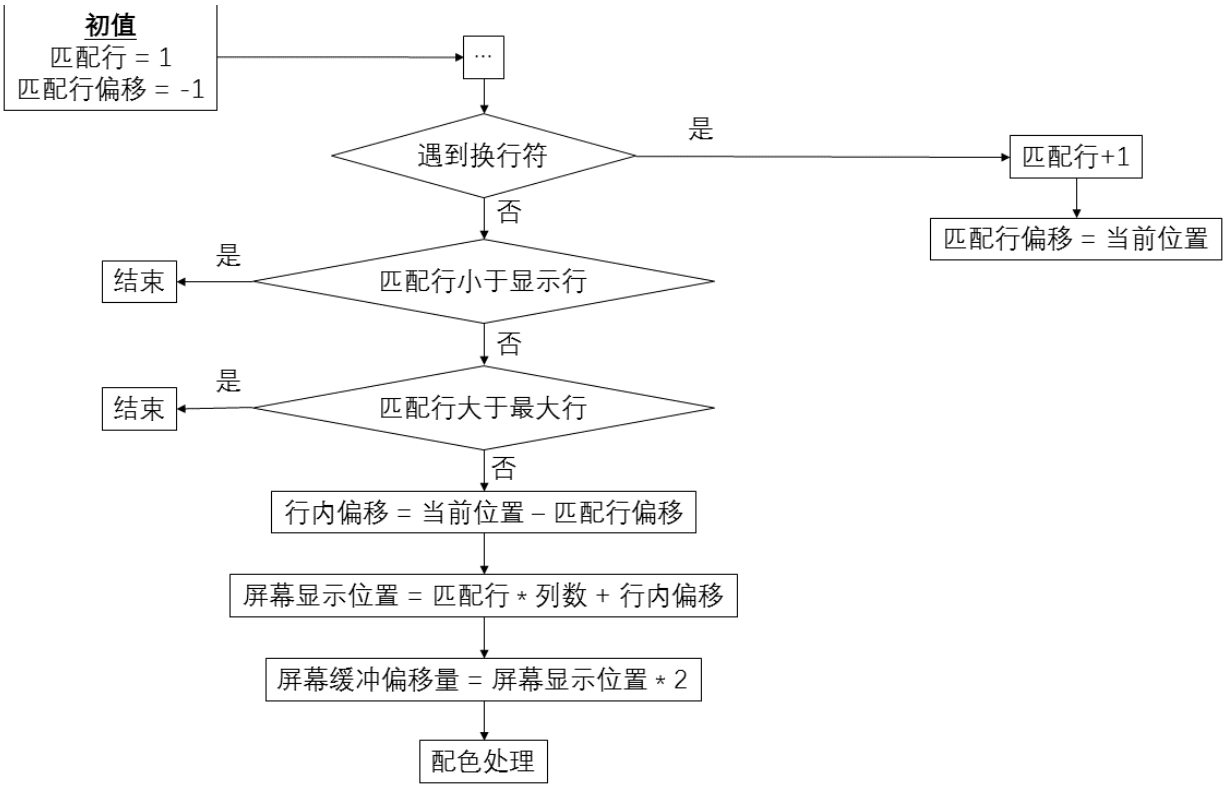
通过 pbuffer 缓冲偏移量，可以知道屏幕第一个字符相对原文的位置。

搜索中，还可以判断字符是否为换行符，以此得知当前匹配到第几行。

最后，我们还知道显示的第一行是原文的第几行。

有了以上数据，我们便可以定位匹配的字符在屏幕中的位置，以高亮显示。

1.5.1 流程图



1.5.2 程序中的体现

```
            mov     es, sbuffer
            mov     di, dx                  ; 载入位置
            dec     di                      ; 回退一个（因为搜索时递增了）
            mov     al, 13
            cmp     es:[di], al             ; 当前字符是否为换行
            jne     nonewline2              ; 不是，nonewline
            inc     hltline                 ; 是，当前高亮行++
            mov     hltpos, di              ; 记录行首
            inc     hltpos
            jmp     skipfill2               ; 跳出
nonewline2:
            mov     ax, hltline             ; 当前高亮行
            cmp     ax, linenum             ; 显示的行号
            jl      skipfill2               ; 没到，跳过
            sub     ax, linenum             ; 否则，作差
            inc     ax                      ; 补 1
            cmp     ax, rows                ; 是否超出
            jge     skipfill2               ; 超出，跳过

            sub     di, hltpos              ; 行内偏移
            add     di, di                  ; 偏移*2，变成字节

            mov     bx, 80 * 2              ; 加上行偏移
```

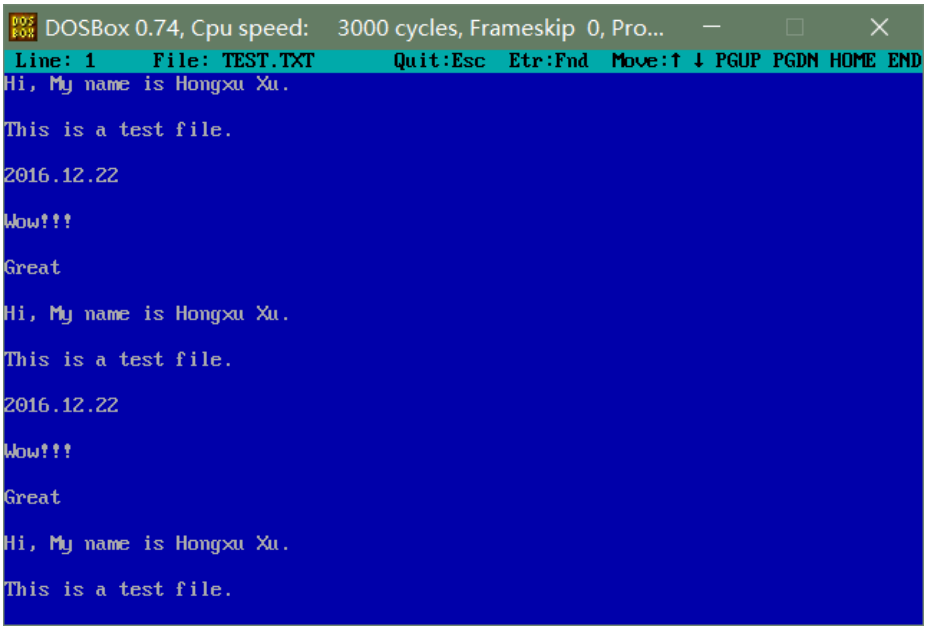
```
            mul     bl
            add     di, ax
nexthlt:
            mov     es, vidadr              ; 屏幕缓冲区
            mov     ax, es:[di]
            mov     ah, hltatr              ; 修改配色
            mov     es:[di], ax
```

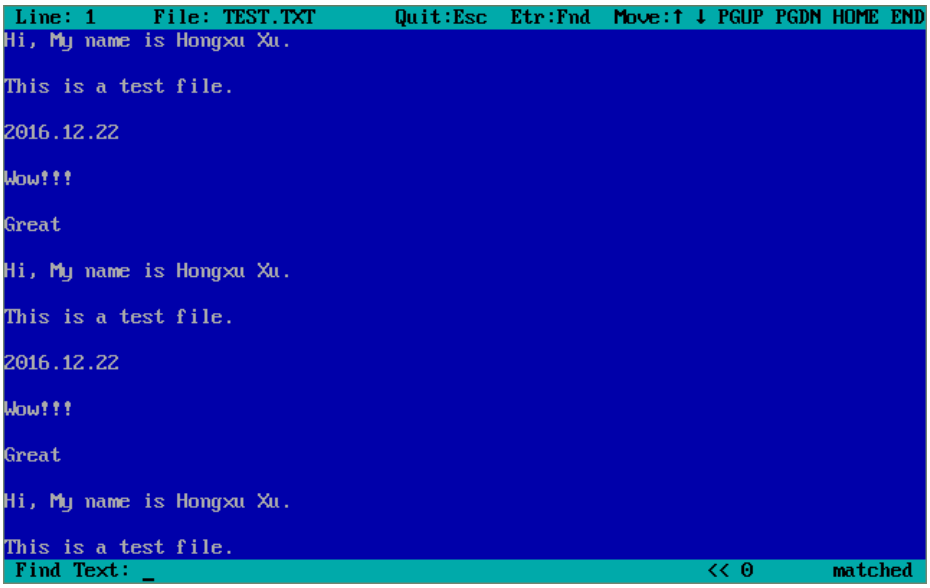
2 设计

本程序人机交互十分友好，原有功能均未受影响，新增功能与原有功能配合完美。

2.1 切换搜索栏

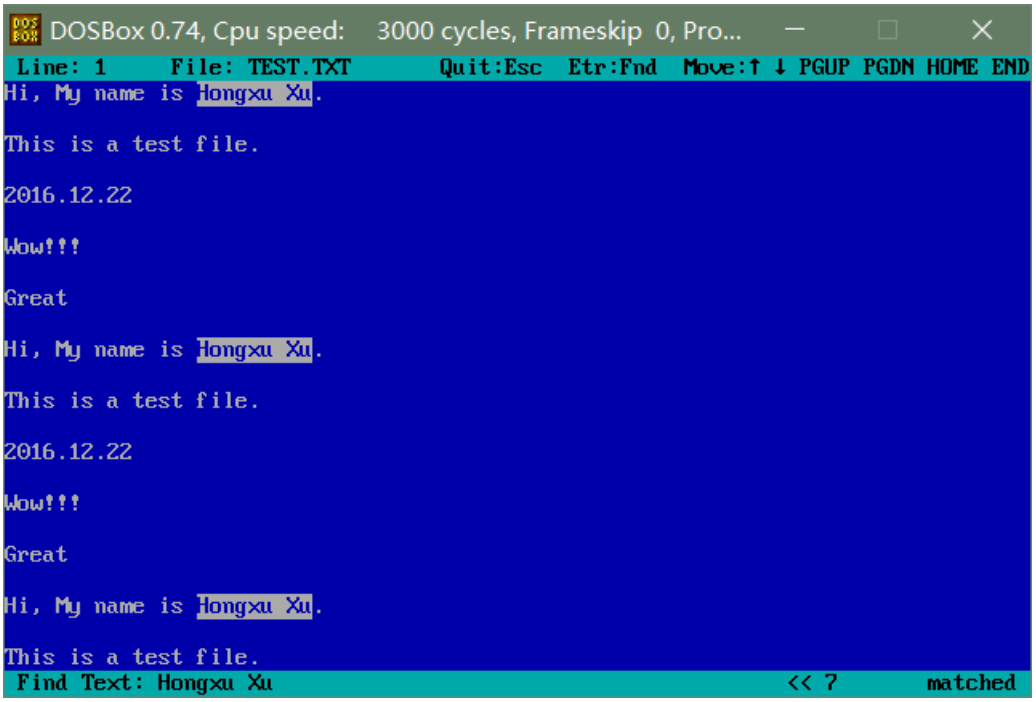
Enter 键切换搜索栏，在主界面下方显示。





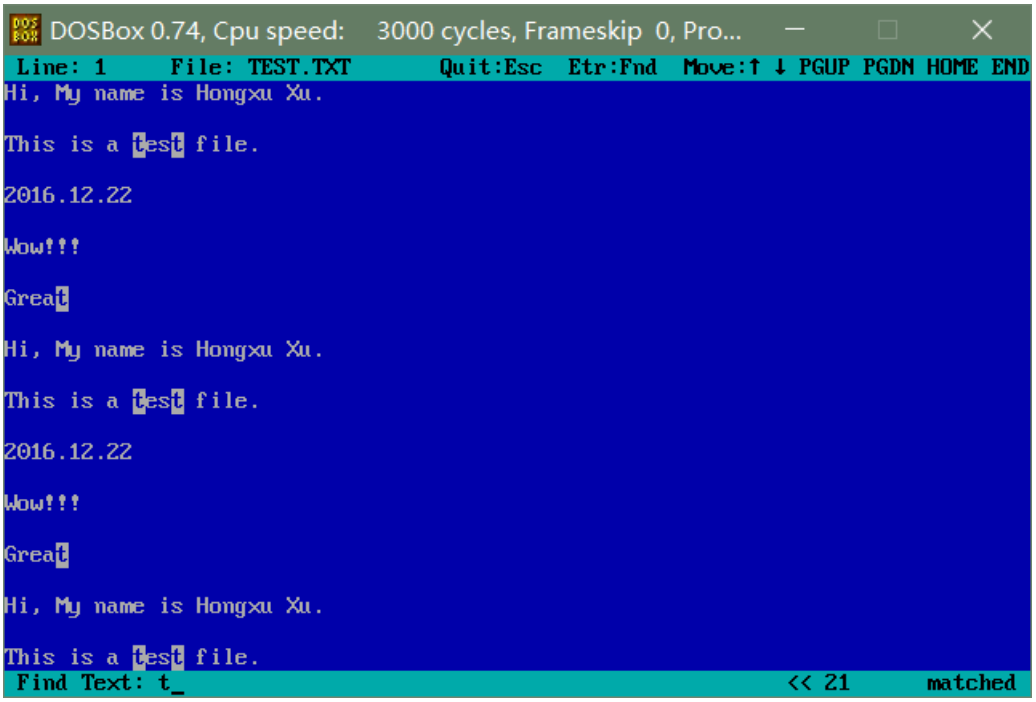
2.2 输入字符实时搜索

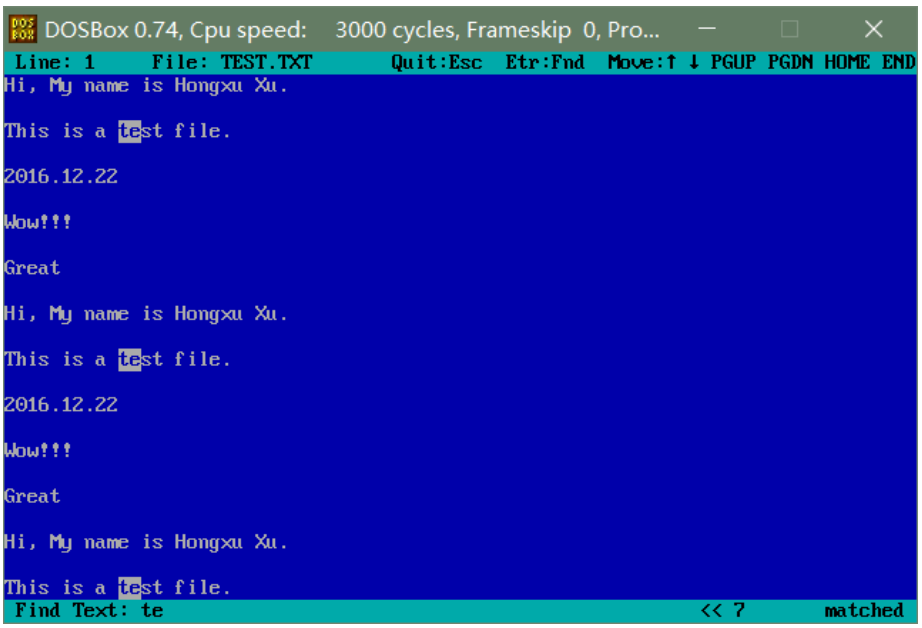
随着字符输入，自动更新右下角匹配数目并高亮显示匹配字符。

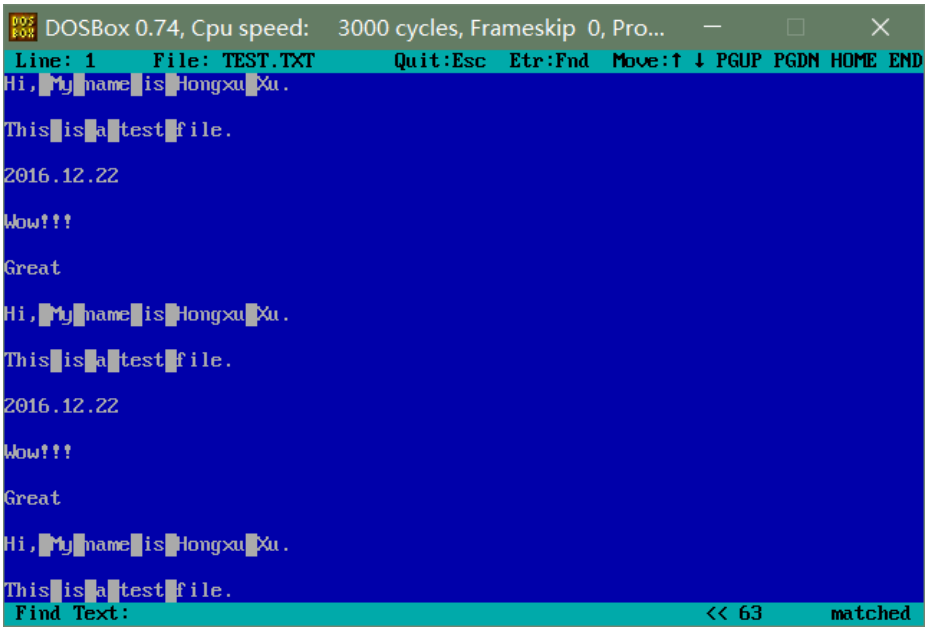


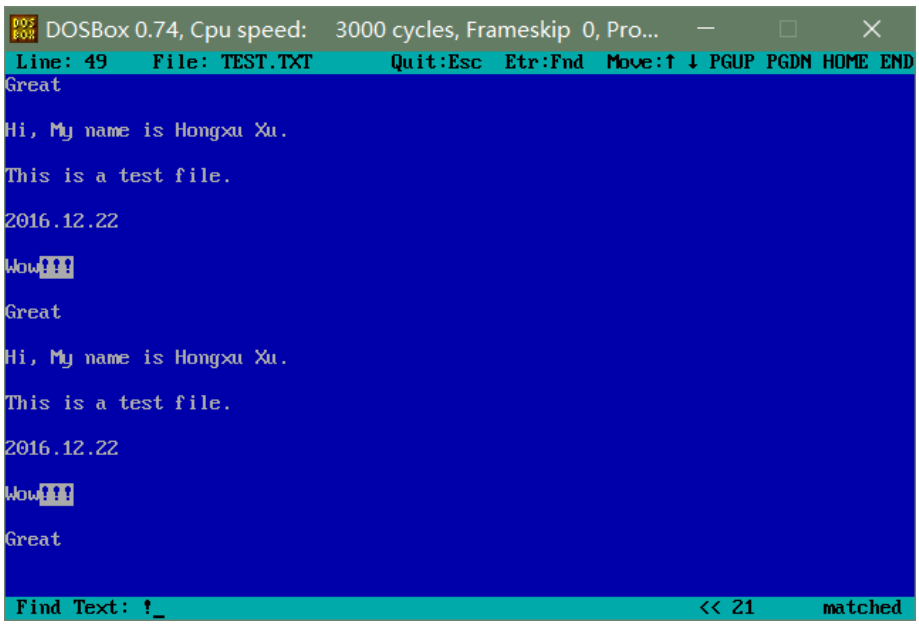
3 结果展示

更多图片展示结果。




DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Line: 1 File: TEST.TXT Quit:Esc Etr:Fnd Move:↑ ↓ PGUP PGDN HOME END
Hi, My name is Hongxu Xu.
This is a test file.
2016.12.22
Wow!!!
Great
Hi, My name is Hongxu Xu.
This is a test file.
2016.12.22
Wow!!!
Great
Hi, My name is Hongxu Xu.
This is a test file.
Find Text: te
<< 7 matched


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Line: 1 File: TEST.TXT Quit:Esc Etr:Fnd Move:↑ ↓ PGUP PGDN HOME END
Hi, My name is Hongxu Xu.
This is a test file.
2016.12.22
Wow!!!
Great
Hi, My name is Hongxu Xu.
This is a test file.
2016.12.22
Wow!!!
Great
Hi, My name is Hongxu Xu.
This is a test file.
Find Text:
<< 63 matched


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Line: 49 File: TEST.TXT Quit:Esc Etr:Fnd Move:↑ ↓ PGUP PGDN HOME END
Great
Hi, My name is Hongxu Xu.
This is a test file.
2016.12.22
Wow!!!
Great
Hi, My name is Hongxu Xu.
This is a test file.
2016.12.22
Wow!!!
Great
Find Text: !
<< 21 matched

4 总结

最开始，阅读所给程序就是一个挑战。原程序书写风格较乱，阅读不便；于是我用自己的代码风格，重新抄写了一遍，美化了格式，增加了详细的注释，方便了后期阅读和修改。

仔细阅读之后，了解了文件加载和屏幕显示等功能的实现方法，便开始着手设计和实现搜索功能。

受 Vim 的启发，我把搜索栏涉及到了最下方，正好也不妨碍顶栏。

书写程序过程中，屏幕显示部分是最困难的。从模仿原程序代码，到直接自己写显示代码，花费了很大精力。也经常出错，在 DOSBox 下也不能有效调试，只能重新审视代码，最终才得以实现。

作业要求的每一项都涵盖在了本文档和程序代码及其注释中，经过期末作业的实践，对汇编程序已经非常熟悉，在 DOS 下的大部分应用也都能够做到。同时，80x86 汇编的学习，也帮助我更好的理解了计算机，还帮助我在 x86/x64 汇编使用上事半功倍，更是让我在微机原理这门课上能够更加迅速的理解和完成作业。总之，受益颇多。