汇编实验安排

时间

第 8 周至 17 周，周一， 1~2 节

第 12 周至 17 周，周三， 3~4 节

地点&分组

1 组：N3-101(103) 马一帆TA 2 组：N3-105(107) 张立芳TA 3 组：N3-109(111) 王均平TA

实验要求

- 按照学院实验室疫情防控要求，戴好口罩，隔开就座
- 可携带自己的笔记本进行上机实验
- 每次实验结束后需要提交实验报告
  - 分组1：https://icloud.qd.sdu.edu.cn:7777/#/link/BF5178D31B4FB6301B31D17AA4CF0C01
  - 分组2：https://icloud.qd.sdu.edu.cn:7777/link/884C632F9289ED1A2DEE7C5C0C1DA596
  - 分组3：https://icloud.qd.sdu.edu.cn:7777/#/link/1BD91A327450154133CEEED5D6DC097C
  - 实验报告模板文件： 实验报告模板.doc
  - 实验报告提交文件名示例： 实验1_201812345678_张三.doc （不是 docx ）
  - 实验报告提交截止时间：实验结束后 7天 内，即下周当天的凌晨 23:59:59 以前
- **课程材料：https://icloud.qd.sdu.edu.cn:7777/#/link/B38258E8B0A7C73C8C2006178189E866**
- 实验材料：
  - IBM-PC汇编语言程序设计 实验教程.pdf
  - ``汇编调试器的使用.pdf`
  - 中断 21 表： IBM-PC汇编语言程序设计 第2版.pdf 附录 4

实验 1

实验时间：

- 第 7 周，周二， 5~6 节 (特殊情况，调课)
- 2021.10.21

实验标题：

- 实验1：示例1.1

实验内容：

- 配置环境
  - 学习使用和熟悉 MASM 、 LINK 、 DEBUG 、 TD 等汇编工具
  - 一个更为方便的使用方式： Assembly 实验环境安装教程 for windows.mp4 、 Assembly 实验环境安装中出现的一些问

题 for windows.mp4

- 阅读理解调试程序
  - 例 1.1 ，理解、运行、调试
  - 学习、掌握一般汇编程序的编写框架
  - 学习 DEBUG 、 TD 单步调试、查看/更改寄存器、查看/更改内存单元
  - 学习查找中断 21 的表格，熟悉输出文本等系统调用
  - 学习优秀的汇编程序编写习惯，包括但不限于寄存器使用规范、变量/标号命名、注释、对齐、分段、缩进。

~~实验 2~~

实验时间：

- 第 7 周，周 二 ， 7~8 节
- 2021.10.21

实验标题：

- 实验2：示例2.1和2.2

实验内容：

- 阅读理解调试程序
  - 例 2.1 ，理解、运行、调试
  - 例 2.2 ，理解、运行、调试
  - 掌握示例中出现的寻址方式
  - 思考代码中的常量（整型数）的含义
  - 熟悉了解汇编 label, dup 伪指令的功能
  - 学习查找中断 21 表，熟悉键盘输入（掌握返回参数）等系统调用
  - 了解字符串在内存中的存储方式，存储形式 dw、db 的不同
  - 养成良好的汇编程序编写习惯，合理使用寄存器存放对应信息，对于嵌套结构，要合理保存并恢复相关寄存器的信息
  - DEBUG 或 TD 调试查看代码执行后内存中的 rank 数组的内容

~~实验 3~~

实验时间：

- 第 9 周，周 一 ， 1~2 节
- 2020.11.2

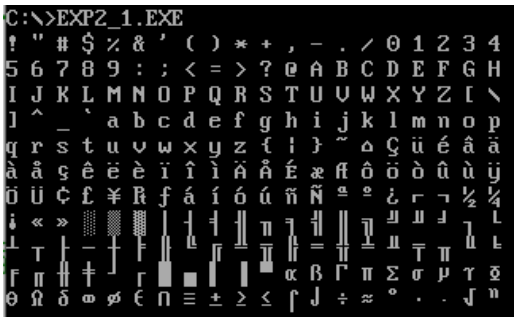
实验标题：

- 实验3：实验2.1和2.2

实验内容：

- 完成 实验 2.1
  - 了解并使用 02h 功能号的系统调用

- 掌握汇编中多层循环的使用
- 学习寄存器使用冲突的处理
- 改动点：原实验书中要求输出 10H 到 100H 范围的字符，现在改为 21H 到 FCH 范围，要求按 11 行 × 20 列的表格形式显示（如果顺利，你将得到以下输出）



- 拓展：如果不进行实验改动，按照实验书中要求输出的字符范围为 10H 到 100H ，要求按 15 行 × 16 列的表格形式显示。在**你的实验环境**中能正确输出他们吗？若不能，有哪些字符有问题，为什么？在思考 "为什么" 的过程中，你是怎么逐步得到结论的？（或者说你分析问题、思考问题、解决问题的思路是什么）

- 完成 实验 2.2
  - 掌握 0ah 功能号的系统调用
  - 熟练使用串比较指令，理解附加段的作用
  - 练习 2 进制到 16 进制的转换，并以 ASCII 码的形式输出
  - 理解 si、di 寄存器在实验中的作用
- 你在编写代码时遇到过输出和输入交替反复使用，而出现的输出输入相互覆盖的情况吗？如果有，你是怎么解决的。

~~实验 4 (4 个学时)~~

实验时间：

- 2020.11.9：第 10 周，周一， 1~2 节
- 2020.11.11：第 10 周，周三， 3~4 节 (特殊情况，调课)

实验标题：

- 实验4：示例2.3、示例2.4、实验2.3

实验内容：

- 阅读理解调试程序 例 2.3
  - 看蓝猫学蓝猫，尝试学习理解和掌握汇编分支程序设计思路、循环程序设计思路
  - 理解条件转移指令和无条件转移指令的机理（哪个是根据状态寄存器，哪个是位移，哪个是段首址加有效地址）。了解段内短转移、段内近转移、段间转移、直/间接等的含义及其跳转范围。
- 阅读理解调试程序 例 2.4
  - 理解、掌握 ASCII 转 Binary 逻辑
- 完成 实验 2.3
  - 原实验是实验书中的 实验 2.3
  - 现在进行改动： 根据给定的框架完成程序，输入一行字符串（长度最多100），你需要统计其中各个小大写字母和数字字符的数量。要求用数组实现，存放到数据段中的 counter 数组中。
  - 实践和掌握分支/条件汇编程序设计
  - 了解 4ch 号 DOS 系统调用
  - **请在 https://oj.qd.sdu.edu.cn/problem/SDUOJ-1010 （请在校园网内访问），即 SDUOJ 的网站中提交你的代码。 你的账号密码都是你的学号，注意及时改密 （由于网站还在小范围内测中，所以可能会存在一些 bug，若发现请告知助教）**
  - 汇编代码框架（以 OJ 上为准）

~~实验 5~~

实验时间：

- 2020.11.16：第 11 周，周一， 1~2 节

实验标题：

- 实验5：示例2.6、示例2.7

实验内容：

- 检查至今自己有没有完全掌握汇编程序设计的相关方法论
  - 读程序
    - 每一行代码都确保搞懂，不理解 -> 课本、PPT、搜索引擎，去查找该指令的含义和用法
    - 结合上下文的理解，搞懂几个标号之间的汇编代码块逻辑
    - 搞懂各个子程序的代码逻辑、入参、出参、入参和出参使用的方法（寄存器法、约定内存法、压栈法）
    - 搞懂整个代码的逻辑，能用 1、2 句话简明易懂、总结性地说出来

        - 会查课本 附录 4 DOS 系统功能调用 的表格，即中断 21h 的表
        - 会查课本 附录 3 中断向量地址一栏 的表格，如果你看到 int 后面跟的不是 21h ，那么你可能需要查询一下这个中断是什么含义
    - 写程序
        - 通过读程序初步掌握**程序设计思维**，现在是做**实践**部分
    - debug
        - 方法论和工具使用都很重要!
        - 掌握 debug 的使用方式 https://wenku.baidu.com/view/74439e7a9a89680203d8ce2f0066f5335b816729.html
        - 学习 td 的使用方法（一个更好用的 debug 工具） https://wenku.baidu.com/view/bb61302c9b6648d7c1c746fc.html
        - debug 的核心目的一定是为了**确定哪行代码**或者哪几行**造成了预期不符**

- 阅读理解调试程序 例 2.6
    - 学习，掌握子程序的模块化设计，以及子程序的嵌套
    - 理解、掌握二进制数转十进制的逻辑以及代码编写
- 阅读理解调试程序 例 2.7
    - 学习查找中断 10h（即 BIOS 系统调用）的相关功能（包括但不限于本次实验出现的功能，如 02H）
    - 学习，理解屏幕输出界面的清屏操作以及设置光标位置的功能
    - 熟悉将字操作数扩展乘双字的操作以及扩展后的双字操作数的计算处理
    - 了解例 2.7 中关于小数的乘除法处理
- 熟悉汇编模块化

~~实验 6~~

实验时间：

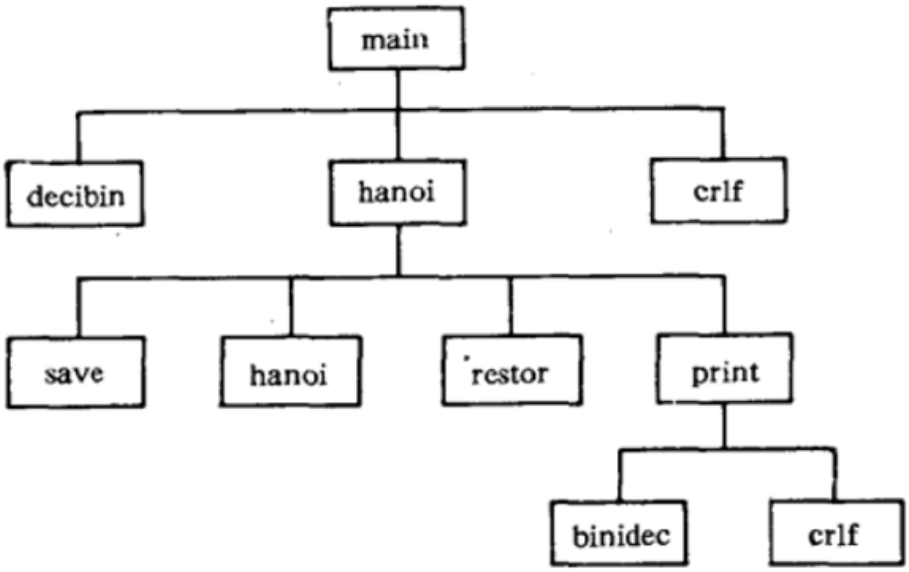
- 2020.11.18 ： 第 11 周，周 三 ， 3~4 节 (特殊情况，调课)

实验标题：

- 实验6：示例2.8、实验2.4

实验内容：

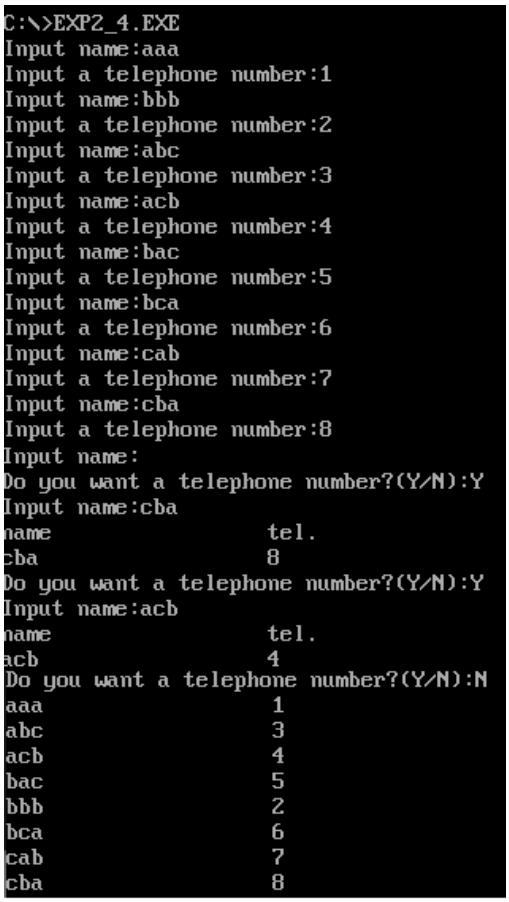
- 阅读理解调试程序 例 2.8
    - 汉诺塔问题，请按照实验书上的步骤和指导进行学习
    - 对于每一行指令都应该了解其作用，结合上下文理解，搞懂每一段汇编代码块的逻辑
    - 搞懂各个子程序的代码逻辑、入参、出参、入参和出参使用的方法（寄存器法、约定内存法、压栈法）
    - 学习汇编的模块化拆分。看着该**模块图**，你能将框内的子程序逻辑理清吗?



- 完成 实验 2.4
  - OJ 链接：https://oj.qd.sdu.edu.cn/contest/1
  - 这是要求自己写的实验，希望你能独立完成。给出一个路线
    - 读实验书中的描述几遍，确保了解程序的需求和逻辑，脑中要知道这个程序是做什么的（What），输入了什么数据，要做什么，要输出什么，至于怎么做（How）最好先不关注。
    - 用纸笔根据你了解的逻辑，初步画出一个程序的流程草图
    - 模块化。将各部分逻辑考虑抽到一个模块，画一个模块图，这会花一些时间，需要你的耐心
    - 细节、编码。加油
  - 实验验收需要提供程序的模块图（形如上面那个图）
  - 为了降低难度，我们约定，不存在对于一个人名对应多个电话号码的情况
  - 还是为了降低难度，我们对实验进行 **改动**：实验中 "对电话号码按人名排序" 变成*选做*， $O(n^2)$ 的排序即可
    - 改动点："重复查号提示符直至用户不再要求查号为止" 后，输出所有排序后的名字和电话号码
  - 为了保证输入输出统一，请使用我们提供的程序数据段

```
datasg    segment
    mess1    db    'Input name:', '$'
    mess2    db    'Input a telephone number:', '$'
    mess3    db     'Do you want a telephone number?(Y/N):', '$'
    mess4    db    'name', 17 dup(' '), 'tel.', 13, 10, '$'
    mess5    db    'Not Found', 13, 10, '$'
    tel_tab    db    50 dup(29 dup(' '), '$')  ; 20+1+8+1
    tel_cnt db    0
    strinp    label    byte
    strmax    db    21
    strlen    db    ?
    strdata    db    21 dup(?)
datasg    ends
```

- 实验效果图：



- 掌握汇编模块化编程

实验 ~~7~~

实验时间：

- 2020.11.23：第 12 周，周一， 1~2 节

实验标题：

- 实验7：示例3.1、实验3.1、示例3.3

实验内容：

- 阅读理解调试程序 例3.1
  - 理解枪声程序驱动发生器发生的方式
  - 学习使用 61H 输出控制寄存器来驱动扬声器
  - 理解子程序 shoot 的结构、使用的寄存器的作用、指令中数据的意义
  - 调试程序，尝试输出不同的枪声
- 阅读理解补全调试程序 实验3.1
  - 提示
    - 驱动扬声器方式采用位触发

- 产生一个音符需要上脉冲宽计数值与持续时间
- 上脉冲宽度计算：

  位触发本质是将 61H 端口的第一位不断进行 0、1 变换，从而产生对应的音频，所以我们需要计算输出端口进行 0/1 变换的延迟时间，即上脉冲宽度。

  从表中可以获取音频对应的频率 `freq` ，那么该音频的脉冲周期为 `1/freq` ，上脉冲宽度为 `1/(2*freq)` ，`loop` 指令执行2801次的时间约为10ms，我们可以使用 `loop` 指令来延迟0/1变换，那么问题就转换为了 `loop` 指令需要执行的次数。 `loop` 指令1s执行 `100*2801` 次，上脉冲宽度为 `1/(2*freq)` ，一个上脉冲宽度需要执行的 `loop` 指令数为 `(2801*100)/(2*freq)` 次，代码如下

```
mov      ax, 2801
mov     bx, 50
mul     bx
div     di            ; (di)=freq
mov        dx, ax        ; (dx)=1/(2*freq)
```

  两只老虎频率如下：

```
freq     dw 262,294,330,262,262,294,330,262            ;do re mi do do re mi do
        dw 330,349,392,330,349,392                    ;mi fa sol mi fa sol
        dw 392,440,392,349,330,262                    ;sol la sol fa mi do
        dw 392,440,392,349,330,262                    ;sol la sol fa mi do
        dw 294,196,262,294,196,262                    ;re so do re so do
```

- 持续时间：持续时间是一个音频的拍数，以0.125s为一拍，0.25s为两拍，0.5S为四拍(这一部分对应实验教程中的节拍计算，实验教程有误，请参照实验指导书)。一个上脉冲宽度的 `loop` 指令数为 `(2801*100)/(2*freq)` 次，持续时间为 `(2801*100*10/2801)/(2*freq)=500/freq` ms。为了得到统一的节拍，用外循环的方式，使得上脉冲宽度循环 `freq/4` 次，得到一拍的时间0.125ms，根据两只老虎的节拍，设置节拍计数值。

  两只老虎节拍如下

```
time    dw 2,2,2,2,2,2,2,2        ;do re mi do do re mi do
        dw 2,2,4,2,2,4            ;mi fa sol mi fa sol
        dw 1,1,1,1,2,2            ;sol la sol fa mi do
        dw 1,1,1,1,2,2            ;sol la sol fa mi do
        dw 2,2,4,2,2,4            ;re so do re so do
```

  外循环计算公式

```
freq*time/4
```

  一拍的时间、节拍数也可以自定义实现，节拍计数值与 CPU 的速度有关，有能力的同学可以不使用示例程序，自行设计，网上大部分代码有误，请勿照搬。

- Dosbox 的 CPU 速度可调，由于上述计算在 10ms 执行 2801 次 loop 指令，所以通过 `ctrl-F11` 与 `ctrl-F12` 调节至 280 cycles左右执行程序（或者设置 dosbox.conf 文件中 `cycles=fixed 280` ）

◦ 需要补全的代码

```
dataseg        segment
    freq    dw 262,294,330,262,262,294,330,262           ;do re mi do do re mi do
            dw 330,349,392,330,349,392                   ;mi fa sol mi fa sol
            dw 392,440,392,349,330,262                   ;sol la sol fa mi do
            dw 392,440,392,349,330,262                   ;sol la sol fa mi do
            dw 294,196,262,294,196,262                   ;re so do re so do
    time    dw 2,2,2,2,2,2,2,2        ;do re mi do do re mi do
            dw 2,2,4,2,2,4            ;mi fa sol mi fa sol
            dw 1,1,1,1,2,2            ;sol la sol fa mi do
            dw 1,1,1,1,2,2            ;sol la sol fa mi do
            dw 2,2,4,2,2,4            ;re so do re so do
dataseg        ends
;
prog    segment
main     proc     far
    assume cs:prog, ds:dataseg
start:
    push     ds
    mov     ax, 0
    push     ax
    mov     ax, dataseg
    mov     ds, ax
    lea     di, freq
    lea     si, time
    mov     cx, 32d
new_one:
    ;请在此处补充
    call     sound
    ;请在此处补充
    ;请在此处补充
    ;请在此处补充
    loop    new_one
    ;请在此处补充
    ;请在此处补充
    ret
main     endp
;
sound    proc    near
    in         al, 61h
    mov     bx, word ptr [si]
    push     ax
    mov     ax, word ptr [di]
    mul        bx
    mov     bx, ax
    ;;;
    ;请在此处补充你认为正确的代码
    ;;;
    pop        ax
    and     al, 11111100b
sing:
    ;请在此处补充
    out     61h, al
    ;请在此处补充
    ;请在此处补充
    call     widt
    ;请在此处补充
    ;请在此处补充
    mov     cx, dx            ; the number of loop instruction
waits:
    loop     waits
```

```
    dec     ;请在此处补充
    jnz     sing
    and     al, 11111100b
    out     61h, al
    ret
sound    endp
;
widt    proc    near
    mov     ax, 2801
    ;;;
    ;请在此处补充你认为正确的代码
    ;;;
    ret
widt    endp
prog    ends
    end     start
```

- 阅读理解调试程序 例3.3
  - 学习更改显示器显示内容的方式
  - 理解 video 数据段
    - video 数据段存储值的意义
    - video 数据段定义方式和普通数据段的不同
    - video 数据段与显示器显示之间的关系
  - 调试程序，尝试不同的显示缓冲区（单色显示缓冲区和彩色显示缓冲区）、不同的显示属性，理解实验结果

实验 8

实验时间：

- 2020.11.30：第 13 周，周一， 1~2 节

实验标题：

- 实验8：实验3.4

实验内容：

- 完成 实验 3.4
  - 进一步加强对显示器 I/O 程序的学习
  - 学习，理解宏汇编的概念按照实验要求设计程序中执行相应功能的程序段，要求将这些程序段定义成宏指令熟悉功能程序段定义成宏指令的方法和步骤
  - 掌握宏，宏指令，宏参数，宏展开等相关知识
  - 熟悉 BIOS 调用的方式进行输入字符，熟悉输入字符中扫描码和字符码的区分读取方向键等键盘输入信息的时候可以使用16h中断（键盘I/O调用）
  - 熟悉 BIOS 调用的方式进行卷屏，包括卷屏的参数，如左上角、右下角、卷屏行数、参数（前景色背景色）
  - 熟悉 BIOS 调用的方式进行输出字符，包括输出字符的参数，如显示页、属性、字符重复次数
  - 熟悉 BIOS 调用的方式进行光标定位，包括其参数，如显示页、横轴坐标
  - 调试程序：可以自己尝试不同的显示缓冲区改变字体，背景的显示属性，了解一些常见颜色的属性例子： 1 101 1011b = 1(闪烁显示) + 101(背景RGB) + 1011(前景IRGB)

实验 9

实验时间：

- 2020.12.2：第 13 周，周三， 3~4 节

实验标题：

- 实验9：示例3.6、示例3.8、示例3.9

实验内容：

- 阅读理解调试程序 例 3.6
  - 通过查表熟悉相关设置中断向量的方法通过设置中断向量的方法来自定义键盘输入的中断处理程序，熟悉在硬件接口的基础上编写键盘输入程序
  - 通过环形缓冲区，类比理解高级程序设计语言中 IO Buffer 的底层实现
  - 2020-12-14 10:54:41 补充，该实验中，如果调低了 DOSBox 的 Cycles，能复现示例代码里的一个 Bug：
    - 现象：快速敲击键盘打下若干可打印字符，最后按下一个控制字符，会导致无字符输出，程序直接退出。
    - 定位：在写缓冲区的中断子程序时，有一个 flag 变量直接置 1。外层代码中，有逻辑特判这个 flag 为退出控制。
    - 原因：时钟频率低时，快速敲键盘使 CPU 一直在响应中断来处理写缓冲区，无法调度处理输出。停止敲键盘后，开始处理输出，但特判了 flag 而直接退出。
- 阅读理解调试程序 例 3.8，**并补全相关注释**
  - 熟悉文字处理程序的设计逻辑，包括对于其各行字符数数组的设计、使用，输入缓冲的使用，每次屏幕操作后重新显示等具体的逻辑及其汇编实现。

```
; 字处理演示程序wspp
; 支持光标插入和左右移动

dseg    segment
kbd_buf    db    96    dup(' ') ; 输入缓冲
cntl    db    16    dup(0)   ; 每一行的字符数
bufpt    dw    0            ; buffer 头指针
buftl    dw    0            ; buffer 尾指针
colpt    db    0            ; 光标所在列
rowpt    db    0            ; 光标所在行
rowmx    dw    0            ; 一个字的最大输入行
dseg    ends

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
curs    macro    row,col
    mov        dh,    row
    mov        dl,    col
    mov        bh,    0
    mov        ah,    2
    int        10h
endm

cseg    segment
main    proc    far
        assume    cs:cseg,ds:dseg,es:dseg
start:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ax, dseg
```

```
    mov        ds, ax
    mov        es, ax
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        buftl, 0
    mov        colpt, 0
    mov        rowpt, 0
    mov        bufpt, 0
    mov        rowmx, 0
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        cx, length cntl; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    xor        al, al         ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    lea        di, cntl
    cld
    rep        stosb
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ah, 6
    mov        al, 0
    mov        cx, 0
    mov        dh, 24
    mov        dl, 79
    mov        bh, 07
    int        10h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    curs    0,0
read_k:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ah, 0
    int        16h
    cmp        al, 1bh  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    jnz        arrow
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ah, 4ch
    int        21h
arrow:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp        ah, 4bh
    jz        left
    cmp        ah, 4dh
    jz        right
inst:
    jmp        ins_k
left:
    jmp        left_k
right:
    jmp        right_k
ins_k:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bx, bufpt
    mov        cx, buftl
    cmp        bx, cx
    je        km
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    lea        di, kbd_buf
    add        di, cx
    mov        si, di
    dec        si
    sub        cx, bx
    std
    rep        movsb
km:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
```

```
    mov        kbd_buf[bx], al
    inc        bufpt       ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    inc        buftl       ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp        al, 0dh
    jnz        kn
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    lea        si, cntl
    add        si, rowmx
    inc        si
    mov        di, si
    inc        di
    mov        cx, rowmx
    sub        cl, rowpt
    std
    rep        movsb
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bl, rowpt
    xor        bh, bh
    mov        cl, colpt
    mov        ch, cntl[bx]
    sub        ch, colpt
    mov        cntl[bx], cl
    mov        cntl[bx+1], ch
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ax, rowmx   ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bh, 7       ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ch, rowpt   ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        dh, 24      ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        cl, 0       ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        dl, 79      ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ah, 6
    int        10h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    inc        rowpt
    inc        rowmx
    mov        colpt, 0
    jmp        short kp
kn:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bl, rowpt
    xor        bh, bh
    inc        cntl[bx]  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    inc        colpt     ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
kp:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    call    dispbf
    curs    rowpt, colpt
    jmp        read_k
left_k:
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp        colpt,0
    jnz        k2
    cmp        rowpt,0
    jz        lret      ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    dec        rowpt
    mov        al, rowpt
    lea        bx, cntl
    xlat    cntl
```

```
    mov        colpt, al
    jmp        k3
k2:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    dec        colpt
k3:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    dec        bufpt
    curs    rowpt,colpt
lret:
    jmp        read_k      ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
right_k:
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bx, bufpt
    cmp        bx, buftl
    je        rret
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    inc        colpt
    cmp        kbd_buf[bx], 0dh
    jnz        k4
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    inc        rowpt
    mov        colpt,0
k4:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    inc        bufpt
    curs    rowpt,colpt
rret:
    jmp        read_k      ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情

dispbf  proc      near
    mov        bx, 0
    mov        cx, 96
    curs    0, 0
disp:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        al, kbd_buf[bx] ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    bx
    mov        bx, 0700        ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ah, 0eh
    int        10h
    pop        bx
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp        al, 0dh
    jnz        kk
    mov        al, 0ah
    mov        ah, 0eh
    int        10h
kk:
    inc        bx
    loop    disp
    ret
dispbf    endp

main    endp
cseg    ends
        end    start
```

- 阅读理解调试程序 例 3.9 , **并补全相关注释**

```
; samp3.9

stack    segment  para  stack  'stack'
db       256    dup(0)
top      label    word
stack    ends

data     segment  para  public  'data'
buffer     db     16h dup(0)
bufpt1     dw     0
bufpt2     dw     0
kbflag     db     0
prompt     db     '     * PLEASE PRACTISE TYPING *',0dh,0ah,'$'
scantab    db     0,0,'1234567890-=',8,0
           db     'qwertyuiop[]',0dh,0
           db     'asdfghjkl;',0,0,0,0
           db     'zxcvbnm,./',0,0,0
           db     ' ',0,0,0,0,0,0,0,0,0,0,0,0,0,0
           db     '789-456+1230.'
even
oldcs9     dw     ?
oldip9     dw     ?
str1       db     'abcd efgh ijkl mnop qrst uvwx yz.'
            db     0dh,0ah,'$'
str2       db     'christmas is a time of joy and love.'
            db     0dh,0ah,'$'
str3       db     'store windows hold togs and gifts.'
            db     0dh,0ah,'$'
str4       db     'people send christmas cards and gifts.'
            db     0dh,0ah,'$'
str5       db     'santa wish all people peace on earth.'
crlf       db     0dh,0ah,'$'
colon      db     ':','$'
even
saddr      dw     str1,str2,str3,str4,str5
count      dw     0
sec        dw     0
min        dw     0
hours      dw     0
save_lc    dw  2  dup(?)
data       ends

code       segment
           assume     cs:code,ds:data,es:data,ss:stack
main       proc    far
start:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ax, stack
    mov     ss, ax
    mov     sp, offset top
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    ds
    sub     ax, ax
    push    ax
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ax, data
    mov        ds, ax
    mov        es, ax
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
```

```
    mov        ah, 35h ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        al, 09h ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    int        21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        oldcs9,  es
    mov        oldip9,  bx

    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    ds
    mov        dx, seg kbint
    mov        ds, dx
    mov        dx, offset kbint
    mov        al, 09h
    mov        ah, 25h
    int        21h
    pop        ds
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        ah, 35h ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        al, 1ch ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    int        21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        save_lc, bx
    mov        save_lc+2, es

    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    ds
    mov        dx, seg clint;
    mov        ds, dx
    mov        dx, offset clint
    mov        al, 1ch
    mov        ah, 25h
    int        21h
    pop        ds

    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    in        al, 21h
    and        al, 11111100b
    out        21h, al

first:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov     ah, 0
    mov     al, 3
    int     10h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov     dx, offset prompt
    mov     ah, 9
    int     21h

    mov     si, 0
next:
    mov     dx, saddr[si]
    mov     ah, 09h
    int     21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov     count, 0
    mov     sec, 0
    mov     min, 0
    mov     hours, 0
    sti
```

```
forever:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    call    kbget
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    test    kbflag, 80h
    jnz     endint
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    ax
    call    dispchar
    pop     ax
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp     al, 0dh
    jnz     forever
    mov     al, 0ah ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    call    dispchar; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    call    disptime; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    lea     dx, crlf
    mov     ah, 09h
    int     21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    add     si, 2
    cmp     si, 5*2
    jne     next
    jmp     first
endint:
    cli
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    ds
    mov     dx, save_lc
    mov     ax, save_lc+2
    mov     ds, ax
    mov     al, 1ch
    mov     ah, 25h
    int     21h
    pop     ds
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    ds
    mov     dx, oldip9
    mov     ax, oldcs9
    mov     ds, ax
    mov     al, 09h
    mov     ah, 25h
    int     21h
    pop     ds

    sti
    ret
main   endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
clint  proc    near
    push    ds
    mov     bx, data
    mov     ds, bx
    lea     bx, count
    inc     word ptr[bx]
    cmp     word ptr[bx], 18
    jne     return
    call    inct
```

```
adj:
    cmp     hours, 12
    jle     return
    sub     hours, 12
return:
    pop     ds
    sti
    iret
clint  endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
inct   proc    near
    mov     word ptr[bx], 0
    add        bx, 2
    inc     word ptr[bx]
    cmp     word ptr[bx], 60
    jne     exit
    call    inct
exit:
    ret
inct   endp

disptime    proc     near
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov     ax, min
    call    bindec

    mov     bx, 0
    mov     al, ':'
    mov     ah, 0eh
    int     10h

    mov     ax, sec
    call    bindec

    mov     bx, 0
    mov     al, ':'
    mov     ah, 0eh
    int     10h

    mov     bx, count
    mov     al, 55d
    mul     bl
    call    bindec

    ret
disptime    endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
bindec     proc   near
    mov    cx, 100d
    call   decdiv
    mov    cx, 10d
    call   decdiv
    mov    cx, 1
    call   decdiv
    ret
bindec     endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
decdiv     proc   near
```

```
    mov    dx, 0
    div    cx
    mov    bx, 0
    add    al, 30h
    mov    ah, 0eh
    int    10h
    mov    ax, dx
    ret
decdiv    endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
kbget    proc    near
    push   bx
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cli
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bx, bufpt1 ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp        bx, bufpt2 ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    jnz        kbget2
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp        kbflag, 0
    jnz        kbget3     ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    sti                ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    pop     bx
    jmp     kbget
kbget2:
    mov        al, [buffer+bx] ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    inc        bx
    cmp        bx, 16h
    jc        kbget3
    mov        bx, 0      ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
kbget3:
    mov        bufpt1, bx ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    pop        bx
    sti               ; 这行实验书里没有，不加也不影响功能，加了更有助于可读性吧
    ret
kbget endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
kbint  proc    far
    push    bx  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    push    ax

    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    in        al, 60h
    push    ax
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    in      al, 61h
    or      al, 80h
    out     61h, al
    and     al, 7fh
    out     61h, al
    pop        ax      ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情

    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    test    al, 80h
    jnz        kbint2
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bx, offset scantab
```

```
    xlat    scantab ;
    cmp        al, 0
    jnz     kbint4
    mov        kbflag, 80h
    jmp        kbint2
kbint4:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov        bx, bufpt2
    mov        [buffer+bx], al
    inc        bx
    cmp     bx, 16h  ; 实验书上的一个 bug，没有加 h
    jc        kbint3
    mov        bx, 0
kbint3:
    cmp        bx, bufpt1
    jz        kbint2
    mov        bufpt2, bx
kbint2:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cli
    mov        al, 20h ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    out        20h, al
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    pop        ax
    pop        bx
    sti
    iret
kbint    endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
dispchar    proc    near
    push    bx
    mov        bx, 0
    mov        ah, 0eh
    int        10h
    pop        bx
    ret
dispchar    endp

code    ends
        end     start
```

~~实验 10（6 个学时）~~

实验时间：

- 2020.12.7：第 14 周，周一， 1~2 节
- 2020.12.9：第 14 周，周三， 3~4 节
- 2020.12.14：第 15 周，周一， 1~2 节

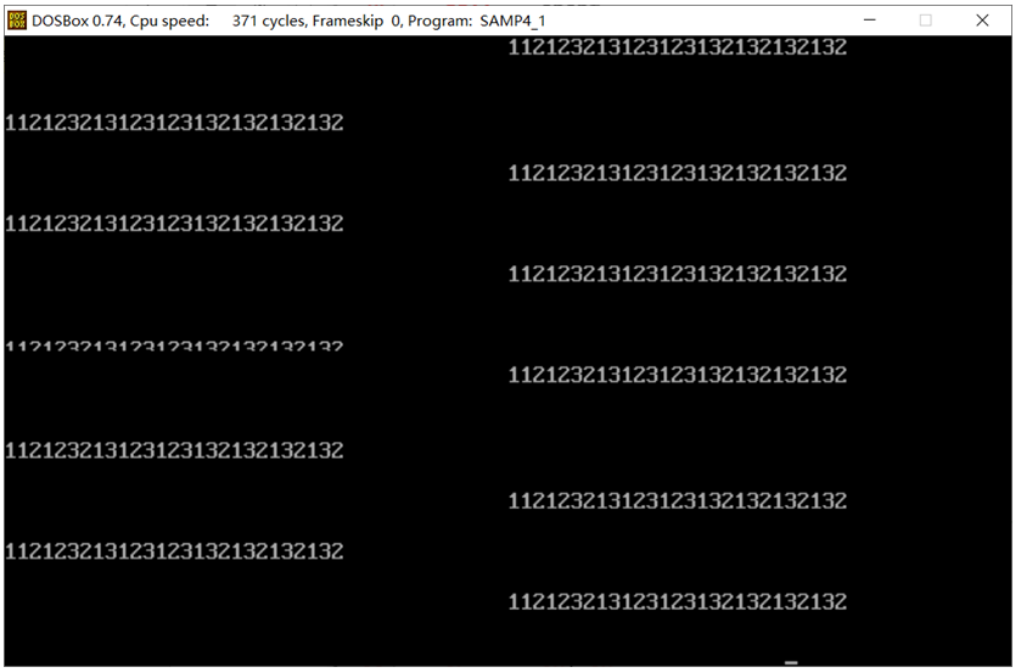
实验标题：

- 实验10：示例4.1、示例4.2、实验4.1
- 3 个子任务权重分别为： 3:2:5 ，这在 https://oj.qd.sdu.edu.cn/contest/8 中也有体现。

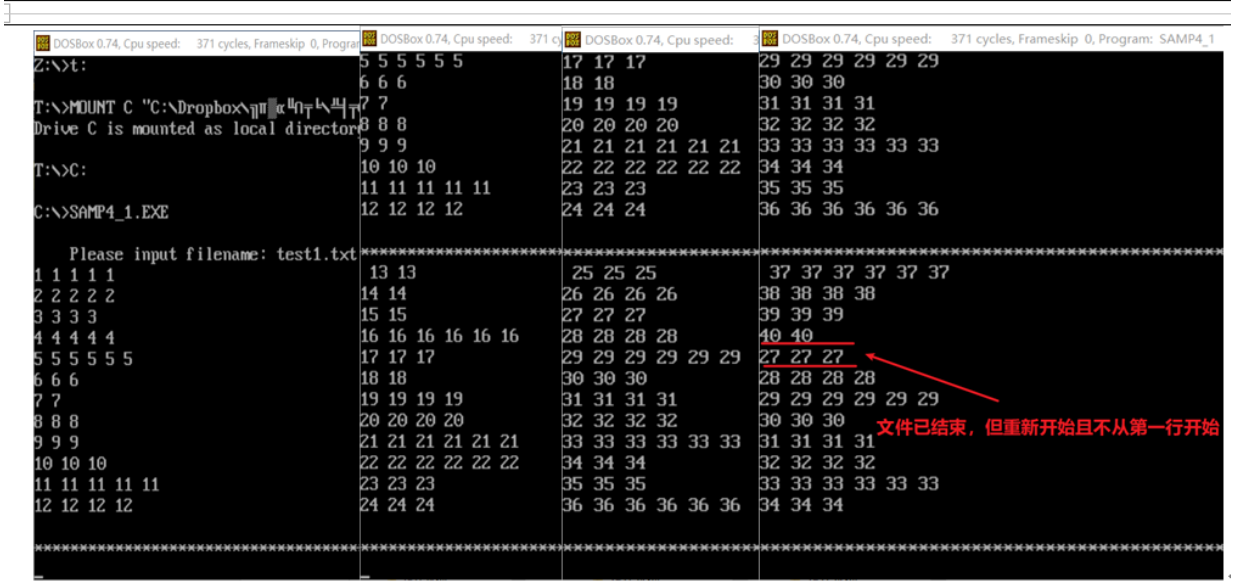
实验内容：

- 阅读理解调试程序 例 4.1 ，修复 Bug，最后在 SDUOJ 上交题
  - 学习有关打开文件，读文件的系统调用，理解文件读取的过程。
  - 查阅教材，理解文件代号和错误返回码。
  - 课本代码中存在的 bug：

原未经过修改的 samp4_1 程序，在测试数据为一行时会出现如同下图的死循环：



在测试数据为多行时，使用空格进行分页输出，程序仍然无法在有限次数内停止：



  - 请读懂程序，修复这个 Bug，给出你的修复方案
    - 查找、定位该 Bug（之前介绍的 Debug 使用）
    - 分析 Bug 原因
    - 制定 Bugfix 方案
    - 实施、验证

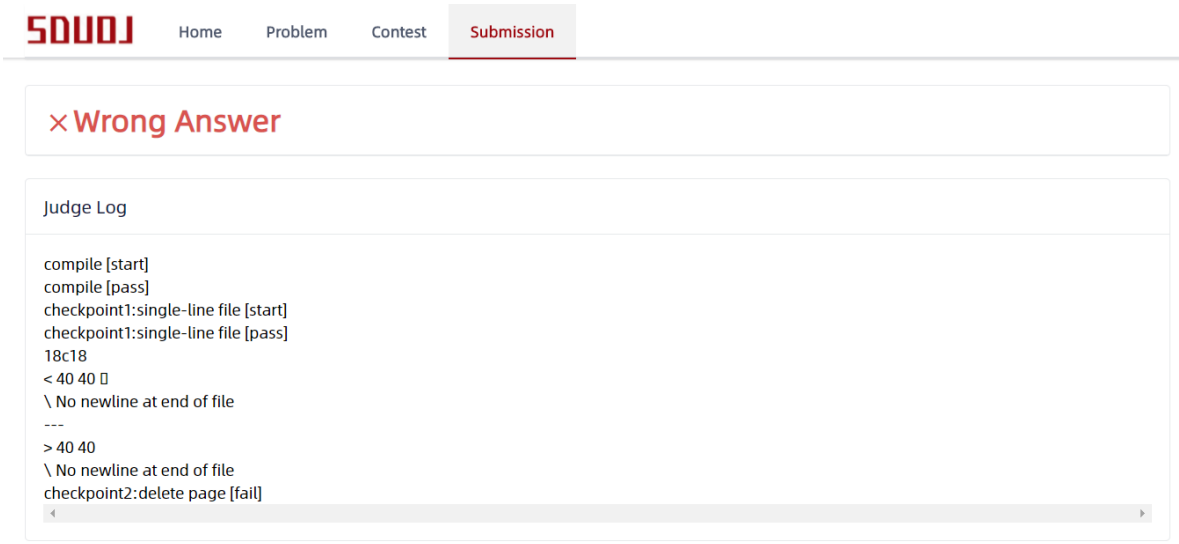
- 完成后在 SDUOJ 上进行提交，链接：https://oj.qd.sdu.edu.cn/contest/8（初始账号密码都是学号）
- 给出一个数据段如下，可自行地任意修改：

```
data    segment
Pgsize            dw    ?

buf_size        db    80
s_buf            db    ?
buf               db    200 dup(?)

cur                 dw    ?
handle          dw    ?
mess_getname    db  0dh,0ah,"    Please input filename: $"
mess_err1        db  0ah,0dh,"    Illegal filename ! $"
mess_err2        db  0ah,0dh,"    File not found !$"
mess_err3        db  0ah,0dh,"    File read error !$"
mess_psize        db  0ah,0dh,"    Page Size : $"
crlf            db  0ah,0dh,"$"
mess_star        db  0ah,0dh,"*******************************************"
                db  0ah,0dh,"$"
data    ends
```

- 阅读理解调试程序 例 4.2，**并补全相关注释**，修复 Bug，最后在 SDUOJ 上交题
  - 学习建立文件、写文件、关闭文件的系统调用，理解删除页的过程。
  - 实验书 pdf 和 实验书实体书 印刷有些许差异， 例 4.2 代码在本实验指导书中有给出
  - 任务：尽管作为实验书上印刷的程序，但该程序中也有 Bug，请你修复
    a. 例 4.1 中提到的 bug
  - 在子程序 show_and_reserve 中，对 1ah 进行特判，是为了兼容一些文件系统以 1ah 作为文件结束符的规范。如果在你测试输入文件中间加入 1 个 1ah 字符，那么正常的现象应该是后面的字符都不显示了。在 DOSBox 中，不以 1ah 作为文件结束符，此时这个特判是 dead code。
    - 但如果你在修复 例 4.1 的 Bug 时采用了某些不 perfect 的方案，并且这种方案在写临时文件的时候，会在末尾多带上 1ah 字符。那么在这次实验中，会导致你在 OJ 上得到这个结果，其中的 40 40 方块 中的 方块 就是 1ah 这个非打印字符。



- 写完后在 SDUOJ 上进行提交，链接：https://oj.qd.sdu.edu.cn/contest/8（初始账号密码都是学号）

```
; samp4_2
data    segment
Pgsize            dw    ?

buf_size        db    80
s_buf            db    ?
buf               db    200 dup(?)

names            db    20 dup(?)
cur                 dw    ?
handle          dw    ?
buf_tmp            db  24*80 dup(?)
cur_tmp            dw    ?
name_tmp        db  "t0m1p",0
handle_tmp        dw    ?
mark            db  ?
mess_getname    db  0dh,0ah,"    Please input filename: $"
mess_err1        db  0ah,0dh,"    Illegal filename ! $"
mess_err2        db  0ah,0dh,"    File not found !$"
mess_err3        db  0ah,0dh,"    File read error !$"
mess_psize        db  0ah,0dh,"    Page Size : $"
mess_dele        db  0dh,0ah,"    The last page is deleted !"
crlf            db  0ah,0dh,"$"
mess_star        db  0ah,0dh,"*********************************************"
                db  0ah,0dh,"$"
data    ends

code    segment
      assume ds:data, cs:code
main proc far
start:
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  push    ds
```

```
  sub        ax, ax
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  push    ax
  mov        ax, data
  mov        ds, ax
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        mark, 0
  mov        PgSize, 12
  mov        cur, 200
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  call    getline
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  call    openf
  or        ax, ax ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  jnz        display
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset mess_err2
  mov        ah, 09h
  int        21h
  jmp        file_end
display:
  mov        cx, Pgsize ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所
  mov        cur_tmp, 0
show_page:
  call    read_block ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的
  or        ax, ax
  jnz        next2
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset mess_err3
  mov        ah, 09h
  int        21h; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  jmp        file_end
next2:
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  call    show_and_reserve
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  or        bx, bx
  jz        file_end
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  or        cx, cx
  jnz        show_page
  mov        dx, offset mess_star
  mov        ah, 09h
  int        21h
   ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
wait_space:
  mov        ah, 1
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
```

```
  cmp        al, " "
  jnz        psize
  call    write_buf_tmp
  jmp        display
psize:
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  cmp        al,"p"
  jnz        delete
  call    write_buf_tmp
  call    change_psize
  jmp        stick
delete:
  cmp        al, "d"
  jnz        wait_space
  mov        mark, 1    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset mess_dele
  mov        ah, 09h
  int        21h
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
stick:
  mov        ah, 1
  int        21h
  cmp        al, " "
  jnz        stick
  jmp        display
file_end:
  call    write_buf_tmp
  cmp        mark,0
  jz        ok
  call    write_tmp_back ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所
ok:
  ret
main    endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
change_psize proc near
  push    ax
  push    bx
  push    cx
  push    dx
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset mess_psize
  mov        ah, 09h
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        ah, 01
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  cmp        al, 0dh
```

```
  jz        illeg
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  sub        al, "0"
  mov        cl, al
getp:
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        ah, 1
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  cmp        al, 0dh
  jz        pgot
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  sub        al, "0"
  mov        dl, al
  mov        al, cl
  mov        cl, dl
  mov        bl, 10
  mul        bl
  add        cl, al
  jmp        getp
pgot:
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dl, 0ah
  mov        ah, 2
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  cmp        cx, 0
  jle        illeg
  cmp        cx, 24
  jg        illeg
  mov        PgSize, cx
illeg:
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dl, 0ah
  mov        ah, 2          ; 实验书pdf 和 实体书不一样的地方
  int        21h
  pop        dx
  pop        cx
  pop        bx
  pop        ax
  ret
change_psize endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
openf proc near
  push    bx
  push    cx
  push    dx
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset names ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程
```

```
  mov        al, 2            ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程
  mov        ah, 3dh
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        handle, ax
  mov        ax, 0
  jc        quit        ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset name_tmp
  mov        cx, 0
  mov        ah, 3ch
  int        21h
  mov        handle_tmp, ax
  jc        quit
  mov        ax, 1
quit:
  pop        dx
  pop        cx
  pop        bx
  ret
openf  endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
getline  proc near
  push    ax
  push    bx
  push    cx
  push    dx
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset mess_getname
  mov        ah, 09h
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset buf_size
  mov        ah, 0ah
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        dx, offset crlf
  mov        ah, 09h
  int        21h
   ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        bl, s_buf
  mov        bh, 0
  mov        names[bx],0
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
name_move:
  dec        bx
  mov        al, buf[bx]
  mov        names[bx], al
  jnz        name_move
```

```
  pop        dx
  pop        cx
  pop        bx
  pop        ax
  ret
getline  endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
read_block    proc    near
  push    bx
  push    cx
  push    dx
  mov        ax, 1
  cmp        cur, 200
  jnz        back
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        cx, 200        ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序
  mov        bx, handle     ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序
  mov        dx, offset buf ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序
  mov        ah, 3fh
  int        21h
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  mov        cur, 0
  mov        ax, 1
  jnc        back           ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序
  mov        cur, 200
  mov        ax, 0
back:
  pop        dx
  pop        cx
  pop        bx
  ret
read_block endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
show_and_reserve proc near
  push    ax
  push    dx
  mov        bx, cur
  mov        bp, cur_tmp
loop1:
  cmp        bx, 200
  jl        lp
  jmp        exit            ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程
lp:
  mov        dl, buf[bx]
  mov        ds:buf_tmp[bp], dl; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的
  ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
  inc        bx
  inc        cur
```

```
    inc         bp
    inc       cur_tmp
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp         dl, 1ah
    jz         exit_eof
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         ah, 02
    int         21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    cmp         dl, 0ah
    jz         exit_ln
    jmp         loop1
exit_eof:
    mov         bx,0
exit_ln:
    dec         cx ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
exit:
    pop         dx
    pop         ax
    ret
show_and_reserve endp

; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
write_buf_tmp proc near
    push    ax
    push    bx
    push    cx
    push    dx
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         dx, offset buf_tmp
    mov         cx, cur_tmp
    mov         bx, handle_tmp
    mov         ah, 40h
    int         21h
    pop         dx
    pop         cx
    pop         bx
    pop         ax
    ret
write_buf_tmp endp



write_tmp_back  proc near
    push    ax
    push    bx
    push    cx
    push    dx
; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         bx, handle_tmp
    mov         ah, 3eh
```

```
    int         21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         bx, handle
    mov         ah, 3eh
    int         21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         dx, offset name_tmp
    mov         al, 0
    mov         ah, 3dh
    int         21h
    mov         handle_tmp, ax
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         dx, offset names
    mov         al, 1
    mov         ah, 3dh
    int         21h
    mov         handle,ax

    mov         si, 1
wrt_back:
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         bx, handle_tmp
    mov         ah, 3fh
    mov         cx, 200
    mov         dx, offset buf
    int         21h
    jc         wrt_end         ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情

    mov         si, ax          ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
       ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    mov         bx, handle
    mov         ah, 40h
    mov         cx, si           ; 实验书pdf 和 实体书不一样的地方
    mov         dx, offset buf
    int         21h
    ; 在这里加一行注释，来表示这行代码或接下来几行代码或接下来的程序段所做的事情
    or         si, si
    jnz         wrt_back
    mov         ah, 3eh
    mov         bx, handle
    int         21h
wrt_end:
    pop         dx
    pop         cx
    pop         bx
    pop         ax
    ret
write_tmp_back endp

code    ends
        end    start
```

- 完成 实验 4.1
    - 学习移动文件指针的系统调用。区分文件指针移动的方式，针对 实验 4.1 选择合适的移动方式，灵活使用文件指针（不局限于文件指针）
    - 在通过 OJ 上的评测的无 Bug 示例 4.2 代码上进行功能新增，要求详见实验书

- 完成后在 SDUOJ 上进行交题，链接：https://oj.qd.sdu.edu.cn/contest/8（初始账号密码都是学号）
- 给出一个数据段如下，可自行地任意修改：

```
data    segment
Pgsize            dw    ?
```