实验报告

1. 实验内容

- 大数相乘

  要求实现两个十进制大整数的相乘，输出乘法运算的结果

- C语言库函数

  C库函数的底层分析和实现(printf / scanf / sctrcpy)分析C库函数的底层实现的细节，并用汇编语言实现相应的功能

- C语言编写多重循环

  使用C语言编写多重循环的代码，并反汇编生成对应的汇编代码，分析各条语句的功能，并用汇编语言编写相同的功能的程序

- 计算器功能

  实现完善的计算器功能的实现

- 随机数生成

  通过查阅资料，实现对应的随机数生成的汇编代码

- 文本内容比对

  实现两个文本的文本内容的比对，如果两个文件的内容一致，输出相应的提示，如果两个文件的内容不一致，输出不一致对应的行号

- 数据块复制

  实现内存与外存或者内存之间的大数据块的复制，尽可能对代码进行性能优化，在界面上输出显示转移的数据量和对应的执行的时间

2. 实验要求

- 大数相乘

  通过大数相乘的汇编上机实验，掌握

  - 数据寻址方式
  - 循环的使用
    - LOOP指令
    - JMP指令
  - 条件伪指令
  - 中断调用指令
  - 模块化的子程序设计

- C库函数

  通过实现部分的C的库函数，掌握

  - 字符串操作
  - 中断的熟悉

- C语言多重循环

  通过实现多重循环，掌握

  - 基本的循环结构的实现
    - JMP
    - LOOP
  - 反汇编代码的具体解释

    - 了解真是的汇编代码的实现的多重循环的机制
- 计算器

  通过实现计算器，掌握

    - 自学了解586模式下的FPU的相关的浮点操作
- 随机数

  通过编写随机数，掌握

    - CPU时钟的读取和计时使用方式
    - 循环实现线性同余算法
- 文本内容的比对

  通过编写文本内容比对程序，掌握

    - 文件读取的DOS中断调用
    - 字符串拷贝函数
    - 循环扫描字符串的操作
- 数据块的赋值

  通过编写数据块的复制，掌握

    - 外存到内存的数据的调用
    - 内存中的数据块的操作

3. 实验过程

- 大数相乘
    1. 准备
        - 代码框架的准备
        - 子结构的设计
            - 主逻辑子程序

              模拟手工计算的双层循环框架
            - 回车换行子程序封装
            - 数据初始化

              将输入的数据初始化成对应的ASCII码值
            - 数据输出子程序
            - 数据输入子程序
            - 数组翻转子程序
            - 符号位处理子程序
            - 测试子程序
        - 数据的设计
            - 加数数组1
            - 加数数组2
            - 结果数组
            - 加法和乘法进位位
            - 符号位
    2. 编写
    3. 困难
        - 主逻辑调试的时候出现了近转移失败的情况
        - 测试代码出现失误
        - 寄存器不够使用
    4. 解决

        - 重新多次编写适合问题的测试子程序，审核了初始化子程序
        - 调试的时候加入了远转移
        - 采用堆栈保存中间结果的做法节省寄存器的使用
- C库函数
    1. 准备
        - 测试子程序中，编写对于输出数值的子程序 PRINTAX
    2. 编写
    3. 困难
        - 测试子程序中出现了错误
    4. 解决
        - 检验更改了测试主程序
- C语言多重循环
    1. 准备
        - 编写简单的C语言的双重循环的代码进行反汇编
        - 学习了解反汇编的代码的本质
        - 反汇编代码

```
int main()
{
01011850  push        ebp
01011851  mov         ebp,esp
01011853  sub         esp,0E4h
01011859  push        ebx
0101185A  push        esi
0101185B  push        edi
0101185C  lea         edi,[ebp-0E4h]
01011862  mov         ecx,39h
01011867  mov         eax,0CCCCCCCCh
0101186C  rep stos    dword ptr es:[edi]
    int num = 0;
0101186E  mov         dword ptr [num],0
    for (int i = 0; i < 5; i++)
01011875  mov         dword ptr [ebp-14h],0
0101187C  jmp         main+37h (01011887h)
0101187E  mov         eax,dword ptr [ebp-14h]
01011881  add         eax,1
01011884  mov         dword ptr [ebp-14h],eax
01011887  cmp         dword ptr [ebp-14h],5
0101188B  jge         main+62h (010118B2h)
    {
        for (int j = 0; j < 5; j++)
0101188D  mov         dword ptr [ebp-20h],0
01011894  jmp         main+4Fh (0101189Fh)
01011896  mov         eax,dword ptr [ebp-20h]
01011899  add         eax,1
0101189C  mov         dword ptr [ebp-20h],eax
0101189F  cmp         dword ptr [ebp-20h],5
010118A3  jge         main+60h (010118B0h)
        {
            num++;
010118A5  mov         eax,dword ptr [num]
010118A8  add         eax,1
010118AB  mov         dword ptr [num],eax
        }
010118AE  jmp         main+46h (01011896h)
    }
010118B0  jmp         main+2Eh (0101187Eh)
    printf("%d\n", num);
010118B2  mov         eax,dword ptr [num]
010118B5  push        eax
010118B6  push        offset string "%d\n" (01017B30h)
010118BB  call        _printf (0101133Eh)
010118C0  add         esp,8
}
010118C3  xor         eax,eax
010118C5  pop         edi
010118C6  pop         esi
010118C7  pop         ebx
010118C8  add         esp,0E4h
010118CE  cmp         ebp,esp
010118D0  call        __RTC_CheckEsp (01011127h)
010118D5  mov         esp,ebp
010118D7  pop         ebp
010118D8  ret
```

2. 编写

3. 困难
4. 解决

- 计算器
  1. 准备
     - 查阅了解有关于浮点运算的相关指令细节
     - 查阅了解相对的计算的核心的函数指令(浮点)
- 随机数
  1. 准备
     - 查阅资料，了解了线性同余算法生成伪随机数

$$X_{n+1} = (A \cdot X_n + B)\%C \quad A, B, C \ is \ the \ prime$$

- 文本内容的比对
  1. 准备
     - 查询对应的文件中断**21H**
       - 文件打开
       - 文件读取
       - 文件关闭
     - 对于行的判断问题

       引入行记录(内存的存储单元)进行实时的记录
     - 为了保证程序的鲁棒性

       对与文本的内容一次性的全部读入内存进行处理，减少因为行的的不一致导致的错误
  2. 编写
  3. 困难
     - 在程序编写完的时候，频繁出现了大量的乱码错误
  4. 解决
     - 对于乱码的不正确读取问题，很快的意识到了可能是因为的文件读取的句柄被覆盖的问题
       1. 检查发现，对于中断的理解不是很深刻，有查看了大量的文档，发现了保存句柄的寄存器
       2. 对与保存句柄的重要寄存器内容进行栈操作 PUSH, POP保存现场
       3. 再次运行，错误消失
- 数据块的赋值
  1. 准备
     - 明确实验的过程
       1. 采用内存到内存的数据块的复制
       2. 将数组的内容一次性的拷贝到另外的一个数组中

4. 源代码

- 大数相乘

```
; 大数相乘
; Author : GMFTBY
; Time   : 2017.12.1

; 数据段定义
DATA    SEGMENT
IDDA1   DB      100
        DB      ?
        DB      100 DUP(0)
IDDA2   DB      100
        DB      ?
        DB      100 DUP(0)
RESULT  DB      205
        DB      ?
        DB      205 DUP(0)
MULTIFLAG       DW      ?
ADDFLAG         DW      ?
; IDDA1 / IDDA2 / RESULT 的符号位
IDDA1FLAG       DB      ?
IDDA2FLAG       DB      ?
DATA    ENDS

; 代码段定义
CODE    SEGMENT
MAIN    PROC    FAR
        ASSUME  CS:CODE,DS:DATA,ES:NOTHING
        PUSH    DS
        XOR     AX,AX
        PUSH    AX
        MOV     AX,DATA
        MOV     DS,AX

        ; 主逻辑
        CALL    MYIN1
        CALL    MYIN2
        CALL    INIT
        CALL    REVERSE1
        CALL    REVERSE2
        CALL    CALCULATE
        CALL    REVERSE3
        ; CALL    PRINT1
        ; CALL    PRINT2
        CALL    PRINT3

        RET
MAIN    ENDP

; 主逻辑
CALCULATE   PROC
        ; 外层循环使用BX计数调节，内层循环使用BP计数调节
        MOV     BX,0
        LOOP_OUT:
                MOV     MULTIFLAG,0
                MOV     ADDFLAG,0
                MOV     BP,0
                LOOP_IN:
                        MOV     AL,[IDDA1+2+BP]
                        MUL     [IDDA2+2+BX]

                        ADD     AX,MULTIFLAG    ; 计算结果在AX
```

```
                        MOV     DL,10
                        DIV     DL
                        PUSH    AX
                        XOR     AH,AH
                        MOV     MULTIFLAG,AX    ; 求取新的乘法进位
                        POP     AX
                        MOV     AL,AH
                        XOR     AH,AH           ; temp1是余数(AX)
                        MOV     SI,BX
                        ADD     SI,BP
                        ADD     AL,[RESULT+2+SI]
                        ADD     AX,ADDFLAG
                        MOV     DL,10
                        DIV     DL
                        PUSH    AX
                        XOR     AH,AH
                        MOV     ADDFLAG,AX
                        POP     AX
                        MOV     AL,AH
                        XOR     AH,AH           ; temp2是余数(AX)
                        MOV     SI,BX
                        ADD     SI,BP
                        MOV     [RESULT+2+SI],AL
                        INC     BP
                        MOV     DL,[IDDA1+1]
                        XOR     DH,DH
                        CMP     BP,DX
                        JNE     LOOP_IN
                MOV     AX,MULTIFLAG
                ADD     AX,ADDFLAG
                XOR     AH,AH
                MOV     SI,BX
                ADD     SI,BP
                MOV     [RESULT+2+SI],AL
                INC     BX
                MOV     DL,[IDDA2+1]
                XOR     DH,DH
                CMP     BX,DX
                JNE     LOOP_OUT
        ; 计算新串的长度
        MOV     AL,[IDDA1+1]
        ADD     AL,[IDDA2+1]
        MOV     [RESULT+1],AL
        RET
CALCULATE   ENDP

; 输出回车换行
SPACE   PROC
        MOV     AH,0EH
        MOV     AL,0DH
        INT     10H
        MOV     AH,0EH
        MOV     AL,0AH
        INT     10H
        RET
SPACE   ENDP

; 初始化数据串,ASCII -> 数据,统一减30H
INIT    PROC

        MOV     BX,0
```

```
        LOOP_DEC1:
                SUB     [IDDA1+2+BX],30H
                INC     BX
                CMP     BL,[IDDA1+1]
                JNE     LOOP_DEC1
        MOV     BX,0
        LOOP_DEC2:
                SUB     [IDDA2+2+BX],30H
                INC     BX
                CMP     BL,[IDDA2+1]
                JNE     LOOP_DEC2
        ; 清空RESULT
        MOV     BX,0
        LOOP_CLEAR:
                MOV     [RESULT+2+BX],0
                INC     BX
                CMP     BX,204
                JNE     LOOP_CLEAR
        RET
INIT    ENDP

; 输出数据串1
PRINT1  PROC
        PUSH    BX
        MOV     BX,0
        LOOP_PRINT1:
                MOV     AH,2
                MOV     DH,[IDDA1+2+BX]
                ADD     DH,30H
                MOV     DL,DH
                INT     21H
                INC     BX
                CMP     BL,[IDDA1+1]
                JNE     LOOP_PRINT1
        CALL    SPACE
        POP     BX
        RET
PRINT1  ENDP

; 输出数据串2
PRINT2  PROC
        PUSH    BX
        MOV     BX,0
        LOOP_PRINT2:
                MOV     AH,2
                MOV     DH,[IDDA2+2+BX]
                ADD     DH,30H
                MOV     DL,DH
                INT     21H
                INC     BX
                CMP     BL,[IDDA2+1]
                JNE     LOOP_PRINT2
        CALL    SPACE
        POP     BX
        RET
PRINT2  ENDP

; 输出结果串
PRINT3  PROC

        ; 输出符号
```

```
        MOV     AL,IDDA1FLAG
        MOV     BL,IDDA2FLAG
        XOR     AL,BL
        CMP     AL,0
        JE      NEXT
        MOV     AH,2
        MOV     DL,'-'
        INT     21H
        NEXT:
        ; 结果是0
        CMP     [RESULT+3],0
        JE      EN
        MOV     BX,1
        LOOP_PRINT3:
                MOV     AH,2
                MOV     DH,[RESULT+2+BX]
                ADD     DH,30H
                CMP     DH,30H
                MOV     DL,DH
                INT     21H
                INC     BX
                CMP     BL,[RESULT+1]
                JNE     LOOP_PRINT3
        CALL    SPACE
        JMP     EE
        EN:
                MOV     AH,2
                MOV     DL,30H
                INT     21H
        EE:
                RET
PRINT3  ENDP

; 翻转子函数1
REVERSE1    PROC
        MOV     CL,[IDDA1+1]
        MOV     CH,CH
        MOV     BX,0
        LOOP_PUSH1:
                MOV     AL,[IDDA1+2+BX]
                XOR     AH,AH
                PUSH    AX
                INC     BX
                LOOP    LOOP_PUSH1
        MOV     CL,[IDDA1+1]
        XOR     CH,CH
        MOV     BX,0
        LOOP_POP1:
                POP     AX
                MOV     [IDDA1+2+BX],AL
                INC     BX
                LOOP    LOOP_POP1
        RET
REVERSE1    ENDP

; 翻转子函数2
REVERSE2    PROC
        MOV     CL,[IDDA2+1]
        XOR     CH,CH

        MOV     BX,0
```

```
        LOOP_PUSH2:
                MOV     AL,[IDDA2+2+BX]
                XOR     AH,AH
                PUSH    AX
                INC     BX
                LOOP    LOOP_PUSH2
        MOV     CL,[IDDA2+1]
        XOR     CH,CH
        MOV     BX,0
        LOOP_POP2:
                POP     AX
                MOV     [IDDA2+2+BX],AL
                INC     BX
                LOOP    LOOP_POP2
        RET
REVERSE2    ENDP

; 翻转子函数3
REVERSE3    PROC
        MOV     CL,[RESULT+1]
        XOR     CH,CH
        MOV     BX,0
        LOOP_PUSH3:
                MOV     AL,[RESULT+2+BX]
                XOR     AH,AH
                PUSH    AX
                INC     BX
                LOOP    LOOP_PUSH3
        MOV     CL,[RESULT+1]
        XOR     CH,CH
        MOV     BX,0
        LOOP_POP3:
                POP     AX
                MOV     [RESULT+2+BX],AL
                INC     BX
                LOOP    LOOP_POP3
        RET
REVERSE3    ENDP

; 前移消去IDDA1符号位
FORWARD1     PROC
        MOV     AL,[IDDA1+2]
        CMP     AL,45
        JNE     ENFORWARD1
        ; 前移消去符号位
        MOV     IDDA1FLAG,45
        MOV     BX,1
        LOOP_FORWARD1:
                MOV     AL,[IDDA1+2+BX]
                MOV     [IDDA1+1+BX],AL
                INC     BX
                CMP     BL,[IDDA1+1]
                JNE     LOOP_FORWARD1
        SUB     [IDDA1+1],1
        ENFORWARD1:
        RET
FORWARD1     ENDP

; 前移消去IDDA2符号位

FORWARD2    PROC
```

```
        MOV     AL,[IDDA2+2]
        CMP     AL,45
        JNE     ENFORWARD2
        MOV     IDDA2FLAG,45
        MOV     BX,1
        LOOP_FORWARD2:
                MOV     AL,[IDDA2+2+BX]
                MOV     [IDDA2+1+BX],AL
                INC     BX
                CMP     BL,[IDDA2+1]
                JNE     LOOP_FORWARD2
        SUB     [IDDA2+1],1
        ENFORWARD2:
        RET
FORWARD2    ENDP

; 输入数据串1
MYIN1   PROC
        MOV     AH,0AH
        LEA     DX,IDDA1
        INT     21H
        CALL    SPACE
        CALL    FORWARD1
        RET
MYIN1   ENDP

; 输入数据串2
MYIN2   PROC
        MOV     AH,0AH
        LEA     DX,IDDA2
        INT     21H
        CALL    SPACE
        CALL    FORWARD2
        RET
MYIN2   ENDP

; 测试REVERSE3, PRINT3, 以及必要的检验
TESTRESULT  PROC
        ; 先保证之前调用过INIT,清空了RESULT
        MOV     [RESULT+1],20
        CALL    PRINT3
        CALL    REVERSE3
        CALL    PRINT3
        RET
TESTRESULT  ENDP

CODE    ENDS
        END     MAIN
```

- C库函数

```
; C库函数的实现 : strcmp
; Author : GMFTBY
; Time   : 2017.12.2

; 数据段定义
DATA    SEGMENT
BUFFER1 DB      205
        DB      ?
        DB      205 DUP(0)
BUFFER2 DB      205
        DB      ?
        DB      205 DUP(0)
MESSAGE_NO      DB      'NOT MATCH','$'
MESSAGE_YES     DB      'MATCH','$'
CRLF    DB      0DH,0AH,'$'
DATA    ENDS

; 代码段定义
CODE    SEGMENT
MAIN    PROC    FAR
        ASSUME  CS:CODE,DS:DATA,ES:NOTHING
        PUSH    DS
        XOR     AX,AX
        PUSH    AX
        MOV     AX,DATA
        MOV     DS,AX
        ; 输入字符串
        LEA     DX,BUFFER1
        CALL    MY_SCANF
        LEA     DX,BUFFER2
        CALL    MY_SCANF
        CALL    MY_STRCPY
        RET
MAIN    ENDP

; 输入函数
MY_SCANF        PROC
        MOV     AH,0AH
        INT     21H
        ; 输出换行和回车
        MOV     AH,9
        LEA     DX,CRLF
        INT     21H
        RET
MY_SCANF        ENDP

; 实现的C库函数 : strcmp
MY_STRCPY       PROC
        MOV     AL,[BUFFER1+1]
        MOV     AH,[BUFFER2+1]
        CMP     AL,AH
        JNE     END_OF_CALL
        MOV     DL,[BUFFER1+1]
        XOR     DH,DH
        MOV     BX,0
        LOOP_FOR:
                MOV     AL,[BUFFER1+2+BX]
                MOV     AH,[BUFFER2+2+BX]

                CMP     AL,AH
```

```
                JNE     END_OF_CALL
                INC     BX
                CMP     BX,DX
                JNE     LOOP_FOR
        JMP     END_OF_YES
END_OF_CALL:
        MOV     AH,9
        LEA     DX,MESSAGE_NO
        INT     21H
        RET
END_OF_YES:
        MOV     AH,9
        LEA     DX,MESSAGE_YES
        INT     21H
        RET
MY_STRCPY       ENDP

PRINTAX     PROC
        MOV     BX,100
        OR      AX,AX
        JZ      _0_
LOOP_P:
        XOR     DX,DX
        DIV     BX
        MOV     CX,AX
        OR      CX,DX
        JZ      _E_
        PUSH    DX
        CALL    LOOP_P
        POP     DX
        ADD     DL,'0'
        JMP     _1_
_0_:
        MOV     DL,'0'
_1_:
        MOV     AH,2
        INT     21H
_E_:
        RET
PRINTAX     ENDP

CODE    ENDS
        END     MAIN
```

- 多重循环

```
; 编写多重循环(2)

; Author : GMFTBY
; Time   : 2017.12.1

; 数据段
DATA    SEGMENT
DATA    ENDS

; 代码段
CODE    SEGMENT
MAIN    PROC    FAR
        ASSUME  CS:CODE,DS:DATA,ES:NOTHING
        PUSH    DS
        XOR     AX,AX
        PUSH    AX
        MOV     AX,DATA
        MOV     DS,AX

        ; 5多重循环
        MOV     AX,0
        LOOP_OUT:
                ADD     AX,1
                MOV     BX,0
                LOOP_IN:
                        ADD     BX,1
                        CMP     BX,10
                        JNE     LOOP_IN
                CMP     AX,10
                JNE     LOOP_OUT
RET
MAIN    ENDP

; 测试子程序(出错的部分，有时候会出现bug，bug原因不明)
PRINTAX     PROC
MOV     BX,10
OR      AX,AX
JZ      _0_
LOOP_P:
XOR     DX,DX
DIV     BX
MOV     CX,AX
OR      CX,DX
JZ      _E_
PUSH    DX
CALL    LOOP_P
POP     DX
ADD     DL,'0'
JMP     _1_
_0_:
MOV     DL,'0'
_1_:
MOV     AH,2
INT     21H
_E_:
RET
PRINTAX  ENDP


CODE    ENDS
```

```
        END     MAIN
```

- 计算器

```
.386
.model flat, stdcall
option casemap :none

include kernel32.inc
include fpu.inc

includelib msvcrt.lib
includelib kernel32.lib
includelib fpu.lib

scanf   PROTO C :ptr sbyte, :vararg
printf  PROTO C :ptr sbyte, :vararg

; 数据段定义
.data

inputString byte    "输入 : ", 0
string      byte    50 dup(?)
num1        tbyte   ?
num2        tbyte   ?
result      byte    50 dup(0)

inFmt   byte    "%s", 0
outFmt  byte    "%s", 0ah, 0

; 代码段定义，进行相关的计算
.code
start:
    invoke  printf, offset inputString
    invoke  scanf, offset inFmt, offset string

    mov esi, offset string
    inc esi
l1:
    mov al, [esi]
    cmp al, '+'
    je  l2
    cmp al, '-'
    je  l2
    cmp al, '*'
    je  l2
    cmp al, '/'
    je  l2
    inc esi
    jmp l1
l2:
    mov bl, [esi]
    xor al, al
    mov [esi], al
    inc esi

    invoke FpuAtoFL, offset string, offset num1, DEST_MEM
    invoke FpuAtoFL, esi, offset num2, DEST_MEM

    cmp bl, '+'
    je  Addition
    cmp bl, '-'

    je Subtraction
```

```
    cmp bl, '*'
    je Multiplication
    cmp bl, '/'
    je Division

Addition:
    invoke FpuAdd, offset num1, offset num2, 0, SRC1_REAL or SRC2_REAL or DEST_FPU
    jmp output
Subtraction:
    invoke FpuSub, offset num1, offset num2, 0, SRC1_REAL or SRC2_REAL or DEST_FPU
    jmp output
Multiplication:
    invoke FpuMul, offset num1, offset num2, 0, SRC1_REAL or SRC2_REAL or DEST_FPU
    jmp output
Division:
    invoke FpuDiv, offset num1, offset num2, 0, SRC1_REAL or SRC2_REAL or DEST_FPU

output:
    invoke FpuFLtoA, 0, 7h, offset result, SRC1_FPU or SRC2_DIMM
    invoke printf, offset outFmt, offset result
    invoke ExitProcess, 0
end start
```

- 随机数

```
; 使用线性迭代的思路实现随机数的生成,随机数范围 [0, 29],种子[0, 10)
; X_{n+1} = (a * X_n + b) mod c
; a, b, c 均为质数，c的目的是为了将结果更好的散射
; a = 17, b = 23, c = 97

; Author : GMFTBY
; Time   : 2017.12.1

; 数据段定义
DATA    SEGMENT
DATA    ENDS

; 代码段定义
CODE    SEGMENT
MAIN    PROC    FAR
    ASSUME  CS:CODE,DS:DATA,ES:NOTHING
    PUSH    DS
    XOR     AX,AX
    PUSH    AX
    MOV     AX,DATA
    MOV     DS,AX

    ; 输入的种子在AL中
    MOV     AH,0
    INT     16H
    SUB     AL,48   ; 转换得到数字
    XOR     AH,AH

    MOV     CX,1000
    LOOP_FOR:
            MOV     BL,17
            MUL     BL      ; AL * 17
            XOR     AH,AH
            ADD     AL,23
            XOR     AH,AH
            MOV     BL,29
            DIV     BL
            MOV     AL,AH
            LOOP    LOOP_FOR
    XOR     AH,AH
    CALL    PRINTAX
    RET
MAIN    ENDP

; 输出AX的数值
PRINTAX     PROC
        MOV     BX,10
        OR      AX,AX    ; ax won't be changed , but if the ax's content is 0, jump to
_0_
        JZ      _0_
LOOP_P:
        XOR     DX,DX
        DIV     BX
        MOV     CX,AX
        OR      CX,DX
        JZ      _E_
        PUSH    DX      ; push the remaining number nto the stack
        CALL    LOOP_P

        POP     DX
```

```
        ADD     DL,'0'  ; change the ASCII into the number
        JMP     _1_
_0_:
        MOV     DL,'0'
_1_:
        MOV     AH,2    ; print one character to the terminal
        INT     21H
_E_:
        RET
PRINTAX     ENDP

CODE    ENDS
        END     MAIN
```

- 文件内容对比

```
; 文件读写比较
; Author : GMFTBY
; Time   : 2017.12.6

DATA SEGMENT
PATHNM1     DB  'C:/TEST1.TXT'
BUFFER1     DB  100 DUP (0)
BAK1        DB  100 DUP (0)

PATHNM2     DB  'C:/TEST2.TXT'
BUFFER2     DB  100 DUP (0)
BAK2        DB  100 DUP (0)

MESSAGE_NO  DB  'NO MATCH ','$'
MESSAGE_YES DB  'MATCH','$'

LINENUMBER  DB  ?
DATA ENDS

CODE SEGMENT
MAIN    PROC    FAR
    ASSUME  CS:CODE,DS:DATA,ES:NOTHING
    PUSH    DS
    XOR     AX,AX
    PUSH    AX
    MOV     AX,DATA
    MOV     DS,AX

    ; 打开文件1TEST.TXT
    LEA     DX,PATHNM1
    CALL    OPEN_FILE

    ; 读取文件内容，存入BUFFER1
    LEA     DX,BUFFER1
    CALL    READ_FILE

    CALL    COPY_STRING1
    ; CALL    CLOSE_FILE

    LEA     SI,BAK1
    CALL    PRINT_STRING
    CALL    CLOSE_FILE

    LEA     DX,PATHNM2
    CALL    OPEN_FILE

    LEA     DX,BUFFER2
    CALL    READ_FILE

    CALL    COPY_STRING2
    CALL    CLOSE_FILE

    LEA     SI,BAK2
    CALL    PRINT_STRING

    ; 文件内容比对
    CALL    LOOK
    RET

MAIN   ENDP
```

```
LOOK    PROC
    MOV     LINENUMBER,1
    MOV     BX,0
    LOOP_LOOK:
            MOV     AL,[BAK1+BX]
            MOV     AH,[BAK2+BX]
            CMP     AL,AH
            JNE     END_OF_NO
            INC     BX
            CMP     AL,0AH
            JE      ADD_LINE
            FOR_NEXT:
                    CMP     BX,100
                    JNE     LOOP_LOOK
    JMP     END_OF_YES
    END_OF_NO:
            MOV     AH,9
            LEA     DX,MESSAGE_NO
            INT     21H
            MOV     AL,LINENUMBER
            XOR     AH,AH
            CALL    PRINTAX
            RET
    ADD_LINE:
            INC     LINENUMBER
            JMP     FOR_NEXT
    END_OF_YES:
            MOV     AH,9
            LEA     DX,MESSAGE_YES
            INT     21H
            RET
LOOK    ENDP

; 拷贝字符串1
COPY_STRING1    PROC
    PUSH    BX
    PUSH    AX
    PUSH    CX      ; CX必须保护好
    MOV     BX,0
    LOOP_COPY:
            MOV     AL,[BUFFER1+BX]
            MOV     [BAK1+BX],AL
            INC     BX
            CMP     BX,100
            JNE     LOOP_COPY
    POP     CX
    POP     AX
    POP     BX
    RET
COPY_STRING1    ENDP

COPY_STRING2    PROC
    PUSH    BX
    PUSH    AX
    PUSH    CX
    MOV     BX,0
    LOOP_COPY2:
        MOV     AL,[BUFFER2+BX]

        MOV     [BAK2+BX],AL
```

```
        INC     BX
        CMP     BX,100
        JNE     LOOP_COPY2
        POP     CX
        POP     AX
        POP     BX
        RET
COPY_STRING2    ENDP

; 打开文件
OPEN_FILE   PROC
   MOV     AH,3DH
   MOV     AL,2
   INT     21H
   RET
OPEN_FILE   ENDP

; 读取文件
READ_FILE   PROC
   MOV     BX,AX
   MOV     CX,100
   MOV     AH,3FH
   INT     21H
   RET
READ_FILE   ENDP

; 关闭文件
CLOSE_FILE  PROC
   MOV     AH,3EH
   INT     21H
   RET
CLOSE_FILE  ENDP

; 输出字符串检验
PRINT_STRING    PROC
   PUSH    CX
   PUSH    AX
   PUSH    BX
   PUSH    DX
   MOV     CX,100
   MOV     BX,0
   NEXT:
           MOV     DL,[SI+BX]
           MOV     AH,2
           INT     21H
           INC     BX
           CMP     BX,CX
           JNE     NEXT
   POP     DX
   POP     BX
   POP     AX
   POP     CX
   RET
PRINT_STRING    ENDP

PRINTAX     PROC
       MOV     BX,10
       OR      AX,AX
       JZ      _0_

          LOOP_P:
```

```
                XOR     DX,DX
        DIV     BX
        MOV     CX,AX
        OR      CX,DX
        JZ      _E_
        PUSH    DX
        CALL    LOOP_P
        POP     DX
        ADD     DL,'0'
        JMP     _1_
_0_:
            MOV     DL,'0'
_1_:
                    MOV     AH,2
        INT     21H
_E_:
         RET
PRINTAX     ENDP
CODE        ENDS
            END     MAIN
```

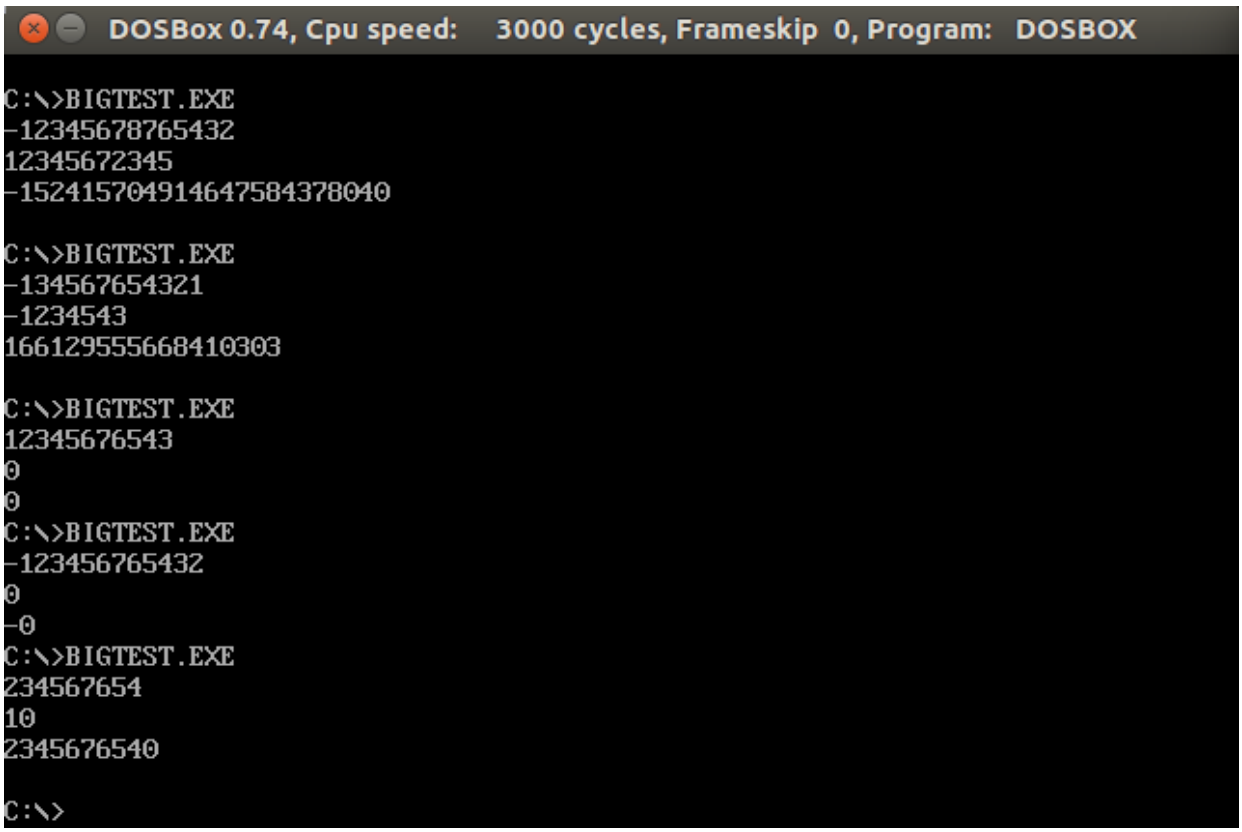
- 数据块复制

```
; 数据块复制(内存->内存)
; Author : GMFTBY
; Time   : 2017.12.6

DATA    SEGMENT
BUFFER1     DB  100
            DB  ?
            DB  100 DUP(?)

BUFFER2     DB  100
            DB  ?
            DB  100 DUP(?)
CRLF        DB  0DH,0AH,'$'
DATA    ENDS

CODE    SEGMENT
MAIN    PROC    FAR
        ASSUME  CS:CODE,DS:DATA,ES:NOTHING
        PUSH    DS
        XOR     AX,AX
        PUSH    AX
        MOV     AX,DATA
        MOV     DS,AX

        ; 输入数据段
        LEA     DX,BUFFER1
        CALL    INPUT_BUFFER

        ; 数据块赋值
        CALL    COPY

        ; 输出换行
        MOV     AH,9
        LEA     DX,CRLF
        INT     21H

        ; 数据块输出
        CALL    OUTPUT_STRING
        RET
MAIN    ENDP

; 输入数据段
INPUT_BUFFER    PROC
        MOV     AH,0AH
        INT     21H
        RET
INPUT_BUFFER    ENDP

; 数据块赋值
COPY    PROC
        MOV     AL,[BUFFER1+1]
        MOV     [BUFFER2+1],AL
        MOV     CL,AL
        XOR     CH,CH
        MOV     BX,0
        LOOP_COPY:
                MOV     AL,[BUFFER1+2+BX]
                MOV     [BUFFER2+2+BX],AL

                INC     BX
```

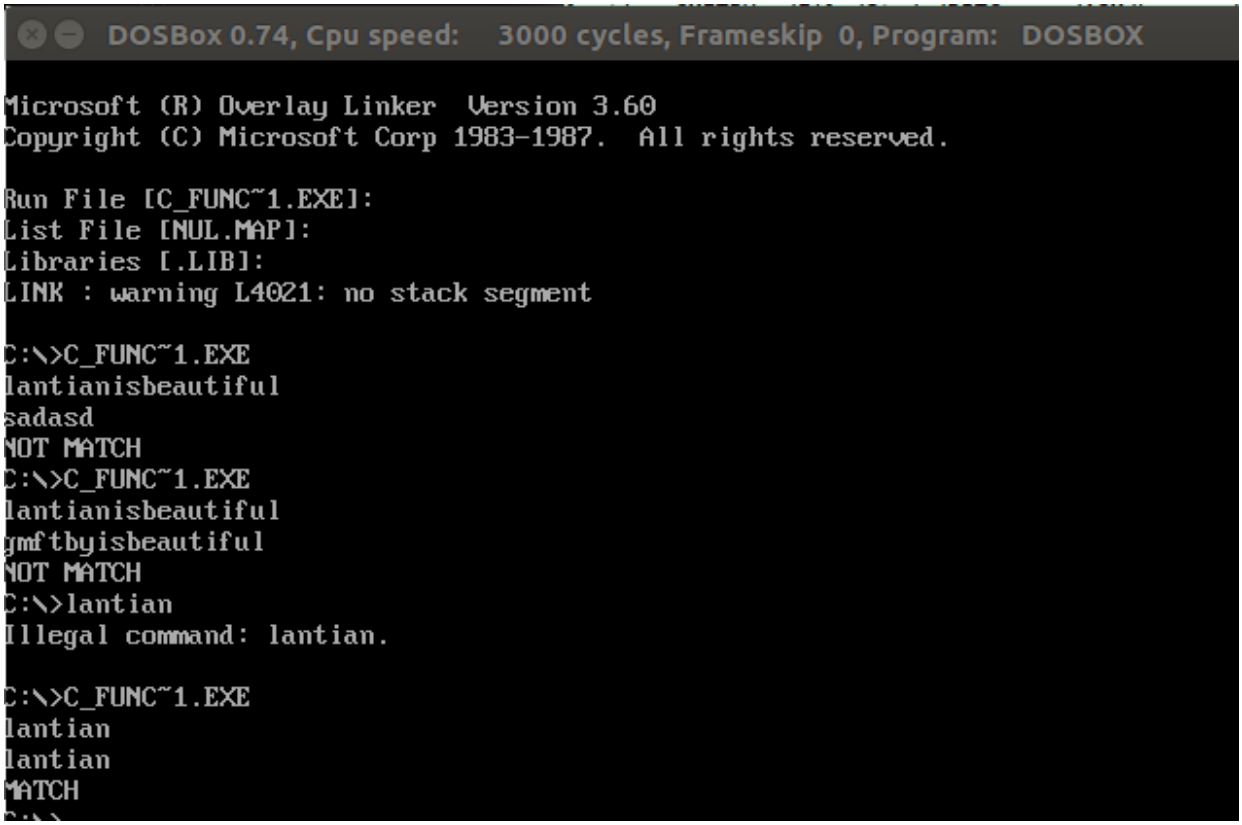
```
                CMP     BX,CX
                JNE     LOOP_COPY
        RET
COPY    ENDP

; 数据块输出
OUTPUT_STRING   PROC
        MOV     CL,[BUFFER2+1]
        XOR     CH,CH
        MOV     BX,0
        LOOP_OUT:
                MOV     AH,2
                MOV     DL,[BUFFER2+2+BX]
                INT     21H
                INC     BX
                CMP     BX,CX
                JNE     LOOP_OUT
        RET
OUTPUT_STRING   ENDP

CODE    ENDS
        END     MAIN
```

5. 实验效果

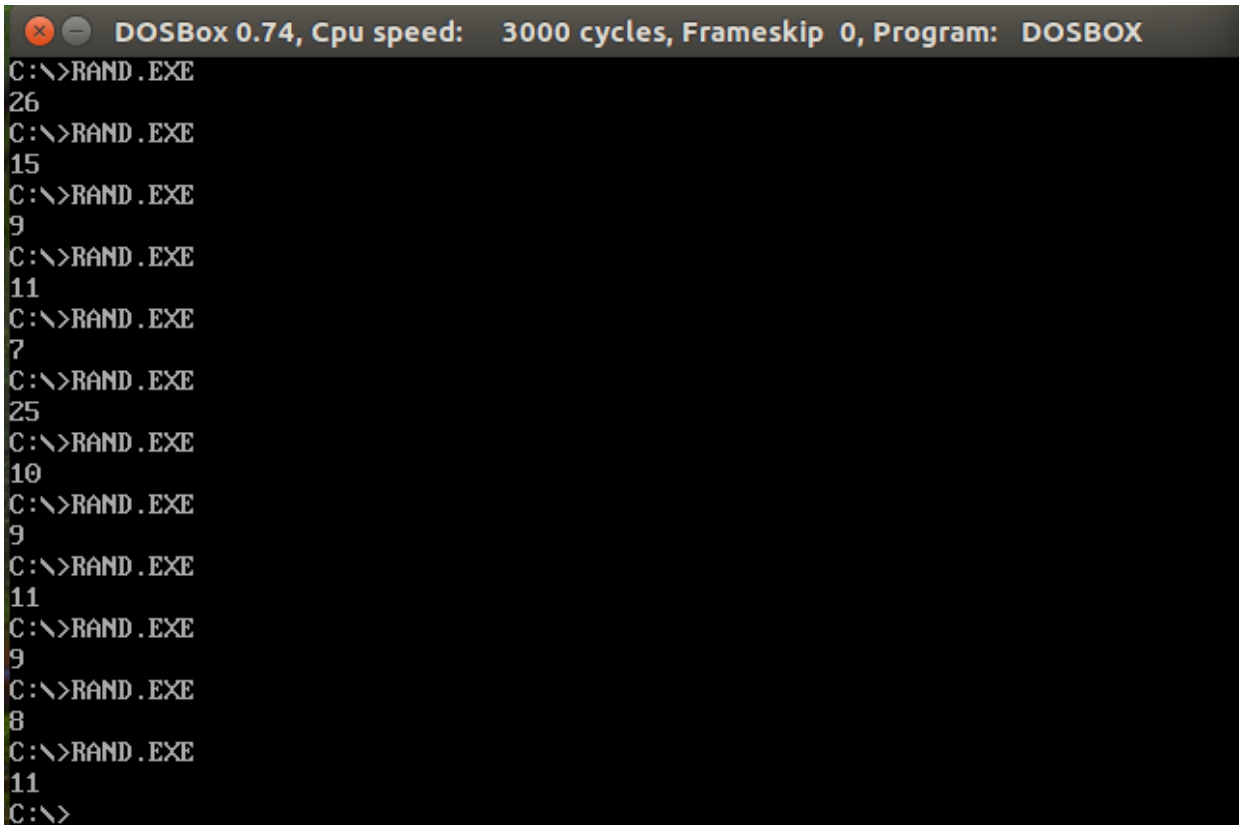
1. 大数相乘



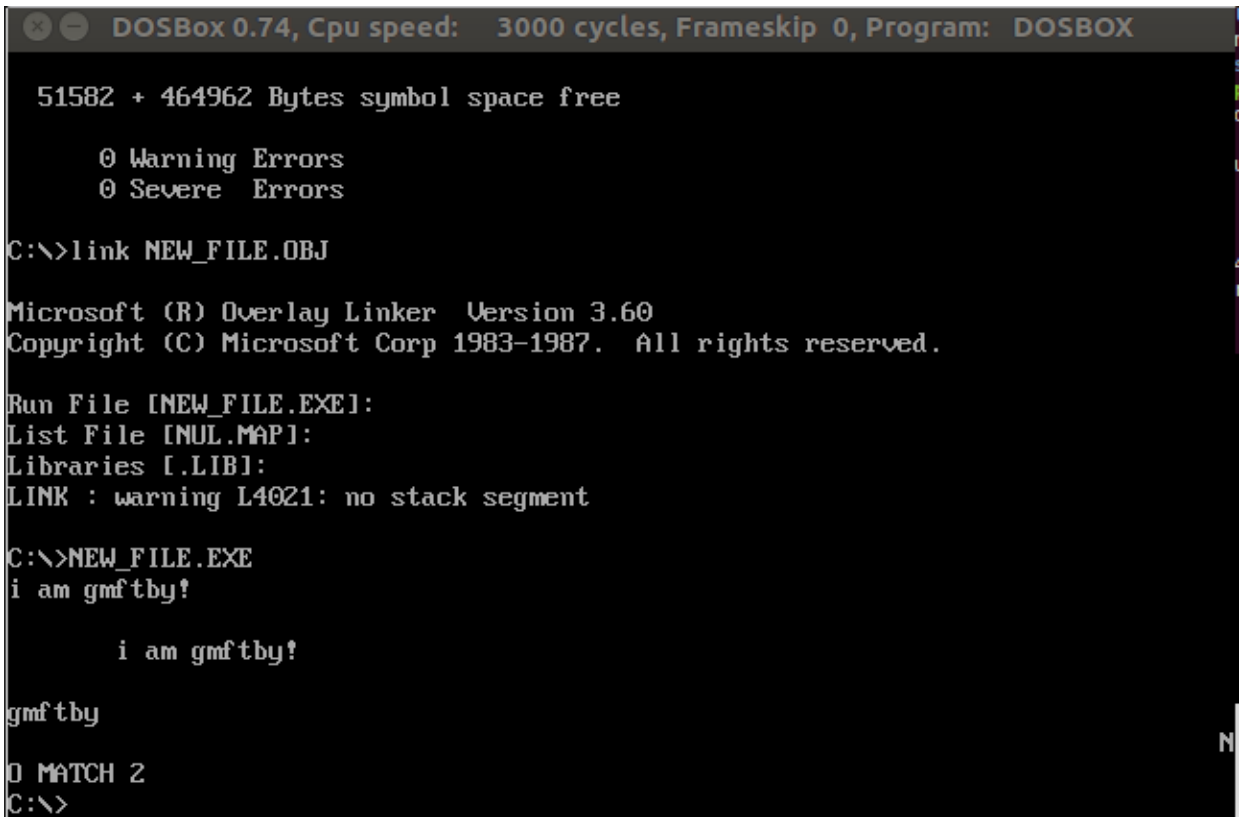
2. C语言库函数



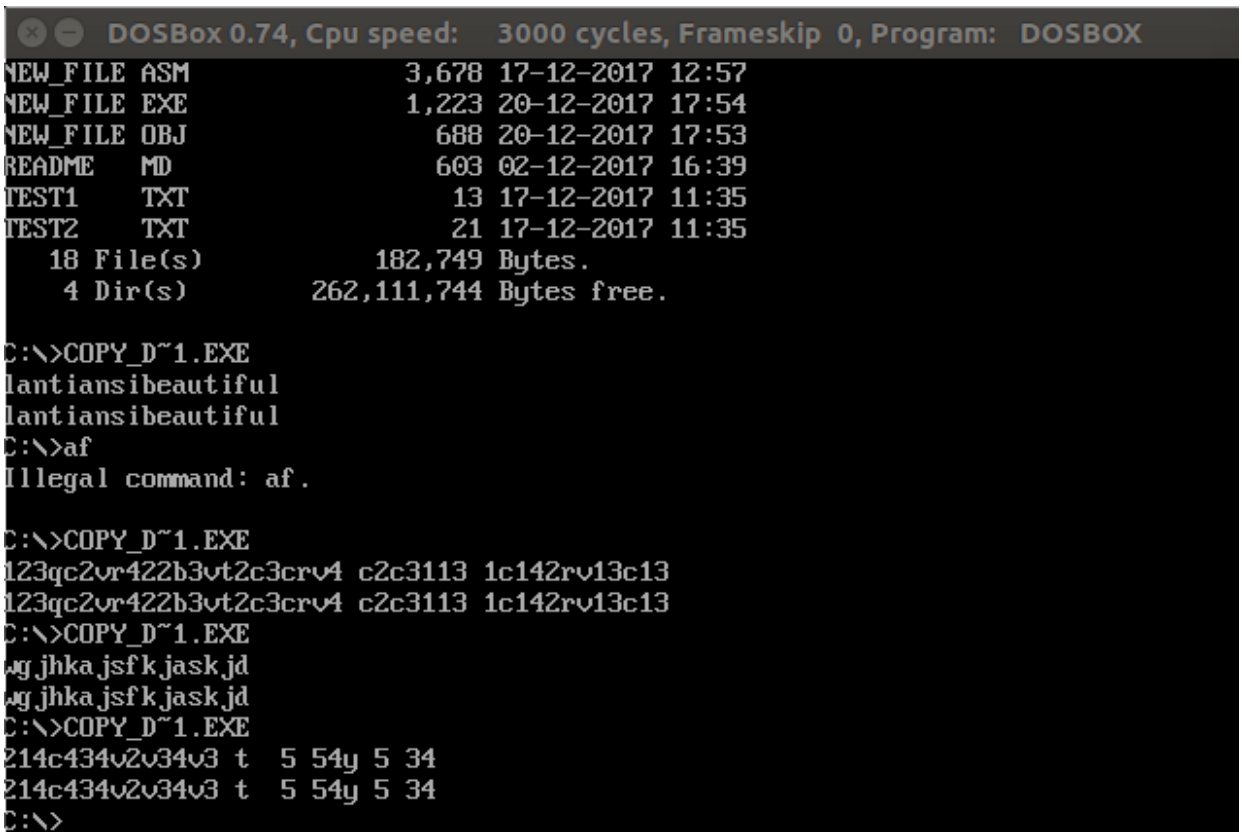
3. C多重循环
4. 计算器
5. 随机数



6. 文本比较



7. 数据块复制



6. 心得体会

1. 非常感谢老师和学长学姐的照顾
    1. 老师对我平常上课的一些疑问都进行了耐心的指导和讲解
    2. 对于系统平台的问题，老师对我在Linux平台下的实践照顾了很多，非常的感谢老师
    3. 在验收的时候也非常感谢老师和学长学姐对我的认可和鼓励，非常感谢

2. 因为我本来是想学NASM的汇编，但是学校的教材是MASM的汇编，我上网查了一些资料，对于不同的汇编语言助记符的学习中找到了一些认识
    1. nasm / masm说起来都是差不多的东西，汇编语言这种东西其实和硬件平台息息相关，没有必要纠结，何况nasm下也不一定会学到太多
    2. 我们需要做的是，熟悉和了解汇编的格式和书写的范例和思路，熟悉CPU的各种的操作加深了解计算机的处理数据的过程
    3. 一种汇编助记符和容易翻译成另一个助记符，但是核心的思路是针对于CPU的，所以没有必要过分的纠结于使用哪一种的具体的汇编助记符
3. 在学习之前，对于汇编的感觉是有一些恐惧的，但是在一次一次的实践中，加深了对寄存器和内存的控制，现在我反而觉得能够直接操纵CPU和内存中的一些直接的细节部分来编程也是一种享受
4. 对于调试和纠错方面
    1. 因为我的环境的特殊性，对于输入输出和调试的所有功能方面我都只能手写**BIOS**中断来进行实现，对我来说并不是很轻松，但是也加深了我对汇编语言的实习程度，我认为老师在以后的教学中可以让同学放弃集成开发环境的便利，编写最底层的**8086**代码，我认为是可以非常有效的训练学生对于汇编语言的理解程度的
    2. 调试纠错的过程里我认为看懂报错信息是非常重要的一件事情，老师可以让同学们自发的总结一些常见的报错错误并提交可以锻炼学生的调试和纠错的能力

学生 : 兰天

日期 : 2017.12.20