2017.3.17

目录

汇 编 语 言 程 序 设 计 实 验 报 告

1 实验目的与要求

本次实验的主要目的与要求有下面 6 点，所有的任务都会围绕这 6 点进行，希望大家事后检查自己是否达到这些目的与要求。

1. 掌握子程序设计的方法与技巧，熟悉子程序的参数传递方法和调用原理；
2. 掌握宏指令、模块化程序的设计方法;
3. 掌握较大规模程序的合作开发与调试方法；
4. 掌握汇编语言程序与 C 语言程序混合编程的方法；
5. 熟悉 C 编译器的基本优化方法;
6. 了解 C 语言编译器的命名方法，主、子程序之间参数传递的机制

2 实验内容

任务 1：宏与子程序设计

要求：

(1) 程序执行时首先显示一个功能菜单:选择 1=录入学生姓名和各科考试成绩，2=计算平均分，3=计算排名，4=输出成绩单，5=程序退出。

(2) 2 人一组，一人负责包括菜单显示、程序退出在内的主程序，以及菜单中的功能 1 和 2；另一人负责菜单中的功能 3 和 4。各自汇编自己的模块，然后连接生成一个程序。

(3) 录入学生姓名和各科考试成绩时，首先显示录入的是第几个学生的信息，然后分别在提示之后输入姓名和各科成绩（可以借鉴书上十进制转二进制的子程序 F10T2）。所有学生信息录入完毕后回到菜单显示的位置。姓名及考试成绩的存放、平均分的计算，按照实验二的要求。

(4) 排名的基本要求是按照平均成绩从高到低计算名次，也可以考虑按照指定课程的成绩排名。相同平均分时排名相同，下一个相邻平均分的排名应该是排名在前的所有人数和的下一个数值。输出成绩单的基本要求是依次显示每个学生的姓名、平均成绩和排名，也可以考虑按照指定课程、指定进制的形式显示（可以借鉴书上二进制转十进制的子程序 F2T10）。

任务 2：在 C 语言程序中调用 汇编语言实现的函数

要求：

对于任务 1 的程序进行改造，主控程序、以及输入输出等功能用 C 语言实现，其他功能用独立的汇编语言子程序的方式实现； 在 C 语言程序中调用汇编语言子程序实验过程

3 实验过程

3.1 任务 1

3.1.1 设计思想及寄存器分配

1. 总体设计要求

这次实验要求我们进行分工合作，我和我的队友：李培昊，进行了一些简要的分工。首先按照实验报告的要求我们将前两个功能交给李培昊处理，后两个功能交给我来处理，最后整体进行链接。

为了实现链接时的方便，我们对每个功能做了一些规定：每个功能都写成一个文件，然后还有一个主菜单的文件。所有文件共享菜单文件的函数，每次执行后返回到主菜单。每个分文件各自独立的数据段设置这个函数要输出的信息以及一些必要的数据。

最后实现的总方针应该如此：首先执行主函数，然后进行选择，最开始进行录入，然后录入之后返回主菜单，继续选择其他功能：计算排名必须在计算平均成绩之前，输出成绩必须在计算排名之前。按照这样的规则对此系统进行使用。

下面将叙述我负责的两个功能。

2. 功能三设计思想

功能三的任务是让我们实现一个计算排名的程序，即我们需要得到有多少个人比这个人的分数高。我决定使用两次循环，第一个循环作为当前遍历的人，即要计算排名的人，第二个循环即为遍历全部的人，计算有多少人成绩比他高，第二个循环结束之后就把第一个循环的目标人的排名写入内存。这样一个嵌套循环解决问题。

3. 功能三寄存器分配

循环次数控制：CX。初始化为 N，作为第一个循环的循环次数。在第二个循环开始前将此时的 CX 入栈，然后再初始化为 N 作为第二个循环的次数。第二个循环结束后将其 POP。

第一个循环同学指示器：BX + SI，初始化 BX + SI 为第一个学生的平均成绩的便宜地址，然后每次第一个循环结束之后就将他指向下一个同学的平均成绩。（通过改变 SI 的值）。

第二个循环同学指示器：BX + DI，BX 和上面一样都是学生段的偏移地址，通过控制 DI 的大小来控制遍历的是第几个同学，第二个循环每循环一次便改变 DI，第二个循环结束时再让他初始化。

排名指示器：DH，记录了排名信息，每次初始化为 1，遇到第二个循环大于第一个循环的就加一，第二层循环结束后将它写到内存。

4. 功能四设计思想

功能四要求我们设计一个程序输出所有学生的成绩单。我觉得我首先要设计出输出的模式，然后每次鲟鳇将这个模式中的一些信息进行替换，然后输出，最后循环结束也输出了全部的信息。

但是如果学生数目太多怎么办？我决定采用翻页的方法来设计这一点。每页只显示五个同学的信息，然后按下 AD 键进行翻页。这并不难实现。

遇到一个问题就是成绩是一个整数，而我们要输出的是一个字符串，所以要将这个整数变成一个个的字符。变化的过程就是除以 10 取模，然后写回到指定位置。这样得话我们就需要对成绩为 100 的同学进行单独处理。

5. 功能四寄存器分配

CX ：作为循环控制变量，初始化为 N，每次循环输出一个人的信息。

CL：存储 10，作为要做除法的中专寄存器

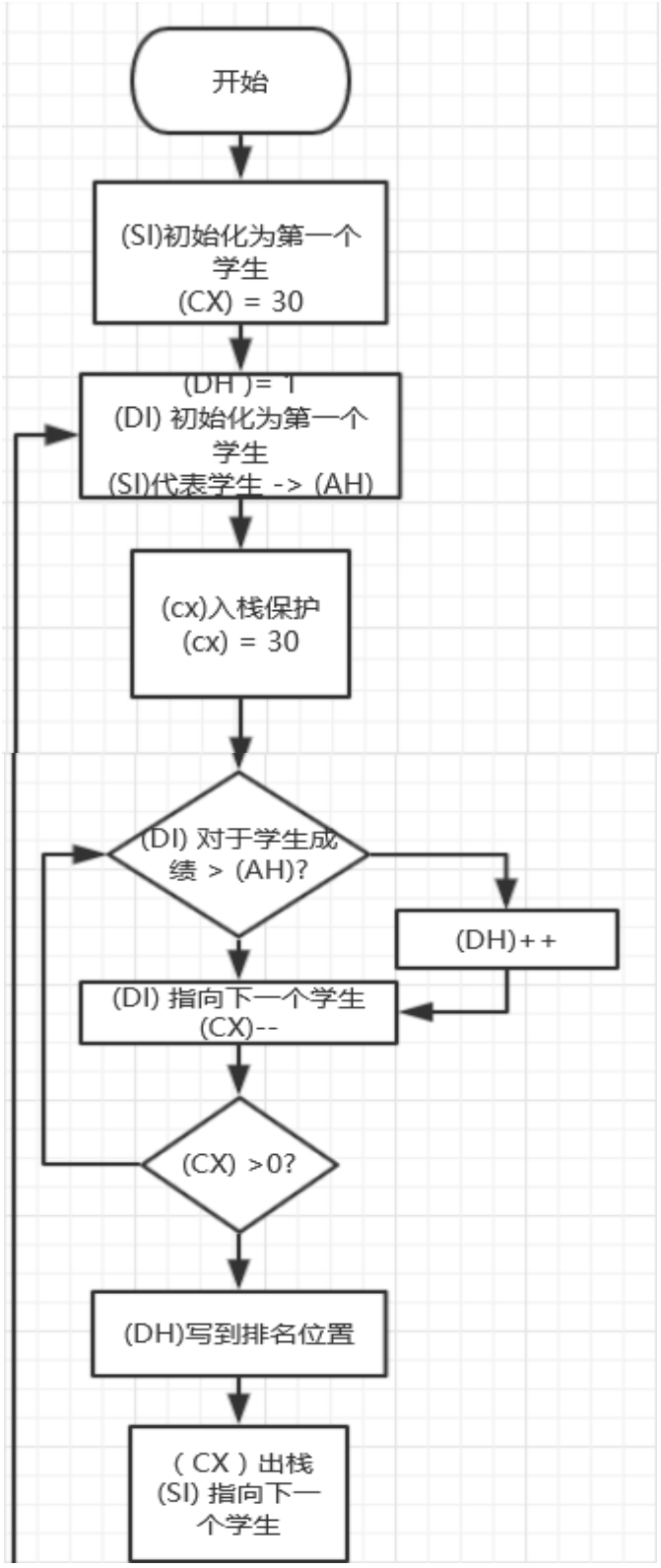
BX、BP ：作为 GRADE 和 STU 的段首地址，与 SI 和 DI 配合进行寻址。

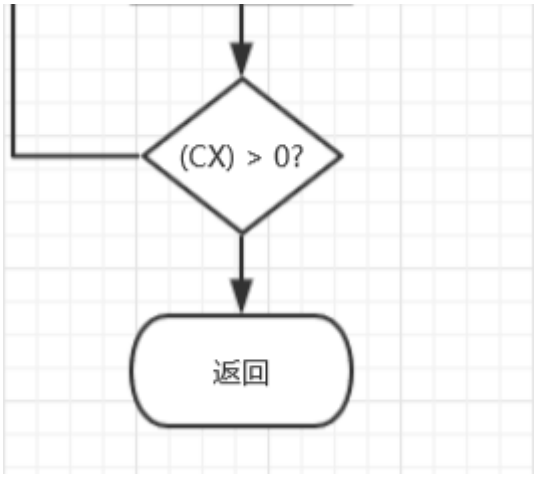
SI: 存储 GRADE 段的偏移地址，每次循环时候控制它的值将其放入指定的位置

DI: 存储 STU 段的偏移地址，每次循环的时候控制他的大小遍历指定的学生。

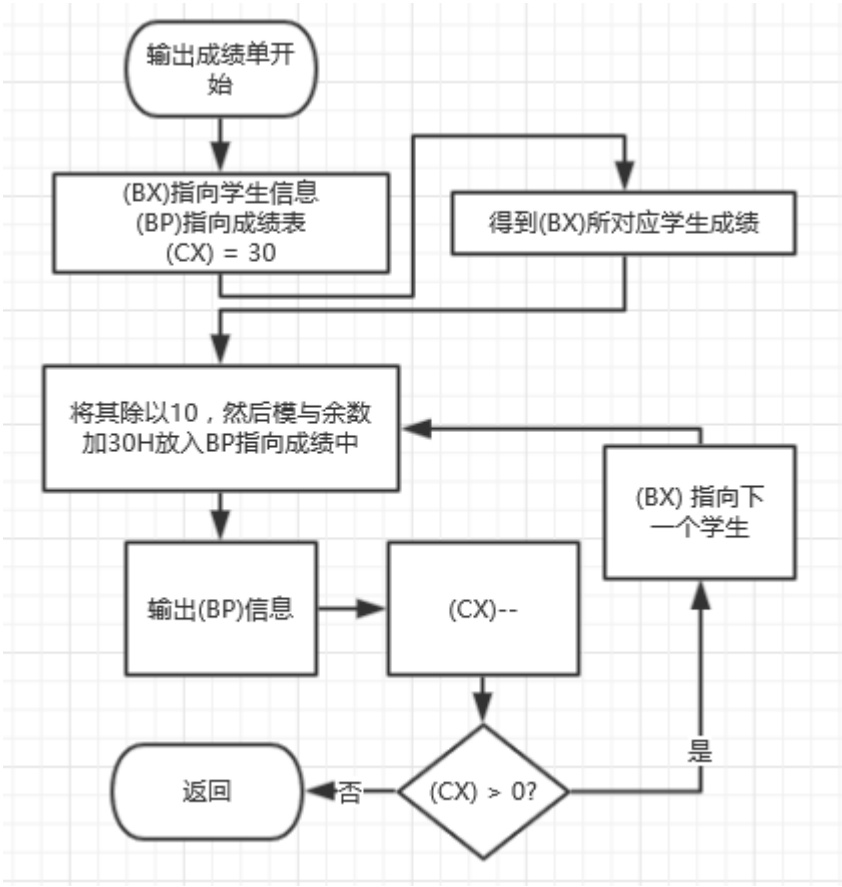
AX: 做除法时候存储被除数和结果。

3.1.2 流程图


开始
(SI)初始化为第一个学生
(CX) = 30
(DH)= 1
(DI) 初始化为第一个学生
(SI)代表学生 -> (AH)
(cx)入栈保护
(cx) = 30
(DI) 对于学生成绩 > (AH)?
(DH)++
(DI) 指向下一个学生
(CX)--
(CX) >0?
(DH)写到排名位置
(CX) 出栈
(SI) 指向下一个学生



输出成绩单：



3.1.3 源程序

```
.386
STACK    SEGMENT    USE16    STACK
         DB    256    DUP(0)
STACK    ENDS

DATA     SEGMENT USE16
```

```
N EQU 30
STU DB  'TEMPVALUE',0,90,100,95,80,?,?
      DB    'xinjie',0,0,0,0,90,90,8,90,?,?
      DB  3    DUP('TEMPVALUE',0,90,100,95,80,?,?)
      DB    'xinjie',0,0,0,0,90,90,8,90,?,?
      DB    'xinjie',0,0,0,0,90,90,8,90,?,?
      DB    'xinjie',0,0,0,0,90,90,8,90,?,?
      DB    'xinjie',0,0,0,0,90,90,8,90,?,?
      DB    'xinjie',0,0,0,0,90,90,8,90,?,?
      DB  N-10    DUP('TEMPVALUE',0,90,100,95,80,?,?)

MODE    DB '+ Student     Chinese    Math    English    Aveg      Rank    +',0ah,0dh,'$'
GRADE DB '+                                                                +',0ah,0dh,'$'
xiantiao DB '-------------------------------------------------',0ah,0dh,'$'
fanye db 'Please input A or D to change page A is Left D is Right ! ',0ah,0dh,'$'
DATA     ENDS

CODE     SEGMENT USE16
          ASSUME CS: CODE, DS: DATA, SS: STACK
START:
          MOV AX,DATA
          MOV DS,AX
          CALL RANK
          CALL GRADE_CODE
          mov ah, 4ch
          int 21h

RANK PROC NEAR
            PUSH AX
            PUSH BX
            PUSH CX
            PUSH DX
            PUSH SI
            PUSH DI

          LEA BX,STU
          MOV SI,13
          MOV CX,N
      FIRST_loop:
                MOV DH,1
                MOV DI,13
```

```
            MOV AH,DS:[BX + SI]
            PUSH CX
            MOV CX,N
    SECOND_loop:
            PUSH AX
            SUB AH, BYTE PTR DS:[DI + BX]
            JS _ONE_
    _TWO_:
            POP AX
            ADD DI,16
            LOOP SECOND_loop
            MOV [BX + SI + 1],DH
            ADD SI,16
            POP CX
            LOOP FIRST_loop

              POP DI
              POP SI
              POP DX
              POP CX
              POP BX
              POP AX
            RET
    _ONE_:
            INC DH
            JMP _TWO_
RANK ENDP

GRADE_CODE PROC NEAR
        PUSH EAX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH DI
        PUSH BP
        mov dx, offset xiantiao
        mov ah,9
        int 21h
        mov DX, OFFSET MODE
        mov AH, 9H
        int 21h
```

```
        MOV CX,4
        LEA BX,GRADE
        LEA BP,STU
        MOV DI,10

OUT_GRADE:
        PUSH CX
        MOV CL,10

        MOV EAX,DS:[BP + DI -10]
        MOV [BX + 2], EAX
        MOV EAX,DS:[BP + DI -6]
        MOV [BX + 6], EAX
        MOV AX,DS:[BP + DI -2]
        MOV [BX + 10], AX
Chinese:
        MOV AX, DS:[BP + DI]
        MOV AH, 0
        DIV CL
        ADD AH, 30H
        ADD AL, 30H
        MOV [BX + 17], AL
        MOV [BX + 18], AH
        CMP AX, 303Ah
        JZ _CH_M_
        CMP AL, 30H
        JZ _CO_M_

Math:
        MOV AX, DS:[BP + DI + 1]
        MOV AH, 0
        DIV CL
        ADD AH, 30H
        ADD AL, 30H
        MOV [BX + 23], AL
        MOV [BX + 24], AH
        CMP AX, 303Ah
        JZ _CH_E_
        CMP AL, 30H
        JZ _CO_E_
```

English:

```
MOV AX, DS:[BP + DI + 2]
MOV AH, 0
DIV CL
ADD AH, 30H
ADD AL, 30H
MOV [BX + 32], AL
MOV [BX + 33], AH
CMP AX, 303Ah
JZ _CH_A_
CMP AL, 30H
JZ _CO_A_
```

Aveg:

```
MOV AX, DS:[BP + DI + 3]
MOV AH, 0
DIV CL
ADD AH, 30H
ADD AL, 30H
MOV [BX + 38], AL
MOV [BX + 39], AH
CMP AL, 30H
JZ _CO_R_
```

Rank2:

```
MOV AX, DS:[BP + DI + 4]
MOV AH, 0
DIV CL
ADD AH, 30H
ADD AL, 30H
MOV [BX + 45], AL
MOV [BX + 46], AH
CMP AL, 30H
JZ _CO_Re_
```

EN:

```
mov dx, offset GRADE
mov ah, 9h
int 21h

ADD DI,16
POP CX
MOV AH,0
```

```
            MOV [BX + 31], AH
            MOV [BX + 22], AH
            MOV [BX + 16], AH
            SUB CX,1
            JNS OUT_GRADE
            mov dx, offset xiantiao
            mov ah,9
            int 21h
            mov dx,offset fanye
            mov ah,9
            int 21h
            mov ah,8
            int 21h
            cmp al,'d'
            je right
            cmp al,'a'
            je left
rett:
        POP BP
        POP DI
        POP DX
        POP CX
        POP BX
        POP EAX
        RET
right:
        ;MOV AX,0003H;清屏
        ;INT 10H
        mov cx,4
        jmp OUT_GRADE
left:
        ;MOV AX,0003H;清屏
        ;INT 10H
        sub DI,160
        JS rett
        mov cx,4
        jmp OUT_GRADE
_CH_M_:
        MOV AH,31h
        MOV AL,30H
        MOV [BX + 17], AL
```

```
    MOV [BX + 16], AH
    JMP Math
_CH_E_:
    MOV AH,31h
    MOV AL,30H
    MOV [BX + 23], AL
    MOV [BX + 22], AH
    JMP English
_CH_A_:
    MOV AH,31h
    MOV AL,30H
    MOV [BX + 32], AL
    MOV [BX + 31], AH
    JMP Aveg
_CO_M_:
    MOV AL,0
    MOV [BX + 17], AL
    JMP Math
_CO_E_:
    MOV AL,0
    MOV [BX + 23], AL
    JMP English
_CO_A_:
    MOV AL,0
    MOV [BX + 32], AL
    JMP Aveg
_CO_R_:
    MOV AL,0
    MOV [BX + 38], AL
    JMP Rank2
_CO_Re_:
    MOV AL,0
    MOV [BX + 45], AL
    JMP EN

GRADE_CODE ENDP

CODE ENDS
END START
```

3.1.4 实验步骤

1. 准备上机实验环境。
   操作系统：windows 10 下 DosBox 0.74
   编辑器：sublime text
   编译器：masm 5.10
2. 在 TD 中有多种跟踪到子程序内的方法，尝试并记录，并观察函数返回时堆栈的变化
3. 编程中使用多种子程序参数传递的方法来传递子程序
   a) 阅读课本，重新理解子程序的调用方法
   b) 使用寄存器进行子程序传参
   c) 使用约定单元法进行主程序与子程序的传参
   d) 使用栈进行子程序传参
4. 修改自己的输出格式，让输出的信息更具有可观赏性
   a) 将原来竖着排的学生成绩单改成类似表格排列的形式
      i. 首先输出 成绩单 类似的提示字符
      ii. 然后输出需要输出信息的名称：人名；语文；数学；英语；总成绩；排名
      iii. 然后依次输出每个同学的相关成绩
   b) 调节程序，使之没有 BUG
   c) 运行自己的程序，将结果截图保存
5. 将自己的两个功能打包成两个文件，与队友的文件相链接，编译运行
6. 在 TD 中观察子程序的运行
   a) 在进入子程序前记录当前的堆栈段，CS、IP 的值
   b) 使用 TD 中的控制命令进入子程序
   c) 子程序结束返回后观察当前的堆栈段，CS、IP 的值
   d) 总结 CS、IP、堆栈段的变化。
7. 观察 FAR NEAR 类型子程序的 RET 指令的机器码有何不同？ 观察 FAR 类型子程序被调用时堆栈的变化
   a) 为了观察方便，自行写一个小程序，内含两个函数，一个使用 NEAR 类型，一个使用 FAR 类型
   b) 编译链接，生成 EXE 文件
   c) 使用 TD 对 EXE 文件反汇编，单步调试
   d) 观察两个 RET 语句的机器码的不同，记录
   e) 观察调用时堆栈段的变化，记录
8. 在 TD 中观察宏指令在执行过程中的替换和扩展，并解释宏和子程序的调用有何不同
   a) 写宏指令的小程序，编译链接，TD 反汇编
   b) 观察执行到宏指令处的指令和定义宏指令的语句
   c) 观察后解释现象，对比与子程序调用，解释不同。
9. 观察模块间的参数的传递方法，包括公共符号的定义和外部符号的引用，若符号名不一致或类型不一致会有什么现象发生？

a） 在自己的几个功能定义中使用不一致公共符号和外部符号

b） 观察运行现象使用一致的符号名，观察参数的传递方式

10. 通过把一个模块拆成多个模块或反之，体会子程序和模块化程序设计的方法，体会模块调用关系图、子程序功能说明、输入/输出说明在程序设计中的作用。

3.1.5 实验记录与分析

1. 准备上机环境，打开 dosbox，挂载工作目录

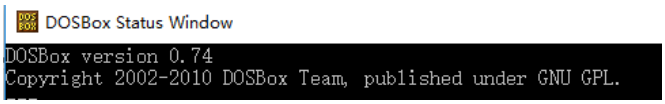


图 1- 1 DoxBox 版本

2. 记录 TD 进入子程序的方法

a) 阅读 TD 相关文档，得到进入子程序的方法

i. F7 键，当遇到 call 时，按下 F7 键将跟踪到 call 的函数内部

图 1- 2 使用 F7 进入子程序

ii. 使用断点也可以进入子程序中，在子程序

图 1- 3 使用 F2 打断点进入子程序

b) 使用 TD 观察函数返回时堆栈的变化

i. 进入子程序前

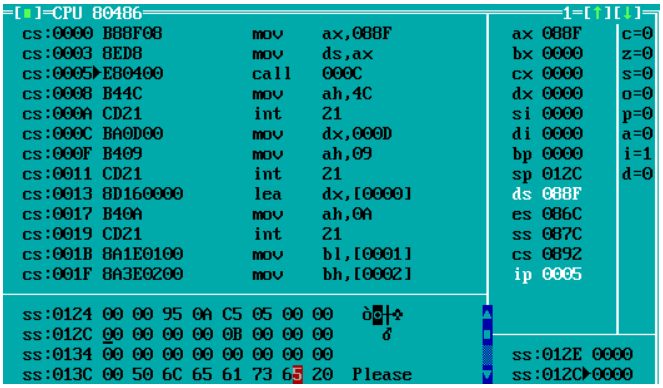


图 1- 3 TD 情况（进入子程序前）

记录 SS =087CH SP =012CH 指向元素：0000

当前语句 ip:0005H 下一句: ip:0008H

ii. 返回前

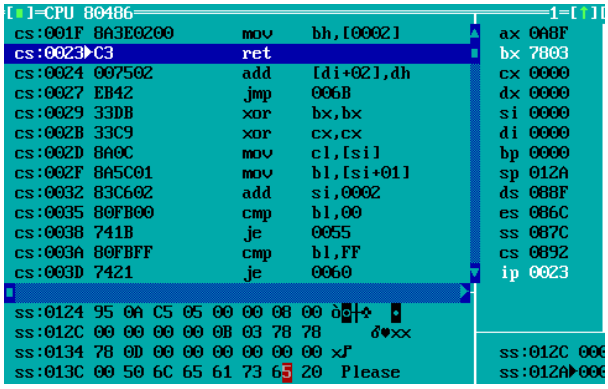


图 1- 4 TD 情况（返回前）

记录 SS = 087CH　　SP = 012AH　指向：　0008

iii.　返回后

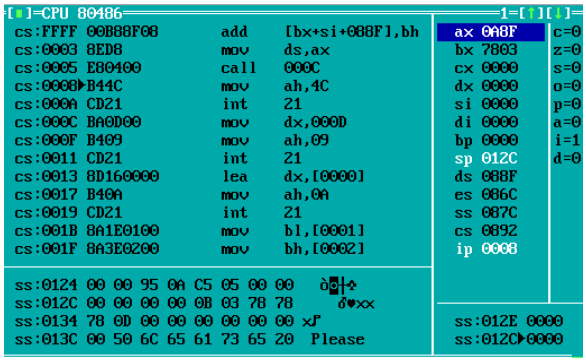


图 1- 5 TD 返回后

记录 SS = 087CH　　SP = 012CH

IP 变成 0008

可以观察到 call 语句将 call 语句的下一条语句的 ip 入栈，而 ret 语句将栈顶元素出栈到 ip，这便是 call 和 ret 语句的栈的变化。由于这个代码是段内调用，所以没有将 cs 入栈。

3. 使用多种子程序传参的方法传参数

阅读书本，总结主程序与子程序之间传递参数的方法

i.　寄存器法

子程序的入口参数和出口参数都存放在约定的寄存器中，传输速度快，但数量有限，适合参数不太多的时候。此方法记住一定不要将作为参数的寄存器保护起来

ii.　约定单元法

将入口参数和出口参数都约定到同一个内存单元中。

优点：不限制数量，不易出错

缺点：定义变量、数据出现复杂性

iii.　堆栈法

放在公开的堆栈区，处理完后将其回复原样即可，具有较高灵活性。

缺点：一定要注意他在堆栈中的位置，谨防 ret 时出栈的不是返回地址。

4. 修改自己的输出格式，让输出的信息更具有可观赏性

一开始自己的成绩单输出是一次性全部列出来，一个个输出，然后发现这样太不美观，而且输出信息太多时，无法看到被覆盖的信息，于是便使用如下的样式，并赋予翻页功能。

Student	Chinese	Math	English	Aveg	Rank
TEMPVALUE	80	90	95	0	0
TEMPVALUE	80	90	95	0	0
TEMPVALUE	80	90	95	0	0
TEMPVALUE	80	90	95	0	0
TEMPVALUE	80	90	95	0	0

图 1-6 输出格式

5. 将自己的两个功能打包成两个文件，与队友的文件相链接，编译运行

Link task + sub1 + sub2 + sub3 + sub4;

之后，便生成了我们最终需要的 exe 文件。

这其中发现了一个问题：就是同样的代码一开始在我的电脑上编译不过，但是复制到我队友的电脑上时却可以轻松通过，不禁引起了我们的困惑。后来发现我的 masm 较他的 masm 版本较老，我便把他的 masm 等一系列东西复制过来编译便可以通过。这也说明了我们一定要对自己现在所在的环境了如指掌，如此才能应对发生的一些本不应该发生的错误。

最后 exe 文件生成成功，运行如下所示：

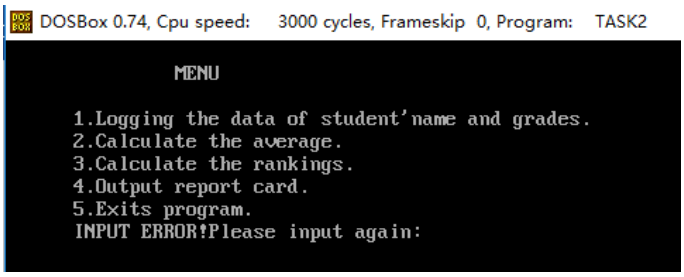


图 1-7 运行结果

6. 在 TD 中观察子程序的运行

a) 进入子程序前

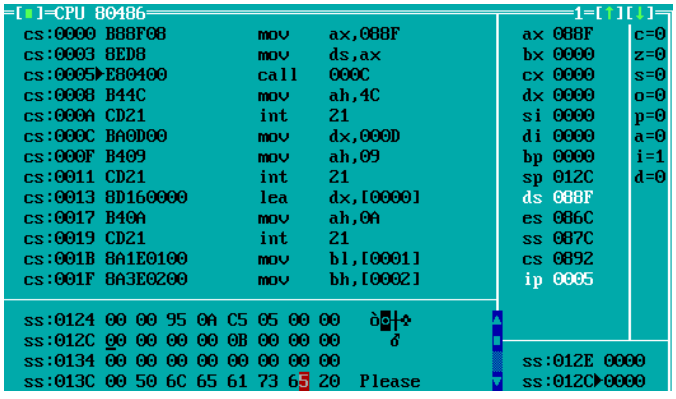


图 1-8 TD

记录 SS =087CH SP =012CH 指向元素：0000

当前语句 ip:0005H 下一句: ip:0008H

b) 返回前

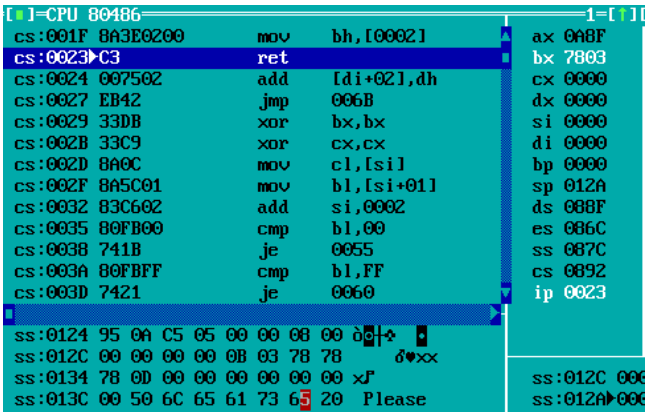


图 1- 9 TD 返回前（2）

记录 SS = 087CH　　SP = 012AH　指向：　0008

c)　返回后

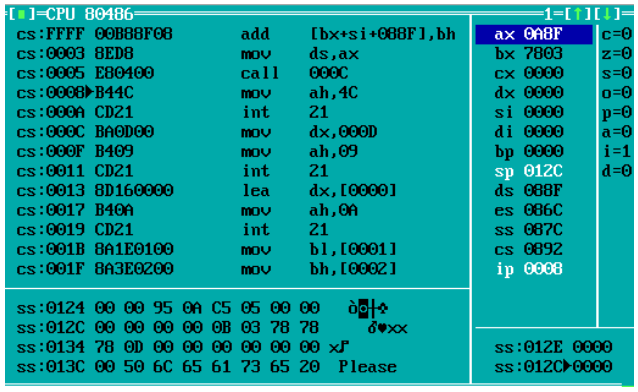


图 1- 10 TD 返回后(2)

记录 SS = 087CH　　SP = 012CH

IP 变成 0008

可以观察到 call 语句将 call 语句的下一条语句的 ip 入栈，而 ret 语句将栈顶元素出栈到 ip，这便是 call 和 ret 语句的栈的变化。由于这个代码是段内调用，所以没有将 cs 入栈。

7. 观察 FAR NEAR 类型子程序的 RET 指令的机器码有何不同？ 观察 FAR 类型子程序被调用时堆栈的变化

FAR 调用和 near 调用的区别就是一个是段间调用，一个是段内调用，所以 call 的时候就是一个是 CS:IP 都入栈了，而另一个只入栈了 IP，所以返回的时候一个需要 pop cs pop ip，而另一个只需要 pop ip 就可以了。

3.2 任务二

3.2.1 实验环境确定、设计思想及寄存器分配

1. 实验环境确定

实验的主要内容是实现不同语言的混合编程，发挥不同语言的优势。这次实验首先要选定一个 C 的开发环境。由于我的汇编程序大量使用了 DOS 调用，所以这个环境选定位 DOS 环境

会方便很多。

此实验环境下注意的地方：地址作为一个 16 位数据进行保存，所以在 C 语言向汇编传入地址信息时压入的信息长度刚好作为了地址。

最后选定 bc++3.1 为实验的开发环境，这样可以不必将大量精力放在环境本身。

2. 设计思想

主要遵循下面的原则：主菜单、输入输出使用 C 语言，计算平均成绩、排名使用汇编

C 语言作为主控制函数，在一个死循环中接收一个功能选择参数，1，2，3，4，5 对应五个功能，1 和 4 功能使用 C 语言实现，2 和 3 功能使用汇编和 C 的混合编程实现。每次选择后然后进入对应子功能，然后再返到主控制程序。若输入 5 退出死循环，结束程序。

C 语言主函数思想：while(1) + switch + getchar() 实现起来非常简单， 将学生信息定义成一个 定长 char 型数组。

与汇编语言混合编程的传参方法：使用栈进行传参，传入参数自动压栈，进入汇编程序后将栈中参数取出，如此操作即可让汇编程序也操作自己在 C 语言中定义的学生信息。

功能一实现：使用 C 语言重新编写。我们的目的是通过 C 语言函数将学生信息进行修改，所以分以下几步完成：第一步，接收数据，第二步，将数据写回学生信息。要完成两个问题：如何接收数据？如何写回？我们的上一个任务数据的存储方式是：姓名每个字母使用 8 位存储，在 C 语言中对应的类型有 char 类型，很容易做到。而成绩排名之类的数据我们是直接将该位数据的大小存进去，和 int 类型类似，但是直接使用 int 类型会超过原来数据的长度，所以我们应该将这个 int 类型写入 char，然后再放进去。对于写入，直接计算该位置下标，写入。

功能二 C 语言汇编混合编程：由于对混合编程还不是很熟悉，需要在编写过程中进行一步步调试，所以这里只能做粗略阐述。接收了学生信息的参数后，将原来得到 STU 地址的语句改变为进行出栈操作，其他代码即可保持不变。功能三和功能二的实现是相同的。

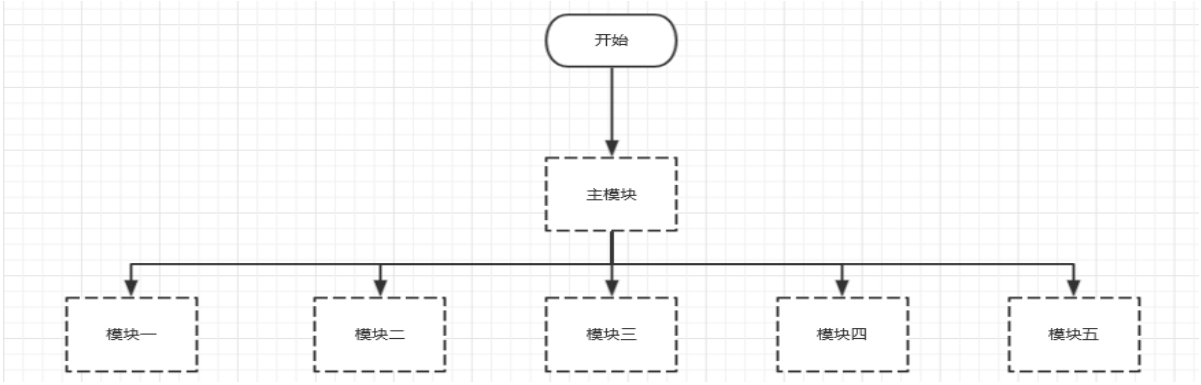
功能四是输出功能，使用 C 语言来实现，这非常之简单。

3. 寄存器分配

寄存器分配和上一个任务是相同的。

3.2.2 流程图

C 语言控制主流程图：



对各模块内部的调用任务一中已经有了阐述

3.2.3 源程序

```
#include <stdio.h>
#define N 5
char BUF[N * 16 + 1] ={
        'T','E','M','P','V','A','L','U','E','\0',80,90,95,50,0,0,
        'T','E','M','P','V','A','L','U','E','\0',80,90,95,0,0,0,
       'T','E','M','P','V','A','L','U','E','\0',80,90,95,0,0,0,
       'T','E','M','P','V','A','L','U','E','\0',80,90,95,0,0,0,
       'T','E','M','P','V','A','L','U','E','\0',80,90,95,0,0,0
        };
void SUB1(char* STU);
extern "C" void _Cdecl SUB2(char* STU);
extern "C" void _Cdecl SUB3(char* STU);
void SUB4(char* STU);

int main(int argc, char const *argv[])
{
        char c;
        int flag = 1;
        printf("-------------------------MENU-----------------------\n");
        printf("1.Logging the data of student',27h,'name and grades.\n");
        printf("2.Calculate the average.\n");
        printf("3.Calculate the rankings.\n");
        printf("4.Output report card.\n");
        printf("5.Exits program.\n");
        printf("Please input the number of function to choose:\n");
        while(1)
        {
                scanf("%c",&c);
                switch(c)
                {
                        case '1': SUB1(BUF); break;
                        case '2': SUB2(BUF); break;
                        case '3': SUB3(BUF); break;
                        case '4': SUB4(BUF); break;
                        case '5': flag = 0 ; break;
                }
                if(flag == 0) break;
        }
        return 0;
}

void SUB1(char* STU)
{
        int grade;
```

```
    printf("This is SUB1 function.\n");
    printf("Please Input the Student's ID which you want to change\n");
    while(1)
    {
        char c = getchar();
        if(c >= '0' && c <= N + '0')
        {
            printf("Please Input the New Name:");
            scanf("%s",STU + 16 * (c - '0'));
            printf("Please Input the New Chinese");
            scanf("%d",&grade);
            STU[10 + 16 * (c - '0')] = (char)grade;
            printf("Please Input the New Math");
            scanf("%d",&grade);
            STU[11 + 16 * (c - '0')] = (char)grade;
            printf("Please Input the New Chinese");
            scanf("%d",&grade);
            STU[12 + 16 * (c - '0')] = (char)grade;
            break;
        }
        else
        {
            printf("Input Error, ReInput\n");
            continue;
        }
    }
}

void SUB4(char* STU)
{
    int i = 0;
    printf("+ Student    Chinese    Math    English    Aveg    Rank    +\n");
    char grade;
    for(i = 0; i < N; ++i)
    {
        printf("%10s",&STU[i * 16]);
        grade = STU[i * 16 + 10];
        printf("%10d",(int)grade);
        grade = STU[i * 16 + 11];
        printf("%6d", (int)grade);
        grade = STU[i * 16 + 12];
        printf("%9d", (int)grade);
        grade = STU[i * 16 + 13];
        printf("%6d", (int)grade);
        grade = STU[i * 16 + 14];
        printf("%7d\n", (int)grade);
    }
    printf("-------------------------------------------------\n");
```

```
}

Sub2:
.model small
.code
public _SUB2
_SUB2 proc near
        push si
       push ax
       push cx
       push dx
       push bp
       mov bp,sp
       mov BX,[bp+12]

              MOV CX,5
S00:  PUSH BX
              MOV    AL,DS:[BX+10];the    grade -> AL
              MOV AH,0
              SHL AX,1
              SHL AX,1
              MOV DH,0
              MOV DL,DS:[BX+11]
              SHL DX,1
              ADD AX,DX
              MOV DL,DS:[BX+12]
              MOV DH,0
              ADD AX,DX
              MOV BL,7
              DIV BL

              POP BX
              ADD BX,16
              MOV DS:[BX-3],AL;restore the average grade
              LOOP S00

           pop bp
           pop dx
           pop cx
           pop ax
           pop si
           ret
_SUB2 endp
_text ends
       END

Sub3
.model small
```

```
.data
   N equ 5
.code
public _SUB3
_SUB3 PROC NEAR
              PUSH AX
              PUSH BX
              PUSH CX
              PUSH DX
              PUSH SI
              PUSH DI
              push bp

              mov bp, sp
              MOV BX,[bp + 16]

          MOV SI,13
          MOV CX,N
FIRST_loop:
           MOV DH,1
           MOV DI,13
           MOV AH,[BX + SI]
           PUSH CX
           MOV CX,N
SECOND_loop:
           PUSH AX
          SUB AH, BYTE PTR [DI + BX]
          JS _ONE_
_TWO_:
           POP AX
          ADD DI,16
          LOOP SECOND_loop
          MOV [BX + SI + 1],DH
          ADD SI,16
          POP CX
          LOOP FIRST_loop


              pop bp
              POP DI
              POP SI
              POP DX
              POP CX
              POP BX
              POP AX
          RET
_ONE_:
           INC DH
```

```
        JMP _TWO_
_SUB3 ENDP
    END
```

3.2.4 实验步骤

1. 准备上机环境
   a) 操作系统：windows 10 下 DoxBox 0.73
   b) 编译环境：BC++ 3.1 集成开发环境
   c) 编辑器：sublime text 3
2. 测试样例程序，确认上机环境无误
   a) 在汇编网站上下载样例程序
   b) 打开 bc，新建一个 obj 文件
   c) 将两个文件添加进去，并修改编译选项，然后 run
   d) 若无问题，继续下一步
3. 观察 C 语言编译器中对各种符号的命名规则（指编译器内部可以识别的命名规则，比如，符号名前面是否加下划线“_”，等），主、子程序之间参数传递的机制，通过堆栈传递参数后堆栈空间回收的方法。
4. 请尝试在 C 语言源程序中不合理地嵌入汇编语言的指令语句，达到破坏 C 语言程序的正确性的目的。
   a) 写一个测试用的 C 语言程序，包含 C 语言经常使用的全部语法，但是保持结构简单，这样易于观察调试。
   b) 在连续的几条 C 语言语句中间加入一条修改 AX 寄存器（或 DS 等其他寄存器）的汇编指令语句，而 AX 的内容在此处本不该被修改（即使用内嵌汇编指令）
   c) 在连续的 C 语言中插入 dos 中断指令
   d) 观察到破坏 C 语言程序正确性的效果
5. 观察 C 编译器的优化策略对代码的影响
   a) 打开 bc++3.1 的编译优化选项，全部勾选
   b) 编译运行，反汇编观察汇编代码，做记录
   c) 与第一次生成的反汇编进行对比，比较优化策略的影响
   d) 减少优化选项的勾选，例如将循环展开取消勾选，只勾选寄存器优化，再次观察代码做记录。
6. 通过调试混合编程的程序，体会与纯粹汇编语言编写的程序的调试过程的差异。

3.2.5 实验记录和分析

1. 准备上机环境
2. 测试样例程序

   使用 bc 建立好工程之后添加文件，运行：

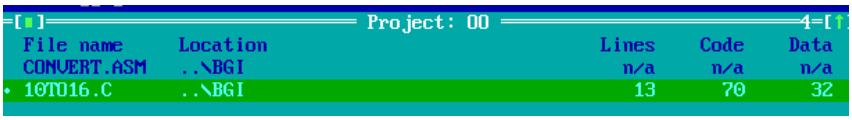


图 2-1 工程结构

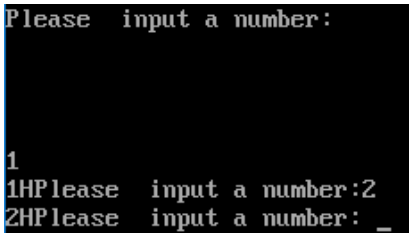


图 2-2 运行

样例程序执行无误，说明 bc 环境没有问题。

3. 了解 C 语言的对符号的命名规则，参数传递的规则

探究符号命名规则：

对下划线的探究：

图 2-3 下划线命名

图 2-4 运行

Bc 编译器允许变量以下划线开头

探讨参数传递方法：

进入 debug 模式：

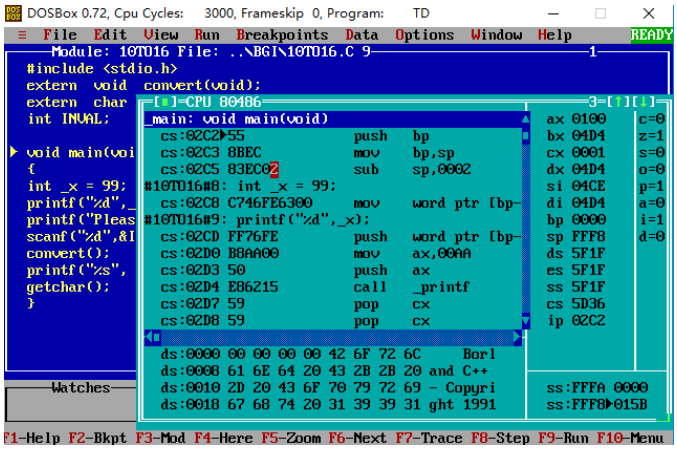


图 2-5 Debug 模式

观察堆栈段的变化：

我使用了 C 语言和汇编任务代码，进行子功能调用 2 来观察：

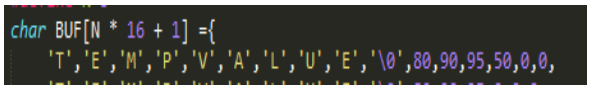


图 2- 6 入栈的元素

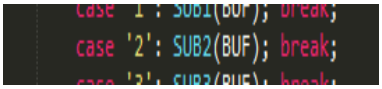


图 2- 7 函数调用

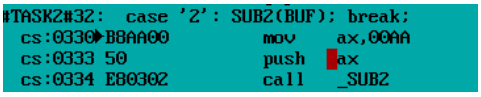


图 2- 8 debug 观察函数调用

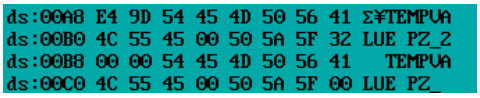


图 2- 9 堆栈段

观察发现在 call 子函数之前先将内容为参数的 AX 入栈，即可说明子程序调用时候的入栈顺序：先参数，然后 CS:IP，在编写汇编子程序时一定要注意这一点。

对于多个参数的函数，可以使用_Cdecl 这样的参数来指定他是从左向右入栈还是从右向左入栈。

4. 在 C 语言中不合理得嵌入汇编程序得到破坏汇编程序的目的

在上一个实验步骤中观察到在 C 语言中定义一个局部变量 flag 时观察到他是使用 si 来存放的，我便想在定义他之后使用内嵌汇编修改 si 的值，使 C 语言出错，但是当我内嵌修改 si 的值的汇编代码后并没有想象中的错误出现，再次调试之后发现这次编译器将 flag 存放在了 di 中，当我既修改 si 又修改 di 之后，他又把值存放到了寄存器中。可见编译器对代码的维护让你没有那么容易使 C 语言出错。

试试 dos 中断：

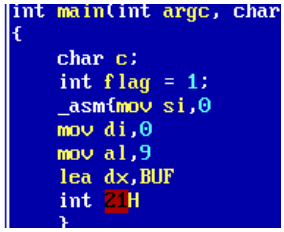


图 2- 10 插入汇编

这样内嵌汇编也没有问题

C 语言编译器会对 C 代码和汇编代码进行一定的优化，使你很难破坏他。

5. 通过调试混合编程的程序，体会与纯粹汇编语言编写的程序的调试过程的差异。

对于 C 语言和汇编的混合编程，我们可以充分发挥 C 语言和汇编语言各自的优势。我在实验中只使用了 C 语言调用汇编语言的内容。对于调用汇编语言，我们首先要注意到经过编译之后，C 语言中的变量名会在前方增添下划线，所以在汇编代码中定义自己要定义的函数需要在前方加上下划线。然后 C 语言调用汇编函数的时候需要使用 extern 关

键字来进行关联，格式要和老师的样例相同，不然会出现很多自己想不到的错误。

对于两个 C 语言向汇编函数传参，这一点我觉得是 C 语言汇编混合编程与单纯的汇编代码最大的不同，我们需要对汇编代码进行栈的相关维护，调入汇编子函数后，栈顶指向的是 CS：IP，而下一个元素就是入栈的参数，这一点要好好把握。

4 总结与体会

任务一使用了分工的方法进行编写程序,从中体会到了分工合作的方便性,也体会到了一定要将自己的程序测试到无误之后再去和队友的程序进行合并,否则容易发生错误。对于调试，td 仍然是一个强大的调试工具。这次实验在编写和调试中占用了大量时间，不过收获也很多。一开始甚至不知道 DOS 功能调用不可以在 windows 下使用，从而一开始想过在 windows 下实现他,这无疑是相当蠢的。

任务一自己也实现了输入输出的宏指令，使用起来的确非常方便，可以大大加快开发速率。模块化自己也踩了很多的坑，关键就是 public 和 extrn 这两个指令的使用。而关于段的合并却还不是特别熟悉，需要进行进一步复习。

大规模软件的调试，我和队友的程序曾经遇到这样的 bug：第一次输出成绩单无措，但是录入一个人之后再输出就出现了错误。由于这个操作涉及了两个人的代码，所以调试起来十分困难，因为不确定是录入的问题还是输出的问题，所以只好求助于 TD，在 td 中观察后发现是输入的问题，就此解决 bug。

任务二混合编程，由于老师给了代码的样例和如何在开发环境中使用他们，所以进行起来非常顺利。主要是在选择开发环境中进行了几次尝试，教程有 vc、vs、bc 三种，我的电脑上没有 vc 而且自己对 vs 比较熟悉所以一开始选择的是 vs 这个开发工具。一开始的内嵌汇编语句没有问题。但是在使用 obj 文件和 C 语言进行链接时，却发现一直报错。修改了很久的汇编代码格式，但是一直没有修改成功。最后求助于搜索引擎，搜索出的结果让我十分尴尬，int 中断是 dos 下面的东西而 windows 下面并不支持，一种自己好蠢的想法油然而生。最后选用 bc 之后一切便变得顺利了。

参考文献

[1] 王爽.汇编语言.第三版.清华大学出版社,2003 01- 310
[2] 曹忠升 80X86 汇编语言程序设计 华中科技大学出版社
