

華中科技大学

课程实验报告

课程名称：汇编语言程序设计实验

实验时段：2017年3月～2017年5月
指导教师：周英飚
专业班级：计算机卓越1501班
学号：U201514898
姓名：胡思勘

实验报告成绩：

实验序号	一	二	三	四	五	总评
成绩						

备注：总评是实验报告的最终成绩，是五次成绩的平均值，它将与平时实验过程中的表现一起构成本实验课程的最终成绩。

指导教师签字：
日期：

计算机科学与技术学院

目录

实验一 编程基础	1
I 实验目的与要求	1
II 实验内容	1
II.1 任务 1	1
II.2 任务 2	1
II.3 任务 3	2
II.4 任务 4	2
III 实验过程	3
III.1 任务一	3
III.1.1 实验步骤	3
III.1.2 实验记录与分析	3
III.2 任务二	6
III.2.1 设计思想及存储单元分配	6
III.2.2 流程图	7
III.2.3 源程序	7
III.2.4 实验步骤	8
III.2.5 实验记录与分析	9
III.3 任务三	11
III.3.1 设计思想	11
III.3.2 存储单元分配	11
III.3.3 流程图	12
III.3.4 源程序	12
III.3.5 实验步骤	13
III.3.6 实验记录与分析	14
III.4 任务四	16
III.4.1 设计思想	16
III.4.2 存储单元分配	16
III.4.3 流程图	16
III.4.4 源程序	17
III.4.5 实验步骤	17
III.4.6 实验记录分析	18
IV 总结与体会	20

实验二 程序执行时间与代码长度优化	22
I 实验目的与要求	22
II 实验内容	22
II.1 任务 1	22
II.2 任务 2	22
II.3 任务 3	23
III 实验过程	24
III.1 任务一	24
III.1.1 设计思想	24
III.1.2 流程图	24
III.1.3 源程序变更	24
III.1.4 实验步骤	24
III.1.5 实验记录与分析	25
III.2 任务二	26
III.2.1 优化思想	26
III.2.2 流程图变更	27
III.2.3 源程序变更	28
III.2.4 实验步骤	28
III.2.5 实验记录与分析	28
III.3 任务三	29
III.3.1 设计思想	29
III.3.2 流程图	29
III.3.3 源代码	30
III.3.4 实验步骤	30
III.3.5 实验记录与分析	30
IV 总结与体会	32
实验三 模块化程序设计	33
I 实验目的与要求	33
II 实验内容	33
II.1 任务一	33
II.2 任务二	34
III 实验过程	35
III.1 任务一	35

目录	III
III.1.1 设计思想	35
III.1.2 模块关系图及流程图	37
III.1.3 源程序	40
III.1.4 实验步骤	40
III.1.5 实验记录与分析	41
III.2 任务二	43
III.2.1 设计思想	43
III.2.2 模块关系图	44
III.2.3 源程序	44
III.2.4 实验步骤	45
III.2.5 实验记录与分析	45
IV 总结与体会	47
实验四 中断与反跟踪	
I 实验目的与要求	49
II 实验内容	49
II.1 任务一	49
II.2 任务二	49
II.3 任务三	50
II.4 任务四	50
II.5 任务五	50
III 实验过程	50
III.1 任务一	50
III.1.1 实验步骤	50
III.1.2 源程序	51
III.1.3 实验记录与分析	51
III.2 任务二	53
III.2.1 设计思想	53
III.3 中断地址	53
III.3.1 源程序	54
III.3.2 实验步骤	54
III.3.3 实验记录与分析	54
III.4 任务三	55
III.4.1 设计思想	55
III.4.2 源程序	55
III.4.3 实验步骤	55
III.4.4 实验记录与分析	56

目录	IV
----	----

III.5 任务四	56
III.5.1 设计思想	56
III.5.2 源程序	57
III.5.3 实验步骤	57
III.5.4 实验记录与分析	57
III.6 任务五	59
III.6.1 实验思想	59
III.6.2 实验步骤	59
III.6.3 实验记录与分析	60

IV 总结与体会	62
----------	----

实验五 WIN32 编程

I 实验目的与要求	63
II 实验内容	63
III 实验过程	64
III.1 设计思想	64
III.2 模块图	64
III.3 实验步骤	65
III.4 源程序	65
III.5 实验记录与分析	65
IV 总结与体会	67

附录 A 参考文献

附录 B 源代码	69
II.1 实验一	69
II.1.1 任务 4	69
II.2 实验二	75
II.2.1 任务 1	75
II.2.2 任务 2	78
II.2.3 任务 3	79
II.3 实验三	83
II.3.1 任务 1	83
II.3.2 任务 2	107

II.4 实验四	118
II.4.1 任务 1	118
II.4.2 任务 2	120
II.4.3 任务 3	122
II.4.4 任务 4	123
II.5 任务五	177

实验一 编程基础

I 实验目的与要求

1. 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具 TD 的使用；
2. 理解数、符号、寻址方式等在计算机内的表现形式；
3. 理解指令执行与标志位改变之间的关系；
4. 熟悉常用的 DOS 功能调用；
5. 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
6. 加深对转移指令及一些常用的汇编指令的理解。

II 实验内容

II.1 任务 1：《80×86 汇编语言程序设计》教材中 P31 的 1.14 题。

要求：

1. 直接在 TD 中输入指令，完成两个数的求和、求差的功能。求和/差后的结果放在 (AH) 中。
2. 请事先指出执行指令后 (AH)、标志位 SF、OF、CF、ZF 的内容。
3. 记录上机执行后的结果，与 (2) 中对应的内容比较。
4. 求差运算中，若将 A、B 视为有符号数，且 A>B，标志位有何特点？若将 A、B 视为无符号数，且 A>B，标志位又有何特点？

II.2 任务 2. 《80×86 汇编语言程序设计》教材中 P45 的 2.3 题。

要求：

1. 分别记录执行到“MOV CX, 10”和“INT 21H”之前的 (BX), (BP), (SI), (DI) 各是多少。
2. 记录程序执行到退出之前数据段开始 40 个字节的内容，指出运行结果是否与设想的一致。
3. 在标号 LOPA 前加上一段程序，实现新的功能：先显示提示信息“Press any key to begin!”，然后，在按了一个键之后继续执行 LOPA 处的程序。

II. 实验内容

II.3 任务 3. 《80×86 汇编语言程序设计》教材中 P45 的 2.4 题的改写。

要求：

1. 实现的功能不变，对数据段中变量访问时所用到的寻址方式中的寄存器改成 32 位寄存器。
2. 内存单元中数据的访问采用变址寻址方式。
3. 记录程序执行到退出之前数据段开始 40 个字节的内容，检查运行结果是否与设想一致。
4. 在 TD 代码窗口中观察并记录机器指令在内存中的存放形式，并与 TD 中提供的反汇编语句及自己编写的源程序语句进行对照，也与任务 2 做对比。（相似语句记录一条即可，重点理解机器码与汇编语句的对应关系，尤其注意操作数寻址方式的形式）。
5. 观察连续存放的二进制串在反汇编成汇编语言语句时，从不同字节位置开始反汇编，结果怎样？理解 IP/EIP 指明指令起始位置的重要性。

II.4 设计实现一个学生成绩查询的程序

实验背景：

1. 在以 BUF 为首址的字节数据存储区中，存放着 n 个学生的课程成绩表（百分制），每个学生的信息包括：姓名（占 10 个字节，结束符为数值 0），语文成绩（1 个字节），数学成绩（1 个字节），英语成绩（1 个字节），平均成绩（1 个字节）。例如：

```
1 N      EQU 30
2 BUF    DB  'zhangsan', 0, 0 ; 学生姓名，不足10个字节的部分用0填充
3 DB    100, 85, 80, ? ; 平均成绩还未计算
4 DB    'lisi', 6 DUP(0)
5 DB    80, 100, 70, ?
6 DB    N-3 DUP( 'TempValue', 0, 80, 90, 95, ?)
7 DB    'wangwu', 0, 0, 0, 0 ; 最后一个必须是自己名字的拼音
8 DB    85, 85, 100, ?
```

2. 功能一：提示并输入待查询成绩的学生姓名

- (a) 使用 9 号 DOS 系统功能调用，提示用户输入学生姓名。
- (b) 使用 10 号 DOS 系统功能调用，输入学生姓名。输入的姓名字符串放在以 in_name 为首址的存储区中。
- (c) 若只是输入了回车，则回到“(1)”处重新提示与输入；若仅仅输入字符 q，则程序退出，否则，准备进入下一步处理。

3. 功能二：以学生姓名查询有无该学生

- (a) 使用循环程序结构，在成绩表中查找该学生。
- (b) 若未找到，就提示用户该学生不存在，并回到“功能一 (1)”的位置，提示并重新输入姓名。
- (c) 若找到，则将该学生课程成绩表的起始偏移地址保存到 POIN 字变量中。

III 实验过程

III.1 任务一

III.1.1 实验步骤

1. 准备上机实验环境。
2. 在 TD 的代码窗口中的当前光标下输入三个运算式对应的两个 8 位数值对应的指令

```
1 MOV    AH, 011001B;
2 MOV    AL, 1011010B;
3 ADD    AH, AL;
4
5 MOV    AH, -0101001B;
6 MOV    AL, -1011101B;
7 ADD    AH, AL;
8
9 MOV    AH, 1100101B;
10 MOV   AL, -1011101B;
11 ADD   AH, AL;
```

观察代码区显示的内容与自己输入字符之间的关系；然后确定 CS:IP 指向的是自己输入的第一条指令的位置，单步执行三次，观察寄存器内容的变化，记录标志寄存器的结果。

3. 预计 ADD 执行之后：
 - 第一组：(AH) = 8D5AH SF=1 OF=1 CF=0 ZF=0.
 - 第二组：(AH) = 7AA3H SF=0 OF=1 CF=1 ZF=0.
 - 第三组：(AH) = 08A3H SF=0 OF=0 CF=1 ZF=0.
4. 输入 MOV AH,10H; MOV AL,-5H; SUB AH,AL; 观察标志位特点。
5. 输入 MOV AH,0FFH; MOV AL,-5H; SUB AH,AL; 观察标志位特点。

III.1.2 实验记录与分析

1. 实验环境条件：i7 3.6GHz, 8G 内存；Archlinux 下 DOSBox 0.74；TD.EXE 3.1。
2. 分别输入 3.1.1 中所记录的三组指令，执行三组指令后的结果分别如 I-III-1-1, I-III-1-2, I-III-1-3 所示。可以看出，计算结果在 AX 的高字节中与标志位的状态均与事前预期的一致的。

III. 实验过程

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip: 0, Program: TD
CPU 80486
ds:0000 = CD
ax 8D5A c=0
bx 0000 z=0
cx 0000 s=1
dx 0000 o=1
si 0000 p=1
di 0000 a=0
bp 0000 i=1
sp 0080 d=0
ds 4899
es 4899
ss 4899
cs 4899
ip 0106
[1]
cs:0100 B433 mov ah,33
cs:0102 B05A mov al,5A
cs:0104 02E0 add ah,al
cs:0106 0000 add [bx+si],al
cs:0108 0000 add [bx+si],al
cs:010A 0000 add [bx+si],al
cs:010C 0000 add [bx+si],al
cs:010E 0000 add [bx+si],al
cs:0110 0000 add [bx+si],al
cs:0112 0000 add [bx+si],al
cs:0114 0000 add [bx+si],al
cs:0116 0000 add [bx+si],al
cs:0118 0000 add [bx+si],al
ds:0000 CD 20 FF 9F 00 EA FF FF = f
ds:0008 AD DE C8 20 00 F0 CC 0A i¹ e¹ s⁰
ds:0010 A8 01 89 02 26 10 98 01 ZEE88>y
ds:0018 01 01 01 00 02 03 FF FF 0000 D
ss:0082 0000
ss:0080>0D00

图 I-III-1-1: 执行第一组语句后的状态

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip: 0, Program: TD
CPU 80486
ds:0000 = CD
ax 7AA3 c=1
bx 0000 z=0
cx 0000 s=0
dx 0000 o=1
si 0000 p=0
di 0000 a=0
bp 0000 i=1
sp 0080 d=0
ds 4899
es 4899
ss 4899
cs 4899
ip 010C
[1]
cs:0100 B433 mov ah,33
cs:0102 B05A mov al,5A
cs:0104 02E0 add ah,al
cs:0106 B4D7 mov ah,D7
cs:0108 B0A3 mov al,A3
cs:010A 02E0 add ah,al
cs:010C 0000 add [bx+si],al
cs:010E 0000 add [bx+si],al
cs:0110 0000 add [bx+si],al
cs:0112 0000 add [bx+si],al
cs:0114 0000 add [bx+si],al
cs:0116 0000 add [bx+si],al
cs:0118 0000 add [bx+si],al
ds:0000 CD 20 FF 9F 00 EA FF FF = f
ds:0008 AD DE C8 20 00 F0 CC 0A i¹ e¹ s⁰
ds:0010 A8 01 89 02 26 10 98 01 ZEE88>y
ds:0018 01 01 01 00 02 03 FF FF 0000 D
ss:0082 0000
ss:0080>0D00

图 I-III-1-2: 执行第二组语句后的状态

III. 实验过程

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD

[CPU 80486] ds:0000 = CD 1=[↑][↓]

Register	Value	Description
ax	08A3	c=1
bx	0000	z=0
cx	0000	s=0
dx	0000	o=0
si	0000	p=0
di	0000	a=0
bp	0000	i=1
sp	0080	d=0
ds	4899	
es	4899	
ss	4899	
cs	4899	
ip	0112	

```

cs:0100 B433    mov    ah,33
cs:0102 B05A    mov    al,5A
cs:0104 02E0    add    ah,al
cs:0106 B4D7    mov    ah,D7
cs:0108 B0A3    mov    al,A3
cs:010A 02E0    add    ah,al
cs:010C B465    mov    ah,65
cs:010E B0A3    mov    al,A3
cs:0110 02E0    add    ah,al
cs:0112 0000    add    [bx+si],al
cs:0114 0000    add    [bx+si],al
cs:0116 0000    add    [bx+si],al
cs:0118 0000    add    [bx+si],al
ds:0000 CD 20 FF 9F 00 EA FF FF = f 
ds:0008 AD DE C8 20 00 F0 CC 0A i l s o
ds:0010 A8 01 89 02 26 10 98 01 2E 8D 48 00
ds:0018 01 01 01 00 02 03 FF FF 0000 00

```

图 I-III-1-3: 执行第三组语句后的状态

- 输入 3.1.1 中剩下的两组指令，分别执行后所得结果如I-III-1-4和I-III-1-5所示，可以看看出，在求差运算中，如果两组数均视为有符号数，且 A>B，则 SF=0, OF=0, CF=1 运算产生了进位，若两组数均视为无符号数，且 A>B，则 SF=0, OF=0, CF=0。未产生借位。这说明无论是有符号数还是无符数，在内存中一律看做是无符号数来处理。

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD

[CPU 80486] ds:0000 = CD 1=[↑][↓]

Register	Value	Description
ax	15FB	c=1
bx	0000	z=0
cx	0000	s=0
dx	0000	o=0
si	0000	p=0
di	0000	a=1
bp	0000	i=1
sp	0080	d=0
ds	4899	
es	4899	
ss	4899	
cs	4899	
ip	0118	

```

cs:010E B0A3    mov    al,A3
cs:0110 02E0    add    ah,al
cs:0112 B410    mov    ah,10
cs:0114 B0FB    mov    al,FB
cs:0116 2AE0    sub    ah,al
cs:0118 0000    add    [bx+si],al
cs:011A 0000    add    [bx+si],al
cs:011C 0000    add    [bx+si],al
cs:011E 0000    add    [bx+si],al
cs:0120 0000    add    [bx+si],al
cs:0122 0000    add    [bx+si],al
cs:0124 0000    add    [bx+si],al
cs:0126 0000    add    [bx+si],al
ds:0000 CD 20 FF 9F 00 EA FF FF = f 
ds:0008 AD DE C8 20 00 F0 CC 0A i l s o
ds:0010 A8 01 89 02 26 10 98 01 2E 8D 48 00
ds:0018 01 01 01 00 02 03 FF FF 0000 00

```

图 I-III-1-4: 执行 10H-(5H) 后的状态

III. 实验过程

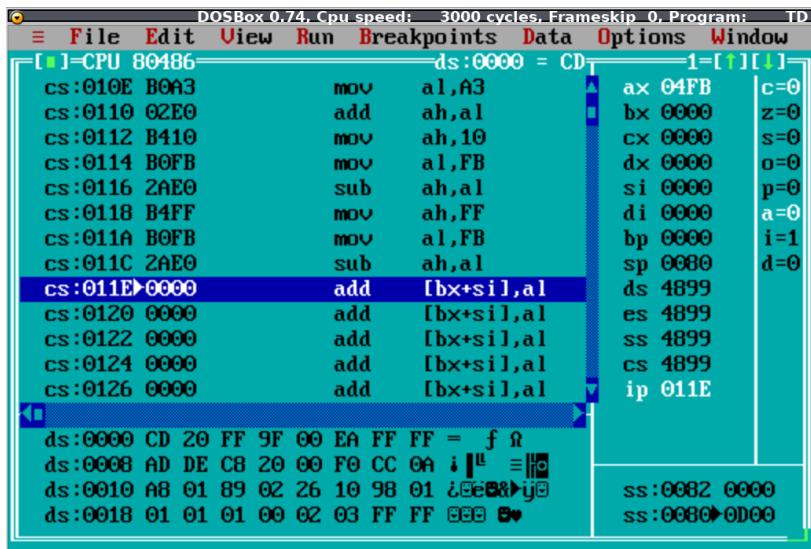


图 I-III-1-5: 执行 FFH-(-5H) 后的状态

III.2 任务二

III.2.1 设计思想及存储单元分配

求一个数的立方值可以用乘法运算实现，也可以造一立方表，运行时查表实现。依据本次实验的要求，此处用查表法。输入数据为 0 至 9 中任一自然数（可以考虑判断输入值的范围是否合乎要求），用一字节单元存放其值；输出数据是该数的立方，用一字单元存放其值。

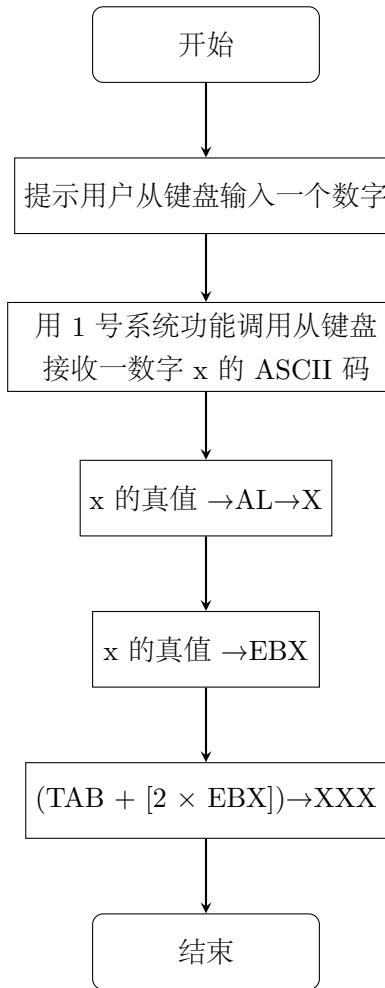
存储单元分配

- X: 字节变量 X 中存放键入的自然数 x。
 - XXX: 字变量 XXX 中存放 x 的立方值。
 - TAB: 立方表的首地址。表中共 10 项，每项占一个字，依次存放 0-9 的立方值。从表的结构可知，x 的立方值在表中的存放地址与 x 有如下的对应关系：
 - $(TAB + 2 * x) = x$ 的立方值
 - 对于每个键入的 x，从字单元 $TAB + 2 * x$ 之中取出的数据便是其立方值。
 - 从键盘接受数字使用 1 号系统功能调用，此时送入 AL 之中的是 x 的 ASCII 码而不是 x 的真值。所以，要首先将 x 的 ASCII 码换成 x 的真值，然后用 $TAB + 2 * x$ 计算 x 的立方值的存放地址，按此地址查到 x 的立方值。
 - INPUT: 字节存储区，用于存放提示信息。

寄存器分配

- EBX: 存放 x 的真值, 利用带比例因子的变址寻址方式访问立方表。
 - AX、DX: 临时寄存器。

III.2.2 流程图



III.2.3 源程序

```

1 \.386
2 STACK      SEGMENT USE16 STACK
3     DB          200 DUP(0)
4 STACK      ENDS
5
6 DATA       SEGMENT USE16
7 INPUT      DB \'PLEASE INPUT X(0-9):$\'
8     TAB      DW 0,1,8,27,64,125,216,343,512,729
9     X       DB 0
10    XXX     DW 0
11 DATA      ENDS
12
13 CODE      SEGMENT USE16
14 ASSUME    CS:CODE, DS:DATA, SS:STACK
15 BEGIN:
16     MOV      AX, DATA
17     MOV      DS, AX
18     MOV      DX, OFFSET INPUT
  
```

III. 实验过程

```
19      MOV     AH, 9
20      INT     21H          ; 显示 PLEASE INPUT X(0—9) :
21      MOV     AH, 1
22      INT     21H          ; 从键盘接受一数字 x 的 ASCII 码
23      AND     AL, 0FH
24      MOV     X, AL          ; x 的真值 → AL → X
25      MOV     EBX, EAX        ; x 的真值 → EBX
26      MOV     AX, TAB[EBX*2]    ; (TAB + [2 * EBX]) → AX
27      MOV     XXX, AX        ; 保存立方值
28      MOV     AH, 4CH
29      INT     21H
30  CODE   ENDS
31  END     BEGIN
```

III.2.4 实验步骤

1. 准备上机实验环境。
2. 使用 vim 录入源程序，存盘文件名为 CUBE.ASM。使用 MASM 6.0 汇编源文件。即 MASM CUBE；观察提示信息，若出错，则用编辑程序修改错误，存盘后重新汇编，直至不再报错为止。
3. 使用连接程序 LINK.EXE 将汇编生成的 CUBE.OBJ 文件连接成执行文件。即 LINK CUBE；若连接时报错，则依照错误信息修改源程序。之后重新汇编和连接，直至不再报错并生成 CUBE.EXE 文件。
4. 执行该程序。即在命令行提示符后输入 CUBE 后回车，观察执行现象。准备输入数字 3 进行测试，观察执行的结果。
5. 使用 TD.EXE 观察 CUBE 的执行情况。即 TD CUBE.EXE <CR>
 - (a) 观察 CS、IP、SP、DS、ES、SS 的值。
 - (b) 单步执行开始 2 条指令，观察 DATA 的实际值，以及 DS 的改变情况。
 - (c) 观察 SS: 0 至 SS: SP 区域的数据值。
 - (d) 观察 DS: 0 开始数据区，找到各变量在数据段中的位置和值。
 - (e) 观察第三条语句中源操作数的值，是否和 INPUT 变量的偏移地址相同。
 - (f) 执行第 3 至 7 条指令，输入数字 3。观察 AL 的值是否为 33H。
 - (g) 执行到 MOV AX, TAB[EBX*2]，观察源操作数的具体值。
 - (h) 执行 MOV XXX, AX，观察目的操作数的形式。到数据段中观察 XXX 的值是否是 3 的立方值。
 - (i) 依次输入 0 9，观察结果是否都正确。
 - (j) 输入字母 A，逗号标点等，观察运算结果。

III. 实验过程

6. 将程序重新装入 TD 中（或将 CS: IP 重置到 MOV AH, 9 的位置），在执行 9 号功能调用之前，用 TD 将数据段中 INPUT 缓冲区的 ‘\$’ (24H) 改成其他数值（如 00H），再执行 9 号功能调用，观察现象。
7. 当调用 1 号功能时，若输入大写字母 ‘A’，则送到 XXX 的值是哪个存储单元的值；若输入的是 ‘K’，则送到 XXX 的值又是哪个存储单元的值。

III.2.5 实验记录与分析

1. 实验环境条件：i7 3.6GHz, 8G 内存；Archlinux 下 DOSBox0.74；VIM.EXE 7.3；MASM.EXE 6.11；LINK.EXE 5.2；TD.EXE 3.1。
汇编源程序时，汇编程序没有出现错误。
连接过程显示了一个警告：LINK: warning L4201: no stack segment。
这指示了程序没有堆栈段，但对程序的正常测试不产生影响，因为程序中没有用到堆栈段，故忽略之。
2. 执行之后在新的一行上显示了字符串 PLEASE INPUT X(0-9): (如I-III-2-1所示)

```
C:\PLAYGROU>link CUBE.OBJ
Microsoft (R) Segmented Executable Linker Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992. All rights reserved.

Run File [CUBE.exe]:
List File [nul.map]:
Libraries [.lib]:
Definitions File [nul.def]:
LINK : warning L4021: no stack segment

C:\PLAYGROU>CUBE.EXE
PLEASE INPUT X(0-9):
```

图 I-III-2-1: 直接执行程序

输入 3 之后在冒号后显示了一个 3，程序就退出到命令行提示符。说明程序执行基本正常，但由于结果没有显示，所以需要用 TD 去观察计算结果是否正确。

3. 用 TD 调入 CUBE.EXE 后
 - (a) (CS)=0000H、(IP)=0000H、(SP)=0000H、(DS)=0884H、(ES)=0884H、(SS)=0893H。
可从段首址取值的不同得知代码段、数据段、堆栈段处在不同位置，且前后次序是数据段、堆栈段、代码段。
 - (b) 单步执行开始 2 条指令，DATA 的值 =08A1H，(DS)→0884H。观察到 TD 的数据显示区被切换到了 ES:0，若还希望显示 DS:0，就需要用 Goto 指令重新设定。发现此时数据段才是程序的实际位置，各个段的实际次序为堆栈段、数据段、代码段，这与源程序定义各段的次序一致。
 - (c) SS: 0 至 SS: SP 区域的数据值在程序没有执行时均为 0。单步执行一次后靠近栈顶的几个字发生了变化，不知为何？

III. 实验过程

- (d) DS:0 开始数据区存放了 INPUT 变量为首址定义的字符串。EA=15H 开始存放 TAB 立方值表。EA=29H 存放 X (当前值为 0); EA=30H 存放 XXX (当前值为 0)。
 - (e) TD 中显示的第三条语句为 MOV DX, 0000, 源操作数的值和 INPUT 变量的偏移地址相同 (均为 0)。
 - (f) 输入数字 3。AL 的值从 24H 变成了 33H。
 - (g) MOV AX, TAB[EBX*2] 在 TD 显示的形式为 MOV AX, [2*EBX+00000015] 说明 TAB 代表的 EA=00000015H, 且是按照双字处理的。
 - (h) MOV XXX, AX 在 TD 显示的形式为 MOV [002A], AX。执行后 DS:(002A)=001BH (即 27) 是 3 的立方。(如图I-III-2-2所示)。

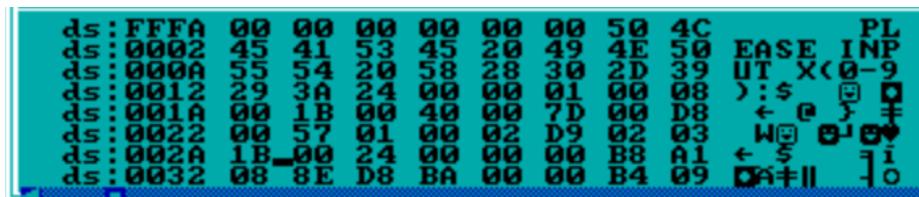


图 I-III-2-2: DS:(002A) 的值

- (i) 为了方便起见，打开 Watchs 窗口，输入 X 和 XXX，直接观察 X 和 X 的 3 次方的运行结果，并分别输入 0-9，发现 X 的立方值计算均正确。X=9 时的运算结果如下 I-III-2-3 所示



图 I-III-2-3: 当 X 为 9 时的结果

- (j) 当输入的是字母 A 时，在进行 AND AL, 0FH 的时候讲 ‘A’ 的 ASCII 码截断为了 1，因此计算出来的立方值为 1，当输入的是逗号的时候，在进行 AND AL, 0FH 的时候将 AL 截断为了 12，超出了 X 的预计范围，ebx 为 0CH, 取到了 TAB[2*12] 的界外值，在实际操作中为 0。

4. 重新载入程序，进行 9 好系统调用前，将 INPUT 字符串末尾的’\$’ 改为 ‘0’，然后继续进行系统调用，输出结果如I-III-2-4所示。可以看出，由于没有遇到终结字符，系统中断一直输出知道遇到了’\$’ 字符。因此在仅有的程序编写中要尤其注意不要忘记终结字符，否则会导致输出错误。

III. 实验过程

```
C:\>td
Turbo Debugger Version 5.0 Copyright (c) 1988,96 Borland International
PLEASE INPUT X(0-9):0 0 } + WE BS -
```

图 I-III-2-4: 错误的输出结果

- 重新加载程序，若输入的是‘A’，在与 0FH 进行‘与’操作后变为了 1，因此 XXX 的值是 TAB[1] 的值，若输入的是‘K’，在进行与操作后 AL 的值为 0BH，所以 TAB[2×EBX] 为 TAB[22] 实际读取的是 XXX 的高位字节和 XXX 的后一个字节组成的双字，实际结果也验证了这一想法（见 I-III-2-5）

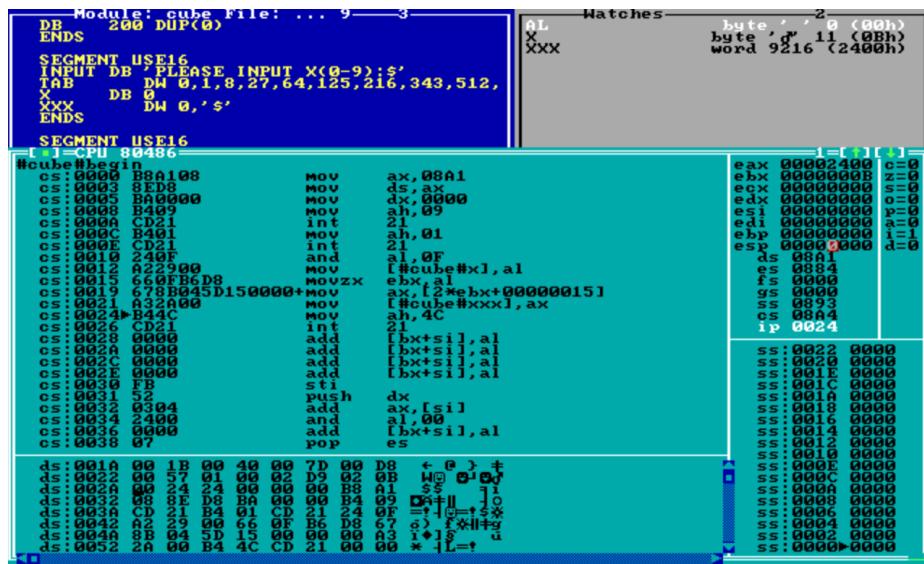


图 I-III-2-5: 输入‘K’的运行结果

III.3 任务三

III.3.1 设计思想

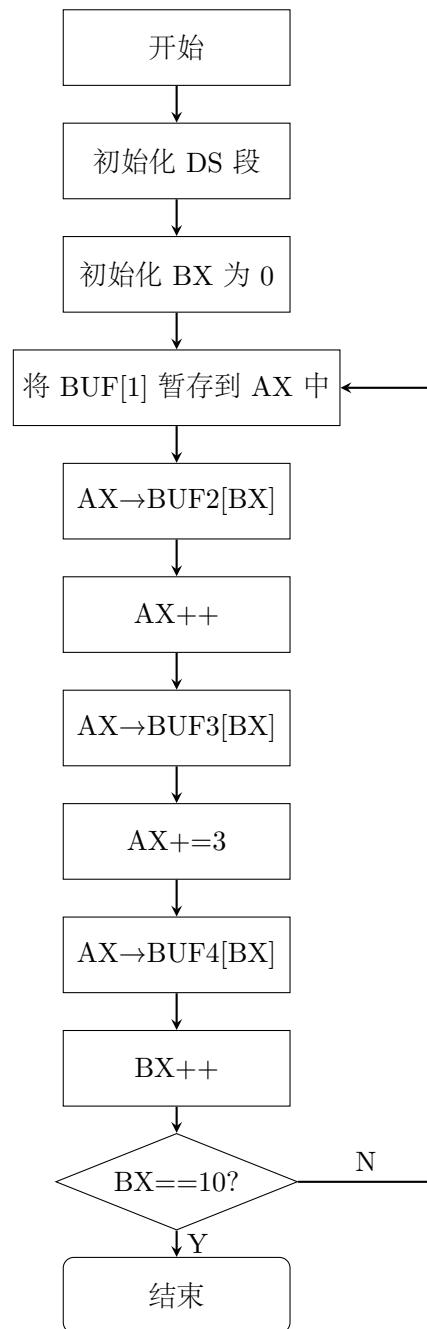
原程序的功能是在 LOPA 的十次循环中，将 BUF1 的值一一对应复制到 BUF2 中，将 BUF1 中的每个值加上 1 然后一一对应复制到 BUF3 中，将 BUF1 中的值加上 4 然后一一对应复制到 BUF4 中。

目的是将原程序改为使用变址实现同样的功能。原程序使用寄存器寻址，要将其改为变址寻址，可以将 BX 作为计数器，同时作为变址寻址使用的寄存器

III.3.2 存储单元分配

BUF1 到 BUF4 存储位置不变，依然在 DATA 区，计数器由 CX 变为 BX，同时 BX 作为变址寻址寄存器，这样就省去了寄存器寻址所需的寄存器。

III.3.3 流程图



III.3.4 源程序

以下是修改过后的源程序：

```
1 .386
2
3 STACK SEGMENT USE16 STACK
4     DB 200 DUP(0)
5 STACK ENDS
6
```

III. 实验过程

```
7  DATA SEGMENT USE16
8      BUF1 DB 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
9      BUF2 DB 10 DUP(0)
10     BUF3 DB 10 DUP(0)
11     BUF4 DB 10 DUP(0)
12 DATA ENDS
13
14 CODE SEGMENT USE16
15 ASSUME CS:CODE, DS:DATA, SS:STACK
16 START:
17     MOV    EAX, BYTE PTR DATA
18     MOV    DS, WORD PTR EAX
19
20     MOV    EBX, 0
21 LOPA:
22     MOV    AL, BUF1[EBX]
23     MOV    BUF2[EBX], AL
24
25     INC    AL
26     MOV    BUF3[EBX], AL
27
28     ADD    AL, 3
29     MOV    BUF4[EBX], AL
30
31     INC    EBX
32     CMP    EBX, 10
33     JNZ    LOPA
34     MOV    AH, 4CH
35     INT    21H
36 CODE ENDS
37 END START
```

III.3.5 实验步骤

1. 打开 dosbox，使用 vim 录入源程序，保存为 24_MODI.ASM。
2. 进行修改，将源程序改为 3.3.4 所示状态。
3. 使用 TASM 进行编译，同时生成符号表
4. 使用 TLINK 进行链接，生成 24_MODI.EXE，同时链接符号表
5. 打开 TD，装载 24_MODI.EXE
6. 在 watch 窗口中添加 BUF1、BUF2、BUF3、BUF4、AX、BX
7. 将 ds 监视窗口置 u 于 BUF1 处
8. 单步执行，观察 AX、BX、BUF1 ~ 4 的变化情况。

III.3.6 实验记录与分析

1. 实验环境条件: i7 3.6GHz, 8G 内存; Archlinux 下 DOSBox0.74; VIM.EXE 7.3; 编译、链接、调试使用 TASM 套件。编译、链接时没有出现错误。
2. 直接运行二进制文件, 没有任何输出, 程序正常推出, 说明要进入调试器进一步调试。
3. 打开 TD, 装载 24_MODI.EXE。
4. 在 WATCH 窗口中添加 BUF1 ~ 4, BX, AX
5. 在 CMP 处添加一断点。初始化完成后界面如图I-III-3-1所示

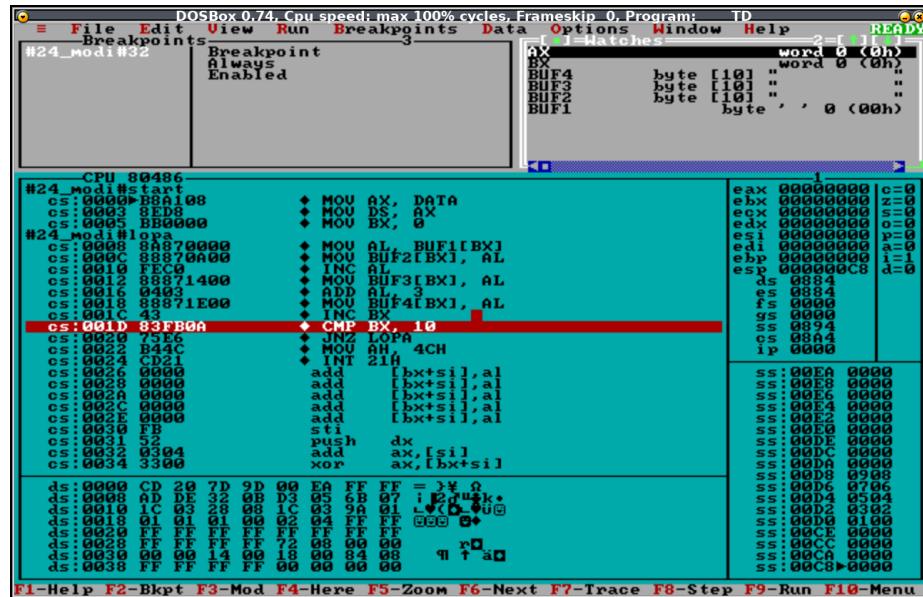


图 I-III-3-1: 程序装载完成界面

6. 单步执行, 观察寄存器和内存的变化, 发现在执行到断点处时, 寄存器和内存的变化均如意料进行, 第一次执行到断电处时内存如图I-III-3-2所示, 其中光标所在处为 BUF1 地址。

III. 实验过程

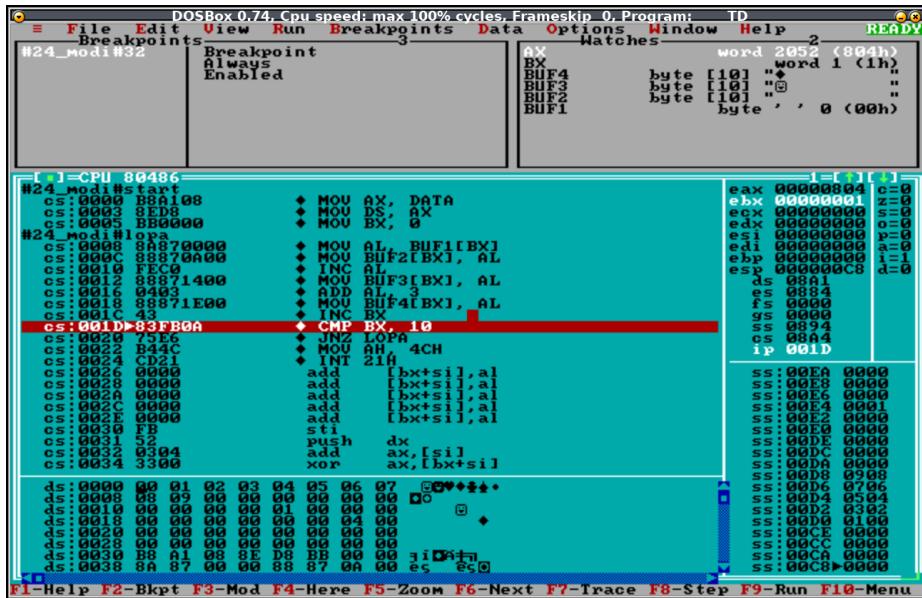


图 I-III-3-2: 第一次运行到断点

7. 然后使用 run 指令 (F9) 观察到每一次循环均将 BUF1 中对应的值移动到了 BUF2、BUF3、BUF4 中 (BUF3 中加上了 1、BUF4 中加上了 3)
8. 一直运行到第十次 (最后一次) 运行到断点, 结果如图I-III-3-3所示, BUF1 中所有的数均已正确转移。

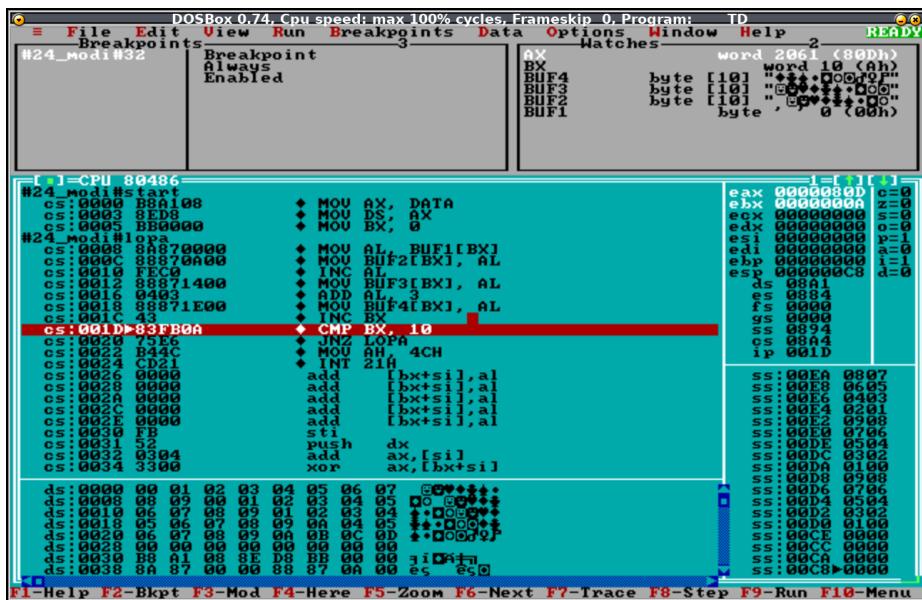


图 I-III-3-3: 最后一次循环完成

III.4 任务四

III.4.1 设计思想

本程序要求实现一个学生成绩查询程序，并对输入的信息，查找到的成绩做处理后存储并且输出。为了清晰结构，将程序分为以下几个部分：输入部分，查找部分（比较部分），成绩计算部分和输出部分，详细设计如下：

- 总体上，所有的字符串输出使用 int 21H 的 09H 号中断，需要输出的内容全部保存在数据段；所有需要输出的字符使用 int 21H 的 02H 号中断，需要输出的字符使用立即数表示；需要输入数据则使用 int 21H 的 0AH 号中断，需要输入的数据存在数据区的数据缓冲区中。由于本题所需输入的数据仅为学生的姓名，且长度不超过 10，因此数据输入缓冲区的长度为 10；
- 输入部分，对用户输入的数据进行判断，若为回车字符 (DH)，则判定为非法输入，提示重新输入；若为' q ' (71H, 0DH)，则判断为退出，直接终止程序；若为一串字母（含空格符，即 ascii 码为 20H, 41H 5AH, 61H 7AH）时，判断为姓名，进入查找部分继续执行；其余所有情况判断为非法输入，要求重新输入。
- 查找部分，对于预先存在数据区的姓名进行遍历，在遍历的主循环中：对姓名的每一个字进行逐字匹配，一旦发现失配则匹配不成功，进入下一个循环；若同时匹配到结尾，表明匹配成功，跳转进入计算部分；若所有的姓名均遍历完成仍没有匹配成功者，则表明姓名不存在，重新进入输入部分。
- 计算部分，利用 MUL, DIV 指令对查找到的姓名所对应的三个成绩进行计算，所得的结果存入对应姓名的平均成绩存储区中，然后对所得成绩进行评级，以 90, 80, 70, 60 为分界，落入五个区间的成绩分别评级为 A, B, C, D, F 并进行显示输出。

III.4.2 存储单元分配

大部分数据存储于数据段中，其中包括：

1. 100 个随机生成的姓名数据，和分数，每一项占 14 字节，其中姓名占 10 个字节，不足的部分用 0 补齐，语文、数学、英语成绩分别占用一个字节，平均成绩占用一个字节。
2. 输入缓冲区，一共 13 个字节，第一个字节存缓冲区长度，第二个字节存缓冲区读入字节数，第 3 到第 13 个字节存放读入的字符。
3. 提示用户输入姓名的字符串。
4. 提示用户姓名输入错误的字符串。
5. 显示学生信息的提示字符串。

III.4.3 流程图

图I-III-4-1展示了程序的主要流程

III. 实验过程

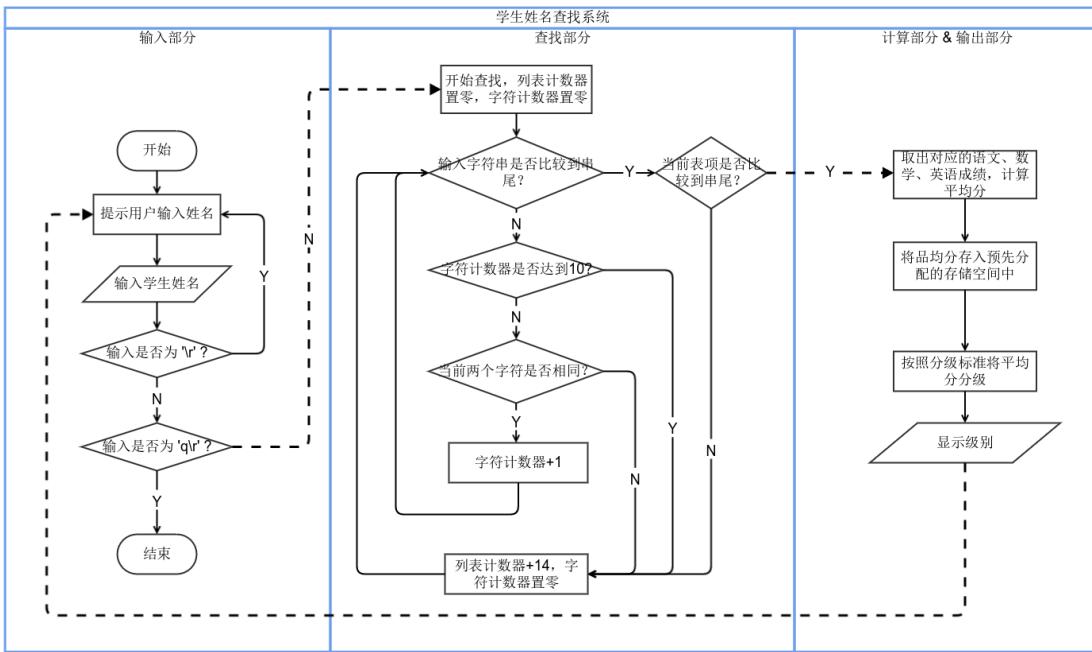


图 I-III-4-1: 程序总体流程

III.4.4 源程序

源程序详见附录部分。(第69页)

III.4.5 实验步骤

对应本程序的测试分为以下几个步骤：

1. 编译链接，修正编译器和连接器所提示的错误。
2. 直接运行，观察程序对输入的处理和输出有无明显错误。
3. 进入 Turbo Debugger 调试，分别对输入部分，查找部分，计算部分和输出部分进行测试，观察各个部分的功能是否正常。
4. 对于输入部分，分别测试存在于表中的姓名，不存在于表中的姓名，以及非法字符，观察程序是否能够正常处理。
5. 对于查找部分，更改查找部分的数据集，使得查找部分能够覆盖每一个跳转分支，并观察两个计数器的变化情况，判断程序是否正常执行。
6. 对于计算部分，手动修改计算部分的三门科目的成绩，观察计算所得的结果是否正确，以及是否有溢出产生。

III.4.6 实验记录分析

1. 实验环境条件: i7 3.6GHz, 8G 内存; Archlinux 下 DOSBox0.74; VIM.EXE 7.3; TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。
汇编和链接过程均没有出现错误。
2. 编译完成, 直接执行 STUDENT.EXE, 出现提示字符串' Please input the name of the student:'
3. 输入一个列表中已经存在的姓名, 如 Sixu Hu<>, 程序显示该学生平均分所对应的等级, 并继续提示用户输入学生姓名。
4. 输入一个列表中不存在的姓名, 如 asdf<>, 程序提示该学生不存在, 并要求重新输入。
5. 直接输入回车, 程序再次要求输入学生姓名。
6. 输入 q<>, 程序直接退出。2 6 步输出结果如图I-III-4-2所示, 初步表明程序能够正常运行。

```
C:\PLAYGROUN\STUDENT>STUDENT.EXE
Please input the name of the student: Sixu Hu
A
Please input the name of the student: asdf
Student not found, please re-input ...
Please input the name of the student:
Please input the name of the student: q

C:\PLAYGROUN\STUDENT>_
```

图 I-III-4-2: 直接运行程序的输出结果

7. 在平均分的计算部分以及存储部分需要进入调试器观察内存的具体情况。
8. 打开 TD, 加载 STUDENT.EXE, 在输入部分、查找部分和计算部分的入口分别插入断点, 然后执行, 监视列表计数器 esi、字符计数器 ecx, 列表当前字符 tab[esi+ecx], 输入串当前字符 in_buff[ecx], 按下 F9-Run 运行并观察变化。
9. 测试输入功能: 在提示输入姓名后, 输入 Sixu Hu<>, 程序运行到查找入口断点处, 在内存窗口跳转到 in_buff 输入缓冲区首地址处, 观察内存, 发现输入缓冲区存储的内容正确, 由于 dos 会自动限制输入字符个数, 因此输入不会超过 10 个字符。内存窗口如图I-III-4-3所示。这表明输入功能正常。
10. 测试查找功能: 单步执行, 在每一个判断跳转处观察当前的列表计数器以及字符计数器是否正常, 并观察跳转位置是否正常。发现程序在首次遇到不匹配字符时即刻进入列表下一行, 与预想一致。而当两个字符串同时匹配到结尾时, 程序跳转到计算部分。

```

ds:0572 00 00 64 01 60 00 0A 07 d@` @.
ds:057A 53 69 78 75 20 48 75 0D Sixu Hu
ds:0582 00 00 50 6C 65 61 73 65 Please
ds:058A 20 69 6E 70 75 74 20 74 input
ds:0592 68 65 20 6E 61 6D 65 20 he name
ds:059A 6F 66 20 74 68 65 20 73 of the s
ds:05A2 74 75 64 65 6E 74 3A 20 tudent:

```

图 I-III-4-3: 输入姓名后的内存缓冲区

11. 重复执行程序，改变输入名字，确保覆盖到查找部分的条件判断的每一种情况和每一个跳转分支，每层循环的次数测试尽可能覆盖到 0, 1, 2, m, n-1, n, n+1 次，其中 n 为最大可能循环次数 (对外层而言为 99, 在存在对应名字的情况下) 且 $2 < m < n$ ，以确保查找部分运作无误，所用到的测试用名字如表I-III-4-1所示 (分支对照表见图I-III-4-4)。经过测试，没有错误出现，查找部分功能正常。

表 I-III-4-1: 测试用姓名及覆盖分支

姓名	分支	内层循环次数覆盖	外层循环次数
Ju Song	1,2,3,6,8	7	0
Jin Lu	1,2,3,6,7,8	1,6	1
Yin Ma	1,2,3,6,7,8	0,6	2
Sixu Hu	1,2,3,6,7,8	0,7	4
Dong Ma	1,2,3,6,7,8	-	98
Jia Bai	1,2,3,6,7,8	-	99
Asdf	1,2,3,6,7,8	-	100
JuAAA	1,2,3,6,7,8	2	100
A	1,2,3,4,6,7,8	0	100

12. 测试计算部分：程序跳转到测试部分的入口处，手工修改内存中的程序数据，分别测试平均成绩为 0 分时，平均成绩为满分时以及可能造成溢出时的成绩数据，然后单步执行程序，观察平均分的计算以及存储是否正常运行，级别判断是否正确以及程序是否溢出。
13. 测试表明，即使总分数全部取最大，即 100,100,100，在执行加权以后总分为 350，没有超过一个 16 位寄存器的表示范围，而乘法的积与除法的被除数均为 16 位寄存器，因此不会溢出。
14. 至此，所有主要功能测试完成，没有发现异常。输出部分由于程序较为简单，在调试其他部分的同事观察每次输出无误，即表明输出部分功能正常。

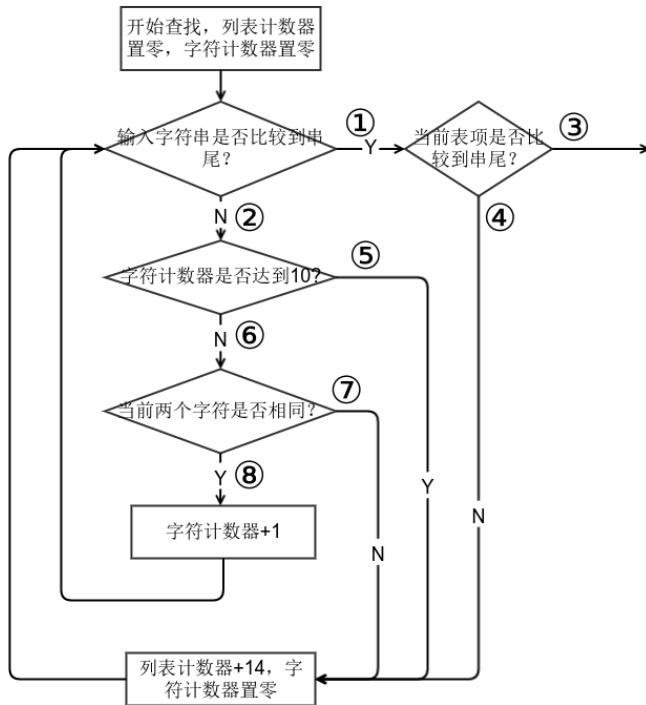


图 I-III-4-4: 分支对照图

IV 总结与体会

通过任务 1 的实验，让我知道了 TD 不仅可以调入现成的执行程序进行调试，而且能随时输入和测试单一一条指令是否正确，执行效果如何，这将方便未来的学习过程。另外，通过观察，计算机内的加减运算，无论是否有符号数，对应的标志位都是设定好了的，如何使用这些标志，完全由程序员选择的指令来决定，这就要求我们编写程序时要理解好题目的语义，选择合适的指令语句，而不是我写了个负数交给汇编程序，系统就会自动选择有符号指令的。

任务 2 的实验表明，源程序定义的段的次序，与调到内存中段的次序是有对应关系的。但数据段开始没有被赋值到实际的数据段首址中，需要执行了我们的相关赋值语句后才变得实际有效。另外，各种寻址方式在内存中确实不如源程序直观了，但可以看到变量的偏移值都被很好的计算出来了，我们可以通过源程序中变量与内存中对应位置的信息获得该变量的偏移地址，方便我们在数据显示区输入该地址，查看对应的变量内容。

任务 2 中若输入 0 9 之外的字符，结果其实是没有意义的。要解决这个问题，可以增加对输入的值的范围进行判断的语句，超出范围时报错，并要求重新输入。

在任务 4 中，我充分的学习到了使用汇编编写一个完整的程序、并且不断调试直到它能够正常运行的方法，题目 4 综合了前面三道题所学的几乎所有知识，并且让我们创造自己的程序，极大地增强了我们的编码能力和对代码的解读能力。同时，由于大量的调试工作的进行，我更加熟悉了 Turbo Debugger 的各种便于调试的功能和调试的方法（如打断点，监视，查找，跳转，录制宏等），提高了我的综合能力。

本次上机不仅提高了编程水平，熟悉了工具的使用，而且加深了对一些知识的理解。主要

的经验教训如下：

首先，更加感受到实验前准备的意义。例如：上机前准备越充分（如先编好源程序，制定好准备做的一些步骤），上机的时候目的越明确，可以解决较多的问题。

其次，录入程序时要注意一些细节，比如中文分号、字母 O 等问题，虽然汇编程序指出其所在行有错，但很难发现具体是哪个符号错了，耽误了不少时间。TD 在程序细节的观察、动态修改方面有很大的作用，要主动用 TD 的调试功能帮助自己发现、理解与解决问题。

最后，由于操作不够熟练，时间比较紧张等原因，还有些问题需要以后进一步解决，如堆栈中数据变化的原因、各个段在内存中存放的关系、是否可跟踪到 INT 21H 中去、多次调入程序时初始的段值是否相同等等。

实验二 程序执行时间与代码长度优化

I 实验目的与要求

1. 熟悉汇编语言指令的特点，掌握代码优化的基本方法；
2. 理解高级语言程序与汇编语言程序之间的对应关系。

II 实验内容

II.1 任务 1：观察多重循环对 CPU 计算能力消耗的影响

要求：

若有 m 个用户在同一台电脑上排队使用实验一任务四的程序，想要查询成绩列表中最后一个学生“wangwu 的”平均成绩，那就相当于将实验一任务四的程序执行了 m 次。为了观察从第一个用户开始进入查询至第 m 个用户查到结果之间到底延迟了多少时间，我们让实验一任务四的功能二和功能三的代码重复执行 m 次，通过计算这 m 次循环执行前和执行后的时间差，来感受其影响。由于功能一和功能四需要输入、输出，速度本来就较慢，所以，没有纳入到这 m 次循环体内（但可以保留不变）。请按照上述设想修改实验一任务四的程序，并将 m 值尽量取大（建议 $m \geq 1000$ ，具体数值依据实验效果来改变，逐步增加到比较明显的程度，比如秒级延迟），以得到较明显的效果。

提示：

1. 在进入功能二之前增加 m 次循环的初始化工作，在功能三结束之后增加 m 次循环的条件判断和转移语句。
2. 学校汇编教学网站的软件下载中提供了显示当前时间“秒和百分秒”的子程序。若在 m 次循环前调用一下该子程序， m 次循环执行完之后再调用一下该子程序，就能在屏幕上观察并感受到执行循环前后的时间差（时间差值需要自行手工计算，当然，你也可以选用网站上另一个计时程序，它是可以帮你计算好差值的）。注意，由于虚拟机环境下 CPU 会被分时调度，故该时间差值会因计算机运行环境与状态的不同而不同。

II.2 任务 2：对任务 1 中的汇编源程序进行优化

要求：

优化工作包括代码长度的优化和执行效率的优化，本次优化的重点是执行效率的优化。请通过优化 m 次循环体内的程序，使程序的执行时间尽可能减少 10% 以上。减少的越多，评价越高。

优化方法提示：

首先是通过选择执行速度较快的指令来提高性能，比如，把乘除指令转换成移位指令、加法指令等；其次，内循环体中每减少一条指令，就相当于减少了 $m*n$ 条指令的执行时间，需要仔细斟酌；第三，尽量采用 32 位寄存器寻址，能有更多的机会提高指令执行效率。

II.3 任务 3：观察用 C 语言实现的实验一任务四中功能一的程序与汇编语言实现的程序的差异

要求：

用汇编语言和 C 语言分别实现实验一任务四中功能一的功能（对汇编语言而言，就是把实验一中相关程序摘取出来成为独立的程序），对比两种语言实现的程序的代码情况，观察和总结 C 语言编写程序和自己的汇编语言程序的对应关系及差异，总结其中可以简化的地方。

提示：

采用反汇编方法观察 C 语言编写生成的执行程序时，首先观察程序的整体结构特点（包括段的情况），然后重点分析、比较输出输入字符串对应的代码。

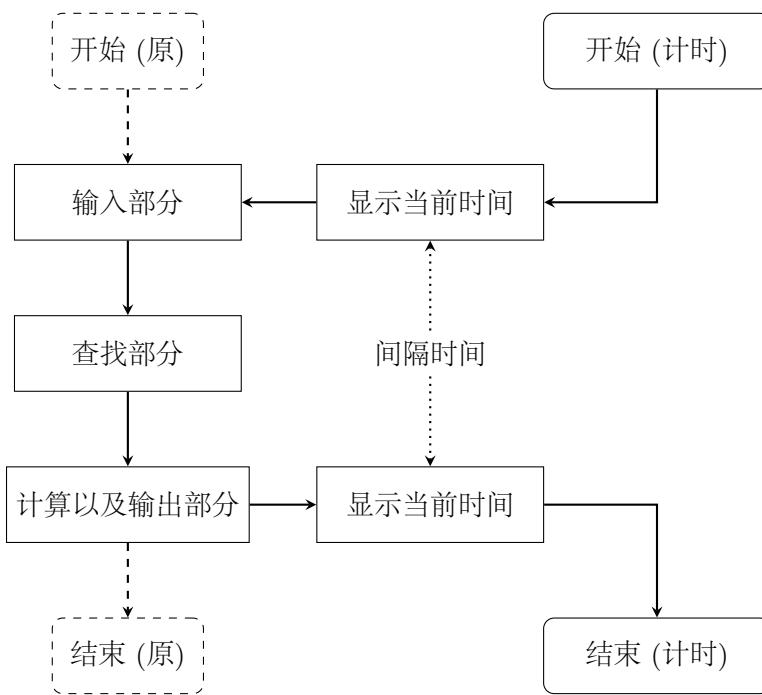
III 实验过程

III.1 任务一

III.1.1 设计思想

将查找最后一个学生的成绩并计算平均分的步骤进行重复，重复多次以达到时间统计的误差不超过 3%，并且可以忽略循环控制部分所消耗的时间。最后直接通过时间显示函数观察所消耗的时间，从而观察查找部分和计算部分对于 CPU 消耗的影响。同时，可以考虑单独测试循环部分和计算部分循环同样次数所消耗的时间，从而统计两个部分分占用的时间比例。

III.1.2 流程图



III.1.3 源程序变更

源程序变更详见附录部分。(第75页)

III.1.4 实验步骤

修改后的程序的测试分为以下几个步骤：

1. 修改源程序，进行编译链接，根据编译链接的错误修改程序并重复编译链接，直到没有错误出现。
2. 直接运行，输入最后一个学生的名字，观察有无运行时错误出现，若有，根据错误修改程序，并重新编译，链接，运行。
3. 重复运行，在相同条件下统计运行时间间隔并求平均值。

III. 实验过程

4. 修改源程序中时间测试循环的位置，同条件下重复运行，测试每个部分所占时间比例，并统计平均值。

III.1.5 实验记录与分析

1. 实验环境条件：i7 3.6GHz, 8G 内存；Archlinux 下 DOSBox0.74; VIM.EXE 7.3; TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。模拟 CPU 频率 3000cycle/sec
2. 修改源程序，使查找及输出部分重复 30000 次，并在循环前后打印当前时间。
3. 编译时出现了一个错误：标号超出距离范围。观察程序，由于时间测试循环的标号距离跳转位置太远，因此不能进行断转移，于是对程序作出以下修改：
 - 将标号 ‘loop_big_s:’ 更改为 ‘loop_big_s label far’
 - 将跳转语句 ‘loop loop_big_s’ 更改为 ‘dec cx; jmp far ptr loop_big_s’

修改后编译链接成功

4. 直接执行程序，输入最后一个学生的名字。发现由于循环了 30000 变输出，刷新了屏幕，而 DOS 的显示缓冲区不能向上滚动，从而不能显示开始时间，因此修改源代码，将计算部分的显示输出释掉，并重新编译运行。
5. 此时软件能够显示开始的时间和结束的时间。然而由于时间显示子程序的局限性，只能显示秒和百分秒，不能显示分钟，因此当运行跨过一分钟的末尾或者超过一分钟的时候，读数将不能正常显示。
6. 重新修改时间显示子程序，令其能够显示分钟，多次运行程序。
7. 并调节时间测量循环的位置，分别测量查找部分和计算部分所占用的时间比例，从而缩小程序热点所在的位置。计算所得结果如表II-III-1-1所示，查找 + 计算部分的程序输出如图II-III-1-1所示，另外两部分的时间测试输出结果格式与之相同。

表 II-III-1-1: 30000 次循环的时间测量结果

次数	查找 + 计算部分/秒	查找部分/秒	计算部分/秒
1	20.99	20.92	0.72
2	20.92	20.92	0.65
3	20.93	20.93	0.71
4	20.93	20.94	0.66
5	20.98	20.96	0.66
平均	20.95	20.934	0.68

```

C:\PLAYGROU\STUMOD1>STUMOD1.EXE
Please input the name of the student: Jia Bai
51'53"77
52'14"76
Please input the name of the student: Jia Bai
52'18"22
52'39"14
Please input the name of the student: Jia Bai
52'44"25
53'05"18
Please input the name of the student: Jia Bai
53'26"05
53'46"98
Please input the name of the student: Jia Bai
53'48"95
54'09"93
Please input the name of the student: q
C:\PLAYGROU\STUMOD1>_

```

图 II-III-1-1: 时间测试查找 + 计算部分运行输出

通过结果可以看出，相较于计算部分，查找部分占据了主要的时间，因此在优化的时候主要需要对查找部分进行优化而不是计算部分。

III.2 任务二

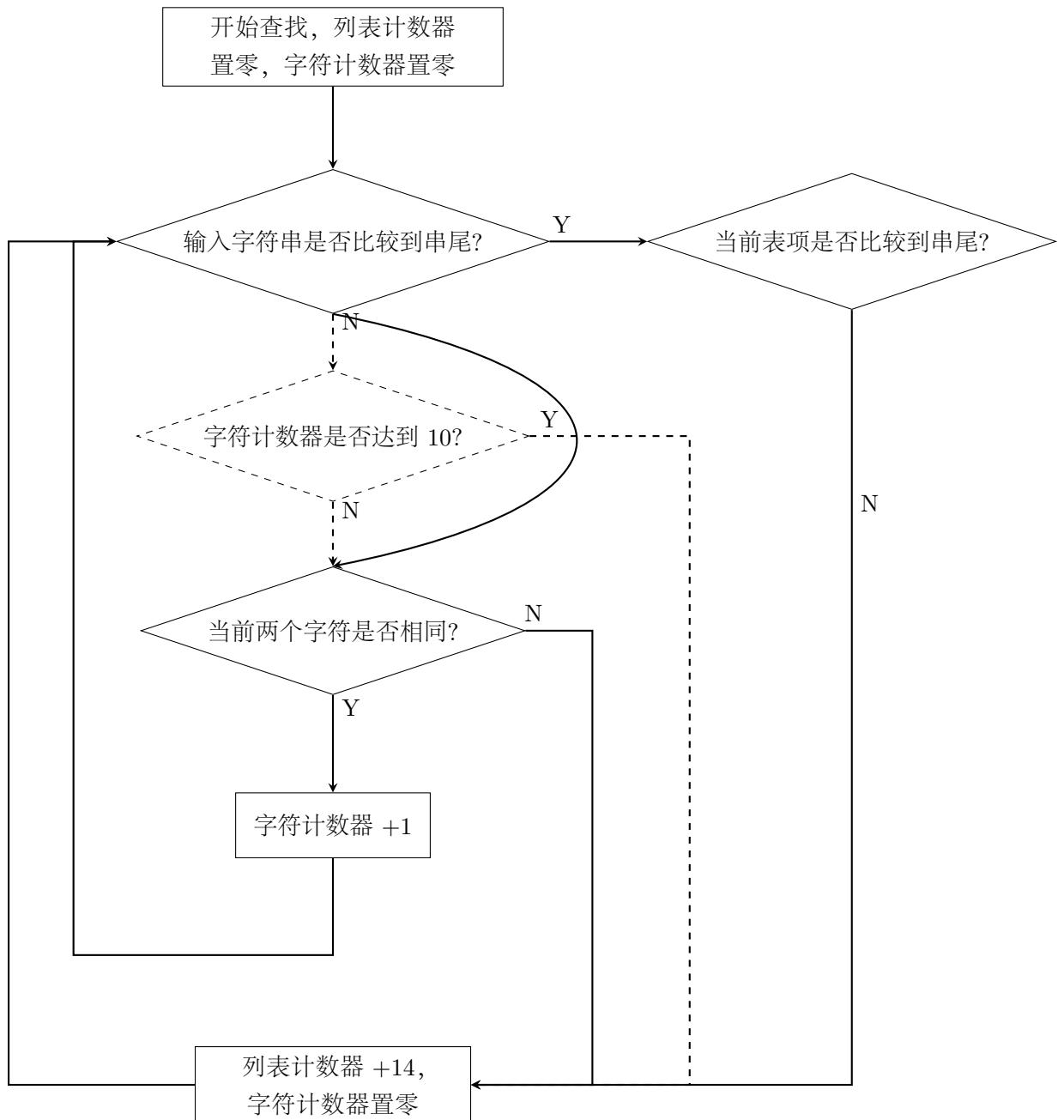
III.2.1 优化思想

- 在任务 1 的基础上对源程序进行优化，保留查找部分 + 计算部分的计时循环。
- 由于程序的主体逻辑在编写的过程中已经对查找时间进行了优化，在遇到不匹配时会直接进入下一循环，故不更改主体逻辑。
- 由于相对于计算部分而言，查找部分占据 99% 以上的时间（见表II-III-1-1），故主要对查找部分进行优化。

III. 实验过程

- 在查找主循环中，为了程序编写方便，使用了 esi 和 edi 两个计数器。现优化性能，只使用其中一个计数器进行计数。
- 在查找部分的字符比较次级循环中，由于添加了不必要的判断（即输入字符串到达串尾 '\r' 字符时判断表内字符串是否到达第 10 个字符且不为 0），而实际上由于输入字符串被限制在 9 个字符（除掉 '\r' 字符），因而不可能出现这种情况，故将此判断删去。
- 由于 dos 系统限制，输入字符串的最后一个字符一定是”，故不需要判断是否已经比较到第 10 个字符以避免数组越界。将这一部分删除。

III.2.2 流程图变更



III.2.3 源程序变更

源程序变更详见附录部分。(第78页)

III.2.4 实验步骤

1. 对任务一的程序进行修改，并编译链接，若有错误，重复修改、编译、链接步骤，直到没有错误出现为止。
2. 直接运行程序，输入最后一个学生的名字，观察有无运行时错误，若有，重复修改、编译、链接、运行的步骤，直到没有错误出现为止。
3. 在和任务一相同的条件下（即模拟 CPU 频率相同，输入相同）重复运行程序，统计运行时间结果。
4. 将得到的时间结果与任务一所得结果进行对比，观察优化程度。

III.2.5 实验记录与分析

1. 实验环境条件：i7 3.6GHz，8G 内存；Archlinux 下 DOSBox0.74；VIM.EXE 7.3；TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。**模拟 CPU 频率 3000cycle/sec**
2. 按照优化思想对源程序进行修改，主要是去除不需要的判断和跳转，优化查找时间。
3. 对修改后的源程序进行编译、链接。在编译、链接的过程中没有出现错误。
4. 在与任务一相同的条件下直接运行程序，输入最后一个学生的名字，没有出现运行时错误。
5. 重复运行优化后的程序，记录并统计平均值，统计结果如表II-III-2-1，程序输出如图II-III-2-1。

表 II-III-2-1: 优化前后查找 + 输出时间对比

次数	优化前/秒	优化后/秒	节约时间/秒	优化程度/%
1	20.99	16.92	4.07	19.39
2	20.92	16.98	3.94	18.83
3	20.93	16.92	4.01	19.16
4	20.93	16.97	3.96	18.87
5	20.98	16.92	4.06	19.35
平均	20.95	16.94	4.01	19.12

III. 实验过程

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\PLAYGROUN\WORKSET2\STUTUNE>STUTUNE.EXE
Please input the name of the student: Jia Bai
35'51"48
36'08"40
Please input the name of the student: Jia Bai
36'11"03
36'28"01
Please input the name of the student: Jia Bai
36'31"63
36'48"55
Please input the name of the student: Jia Bai
36'50"53
37'07"50
Please input the name of the student: Jia Bai
37'10"63
37'27"55
Please input the name of the student: q
C:\PLAYGROUN\WORKSET2\STUTUNE>_
```

图 II-III-2-1: 优化后程序运行输出

6. 可以看出，对于循环较多的部分以及程序的热点有针对性的优化能够大幅度的提高程序的性能。

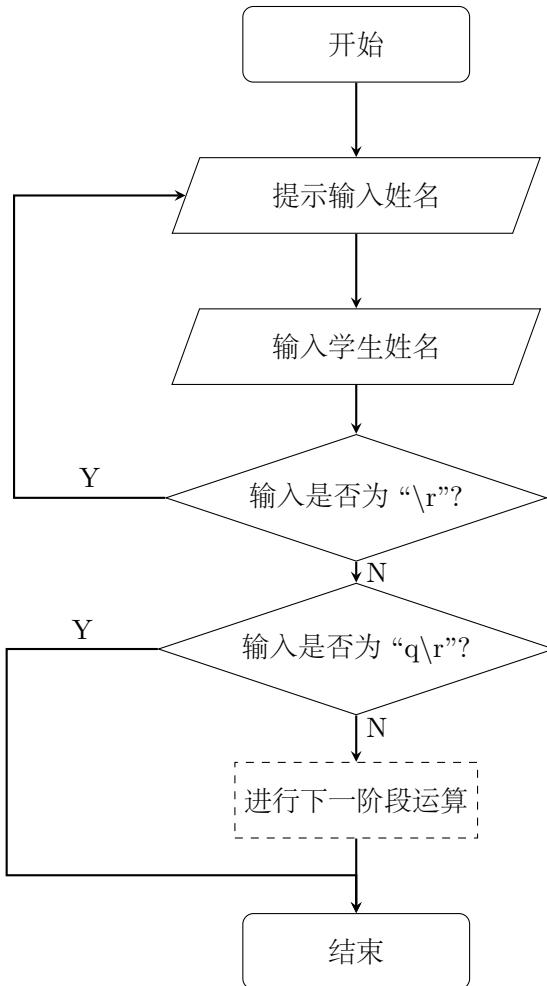
III.3 任务三

III.3.1 设计思想

- 使用 C 语言实现与汇编语言相同的功能，并进行反汇编后和汇编语言进行比对，了解编译器是如何将 C 语言转换为汇编语言的。
- 通过观察反汇编的内容，了解手写汇编与编译器编译相比的优势和劣势。

III.3.2 流程图

以下是 C 语言编写功能一程序的流程图：



III.3.3 源代码

源代码详见附录部分。(第79页)

III.3.4 实验步骤

- 对于汇编而言，将第一部分（输入部分）单独提取进行编译。
- 对于 C 语言而言，实现与输入部分相同的功能，并在相同的环境中进行编译。
- 分别运行、调试 C 程序和汇编程序，以确保功能正常。
- 对比 C 程序转换成的汇编和原汇编程序，了解编译器对于 C 语言的汇编实现、编译器的实现与手写汇编的区别。

III.3.5 实验记录与分析

- 实验环境条件：i7 3.6GHz, 8G 内存；Archlinux 下 DOSBox0.74; VIM.EXE 7.3; TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。C 语言编译使用 Borland Turbo C++ 3.0

III. 实验过程

- 按照设计思想编写 C 程序，使用 TCC 进行编译，运行程序，程序能够正常运行，运行输出如图II-III-3-1所示。

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX  
C:\PLAYGROUN\WORKSET2\STU_C>STU_C.EXE  
Please input the student's name: q  
C:\PLAYGROUN\WORKSET2\STU_C>STU_C.EXE  
Please input the student's name: abcd  
entering next procedure...  
C:\PLAYGROUN\WORKSET2\STU_C>STU_C.EXE  
Please input the student's name:  
Please input the student's name:  
Please input the student's name: q  
C:\PLAYGROUN\WORKSET2\STU_C>_
```

图 II-III-3-1: C 语言程序编译运行输出

- 编译提取出的汇编程序，使用 TASM 进行编译，运行程序，程序运行结果与实验一任务四功能一的输出相同。
- 使用 TCC -S <input-file> 对 C 语言进行翻译，得到汇编结果如源代码部分所示。
- 将手写的汇编和生成的汇编进行对比，可以发现，在长度上，生成的汇编长度远大于手写的汇编长度。但对于源程序的编写而言，C 语言的编写逻辑更为清晰，代码更为简洁。
- 对于生成的汇编代码进行分析，发现在程序主题部分（main 函数部分）被定义到 _TEXT 段中，所有用到的常量字符串都被预先定义到 _DATA 段中，而数组也事先在 DATA 段中分配了空间。
- 在输入输出时，直接调用了 printf 和 scanf 子程序，并通过 push ax 压栈传递参数。
- 对于判断和跳转部分，编译器生成的代码和手写的代码非常相似，均为使用 cmp 进行比较以后根据标志位进行跳转。
- 可以看出，C 语言和汇编在编写这一程序时，主要不同之处在于输入与输出部分，使用汇编编写时直接使用系统中断实现输入与输出，而 C 语言则是调用子程序进行输入输出，这也是主要的性能差异所在。

IV 总结与体会

通过任务一的实验，我了解到了测量汇编程序对于 CPU 占用的基本方法，并成功的对程序进行了基本的分析，定位了占用 CPU 较多的程序段，为下一步的针对性优化打下了基础。

在任务二中，动手实践对程序的优化使我对汇编语言的指令集有了更为深入的了解，并了解到了程序优化的基本方法以及程序优化的重要性。在汇编程序中，有一些指令可以用另一些更为简洁、高效率的指令所代替，还有一些经过逻辑判断以后发现不需要的分支，这些都构成了优化的主体部分。更为重要的是，我了解到了有针对性优化的重要性。若在第一步中没有对程序进行分段分析，那么我可能会对计算部分段进行较为深入的优化，然而实际上计算部分所占用的时间比例非常小，优化的价值不高，因此对于查找部分的优化才能够显著提高程序性能。

在任务三中，我了解到了编译器将 C 语言转换为汇编语言的基本途径，以及对于常量、变量、判断以及转移的处理。编译器对于 C 语言的处理有很强的模式性，例如在调用子程序时总是通过堆栈传参（内联函数除外），不同循环中的跳转指令除标号以外基本一致等。这也导致了在使用 C 语言的时候有一些可以使用寄存器以及简单指令来提升速度的地方不能得到优化。对于输入输出部分，C 语言与汇编有较大的区别。C 语言常用的输入输出均是通过调用标准库中的函数实现的。在汇编中，除了通过调用函数以外，还可以直接通过系统中断来实现，这样可以在一定程度上节省时间。

总体而言，这一次实验使我了解到了程序测试与优化的基本方法，了解到了底层语言与高级语言之间的主要区别以及转换关系，这些知识在今后的学习尤其是对程序的优化过程中将会发挥更大的作用。

实验三 模块化程序设计

I 实验目的与要求

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。

II 实验内容

II.1 任务一：宏与子程序设计

进一步修改与增强实验二的学生成绩查询程序的功能，具体要求如下：

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。
提示：由于学生姓名和成绩是通过程序录入的，因此，定义学生成绩表缓冲区时，初始值都可以置零。为避免录入成绩的时间过程太长，假定学生人数在 5 人左右，具体人数自行决定。每个学生成绩表中增加一个字用来保存排名。平均分的结果只需保留整数部分。
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。注意，在每个模块的开始，注明编写者的名字以及同组同学的名字。
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）。
5. 将 9 号和 10 号 DOS 系统功能调用定义成宏指令并调用。

提示：

II. 实验内容

1. 在 TD 中跟踪到子程序内部有几种方法？在 TD 中观察子程序调用和返回时堆栈的变化。
2. 注意观察 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情况。
3. 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。
4. 观察不同模块的可合并段合并后变量偏移地址的变化情况。观察不同段在内存里的放置次序。体会模块间段的定义及其对应的装配方法。
5. 在编程中使用不同的子程序参数传递方法来编写子程序。
6. 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？
7. 通过 TD 观察宏指令在执行程序中的替换和扩展，解释宏和子程序的调用有何不同。
8. 如何使菜单和成绩单显示得更漂亮一点？
9. EXTRN 说明语句放在 .386 之前或者之后有什么区别？
10. EXTRN 说明的变量的段与段寄存器的关联关系（ASSUME 伪指令所表达的信息）是否能带入到本模块中？如果不能带入，是否可以通过加段前缀的方法来解决？
11. 如何利用宏功能使汇编语言的程序变得更加直观易读？

例：下面是一个利用宏功能直观化后的完整代码段程序，请写出对应的宏定义，并模仿该方式对自己编写的某段程序进行类似的改写。

```
1 , start
2 Initial_ds
3 GetStringTo    BUF
4 DisplayStringFrom  BUF
5 ExitToDOS
6 EndProgram  code, start
```

II.2 任务二：在 C 语言程序中调用汇编语言实现的函数

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。

要求与提示：

1. 在不同的 C 语言开发环境中实现与汇编语言程序的混合编程，其操作方法有可能是不同的。请大家选择自己熟悉的 C 语言开发环境并查找相关的资料完成本实验。
2. 在实验报告中，比较详细的给出你的开发环境及其实现方法。

III. 实验过程

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. 对混合编程形成的执行程序，用调试工具观察由 C 语言形成的程序代码与由汇编语言形成的程序代码之间的相互关系，包括段、偏移的值，汇编指令访问 C 的变量时是如何翻译的，等。
5. 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。比如，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，这样就可观察到破坏 C 语言程序正确性的效果（该项实验表明：在 C 语言程序中，若不考虑上下语句翻译成怎样的机器码而随意嵌入汇编指令语句时，有可能存在出错的风险）。
6. 观察 C 编译器的优化策略对代码的影响。
7. 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。
8. 通过本次实验，希望大家明白：不同的编程语言是可以协同解决一个问题的，而且可以利用不同语言的特点来更好地解决问题；利用汇编语言的知识，能够更好地理解高级语言的内部处理原理与策略，为编写更好的 C 语言程序、用好 C 编译器提供支持。

III 实验过程

III.1 任务一

III.1.1 设计思想

本任务需要在实验一的基础上，讲程序的功能增加为录入学生信息、计算平均分、计算排名、输出成绩单，同时，将使用较为频繁的小程序段定义为宏。使用模块和宏有如下的优点：

- 将程序模块化可以使程序的条理和逻辑结构更加清晰。
- 模块可以多次调用，在需要重复调用某一模块时，可以减少编写代码的工作量。
- 在项目较大时，需要多人合作，而模块化的编程可以使不同的人负责不同的模块，分工明确。
- 使用模块化编程易于是整个项目增量式的进行，及时有的模块没有编写完成，也可以先测试已完成的模块，增加开发效率。
- 在调试的时候，先从总体框架的层面确定问题所在的模块，可以迅速缩小问题的范围，调试方便。

在学生成绩查找系统中，录入，计算平均分，计算排名，输出成绩单为互相独立的部分，均可以脱离其他部分单独工作，因此将之分为四个模块，并由一个功能菜单模块来选择这四个模块，

III. 实验过程

同时，由于输入和输出较为频繁，且程序长度较为简短，因此将之定义为宏，以便于在其他模块中使用。

在各个模块中，常常需要将姓名转换成列表中的序号（即查找某个名字），且经常需要输出姓名。但由于这两个程序段较长，因此将之定义为函数供各个模块使用。

除此之外，还需要考虑程序模块原子性的问题，如在进行录入的过程中，若对于一个表项先录入的数据（如姓名）正确，而后录入的数据（如成绩）出错，则不能将此项存入表中。因此需将模块录入部分与存储部分分开操作。

在编译方面，由于程序模块较多，每次手动输入命令较为麻烦，因此采用 nakefile 进行管理，nmake 语法与 GNU make 较为相似，因此可以参考 linux 下 C 语言 makefile 来编写本汇编程序的 makefile。

III.1.2 模块关系图及流程图

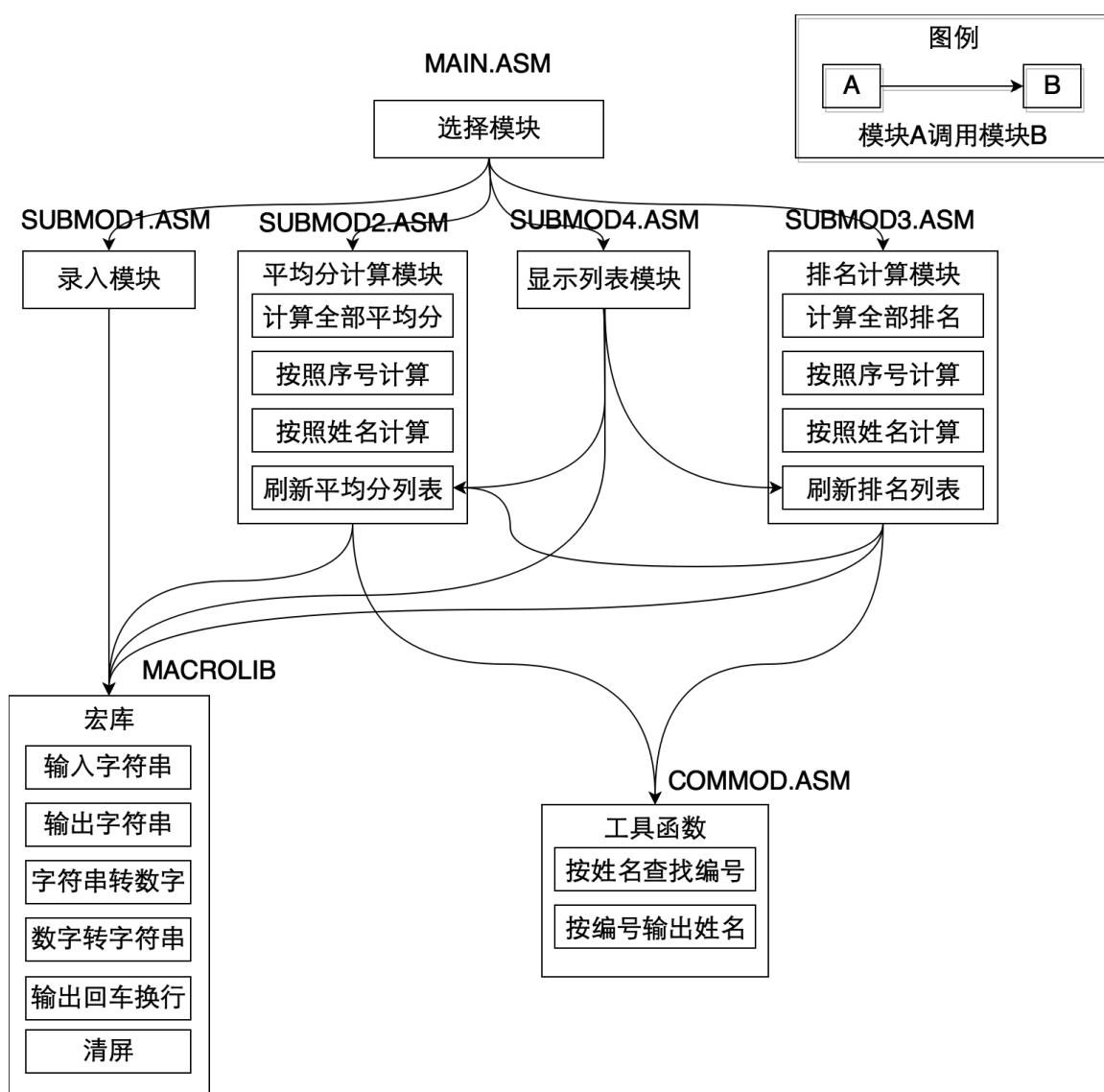


图 III-III-1-1: 学生成绩管理系统模块调用图

III. 实验过程

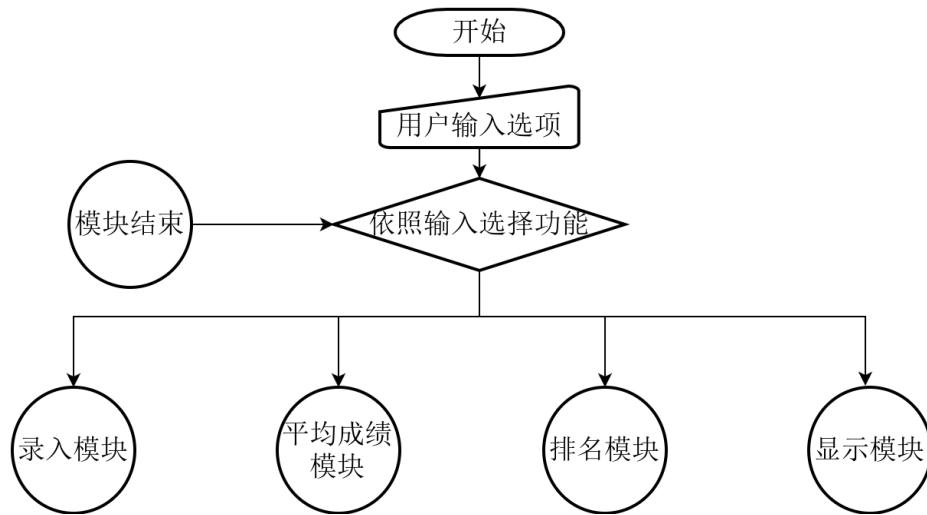


图 III-III-1-2: 学生成绩管理系统总体模块流程图

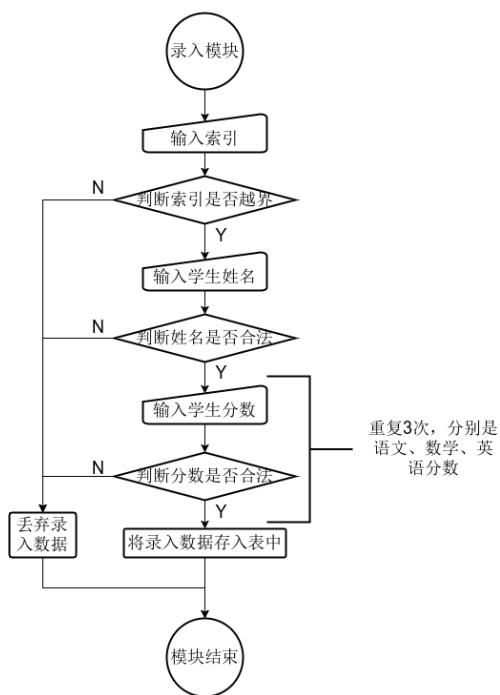


图 III-III-1-3: 录入模块流程图

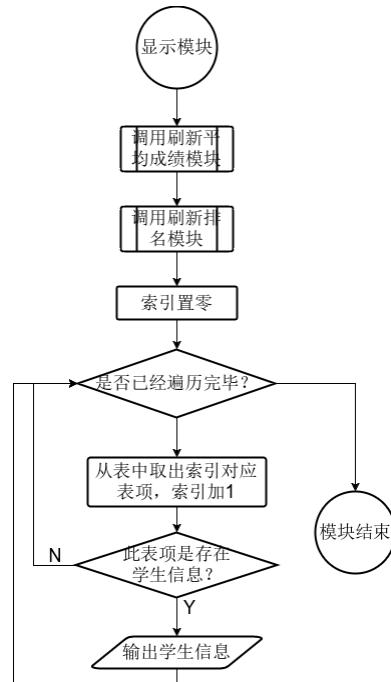


图 III-III-1-4: 显示输出模块流程图

III. 实验过程

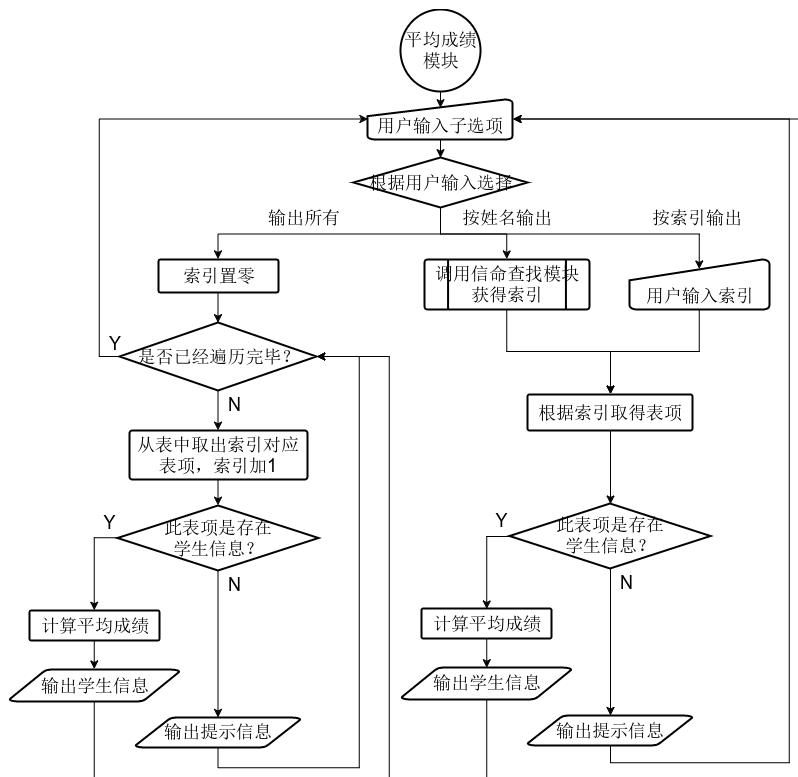


图 III-III-1-5: 平均分计算模块流程图

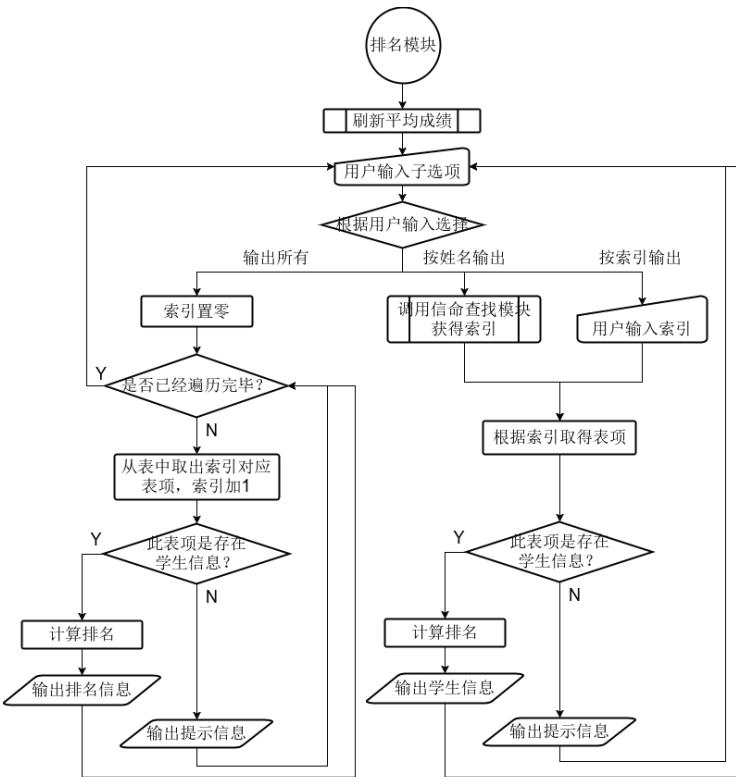


图 III-III-1-6: 排名计算模块流程图

III.1.3 源程序

源程序分为 8 个文件：

1. 主程序（菜单选择及模块调用）— MAIN.ASM
2. 录入模块 — SUBMOD1.ASM
3. 平均分计算模块 — SUBMOD2.ASM
4. 排名计算模块 — SUBMOD3.ASM
5. 成绩单输出模块 — SUBMOD4.ASM
6. 宏库 — MACROLIB
7. 工具函数模块 — COMMOD.ASM
8. nmake 文件 — MAKEFILE

由于源程序较长，为不影响排版，详细源程序请见附录部分。（第83页）

III.1.4 实验步骤

1. 按照模块编写程序，并分别编译各个模块。若编译出错，则分别修改每一个模块中的错误，直到所有模块都能够通过编译。

III. 实验过程

2. 链接各个模块，依照连接过程中可能出现的错误信息重新修改、编译、链接程序。
3. 直接运行程序，测试所有功能，观察有无错误产生。若有，则重复修改，并执行 nmake。
4. 在功能测试的过程中，需要加入可能出现的非法输入，测试程序在应对错误的输入时是否能正常处理。
5. 由于直接运行程序不能观察到程序的内部运行状况，打开调试器调试程序：
 - (a) 首先观察程序编译后的状况，包括不同文件的合并方式、段的位置、宏展开的情况、子程序调用传参等情况等。
 - (b) 在程序的关键位置（输入输出、子程序调用、宏展开等）打上断点，并运行程序，动态的观察在程序运行的过程中是如何处理程序的调用、外部符号等。

III.1.5 实验记录与分析

1. 实验环境条件：i7 3.6GHz，8G 内存；Archlinux 下 DOSBox0.74；VIM.EXE 7.3；TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件，使用 nmake1.20 管理编译与链接。
2. 按设计思想编写源文件，进行编译，并进行运行测试，根据所出现的错误重复修改、编译连接、运行的过程。在此过程中，主要出现过以下错误：
 - (a) 将 extrn 输入为 extern, TASM 不能正确识别。
 - (b) 在调用宏的时候参数为带有 offset 的两个单词，但是没有加上尖括号。
 - (c) 使用 masm 的进行编译时候 extern 符号后没有加入 byte、near、abs 等修饰符会被识别为错误，但使用 TASM 编译则不会。
 - (d) 将 equ 宏前面加上了 word ptr, 导致立即数被当成指针处理，进行了寻址
 - (e) 在进行录入操作的时候，没有清空上一次录入的缓冲区，导致录入不正确。
 - (f) 在进行字乘法运算过程中，由于 dx 存放结果的高位，但再次之前又将其作为计数器使用并且未进行压栈保护，因此出错。
3. 直接运行程序，程序正常运行，并弹出主菜单，如图III-III-1-7所示

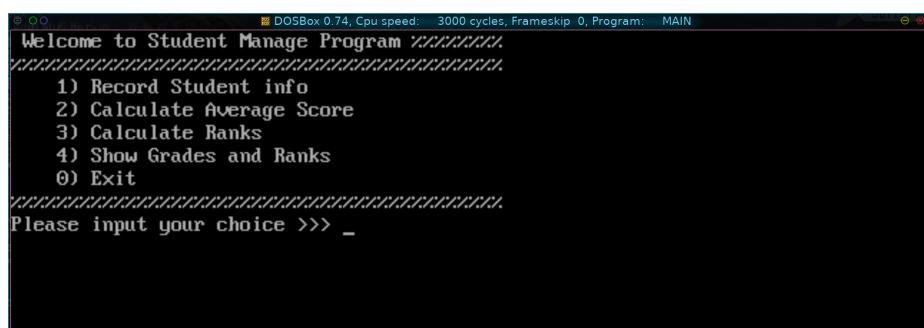


图 III-III-1-7: 程序运行主菜单

III. 实验过程

4. 录入部分学生及其成绩，学生及成绩如表III-III-1-1所示。

表 III-III-1-1: 录入学生编号、姓名、成绩

索引	姓名	语文成绩	数学成绩	英语成绩	预期平均分	预期排名
0	alpha	95	96	97	95.6	3
1	beta	80	90	100	85.7	4
2	gamma	100	95	100	98.6	1
3	delta	60	65	66	62.3	6
4	epsilon	50	97	86	68.6	5
6	Sixu Hu	98	99	88	96.9	2

5. 直接选择选项 4，显示模块会调用平均分计算模块和排名模块的子模块，因此相当于直接测试了三个模块的功能，输出结果如图III-III-1-8所示。可以看出，计算所得结果与预想一致。

Name	CHN	MAT	ENG	AUG	RANK
alpha	95	96	97	95	3
beta	80	90	100	85	4
gamma	100	95	100	98	1
delta	60	65	66	62	6
epsilon	50	97	86	68	5
--	-	-	-	-	--
Sixu Hu	98	99	88	96	2
--	-	-	-	-	--
--	-	-	-	-	--
--	-	-	-	-	--

图 III-III-1-8: 显示模块输出结果

6. 打开 Turbo Debugger，装载程序进行调试，可以观察到，各个模块间的链接方式就是按照 tlink 的顺序进行的，而程序段间会有少于 15 个字节的空间 (III-III-1-9)，以保证段的起始地址能被 16 整除，说明 TASM 默认选择 PARA 作为段的组合方式。

III. 实验过程

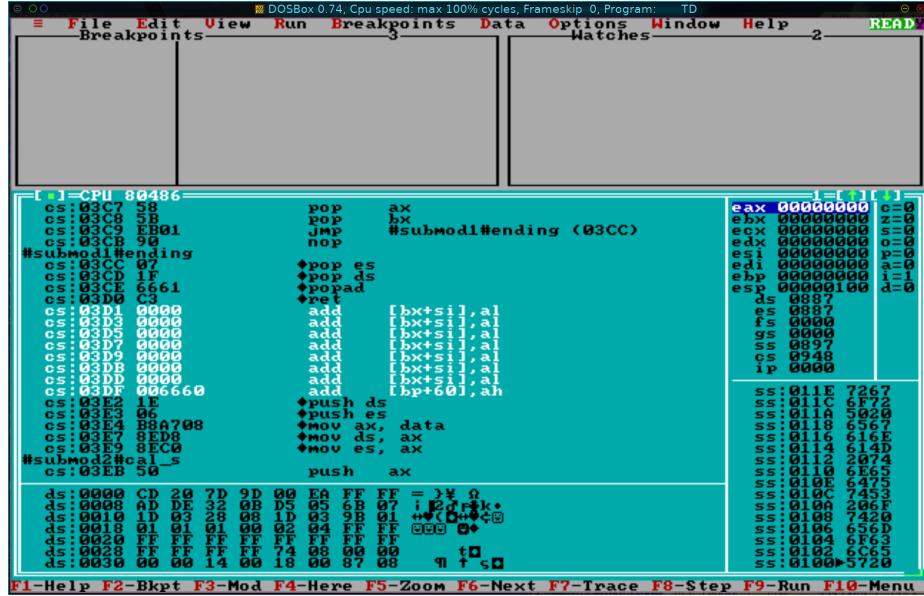


图 III-III-1-9: 段之间的空白部分

7. 对于不同段内的相同标号，连接器是这样处理的：在每个标号前加上 **# 段名 #**，由于段名不会重复，因此所有的标号也会是不同的。
8. 而对于内嵌于程序段中的宏，编译器是这样处理的：
 - 对于宏内的语句直接展开，对于宏内的参数，若不是局部符号，则直接替换。
 - 对于宏内的局部符号，在一个段内从 0 开始依次此编号，编号格式为 **# 段名 #?? 编号**，在一个段内，只要出现了包含有私有符号的宏就对此私有符号依次进行编号，通过特殊符号（问号）和全局的顺序编号来保证每一个标号的不同。
9. 在函数传参方面，程序编写时均使用的为堆栈传递参数，在子程序使用参数是要注意保护现场所使用的栈空间，以便于正确的取得参数。若需要返回的值所占用的空间大于传入的参数，则需要先分配好栈空间（通过重复压入需要保护的寄存器）再保护现场。在进行近调用时，存储 IP 消耗 2 字节，而进行远调用时，存储 CS:IP 占用 4 字节，在函数调用时也需注意。在本程序的测试过程中就因为忽略了原调用和近调用压栈数目不同而导致取得和返回了错误的参数。
10. 同时，我观察到使用调用时子程序的 ret 语句被翻译为 retf，而近调用则被翻译为 ret，这是由于返回时需要弹出不同数量的栈元素所造成的。

III.2 任务二

III.2.1 设计思想

除了将输入与输出部分改造为 C 语言实现外，其余部分均保留为汇编实现，这就需要汇编与 C 语言互相调用。其中，选择部分调用各个模块为 C 语言调用汇编，而汇编进行显示输出则为汇编调用 C 语言。

III. 实验过程

编写程序时，首先需要了解在相应的编译环境下的 C 语言与汇编互相调用的方法，然后将 MAIN.ASM 改为 MAIN.C 并用 C 语言实现，将 MACROLIB 也改为使用 C 语言实现。对于其余的文件进行必要的更改，以保证能够同时正常工作。

III.2.2 模块关系图

除了选择菜单与输入输出使用 C 语言实现外，其余部分均与任务一相同。具体见图 III-III-2-1

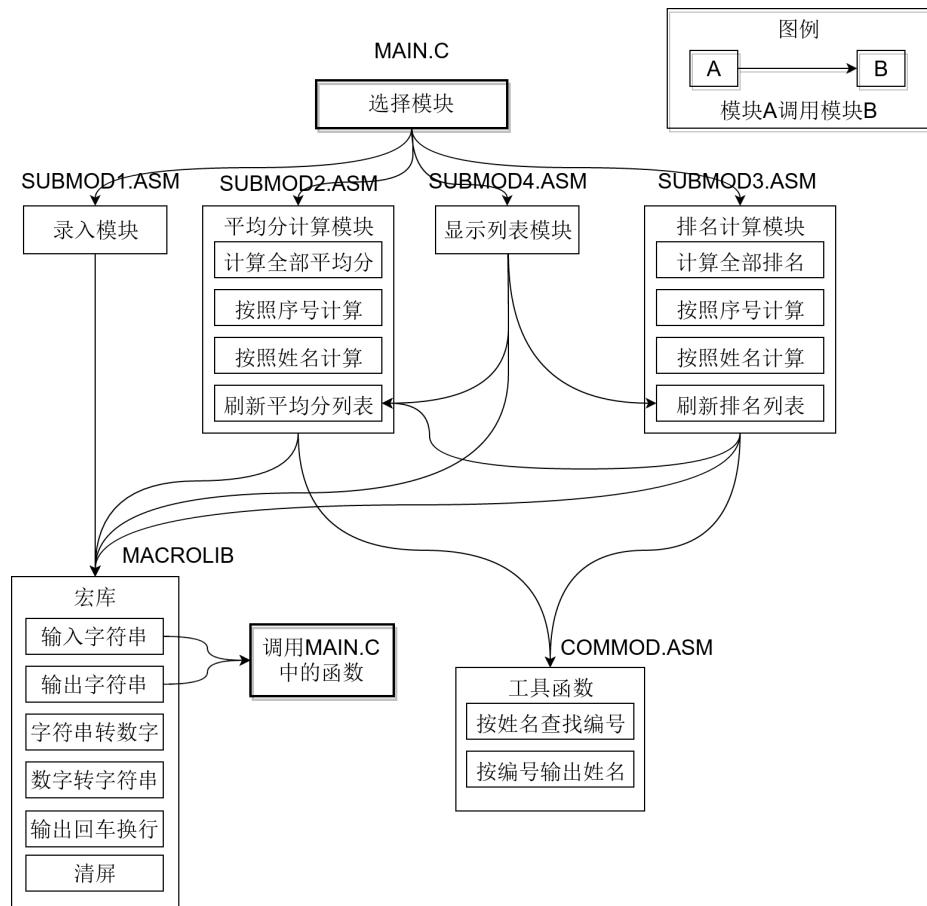


图 III-III-2-1: 替换后的模块关系调用图

III.2.3 源程序

源程序分为 8 个文件：

1. 主程序（菜单选择及模块调用）— MAIN.C
2. 录入模块 — SUBMOD1.ASM
3. 平均分计算模块 — SUBMOD2.ASM
4. 排名计算模块 — SUBMOD3.ASM

5. 成绩单输出模块 — SUBMOD4.ASM

6. 宏库 — MACROLIB

7. 工具函数模块 — COMMOD.ASM

8. nmake 文件 — MAKEFILE

其中录入模块，平均分计算模块，排名计算模块，成绩单输出模块，工具函数模块与任务一基本相同，除了段的属性与名字，子过程的名字有所更改外，其余部分不变，以 diff 的形式在附录中给出。而重新使用 C 语言编写的部分（主程序、宏库、MAKEFILE）则在附录（107）中完整给出。

III.2.4 实验步骤

1. 将任务一中的主函数与工具函数库更改为使用 C 语言实现，并分别编译各个模块。若编译出错，则分别修改每一个模块中的错误，直到所有模块都通过编译。
2. 链接各个模块，依照连接过程中可能出现的错误信息重新修改、编译、链接程序。
3. 依照任务一的方法对与程序进行测试，观察程序是否能够正常运行。
4. 由于直接执行程序不能看到编译器对于 C 语言与汇编语言的处理情况，因此使用 TD 进行调试，主要观察以下几项：
 - C 语言对于段符号，函数符号等符号的处理规则。
 - C 语言调用 C 语言函数的参数传递方式
 - C 语言函数调用的参数清理方式
 - 混合编译后段之间的关系
 - 汇编语言调用 C 语言和 C 语言调用汇编语言的细节

III.2.5 实验记录与分析

1. 实验环境条件：i7 3.6GHz，8G 内存；Archlinux 下 DOSBox0.74；VIM.EXE 7.3；直接使用 BCC3.0 进行编译与链接，TASM3.0 进行辅助汇编，使用 nmake1.20 管理编译与链接。
2. 在尝试编译的过程中，遇到了许多错误，其中主要的错误如下：
 - 使用 bcc 生成 main.obj, tasm 生成汇编 obj, bcc 进行链接：出现错误 undefined symbol _<symbol>，其中 <symbol> 是汇编过程名，未能正确识别汇编过程。
 - 使用 bcc 生成 main.obj, tasm 生成汇编 obj, tlink 进行链接：出现错误 undefined symbol _<symbol>，其中 <symbol> 是标准库中的函数名，如 printf 等，没有对标准库进行链接。
 - 使用 bcc 生成 main.asm, tasm main.obj 以及其他汇编 obj, tlink 链接：出现错误 c0.asm: undefined reference main，未能正确识别 C 语言的 main 函数入口。

III. 实验过程

- 使用 bcc 直接编译链接所有文件生成 exe，将汇编文件的 model 改为 C，汇编过程名前不加下划线，生成成功，但不能正确运行。反汇编表明生成的可执行文件对于汇编语言的翻译不正确。fixup overflow
 - 使用 bc.exe 建立 project，并添加 C 语言和汇编语言文件进行生成：出现错误 32-bit record in asm file，而在 bc 的链接器选项中没有对 32 位支持的选项，故不能正确识别如 eax 的 32 为寄存器名。
3. 最后通过 bcc -3 -v -c -S MAIN.C 生成 MAIN.ASM，并对 MAIN.ASM 进行研究后发现，在进行处理时，C 语言文件的函数体部分均在 _TEXT 段内，数据均在 _DATA 段内，并且函数名前都被加上下划线。因此对 asm 文件进行相似的处理，成功生成 exe 文件，并可以正常执行，具体进行的处理如下：
- 代码段的段定义改为 _TEXT segment use16 public "CODE"
 - 数据段的段定义改为 _DATA segment use16 public "DATA"
 - 在需要被 C 语言调用的过程名前加下划线
 - 在 C 语言调用汇编函数前先声明 extern int <procname>(<parameters>);
 - 在汇编调用 C 函数前先声明 extrn _<funcname>:near
 - 不需要在汇编文件前使用.MODEL 说明模式。
 - 直接使用 bcc 编译连接所有的文件，命令及参数为 bcc -3 -v -l3 -lv -Tla -e<output> <file_1>.c <file_2>.asm ... <file_n>.asm，其中 output 为输出文件名，<file_x>.c 和 <file_x>.asm 为需要编译的文件。
4. 在运行时又遇到部分问题，经过不断修改源代码最终可以正常运行并达到于任务一相同的效果，其中出现的问题如下：
- 反汇编表明，在 C 语言生成的代码中进入函数后会有 enter 指令，在函数返回前会有 leave 指令，它们都会影响栈空间的分配，其中 enter 会压入两个字节到栈中，leave 会从栈中弹出两个字节。
 - 在本编译条件下，C 语言调用函数时不会对寄存器做出保护，因此从汇编中调用 C 语言函数时要先对寄存器进行保护。
 - C 语言参数传递默认是从右向左压栈，并由调用者清理栈中的参数，也就是说，在本编译条件下，函数的调用默然采用 cdecl(C Declaration) 的形式。因此，在汇编调用 C 语言的过程中要注意清理栈中的参数。
 - C 语言的代码在显示输出是默认使用第 0 页，显示模式 03 作为显示输出¹
5. 最终程序可正常运行。按照任务一的方式对其进行测试，测试结果表明程序可以正常运行，其功能与任务一相同。
6. 打开 TD 进行调试如图III-III-2-2所示，可以看出，进入主函数前，有 enter 语句，且 C 语言的 getchar() 函数是由 fgetc 实现的。在需要传参的函数调用后均有 pop cx 语句，说明栈中的参数是由调用者进行清理的。函数名前均有下划线。

¹ INT 10H. (2017). En.wikipedia.org. Retrieved 9 April 2017, from https://en.wikipedia.org/wiki/INT_10H

IV. 总结与体会



图 III-III-2-2: 对程序进行调试

- 在汇编函数调用 C 函数的时候，需要模仿编译器对于 C 语言函数的处理：函数名前加下划线，并对调用后的栈空间进行清理。

IV 总结与体会

通过这次实验，我掌握了子程序设计的方法与技巧，宏指令、模块化程序的设计方法，较大规模程序的合作开发与调试方法，以及 C 语言与汇编语言混合编译的一些细节问题。

通过实验一，我深刻的体会到了将一个程序分为多个模块的优越性。他不仅使程序编写的逻辑更为清晰，调试更为方便，可以快速缩小调试范围，而且模块化的程序编写更适合多人合作开发，加速了开发的过程。此外，通过这一次实验，我了解到了模块化汇编中的一些细节问题。如在模块间的程序调用时，需要考虑参数的存放位置，特别是在堆栈中存放参数的时候，需要考虑子程序类型（near、far）对参数位置的影响。此外我还观察到编译器对于一些符号的处理，比如不同模块中的同名符号和宏汇编中的局部变量等等，汇编器的处理使他们不会产生冲突，但是需要对于外部符号进行引用的时候，需要先声明才可以引用，否则编译器会产生错误。

使用宏汇编和子过程时，我将其进行了比较，发现他们有相同之处，即都提高了代码的复用率，但是也有其各自的优缺点：宏汇编虽然使用较为简单，参数的传递方式也更为清晰，但是编译后会在每一处展开，不仅占用了空间，如果频繁使用，在调试时也会影响可读性；而子过程则较为节省空间，但需要考虑段转移带来的影响以及参数传递的方式，使用堆栈进行传参和返回时需要仔细的计算，否则参数的位置容易出错。

而实验二使我明白了不同语言是可以协同解决同一个问题的，并且正确的混合使用不同的语言可以利用到每个语言的优势。

在实验二刚开始进行的时候我面对了不小的挑战：不论怎样替换操作系统、编译工具和源

IV. 总结与体会

代码总是不能编译成功，或者变异成功后不能得到正确的可执行文件。但当我将 C 语言文件用编译器生成汇编病仔细的分析过后终于发现了问题的所在和解决的方法，同时也对编译器的行为有了更深的了解。而在能够正确编译后，为了使程序正常运行，我又岁程序进行了漫长的调试。在这个过程中，我对混合调试以及 C 语言和汇编之间的关系有了更深一层的认识。

总体而言，这次实验使我收获颇丰，在实验中所学到的知识可能会对未来将要进行的程序优化以及调试起到帮助。

实验四 中断与反跟踪

I 实验目的与要求

1. 掌握中断矢量表的概念；
2. 熟悉 I/O 访问， BIOS 功能调用方法；
3. 掌握实方式下中断处理程序的编制与调试方法；
4. 熟悉跟踪与反跟踪的技术；
5. 提升对计算机系统的理解与分析能力。

II 实验内容

II.1 任务一：获取中断入口地址

用三种方式获取中断类型码 21H 对应的中断处理程序的入口地址。要求：

首先要进入虚拟机状态，然后：

1. 直接运行调试工具 (TD.EXE)，观察中断矢量表中的信息。
2. 编写程序，用 DOS 系统功能调用方式获取，观察功能调用相应的出口参数与 “(1)” 看到的结果是否相同 (使用 TD 观看出口参数即可)。
3. 编写程序，直接读取相应内存单元，观察读到的数据与 “(1)” 看到的结果是否相同 (使用 TD 观看程序的执行结果即可)。

II.2 任务二：接管键盘中断

编写一个接管键盘中断的中断服务程序并驻留内存，要求在程序返回 DOS 操作系统后，键盘上的小写字母都变成了大写字母。

要求：

1. 在 DOS 虚拟机或 DOS 窗口下执行程序，中断服务程序驻留内存。
2. 在 DOS 命令行下键入小写字母，屏幕显示为大写，键入大写时不变。执行 TD，在代码区输入指令 “`mov AX,0`” 看是否能发生变化。
3. 选作：另外编写一个中断服务程序的卸载程序，将键盘中断服务程序恢复到原来的状态 (也就是还原中断矢量表的信息，先前驻留的程序可以不退出内存)。

II.3 任务三：读取 CMOS 信息

读取 CMOS 内指定单元的信息，按照 16 进制形式显示在屏幕上。

要求：

- 先输入待读取的 CMOS 内部单元的地址编号（可以只处理编号小于 10 的地址单元）。再使用 IN/OUT 指令，读取 CMOS 内的指定单元的信息。
- 将读取的信息用 16 进制的形式显示在屏幕上。若是时间信息，可以人工判断一下是否正确。

II.4 任务四：数据加密与反跟踪

在实验一的学生成绩查询程序的基础上，增加查询前输入密码的功能，密码不对则程序退出，只有密码正确之后才能完成后续的功能。密码采用密文的方式存放在数据段中。各科成绩也以密文方式存放在数据段中。加密方法自选。

可以采用计时、中断矢量表检查、堆栈检查、间接寻址等方式中的一种或多种方式反跟踪（建议只采用一到两种反跟踪方法，重点是深入理解和运用好所选择的反跟踪方法）。

成绩表中要有编程者自己的名字（姓的全拼 + 名字的拼音首字母，但大小写可以随意组合）和各科成绩（姓名和成绩都密文存放）。成绩表中只需要定义三个学生的信息即可。**提示：**为了使源程序的数据段中定义的密码、学生姓名、各科成绩能在汇编之后变成密文（也就是在最后交付出去的执行程序中看不到明文），可以使用数值运算符（参见教材 P48）对变量的初始值进行变换。例如，如果想使语文成绩 90 分变成密文，加密算法是与密钥字符“W”做异或运算，则可写成：

```
1 YUWEN      DB  90  XOR   'W
```

II.5 任务五：跟踪与数据解密

解密同组同学的加密程序，获取该同学的各科成绩。

注意：两人一组，每人实现一套自己选择的加密与反跟踪方法，把执行程序交给对方解密（解密时间超过半小时的，说明反跟踪方法基本有效）。如何设计反跟踪程序以及如何跟踪破解，是本次实验报告中重点需要突出的内容。（分组可以按照学号顺序依次构成两人一组。也可以自行调整。如果班上人数是奇数，则三人一组，甲解密乙的，乙解密丙的，丙解密甲的）

III 实验过程

III.1 任务一

III.1.1 实验步骤

- 首先使用第一种方法获得中断入口地址：打开 TD 直接观察，将 int 21H 的地址记录下来。
- 编写程序，在程序中首先使用系统功能调用的方法，再使用直接读取的方法获得地址。

III. 实验过程

3. 编译、链接、调试程序。
4. 观察并记录程序的运行结果，并与第一种方法获得的结果进行比较。

III.1.2 源程序

使用第二种和第三种方法获得中断入口地址的源程序见附录部分 (第118页)

III.1.3 实验记录与分析

1. 实验环境条件: i7 3.6GHz, 8G 内存; Archlinux 下 DOSBox0.74; VIM.EXE 7.3; TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。
2. 由 CPU 响应中断的过程可知, int 21H 的地址应该存储于 $0x21 \times 4 = 132(0x84)$ 处, 并占用 4 个字节, 因此在 TD 中切换到数据显示区, 按下 Ctrl+G 并输入 00:84 使光标跳转到对应位置, 并读出 int 21H 的地址为 IP:01CC CS:031D, 如图IV-III-1-1所示。

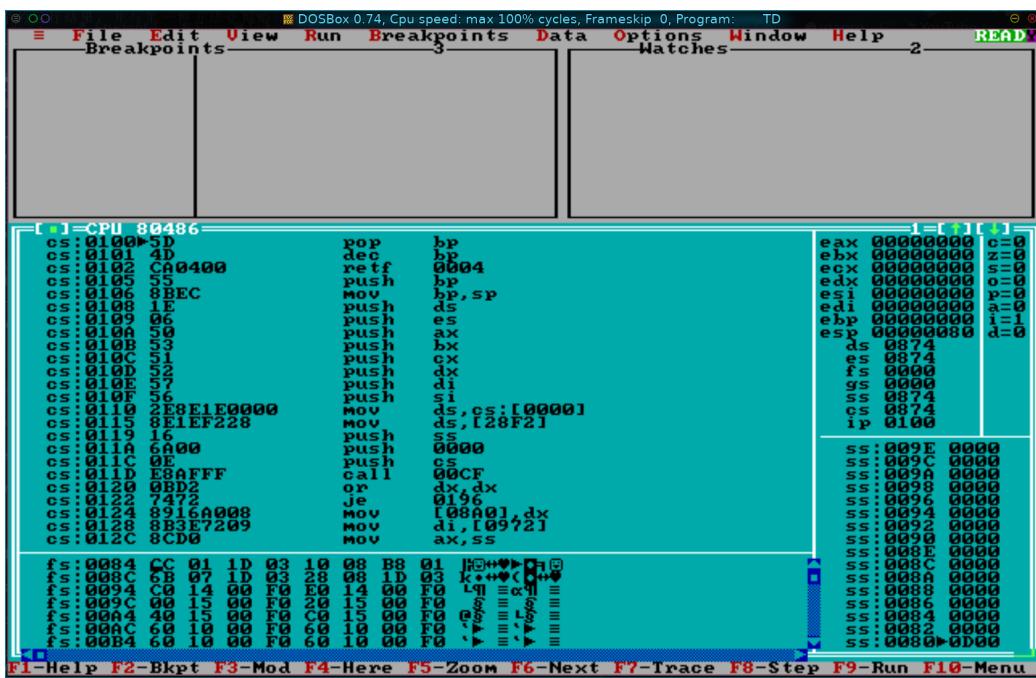


图 IV-III-1-1: 从 TD 中读取的 int 21H 地址

3. 编译链接程序, 程序并未出错, 直接运行程序, 结果如图IV-III-1-2所示, IP=5280=14A0H, CS=61440=F000H, 结果发现, 与在 TD 中调试的结果并不相同。

III. 实验过程

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\PLAYGROUN\WORKSET4>cd PROBLEM1

C:\PLAYGROUN\WORKSET4\PROBLEM1>a GETADDR.ASM
C:\PLAYGROUN\WORKSET4\PROBLEM1>TASM.EXE /c /z /zi /la GETADDR.ASM
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file: GETADDR.ASM
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 459k

C:\PLAYGROUN\WORKSET4\PROBLEM1>l GETADDR.OBJ
C:\PLAYGROUN\WORKSET4\PROBLEM1>TLINK /Tde /3 /v GETADDR.OBJ
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International

C:\PLAYGROUN\WORKSET4\PROBLEM1>GETADDR.EXE
5280
61440
5280
61440

C:\PLAYGROUN\WORKSET4\PROBLEM1>

```

图 IV-III-1-2: 直接运行程序结果

4. 换用 int20 进行实验，在 td 中调试的结果如图IV-III-1-3所示，其中 IP=1480，CS=F000。

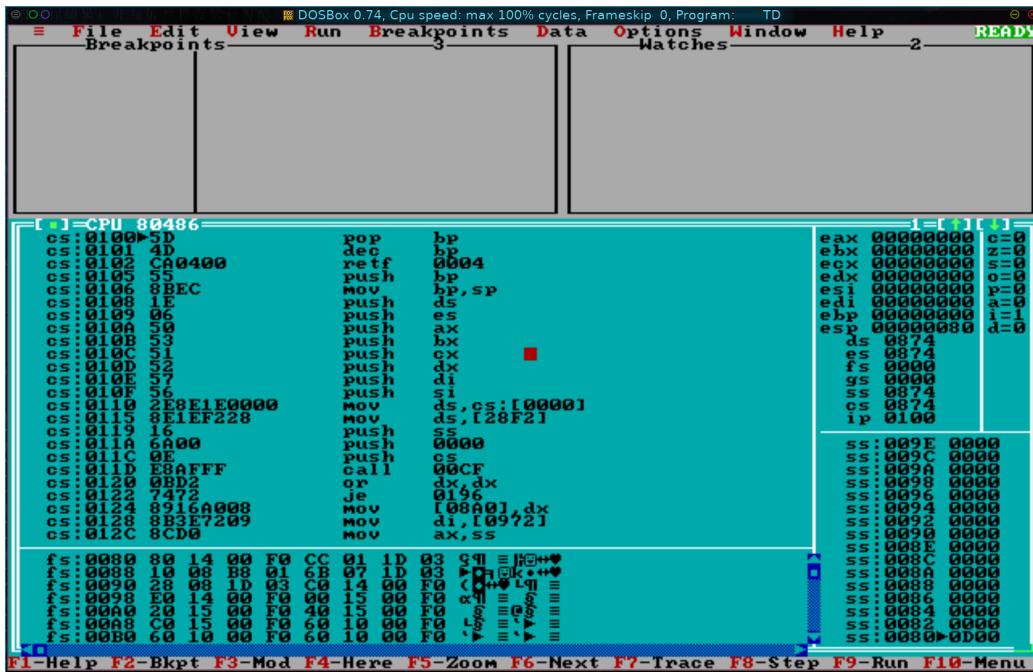
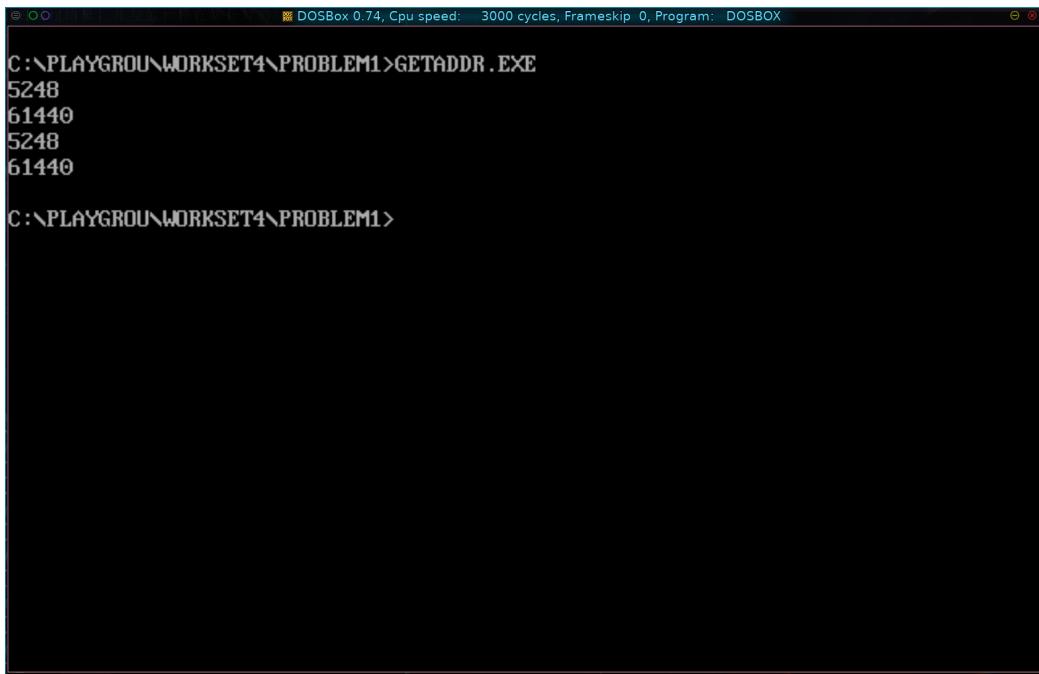


图 IV-III-1-3: 从 TD 中读取的 int 20H 地址

5. 修改源程序,使之显示 int 20h 的入口地址,结果如图IV-III-1-4所示,其中 IP=5248=1480H, CS=61440=F000H, 与 TD 中结果一致。

III. 实验过程



The screenshot shows a DOSBox window with the title bar "DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX". The command prompt is at the top: "C:\PLAYGROU\WORKSET4\PROBLEM1>GETADDR.EXE". The output of the program is displayed below, showing two sets of values: "5248" and "61440", each appearing twice. The final prompt is "C:\PLAYGROU\WORKSET4\PROBLEM1>".

图 IV-III-1-4: 直接运行程序结果

6. 由此可知，TD 接管了 int 21h 中断，但是未接管 int 20h 中断，使用 20h 号中断验证结果依然正确。

III.2 任务二

III.2.1 设计思想

通过将原有的中断替换为新编写的中断，并将原有中断存储起来以供恢复，即可完成中断的接管。在设计新的中断程序时，对于原有的中断不需要修改的部分可以直接调用原有的中断，而对于需要修改的部分可以调用后对结果进行修改，也可以直接编写一个新的中断。

III.3 中断地址

图IV-III-3-1是修改中断时的地址关系，将原来的 16h 中断保留在 60h 中断处，然后在 16h 处加入自己的中断。而图IV-III-3-2是将原始的中断恢复的过程，直接将保留的 60h 处的原始中断地址移动到 16h 处。

III. 实验过程

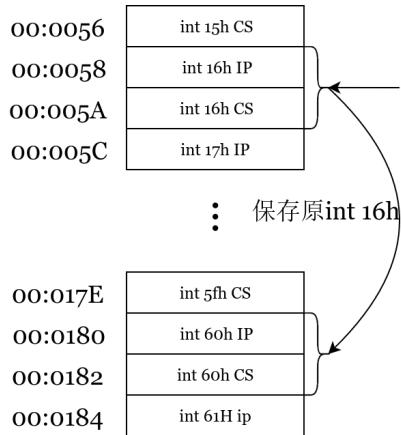


图 IV-III-3-1: 修改中断地址

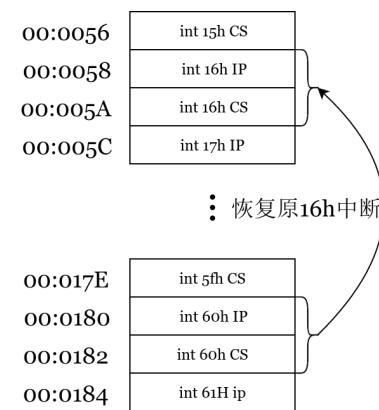


图 IV-III-3-2: 还原中断地址

III.3.1 源程序

设置中断与恢复中断程序见附录部分 (第II.4.2页)

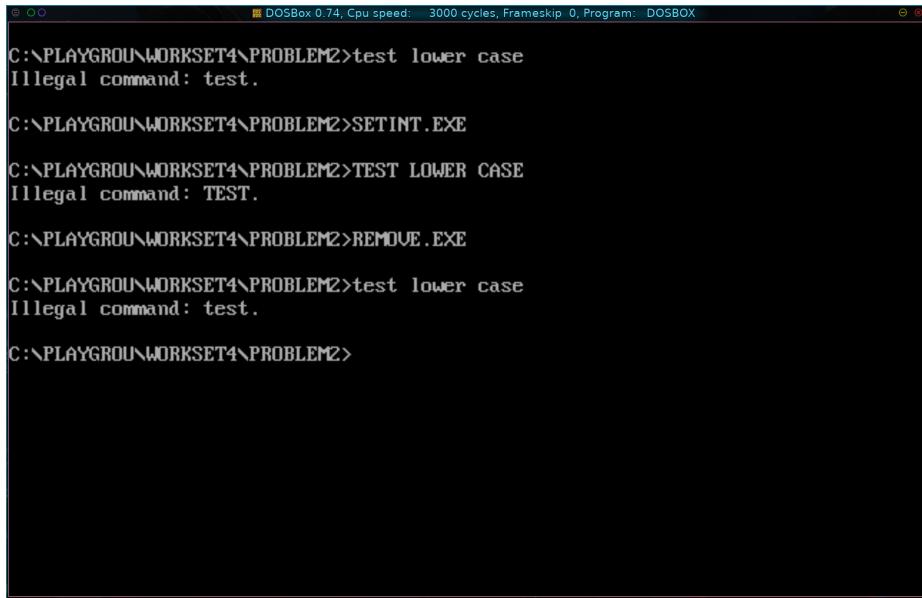
III.3.2 实验步骤

1. 按照设计思想编写源程序，注意在设置中断的过程中需要关中断。
2. 编译、链接，并重复修改源程序，直到没有错误。
3. 直接运行编译结果 SETINT.EXE，观察输入的小写字母是否变为了大写。
4. 运行编译结果 REMOVE.EXE，观察中断是否被还原。
5. 打开 TD 调试，观察中断被替换的过程。

III.3.3 实验记录与分析

- 实验环境条件: i7 3.6GHz, 8G 内存; Archlinux 下 DOSBox0.74; VIM.EXE 7.3; TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。
- 在编译与链接的过程中没有出错，但在运行时出现了输入无响应的错误。
- 经过对源代码的观察发现，进入中断时使用了 PUSHAD，而退出中断前使用了 POPAD，导致中断的出口参数丢失。
- 将源程序修改后重新编译、连接、运行。程序返回 DOS 后，发现输入的小写字母均变为了大写字母。
- 运行 REMOVE.EXE 再次输入字母测试，发现输入的字母又还原为小写字母，两次运行的结果如图IV-III-3-3所示。

III. 实验过程



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

```
C:\PLAYGROU\WORKSET4\PROBLEM2>test lower case
Illegal command: test.

C:\PLAYGROU\WORKSET4\PROBLEM2>SETINT.EXE

C:\PLAYGROU\WORKSET4\PROBLEM2>TEST LOWER CASE
Illegal command: TEST.

C:\PLAYGROU\WORKSET4\PROBLEM2>REMOVE.EXE

C:\PLAYGROU\WORKSET4\PROBLEM2>test lower case
Illegal command: test.

C:\PLAYGROU\WORKSET4\PROBLEM2>
```

图 IV-III-3-3: 接管中断与恢复中断

- 打开 TD，对于源程序进行调试，发现原来的 16h 中断地址被顺利的 00:0180h 处，但是 00:0058h 的值在设置时没有发生变化，在 TD 中的输入也没有变成大写。猜想 TD 为了保证在调试时键盘输入不至于被破坏，接管了 16h 中断或对此中断入口进行了保护。

III.4 任务三

III.4.1 设计思想

直接使用 in 和 out 指令，对于 CMOS 进行操作。在 70h 端口写，71h 端口读出数据。在程序的编写过程中可以调用实验 2 宏库中的输入输出及字符串与数字转换的部分。

III.4.2 源程序

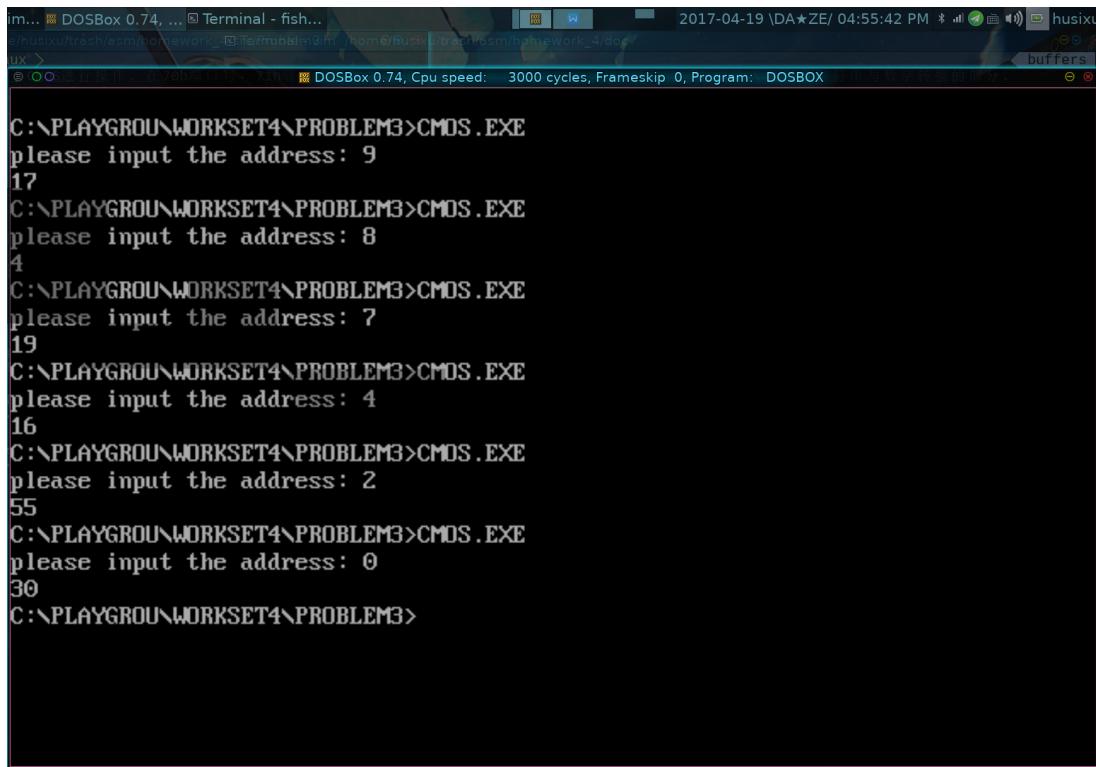
CMOS 操作源程序见附录部分 (第122页)，其中 MACROLB 与实验二任务一的 MACROLIB 相同。

III.4.3 实验步骤

- 编写源程序，请求用户输入，将输入转换为数字后使用 in/out 指令通过 70h/71h 端口与 CMOS 芯片进行交流，并将输出转换为 16 进制文本输出。
- 编译、链接，并重复修改源程序，直到没有错误。
- 直接运行源程序，读取年、月、日、时、分、秒等信息，并与计算机信息进行对比，观察是否正确。

III.4.4 实验记录与分析

- 实验环境条件: i7 3.6GHz, 8G 内存; Archlinux 下 DOSBox0.74; VIM.EXE 7.3; TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。
- 编写源程序, 将 in 所获得的值转换为十进制输出, 并进行编译、链接。在此过程中未出现错误。
- 连接运行程序, 观察到所获得的时间值与计算机上显示的时间值并不一致, 而是将计算机显示时间当做 16 进制数, 转换为 10 进制的结果。
- 修改源程序, 将所获得的数字以 16 进制文本呢的格式进行输出, 观察到与计算机显示一致, 如图IV-III-4-1所示 (右上角为系统时间)



```
im... DOSBox 0.74, ... Terminal - fish...
2017-04-19 \DA★ZE/ 04:55:42 PM * all buffers
husixu@husixu:~/.trash/asm/linework$ file /tmp/vim1.vim /home/Husixu/Desktop/asm/linework/4/doc/
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\PLAYGROUP\WORKSET4\PROBLEM3>CMOS.EXE
please input the address: 9
17
C:\PLAYGROUP\WORKSET4\PROBLEM3>CMOS.EXE
please input the address: 8
4
C:\PLAYGROUP\WORKSET4\PROBLEM3>CMOS.EXE
please input the address: 7
19
C:\PLAYGROUP\WORKSET4\PROBLEM3>CMOS.EXE
please input the address: 4
16
C:\PLAYGROUP\WORKSET4\PROBLEM3>CMOS.EXE
please input the address: 2
55
C:\PLAYGROUP\WORKSET4\PROBLEM3>CMOS.EXE
please input the address: 0
30
C:\PLAYGROUP\WORKSET4\PROBLEM3>
```

图 IV-III-4-1: 读取 CMOS 显示时间

- 从结论可以看出, CMOS 中的时间存储的数值为 $\frac{time}{10} \times 16 + (time \% 10)$, 其中 time 为时间数值 (年、月、日等), 直接转换十六进制文本后即为当前时间。

III.5 任务四

III.5.1 设计思想

使用简单的加密方式对于姓名, 分数以及需要输入的密码进行加密, 在此基础上使用较为复杂的反跟踪方式进行混淆, 从而使加密的步骤难以被识别, 达到保护数据的效果。对于本程

III. 实验过程

序而言，采用如图IV-III-5-1的简单加密方法为逐字节异或，由于采用的是简单的双边密码形式，秘钥就是加密算法，因此加密算法不应被泄露。本程序采用的反跟踪方法为堆栈检查和多重跳转。

在进行程序编写的时候，将明文计算为密文后存储在数据区中，而不直接存储明文。在整个程序的运行过程中，应尽量使明文在内存中出现的时间减少，避免程序被破解，同事配合反跟踪部分，使程序的运行过程更不容易被解读。

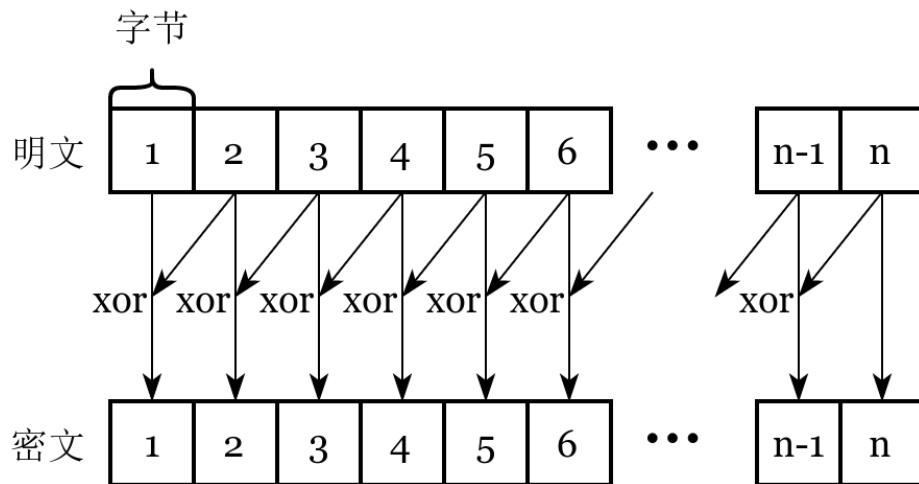


图 IV-III-5-1: 数据加密方式

III.5.2 源程序

由于源程序较长，本部分的源程序见附录部分。(第123页)

III.5.3 实验步骤

1. 对于实验三任务一中的源程序进行修改，在录入与显示部分对于数据进行加密与解密。
2. 采用堆栈法和间接跳转法对于整个程序进行反跟踪处理。
3. 对源程序进行编译和链接。若出现错误，对于整个程序进行修改并重新编译和链接。
4. 对于源程序进行调试，确保反跟踪部分和加密部分有效。
5. 重新对源程序进行编译和链接，此时**不生成调试信息**，以增加反跟踪难度。
6. 交付源程序。

III.5.4 实验记录与分析

1. 实验环境条件：i7 3.6GHz, 8G 内存；Archlinux 下 DOSBox0.74; VIM.EXE 7.3; TASM 5.0、TLINK 5.0、TD 5.0 编译、链接、调试套件。
2. 首先按照设计思想中的加密方法对源程序追加加密部分，同时，将预存的数据按照此加密方法进行加密，并以密文的形式写入汇编文件中。

III. 实验过程

3. 对于源程序进行编译、链接、调试，发现在运行过程中出现错误，具体原因为加密和解密算法不匹配。
4. 重新对于文件进行编译和调试，最后达到如图IV-III-5-2的效果，即当密码输入错误时程序直接退出。

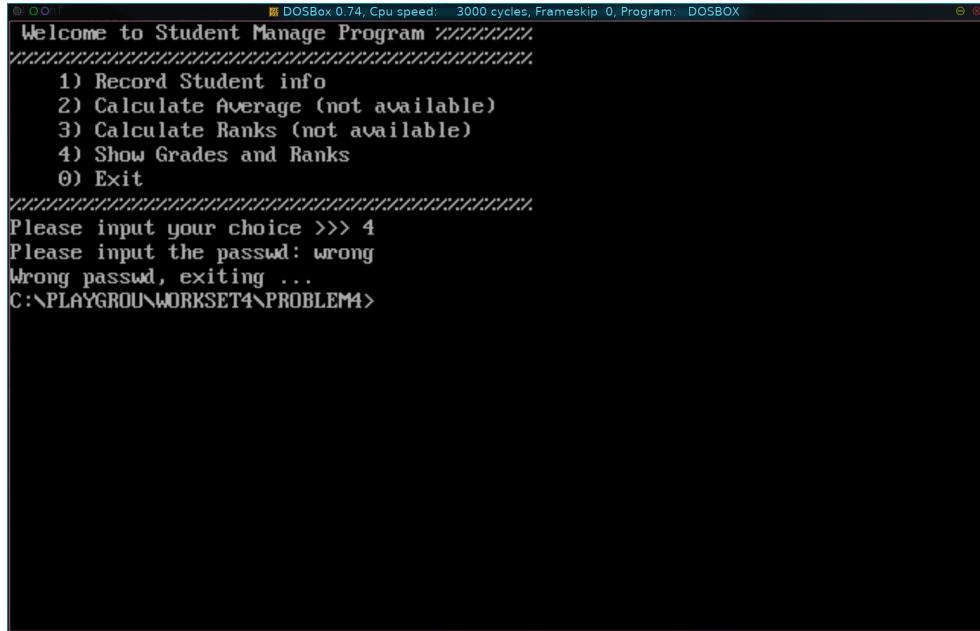


图 IV-III-5-2: 将程序的查找部分进行加密

5. 加密部分编写完成后，将源程序进行更改，追加反跟踪部分，并进行测试。
6. 在对于反跟踪部分的编写过程中，出现了由于标号太远超出跳转范围的错误。将其改为远标号重新进行编译后即可正常运行。
7. 对于程序进行调试，发现在堆栈检查的部分由于调试程序修改了堆栈，程序没有正常向下执行，而是跳转到了别的地方。说明堆栈检反跟踪是成功的。
8. 重新对源程序进行编译，此时关闭调试信息开关（即在编译时仅使用 TASM /z /zi 进行编译，在链接时仅使用 TLINK /3 /Tde 进行链接）。
9. 再次打开调试器，观察到所有的调试信息均已消失，如图IV-III-5-3所示，至此，程序处理完成，可以进行交付。

III. 实验过程

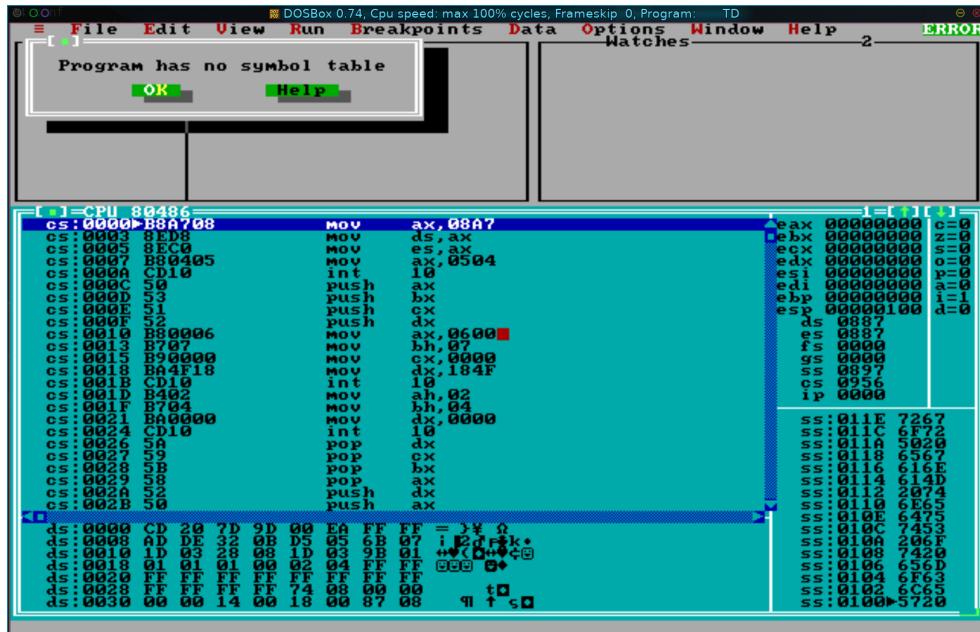


图 IV-III-5-3: 除去调试信息后的调试界面

III.6 任务五

III.6.1 实验思想

对于程序的破解，主要从以下几个方面进行：

- 直接运行程序，观察程序的输出以及程序的关键性位置，以找到破解的入口。
- 对于密码输入的位置应该重点观察，尝试直接读出加密的方式。
- 在成功获得密码之后直接运行程序，尝试直接从程序中获得所需要的信息，若不能获得所需要的信息，重新观察关键位置并破解。
- 对于反跟踪的部分需要首先识别出其位置，然后对其进行跳过。

III.6.2 实验步骤

1. 直接运行程序，观察程序输出以及请求密码的位置。
2. 打开 TD 进行调试，跳转到密码请求的位置。任意输入一个密码，并观察程序的运行状态。
3. 对于加密的部分进行选择性跳过。
4. 重复2到3之间的步骤，直到所有的混淆性程序均已经被跳过为止。
5. 根据所获得的加密算法推算出密码，或者直接取得数据。
6. 直接运行程序，输入密码，并尝试获得数据字段。
7. 若通过直接运行程序不能获得数据字段，重新调试并尝试从内存数据中推算出数据字段。

III.6.3 实验记录与分析

1. 实验环境条件: i7 3.6GHz, 8G 内存; Archlinux 下 DOSBox0.74; 使用 TD5.0 作为调试工具, xdotool 作为自动化输入工具。
2. 首先直接运行程序, 发现要求输入密码。任意输入一个错误的密码, 程序退出。过程如图IV-III-6-1所示。

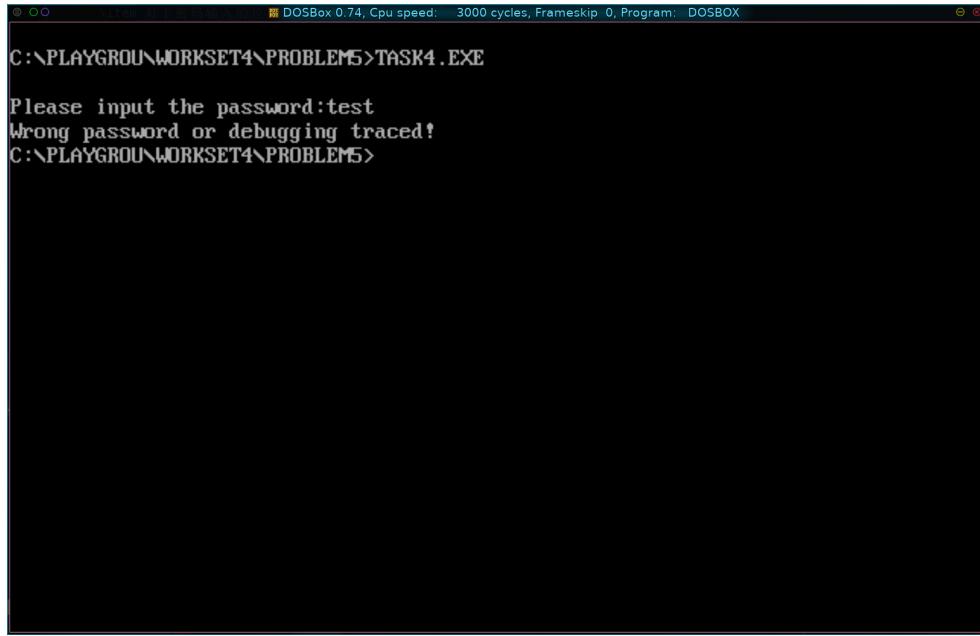


图 IV-III-6-1: 直接运行程序

3. 打开 TD 直接单步执行程序, 发现在执行到特定语句后, 程序直接跳转然后结束, 对程序进行观察, 发现一部分的程序进行了时间检查, 如图IV-III-6-2所示。

III. 实验过程

The screenshot shows the DOSBox interface with the CPU window active. The assembly code is as follows:

```

CPU 80486
[...]
cs:0037 FA cli
cs:0038 B42C MOV ah, 2C
cs:003A CD21 int 21
cs:003D A6ED01 MOV al, [01ED]
cs:0040 4290 push dx
cs:0042 2E20061300 sub al, ss:[0013]
cs:0047 0FBED8 movsx bx, al
cs:004A 81C3FA01 add bx, 01FA
cs:004E B42C mov ah, 2C
cs:0050 CD21 int 21
cs:0052 FB sti
cs:0053 6A3B1424 cmp dx, [esp]
cs:0058 7403 je 005D
cs:005A BBFC01 mov bx, 01FC
cs:005D 8BF1 mov bx, [bx]
cs:005F 2EB13FB90B cmp cs:[word ptr [bx]], 0BB9
cs:0064 7404 je 006A
cs:0066 FF26FC01 jmp [01FC]

ds:0000 CD 20 7D 9D 00 EA FF FF = ??
ds:0008 AD DE 32 0B D5 05 6B 07 i 2d F k*
ds:0010 1D 03 28 0B 1D 03 9B 01 +* C O P *#
ds:0018 01 01 01 00 02 04 FF FF 0000 00
ds:0020 FF FF FF FF FF FF 0000 00
ds:0028 FF FF FF 74 00 00 00 00 q t#
ds:0030 00 00 14 00 18 00 87 00 q t#

```

The CPU window shows assembly code for time checking. The registers and stack are also visible.

图 IV-III-6-2: 时间检查部分的反汇编代码

4. 跳过这些程序后继续执行，发现程序依然在执行某间接跳转后程序直接停止。
5. 将上述两个步骤的代码用 nop 填充，并重新执行程序，将其他干扰性代码用 nop 填充，直到执行到显示主界面位置。显示主界面如图IV-III-6-3所示。发现有检查输入的姓名是否存在的功能。又已知姓名的格式为姓的全拼加上名的简拼，于是将‘chenzh’字符串的所有大小写组合使用脚本进行自动输入测试，并观察检测所得的结果，发现最终的正确姓名为‘ChenZh’。

The screenshot shows the DOSBox interface with the CPU window active. The assembly code is as follows:

```

CPU 80486
[...]
cs:0037 FA cli
cs:0038 B42C MOV ah, 2C
cs:003A CD21 int 21
cs:003D A6ED01 MOV al, [01ED]
cs:0040 4290 push dx
cs:0042 2E20061300 sub al, ss:[0013]
cs:0047 0FBED8 movsx bx, al
cs:004A 81C3FA01 add bx, 01FA
cs:004E B42C mov ah, 2C
cs:0050 CD21 int 21
cs:0052 FB sti
cs:0053 6A3B1424 cmp dx, [esp]
cs:0058 7403 je 005D
cs:005A BBFC01 mov bx, 01FC
cs:005D 8BF1 mov bx, [bx]
cs:005F 2EB13FB90B cmp cs:[word ptr [bx]], 0BB9
cs:0064 7404 je 006A
cs:0066 FF26FC01 jmp [01FC]

ds:0000 CD 20 7D 9D 00 EA FF FF = ??
ds:0008 AD DE 32 0B D5 05 6B 07 i 2d F k*
ds:0010 1D 03 28 0B 1D 03 9B 01 +* C O P *#
ds:0018 01 01 01 00 02 04 FF FF 0000 00
ds:0020 FF FF FF FF FF FF 0000 00
ds:0028 FF FF FF 74 00 00 00 00 q t#
ds:0030 00 00 14 00 18 00 87 00 q t#

```

The CPU window shows assembly code for time checking. The registers and stack are also visible.

图 IV-III-6-3: 跳过密码检查后的程序主界面

6. 输入姓名后，得知其分数评价为 A，但是依然不知道其平均分。对于计算分数评价的函数进行定位，加上断点后重新进行调试，单步执行并分析代码，获得分数评价算法后直接在计算的过程中从寄存器中读出其平均分为 5AH，即 90 分，图IV-III-6-4中 bl 字段即为其平均分，高亮的代码部分为进行分数评价的代码。

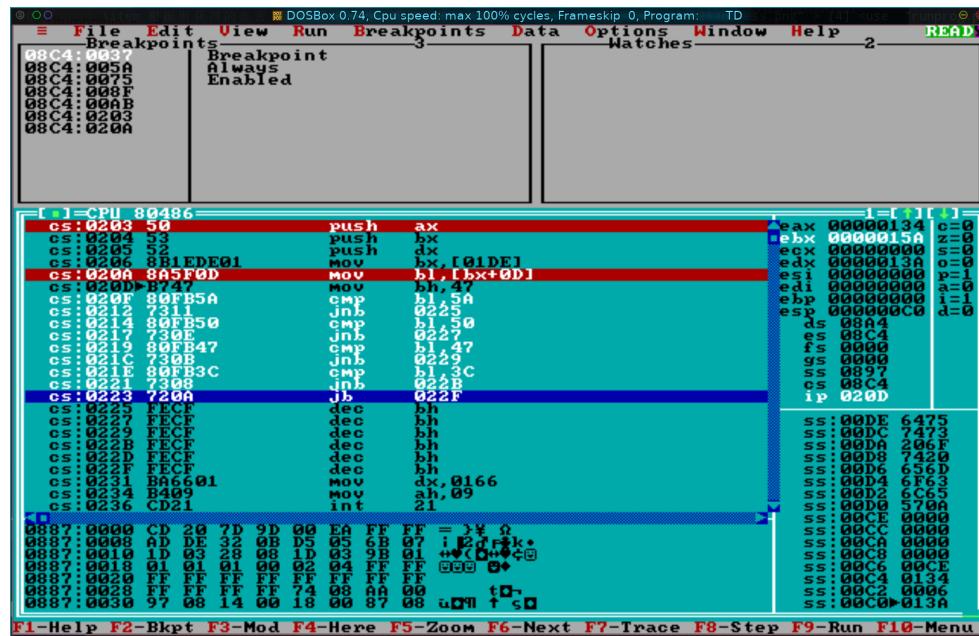


图 IV-III-6-4: 取得最终成绩

IV 总结与体会

通过这次实验，我对于汇编语言有了更为深刻的认识。在前三个实验中，我不见了解到汇编中断的工作原理以及对于硬件的读写操作，与此同时更是对于计算机低层工作原理的一次探索。在尝试获得中断地址时由于中断被 TD 接管而未能获得正确的地址，这也从另一个方面说明了 dos 系统中断的工作方式，同时，这也锻炼了我对于问题的排查以及纠正能力。

在任务 4(加密实验) 中，我对于程序的加密和反跟踪方法有了初步的了解，并将之用于实践。同时我也了解到，反跟踪也是解释型语言相对于编译型语言所缺少的一项特性（由于解释型语言源码必须开放），这也是我对于各个商业软件的保全保护方式的一个初步探知。

而在任务 5(解密实验) 中，我对于程序的加密与反跟踪有了更为深入的了解，并深刻的体会到了破解程序要难于加密程序。同时，对于解密方法的尝试也让我了解到如何加密才能使程序更为安全，更不易被破解，从而相辅相成的了解了程序的加密解密的原理和方法。

实验五 WIN32 编程

I 实验目的与要求

- 熟悉 WIN32 程序的设计和调试方法；
- 熟悉宏汇编语言中 INVOKE、结构变量、简化段定义等功能；
- 进一步理解机器语言、汇编语言、高级语言之间以及实方式、保护方式之间的一些关系。

II 实验内容

编写一个基于窗口的 WIN32 程序，实现学生成绩表信息的平均值计算及显示功能（借鉴前面实验中的一些做法），具体要求如下描述。

功能一：编写一个基于窗口的 WIN32 程序的菜单框架，具有以下的下拉菜单项：

File	Action	Help
Exit	Average	About
	List	

点菜单 File 下的 Exit 选项时结束程序；点菜单 Help 下的选项 About，弹出一个消息框，显示本人信息，类似图 V-II-0-1 所示。点菜单 Action 下的选项 Average、List 将分别实现计算平均值或显示所有成绩的功能（详见功能二的描述）。

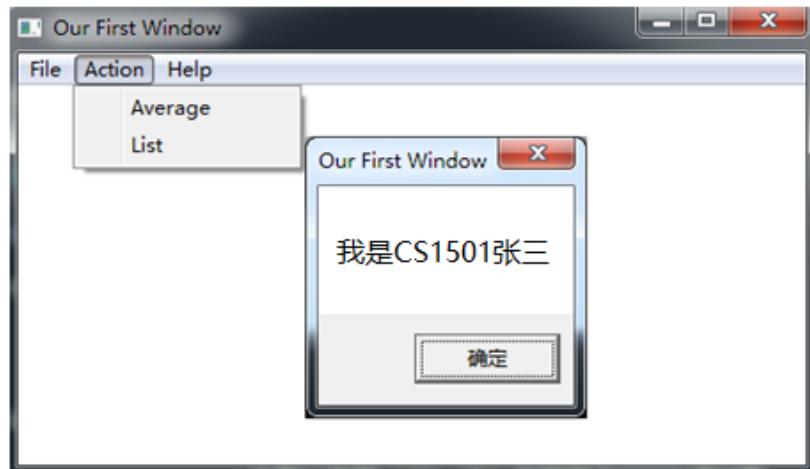


图 V-II-0-1: 菜单示例

功能二：每个学生的相关信息包括：姓名（结尾含 1 个以上的数值 0，共占 10 个字节），语文成绩（1 个字节），数学成绩（1 个字节），英语成绩（一个字节），平均成绩（1 个字节），等级（1 个字节）。要求采用结构变量存放学生的信息。学生人数至少 5 人。姓名和各科成绩直接在数据段中给定，不必运行时输入。成绩表中最后一个学生必须使用自己的姓名。

III. 实验过程

- 点菜单项 Average 时，计算平均成绩并给出等级（等级的定义见实验一，但这里不用单独显示等级）。平均成绩的计算仍按照实验一的公式进行。平均成绩和等级保存到上述结构变量的相应字段中。用 TD32 观察计算结果。
- 点菜单项 List 时，要求能在窗口中列出所有学生信息，包括姓名、各科成绩、平均成绩、等级等。如图 V-II-0-2 所示。平均成绩尚未计算时，平均成绩及等级显示为空白。

The screenshot shows a Windows application window titled "Our First Window". The menu bar includes "File", "Action", and "Help". The main area is labeled "List" and displays a table of student information:

Name	Chinese	Maths	English	Average	Grade
xueba	80	86	87	82	B
xuezha	46	55	58	50	F
zhangsan	70	86	87	77	C
lisigisan	67	76	61	68	D
wangwuan	77	56	69	69	D
chenliun	99	100	99	99	A

图 V-II-0-2: 成绩单显示示意图

III 实验过程

III.1 设计思想

win32 窗口程序与 dos 程序主要的不同在于其程序流程不同。win32 程序对于输入的轮询由 win32 库代替用户完成，各个组件（句柄）之间使用信号进行通信，而用户主要负责组件的构造以及信号的处理。除此之外，可以将数据使用 STRUCT 进行结构化存储，使用更为方便。此外，每一个句柄可将其视为一个对象，虽然各个函数没有被封装为类方法，但是整个程序依然采用了面向对象的设计思想。

III.2 模块图

图V-III-2-1展示了程序的各个模块及其关系：

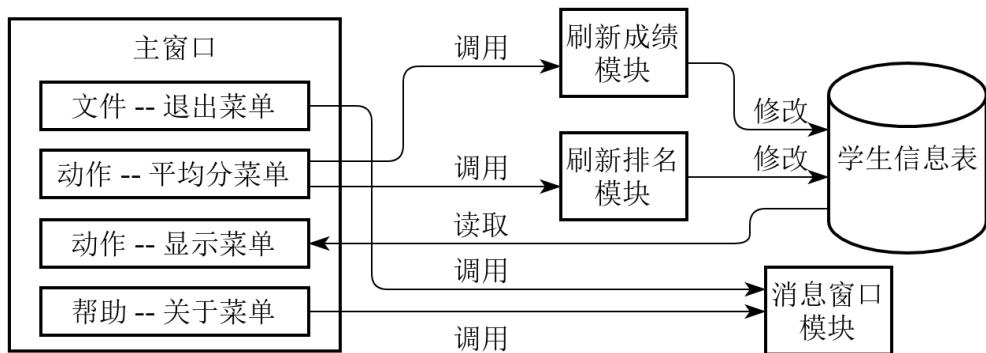


图 V-III-2-1: 程序模块图

III.3 实验步骤

1. 按照设计思想编写源程序，然后进行编译、链接。
2. 对于出现的错误定位其位置，改正后重新编译、链接，重复上述步骤直到没有错误出现。
3. 直接运行程序，对程序进行测试。
4. 打开 OllyDbg 对程序进行调试，观察程序的运行状况以及运行细节。

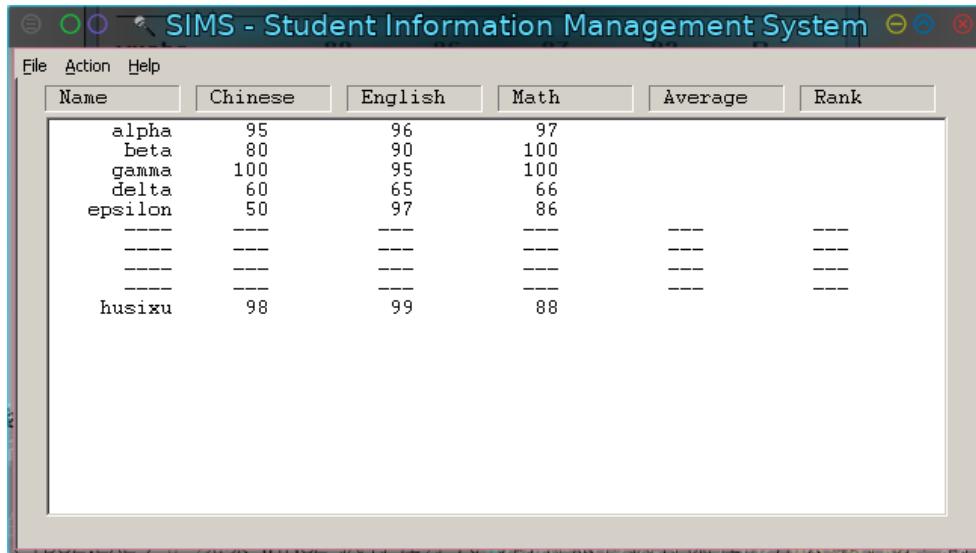
III.4 源程序

源程序 MAIN.ASM, RES.RC 详见附录部分 (第177页)

III.5 实验记录与分析

1. 实验环境条件: i7 3.6GHz, 8G 内存; Archlinux 下 Wine win32 模拟器; 使用 OllyDbg.exe 作为调试工具，ML.EXE 作为编译工具，LINK.EXE 作为链接工具，RC.EXE 作为资源编译工具，NMAKE 作为 makefile 处理工具。
2. 按照设计思想编写源程序，然后对于源程序进行编译和链接，在编译和链接的过程中没有出现错误。
3. 直接运行程序，选择 Action→show/refresh list，程序正常显示，如图V-III-5-1所示，对于未计算的平均分以及排名不予显示。

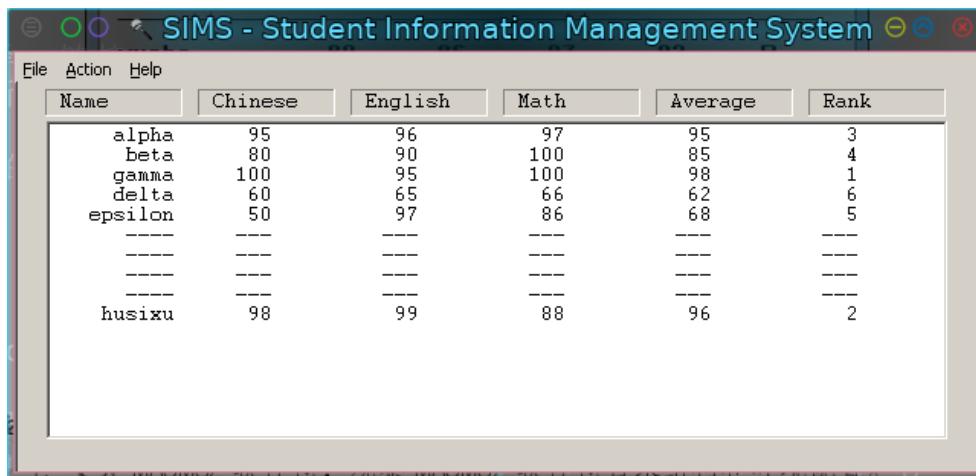
III. 实验过程



Name	Chinese	English	Math	Average	Rank
alpha	95	96	97		
beta	80	90	100		
gamma	100	95	100		
delta	60	65	66		
epsilon	50	97	86		
----	---	---	---	---	---
----	---	---	---	---	---
----	---	---	---	---	---
----	---	---	---	---	---
husixu	98	99	88		

图 V-III-5-1: 直接运行程序

4. 选择 Action→refresh Average and Rank, 出现提示信息后再次选择 Action→show/refresh list, 经过验证发现成绩与排名均计算正常, 如图V-III-5-2所示。



Name	Chinese	English	Math	Average	Rank
alpha	95	96	97	95	3
beta	80	90	100	85	4
gamma	100	95	100	98	1
delta	60	65	66	62	6
epsilon	50	97	86	68	5
----	---	---	---	---	---
----	---	---	---	---	---
----	---	---	---	---	---
----	---	---	---	---	---
husixu	98	99	88	96	2

图 V-III-5-2: 刷新平均分和排名之后的显示

5. 最后对于 About 和 Exit 进行测试, 发现其均能正常运行。
6. 打开 OllyDbg, 加载程序, 发现其各个段地址均为 32 位, 如图V-III-5-3所示, 而在对于一个函数进行 invoke 操作时, 在反汇编长口中其实是以 call 的方式进行, 但编译器会自动加上压栈与出栈命令来保护寄存器, 参数的传递也由编译器自动完成。

IV. 总结与体会

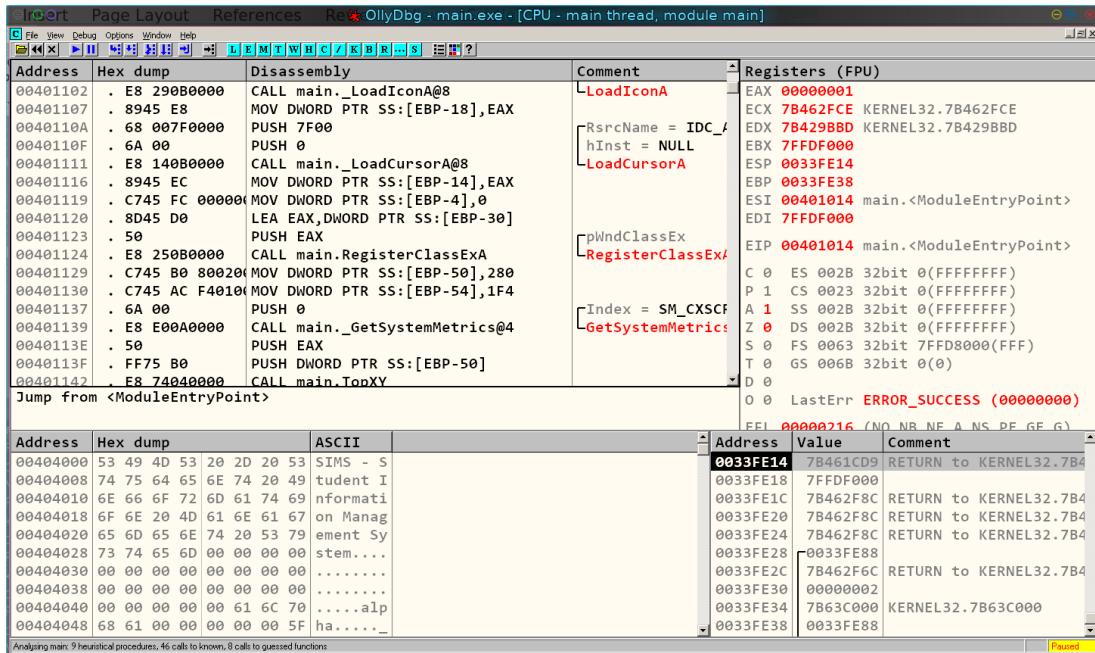


图 V-III-5-3: 打开调试界面

- 在使用 OllyDbg 的过程中，发现结构化数据对于程序的编写提供了极大的便利，并且相较于 Turbo Debugger 此调试器可以显示更多有关于源代码的信息，包括函数名称，系统调用，以及堆栈中的参数信息等，使用更加方便。

IV 总结与体会

通过本次试验，我对于汇编的了解更为深刻与广泛。我了解到汇编不仅仅限于较小的命令行程序的编写，同样可使用汇编较为容易的编写窗口程序，并使用 win32 汇编成功的将前几次的任务进行改写。此外，win32 汇编中的各个特性（如结构的引入）使我更加清晰的看到了汇编与编译型高级语言之间的关系，更深入的了解到了高级语言在编译的过程中是怎样被翻译的。

此外，我还了解到了几种反汇编 win32 程序的工具，如 TD32，OllyDbg，IDA Pro 等等，并尝试使用它们。在使用的过程中，TD32 未能正确识别 ML.EXE 生成的符号表，因此转而使用 OllyDbg 进行调试，这使我对于反汇编工具的特性有了更深入的了解，在今后的学习过程中可以使用其对程序进行调试与优化。

附录 A 参考文献

- [1] 元珍, 忠升, 宗芬. 80X86 汇编语言程序设计 [M]. 华中科技大学出版社, 2005.
- [2] Barry Wilks. DOS INT 21h - DOS Function Codes[EB/OL]. [2017-03-23].
<http://spike.scu.edu.au/barry/interrupts.html>.
- [3] Borland International. Turbo Debugger, Version 2.5: User's Guide. 1991.
- [4] Barry Wilks. DOS INT 21h - DOS Function Codes[EB/OL]. [2017-03-23].
<http://spike.scu.edu.au/barry/interrupts.html>.
- [5] 汇编语言教学网站 (<http://115.156.187.251/huibian1/site/index.jsp>) → 资料下载 → 案例
→win32 程序、编译和连接。
- [6] DOS Interrupts. (2017). Spike.scu.edu.au. Retrieved 4 May 2017, from
<http://spike.scu.edu.au/barry/interrupts.html>
- [7] Embedded Systems/Mixed C and Assembly Programming - Wikibooks, open books
for an open world. (2017). En.wikibooks.org. Retrieved 4 May 2017, from
https://en.wikibooks.org/wiki/Embedded_Systems/Mixed_C_and_Assembly_Programming
- [8] Heng, C. (2017). Free Disassemblers, Decompilers, Hexadecimal viewers, Hex ed-
itors (thefreecountry.com). Thefreecountry.com. Retrieved 4 May 2017, from
<https://www.thefreecountry.com/programming/disassemblers.shtml>
- [9] Inline Assembly - OSDev Wiki. (2017). Wiki.osdev.org. Retrieved 4 May 2017, from
http://wiki.osdev.org/Inline_Assembly
- [10] Mixing Assembly and C. (2017). Courses.engr.illinois.edu. Retrieved 4 May 2017, from
<https://courses.engr.illinois.edu/ece390/books/labmanual/c-prog-mixing.html>

附录 B 源代码

II.1 实验一

II.1.1 任务 4

```

1 .386
2
3 _stack segment use16 stack
4      db      100 dup(0)
5 _stack ends
6
7 ; CAUTION: use with care, affects ah and dl
8 outreturn macro
9      mov     ah, 02h      ; output an \r
10     mov     dl, 0dh
11     int     21h
12     mov     ah, 02h      ; output an \n
13     mov     dl, 0ah
14     int     21h
15 endm
16
17 data segment use16
18     n      equ 064h
19     tab    db 'Ju Song', 0, 0, 0, 100, 100, 99 ,0
20             db 'Jing Lu', 0, 0, 0, 100, 100, 100 ,0
21             db 'Yin Ma', 0, 0, 0, 0, 40, 80, 60 ,0
22             db 'Yi Ng', 0, 0, 0, 0, 0, 0, 0 ,0
23             db 'Sixu Hu', 0, 0, 0, 95, 90, 87 ,0
24
25             db 'Xiao Huan', 0, 3, 62, 7 ,0
26             db 'Fu Lim', 0, 0, 0, 0, 3, 38, 54 ,0
27             db 'Ru Chan', 0, 0, 0, 50, 59, 90 ,0
28             db 'Ya Hu', 0, 0, 0, 0, 0, 53, 11, 63 ,0
29             db 'Bao Han', 0, 0, 0, 74, 69, 10 ,0
30             db 'Wen Zhu', 0, 0, 0, 64, 85, 0 ,0
31             db 'Min Li', 0, 0, 0, 0, 89, 11, 51 ,0
32             db 'Mu Song', 0, 0, 0, 40, 49, 71 ,0
33             db 'Ya Ma', 0, 0, 0, 0, 0, 93, 49, 75 ,0
34             db 'Xun Hu', 0, 0, 0, 0, 54, 34, 79 ,0
35             db 'Xue Lu', 0, 0, 0, 0, 56, 80, 77 ,0
36             db 'Jia Lee', 0, 0, 0, 100, 86, 43 ,0
37             db 'Wen Man', 0, 0, 0, 23, 87, 44 ,0
38             db 'Ju Yu', 0, 0, 0, 0, 1, 38, 1 ,0
39             db 'Zhi Wen', 0, 0, 0, 27, 77, 29 ,0
40             db 'Ya Bai', 0, 0, 0, 0, 85, 84, 47 ,0
41             db 'Ru Guo', 0, 0, 0, 0, 63, 28, 80 ,0
42             db 'Mu Hsu', 0, 0, 0, 0, 13, 69, 96 ,0
43             db 'Ru Jin', 0, 0, 0, 0, 30, 47, 42 ,0
44             db 'Chun Wu', 0, 0, 0, 91, 58, 55 ,0
45             db 'Xue Li', 0, 0, 0, 0, 100, 69, 0 ,0
46             db 'Lan Lau', 0, 0, 0, 69, 99, 42 ,0
47             db 'Yong Xu', 0, 0, 0, 24, 60, 3 ,0
48             db 'Lim Wu', 0, 0, 0, 0, 21, 59, 95 ,0

```

```

49      db 'Fu Li', 0, 0, 0, 0, 0, 68, 11, 68 ,0
50      db 'Xun Lim', 0, 0, 0, 7, 90, 76 ,0
51      db 'Ru Han', 0, 0, 0, 0, 97, 42, 98 ,0
52      db 'Wu Ng', 0, 0, 0, 0, 0, 92, 6, 14 ,0
53      db 'Ju Hou', 0, 0, 0, 0, 91, 100, 77 ,0
54      db 'Ah Xun', 0, 0, 0, 0, 60, 82, 100 ,0
55      db 'Min Yu', 0, 0, 0, 0, 19, 70, 91 ,0
56      db 'Shu Hou', 0, 0, 0, 93, 19, 65 ,0
57      db 'Shi Liu', 0, 0, 0, 19, 9, 23 ,0
58      db 'Hua Wen', 0, 0, 0, 23, 3, 58 ,0
59      db 'Tu Chan', 0, 0, 0, 76, 96, 59 ,0
60      db 'Ya Tan', 0, 0, 0, 0, 4, 24, 62 ,0
61      db 'Ru Ng', 0, 0, 0, 0, 0, 26, 77, 39 ,0
62      db 'Xia Wen', 0, 0, 0, 96, 53, 81 ,0
63      db 'Da Lu', 0, 0, 0, 0, 0, 46, 86, 54 ,0
64      db 'Hui Guo', 0, 0, 0, 47, 60, 97 ,0
65      db 'Na Tan', 0, 0, 0, 0, 51, 51, 41 ,0
66      db 'Mu Liu', 0, 0, 0, 0, 31, 48, 1 ,0
67      db 'Yang Lu', 0, 0, 0, 62, 7, 83 ,0
68      db 'Zan Sun', 0, 0, 0, 10, 0, 15 ,0
69      db 'Zan Chu', 0, 0, 0, 7, 34, 34 ,0
70      db 'Tai Hu', 0, 0, 0, 0, 66, 97, 56 ,0
71      db 'Tu Yu', 0, 0, 0, 0, 0, 14, 69, 47 ,0
72      db 'Ya Ruan', 0, 0, 0, 86, 49, 7 ,0
73      db 'Bai Li', 0, 0, 0, 0, 51, 75, 78 ,0
74      db 'He Fan', 0, 0, 0, 0, 9, 76, 9 ,0
75      db 'Fu Zhao', 0, 0, 0, 8, 47, 50 ,0
76      db 'Chao Hu', 0, 0, 0, 43, 70, 20 ,0
77      db 'Jin Bai', 0, 0, 0, 35, 19, 69 ,0
78      db 'Xiu Li', 0, 0, 0, 0, 4, 54, 89 ,0
79      db 'Tao Guo', 0, 0, 0, 60, 53, 21 ,0
80      db 'Yun Yu', 0, 0, 0, 0, 61, 0, 67 ,0
81      db 'Ru Chen', 0, 0, 0, 82, 69, 68 ,0
82      db 'Bao Wu', 0, 0, 0, 0, 87, 20, 94 ,0
83      db 'Min Chu', 0, 0, 0, 17, 45, 13 ,0
84      db 'Jie Yu', 0, 0, 0, 0, 48, 82, 81 ,0
85      db 'Zan Ng', 0, 0, 0, 0, 54, 69, 7 ,0
86      db 'Na Guan', 0, 0, 0, 61, 49, 96 ,0
87      db 'Chao Xu', 0, 0, 0, 60, 27, 62 ,0
88      db 'Lim Yu', 0, 0, 0, 0, 74, 65, 63 ,0
89      db 'Hong Yu', 0, 0, 0, 88, 86, 59 ,0
90      db 'Bai Wen', 0, 0, 0, 74, 94, 92 ,0
91      db 'Jin Lim', 0, 0, 0, 31, 44, 2 ,0
92      db 'Hui Pan', 0, 0, 0, 9, 61, 21 ,0
93      db 'Jin Xu', 0, 0, 0, 0, 86, 1, 7 ,0
94      db 'Ru Wang', 0, 0, 0, 9, 25, 80 ,0
95      db 'Mu Tan', 0, 0, 0, 0, 73, 41, 97 ,0
96      db 'Qing Yu', 0, 0, 0, 49, 47, 8 ,0
97      db 'Huan Xu', 0, 0, 0, 58, 36, 70 ,0
98      db 'Wei Yu', 0, 0, 0, 0, 24, 63, 17 ,0
99      db 'Yin Xun', 0, 0, 0, 13, 28, 88 ,0
100     db 'Bao Man', 0, 0, 0, 11, 97, 41 ,0
101     db 'Jun Lin', 0, 0, 0, 77, 88, 71 ,0
102     db 'Ru Guan', 0, 0, 0, 55, 94, 6 ,0
103     db 'Da Sung', 0, 0, 0, 24, 63, 25 ,0

```

B. 源代码

```
104         db 'Wen Man', 0, 0, 0, 38, 56, 56 ,0
105         db 'Wen Guo', 0, 0, 0, 84, 57, 90 ,0
106         db 'Lan Xun', 0, 0, 0, 80, 88, 28 ,0
107         db 'Hua Hsu', 0, 0, 0, 93, 66, 33 ,0
108         db 'Da Lim', 0, 0, 0, 0, 49, 95, 24 ,0
109         db 'Tu Yu', 0, 0, 0, 0, 8, 9, 88 ,0
110         db 'Ming Wu', 0, 0, 0, 78, 77, 2 ,0
111         db 'Yun Yu', 0, 0, 0, 0, 3, 10, 73 ,0
112         db 'Wen Hu', 0, 0, 0, 0, 35, 57, 56 ,0
113         db 'Xia Lee', 0, 0, 0, 62, 58, 27 ,0
114         db 'Ah Sung', 0, 0, 0, 9, 26, 89 ,0
115         db 'Wu Wu', 0, 0, 0, 0, 0, 32, 49, 61 ,0
116         db 'Lan Chu', 0, 0, 0, 22, 0, 65 ,0
117         db 'Min Lau', 0, 0, 0, 55, 60, 95 ,0
118         db 'Dong Ma', 0, 0, 0, 46, 74, 33 ,0
119         db 'Jia Bai', 0, 0, 0, 100, 1, 96 ,0
120     in_name db 10
121     db ?
122     in_buff db 10 dup(0)
123     output1 db 'Please input the name of the student: $'
124     output2 db 'Student not found, please re-input ... ',0dh,0ah,'$'
125     output3 db 'the grades of the student are: $'
126 data ends
127
128 code segment use16
129 assume ss:_stack, ds:data, cs:code
130 start:
131     mov ax, data
132     mov ds, ax
133 input:    ; input the name of student
134     mov ah, 09h      ; show input message
135     lea dx, output1
136     int 21h
137
138     mov ah, 0ah      ; input the name
139     lea dx, in_name
140     int 21h
141
142     outreturn
143
144     cmp in_buff, 0dh      ; if input is \r
145     jz input
146
147     cmp in_buff, 71h      ; test if quit
148     jnz find_s
149     cmp in_buff+01h, 0dh
150     jnz find_s          ; q but not q\r
151     mov ah, 4ch
152     int 21h
153
154     ;-----
155 find_s:
156     mov esi, 0      ; if input is a name
157     mov edi, 0      ; counter
158     ;TODO check the legitimacy of the name
```

B. 源代码

```
159 find:      ; find the student's name
160     mov    ecx, 0          ; name length counter
161     mov    eax, edi
162     mov    bx, 0eh        ; mov bx, 14
163     mul    bx
164     mov    esi, eax
165 if_equal:   ; if the two string are equal
166     cmp    in_buff[ecx], 0dh
167     jnz    find_w        ; if input not reach end
168     cmp    byte ptr tab[esi][ecx], 0 ; if reach end in the same time
169     jz     found         ; if found
170     cmp    ecx, 0ah       ;
171     jz     found         ;
172     jmp    next          ; if not found in this row
173 find_w:    ; if the two word are equal
174     cmp    ecx, 0ah       ; avoid length go over 10
175     jz     next          ; if the length is over 10
176
177     mov    al, in_buff[ecx]
178             ; compare two character
179     cmp    byte ptr tab[esi][ecx], al
180     jne    next
181     inc    ecx
182     jmp    if_equal      ;if this character matches
183 next:     ; two string not equal, go next string
184     inc    edi
185     cmp    edi, n
186     jne    find          ; if length is not equal
187             ; if not found
188     mov    ah, 09h
189     lea    dx, output2
190     int    21h
191     jmp    input
192
193
194 found:    ; if found
195             ;TODO show detail grade of the student
196     mov    bl, 02h        ; calculate the average grade of the student
197     mov    ecx, 0ah
198     movzx  ax, tab[esi][ecx]
199     mul    bx
200     push   ax
201
202     inc    ecx
203     movzx  ax, tab[esi][ecx]
204     push   ax
205
206     inc    ecx
207     movzx  ax, tab[esi][ecx]
208     div    bl
209     mov    ah, 0
210
211     add    ax, [esp]
212     add    esp, 02h      ; perform a pop
213     add    ax, [esp]
```

B. 源代码

```
214      add    esp, 02h      ; perform a pop
215
216      mov    bx, 02h
217      mul    bx
218      mov    bx, 07h
219      div    bl
220
221      inc    ecx          ; store the average grade
222      mov    tab[esi][ecx], al
223
224  ;-----+
225      ;judge the Level
226      movzx  cx, al
227      mov    bx, 5ah      ; mov bx, 90
228      cmp    cx, bx
229      js     b
230      mov    dl, 'A'
231      jmp    outgrade
232  b:
233      mov    bx, 50h      ; mov bx, 80
234      cmp    cx, bx
235      js     c
236      mov    dl, 'B'
237      jmp    outgrade
238  c:
239      mov    bx, 46h      ; mov bx, 70
240      cmp    cx, bx
241      js     d
242      mov    dl, 'C'
243      jmp    outgrade
244  d:
245      mov    bx, 3ch      ; mov bx, 60
246      cmp    cx, bx
247      js     f
248      mov    dl, 'D'
249      jmp    outgrade
250  f:
251      mov    dl, 'F'
252
253  outgrade:
254      push   dx          ; temporarily store the Level
255
256      mov    dx, offset output3 ; output the grade notification
257      mov    ah, 09H
258      int    21H
259
260      ; output the average grade
261      mov    si, 0          ; serve as a counter
262  hundreds:
263      mov    ax, cx          ; the grade
264      mov    bl, 64H
265      div    bl          ; get the hundreds
266      cmp    al, 0          ; compare the quotient
267      jz     tens          ; if hundreds is zero, then not output
268      inc    si
```

B. 源代码

```
269  hundreds_out:  
270      add     al, 30H  
271      mov     dl, al  
272      mov     ah, 02H  
273      int     21H  
274  
275  tens:  
276      mov     ax, cx      ; the grade  
277      mov     bl, 64H  
278      div     bl          ; get the tens  
279      cmp     ah, 0  
280      mov     al, ah  
281      cbw  
282      mov     bl, 0ah  
283      div     bl  
284      cmp     al, 0  
285      jnz     tens_out   ; if tens is zero and hundreds is zero, then not output  
286      cmp     si, 0  
287      jz      units  
288  tens_out:  
289      add     al, 30H  
290      mov     dl, al  
291      mov     ah, 02H  
292      int     21H  
293  
294  units:  
295      mov     ax, cx      ; the grade  
296      mov     bl, 0ah  
297      div     bl          ; get the tens  
298      cmp     ah, 0          ; compare remainings  
299  units_out:  
300      add     ah, 30H  
301      mov     dl, ah  
302      mov     ah, 02H  
303      int     21H  
304  
305      mov     dl, ','  
306      mov     ah, 02h  
307      int     21H  
308  
309      pop     dx          ; get the level back  
310      mov     ah, 02h  
311      int     21h  
312      outreturn  
313  
314      jmp     start  
315  
316  stop:  
317  code ends  
318  end start
```

II.2 实验二

II.2.1 任务 1

原程序见报告 1，此处为修改后的程序。

代码修改由命令 ‘diff -u <old-file> <new-file>’ 生成，‘+’表示增加的程序行，‘-’表示删除的程序行，‘@@ ... @@’ 表示修改的段落（新旧文件的行数），未显示部分为未修改部分

```

1 —— ../../homework_1/src/STUDENT.ASM      2017-03-22 16:20:17.023425342 +0800
2 +++ STUMOD1.ASM 2017-03-28 14:59:37.691770314 +0800
3 @@ -151,8 +151,13 @@
4         mov     ah, 4ch
5         int     21h
6
7 —————
8 find_s:
9 +—— Adding the big loop for time test ———
10 +    mov     cx, 7530H ;loop time
11 +    call    far ptr dispTime
12 +loop_big_s label far
13 +    push   cx          ;protect cx
14 +
15         mov     esi, 0      ; if input is a name
16         mov     edi, 0      ; counter
17         ;TODO check the legitimacy of the name
18 @@ -255,7 +260,7 @@
19
20         mov     dx, offset output3 ; output the grade notification
21         mov     ah, 09H
22         int     21H
23 +;    int     21H
24
25         ; output the average grade
26         mov     si, 0      ; serve as a counter
27 @@ -270,7 +275,7 @@
28         add    al, 30H
29         mov    dl, al
30         mov    ah, 02H
31         int     21H
32 +;    int     21H
33
34 tens:
35         mov     ax, cx      ; the grade
36 @@ -289,7 +294,7 @@
37         add    al, 30H
38         mov    dl, al
39         mov    ah, 02H
40         int     21H
41 +;    int     21H
42
43 units:
44         mov     ax, cx      ; the grade
45 @@ -300,19 +305,80 @@
46         add    ah, 30H
47         mov    dl, ah

```

B. 源代码

```
48      mov     ah, 02H
49      -      int    21H
50      +;     int    21H
51
52      mov     dl, ','
53      mov     ah, 02h
54      -      int    21H
55      +;     int    21H
56
57      pop     dx          ; get the level back
58      mov     ah, 02h
59      -      int    21h
60      -      outreturn
61      +;     int    21h
62      +;     outreturn
63
64      +;----- End of the time testing -----
65      +      pop     cx
66      +      dec     cx
67      +      jnz     far ptr loop_big_s
68      +      call    far ptr disptime
69      +;-----
70      jmp     start
71
72 stop:
73      +      mov     ah, 4ch
74      +      int    21h
75      +
76 +===== other functions =====
77 +disptime proc far
78 +  local timestr[11]:byte      ;m,m,\',s,s,\",c,c,\r,\n,'$'
79 +  push    cx
80 +  push    dx
81 +  push    ds
82 +  push    bx
83 +
84 +  push    ss
85 +  pop     ds
86 +
87 +  mov     ah,2ch
88 +  int    21h
89 +
90 +  mov     bl,10
91 +
92 +  xor     ax,ax          ; store the minutes
93 +  mov     al,cl
94 +  div     bl
95 +  add     ax,3030h
96 +  mov     word ptr timestr, ax
97 +
98 +  mov     timestr+2,27H      ;\' '
99 +
100 +   xor    ax,ax          ; store the seconds
101 +   mov     al,dh
102 +   div     bl
```

B. 源代码

```
103 +     add    ax,3030h
104 +     mov    word ptr timestr+3,ax
105 +
106 +     mov    timestr+5,'"'
107 +
108 +     xor    ax,ax           ; store the centiseconds
109 +     mov    al,dl
110 +     div    bl
111 +     add    ax,3030h
112 +     mov    word ptr timestr+6,ax
113 +
114 +     mov    word ptr timestr+8,0a0dh
115 +     mov    timestr+10,'$'
116 +
117 +     lea    dx,timestr
118 +     mov    ah,9
119 +     int    21h
120 +
121 +     pop    bx
122 +     pop    ds
123 +     pop    dx
124 +     pop    cx
125 +     ret
126 +disptime  endp
127 +-----
```

```
128 code ends
129 end start
```

II.2.2 任务 2

在任务一的基础上进行修改过后的源程序变更如下：

代码修改由命令 ‘`diff -u <old-file> <new-file>`’ 生成，‘+’表示增加的程序行，‘-’表示删除的程序行，‘@@ ... @@’ 表示修改的段落（新旧文件的行数），未显示部分为未修改部分

```

1 —— STUMOD1.ASM 2017-03-28 14:59:37.691770314 +0800
2 +++ STUTUNE.ASM 2017-03-28 20:28:09.054451103 +0800
3 @@ -159,21 +159,13 @@
4     push    cx          ;protect cx
5     -
6     mov     esi, 0       ; if input is a name
7     -     mov     edi, 0       ; counter
8     find:    ; find the student's name
9     mov     ecx, 0       ; name length counter
10    -    mov     eax, edi
11    -    mov     bx, 0eh      ; mov bx, 14
12    -    mul     bx
13    -    mov     esi, eax
14    if_equal:   ; if the two string are equal
15    cmp     in_buff[ecx], 0dh
16    jnz     find_w      ; if input not reach end
17    cmp     byte ptr tab[esi][ecx], 0 ; if reach end in the same time
18    jz     found        ; if found
19    -    cmp     ecx, 0ah      ;
20    -    jz     found        ;
21    jmp     next        ; if not found in this row
22    find_w:    ; if the two word are equal
23    cmp     ecx, 0ah      ; avoid length go over 10
24 @@ -186,8 +178,8 @@
25     inc     ecx
26     jmp     if_equal      ;if this character matches
27    next:    ; two string not equal, go next string
28    -    inc     edi
29    -    cmp     edi, n
30    +    add     esi, 0eh
31    +    cmp     esi, 0578h
32    jne     find        ; if length is not equal
33           ; if not found
34    mov     ah, 09h

```

II.2.3 任务 3

源代码分为 3 个部分：汇编部分，C 程序部分以及由 C 程序生成的汇编部分。

汇编部分：

```
1 data segment use16
2     in_name db 10
3     db ?
4     in_buff db 10 dup(0)
5     output1 db 'Please input the name of the student: $'
6 data ends
7
8 code segment use16
9 assume ss:_stack, ds:data, cs:code
10 start:
11     mov ax, data
12     mov ds, ax
13 input:    ; input the name of student
14     mov ah, 09h           ; show input message
15     lea dx, output1
16     int 21h
17
18     mov ah, 0ah           ; input the name
19     lea dx, in_name
20     int 21h
21
22     outreturn
23
24     cmp in_buff, 0dh      ; if input is \r
25     jz input
26
27     cmp in_buff, 71h      ; test if quit
28     jnz find_s
29     cmp in_buff+01h, 0dh
30     jnz find_s           ; q but not q\r
31     mov ah, 4ch
32     int 21h
33 find_s:
34     ;enter next procedure
35     mov ah, 4ch
36     int 21h
37 code ends
38 end start
```

C 程序部分：

```
1 #include <stdio.h>
2
3 int main(void){
4     char in_name[10]="";
5
6     do{
7         printf("Please input the student's name: ");
8         scanf("%[^\\n]",in_name);
9         getchar();
10    }while(!in_name[0]);
```

B. 源代码

```
11     if(in_name[0]=='q' && !in_name[1])
12         return 0;
13
14
15     printf("entering next procedure...\n");
16     return 0;
17 }
```

生成汇编部分：

```
1 ifndef ??version
2 ?debug macro
3 endm
4 $comm macro name,dist,size,count
5     comm dist name:BYTE:count*size
6 endm
7 else
8 $comm macro name,dist,size,count
9     comm dist name[size]:BYTE:count
10 endm
11 endif
12 ?debug S "stu_c.c"
13 ?debug C E974797D4A077374755F632E63
14 ?debug C E98D0A7D4A15433A5C54435C494E434C5544455C737464696F2E68
15 ?debug C E98D0A7D4A15433A5C54435C494E434C5544455C5F646566732E68
16 ?debug C E98D0A7D4A15433A5C54435C494E434C5544455C5F6E756C6C2E68
17 _TEXT segment byte public 'CODE'
18 _TEXT ends
19 DGROUP group _DATA,_BSS
20 assume cs:_TEXT,ds:DGROUP
21 _DATA segment word public 'DATA'
22 d@ label byte
23 d@w label word
24 _DATA ends
25 _BSS segment word public 'BSS'
26 b@ label byte
27 b@w label word
28 _BSS ends
29 _DATA segment word public 'DATA'
30 db 10 dup (?)
31 _DATA ends
32 _TEXT segment byte public 'CODE'
33 ;
34 ; int main(void){
35 ;
36 assume cs:_TEXT
37 _main proc near
38     push bp
39     mov bp,sp
40     sub sp,10
41 ;
42 ;     char in_name[10]="";
43 ;
44     lea ax,word ptr [bp-10]
45     push ss
46     push ax
```

B. 源代码

```
47     mov ax,offset DGROUP:d@w+0
48     push    ds
49     push    ax
50     mov cx,10
51     call    near ptr N_SCOPY@  
52 @1@58:
53     ;
54     ;
55     ;      do{
56     ;          printf("Please input the student's name: ");
57     ;
58     mov ax,offset DGROUP:s@  
59     push    ax
60     call    near ptr _printf
61     pop cx
62     ;
63     ;          scanf("%[^\\n]",in_name);
64     ;
65     lea ax,word ptr [bp-10]
66     push    ax
67     mov ax,offset DGROUP:s@+34
68     push    ax
69     call    near ptr _scanf
70     pop cx
71     pop cx
72     ;
73     ;          getchar();
74     ;
75     dec word ptr DGROUP:_streams
76     jl short @1@114
77     mov bx,word ptr DGROUP:_streams+10
78     inc word ptr DGROUP:_streams+10
79     mov al,byte ptr [bx]
80     mov ah,0
81     jmp short @1@142
82 @1@114:
83     mov ax,offset DGROUP:_streams
84     push    ax
85     call    near ptr __fgetc
86     pop cx
87 @1@142:
88     ;
89     ;      }while(!in_name[0]);
90     ;
91     mov al,byte ptr [bp-10]
92     cbw
93     or    ax,ax
94     je    short @1@58
95     ;
96     ;
97     ;      if(in_name[0]=='q' && !in_name[1])
98     ;
99     cmp byte ptr [bp-10],113
100    jne short @1@254
101    mov al,byte ptr [bp-9]
```

B. 源代码

```
102     cbw
103     or  ax,ax
104     jne short @1@254
105     @1@226:
106     ;
107     ;           return 0;
108     ;
109     xor ax,ax
110     jmp short @1@282
111     @1@254:
112     ;
113     ;
114     ;           printf("entering next procedure...\\n");
115     ;
116     mov ax,offset DGROUP:s@+40
117     push   ax
118     call    near ptr _printf
119     pop   cx
120     jmp short @1@226
121     @1@282:
122     ;
123     ;           return 0;
124     ;       }
125     ;
126     mov sp,bp
127     pop bp
128     ret
129     _main  endp
130     ?debug C E9
131     _TEXT  ends
132     _DATA  segment word public 'DATA'
133     s@  label byte
134     db  'Please input the student'
135     db  39
136     db  's name: '
137     db  0
138     db  '%[^'
139     db  10
140     db  ']'
141     db  0
142     db  'entering next procedure...'
143     db  10
144     db  0
145     _DATA  ends
146     _TEXT  segment byte public 'CODE'
147     _TEXT  ends
148     extrn N_SCOPY@:far
149     public _main
150     extrn __fgetc:near
151     extrn __scanf:near
152     extrn __printf:near
153     extrn __streams:word
154     _s@ equ s@
155     end
```

II.3 实验三

II.3.1 任务 1

主程序 MAIN.ASM：

```

1 .386
2 include macrolib
3
4 ; all submodule share the same table
5 public      tab, stu_max_num
6
7 extrn      register:near
8 extrn      calcavg:near
9 extrn      calcrank:near
10 extrn     printall:near
11
12 _stack segment use16 stack "stack"
13     db  OFFH DUP(0)
14 _stack ends
15
16 data segment use16 public "data"
17     input_max_num    equ 10
18     stu_max_num     equ 10
19     menu      db  ' Welcome to Student Manage Program %%%%%%%%%%', 0dh, 0ah
20             db  ' %%%%%%%%%%%%%%%%', 0dh, 0ah
21             db  ' 1) Record Student info          ', 0dh, 0ah
22             db  ' 2) Calculate Average Score      ', 0dh, 0ah
23             db  ' 3) Calculate Ranks            ', 0dh, 0ah
24             db  ' 4) Show Grades and Ranks      ', 0dh, 0ah
25             db  ' 0) Exit                      ', 0dh, 0ah
26             db  ' %%%%%%%%%%%%%%%%', 0dh, 0ah
27             db  'Please input your choice >>> $'
28
29     info1   db  ' %%%%%%%%%%%%%%%%', 0dh, 0ah
30             db  ' Thanks for using this system, bye %%%%%%%%%%', 0dh, 0ah
31             db  ' %%%%%%%%%%%%%%%%', 0dh, 0ah, '$'
32
33     err1    db  'Unidentified choice, Please re-input ... ', 0dh, 0ah, '$'
34 ; modules do not share the input buffer
35     input    db  input_max_num
36             db  ?
37     buffer   db  input_max_num dup(0)
38
39 ; 10 bytes name, 6 bytes score (3*2), 2 bytes average, 2 bytes rank
40     tab      dw  stu_max_num dup(5 dup(0), -1h, -1h, -1h, -1h, -1h)
41 data ends
42
43 code segment use16 public "code"
44     assume ss:_stack, cs:code, ds:data
45 start:
46     mov      ax, data
47     mov      ds, ax
48     mov      ax, 0504h           ; set current page to 4
49     int      10h
50

```

B. 源代码

```
51 main:  
52     cls  
53     puts    <offset menu>  
54     gets    <offset input>, 2  
55  
56 casel:  
57     cmp     buffer, '1'  
58     jne    case2  
59  
60     call    register  
61     jmp    main  
62  
63 case2:  
64     cmp     buffer, '2'  
65     jne    case3  
66  
67     call    calcavg  
68     jmp    main  
69  
70 case3:  
71     cmp     buffer, '3'  
72     jne    case4  
73  
74     call    calcrank  
75     jmp    main  
76  
77 case4:  
78     cmp     buffer, '4'  
79     jne    case0  
80  
81     call    printall  
82     jmp    main  
83  
84 case0:  
85     cmp     buffer, '0'  
86     jne    default  
87  
88     puts    <offset infol>  
89     mov    ah, 4ch  
90     int    21h  
91  
92 default:  
93     puts    <offset err1>  
94     jmp    main  
95  
96 code ends  
97 end start
```

录入模块 SUBMOD1.ASM :

```
1 .386  
2 include macrolib  
3  
4 public    register  
5  
6 extern    tab:byte
```

B. 源代码

```
7  extern      stu_max_num:abs
8
9  _stack segment use16 stack "stack"
10 stack ends
11
12 data segment use16 public "data"
13     input_max_num    equ 10
14     grade_max_num   equ 100
15
16     info1  db  'Please input the index you want to edit: $'
17     info2  db  'Please input the Name of the student: $'
18     info3  db  'Please input the Chinese score: $'
19     info4  db  'Please input the Math score : $'
20     info5  db  'Please input the English score: $'
21     info6  db  'Student info registered.', 0dh, 0ah, '$'
22
23     ; the number should be equal to stu_max_num
24     err1   db  'Index exceeded, range is 0-9', 0dh, 0ah, '$'
25     err2   db  'Grade exceeded, max is 100', 0dh, 0ah, '$'
26
27     input   db  input_max_num
28     db ?
29     buffer  db  input_max_num dup(0)
30
31     ; store the input info temporarily, for atomic operation
32     stu_buf db  20 dup(0)
33 data ends
34
35 code segment use16 public "code"
36     assume cs:code, ds:data, ss:_stack, es:data
37 start:
38
39 register proc
40 pushad
41 push    ds
42 push    es
43 mov    ax, data
44 mov    ds, ax
45 mov    es, ax
46 reg_s:
47     ; get the index of the table
48     puts    <offset info1>
49     gets    <offset input>, 3
50
51     ; if doesn't get anything
52     cmp    input+1h, 0
53     jz    reg_s
54
55     atoi    <offset buffer>, 0dh      ; convert to num
56     pop    eax
57
58     cmp    eax, 0                  ; make sure the index does not exceeds
59     jl    error1
60     cmp    eax, stu_max_num
61     jge    error1
```

B. 源代码

```
62      mov     ebx, 20          ; get the real position of the student
63      mul     ebx
64      push    eax          ; store the position
65
66 stuname:
67      ; get the name of the student
68      puts   <offset info2>
69      gets   <offset input>, 10
70
71      cmp     input+1, byte ptr 00h  ; make sure there's input
72      je      stuname
73
74      mov     cx, 0ah          ; clear the buffer
75      lea     eax, stu_buf
76 clear:
77      mov     [eax], byte ptr 00h
78      inc     eax
79      loop   clear
80
81      movzx  ecx, input+1h    ; get the input length
82      lea     esi, buffer
83      lea     edi, stu_buf    ; store the name temporarily
84      rep     movsb
85
86      ; get the scores of the student
87 chinese:
88      puts   <offset info3>      ; chinese
89      gets   <offset input>, 4
90
91      cmp     input+1, byte ptr 00h  ; make sure there's input
92      je      chinese
93
94      atoi   <offset buffer>, 0dh
95      pop    eax
96      cmp     eax, 0          ; make sure chinese score does not exceeds
97      jl     error2
98      cmp     eax, grade_max_num
99      jg     error2
100     mov    word ptr stu_buf+0ah, ax; store chinese score temporarily
101 math:
102     puts   <offset info4>      ; math
103     gets   <offset input>, 4
104
105     cmp     input+1, byte ptr 00h  ; make sure there's input
106     je      math
107
108     atoi   <offset buffer>, 0dh
109     pop    eax
110     cmp     eax, 0          ; make sure math score does not exceeds
111     jl     error2
112     cmp     eax, grade_max_num
113     jg     error2
114     mov    word ptr stu_buf+0ch, ax; store math score temporarily
115 english:
116     puts   <offset info5>      ; english
```

B. 源代码

```
117     gets    <offset input>, 4
118
119     cmp     input+1, byte ptr 00h ; make sure there's input
120     je      english
121
122     atoi   <offset buffer>, 0dh
123     pop    eax
124     cmp    eax, 0                ; make sure english score does not exceeds
125     jl     error2
126     cmp    eax, grade_max_num
127     jg     error2
128     mov    word ptr stu_buf+0eh, ax; store english score temporarily
129 pass:
130     ; if passed all test
131     pop    eax                ; get the position back
132     mov    ecx, 14H            ; transport all data (20bytes)
133
134     ; sent the name to correct position
135     lea    esi, stu_buf
136     lea    edi, tab[eax]
137     rep    movsb
138     puts   <offset info6>
139     gets   <offset input>, 1
140     jmp    ending
141
142 ; index exceeds
143 error1:
144     puts   <offset err1>
145     gets   <offset input>, 1      ; wait for return
146     jmp    ending
147
148 ; grade exceeds
149 error2:
150     pop    eax                ; index not exceeds, but get an error, pop the temporarily
151     stored index
152     puts   <offset err2>
153     gets   <offset input>, 1      ; wait for return
154     jmp    ending
155 ending:
156     pop    es
157     pop    ds
158     popad
159     ret
160     register endp
161
162 code ends
163 end start
```

平均分计算模块 SUBMOD2.ASM :

```
1 .386
2 include macrolib
3
4 public    calcavg
5 public    refresh_avg
```

B. 源代码

```
6
7  extrn      tab:byte
8  extrn      stu_max_num:abs
9  extrn      name_to_index:far
10 extrn      printname:far
11
12 _stack segment use16 stack "stack"
13 _stack ends
14
15 data segment use16 public "data"
16     input_max_num    equ 10
17     info1   db  ' Submenu - Average Score Calculate %%%%%%', 0dh, 0ah
18     db  '%%%%%%%%%%%%%', 0dh, 0ah
19     db  '    1) Calculate/Refresh all students      ', 0dh, 0ah
20     db  '    2) Specify a student (by index)      ', 0dh, 0ah
21     db  '    3) Specify a student (by name)      ', 0dh, 0ah
22     db  '    0) Exit                          ', 0dh, 0ah
23     db  '%%%%%%%%%%%%%', 0dh, 0ah
24     db  'Please input your choice >>> $'
25
26     info2   db  'Index $'
27     info3   db  ': average Score is $'
28     info4   db  ': Name is $'
29
30     info5   db  'Please input the name: $'
31     info6   db  ': Student not registered $'
32     info7   db  'Please input the index: $'
33
34     err1   db  'Unidentified choice, Please re-input ... ', 0dh, 0ah, '$'
35     err2   db  'Student not found ... ', 0dh, 0ah, '$'
36     err3   db  'Index exceeded, range is 0-9', 0dh, 0ah, '$'
37
38     input    db  input_max_num
39     db  ?
40     buffer   db  input_max_num dup(0)
41 data ends
42
43 code segment use16 public "code"
44     assume cs:code, ds:data, ss:_stack, es:data
45 start:
46
47 calcavg proc
48 pushad
49 push    ds
50 push    es
51 mov     ax, data
52 mov     ds, ax
53 mov     es, ax
54 cal_s:
55     cls
56     puts    <offset info1>
57     gets    <offset input>, 2
58
59 casel:
60     cmp     buffer, '1'
```

B. 源代码

```
61      jne      case2
62
63      mov      cx, 00h
64  loop_a  label far
65      mov      eax, 00h
66      mov      ax, cx
67      mov      bx, 14h
68      mul      bx
69                      ; if the not registered
70      cmp      tab[eax], byte ptr 00h
71      jne      case1_cal_this_one
72      puts    <offset info2>
73      itoa    <offset buffer>, cx
74      puts    <offset buffer>
75      puts    <offset info6>
76      outreturn
77      jmp      far ptr cal_one_end
78
79  case1_cal_this_one:
80      push    cx           ; pass the parameter
81      call    cal_one_avg
82      pop     ax           ; get the result (but do not use)
83
84      puts    <offset info2>
85      itoa    <offset buffer>, cx
86      puts    <offset buffer>
87
88      puts    <offset info4>
89      push    cx           ; print name
90      call    far ptr printname
91      pop     cx
92
93      puts    <offset info3>
94      itoa    <offset buffer>, ax
95      puts    <offset buffer>
96      outreturn
97
98  cal_one_end label far
99      inc     cx
100     cmp     cx, stu_max_num
101     jnz     far ptr loop_a
102
103     jmp     far ptr looping
104
105  case2:
106     cmp     buffer, '2'
107     jne     case3
108
109  case2_input:
110     puts    <offset info7>       ; get the index
111     gets    <offset input>, 3
112
113     cmp     input+1h, 0          ; if doesn't get anything
114     jz      case2_input
115
```

B. 源代码

```
116     atoi    <offset buffer>, 0dh      ; convert to num
117     pop     ecx
118
119     cmp     ecx, 0                  ; make sure the index does not exceeds
120     jl      error3
121     cmp     ecx, stu_max_num
122     jge    error3
123
124     mov     eax, 00h
125     mov     ax, cx
126     mov     bx, 14h
127     mul     bx
128
129     cmp     tab[eax], byte ptr 00h  ; if not registered
130     jne    case2_cal_this_one
131     puts   <offset info2>
132     itoa   <offset buffer>, cx
133     puts   <offset buffer>
134     puts   <offset info6>
135     outreturn
136     jmp    far ptr looping
137
138 case2_cal_this_one:
139     push   cx                      ; pass the parameter
140     call   cal_one_avg
141     pop    ax                      ; get the result (but do not use)
142
143     puts   <offset info2>
144     itoa   <offset buffer>, cx
145     puts   <offset buffer>
146
147     puts   <offset info4>
148     push   cx                      ; print name
149     call   far ptr printname
150     pop    cx
151
152     puts   <offset info3>
153     itoa   <offset buffer>, ax
154     puts   <offset buffer>
155     outreturn
156
157     jmp    far ptr looping
158
159 case3:
160     cmp     buffer, '3'
161     jne    case0
162
163 case3_input:
164     puts   <offset info5>
165     gets   <offset input>, 10
166
167     cmp     input+1, byte ptr 00h  ; make sure there's input
168     je     case3_input
169
170     mov     si, offset buffer       ; make the buffer end with \0...\0
```

B. 源代码

```
171      mov     cx, 0ah
172  a0:
173      cmp     ds:[si], byte ptr 0dh
174      je      b0
175      inc     si
176      loop   a0
177  b0:
178      mov     ds:[si], byte ptr 0
179      inc     si
180      loop   b0
181
182      mov     si, offset buffer
183      push   si
184      call   far ptr name_to_index
185
186      pop    cx           ; retreive the index into cx
187      cmp    cx, -1h        ; if not found
188      je     error2
189      push   cx
190
191      call   cal_one_avg    ; the parameter is already in stack
192      pop    ax
193
194      puts   <offset info2>    ; show the grade
195      itoa   <offset buffer>, cx
196      puts   <offset buffer>
197
198      puts   <offset info4>
199      push   cx           ; print name
200      call   far ptr printname
201      pop    cx
202
203      puts   <offset info3>
204      itoa   <offset buffer>, ax
205      puts   <offset buffer>
206      outreturn
207
208      jmp    far ptr looping
209
210  case0:
211      cmp     buffer, '0'
212      jne     default
213      jmp     ending
214
215  default:
216      puts   <offset err1>
217      jmp    far ptr looping
218
219  error2:
220      puts   <offset err2>
221      jmp    far ptr looping
222
223  error3:
224      puts   <offset err3>
225      jmp    far ptr looping
```

B. 源代码

```
226
227 looping label far
228     gets    <offset input>, 1      ;wait for an enter
229     jmp    cal_s
230
231 ending:
232 pop    es
233 pop    ds
234 popad
235 ret
236 calcavg endp
237
238 ; \brief calculate a student's avgscore and store it in tab
239 ; \para (2 byte) index of the student in stack
240 ; \return (2 byte) avg score in stack (CAN NOT BE IGNORED)
241 cal_one_avg proc
242 push   eax
243 push   ebx
244 push   ecx
245 push   edx
246     mov   eax, 00h
247     mov   edx, 00h          ; dx stores the avg score
248     mov   ax, [esp+12h]      ; get index (parameter)
249     mov   ebx, 14h          ; multiple by 20
250     mul   ebx
251     mov   ecx, eax         ; ecx as the pointer
252     mov   ax, 0             ; ax as the total sum
253
254     mov   dx, word ptr tab+10[ecx]    ; get chinese score
255     shl   dx, 1
256     add   ax, dx
257
258     mov   dx, word ptr tab+12[ecx]    ; get math score
259     add   ax, dx
260
261     mov   dx, word ptr tab+14[ecx]    ; get english score
262     shr   dx, 1
263     add   ax, dx
264
265     mov   dx, 0h
266     shl   ax, 1
267     mov   bx, 07H            ; divide by 3.5
268     div   bx
269
270     mov   word ptr tab+16[ecx], ax    ; store it to correct position
271     mov   [esp+12h], ax           ; return
272 pop    edx
273 pop    ecx
274 pop    ebx
275 pop    eax
276 ret
277 cal_one_avg endp
278
279 ; \brief refresh the avrage score in table, designed for other modules
280 refresh_avg proc far
```

B. 源代码

```
281 pushad
282 push    ds
283 push    es
284 mov     ax, data
285 mov     ds, ax
286 mov     es, ax
287     mov     cx, 00h
288 refresh_loop:
289     mov     eax, 00h
290     mov     ax, cx
291     mov     bx, 14h
292     mul    bx
293                     ; if the not registered
294     cmp    tab[eax], byte ptr 00h
295     jne    cal_this
296     jmp    cal_this_end
297
298 cal_this:
299     push   cx           ; pass the parameter
300     call   cal_one_avg
301     pop    ax           ; get the result (but do not use)
302
303 cal_this_end:
304     inc    cx
305     cmp    cx, stu_max_num
306     jnz    refresh_loop
307     pop    es
308     pop    ds
309     popad
310     ret
311 refresh_avg endp
312
313 code ends
314 end start
```

排名计算模块 SUBMOD3.ASM :

```
1 .386
2 include macrolib
3
4 public      calcrank
5 public      refresh_rank
6
7 extrn       tab:byte
8 extrn       stu_max_num:abs
9 extrn       name_to_index:far
10 extrn      printname:far
11 extrn      refresh_avg:far
12
13 _stack segment use16 stack "stack"
14 _stack ends
15
16 data segment use16 public "data"
17     input_max_num equ 10
18     infol db ' Submenu - Ranking Calculate %%%%%%%%%%', 0dh, 0ah
19                 db '%%%%%%%%%%%%%%', 0dh, 0ah
```

B. 源代码

```
20          db    '      1) Calculate/Refresh all students      ', 0dh, 0ah
21          db    '      2) Specify a student (by index)      ', 0dh, 0ah
22          db    '      3) Specify a student (by name)      ', 0dh, 0ah
23          db    '      0) Exit                          ', 0dh, 0ah
24          db    '%%%%%%%%%%%%%', 0dh, 0ah
25          db    'Please input your choice >>> $'
26
27      info2  db    'Index $'
28      info3  db    ': Rank is $'
29      info4  db    ': Name is $'
30      info5  db    'Please input the name: $'
31      info6  db    ': Student not registered $'
32      info7  db    'Please input the index: $'
33
34      err1   db    'Unidentified choice, Please re-input ... ', 0dh, 0ah, '$'
35      err2   db    'Student not found ... ', 0dh, 0ah, '$'
36      err3   db    'Index exceeded, range is 0-9', 0dh, 0ah, '$'
37
38      input   db    input_max_num
39      db    ?
40      buffer  db    input_max_num dup(0)
41 data ends
42
43 code segment use16 public "code"
44     assume cs:code, ds:data, ss:_stack, es:data
45 start:
46
47 calcrank proc
48 pushad
49 push    ds
50 push    es
51 mov     ax, data
52 mov     ds, ax
53 mov     es, ax
54 cal_s:
55     call    far ptr refresh_avg      ; refresh the average score before calculating rank
56     cls
57     puts   <offset info1>
58     gets   <offset input>, 2
59
60 case1:
61     cmp    buffer, '1'
62     jne    case2
63
64     mov    cx, 00h
65 loop_a label far
66     mov    eax, 00h
67     mov    ax, cx
68     mov    bx, 14h
69     mul    bx
70
71     cmp    tab[eax], byte ptr 00h
72     jne    case1_cal_this_one
73     puts   <offset info2>
74     itoa   <offset buffer>, cx
```

B. 源代码

```
75     puts    <offset buffer>
76     puts    <offset info6>
77     outreturn
78     jmp     far ptr cal_one_end
79
80 casel_cal_this_one:
81     push   cx           ; pass the parameter
82     call   cal_one_rank
83     pop    ax           ; get the result (but do not use)
84
85     puts    <offset info2>      ; print index
86     itoa   <offset buffer>, cx
87     puts    <offset buffer>
88
89     puts    <offset info4>      ; print name
90     push   cx           ; print name
91     call   far ptr printname
92     pop    cx
93
94     puts    <offset info3>      ; print rank
95     itoa   <offset buffer>, ax
96     puts    <offset buffer>
97     outreturn
98
99 cal_one_end label far
100    inc    cx
101    cmp    cx, stu_max_num
102    jnz   far ptr loop_a
103
104    jmp   far ptr looping
105
106 case2:
107    cmp    buffer, '2'
108    jne   case3
109
110 case2_input:
111    puts    <offset info7>      ; get the index
112    gets   <offset input>, 3
113
114    cmp    input+1h, 0          ; if doesn't get anything
115    jz    case2_input
116
117    atoi   <offset buffer>, 0dh      ; convert to num
118    pop    ecx
119
120    cmp    ecx, 0              ; make sure the index does not exceeds
121    jl    error3
122    cmp    ecx, stu_max_num
123    jge   error3
124
125    mov    eax, 00h
126    mov    ax, cx
127    mov    bx, 14h
128    mul    bx
129
```

B. 源代码

```
130    cmp      tab[eax], byte ptr 00h ; if not registered
131    jne      case2_cal_this_one
132    puts    <offset info2>
133    itoa    <offset buffer>, cx
134    puts    <offset buffer>
135    puts    <offset info6>
136    outreturn
137    jmp     far ptr looping
138
139 case2_cal_this_one:
140    push   cx           ; pass the parameter
141    call   cal_one_rank
142    pop    ax           ; get the result (but do not use)
143
144    puts    <offset info2>      ; print index
145    itoa    <offset buffer>, cx
146    puts    <offset buffer>
147
148    puts    <offset info4>
149    push   cx           ; print name
150    call   far ptr printname
151    pop    cx
152
153    puts    <offset info3>      ; print rank
154    itoa    <offset buffer>, ax
155    puts    <offset buffer>
156    outreturn
157
158    jmp     far ptr looping
159
160 case3:
161    cmp     buffer, '3'
162    jne     case0
163
164 case3_input:
165    puts    <offset info5>
166    gets    <offset input>, 10
167
168    cmp     input+1, byte ptr 00h ; make sure there's input
169    je      case3_input
170
171    mov    si, offset buffer      ; make the buffer end with \0...\0
172    mov    cx, 0ah
173 a0:
174    cmp     ds:[si], byte ptr 0dh
175    je      b0
176    inc    si
177    loop   a0
178 b0:
179    mov    ds:[si], byte ptr 0
180    inc    si
181    loop   b0
182
183    mov    si, offset buffer
184    push   si
```

B. 源代码

```
185      call    far ptr name_to_index
186
187      pop     cx          ; retreive the index into cx
188      cmp     cx, -1h      ; if not found
189      je      error2
190      push    cx
191
192      call    cal_one_rank ; the parameter is already in stack
193      pop     ax
194
195      puts   <offset info2>      ; print index
196      itoa   <offset buffer>, cx
197      puts   <offset buffer>
198
199      puts   <offset info4>
200      push    cx          ; print name
201      call    far ptr printname
202      pop     cx
203
204      puts   <offset info3>      ; print rank
205      itoa   <offset buffer>, ax
206      puts   <offset buffer>
207      outreturn
208
209      jmp    far ptr looping
210
211 case0:
212      cmp     buffer, '0'
213      jne    default
214      jmp    ending
215
216 default:
217      puts   <offset err1>
218      jmp    far ptr looping
219
220 error2:
221      puts   <offset err2>
222      jmp    far ptr looping
223
224 error3:
225      puts   <offset err3>
226      jmp    far ptr looping
227
228 looping label far
229      gets   <offset input>, 1    ;wait for an enter
230      jmp    cal_s
231
232 ending:
233      pop    es
234      pop    ds
235      popad
236      ret
237      calrank endp
238
239 ; \brief calculate a student's ranking and store it in tab
```

B. 源代码

```
240 ; \para (2 byte) index of the student in stack
241 ; \return (2 byte) rank in stack (CAN NOT BE IGNORED)
242 cal_one_rank proc
243 push    eax
244 push    ebx
245 push    ecx
246 push    edx
247 push    esi
248     mov    eax, 00h
249     mov    esi, 00h          ; esi store the rank
250     mov    ax, [esp+16h]      ; get index (parameter)
251     mov    ebx, 14h          ; multiple by 20
252     mul    ebx
253     mov    bx, word ptr tab+16[eax] ; ax stores the current avgscore
254
255     mov    cx, 00h
256 loop_rank:
257     push   bx
258     mov    eax, 00h
259     mov    bx, 14h
260     mov    ax, cx
261     mul    bx
262
263     cmp    tab[eax], byte ptr 0
264     je     not_reged
265
266     pop    bx
267     cmp    word ptr tab+16[eax], bx
268     jg     add_one
269     jmp    loop_rank_tail
270
271 add_one:
272     inc    si
273     jmp    loop_rank_tail
274
275 not_reged:
276     pop    bx
277
278 loop_rank_tail:
279     inc    cx
280     cmp    cx, stu_max_num
281     jne    loop_rank
282
283 loop_end:
284     inc    si
285
286     mov    ax, [esp+16h]      ; get index (parameter)
287     mov    ebx, 14h          ; multiple by 20
288     mul    ebx
289
290     mov    word ptr tab+18[eax], si ; store it to correct position
291     mov    [esp+16h], si          ; return
292     pop    esi
293     pop    edx
294     pop    ecx
```

B. 源代码

```
295 pop    ebx
296 pop    eax
297 ret
298 cal_one_rank endp
299
300 ; \brief refresh all student's rank, but do NOT refresh avgscore automatically, use with care
301 refresh_rank proc far
302 pushad
303 push    ds
304 push    es
305 mov     ax, data
306 mov     ds, ax
307 mov     es, ax
308     mov    cx, 00h
309 refresh_loop:
310     mov    eax, 00h
311     mov    ax, cx
312     mov    bx, 14h
313     mul    bx
314             ; if the not registered
315     cmp    tab[eax], byte ptr 00h
316     jne    cal_this
317     jmp    cal_this_end
318
319 cal_this:
320     push   cx           ; pass the parameter
321     call   cal_one_rank
322     pop    ax           ; get the result (but do not use)
323
324 cal_this_end:
325     inc    cx
326     cmp    cx, stu_max_num
327     jnz    refresh_loop
328     pop    es
329     pop    ds
330     popad
331     ret
332 refresh_rank endp
333
334 code ends
335 end start
```

成绩单输出模块 SUBMOD4.ASM :

```
1 .386
2 include macrolib
3
4 public      printall
5
6 extrn      tab:byte
7 extrn      stu_max_num:abs
8 extrn      refresh_avg:far
9 extrn      refresh_rank:far
10
11 _stack segment use16 stack "stack"
12 _stack ends
```

B. 源代码

```
13
14 data segment use16 public "data"
15     infol  db '+-----+', 0dh, 0ah
16             db '|      Name    | CHN | MAT | ENG | AVG | RANK |', 0dh, 0ah
17     info2  db '+-----+', 0dh, 0ah, '$'
18     line   db '|      |      |      |      |      |      |', 0dh, 0ah, '$'
19     info3  db '|      — | - | - | - | - | — |', 0dh, 0ah, '$'
20
21     buffer db 80 dup(0)
22     numbuff db 10 dup(0)
23
24 data ends
25
26 code segment use16 public "code"
27     assume cs:code, ds:data, ss:_stack, es:data
28 start:
29 printall proc
30 pushad
31 push ds
32 push es
33 mov ax, data
34 mov ds, ax
35 mov es, ax
36     call far ptr refresh_avg
37     call far ptr refresh_rank
38
39     cls
40     puts <offset infol>
41
42     mov cx, 00h
43 print_loop:
44     ; clear the buffer
45     push cx
46     mov cx, 50h
47     lea si, line
48     lea di, buffer
49     rep movsb
50     pop cx
51
52     ; initialize index
53     mov eax, 00h
54     mov ax, cx
55     mov bx, 14h
56     mul bx
57
58     cmp tab[eax], byte ptr 0
59     je not_reged
60
61     ; fill the name
62     lea di, buffer+2
63     lea si, tab[eax]
64 name_s:
65     movsb
66     cmp [si], byte ptr 0
67     jne name_s
```

B. 源代码

```
68      ; fill the chinese grade
69      mov     dx, word ptr tab+10[eax]
70      itoa   <offset numbuff>, dx
71      lea     si, numbuff
72      lea     di, buffer+0fh
73
74  chinese_s:
75      movsb
76      cmp     [si], byte ptr '$'
77      jne    chinese_s
78
79      ; fill the math grade
80      mov     dx, word ptr tab+12[eax]
81      itoa   <offset numbuff>, dx
82      lea     si, numbuff
83      lea     di, buffer+15h
84
85  math_s:
86      movsb
87      cmp     [si], byte ptr '$'
88      jne    math_s
89
90      ; fill the english grade
91      mov     dx, word ptr tab+14[eax]
92      itoa   <offset numbuff>, dx
93      lea     si, numbuff
94      lea     di, buffer+1bh
95
96  english_s:
97      movsb
98      cmp     [si], byte ptr '$'
99      jne    english_s
100
101     ; fill the average grade
102     mov     dx, word ptr tab+16[eax]
103     itoa   <offset numbuff>, dx
104     lea     si, numbuff
105     lea     di, buffer+21h
106
107  average_s:
108      movsb
109      cmp     [si], byte ptr '$'
110      jne    average_s
111
112      ; fill the rank
113     mov     dx, word ptr tab+18[eax]
114     itoa   <offset numbuff>, dx
115     lea     si, numbuff
116     lea     di, buffer+27h
117
118  rank_s:
119      movsb
120      cmp     [si], byte ptr '$'
121      jne    rank_s
122
123  puts   <offset buffer>
124  jmp    print_loop_tail
125
126  not_reged:
```

B. 源代码

```
123     puts    <offset info3>
124
125 print_loop_tail:
126     inc      cx
127     cmp      cx, stu_max_num
128     jne      print_loop
129
130     puts    <offset info2>
131     gets    <offset numbuff>, 1
132
133 pop      es
134 pop      ds
135 popad
136 ret
137 printall endp
138
139 code ends
140 end start
```

宏库 MACROLIB :

```
1 ; \brief put a string starting at addr
2 ; \usage puts <offset var>
3 ; in which var is an variable name
4 puts macro addr
5 push    dx
6 push    ax
7     mov     dx, addr
8     mov     ah, 09h
9     int    21H
10 pop   ax
11 pop   dx
12 endm
13
14 ; \brief get a string and stores it in addr+2
15 ; \usage gets <offset var>, num
16 ; in which var is an variable name, num is the max length
17 gets macro addr, max_num
18 push    bx
19 push    ax
20 push    dx
21     mov     bx, addr
22     mov     [bx], byte ptr max_num
23     mov     dx, bx
24     mov     ah, 0ah
25     int    21h
26     ; output an return ('\r\n')
27     mov     dl, 0dh
28     mov     ah, 02h
29     int    21h
30     mov     dl, 0ah
31     mov     ah, 02h
32     int    21h
33 pop    dx
34 pop    ax
35 pop    bx
```

B. 源代码

```
36  endm
37
38 ; \brief convert a string to a positive number(no greater than 2^32)
39 ; \brief exceeds are abandond
40 ; \return push the result in stack (4 bytes) (CAN NOT BE IGNORED)
41 atoi macro addr, end_char
42 local end, start
43 push    eax          ; preserve space for result
44 push    eax
45 push    ebx
46 push    ecx
47     mov    eax, 00h
48     mov    ecx, 00h
49     mov    cx, addr
50 start:
51     cmp    [ecx], byte ptr end_char
52     je     end
53
54     mov    ebx, 0ah
55     mul    ebx          ; discard the exceed digit
56     mov    bl, [ecx]
57     add    eax, ebx
58     sub    eax, 30h
59     inc    cx
60
61     jmp    start
62 end:
63     mov    [esp+0ch], eax ; return the result
64 pop    ecx
65 pop    ebx
66 pop    eax
67 endm
68
69 ; \brief convert an number (word) to a string, store it in addr ending with '$'
70 itoa macro addr, reg
71 local pushloop, poploop
72 push    reg
73 push    ax
74 push    bx
75 push    cx
76 push    dx
77     mov    ax, reg
78     mov    bl, 0ah
79     mov    cx, 0
80 pushloop:
81     mov    ah, 00h
82     div    bl
83     movzx  dx, ah
84     push   dx
85     inc    cx
86     cmp    al, 0
87     jnz    pushloop
88
89     mov    bx, addr
90 poploop:
```

B. 源代码

```
91      pop     ax
92      add     al, 30h
93      mov     [bx], al
94      inc     bx
95      loop    poploop
96
97      mov     [bx], byte ptr '$'      ; add an ending char
98      pop     dx
99      pop     cx
100     pop    bx
101     pop    ax
102     pop    reg
103     endm
104
105     outreturn macro
106     push   ax
107     push   dx
108     mov    dl, 0dh
109     mov    ah, 02h
110     int    21h
111     mov    dl, 0ah
112     mov    ah, 02h
113     int    21h
114     pop    dx
115     pop    ax
116     endm
117
118     cls  macro
119     push   ax
120     push   bx
121     push   cx
122     push   dx
123     mov    ax, 0600h           ;clear the screen
124     mov    bh, 07h
125     mov    cx, 0000h
126     mov    dx, 184fh
127     int    10h
128     mov    ah, 02h
129     mov    bh, 04h
130     mov    dx, 00h
131     int    10h
132     Pop    dx
133     Pop    cx
134     Pop    bx
135     Pop    ax
136     endm
```

工具函数库 COMMOD.ASM :

```
1 .386
2 include macrolib
3
4 public    name_to_index
5 public    printname
6
7 extern   tab:byte
```

B. 源代码

```
8  extern      stu_max_num:abs
9
10 _stack segment use16 stack "stack"
11 _stack ends
12
13 data segment use16 public "data"
14 data ends
15
16 code segment use16 public "code"
17     assume cs:code, ds:data, ss:_stack, es:data
18 start:
19
20 ; \para (2 byte) address of the name string in stack, string end with \0\0...\0
21 ; \return (2byte) index of the table, in stack, return -1 if not found
22 name_to_index proc far
23 pushad
24 push    ds
25 push    es
26 mov    ax, data
27 mov    ds, ax
28 mov    es, ax
29     mov    ecx, 0h
30 find_s:
31     mov    si, [esp+28h]    ; get the name address
32     mov    eax, 0h
33     mov    ax, cx
34     mov    bx, 14h
35     mul    bx
36     lea    di, tab[eax]
37
38     push    cx           ; store the index
39     mov    cx, 0ah
40
41     repe    cmpsb       ; compare two string
42     je        found        ; if found
43
44     pop    cx           ; if not found this turn, pop the index
45     inc    cx
46     cmp    cx, stu_max_num
47     jne    find_s        ; loop till found/notfound
48
49 notfound:          ; if not found in table
50     mov    [esp+28h], word ptr -1h
51     jmp    ending
52
53 found:
54     pop    cx           ; get the index
55     mov    [esp+28h], cx
56
57 ending:
58     pop    es
59     pop    ds
60     popad
61     ret
62 name_to_index endp
```

B. 源代码

```
63
64
65 ; \brief print the name in the table, without the return
66 ; \parameter index(2 bytes) in stack
67 ; \return index(2 bytes) in stack, can NOT be ignored
68 printname proc far
69 pushad
70 push    ds
71 push    es
72 mov    ax, data
73 mov    ds, ax
74 mov    es, ax
75     mov    eax, 00h
76     mov    ax, [esp+28h]    ; get the index
77     mov    bx, 14h
78     mul    bx
79     mov    ecx, eax
80
81 printchar:
82     cmp    tab[ecx], byte ptr 0
83     je     printend
84
85     mov    dl, tab[ecx]
86     mov    ah, 02h
87     int   21h
88
89     inc    ecx
90     jmp    printchar
91
92 printend:
93     pop    es
94     pop    ds
95     popad
96     ret
97 printname endp
98
99 code ends
100 end start
```

nmake 文件 MAKEFILE :

```
1 AS=TASM
2 ASFLAGS=/c /z /zi /la
3
4 LINK=TLINK
5 LINKFLAGS=/Tde /3 /v
6
7 ALL: STUDENT.EXE
8     echo " BUILDING FINAL TARGET... "
9
10 STUDENT.EXE: MAIN.OBJ SUBMOD1.OBJ SUBMOD2.OBJ SUBMOD3.OBJ COMMOD.OBJ
11     $(LINK) $(LINKFLAGS) MAIN.OBJ+SUBMOD1.OBJ+SUBMOD2.OBJ+SUBMOD3.OBJ+SUBMOD4.OBJ+COMMOD.OBJ
12
13 .ASM.OBJ:
14     $(AS) $(ASFLAGS) *.ASM
```

II.3.2 任务 2

主程序 MAIN.C：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //TODO: 2-0 error
5
6 extern int registration(void);
7 extern int calcavg(void);
8 extern int calcrank(void);
9 extern int printall(void);
10
11 int main(void){
12     int choice, temp;
13     for(;;){
14         system("cls");
15         printf(
16             " Welcome to Student Manage Program //////////////\n"
17             "/////////////////////////////////////////////////\n"
18             "     1) Record Student info\n"
19             "     2) Calculate Average Score\n"
20             "     3) Calculate Ranks\n"
21             "     4) Show Grades and Ranks\n"
22             "     0) Exit\n"
23             "/////////////////////////////////////////////////\n"
24             "Please input your choice >>> ");
25         choice = getchar();
26         /* flush the input stream */
27         while(choice!='\n' && (temp=getchar())!='\r' && temp!='\n');
28         switch(choice){
29             case '1':
30                 registration();
31                 break;
32             case '2':
33                 calcavg();
34                 break;
35             case '3':
36                 calcrank();
37                 break;
38             case '4':
39                 printall();
40                 break;
41             case '0':
42                 printf(
43                     "/////////////////////////////////////////////////\n"
44                     " Thanks for using this system, bye //////////////\n"
45                     "/////////////////////////////////////////////////");
46                 return 0;
47             default:
48                 break;
49         }
50     }
51 }
52 }
```

B. 源代码

```
53 int cprint(char *pointer){  
54     char *temp = pointer;  
55     for(;*pointer!='$'; pointer++);  
56     *pointer=0;  
57     return printf(temp);  
58 }  
59  
60 int cscan(char *pointer, int len){  
61     char *temp=pointer+2, counter=0;  
62     int result = scanf("%[^\\n]", temp);  
63     getchar(); // remove the return  
64     for(;counter<len-1 && *temp != '\\0'; counter++, temp++);  
65     *temp='\\r';  
66     *(pointer+1)=counter; // store the actually got num  
67     return result;  
68 }
```

diff -u Problem1/SUBMOD1.ASM Problem2/SUBMOD1.ASM :

```
1 —— Problem1/SUBMOD1.ASM 2017-04-04 19:36:51.950275657 +0800  
2 +++ Problem2/SUBMOD1.ASM 2017-04-09 01:10:15.614553013 +0800  
3 @@ -1,15 +1,16 @@  
4 .386  
5 include macrolib  
6  
7 -public register  
8 +public _registration  
9  
10 extrn tab:byte  
11 extrn stu_max_num:abs  
12  
13 _stack segment use16 stack "stack"  
14 + db Offh dup(0)  
15 _stack ends  
16  
17 -data segment use16 public "data"  
18 +_DATA segment use16 public "DATA"  
19     input_max_num equ 10  
20     grade_max_num equ 100  
21  
22 @@ -30,17 +31,17 @@  
23  
24     ; store the input info temporarily, for atomic operation  
25     stu_buf db 20 dup(0)  
26 -data ends  
27 +_DATA ends  
28  
29 -code segment use16 public "code"  
30 - assume cs:code, ds:data, ss:_stack, es:data  
31 +_TEXT segment use16 public "CODE"  
32 + assume cs:_TEXT, ds:_DATA, ss:_stack, es:_DATA  
33 start:  
34  
35 -register proc  
36 +_registration proc  
37 pushad
```

B. 源代码

```
38  push    ds
39  push    es
40 -mov      ax, data
41 +mov      ax, _DATA
42  mov      ds, ax
43  mov      es, ax
44  reg_s:
45  @@ -157,7 +158,7 @@
46  pop     ds
47  popad
48  ret
49 -register endp
50 +_registration endp
51
52 -code ends
53 +_TEXT ends
54 end start
```

diff -u Problem1/SUBMOD2.ASM Problem2/SUBMOD2.ASM :

```
1  ----- Problem1/SUBMOD2.ASM      2017-04-04 19:36:21.830171514 +0800
2  +++ Problem2/SUBMOD2.ASM      2017-04-09 01:14:44.887611795 +0800
3  @@ -1,7 +1,7 @@
4  .386
5  include macrolib
6
7 -public    calcavg
8 +public    _calcavg
9  public    refresh_avg
10
11 extrn    tab:byte
12 @@ -10,17 +10,18 @@
13 extrn    printname:far
14
15 _stack segment use16 stack "stack"
16 + db 0ffh dup(0)
17 _stack ends
18
19 -data segment use16 public "data"
20 +_DATA segment use16 public "DATA"
21     input_max_num equ 10
22 -     info1 db ' Submenu - Average Score Calculate %%%%%%', 0dh, 0ah
23 -     db ' %%%%%%%%%%%%%%', 0dh, 0ah
24 +     info1 db ' Submenu - Average Score Calculate //////////////', 0dh, 0ah
25 +     db '//////////////////////////////', 0dh, 0ah
26     db ' 1) Calculate/Refresh all students      ', 0dh, 0ah
27     db ' 2) Specify a student (by index)      ', 0dh, 0ah
28     db ' 3) Specify a student (by name)      ', 0dh, 0ah
29     db ' 0) Exit                            ', 0dh, 0ah
30 -     db ' %%%%%%', 0dh, 0ah
31 +     db ' ////////////////////////////////', 0dh, 0ah
32     db 'Please input your choice >> $'
33
34     info2 db 'Index $'
35 @@ -38,17 +39,17 @@
36     input db input_max_num
```

B. 源代码

```
37         db ?
38     buffer db input_max_num dup(0)
39 -data ends
40 +_DATA ends
41
42 -code segment use16 public "code"
43 - assume cs:code, ds:data, ss:_stack, es:data
44 +_TEXT segment use16 public "CODE"
45 + assume cs:_TEXT, ds:_DATA, ss:_stack, es:_DATA
46 start:
47
48 -calcavg proc
49 +_calcavg proc
50     pushad
51     push    ds
52     push    es
53     mov     ax, data
54     mov     ax, _DATA
55     mov     ds, ax
56     mov     es, ax
57     cal_s:
58     @@ -233,7 +234,7 @@
59     pop    ds
60     popad
61     ret
62 -calcavg endp
63 +_calcavg endp
64
65 ; \brief calculate a student's avgscore and store it in tab
66 ; \para (2 byte) index of the student in stack
67 @@ -281,7 +282,7 @@
68     pushad
69     push    ds
70     push    es
71     mov     ax, data
72     mov     ax, _DATA
73     mov     ds, ax
74     mov     es, ax
75     mov     cx, 00h
76     @@ -310,5 +311,5 @@
77     ret
78     refresh_avg endp
79
80 -code ends
81 +_TEXT ends
82 end start
```

diff -u Problem1/SUBMOD3.ASM Problem2/SUBMOD3.ASM :

```
1 — Problem1/SUBMOD3.ASM      2017-04-04 19:36:35.506885468 +0800
2 +++ Problem2/SUBMOD3.ASM      2017-04-09 01:14:35.497967135 +0800
3 @@ -1,7 +1,7 @@
4 .386
5 include macrolib
6
7 -public      calcrank
```

B. 源代码

```
8  +public      _calcrank
9   public      refresh_rank
10
11  extrn      tab:byte
12 @@ -11,17 +11,18 @@
13  extrn      refresh_avg:far
14
15  _stack segment use16 stack "stack"
16  +  db  0ffh dup(0)
17  _stack ends
18
19  -data segment use16 public "data"
20 +_DATA segment use16 public "DATA"
21     input_max_num    equ 10
22  -  infol  db  ' Submenu - Ranking Calculate %%%%%%%%%%', 0dh, 0ah
23  -          db  '%%%%%%%%%%%%%%%%', 0dh, 0ah
24  +  infol  db  ' Submenu - Ranking Calculate //////////////', 0dh, 0ah
25  +          db  '//////////////////////////////', 0dh, 0ah
26          db  '      1) Calculate/Refresh all students      ', 0dh, 0ah
27          db  '      2) Specify a student (by index)      ', 0dh, 0ah
28          db  '      3) Specify a student (by name)      ', 0dh, 0ah
29          db  '      0) Exit                          ', 0dh, 0ah
30  -          db  '%%%%%%%%%%%%%%%%', 0dh, 0ah
31  +          db  '//////////////////////////////', 0dh, 0ah
32          db  'Please input your choice >> $'
33
34  info2  db  'Index $'
35 @@ -38,17 +39,17 @@
36     input  db  input_max_num
37     db  ?
38     buffer db  input_max_num dup(0)
39 -data ends
40 +_DATA ends
41
42 -code segment use16 public "code"
43 -  assume cs:code, ds:data, ss:_stack, es:data
44 +_TEXT segment use16 public "CODE"
45 +  assume cs:_TEXT, ds:_DATA, ss:_stack, es:_DATA
46 start:
47
48 -calcrank proc
49 +_calcrank proc
50 pushad
51 push  ds
52 push  es
53 mov   ax, data
54 mov   ax, _DATA
55 mov   ds, ax
56 mov   es, ax
57 cal_s:
58 @@ -234,7 +235,7 @@
59 pop   ds
60 popad
61 ret
62 -calcrank endp
```

B. 源代码

```
63 +_calcrank endp
64
65 ; \brief calculate a student's ranking and store it in tab
66 ; \para (2 byte) index of the student in stack
67 @@ -302,7 +303,7 @@
68     pushad
69     push    ds
70     push    es
71     mov      ax, data
72     +mov    ax, _DATA
73     mov      ds, ax
74     mov      es, ax
75     mov      cx, 00h
76 @@ -331,5 +332,5 @@
77     ret
78     refresh_rank endp
79
80 -code ends
81 +_TEXT ends
82 end start
```

diff -u Problem1/SUBMOD4.ASM Problem2/SUBMOD4.ASM :

```
1 —— Problem1/SUBMOD4.ASM      2017-04-04 18:39:49.821934737 +0800
2 +++ Problem2/SUBMOD4.ASM      2017-04-09 01:11:14.030915824 +0800
3 @@ -1,7 +1,7 @@
4 .386
5 include macrolib
6
7 -public      printall
8 +public      _printall
9
10 extrn       tab:byte
11 extrn       stu_max_num:abs
12 @@ -9,9 +9,10 @@
13 extrn       refresh_rank:far
14
15 _stack segment use16 stack "stack"
16 +   db 0ffh dup(0)
17 _stack ends
18
19 -data segment use16 public "data"
20 +_DATA segment use16 public "DATA"
21     info1  db '+'-----+-----+-----+-----+', 0dh, 0ah
22     info1  db '|      Name    | CHN | MAT | ENG | AVG | RANK |', 0dh, 0ah
23     info2  db '+'-----+-----+-----+-----+', 0dh, 0ah, '$'
24 @@ -21,16 +22,16 @@
25     buffer  db 80 dup(0)
26     numbuff db 10 dup(0)
27
28 -data ends
29 +_DATA ends
30
31 -code segment use16 public "code"
32 - assume cs:code, ds:data, ss:_stack, es:data
33 +_TEXT segment use16 public "CODE"
```

B. 源代码

```
34 + assume cs:_TEXT, ds:_DATA, ss:_stack, es:_DATA
35 start:
36 -printall proc
37 +_printall proc
38 pushad
39 push ds
40 push es
41 -mov ax, data
42 +mov ax, _DATA
43 mov ds, ax
44 mov es, ax
45 call far ptr refresh_avg
46 @@ -134,7 +135,7 @@
47 pop ds
48 popad
49 ret
50 -printall endp
51 +_printall endp
52
53 -code ends
54 +_TEXT ends
55 end start
```

diff -u Problem1/COMMOD.ASM Problem2/COMMOD.ASM :

```
1 ----- Problem1/COMMOD.ASM 2017-04-04 19:19:52.286750943 +0800
2 +++ Problem2/COMMOD.ASM 2017-04-09 01:11:32.046547128 +0800
3 @@ -3,18 +3,21 @@
4
5 public name_to_index
6 public printname
7 -
8 -extrn tab:byte
9 -extrn stu_max_num:abs
10 +public tab
11 +public stu_max_num
12
13 _stack segment use16 stack "stack"
14 + db 0ffh dup(0)
15 _stack ends
16
17 -data segment use16 public "data"
18 -data ends
19 +_DATA segment use16 public "DATA"
20 + stu_max_num equ 10
21 + tab dw stu_max_num dup(5 dup(0), -1h, -1h, -1h, -1h, -1h)
22 + clsstr db 'cls',0
23 +_DATA ends
24
25 -code segment use16 public "code"
26 - assume cs:code, ds:data, ss:_stack, es:data
27 +_TEXT segment use16 public "CODE"
28 + assume cs:_TEXT, ds:_DATA, ss:_stack, es:_DATA
29 start:
30
31 ; \para (2 byte) address of the name string in stack, string end with \0\0...\0
```

B. 源代码

```
32    @@ -23,7 +26,7 @@
33    pushad
34    push    ds
35    push    es
36    -mov      ax, data
37    +mov      ax, _DATA
38    mov      ds, ax
39    mov      es, ax
40    mov      ecx, 0h
41    @@ -69,7 +72,7 @@
42    pushad
43    push    ds
44    push    es
45    -mov      ax, data
46    +mov      ax, _DATA
47    mov      ds, ax
48    mov      es, ax
49    mov      eax, 00h
50    @@ -82,7 +85,7 @@
51    cmp      tab[ecx], byte ptr 0
52    je       printend
53
54    - mov      dl, tab[ecx]
55    + mov      dl, byte ptr tab[ecx]
56    mov      ah, 02h
57    int     21h
58
59    @@ -96,6 +99,17 @@
60    ret
61    printname endp
62
63    -code ends
64    +;clear proc
65    +;extrn _system:near
66    +;pushad
67    +; mov      ax, offset clsstr
68    +; push    ax
69    +; call    _system
70    +; pop     ax
71    +;popad
72    +;ret
73    +;clear endp
74    +
75    +_TEXT ends
76    end start
```

宏库 MACROLIB :

```
1 ; \brief put a string starting at addr
2 ; \usage puts <offset var>
3 ; in which var is an variable name
4 puts macro addr
5 pushad
6   extrn  _cprint:near
7   mov    dx, addr           ; call printf
8   push    dx
```

B. 源代码

```
9      call    _cprint
10     pop    ax
11 popad
12 endm
13
14 ; \brief get a string and stores it in addr+2
15 ; \usage puts <offset var>, num
16 ; in which var is an variable name, num is the max length
17 gets macro addr, max_num
18 pushad
19     extrn  _cscan:near
20     mov    ax, max_num
21     push   ax
22     mov    ax, addr
23     push   ax
24     call   _cscan
25     pop    ax
26     pop    ax
27 popad
28 endm
29
30 ; \brief convert a string to a positive number(no greater than 2^32)
31 ; \brief exceeds are abandond
32 ; \return push the result in stack (4 bytes) (CAN NOT BE IGNORED)
33 atoi macro addr, end_char
34 local end, start
35 push  eax          ; preserve space for result
36 push  eax
37 push  ebx
38 push  ecx
39     mov    eax, 00h
40     mov    ecx, 00h
41     mov    cx, addr
42 start:
43     cmp    [ecx], byte ptr end_char
44     je     end
45
46     mov    ebx, 0ah
47     mul    ebx          ; discard the exceed digit
48     mov    bl, [ecx]
49     add    eax, ebx
50     sub    eax, 30h
51     inc    cx
52
53     jmp    start
54 end:
55     mov    [esp+0ch], eax ; return the result
56 pop    ecx
57 pop    ebx
58 pop    eax
59 endm
60
61 ; \brief convert an number (word) to a string, store it in addr ending with '$'
62 itoa macro addr, reg
63 local pushloop, poploop
```

B. 源代码

```
64  push    reg
65  push    ax
66  push    bx
67  push    cx
68  push    dx
69      mov     ax, reg
70      mov     bl, 0ah
71      mov     cx, 0
72  pushloop:
73      mov     ah, 00h
74      div     bl
75      movzx  dx, ah
76      push    dx
77      inc    cx
78      cmp    al, 0
79      jnz    pushloop
80
81      mov    bx, addr
82  poploop:
83      pop    ax
84      add    al, 30h
85      mov    [bx], al
86      inc    bx
87      loop   poploop
88
89      mov    [bx], byte ptr '$'      ; add an ending char
90  pop    dx
91  pop    cx
92  pop    bx
93  pop    ax
94  pop    reg
95  endm
96
97  outreturn macro
98  push    ax
99  push    dx
100     mov    dl, 0dh
101     mov    ah, 02h
102     int    21h
103     mov    dl, 0ah
104     mov    ah, 02h
105     int    21h
106  pop    dx
107  pop    ax
108  endm
109
110  cls  macro
111  push    ax
112  push    bx
113  push    cx
114  push    dx
115      mov    ah, 05h          ; select page
116      mov    al, 00h
117      int    10h
118      mov    ax, 0632h        ; clear the screen
```

B. 源代码

```
119     mov      bh, 07h
120     mov      cx, 0000h
121     mov      dx, 184fh
122     int      10h          ; cursor position
123     mov      ah, 02h
124     mov      bh, 00h
125     mov      dx, 00h
126     int      10h
127     pop      dx
128     pop      cx
129     pop      bx
130     pop      ax
131     endm
```

nmake 文件 MAKEFILE :

```
1 CC=BCC
2 CFLAGS=-3 -v -l3 -lv
3
4 ALL: STUDENT.EXE
5   echo " BUILDING FINAL TARGET... "
6
7 STUDENT.EXE: MAIN.C SUBMOD1.ASM SUBMOD2.ASM SUBMOD3.ASM SUBMOD4.ASM COMMOD.ASM
8   $(CC) $(CFLAGS) -e$@ $?
```

II.4 实验四

II.4.1 任务 1

```

1 .model small
2 .386
3 puts macro addr
4 push dx
5 push ax
6     mov    dx, addr
7     mov    ah, 09h
8     int    21H
9 pop    ax
10 pop   dx
11 endm
12
13 ; \brief convert an number (word) to a string, store it in addr ending with '$'
14 itoa macro addr, reg
15 local pushloop, poploop
16 push   reg
17 push   ax
18 push   bx
19 push   cx
20 push   dx
21     mov    ax, reg
22     mov    bx, 0ah
23     mov    cx, 0
24 pushloop:
25     mov    dx, 00h
26     div    bx
27     push   dx
28     inc    cx
29     cmp    ax, 0
30     jnz    pushloop
31
32     mov    bx, addr
33 poploop:
34     pop   ax
35     add   al, 30h
36     mov   [bx], al
37     inc   bx
38     loop  poploop
39
40     mov   [bx], byte ptr '$'      ; add an ending char
41 pop    dx
42 pop    cx
43 pop    bx
44 pop    ax
45 pop    reg
46 endm
47
48 outreturn macro
49 push   ax
50 push   dx
51     mov    dl, 0dh

```

B. 源代码

```
52     mov      ah, 02h
53     int      21h
54     mov      dl, 0ah
55     mov      ah, 02h
56     int      21h
57     pop      dx
58     pop      ax
59     endm
60
61 .stack 0ffh
62 .data
63     buffer db 10 dup(0)
64 .code
65 start:
66     mov      ax, 00h
67     mov      ds, ax
68
69     ; read directly
70     mov      ecx, ds:[84h]
71     itoa    <offset buffer>, cx
72     puts    <offset buffer>
73     outreturn
74     shr      ecx, 16
75     itoa    <offset buffer>, cx
76     puts    <offset buffer>
77     outreturn
78
79     ; use the system call
80     mov      ah, 35h
81     mov      al, 21h
82     int      21h
83
84     mov      cx, bx
85     itoa    <offset buffer>, cx
86     puts    <offset buffer>
87     outreturn
88
89     mov      cx, es
90     itoa    <offset buffer>, cx
91     puts    <offset buffer>
92     outreturn
93
94     cmp      ecx, eax
95     mov      ah, 4ch
96     int      21h
97 end start
```

II.4.2 任务 2

设置中断的程序：

```

1 .model small
2 .386
3
4 .stack 0ffh
5
6 .code
7 ; the new int 16h interruption
8     old      dw 0, 0
9 new16h:
10    cmp     ah, 2h
11    jz      skip
12
13    int     60h
14    ; modify ah and al
15    cmp     al, 61h
16    jl     not_lower
17
18    cmp     al, 7ah
19    jg     not_lower
20    sub     al, 20h
21
22    iret
23
24 skip:
25    int     60h
26 not_lower:
27    iret
28
29 start:
30    xor     ax, ax
31    mov     ds, ax
32
33    ; move the interruption to another place
34    mov     ax, ds:[16h * 4]
35    mov     old, ax
36    mov     ds:[60H * 4], ax
37
38    mov     ax, ds:[16h * 4 + 2]
39    mov     old+2, ax
40    mov     ds:[60H * 4 + 2], ax
41
42    ; set the new interruption
43    cli
44    mov     word ptr ds:[16h * 4], offset new16h
45    mov     ds:[16h * 4 + 2], cs
46    sti
47
48    mov     dx, offset start + 0fh
49    mov     cl, 4
50    shr     dx, cl
51    add     dx, 10h
52

```

B. 源代码

```
53     mov      al, 0
54     mov      ah, 31h
55     int      21h
56 end start
```

恢复中断的程序

```
1 .model small
2 .386
3
4 .stack 0ffh
5
6 .code
7 start:
8     ; reset the old interruption
9     cli
10    xor     ax, ax
11    mov     ds, ax
12
13    mov     ax, ds:[60h * 4]
14    mov     ds:[16h * 4], ax
15
16    mov     ax, ds:[60h * 4 + 2]
17    mov     ds:[16h * 4 + 2], ax
18    sti
19
20    mov     ah, 4ch
21    int      21h
22 end start
```

II.4.3 任务 3

```
1 .model small
2 .386
3
4 include MACROLIB
5
6 .stack 0ffh
7
8 .data
9     info1  db 'please input the address: $'
10    input   db 10h
11        db 0
12    buffer  db 10h dup(0)
13
14 .code
15 start:
16    mov     ax, @DATA
17    mov     ds, ax
18 again:
19    puts   <offset info1>
20    gets   <offset input>, 2
21    cmp    input+1, 0           ;if doesn't get input
22    je     again
23
24    atoi   <offset buffer>, 0dh
25    pop    eax
26
27    out    70h, al
28    xor    eax, eax
29    in     al, 71h
30
31    itoa16 <offset buffer>, ax
32    puts   <offset buffer>
33
34    mov    ah, 4ch
35    int    21h
36 end start
```

II.4.4 任务 4

主程序 MAIN.ASM :

```

1 .386
2 include macrolib
3
4 ; all submodule share the same table
5 public      tab, stu_max_num
6
7 extrn      register:near
8 extrn      calcavg:near
9 extrn      calcrank:near
10 extrn     printall:near
11
12 _stack segment use16 stack "stack"
13     db 0FFH DUP(0)
14 _stack ends
15
16 data segment use16 public "data"
17     input_max_num    equ 10
18     stu_max_num     equ 10
19     half_max_num    equ 5
20     menu   db 'Welcome to Student Manage Program %%%%%%%%%%', 0dh, 0ah
21     db '%%%%%%%%%%%%%', 0dh, 0ah
22     db '    1) Record Student info          ', 0dh, 0ah
23     db '    2) Calculate Average (not available) ', 0dh, 0ah
24     db '    3) Calculate Ranks (not available)  ', 0dh, 0ah
25     db '    4) Show Grades and Ranks        ', 0dh, 0ah
26     db '    0) Exit                         ', 0dh, 0ah
27     db '%%%%%%%%%%%%%', 0dh, 0ah
28     db 'Please input your choice >>> $'
29
30     infol  db '%%%%%%%%%%%%%', 0dh, 0ah
31     db ' Thanks for using this system, bye %%%%%%', 0dh, 0ah
32     db '%%%%%%%%%%%%%', 0dh, 0ah, '$'
33
34     err1   db 'Unidentified choice, Please re-input ... ', 0dh, 0ah, '$'
35 ; modules do not share the input buffer
36     input   db input_max_num
37     db ?
38     buffer  db input_max_num dup(0)
39
40     encryptbuf db 14h dup(0)
41
42 ; 10 bytes name, 6 bytes score (3*2), 2 bytes average, 2 bytes rank
43     tab    db 76h, 17h, 7bh, 0bh, 63h, 02h, 02h, 02h ,02h
44             db 02h, 5dh, 5dh, 3dh, 3dh, 5ch, 5ch, 03h, 03h, 00h
45
46             db 2dh, 4fh, 2ah, 5eh, 3fh, 3fh, 3fh, 3fh, 3fh
47             db 3fh, 6fh, 6fh, 35h, 35h, 51h, 51h, 04h, 04h, 00h
48
49             db 5bh, 3ch, 5dh, 30h, 5dh, 3ch, 3ch, 3ch, 3ch
50             db 3ch, 58h, 58h, 07h, 07h, 63h, 63h, 01h, 01h, 00h
51
52             db 7fh, 1bh, 7eh, 12h, 66h, 07h, 07h, 07h, 07h

```

B. 源代码

```
53      db 07h, 3bh, 3bh, 7ah, 7ah, 38h, 38h, 06h, 06h, 00h
54
55      db 26h, 43h, 33h, 40h, 29h, 45h, 2ah, 44h, 44h, 44h
56      db 44h, 76h, 76h, 17h, 17h, 41h, 41h, 05h, 05h, 00h
57
58      db 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
59      db 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh
60
61      db 11h, 42h, 2bh, 53h, 26h, 06h, 4eh, 3bh, 3bh, 3bh
62      db 3bh, 59h, 59h, 3ah, 3ah, 62h, 62h, 02h, 02h, 00h
63
64      db 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
65      db 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh
66
67      db 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
68      db 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh
69
70      db 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
71      db 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh
72
73      db 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
74      db 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh, 00h,0ffh
75
76 data ends
77
78 code segment use16 public "code"
79     assume ss:_stack, cs:code, ds:data
80 start:
81     mov ax, data
82     mov ds, ax
83     mov es, ax
84     mov ax, 0504h           ; set current page to 4
85     int 10h
86
87 main:
88     cls
89     puts <offset menu>
90     gets <offset input>, 2
91
92 case1:
93     cmp buffer, '1'
94     jne case4
95
96     call register
97     jmp main
98
99 ;case2:
100 ;    cmp buffer, '2'
101 ;    jne case3
102 ;
103 ;    call calcavg
104 ;    jmp main
105 ;
106 ;case3:
107 ;    cmp buffer, '3'
```

B. 源代码

```
108 ;    jne      case4
109 ;
110 ;    call     calcrank
111 ;    jmp      main
112
113 case4:
114     cmp      buffer, '4'
115     jne      case0
116
117     call     printall
118     jmp      main
119
120 case0:
121     cmp      buffer, '0'
122     jne      default
123
124     puts    <offset info1>
125     mov      ah, 4ch
126     int     21h
127
128 default:
129     puts    <offset err1>
130     jmp      main
131
132 code ends
133 end start
```

输入模块 SUBMOD1.ASM :

```
1 .386
2 include macrolib
3
4 public      register
5
6 extrn       tab:byte
7 extrn       stu_max_num:abs
8
9 _stack segment use16 stack "stack"
10 _stack ends
11
12 data segment use16 public "data"
13     input_max_num    equ 10
14     grade_max_num   equ 100
15
16     info1  db  'Please input the index you want to edit: $'
17     info2  db  'Please input the Name of the student: $'
18     info3  db  'Please input the Chinese score: $'
19     info4  db  'Please input the Math score : $'
20     info5  db  'Please input the English score: $'
21     info6  db  'Student info registered.', 0dh, 0ah, '$'
22
23 ; the number should be equal to stu_max_num
24     err1   db  'Index exceeded, range is 0-9', 0dh, 0ah, '$'
25     err2   db  'Grade exceeded, max is 100', 0dh, 0ah, '$'
26
27     input   db  input_max_num
```

B. 源代码

```
28         db ?
29         buffer db input_max_num dup(0)
30
31         ; store the input info temporarily, for atomic operation
32         stu_buf db 20 dup(0)
33         data ends
34
35         code segment use16 public "code"
36             assume cs:code, ds:data, ss:_stack, es:data
37         start:
38
39         register proc
40         pushad
41         push    ds
42         push    es
43         mov     ax, data
44         mov     ds, ax
45         mov     es, ax
46         reg_s:
47             jmp far ptr anti_track1
48         anti_track71 label far
49
50             jmp far ptr anti_track72
51         anti_track19 label far
52             stuname:
53             jmp far ptr anti_track20
54         anti_track66 label far
55             jg      error2
56             jmp far ptr anti_track67
57         anti_track88 label far
58             encrypt <offset stu_buf>, 14h
59             jmp far ptr anti_track89
60         anti_track103 label far
61
62             jmp far ptr anti_track104
63         anti_track42 label far
64             gets   <offset input>, 4
65             jmp far ptr anti_track43
66         anti_track45 label far
67             je      chinese
68             jmp far ptr anti_track46
69         anti_track17 label far
70             mul    ebx
71             jmp far ptr anti_track18
72         anti_track32 label far
73             dec    cx
74             cmp    cx, 0
75             jne    far ptr clear
76             jmp far ptr anti_track33
77         anti_track85 label far
78             mov    ecx, 14H           ; transport all data (20bytes)
79             jmp far ptr anti_track86
80         anti_track34 label far
81             movzx  ecx, input+1h       ; get the input length
82             jmp far ptr anti_track35
```

B. 源代码

```
83 anti_track4 label far
84           ; if doesn't get anything
85     jmp far ptr anti_track5
86 anti_track94 label far
87           puts    <offset info6>
88     jmp far ptr anti_track95
89 anti_track81 label far
90           mov     word ptr stu_buf+0eh, ax; store english score temporarily
91     jmp far ptr anti_track82
92 anti_track37 label far
93           rep     movsb
94     jmp far ptr anti_track38
95 anti_track106 label far
96           pop    eax
97           ; index not exceeds, but get an error, pop the
98           ; temporarily stored index
99     jmp far ptr anti_track107
100 anti_track73 label far
101           je      english
102     jmp far ptr anti_track74
103 anti_track109 label far
104           jmp    ending
105     jmp far ptr anti_track110
106 anti_track51 label far
107           cmp    eax, grade_max_num
108     jmp far ptr anti_track52
109 anti_track65 label far
110           cmp    eax, grade_max_num
111     jmp far ptr anti_track66
112 anti_track44 label far
113           cmp    input+1, byte ptr 00h   ; make sure there's input
114     jmp far ptr anti_track45
115 anti_track14 label far
116           jge    error1
117     jmp far ptr anti_track15
118 anti_track23 label far
119           jmp    anti_track24
120           gets   <offset input>, 10
121     jmp far ptr anti_track23
122 anti_track55 label far
123           puts   <offset info4>          ; math
124     jmp far ptr anti_track56
125 anti_track80 label far
126           jmp    error2
127     jmp far ptr anti_track81
128 anti_track64 label far
129           jl     error2
130     jmp far ptr anti_track65
131 anti_track25 label far
132           je      stuname
133     jmp far ptr anti_track26
134 anti_track69 label far
135           puts   <offset info5>          ; english
136     jmp far ptr anti_track70
```

B. 源代码

```
137 anti_track77 label far
138     cmp     eax, 0                                ; make sure english score does not exceeds
139     jmp far ptr anti_track78
140 anti_track41 label far
141         puts    <offset info3>                  ; chinese
142     jmp far ptr anti_track42
143 anti_track102 label far
144         jmp     ending
145     jmp far ptr anti_track103
146 anti_track28 label far
147         lea     eax, stu_buf
148     jmp far ptr anti_track29
149 anti_track82 label far
150     pass:
151     jmp far ptr anti_track83
152 anti_track90 label far
153         ; sent the name to correct position
154     jmp far ptr anti_track91
155 anti_track21 label far
156         puts    <offset info2>
157     jmp far ptr anti_track22
158 anti_track8 label far
159         atoi    <offset buffer>, 0dh      ; convert to num
160     jmp far ptr anti_track9
161 anti_track9 label far
162         pop     eax
163     jmp far ptr anti_track10
164 anti_track91 label far
165         lea     esi, stu_buf
166     jmp far ptr anti_track92
167 anti_track60 label far
168
169     jmp far ptr anti_track61
170 anti_track40 label far
171     chinese:
172     jmp far ptr anti_track41
173 anti_track11 label far
174         cmp     eax, 0                                ; make sure the index does not exceeds
175     jmp far ptr anti_track12
176 anti_track35 label far
177         lea     esi, buffer
178     jmp far ptr anti_track36
179 anti_track75 label far
180         atoi    <offset buffer>, 0dh
181     jmp far ptr anti_track76
182 anti_track78 label far
183         jl      error2
184     jmp far ptr anti_track79
185 anti_track63 label far
186         cmp     eax, 0                                ; make sure math score does not exceeds
187     jmp far ptr anti_track64
188 anti_track53 label far
189         mov     word ptr stu_buf+0ah, ax; store chinese score temporarily
190     jmp far ptr anti_track54
191 anti_track10 label far
```

B. 源代码

```
192      jmp far ptr anti_track11
193      anti_track58 label far
194      cmp      input+1, byte ptr 00h ; make sure there's input
195      jmp far ptr anti_track59
196      anti_track99 label far
197      error1:
198      jmp far ptr anti_track100
199      anti_track57 label far
200
201      jmp far ptr anti_track58
202      anti_track31 label far
203      inc      eax
204      jmp far ptr anti_track32
205      anti_track108 label far
206      gets    <offset input>,1      ; wait for return
207      jmp far ptr anti_track109
208      anti_track54 label far
209      math:
210      jmp far ptr anti_track55
211      anti_track105 label far
212      error2:
213      jmp far ptr anti_track106
214      anti_track96 label far
215      jmp      ending
216      jmp far ptr anti_track97
217      anti_track36 label far
218      lea      edi, stu_buf          ; store the name temporarily
219      jmp far ptr anti_track37
220      anti_track18 label far
221      push    eax                  ; store the position
222      jmp far ptr anti_track19
223      anti_track100 label far
224      puts    <offset err1>
225      jmp far ptr anti_track101
226      anti_track98 label far
227      ; index exceeds
228      jmp far ptr anti_track99
229      anti_track89 label far
230
231      jmp far ptr anti_track90
232      anti_track12 label far
233      jl      error1
234      jmp far ptr anti_track13
235      anti_track26 label far
236
237      jmp far ptr anti_track27
238      anti_track49 label far
239      cmp      eax, 0              ; make sure chinese score does not exceeds
240      jmp far ptr anti_track50
241      anti_track104 label far
242      ; grade exceeds
243      jmp far ptr anti_track105
244      anti_track86 label far
245
246
```

B. 源代码

```
247     jmp far ptr anti_track87
248 anti_track93 label far
249         rep      movsb
250     jmp far ptr anti_track94
251 anti_track62 label far
252         pop      eax
253     jmp far ptr anti_track63
254 anti_track97 label far
255
256     jmp far ptr anti_track98
257 anti_track74 label far
258
259     jmp far ptr anti_track75
260 anti_track5 label far
261         cmp      input+1h, 0
262     jmp far ptr anti_track6
263 anti_track7 label far
264
265     jmp far ptr anti_track8
266 anti_track52 label far
267         jg      error2
268     jmp far ptr anti_track53
269 anti_track2 label far
270         gets    <offset input>, 3
271     jmp far ptr anti_track3
272 anti_track6 label far
273         jz      reg_s
274     jmp far ptr anti_track7
275 anti_track13 label far
276         cmp      eax, stu_max_num
277     jmp far ptr anti_track14
278 anti_track87 label far
279         ; encrypt
280     jmp far ptr anti_track88
281 anti_track101 label far
282         gets    <offset input>,1      ; wait for return
283     jmp far ptr anti_track102
284 anti_track83 label far
285         ; if passed all test
286     jmp far ptr anti_track84
287 anti_track68 label far
288         english:
289     jmp far ptr anti_track69
290 anti_track39 label far
291         ; get the scores of the stuendnt
292     jmp far ptr anti_track40
293 anti_track48 label far
294         pop      eax
295     jmp far ptr anti_track49
296 anti_track50 label far
297         jl      error2
298     jmp far ptr anti_track51
299 anti_track84 label far
300         pop      eax      ; get the position back
301     jmp far ptr anti_track85
```

B. 源代码

```
302 anti_track38 label far
303
304     jmp far ptr anti_track39
305 anti_track46 label far
306
307     jmp far ptr anti_track47
308 anti_track107 label far
309         puts    <offset err2>
310     jmp far ptr anti_track108
311 anti_track47 label far
312         atoi    <offset buffer>, 0dh
313     jmp far ptr anti_track48
314 anti_track56 label far
315         gets    <offset input>, 4
316     jmp far ptr anti_track57
317 anti_track27 label far
318         mov     cx, 0ah                      ; clear the buffer
319     jmp far ptr anti_track28
320 anti_track30 label far
321         mov     [eax], byte ptr 00h
322     jmp far ptr anti_track31
323 anti_track61 label far
324         atoi    <offset buffer>, 0dh
325     jmp far ptr anti_track62
326 anti_track24 label far
327         cmp     input+1, byte ptr 00h      ; make sure there's input
328     jmp far ptr anti_track25
329 anti_track3 label far
330
331     jmp far ptr anti_track4
332 anti_track43 label far
333
334     jmp far ptr anti_track44
335 anti_track92 label far
336         lea     edi, tab[eax]
337     jmp far ptr anti_track93
338 anti_track72 label far
339         cmp     input+1, byte ptr 00h      ; make sure there's input
340     jmp far ptr anti_track73
341 anti_track15 label far
342
343     jmp far ptr anti_track16
344 anti_track76 label far
345         pop     eax
346     jmp far ptr anti_track77
347 anti_track67 label far
348         mov     word ptr stu_buf+0ch, ax; store math score temporarily
349     jmp far ptr anti_track68
350 anti_track59 label far
351         je     math
352     jmp far ptr anti_track60
353 anti_track33 label far
354
355     jmp far ptr anti_track34
356 anti_track70 label far
```

B. 源代码

```
357         gets    <offset input>, 4
358         jmp far ptr anti_track71
359 anti_track29 label far
360         clear label far
361         jmp far ptr anti_track30
362 anti_track79 label far
363         cmp     eax, grade_max_num
364         jmp far ptr anti_track80
365 anti_track16 label far
366         mov     ebx, 20           ; get the real position of the student
367         jmp far ptr anti_track17
368 anti_track95 label far
369         gets    <offset input>, 1
370         jmp far ptr anti_track96
371 anti_track1 label far
372         puts   <offset infol>
373         jmp far ptr anti_track2
374 anti_track20 label far
375         ; get the name of the student
376         jmp far ptr anti_track21
377 anti_track110 label far
378
379
380 ending:
381 pop    es
382 pop    ds
383 popad
384 ret
385 register endp
386
387 code ends
388 end start
```

平均分计算模块 SUBMOD2.ASM :

```
1 .386
2 include macrolib
3
4 public      calcavg
5 public      refresh_avg
6
7 extrn       tab:byte
8 extrn       stu_max_num:abs
9 extrn       name_to_index:far
10 extrn      printname:far
11
12 _stack segment use16 stack "stack"
13 _stack ends
14
15 data segment use16 public "data"
16     input_max_num equ 10
17     infol db ' Submenu - Average Score Calculate %%%%%%', 0dh, 0ah
18     db '%%%%%%%%%%%%%', 0dh, 0ah
19     db ' 1) Calculate/Refresh all students ', 0dh, 0ah
20     db ' 2) Specify a student (by index) ', 0dh, 0ah
21     db ' 3) Specify a student (by name) ', 0dh, 0ah
```

B. 源代码

```
22          db '      0) Exit                                ', 0dh, 0ah
23          db '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%', 0dh, 0ah
24          db 'Please input your choice >>> $'
25
26      info2  db 'Index $'
27      info3  db ': average Score is $'
28      info4  db ': Name is $'
29
30      info5  db 'Please input the name: $'
31      info6  db ': Student not registered $'
32      info7  db 'Please input the index: $'
33
34      err1   db 'Unidentified choice, Please re-input ... ', 0dh, 0ah, '$'
35      err2   db 'Student not found ... ', 0dh, 0ah, '$'
36      err3   db 'Index exceeded, range is 0-9', 0dh, 0ah, '$'
37
38      input   db input_max_num
39      db ?
40      buffer  db input_max_num dup(0)
41      decryptbuf db 14h dup(0)
42 data ends
43
44 code segment use16 public "code"
45     assume cs:code, ds:data, ss:_stack, es:data
46 start:
47
48 calcavg proc
49 pushad
50 push    ds
51 push    es
52 mov     ax, data
53 mov     ds, ax
54 mov     es, ax
55 cal_s:
56     cls
57     puts    <offset info1>
58     gets    <offset input>, 2
59
60 case1:
61     cmp     buffer, '1'
62     jne     case2
63
64     mov     cx, 00h
65 loop_a label far
66     mov     eax, 00h
67     mov     ax, cx
68     mov     bx, 14h
69     mul     bx
70             ; if the not registered
71     cmp     tab[eax], byte ptr 00h
72     jne     case1_cal_this_one
73     puts    <offset info2>
74     itoa   <offset buffer>, cx
75     puts    <offset buffer>
76     puts    <offset info6>
```

B. 源代码

```
77     outreturn
78     jmp      far ptr cal_one_end
79
80 case1_cal_this_one:
81     push    cx           ; pass the parameter
82     call    cal_one_avg
83     pop     ax           ; get the result (but do not use)
84
85     puts   <offset info2>
86     itoa   <offset buffer>, cx
87     puts   <offset buffer>
88
89     puts   <offset info4>
90     push    cx           ; print name
91     call    far ptr printname
92     pop     cx
93
94     puts   <offset info3>
95     itoa   <offset buffer>, ax
96     puts   <offset buffer>
97     outreturn
98
99 cal_one_end label far
100    inc    cx
101    cmp    cx, stu_max_num
102    jnz   far ptr loop_a
103
104    jmp   far ptr looping
105
106 case2:
107    cmp   buffer, '2'
108    jne   case3
109
110 case2_input:
111    puts   <offset info7>          ; get the index
112    gets   <offset input>, 3
113
114    cmp   input+1h, 0             ; if doesn't get anything
115    jz    case2_input
116
117    atoi  <offset buffer>, 0dh    ; convert to num
118    pop    ecx
119
120    cmp   ecx, 0                 ; make sure the index does not exceeds
121    jl    error3
122    cmp   ecx, stu_max_num
123    jge  error3
124
125    mov   eax, 00h
126    mov   ax, cx
127    mov   bx, 14h
128    mul   bx
129
130    cmp   tab[eax], byte ptr 00h  ; if not registered
131    jne  case2_cal_this_one
```

B. 源代码

```
132     puts    <offset info2>
133     itoa    <offset buffer>, cx
134     puts    <offset buffer>
135     puts    <offset info6>
136     outreturn
137     jmp     far ptr looping
138
139 case2_cal_this_one:
140     push    cx           ; pass the parameter
141     call    cal_one_avg
142     pop     ax           ; get the result (but do not use)
143
144     puts    <offset info2>
145     itoa    <offset buffer>, cx
146     puts    <offset buffer>
147
148     puts    <offset info4>
149     push    cx           ; print name
150     call    far ptr printname
151     pop     cx
152
153     puts    <offset info3>
154     itoa    <offset buffer>, ax
155     puts    <offset buffer>
156     outreturn
157
158     jmp     far ptr looping
159
160 case3:
161     cmp    buffer, '3'
162     jne    case0
163
164 case3_input:
165     puts    <offset info5>
166     gets   <offset input>, 10
167
168     cmp    input+1, byte ptr 00h ; make sure there's input
169     je     case3_input
170
171     mov    si, offset buffer      ; make the buffer end with \0...\0
172     mov    cx, 0ah
173 a0:
174     cmp    ds:[si], byte ptr 0dh
175     je     b0
176     inc    si
177     loop   a0
178 b0:
179     mov    ds:[si], byte ptr 0
180     inc    si
181     loop   b0
182
183     mov    si, offset buffer
184     push   si
185     call   far ptr name_to_index
186
```

B. 源代码

```
187      pop    cx           ; retreive the index into cx
188      cmp    cx, -1h       ; if not found
189      je     error2
190      push   cx
191
192      call   cal_one_avg  ; the parameter is already in stack
193      pop    ax
194
195      puts   <offset info2>    ; show the grade
196      itoa   <offset buffer>, cx
197      puts   <offset buffer>
198
199      puts   <offset info4>
200      push   cx           ; print name
201      call   far ptr printname
202      pop    cx
203
204      puts   <offset info3>
205      itoa   <offset buffer>, ax
206      puts   <offset buffer>
207      outreturn
208
209      jmp    far ptr looping
210
211 case0:
212      cmp    buffer, '0'
213      jne   default
214      jmp   ending
215
216 default:
217      puts   <offset err1>
218      jmp    far ptr looping
219
220 error2:
221      puts   <offset err2>
222      jmp    far ptr looping
223
224 error3:
225      puts   <offset err3>
226      jmp    far ptr looping
227
228 looping label far
229      gets   <offset input>, 1    ;wait for an enter
230      jmp    cal_s
231
232 ending:
233      pop    es
234      pop    ds
235      popad
236      ret
237 calcavg endp
238
239 ; \brief calculate a student's avgscore and store it in tab
240 ; \para (2 byte) index of the student in stack
241 ; \return (2 byte) avg score in stack (CAN NOT BE IGNORED)
```

B. 源代码

```
242 cal_one_avg proc
243     push    eax
244     push    ebx
245     push    ecx
246     push    edx
247
248     jmp     anti_trace_1_2
249     anti_trace_1_3:           mov     edx, 00h          ; dx stores the avg score
250     jmp     anti_trace_1_4
251     anti_trace_1_5:           mov     ebx, 14h          ; multiple by 20
252     jmp     anti_trace_1_6
253     anti_trace_1_15:
254     jmp     anti_trace_1_16
255     anti_trace_1_36:
256     jmp     anti_trace_1_37
257     anti_trace_1_2:           mov     eax, 00h
258     jmp     anti_trace_1_3
259     anti_trace_1_8:
260     jmp     anti_trace_1_9
261     anti_trace_1_6:           mul     ebx
262     jmp     anti_trace_1_7
263     anti_trace_1_20:          add     ax, dx
264     jmp     anti_trace_1_21
265     anti_trace_1_10:          lea     di, decryptbuf
266     jmp     anti_trace_1_11
267     anti_trace_1_9:           lea     si, tab[ecx]        ; move the student into thde
268             decryptbuf
269     anti_trace_1_11:          push    ecx
270     jmp     anti_trace_1_12
271     anti_trace_1_21:
272     jmp     anti_trace_1_22
273     anti_trace_1_12:          mov     ecx, 14h
274     jmp     anti_trace_1_13
275     anti_trace_1_18:          mov     dx, word ptr decryptbuf+10 ; get chinese score
276     jmp     anti_trace_1_19
277     anti_trace_1_4:           mov     ax, [esp+12h]       ; get index (parameter)
278     jmp     anti_trace_1_5
279     anti_trace_1_13:          rep     movsb
280     jmp     anti_trace_1_14
281     anti_trace_1_22:          mov     dx, word ptr decryptbuf+12 ; get math score
282     jmp     anti_trace_1_23
283     anti_trace_1_26:          shr     dx, 1
284     jmp     anti_trace_1_27
285     anti_trace_1_43:          mov     [esp+12h], ax        ; return
286     jmp     anti_trace_1_44
287     anti_trace_1_7:           mov     ecx, eax
288     jmp     anti_trace_1_8
289     anti_trace_1_24:
290     jmp     anti_trace_1_25
291     anti_trace_1_16:          mov     ax, 0              ; ax as the total sum
292     jmp     anti_trace_1_17
293     anti_trace_1_14:          decrypt <offset decryptbuf>, 14h
294     jmp     anti_trace_1_15
295     anti_trace_1_17:
```

B. 源代码

```
296  jmp      anti_trace_1_18
297  anti_trace_1_19:          shl      dx, 1
298  jmp      anti_trace_1_20
299  anti_trace_1_23:          add     ax, dx
300  jmp      anti_trace_1_24
301  anti_trace_1_25:          mov     dx, word ptr decryptbuf+14 ; get english score
302  jmp      anti_trace_1_26
303  anti_trace_1_28:
304  jmp      anti_trace_1_29
305  anti_trace_1_37:          pop    ecx
306  jmp      anti_trace_1_38
307  anti_trace_1_30:          shl    ax, 1
308  jmp      anti_trace_1_31
309  anti_trace_1_42:
310  jmp      anti_trace_1_43
311  anti_trace_1_40:          mov    ecx, 14h
312  jmp      anti_trace_1_41
313  anti_trace_1_31:          mov    bx, 07H           ; divide by 3.5
314  jmp      anti_trace_1_32
315  anti_trace_1_33:
316  jmp      anti_trace_1_34
317  anti_trace_1_32:          div    bx
318  jmp      anti_trace_1_33
319  anti_trace_1_34:          mov    word ptr decryptbuf+16, ax ; re-encrypt
320  jmp      anti_trace_1_35
321  anti_trace_1_35:          encrypt <offset decryptbuf>, 14h
322  jmp      anti_trace_1_36
323  anti_trace_1_38:          lea    di, tab[ecx]
324  jmp      anti_trace_1_39
325  anti_trace_1_41:          rep    movsb
326  jmp      anti_trace_1_42
327  anti_trace_1_27:          add    ax, dx
328  jmp      anti_trace_1_28
329  anti_trace_1_29:
330  jmp      anti_trace_1_30
331  anti_trace_1_39:          mov    dx, 0h
332  jmp      anti_trace_1_40
333
334  anti_trace_1_44:
335  pop    edx
336  pop    ecx
337  pop    ebx
338  pop    eax
339  ret
340  cal_one_avg endp
341
342  ; \brief refresh the average score in table, designed for other modules
343  refresh_avg proc far
344  pushad
345  push    ds
346  push    es
347  mov     ax, data
348  mov     ds, ax
349  mov     es, ax
350  jmp      anti_trace_2_1
```

B. 源代码

```
351  
352 anti_trace_2_1:           mov     cx, 00h  
353 jmp     anti_trace_2_2  
354 anti_trace_2_4:           mov     ax, cx  
355 jmp     anti_trace_2_5  
356 anti_trace_2_17:          cal_this_end:  
357 jmp     anti_trace_2_18  
358 anti_trace_2_2:           refresh_loop:  
359 jmp     anti_trace_2_3  
360 anti_trace_2_7:           ; if the not registered  
361 jmp     anti_trace_2_8  
362 anti_trace_2_20:          jnz     refresh_loop  
363 jmp     anti_trace_2_21  
364 anti_trace_2_15:          pop     ax           ; get the result (but do not use)  
365 jmp     anti_trace_2_16  
366 anti_trace_2_14:          call    cal_one_avg  
367 jmp     anti_trace_2_15  
368 anti_trace_2_18:          inc    cx  
369 jmp     anti_trace_2_19  
370 anti_trace_2_3:           mov    eax, 00h  
371 jmp     anti_trace_2_4  
372 anti_trace_2_6:           mul    bx  
373 jmp     anti_trace_2_7  
374 anti_trace_2_12:          cal_this:  
375 jmp     anti_trace_2_13  
376 anti_trace_2_8:           cmp    tab[eax], byte ptr 00h  
377 jmp     anti_trace_2_9  
378 anti_trace_2_9:           jne    cal_this  
379 jmp     anti_trace_2_10  
380 anti_trace_2_11:  
381 jmp     anti_trace_2_12  
382 anti_trace_2_5:           mov    bx, 14h  
383 jmp     anti_trace_2_6  
384 anti_trace_2_10:          jmp    cal_this_end  
385 jmp     anti_trace_2_11  
386 anti_trace_2_13:          push   cx           ; pass the parameter  
387 jmp     anti_trace_2_14  
388 anti_trace_2_16:  
389 jmp     anti_trace_2_17  
390 anti_trace_2_19:          cmp    cx, stu_max_num  
391 jmp     anti_trace_2_20  
392  
393 anti_trace_2_21:  
394 pop    es  
395 pop    ds  
396 popad  
397 ret  
398 refresh_avg endp  
399  
400 code ends  
401 end start
```

排名计算模块 SUBMOD3.ASM :

```
1 .386  
2 include macrolib
```

B. 源代码

```
3
4 public      calcrank
5 public      refresh_rank
6
7 extrn      tab:byte
8 extrn      stu_max_num:abs
9 extrn      name_to_index:far
10 extrn      printname:far
11 extrn      refresh_avg:far
12
13 _stack segment use16 stack "stack"
14 _stack ends
15
16 data segment use16 public "data"
17     input_max_num    equ 10
18     info1  db  ' Submenu - Ranking Calculate %%%%%%%%%%%%%%', 0dh, 0ah
19     db  '%%%%%%%%%%%%%%', 0dh, 0ah
20     db  '    1) Calculate/Refresh all students      ', 0dh, 0ah
21     db  '    2) Specify a student (by index)      ', 0dh, 0ah
22     db  '    3) Specify a student (by name)      ', 0dh, 0ah
23     db  '    0) Exit                          ', 0dh, 0ah
24     db  '%%%%%%%%%%%%%%', 0dh, 0ah
25     db  'Please input your choice >>> $'
26
27     info2  db  'Index $'
28     info3  db  ': Rank is $'
29     info4  db  ': Name is $'
30     info5  db  'Please input the name: $'
31     info6  db  ': Student not registered $'
32     info7  db  'Please input the index: $'
33
34     err1   db  'Unidentified choice, Please re-input ... ', 0dh, 0ah, '$'
35     err2   db  'Student not found ... ', 0dh, 0ah, '$'
36     err3   db  'Index exceeded, range is 0-9', 0dh, 0ah, '$'
37
38     input   db  input_max_num
39     db  ?
40     buffer  db  input_max_num dup(0)
41     decryptbuf db  14h dup(0)
42
43 data ends
44
45 code segment use16 public "code"
46     assume cs:code, ds:data, ss:_stack, es:data
47 start:
48
49 calcrank proc
50 pushad
51 push    ds
52 push    es
53 mov     ax, data
54 mov     ds, ax
55 mov     es, ax
56 cal_s:
57     call    far ptr refresh_avg      ; refresh the average score before calculating rank
```

B. 源代码

```
58      cls
59      puts    <offset info1>
60      gets    <offset input>, 2
61
62  case1:
63      cmp     buffer, '1'
64      jne    case2
65
66      mov     cx, 00h
67  loop_a  label far
68      mov     eax, 00h
69      mov     ax, cx
70      mov     bx, 14h
71      mul     bx
72                      ; if the not registered
73      cmp     tab[eax], byte ptr 00h
74      jne    case1_cal_this_one
75      puts    <offset info2>
76      itoa   <offset buffer>, cx
77      puts    <offset buffer>
78      puts    <offset info6>
79      outreturn
80      jmp    far ptr cal_one_end
81
82  case1_cal_this_one:
83      push   cx           ; pass the parameter
84      call   cal_one_rank
85      pop    ax           ; get the result (but do not use)
86
87      puts    <offset info2>       ; print index
88      itoa   <offset buffer>, cx
89      puts    <offset buffer>
90
91      puts    <offset info4>
92      push   cx           ; print name
93      call   far ptr printname
94      pop    cx
95
96      puts    <offset info3>       ; print rank
97      itoa   <offset buffer>, ax
98      puts    <offset buffer>
99      outreturn
100
101 cal_one_end label far
102      inc    cx
103      cmp     cx, stu_max_num
104      jnz    far ptr loop_a
105
106      jmp    far ptr looping
107
108 case2:
109      cmp     buffer, '2'
110      jne    case3
111
112 case2_input:
```

B. 源代码

```
113     puts    <offset info7>          ; get the index
114     gets    <offset input>, 3
115
116     cmp     input+1h, 0            ; if doesn't get anything
117     jz      case2_input
118
119     atoi   <offset buffer>, 0dh    ; convert to num
120     pop    ecx
121
122     cmp     ecx, 0              ; make sure the index does not exceeds
123     jl     error3
124     cmp     ecx, stu_max_num
125     jge   error3
126
127     mov    eax, 00h
128     mov    ax, cx
129     mov    bx, 14h
130     mul    bx
131
132     cmp     tab[eax], byte ptr 00h ; if not registered
133     jne   case2_cal_this_one
134     puts   <offset info2>
135     itoa   <offset buffer>, cx
136     puts   <offset buffer>
137     puts   <offset info6>
138     outreturn
139     jmp    far ptr looping
140
141 case2_cal_this_one:
142     push   cx                  ; pass the parameter
143     call   cal_one_rank
144     pop    ax                  ; get the result (but do not use)
145
146     puts   <offset info2>          ; print index
147     itoa   <offset buffer>, cx
148     puts   <offset buffer>
149
150     puts   <offset info4>
151     push   cx                  ; print name
152     call   far ptr printname
153     pop    cx
154
155     puts   <offset info3>          ; print rank
156     itoa   <offset buffer>, ax
157     puts   <offset buffer>
158     outreturn
159
160     jmp    far ptr looping
161
162 case3:
163     cmp     buffer, '3'
164     jne   case0
165
166 case3_input:
167     puts   <offset info5>
```

B. 源代码

```
168     gets    <offset input>, 10
169
170     cmp     input+1, byte ptr 00h ; make sure there's input
171     je      case3_input
172
173     mov     si, offset buffer ; make the buffer end with \0...\0
174     mov     cx, 0ah
175     a0:
176     cmp     ds:[si], byte ptr 0dh
177     je      b0
178     inc     si
179     loop   a0
180     b0:
181     mov     ds:[si], byte ptr 0
182     inc     si
183     loop   b0
184
185     mov     si, offset buffer
186     push   si
187     call   far ptr name_to_index
188
189     pop    cx ; retreive the index into cx
190     cmp    cx, -1h ; if not found
191     je     error2
192     push   cx
193
194     call   cal_one_rank ; the parameter is already in stack
195     pop    ax
196
197     puts   <offset info2> ; print index
198     itoa   <offset buffer>, cx
199     puts   <offset buffer>
200
201     puts   <offset info4>
202     push   cx ; print name
203     call   far ptr printname
204     pop    cx
205
206     puts   <offset info3> ; print rank
207     itoa   <offset buffer>, ax
208     puts   <offset buffer>
209     outreturn
210
211     jmp    far ptr looping
212
213     case0:
214     cmp     buffer, '0'
215     jne    default
216     jmp    ending
217
218     default:
219     puts   <offset err1>
220     jmp    far ptr looping
221
222     error2:
```

B. 源代码

```
223     puts    <offset err2>
224     jmp     far ptr looping
225
226 error3:
227     puts    <offset err3>
228     jmp     far ptr looping
229
230 looping label far
231     gets   <offset input>, 1      ;wait for an enter
232     jmp     cal_s
233
234 ending:
235 pop    es
236 pop    ds
237 popad
238 ret
239 calcrank endp
240
241 ; \brief calculate a student's ranking and store it in tab
242 ; \para (2 byte) index of the student in stack
243 ; \return (2 byte) rank in stack (CAN NOT BE IGNORED)
244 cal_one_rank proc
245 push   eax
246 push   ebx
247 push   ecx
248 push   edx
249 push   esi
250 push   edi
251     mov    eax, 00h
252     mov    esi, 00h          ; esi store the rank
253     mov    ax, [esp+1ah]      ; get index (parameter)
254     mov    ebx, 14h          ; multiple by 20
255     mul    ebx
256
257     push   esi              ; get the current score
258     push   edi
259     push   ecx
260     lea    si, tab[eax]
261     lea    di, decryptbuf
262     mov    ecx, 14h
263     rep    movsb
264     decrypt <offset decryptbuf>, 14h
265     pop   ecx
266     pop   edi
267     pop   esi
268     mov    bx, word ptr decryptbuf+16 ; bx stores the current avgscore
269
270     mov    cx, 00h
271 loop_rank:
272     push   bx
273     mov    eax, 00h
274     mov    bx, 14h
275     mov    ax, cx
276     mul    bx
277
```

B. 源代码

```
278     cmp      tab[eax], byte ptr 0
279     je       not_reged
280
281     push    esi          ; decrypt and compare
282     push    edi
283     push    ecx
284         lea      si, tab[eax]
285         lea      di, decryptbuf
286         mov      ecx, 14h
287         rep      movsb
288         decrypt <offset decryptbuf>, 14h
289     pop    ecx
290     pop    edi
291     pop    esi
292
293     pop    bx
294     cmp      word ptr decryptbuf+16, bx
295     jg      add_one
296     jmp    loop_rank_tail
297
298 add_one:
299     inc    si
300     jmp    loop_rank_tail
301
302 not_reged:
303     pop    bx
304
305 loop_rank_tail:
306     inc    cx
307     cmp      cx, stu_max_num
308     jne    loop_rank
309
310 loop_end:
311     inc    si
312
313     mov    ax, [esp+1ah]           ; get index (parameter)
314     mov    ebx, 14h                ; multiple by 20
315     mul    ebx
316
317     push   esi          ; load, decrypt and re-encrypt
318     push   edi
319     push   ecx
320         lea      si, tab[eax]
321         lea      di, decryptbuf
322         mov      ecx, 14h
323         rep      movsb
324         decrypt <offset decryptbuf>, 14h
325     pop    ecx
326     pop    edi
327     pop    esi
328
329     mov    word ptr decryptbuf+18, si ; store it to correct position
330     encrypt <offset decryptbuf>, 14h
331     mov    [esp+1ah], si           ; return value
332
```

B. 源代码

```
333     lea      di, tab[eax]           ; send into tab
334     lea      si, decryptbuf
335     mov      ecx, 14h
336     rep      movsb
337     pop      edi
338     pop      esi
339     pop      edx
340     pop      ecx
341     pop      ebx
342     pop      eax
343     ret
344     cal_one_rank endp
345
346 ; \brief refresh all student's rank, but do NOT refresh avgscore automatically, use with care
347 refresh_rank proc far
348     pushad
349     push    ds
350     push    es
351     mov     ax, data
352     mov     ds, ax
353     mov     es, ax
354     mov     cx, 00h
355     refresh_loop:
356     mov     eax, 00h
357     mov     ax, cx
358     mov     bx, 14h
359     mul     bx
360             ; if the not registered
361     cmp     tab[eax], byte ptr 00h
362     jne     cal_this
363     jmp     cal_this_end
364
365     cal_this:
366     push    cx           ; pass the parameter
367     call    cal_one_rank
368     pop     ax           ; get the result (but do not use)
369
370     cal_this_end:
371     inc     cx
372     cmp     cx, stu_max_num
373     jnz     refresh_loop
374     pop     es
375     pop     ds
376     popad
377     ret
378     refresh_rank endp
379
380 code ends
381 end start
```

查询显示模块 SUBMOD4.ASM :

```
1 .386
2 include macrolib
3
4 public      printall
```

B. 源代码

```
5
6  extrn      tab:byte
7  extrn      stu_max_num:abs
8  extrn      refresh_avg:far
9  extrn      refresh_rank:far
10
11 _stack segment use16 stack "stack"
12 _stack ends
13
14 data segment use16 public "data"
15     infol db '+-----+', 0dh, 0ah
16     db '|     Name    | CHN | MAT | ENG | AVG | RANK |', 0dh, 0ah
17     info2 db '+-----+', 0dh, 0ah, '$'
18     line db '|          |   |   |   |   |   |', 0dh, 0ah, '$'
19     info3 db '|      —     | - | - | - | - | — |', 0dh, 0ah, '$'
20     info4 db 'Please input the passwd: $'
21     info5 db 'Wrong passwd, exiting ...$'
22
23     input db 80
24     db 0
25     buffer db 80 dup(0)
26
27     loadbuf db 80 dup(0)
28     numbuff db 10 dup(0)
29
30     encrypted_passwd db 1dh, 06h, 1ah, 11h, 0dh, 44h, 59h, 1dh, 06h, 73h
31
32 data ends
33
34 code segment use16 public "code"
35     assume cs:code, ds:data, ss:_stack, es:data
36 start:
37 printall proc
38 pushad
39 push ds
40 push es
41 mov ax, data
42 mov ds, ax
43 mov es, ax
44
45 cli
46 push ax
47 push offset anti_track2
48 pop ax
49 mov ax, [esp-2]
50 jmp ax
51 sti
52 anti_track2: pop ax
53         call far ptr refresh_avg
54 cli
55 push ax
56 push offset anti_track3
57 pop ax
58 mov ax, [esp-2]
59 jmp ax
```

B. 源代码

```
60  sti
61  anti_track3:    pop     ax
62          call    far ptr refresh_rank
63  cli
64  push   ax
65  push   offset anti_track4
66  pop    ax
67  mov    ax, [esp-2]
68  jmp    ax
69  sti
70  anti_track4:    pop     ax
71          ;==== input the key =====
72  cli
73  push   ax
74  push   offset anti_track5
75  pop    ax
76  mov    ax, [esp-2]
77  jmp    ax
78  sti
79  anti_track5:    pop     ax
80          puts   <offset info4>
81  cli
82  push   ax
83  push   offset anti_track6
84  pop    ax
85  mov    ax, [esp-2]
86  jmp    ax
87  sti
88  anti_track6:    pop     ax
89          gets   <offset input>, 9
90  cli
91  push   ax
92  push   offset anti_track7
93  pop    ax
94  mov    ax, [esp-2]
95  jmp    ax
96  sti
97  anti_track7:    pop     ax
98
99  cli
100 push  ax
101 push  offset anti_track8
102 pop   ax
103 mov   ax, [esp-2]
104 jmp   ax
105 sti
106 anti_track8:    pop     ax
107          xor    ebx, ebx
108 cli
109 push  ax
110 push  offset anti_track9
111 pop   ax
112 mov   ax, [esp-2]
113 jmp   ax
114 sti
```

B. 源代码

```
115 anti_track9:    pop     ax
116                 mov     bl, input+1
117 cli
118 push  ax
119 push  offset anti_track10
120 pop   ax
121 mov   ax, [esp-2]
122 jmp   ax
123 sti
124 anti_track10:   pop     ax
125                 mov     ecx, 0h
126 cli
127 push  ax
128 push  offset anti_track11
129 pop   ax
130 mov   ax, [esp-2]
131 jmp   ax
132 sti
133 anti_track11:   pop     ax
134
135 cli
136 push  ax
137 push  offset anti_track12
138 pop   ax
139 mov   ax, [esp-2]
140 jmp   ax
141 sti
142 anti_track12:   pop     ax
143             append:
144 cli
145 push  ax
146 push  offset anti_track13
147 pop   ax
148 mov   ax, [esp-2]
149 jmp   ax
150 sti
151 anti_track13:   pop     ax
152             ; append the key to 10 bytes
153 cli
154 push  ax
155 push  offset anti_track14
156 pop   ax
157 mov   ax, [esp-2]
158 jmp   ax
159 sti
160 anti_track14:   pop     ax
161             mov     al, buffer[ecx]
162 cli
163 push  ax
164 push  offset anti_track15
165 pop   ax
166 mov   ax, [esp-2]
167 jmp   ax
168 sti
169 anti_track15:   pop     ax
```

B. 源代码

```
170          mov      buffer[ebx], al
171  cli
172  push    ax
173  push    offset anti_track16
174  pop     ax
175  mov     ax, [esp-2]
176  jmp     ax
177  sti
178  anti_track16:  pop    ax
179          inc    cx
180  cli
181  push    ax
182  push    offset anti_track17
183  pop     ax
184  mov     ax, [esp-2]
185  jmp     ax
186  sti
187  anti_track17:  pop    ax
188          inc    bx
189  cli
190  push    ax
191  push    offset anti_track18
192  pop     ax
193  mov     ax, [esp-2]
194  jmp     ax
195  sti
196  anti_track18:  pop    ax
197          cmp    bx, 0ah
198  cli
199  push    ax
200  push    offset anti_track19
201  pop     ax
202  mov     ax, [esp-2]
203  jmp     ax
204  sti
205  anti_track19:  pop    ax
206          jnz    append
207  cli
208  push    ax
209  push    offset anti_track20
210  pop     ax
211  mov     ax, [esp-2]
212  jmp     ax
213  sti
214  anti_track20:  pop    ax
215
216  cli
217  push    ax
218  push    offset anti_track21
219  pop     ax
220  mov     ax, [esp-2]
221  jmp     ax
222  sti
223  anti_track21:  pop    ax
224          ; change the key format
```

B. 源代码

```
225 cli
226 push ax
227 push offset anti_track22
228 pop ax
229 mov ax, [esp-2]
230 jmp ax
231 sti
232 anti_track22: pop ax
233 mov ecx, 00h
234 cli
235 push ax
236 push offset anti_track23
237 pop ax
238 mov ax, [esp-2]
239 jmp ax
240 sti
241 anti_track23: pop ax
242 change:
243 cli
244 push ax
245 push offset anti_track24
246 pop ax
247 mov ax, [esp-2]
248 jmp ax
249 sti
250 anti_track24: pop ax
251 mov al, buffer[ecx]
252 cli
253 push ax
254 push offset anti_track25
255 pop ax
256 mov ax, [esp-2]
257 jmp ax
258 sti
259 anti_track25: pop ax
260 xor al, buffer[ecx+1]
261 cli
262 push ax
263 push offset anti_track26
264 pop ax
265 mov ax, [esp-2]
266 jmp ax
267 sti
268 anti_track26: pop ax
269 mov buffer[ecx], al
270 cli
271 push ax
272 push offset anti_track27
273 pop ax
274 mov ax, [esp-2]
275 jmp ax
276 sti
277 anti_track27: pop ax
278 inc ecx
279 cli
```

B. 源代码

```
280 push    ax
281 push    offset anti_track28
282 pop     ax
283 mov     ax, [esp-2]
284 jmp     ax
285 sti
286 anti_track28:   pop     ax
287           cmp     ecx, 09h
288 cli
289 push    ax
290 push    offset anti_track29
291 pop     ax
292 mov     ax, [esp-2]
293 jmp     ax
294 sti
295 anti_track29:   pop     ax
296           jnz     change
297 cli
298 push    ax
299 push    offset anti_track30
300 pop     ax
301 mov     ax, [esp-2]
302 jmp     ax
303 sti
304 anti_track30:   pop     ax
305
306 cli
307 push    ax
308 push    offset anti_track31
309 pop     ax
310 mov     ax, [esp-2]
311 jmp     ax
312 sti
313 anti_track31:   pop     ax
314           ; compare the passwd
315 cli
316 push    ax
317 push    offset anti_track32
318 pop     ax
319 mov     ax, [esp-2]
320 jmp     ax
321 sti
322 anti_track32:   pop     ax
323           mov ecx, 00h
324 cli
325 push    ax
326 push    offset anti_track33
327 pop     ax
328 mov     ax, [esp-2]
329 jmp     ax
330 sti
331 anti_track33:   pop     ax
332           compare:
333 cli
334 push    ax
```

B. 源代码

```
335 push    offset anti_track34
336 pop     ax
337 mov     ax, [esp-2]
338 jmp     ax
339 sti
340 anti_track34:   pop     ax
341           mov     al, encrypted_passwd[ecx]
342 cli
343 push    ax
344 push    offset anti_track35
345 pop     ax
346 mov     ax, [esp-2]
347 jmp     ax
348 sti
349 anti_track35:   pop     ax
350           cmp     byte ptr buffer[ecx], al
351 cli
352 push    ax
353 push    offset anti_track36
354 pop     ax
355 mov     ax, [esp-2]
356 jmp     ax
357 sti
358 anti_track36:   pop     ax
359           jnz     wrongpasswd
360 cli
361 push    ax
362 push    offset anti_track37
363 pop     ax
364 mov     ax, [esp-2]
365 jmp     ax
366 sti
367 anti_track37:   pop     ax
368           inc     ecx
369 cli
370 push    ax
371 push    offset anti_track38
372 pop     ax
373 mov     ax, [esp-2]
374 jmp     ax
375 sti
376 anti_track38:   pop     ax
377           cmp     ecx, 0ah
378 cli
379 push    ax
380 push    offset anti_track39
381 pop     ax
382 mov     ax, [esp-2]
383 jmp     ax
384 sti
385 anti_track39:   pop     ax
386           jne     compare
387 cli
388 push    ax
389 push    offset anti_track40
```

B. 源代码

```
390 pop    ax
391 mov    ax, [esp-2]
392 jmp    ax
393 sti
394 anti_track40: pop    ax
395         jmp    correctpasswd
396 cli
397 push   ax
398 push   offset anti_track41
399 pop    ax
400 mov    ax, [esp-2]
401 jmp    ax
402 sti
403 anti_track41: pop    ax
404
405 cli
406 push   ax
407 push   offset anti_track42
408 pop    ax
409 mov    ax, [esp-2]
410 jmp    ax
411 sti
412 anti_track42: pop    ax
413         wrongpasswd:
414 cli
415 push   ax
416 push   offset anti_track43
417 pop    ax
418 mov    ax, [esp-2]
419 jmp    ax
420 sti
421 anti_track43: pop    ax
422         puts    <offset info5>
423 cli
424 push   ax
425 push   offset anti_track44
426 pop    ax
427 mov    ax, [esp-2]
428 jmp    ax
429 sti
430 anti_track44: pop    ax
431         mov    ah, 4ch
432 cli
433 push   ax
434 push   offset anti_track45
435 pop    ax
436 mov    ax, [esp-2]
437 jmp    ax
438 sti
439 anti_track45: pop    ax
440         int    21h
441 cli
442 push   ax
443 push   offset anti_track46
444 pop    ax
```

B. 源代码

```
445 mov ax, [esp-2]
446 jmp ax
447 sti
448 anti_track46: pop ax
449
450 cli
451 push ax
452 push offset anti_track47
453 pop ax
454 mov ax, [esp-2]
455 jmp ax
456 sti
457 anti_track47: pop ax
458 correctpasswd:
459 cli
460 push ax
461 push offset anti_track48
462 pop ax
463 mov ax, [esp-2]
464 jmp ax
465 sti
466 anti_track48: pop ax
467 ;=====
468 cli
469 push ax
470 push offset anti_track49
471 pop ax
472 mov ax, [esp-2]
473 jmp ax
474 sti
475 anti_track49: pop ax
476 cls
477 cli
478 push ax
479 push offset anti_track50
480 pop ax
481 mov ax, [esp-2]
482 jmp ax
483 sti
484 anti_track50: pop ax
485 puts <offset infol>
486 cli
487 push ax
488 push offset anti_track51
489 pop ax
490 mov ax, [esp-2]
491 jmp ax
492 sti
493 anti_track51: pop ax
494
495 cli
496 push ax
497 push offset anti_track52
498 pop ax
499 mov ax, [esp-2]
```

B. 源代码

```
500 jmp ax
501 sti
502 anti_track52: pop ax
503           mov cx, 00h
504 cli
505 push ax
506 push offset anti_track53
507 pop ax
508 mov ax, [esp-2]
509 jmp ax
510 sti
511 anti_track53: pop ax
512           print_loop:
513 cli
514 push ax
515 push offset anti_track54
516 pop ax
517 mov ax, [esp-2]
518 jmp ax
519 sti
520 anti_track54: pop ax
521           ; clear the buffer
522 cli
523 push ax
524 push offset anti_track55
525 pop ax
526 mov ax, [esp-2]
527 jmp ax
528 sti
529 anti_track55: pop ax
530           push cx
531 cli
532 push ax
533 push offset anti_track56
534 pop ax
535 mov ax, [esp-2]
536 jmp ax
537 sti
538 anti_track56: pop ax
539           mov cx, 50h
540 cli
541 push ax
542 push offset anti_track57
543 pop ax
544 mov ax, [esp-2]
545 jmp ax
546 sti
547 anti_track57: pop ax
548           lea si, line
549 cli
550 push ax
551 push offset anti_track58
552 pop ax
553 mov ax, [esp-2]
554 jmp ax
```

B. 源代码

```
555 sti
556 anti_track58:    pop     ax
557             lea     di, buffer
558 cli
559 push    ax
560 push    offset anti_track59
561 pop     ax
562 mov     ax, [esp-2]
563 jmp     ax
564 sti
565 anti_track59:    pop     ax
566             rep     movsb
567 cli
568 push    ax
569 push    offset anti_track60
570 pop     ax
571 mov     ax, [esp-2]
572 jmp     ax
573 sti
574 anti_track60:    pop     ax
575             pop     cx
576 cli
577 push    ax
578 push    offset anti_track61
579 pop     ax
580 mov     ax, [esp-2]
581 jmp     ax
582 sti
583 anti_track61:    pop     ax
584
585 cli
586 push    ax
587 push    offset anti_track62
588 pop     ax
589 mov     ax, [esp-2]
590 jmp     ax
591 sti
592 anti_track62:    pop     ax
593             ; initialize index
594 cli
595 push    ax
596 push    offset anti_track63
597 pop     ax
598 mov     ax, [esp-2]
599 jmp     ax
600 sti
601 anti_track63:    pop     ax
602             mov     eax, 00h
603 cli
604 push    ax
605 push    offset anti_track64
606 pop     ax
607 mov     ax, [esp-2]
608 jmp     ax
609 sti
```

B. 源代码

```
610 anti_track64:    pop     ax
611                 mov     ax, cx
612 cli
613 push    ax
614 push    offset anti_track65
615 pop     ax
616 mov     ax, [esp-2]
617 jmp     ax
618 sti
619 anti_track65:    pop     ax
620                 mov     bx, 14h
621 cli
622 push    ax
623 push    offset anti_track66
624 pop     ax
625 mov     ax, [esp-2]
626 jmp     ax
627 sti
628 anti_track66:    pop     ax
629                 mul     bx
630 cli
631 push    ax
632 push    offset anti_track67
633 pop     ax
634 mov     ax, [esp-2]
635 jmp     ax
636 sti
637 anti_track67:    pop     ax
638
639 cli
640 push    ax
641 push    offset anti_track68
642 pop     ax
643 mov     ax, [esp-2]
644 jmp     ax
645 sti
646 anti_track68:    pop     ax
647                 ; perform a decrypt
648 cli
649 push    ax
650 push    offset anti_track69
651 pop     ax
652 mov     ax, [esp-2]
653 jmp     ax
654 sti
655 anti_track69:    pop     ax
656                 push    cx
657 cli
658 push    ax
659 push    offset anti_track70
660 pop     ax
661 mov     ax, [esp-2]
662 jmp     ax
663 sti
664 anti_track70:    pop     ax
```

B. 源代码

```
665          mov      cx, 14h
666 cli
667 push    ax
668 push    offset anti_track71
669 pop     ax
670 mov     ax, [esp-2]
671 jmp     ax
672 sti
673 anti_track71:  pop    ax
674           lea    di, loadbuf
675 cli
676 push    ax
677 push    offset anti_track72
678 pop     ax
679 mov     ax, [esp-2]
680 jmp     ax
681 sti
682 anti_track72:  pop    ax
683           lea    si, tab[eax]
684 cli
685 push    ax
686 push    offset anti_track73
687 pop     ax
688 mov     ax, [esp-2]
689 jmp     ax
690 sti
691 anti_track73:  pop    ax
692           rep    movsb
693 cli
694 push    ax
695 push    offset anti_track74
696 pop     ax
697 mov     ax, [esp-2]
698 jmp     ax
699 sti
700 anti_track74:  pop    ax
701           decrypt <offset loadbuf>, 14h
702 cli
703 push    ax
704 push    offset anti_track75
705 pop     ax
706 mov     ax, [esp-2]
707 jmp     ax
708 sti
709 anti_track75:  pop    ax
710           pop    cx
711 cli
712 push    ax
713 push    offset anti_track76
714 pop     ax
715 mov     ax, [esp-2]
716 jmp     ax
717 sti
718 anti_track76:  pop    ax
719
```

B. 源代码

```
720 cli
721 push ax
722 push offset anti_track77
723 pop ax
724 mov ax, [esp-2]
725 jmp ax
726 sti
727 anti_track77: pop ax
728 ; if student not registered
729 cli
730 push ax
731 push offset anti_track78
732 pop ax
733 mov ax, [esp-2]
734 jmp ax
735 sti
736 anti_track78: pop ax
737 cmp loadbuf, byte ptr 0
738 cli
739 push ax
740 push offset anti_track79
741 pop ax
742 mov ax, [esp-2]
743 jmp ax
744 sti
745 anti_track79: pop ax
746 je not_reged
747 cli
748 push ax
749 push offset anti_track80
750 pop ax
751 mov ax, [esp-2]
752 jmp ax
753 sti
754 anti_track80: pop ax
755
756 cli
757 push ax
758 push offset anti_track81
759 pop ax
760 mov ax, [esp-2]
761 jmp ax
762 sti
763 anti_track81: pop ax
764 ; fill the name
765 cli
766 push ax
767 push offset anti_track82
768 pop ax
769 mov ax, [esp-2]
770 jmp ax
771 sti
772 anti_track82: pop ax
773 lea di, buffer+2
774 cli
```

B. 源代码

```
775 push    ax
776 push    offset anti_track83
777 pop     ax
778 mov     ax, [esp-2]
779 jmp     ax
780 sti
781 anti_track83:   pop     ax
782             lea     si, loadbuf
783 cli
784 push    ax
785 push    offset anti_track84
786 pop     ax
787 mov     ax, [esp-2]
788 jmp     ax
789 sti
790 anti_track84:   pop     ax
791             name_s:
792 cli
793 push    ax
794 push    offset anti_track85
795 pop     ax
796 mov     ax, [esp-2]
797 jmp     ax
798 sti
799 anti_track85:   pop     ax
800             movsb
801 cli
802 push    ax
803 push    offset anti_track86
804 pop     ax
805 mov     ax, [esp-2]
806 jmp     ax
807 sti
808 anti_track86:   pop     ax
809             cmp     [si], byte ptr 0
810 cli
811 push    ax
812 push    offset anti_track87
813 pop     ax
814 mov     ax, [esp-2]
815 jmp     ax
816 sti
817 anti_track87:   pop     ax
818             jne     name_s
819 cli
820 push    ax
821 push    offset anti_track88
822 pop     ax
823 mov     ax, [esp-2]
824 jmp     ax
825 sti
826 anti_track88:   pop     ax
827
828 cli
829 push    ax
```

B. 源代码

```
830 push    offset anti_track89
831 pop     ax
832 mov     ax, [esp-2]
833 jmp     ax
834 sti
835 anti_track89:   pop      ax
836           ; fill the chinese grade
837 cli
838 push    ax
839 push    offset anti_track90
840 pop     ax
841 mov     ax, [esp-2]
842 jmp     ax
843 sti
844 anti_track90:   pop      ax
845           mov      dx, word ptr loadbuf+10
846 cli
847 push    ax
848 push    offset anti_track91
849 pop     ax
850 mov     ax, [esp-2]
851 jmp     ax
852 sti
853 anti_track91:   pop      ax
854           itoa    <offset numbuff>, dx
855 cli
856 push    ax
857 push    offset anti_track92
858 pop     ax
859 mov     ax, [esp-2]
860 jmp     ax
861 sti
862 anti_track92:   pop      ax
863           lea     si, numbuff
864 cli
865 push    ax
866 push    offset anti_track93
867 pop     ax
868 mov     ax, [esp-2]
869 jmp     ax
870 sti
871 anti_track93:   pop      ax
872           lea     di, buffer+0fh
873 cli
874 push    ax
875 push    offset anti_track94
876 pop     ax
877 mov     ax, [esp-2]
878 jmp     ax
879 sti
880 anti_track94:   pop      ax
881           chinese_s:
882 cli
883 push    ax
884 push    offset anti_track95
```

B. 源代码

```
885 pop    ax
886 mov    ax, [esp-2]
887 jmp    ax
888 sti
889 anti_track95: pop    ax
890         movsb
891 cli
892 push   ax
893 push   offset anti_track96
894 pop    ax
895 mov    ax, [esp-2]
896 jmp    ax
897 sti
898 anti_track96: pop    ax
899         cmp    [si], byte ptr '$'
900 cli
901 push   ax
902 push   offset anti_track97
903 pop    ax
904 mov    ax, [esp-2]
905 jmp    ax
906 sti
907 anti_track97: pop    ax
908         jne    chinese_s
909 cli
910 push   ax
911 push   offset anti_track98
912 pop    ax
913 mov    ax, [esp-2]
914 jmp    ax
915 sti
916 anti_track98: pop    ax
917
918 cli
919 push   ax
920 push   offset anti_track99
921 pop    ax
922 mov    ax, [esp-2]
923 jmp    ax
924 sti
925 anti_track99: pop    ax
926         ; fill the math grade
927 cli
928 push   ax
929 push   offset anti_track100
930 pop    ax
931 mov    ax, [esp-2]
932 jmp    ax
933 sti
934 anti_track100: pop    ax
935         mov    dx, word ptr loadbuf+12
936 cli
937 push   ax
938 push   offset anti_track101
939 pop    ax
```

B. 源代码

```
940 mov ax, [esp-2]
941 jmp ax
942 sti
943 anti_track101: pop ax
944 itoa <offset numbuff>, dx
945 cli
946 push ax
947 push offset anti_track102
948 pop ax
949 mov ax, [esp-2]
950 jmp ax
951 sti
952 anti_track102: pop ax
953 lea si, numbuff
954 cli
955 push ax
956 push offset anti_track103
957 pop ax
958 mov ax, [esp-2]
959 jmp ax
960 sti
961 anti_track103: pop ax
962 lea di, buffer+15h
963 cli
964 push ax
965 push offset anti_track104
966 pop ax
967 mov ax, [esp-2]
968 jmp ax
969 sti
970 anti_track104: pop ax
971 math_s:
972 cli
973 push ax
974 push offset anti_track105
975 pop ax
976 mov ax, [esp-2]
977 jmp ax
978 sti
979 anti_track105: pop ax
980 movsb
981 cli
982 push ax
983 push offset anti_track106
984 pop ax
985 mov ax, [esp-2]
986 jmp ax
987 sti
988 anti_track106: pop ax
989 cmp [si], byte ptr '$'
990 cli
991 push ax
992 push offset anti_track107
993 pop ax
994 mov ax, [esp-2]
```

B. 源代码

```
995 jmp ax
996 sti
997 anti_track107: pop ax
998 jne math_s
999 cli
1000 push ax
1001 push offset anti_track108
1002 pop ax
1003 mov ax, [esp-2]
1004 jmp ax
1005 sti
1006 anti_track108: pop ax
1007
1008 cli
1009 push ax
1010 push offset anti_track109
1011 pop ax
1012 mov ax, [esp-2]
1013 jmp ax
1014 sti
1015 anti_track109: pop ax
1016 ; fill the english grade
1017 cli
1018 push ax
1019 push offset anti_track110
1020 pop ax
1021 mov ax, [esp-2]
1022 jmp ax
1023 sti
1024 anti_track110: pop ax
1025 mov dx, word ptr loadbuf+14
1026 cli
1027 push ax
1028 push offset anti_track111
1029 pop ax
1030 mov ax, [esp-2]
1031 jmp ax
1032 sti
1033 anti_track111: pop ax
1034 itoa <offset numbuff>, dx
1035 cli
1036 push ax
1037 push offset anti_track112
1038 pop ax
1039 mov ax, [esp-2]
1040 jmp ax
1041 sti
1042 anti_track112: pop ax
1043 lea si, numbuff
1044 cli
1045 push ax
1046 push offset anti_track113
1047 pop ax
1048 mov ax, [esp-2]
1049 jmp ax
```

B. 源代码

```
1050 sti
1051 anti_track113: pop      ax
1052           lea      di, buffer+1bh
1053 cli
1054 push   ax
1055 push   offset anti_track114
1056 pop    ax
1057 mov    ax, [esp-2]
1058 jmp    ax
1059 sti
1060 anti_track114: pop      ax
1061           english_s:
1062 cli
1063 push   ax
1064 push   offset anti_track115
1065 pop    ax
1066 mov    ax, [esp-2]
1067 jmp    ax
1068 sti
1069 anti_track115: pop      ax
1070           movsb
1071 cli
1072 push   ax
1073 push   offset anti_track116
1074 pop    ax
1075 mov    ax, [esp-2]
1076 jmp    ax
1077 sti
1078 anti_track116: pop      ax
1079           cmp      [si], byte ptr '$'
1080 cli
1081 push   ax
1082 push   offset anti_track117
1083 pop    ax
1084 mov    ax, [esp-2]
1085 jmp    ax
1086 sti
1087 anti_track117: pop      ax
1088           jne      english_s
1089 cli
1090 push   ax
1091 push   offset anti_track118
1092 pop    ax
1093 mov    ax, [esp-2]
1094 jmp    ax
1095 sti
1096 anti_track118: pop      ax
1097
1098 cli
1099 push   ax
1100 push   offset anti_track119
1101 pop    ax
1102 mov    ax, [esp-2]
1103 jmp    ax
1104 sti
```

B. 源代码

```
1105 anti_track119: pop      ax
1106           ; fill the avreage grade
1107 cli
1108 push    ax
1109 push    offset anti_track120
1110 pop     ax
1111 mov     ax, [esp-2]
1112 jmp     ax
1113 sti
1114 anti_track120: pop      ax
1115           mov     dx, word ptr loadbuf+16
1116 cli
1117 push    ax
1118 push    offset anti_track121
1119 pop     ax
1120 mov     ax, [esp-2]
1121 jmp     ax
1122 sti
1123 anti_track121: pop      ax
1124           itoa   <offset numbuff>, dx
1125 cli
1126 push    ax
1127 push    offset anti_track122
1128 pop     ax
1129 mov     ax, [esp-2]
1130 jmp     ax
1131 sti
1132 anti_track122: pop      ax
1133           lea    si, numbuff
1134 cli
1135 push    ax
1136 push    offset anti_track123
1137 pop     ax
1138 mov     ax, [esp-2]
1139 jmp     ax
1140 sti
1141 anti_track123: pop      ax
1142           lea    di, buffer+21h
1143 cli
1144 push    ax
1145 push    offset anti_track124
1146 pop     ax
1147 mov     ax, [esp-2]
1148 jmp     ax
1149 sti
1150 anti_track124: pop      ax
1151           average_s:
1152 cli
1153 push    ax
1154 push    offset anti_track125
1155 pop     ax
1156 mov     ax, [esp-2]
1157 jmp     ax
1158 sti
1159 anti_track125: pop      ax
```

B. 源代码

```
1160          movsb
1161 cli
1162 push    ax
1163 push    offset anti_track126
1164 pop     ax
1165 mov     ax, [esp-2]
1166 jmp     ax
1167 sti
1168 anti_track126: pop    ax
1169           cmp    [si], byte ptr '$'
1170 cli
1171 push    ax
1172 push    offset anti_track127
1173 pop     ax
1174 mov     ax, [esp-2]
1175 jmp     ax
1176 sti
1177 anti_track127: pop    ax
1178           jne    average_s
1179 cli
1180 push    ax
1181 push    offset anti_track128
1182 pop     ax
1183 mov     ax, [esp-2]
1184 jmp     ax
1185 sti
1186 anti_track128: pop    ax
1187
1188 cli
1189 push    ax
1190 push    offset anti_track129
1191 pop     ax
1192 mov     ax, [esp-2]
1193 jmp     ax
1194 sti
1195 anti_track129: pop    ax
1196           ; fill the rank
1197 cli
1198 push    ax
1199 push    offset anti_track130
1200 pop     ax
1201 mov     ax, [esp-2]
1202 jmp     ax
1203 sti
1204 anti_track130: pop    ax
1205           mov    dx, word ptr loadbuf+18
1206 cli
1207 push    ax
1208 push    offset anti_track131
1209 pop     ax
1210 mov     ax, [esp-2]
1211 jmp     ax
1212 sti
1213 anti_track131: pop    ax
1214           itoa  <offset numbuff>, dx
```

B. 源代码

```
1215 cli
1216 push ax
1217 push offset anti_track132
1218 pop ax
1219 mov ax, [esp-2]
1220 jmp ax
1221 sti
1222 anti_track132: pop ax
1223 lea si, numbuff
1224 cli
1225 push ax
1226 push offset anti_track133
1227 pop ax
1228 mov ax, [esp-2]
1229 jmp ax
1230 sti
1231 anti_track133: pop ax
1232 lea di, buffer+27h
1233 cli
1234 push ax
1235 push offset anti_track134
1236 pop ax
1237 mov ax, [esp-2]
1238 jmp ax
1239 sti
1240 anti_track134: pop ax
1241 rank_s:
1242 cli
1243 push ax
1244 push offset anti_track135
1245 pop ax
1246 mov ax, [esp-2]
1247 jmp ax
1248 sti
1249 anti_track135: pop ax
1250 movsb
1251 cli
1252 push ax
1253 push offset anti_track136
1254 pop ax
1255 mov ax, [esp-2]
1256 jmp ax
1257 sti
1258 anti_track136: pop ax
1259 cmp [si], byte ptr '$'
1260 cli
1261 push ax
1262 push offset anti_track137
1263 pop ax
1264 mov ax, [esp-2]
1265 jmp ax
1266 sti
1267 anti_track137: pop ax
1268 jne rank_s
1269 cli
```

B. 源代码

```
1270 push    ax
1271 push    offset anti_track138
1272 pop     ax
1273 mov     ax, [esp-2]
1274 jmp     ax
1275 sti
1276 anti_track138: pop     ax
1277
1278 cli
1279 push    ax
1280 push    offset anti_track139
1281 pop     ax
1282 mov     ax, [esp-2]
1283 jmp     ax
1284 sti
1285 anti_track139: pop     ax
1286         puts    <offset buffer>
1287 cli
1288 push    ax
1289 push    offset anti_track140
1290 pop     ax
1291 mov     ax, [esp-2]
1292 jmp     ax
1293 sti
1294 anti_track140: pop     ax
1295         jmp    print_loop_tail
1296 cli
1297 push    ax
1298 push    offset anti_track141
1299 pop     ax
1300 mov     ax, [esp-2]
1301 jmp     ax
1302 sti
1303 anti_track141: pop     ax
1304
1305 cli
1306 push    ax
1307 push    offset anti_track142
1308 pop     ax
1309 mov     ax, [esp-2]
1310 jmp     ax
1311 sti
1312 anti_track142: pop     ax
1313         not_reged:
1314 cli
1315 push    ax
1316 push    offset anti_track143
1317 pop     ax
1318 mov     ax, [esp-2]
1319 jmp     ax
1320 sti
1321 anti_track143: pop     ax
1322         puts    <offset info3>
1323 cli
1324 push    ax
```

B. 源代码

```
1325 push    offset anti_track144
1326 pop     ax
1327 mov     ax, [esp-2]
1328 jmp     ax
1329 sti
1330 anti_track144: pop      ax
1331
1332 cli
1333 push    ax
1334 push    offset anti_track145
1335 pop     ax
1336 mov     ax, [esp-2]
1337 jmp     ax
1338 sti
1339 anti_track145: pop      ax
1340         print_loop_tail:
1341 cli
1342 push    ax
1343 push    offset anti_track146
1344 pop     ax
1345 mov     ax, [esp-2]
1346 jmp     ax
1347 sti
1348 anti_track146: pop      ax
1349         inc      cx
1350 cli
1351 push    ax
1352 push    offset anti_track147
1353 pop     ax
1354 mov     ax, [esp-2]
1355 jmp     ax
1356 sti
1357 anti_track147: pop      ax
1358         cmp      cx, stu_max_num
1359 cli
1360 push    ax
1361 push    offset anti_track148
1362 pop     ax
1363 mov     ax, [esp-2]
1364 jmp     ax
1365 sti
1366 anti_track148: pop      ax
1367         jne      print_loop
1368 cli
1369 push    ax
1370 push    offset anti_track149
1371 pop     ax
1372 mov     ax, [esp-2]
1373 jmp     ax
1374 sti
1375 anti_track149: pop      ax
1376
1377 cli
1378 push    ax
1379 push    offset anti_track150
```

B. 源代码

```
1380 pop    ax
1381 mov    ax, [esp-2]
1382 jmp    ax
1383 sti
1384 anti_track150: pop    ax
1385           puts    <offset info2>
1386 cli
1387 push   ax
1388 push   offset anti_track151
1389 pop    ax
1390 mov    ax, [esp-2]
1391 jmp    ax
1392 sti
1393 anti_track151: pop    ax
1394           gets    <offset numbuff>, 1
1395
1396 pop    es
1397 pop    ds
1398 popad
1399 ret
1400 printall endp
1401
1402 code ends
1403 end start
```

宏库 MACROLIB :

```
1 ; \brief put a string starting at addr
2 ; \usage puts <offset var>
3 ; in which var is an variable name
4 puts macro addr
5 push   dx
6 push   ax
7     mov    dx, addr
8     mov    ah, 09h
9     int    21H
10    pop   ax
11    pop   dx
12 endm
13
14 ; \brief get a string and stores it in addr+2
15 ; \usage gets <offset var>, num
16 ; in which var is an variable name, num is the max length
17 gets macro addr, max_num
18 push   bx
19 push   ax
20 push   dx
21     mov    bx, addr
22     mov    [bx], byte ptr max_num
23     mov    dx, bx
24     mov    ah, 0ah
25     int    21h
26 ; output an return ('\r\n')
27     mov    dl, 0dh
28     mov    ah, 02h
29     int    21h
```

B. 源代码

```
30     mov      dl, 0ah
31     mov      ah, 02h
32     int      21h
33     pop      dx
34     pop      ax
35     pop      bx
36     endm
37
38 ; \brief convert a string to a positive number(no greater than 2^32)
39 ; \brief exceeds are abandoned
40 ; \return push the result in stack (4 bytes) (CAN NOT BE IGNORED)
41 atoi macro addr, end_char
42 local end, start
43 push   eax           ; preserve space for result
44 push   eax
45 push   ebx
46 push   ecx
47     mov      eax, 00h
48     mov      ecx, 00h
49     mov      cx, addr
50 start:
51     cmp      [ecx], byte ptr end_char
52     je      end
53
54     mov      ebx, 0ah
55     mul      ebx           ; discard the exceed digit
56     mov      bl, [ecx]
57     add      eax, ebx
58     sub      eax, 30h
59     inc      cx
60
61     jmp      start
62 end:
63     mov      [esp+0ch], eax ; return the result
64     pop      ecx
65     pop      ebx
66     pop      eax
67     endm
68
69 ; \brief convert an number (word) to a string, store it in addr ending with '$'
70 itoa macro addr, reg
71 local pushloop, poploop
72 push   reg
73 push   ax
74 push   bx
75 push   cx
76 push   dx
77     mov      ax, reg
78     mov      bl, 0ah
79     mov      cx, 0
80 pushloop:
81     mov      ah, 00h
82     div      bl
83     movzx  dx, ah
84     push   dx
```

B. 源代码

```
85      inc     cx
86      cmp     al, 0
87      jnz     pushloop
88
89      mov     bx, addr
90  poploop:
91      pop     ax
92      add     al, 30h
93      mov     [bx], al
94      inc     bx
95      loop    poploop
96
97      mov     [bx], byte ptr '$'      ; add an ending char
98      pop     dx
99      pop     cx
100     pop    bx
101     pop    ax
102     pop    reg
103     endm
104
105     outreturn macro
106     push   ax
107     push   dx
108     mov    dl, 0dh
109     mov    ah, 02h
110     int    21h
111     mov    dl, 0ah
112     mov    ah, 02h
113     int    21h
114     pop    dx
115     pop    ax
116     endm
117
118     cls macro
119     push   ax
120     push   bx
121     push   cx
122     push   dx
123     mov    ax, 0600h           ;clear the screen
124     mov    bh, 07h
125     mov    cx, 0000h
126     mov    dx, 184fh
127     int    10h
128     mov    ah, 02h
129     mov    bh, 04h
130     mov    dx, 00h
131     int    10h
132     Pop    dx
133     Pop    cx
134     Pop    bx
135     Pop    ax
136     endm
137
138 ; \brief encrypt a string which has N bytes(N > 0)
139 ; \return store it in place
```

B. 源代码

```
140 encrypt macro addr, N
141 local looping, ending, anti_0, anti_1, anti_2, anti_3, anti_4, anti_5, anti_6, anti_7, anti_8,
     anti_9, anti_10, anti_11, anti_12, anti_13, anti_14
142 jmp    anti_0
143
144 anti_3:          dec    ebx
145 jmp    anti_4
146 anti_6:          cmp    ecx, N
147 jmp    anti_7
148 anti_12:         inc    ecx
149 jmp    anti_13
150 anti_4:          mov    ecx, 1
151 jmp    anti_5
152 anti_10:         mov    [ebx], al
153 jmp    anti_11
154 anti_7:          je     ending
155 jmp    anti_8
156 anti_0:          pushad
157 jmp    anti_1
158 anti_11:         dec    ebx
159 jmp    anti_12
160 anti_1:          mov    ebx, addr+N
161 jmp    anti_2
162 anti_9:          xor   al, [ebx+1]
163 jmp    anti_10
164 anti_5:          looping:
165 jmp    anti_6
166 anti_8:          mov    al, [ebx]
167 jmp    anti_9
168 anti_13:         jmp    looping
169 jmp    anti_14
170 anti_2:          dec    ebx
171 jmp    anti_3
172
173 anti_14:
174 ending:
175 popad
176 endm
177
178 ; \brief decrypt a string which has N bytes(N > 0)
179 ; \return store it in place
180 decrypt macro addr, N
181 local looping, ending, anti_0, anti_1, anti_2, anti_3, anti_4, anti_5, anti_6, anti_7, anti_8,
     anti_9, anti_10, anti_11, anti_12, anti_13, anti_14
182 jmp    anti_0
183
184 anti_13:         jmp    looping
185 jmp    anti_14
186 anti_2:          mov    ecx, 1
187 jmp    anti_3
188 anti_7:          mov    al, [ebx]
189 jmp    anti_8
190 anti_1:          mov    ebx, addr
191 jmp    anti_2
192 anti_12:         inc    ecx
```

B. 源代码

```
193  jmp      anti_13
194  anti_5:          je      ending
195  jmp      anti_6
196  anti_6:
197  jmp      anti_7
198  anti_0:          pushad
199  jmp      anti_1
200  anti_10:         inc     ebx
201  jmp      anti_11
202  anti_8:          xor     al, [ebx+1]
203  jmp      anti_9
204  anti_3:          looping:
205  jmp      anti_4
206  anti_9:          mov     [ebx], al
207  jmp      anti_10
208  anti_4:          cmp     ecx, N
209  jmp      anti_5
210  anti_11:
211  jmp      anti_12
212
213  anti_14:
214  ending:
215  popad
216  endm
```

nmake 文件 MAKEFILE :

```
1  AS=TASM
2  ASFLAGS=/z /zi
3
4  LINK=TLINK
5  LINKFLAGS=/Tde /3
6
7  ALL: STUDENT.EXE
8      echo " BUILDING FINAL TARGET... "
9
10 STUDENT.EXE: MAIN.OBJ SUBMOD1.OBJ SUBMOD2.OBJ SUBMOD3.OBJ COMMOD.OBJ
11      $(LINK) $(LINKFLAGS) MAIN.OBJ+SUBMOD1.OBJ+SUBMOD2.OBJ+SUBMOD3.OBJ+SUBMOD4.OBJ+COMMOD.OBJ
12
13 .ASM.OBJ:
14      $(AS) $(ASFLAGS) *.ASM
```

II.5 任务五

主程序 MAIN.ASM

```

1 .386
2 .model flat, stdcall
3 option casemap :none      ; case sensitive
4
5 include windows.inc
6 include user32.inc
7 include kernel32.inc
8 include comctl32.inc
9 include gdi32.inc
10 include macrolib
11
12 includelib user32.lib
13 includelib kernel32.lib
14 includelib comctl32.lib
15 includelib gdi32.lib
16
17 ; Local macros
18 szText MACRO Name, Text:VARARG
19     LOCAL lbl
20         jmp lbl
21         Name db Text,0
22         lbl:
23 ENDM
24
25 ; move words in memory
26 movm MACRO M1, M2
27     push M2
28     pop M1
29 endm
30
31 return MACRO arg
32     mov eax, arg
33     ret
34 ENDM
35
36 ; prototypes
37 startMainWindow PROTO :DWORD, :DWORD, :DWORD, :DWORD
38 signalHandle     PROTO :DWORD, :DWORD, :DWORD, :DWORD
39 TopXY           PROTO :DWORD, :DWORD
40 Paint_Proc       PROTO :DWORD, :DWORD
41 Static           PROTO :DWORD, :DWORD, :DWORD, :DWORD, :DWORD, :DWORD
42 ListBox          PROTO :DWORD, :DWORD, :DWORD, :DWORD, :DWORD, :DWORD
43 EnmProc          PROTO :DWORD, :DWORD
44 RefreshAvg       PROTO
45 refresh_rank     PROTO
46
47 student struc
48     naming db 10 dup(0)
49     chinese dw 0
50     math    dw 0
51     english dw 0
52     average dw 0

```

B. 源代码

```
53     Rank      dw 0
54 student ends
55
56 .data
57     szDisplayName db "SIMS - Student Information Management System",0
58     CommandLine   dd 0
59     MainWindow    dd 0
60     hInstance     dd 0
61     hStatus       dd 0
62     hToolBar      dd 0
63     hList         dd 0
64
65     stu_max_num  equ 10
66     tab           student <'alpha',95,96, 97,,,>
67                 student <'beta',80,90,100,,,>
68                 student <'gamma',100,95,100,,,>
69                 student <'delta',60,65,66,,,>
70                 student <'epsilon',50,97, 86,,,>
71                 student 4 dup(<>)
72                 student <'husixu',98,99,88,,,>
73
74 /* code segment -----
75 .code
76 start:
77     ; get the handle of the module
78     invoke GetModuleHandle, NULL
79     mov hInstance, eax
80     ; get the handle of the commandline parameter
81     invoke GetCommandLine
82     mov CommandLine, eax
83     ; start the main window process
84     invoke startMainWindow, hInstance,NULL,CommandLine,SW_SHOWDEFAULT
85     ;exit
86     invoke ExitProcess,eax
87
88
89 /* Main Process -----
90 startMainWindow proc hInst      :DWORD,
91                  hPrevInst  :DWORD,
92                  CmdLine    :DWORD,
93                  CmdShow    :DWORD
94     ; local variables
95     LOCAL    windclass :WNDCLASSEX
96     LOCAL    msg        :MSG
97
98     LOCAL    wid        :DWORD
99     LOCAL    height     :DWORD
100    LOCAL   topx       :DWORD
101    LOCAL   topy       :DWORD
102
103    invoke InitCommonControls
104    ; initialize the main window
105    mov windclass.cbSize,           sizeof WNDCLASSEX
106    mov windclass.style,           CS_HREDRAW or CS_VREDRAW or CS_BYTEALIGNWINDOW
107    mov windclass.lpfnWndProc,     offset signalHandle
```

B. 源代码

```
108 mov windclass.cbClsExtra,          0
109 mov windclass.cbWndExtra,          0
110 movm windclass.hInstance,          hInst    ;<< NOTE: macro not mnemonic
111 mov windclass.hbrBackground,      COLOR_BTNFACE+1
112 mov windclass.lpszMenuName,       NULL
113 mov windclass.lpszClassName,      offset szClassName
114
115 invoke LoadIcon, hInst, 500      ; icon ID
116 mov windclass.hIcon,             eax
117
118 invoke LoadCursor, NULL, IDC_ARROW
119 mov windclass.hCursor,           eax
120 mov windclass.hIconSm,           0
121
122 invoke RegisterClassEx, ADDR windclass
123
124 ; set width and height
125 mov wid, 640
126 mov height, 500
127
128 invoke GetSystemMetrics, SM_CXSCREEN
129 invoke TopXY, wid, eax
130 mov topx, eax
131 invoke GetSystemMetrics, SM_CYSCREEN
132 invoke TopXY, height, eax
133 mov topy, eax
134
135 szText szClassName,"Comctl_Class"
136
137 ; create the main window
138 invoke CreateWindowEx, WS_EX_LEFT,
139                 ADDR szClassName,
140                 ADDR szDisplayName,
141                 WS_OVERLAPPEDWINDOW,
142                 topx, topy, wid, height,
143                 NULL, NULL,
144                 hInst, NULL
145 mov MainWindow, eax
146
147 invoke LoadMenu, hInst, 600 ; menu ID
148 invoke SetMenu, MainWindow, eax
149 invoke ShowWindow, MainWindow, SW_SHOWNORMAL
150 invoke UpdateWindow, MainWindow
151
152 ; message loop
153 looping:
154     invoke GetMessage, ADDR msg, NULL, 0, 0
155     cmp eax, 0
156     je ending
157     invoke TranslateMessage, ADDR msg
158     invoke DispatchMessage, ADDR msg
159     jmp looping
160 ending:
161 return msg.wParam
162 startMainWindow endp
```

B. 源代码

```
163
164
165 ;— message handling ——————
166 signalHandle proc hWin :DWORD,
167             uMsg :DWORD,
168             wParam :DWORD,
169             lParam :DWORD
170
171     LOCAL caW :DWORD
172     LOCAL caH :DWORD
173     LOCAL hDC :DWORD
174     LOCAL Rct :RECT
175     LOCAL tbb :TBUTTON
176     LOCAL Tba :TBADDBITMAP
177     LOCAL Ps :PAINTSTRUCT
178
179     LOCAL rLeft :DWORD
180     LOCAL rTop :DWORD
181     LOCAL rRight :DWORD
182     LOCAL rBottom :DWORD
183
184     szText tbSelect,"You have selected"
185
186 ; window creation
187 .if uMsg == WM_CREATE
188     invoke ListBox, 20, 25, 590, 400, hWin, 600
189     mov hList, eax
190     jmp @@F
191     lbl1 db " Name",0
192     lbl2 db " Chinese",0
193     lbl3 db " English",0
194     lbl4 db " Math",0
195     lbl5 db " Average",0
196     lbl6 db " Rank",0
197     @@
198
199     invoke Static, ADDR lbl1, hWin, 20, 5, 90, 18, 500
200     invoke Static, ADDR lbl2, hWin, 120, 5, 90, 18, 501
201     invoke Static, ADDR lbl3, hWin, 220, 5, 90, 18, 502
202     invoke Static, ADDR lbl4, hWin, 320, 5, 90, 18, 503
203     invoke Static, ADDR lbl5, hWin, 420, 5, 90, 18, 504
204     invoke Static, ADDR lbl6, hWin, 520, 5, 90, 18, 504
205
206 ;— Menu Events ——————
207 .elseif uMsg == WM_COMMAND
208     .if wParam == 1000
209         invoke SendMessage, hWin, WM_SYSCOMMAND, SC_CLOSE, NULL
210
211 ; calculate the average score
212 .elseif wParam == 1100
213     ; TODO: refresh all scores
214     szText TheMsg, "The Average Score and Rank has been refreshed",0
215     invoke RefreshAvg
216     invoke refresh_rank
217     invoke MessageBox, hWin, ADDR TheMsg, ADDR szDisplayName, MB_OK
```

B. 源代码

```
218         ; show the whole list
219     .elseif wParam == 1200
220         invoke SendMessage, hList, LB_RESETCONTENT, 0, 0
221         invoke EnumWindows, ADDR EnmProc, 0
222
223         ; about
224     .elseif wParam == 1300
225         szText AboutMsg, "Author: Sixu Hu", 0dh, "Mail: husixu1@hotmail.com", 0
226         invoke MessageBox, hWin, ADDR AboutMsg, ADDR szDisplayName, MB_OK
227
228     .endif
229
230
231     ;— Resize Events ——————
232     .elseif uMsg == WM_SIZE
233         ; move the status bar
234         invoke SendMessage, hToolBar, TB_AUTOSIZE, 0, 0
235         movm caW, lParam[0] ; client area width
236         movm caH, lParam[2] ; client area height
237         invoke GetWindowRect, hStatus, ADDR Rct
238         mov eax, Rct.bottom
239         sub eax, Rct.top
240         sub caH, eax
241         invoke MoveWindow, hStatus, 0, caH, caW, caH, TRUE
242
243         .if hList != 0
244             invoke GetClientRect, hWin, ADDR Rct
245             movm rLeft, Rct.left
246             add rLeft, 20
247             movm rTop, Rct.top
248             add rTop, 25
249             movm rRight, Rct.right
250             sub rRight, 40
251             movm rBottom, Rct.bottom
252             sub rBottom, 45
253             invoke MoveWindow, hList, rLeft, rTop, rRight, rBottom, TRUE
254         .endif
255
256     .elseif uMsg == WM_PAINT
257         invoke BeginPaint, hWin, ADDR Ps
258         mov hdc, eax
259         invoke Paint_Proc, hWin, hdc
260         invoke EndPaint, hWin, ADDR Ps
261         return 0
262
263     ;— closing the window ——————
264     .elseif uMsg == WM_CLOSE
265         szText TheText, "Do you really want to exit?", 0dh, "All changes will be lost."
266         invoke MessageBox, hWin, ADDR TheText, ADDR szDisplayName, MB_YESNO
267         .if eax == IDNO
268             return 0
269         .endif
270
271     .elseif uMsg == WM_DESTROY
272         invoke PostQuitMessage, NULL
```

B. 源代码

```
273         return 0
274
275     .endif
276
277     ; execute the pre-defined window processes
278     invoke DefWindowProc, hWin, uMsg, wParam, lParam
279     ret
280 signalHandle endp
281
282
283 ;== sub processes =====
284 =====
285
286 TopXY proc wDim:DWORD, sDim:DWORD
287     shr sDim, 1      ; divide screen dimension by 2
288     shr wDim, 1      ; divide window dimension by 2
289     mov eax, wDim    ; copy window dimension into eax
290     sub sDim, eax    ; sub half win dimension from half screen dimension
291     return sDim
292 TopXY endp
293
294 ;=====
295
296 Paint_Proc proc hWin:DWORD, hDC:DWORD
297     LOCAL caW :DWORD
298     LOCAL caH :DWORD
299     LOCAL tbH :DWORD
300     LOCAL sbH :DWORD
301     LOCAL Rct :RECT
302
303     invoke GetClientRect,hWin,ADDR Rct
304     movm caW, Rct.right
305     movm caH, Rct.bottom
306
307     invoke GetWindowRect,hToolBar,ADDR Rct
308     mov eax, Rct.bottom
309     sub eax, Rct.top
310     mov tbH, eax
311
312     invoke GetWindowRect,hStatus,ADDR Rct
313     mov eax, Rct.bottom
314     sub eax, Rct.top
315     mov sbH, eax
316
317     mov eax, caH
318     sub eax, sbH
319     mov caH, eax
320
321     mov Rct.left, 0
322     movm Rct.top, tbH
323     movm Rct.right, caW
324     movm Rct.bottom, caH
325
326     invoke DrawEdge,hDC,ADDR Rct,EDGE_SUNKEN,BF_RECT
327     return 0
```

B. 源代码

```
328 Paint_Proc endp
329
330 ;— list box creation ——————
331 ListBox proc a:DWORD, b:DWORD, wd:DWORD, ht:DWORD, hParent:DWORD, ID:DWORD
332     LOCAL hFont :DWORD
333     LOCAL hLst  :DWORD
334     szText listBox,"LISTBOX"
335     invoke CreateWindowEx, WS_EX_CLIENTEDGE, ADDR listBox, 0,
336             WS_VISIBLE or WS_BORDER or WS_CHILD or \
337             LBS_HASSTRINGS or LBS_NOINTEGRALHEIGHT or \
338             LBS_DISABLENOSCROLL,
339             a, b, wd, ht, hParent, ID, hInstance, NULL
340
341     mov hLst, eax
342
343     invoke GetStockObject, SYSTEM_FIXED_FONT      ; ANSI_FIXED_FONT
344     mov hFont, eax
345     invoke SendMessage,hLst,WM_SETFONT,hFont, 0
346
347     mov eax, hLst
348
349     ret
350
351 ListBox endp
352
353 ;— static item creation ——————
354 Static proc lpText:DWORD,hParent:DWORD,
355             a:DWORD,b:DWORD,wd:DWORD,ht:DWORD,ID:DWORD
356     LOCAL hStat :DWORD
357     LOCAL hFont :DWORD
358
359     szText statClass,"STATIC"
360
361     invoke CreateWindowEx,WS_EX_STATICEDGE,
362             ADDR statClass,lpText,
363             WS_CHILD or WS_VISIBLE or SS_LEFT,
364             a,b,wd,ht,hParent,ID,
365             hInstance,NULL
366
367     mov hStat, eax
368
369     invoke GetStockObject,ANSI_FIXED_FONT
370     mov hFont, eax
371     invoke SendMessage,hStat,WM_SETFONT,hFont, 0
372
373     mov eax, hStat
374
375     ret
376
377 Static endp
378
379 ;— List Refreshing ——————
380 EnmProc proc eHandle :DWORD, y :DWORD
381     LOCAL Buffer[256]      :BYTE
382     LOCAL clName[64]       :BYTE
```

B. 源代码

```
383
384     mov ecx, 00h
385     szText ctlstr1,"%10s %7u %1lu %1lu"
386     szText ctlstr2,"%s %1lu %1lu"
387     szText BufferNone, ' — — — — — '
388
389     looping:
390     mov eax, ecx
391     mov ebx, 14h
392     mul ebx
393     push ecx
394         ;TODO: convert the number to string in the tab
395
396     .if tab[eax].naming[0] == 0
397         invoke SendMessage, hList, LB_ADDSTRING, 0, ADDR BufferNone
398     .else
399         pushad
400         xor ebx, ebx
401         xor ecx, ecx
402         xor edx, edx
403         xor esi, esi
404         xor edi, edi
405         mov bx, tab[eax].chinese
406         mov cx, tab[eax].math
407         mov dx, tab[eax].english
408         mov si, tab[eax].average
409         mov di, tab[eax].Rank
410
411         invoke wsprintf, ADDR Buffer, ADDR ctlstr1, ADDR tab[eax].naming, ebx, ecx, edx
412     .if si!=0
413         invoke wsprintf, ADDR Buffer, ADDR ctlstr2, ADDR Buffer, esi, edi
414     .endif
415
416     popad
417     invoke SendMessage, hList, LB_ADDSTRING, 0, ADDR Buffer
418     .endif
419
420     pop ecx
421     inc ecx
422     cmp ecx, 0ah
423     jne looping
424
425     ;mov eax, eHandle
426     mov eax, 0
427     ret
428 EnmProc endp
429
430 /* refresh average score ——————
431
432 ; \brief calculate a student's avgscore and store it in tab
433 ; \para (2 byte) index of the student in stack
434 ; \return (2 byte) avg score in stack (CAN NOT BE IGNORED)
435 cal_one_avg proc
436     push    eax
437     push    ebx
```

B. 源代码

```
438 push    ecx
439 push    edx
440     mov     eax, 00h
441     mov     edx, 00h          ; dx stores the avg score
442     mov     ax, [esp+14h]      ; get index (parameter)
443     mov     ebx, 14h          ; multiple by 20
444     mul     ebx
445     mov     ecx, eax         ; ecx as the pointer
446     mov     ax, 0             ; ax as the total sum
447
448     mov     dx, word ptr tab[ecx].chinese   ; get chinese score
449     shl     dx, 1
450     add     ax, dx
451
452     mov     dx, word ptr tab[ecx].math       ; get math score
453     add     ax, dx
454
455     mov     dx, word ptr tab[ecx].english    ; get english score
456     shr     dx, 1
457     add     ax, dx
458
459     mov     dx, 0h
460     shl     ax, 1
461     mov     bx, 07H           ;s divide by 3.5
462     div     bx
463
464     mov     word ptr tab[ecx].average, ax    ; store it to correct position
465 pop     edx
466 pop     ecx
467 pop     ebx
468 pop     eax
469 ret
470 cal_one_avg endp
471
472 -----
473 RefreshAvg proc
474     mov     cx, 00h
475 refresh_loop:
476     mov     eax, 00h
477     mov     ax, cx
478     mov     bx, 14h
479     mul     bx
480
481     mov     bl, tab[eax].naming[0]
482     cmp     bl, 00h
483     jne     cal_this
484     jmp     cal_this_end
485
486 cal_this:
487     push    cx              ; pass the parameter
488     call    cal_one_avg
489     pop     ax              ; get the result (but do not use)
490
491 cal_this_end:
492     inc     cx
```

B. 源代码

```
493     cmp      cx, stu_max_num
494     jnz      refresh_loop
495     ret
496 RefreshAvg endp
497
498
499 ;— refresh ranking ——————
500
501
502 ; \brief calculate a student's ranking and store it in tab
503 ; \para (2 byte) index of the student in stack
504 ; \return (2 byte) rank in stack (CAN NOT BE IGNORED)
505 cal_one_rank proc
506     push    eax
507     push    ebx
508     push    ecx
509     push    edx
510     push    esi
511     mov     eax, 00h
512     mov     esi, 00h           ; esi store the rank
513     mov     ax, [esp+18h]       ; get index (parameter)
514     mov     ebx, 14h          ; multiple by 20
515     mul     ebx
516     mov     bx, word ptr tab[eax].average ; ax stores the current avgscore
517
518     mov     cx, 00h
519 loop_rank:
520     push    bx
521     mov     eax, 00h
522     mov     bx, 14h
523     mov     ax, cx
524     mul     bx
525
526     cmp     tab[eax].naming[0], byte ptr 0
527     je      not_reged
528
529     pop    bx
530     cmp     word ptr tab[eax].average, bx
531     jg      add_one
532     jmp     loop_rank_tail
533
534 add_one:
535     inc    si
536     jmp     loop_rank_tail
537
538 not_reged:
539     pop    bx
540
541 loop_rank_tail:
542     inc    cx
543     cmp     cx, stu_max_num
544     jne     loop_rank
545
546 loop_end:
547     inc    si
```

B. 源代码

```
548
549     mov      ax, [esp+18h]           ; get index (parameter)
550     mov      ebx, 14h              ; multiple by 20
551     mul      ebx
552
553     mov      word ptr tab[eax].Rank, si ; store it to correct position
554     pop      esi
555     pop      edx
556     pop      ecx
557     pop      ebx
558     pop      eax
559     ret
560     cal_one_rank endp
561
562 ; \brief refresh all student's rank, but do NOT refresh avgscore automatically, use with care
563 refresh_rank proc
564     mov      cx, 00h
565     refresh_loop:
566     mov      eax, 00h
567     mov      ax, cx
568     mov      bx, 14h
569     mul      bx
570
571     cmp      tab[eax].naming[0], byte ptr 00h
572     jne      cal_this
573     jmp      cal_this_end
574
575     cal_this:
576     push     cx                  ; pass the parameter
577     call    cal_one_rank
578     pop      ax                  ; get the result (but do not use)
579
580     cal_this_end:
581     inc      cx
582     cmp      cx, stu_max_num
583     jnz      refresh_loop
584     ret
585     refresh_rank endp
586 ; ######
587
588 end start
```

资源文件 res.rc

```
1 500 ICON MOVEABLE PURE LOADONCALL DISCARDABLE "MAINICON ICO"
2
3 600 MENUEX MOVEABLE IMPURE LOADONCALL DISCARDABLE
4 BEGIN
5   POPUP "&File", , , 0
6   BEGIN
7     MENUITEM "&Exit", 1000
8   END
9
10  POPUP "&Action", , , 0
11  BEGIN
12    MENUITEM "&Refresh Average and Rank", 1100
```

B. 源代码

```
13      MENUITEM "&Show/Refresh List", 1200
14  END
15
16  POPUP "&Help", , , 0
17  BEGIN
18      MENUITEM "&About", 1300
19  END
20 END
```