

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**实验名称： 汇编和C的混合编程及优化**

**院 系 ：计算机科学与技术**

**专业班级 ： 计算机2305班**

**学 号 ： U202315677**

**姓 名 ： 岳皓**

**指导教师 ： 班鹏新**

**2025 年 3 月 28 日**

**实验二 汇编和C的混合编程及优化**

**一、实验目的与要求**

⑴ 熟练掌握程序开发平台(VS2019/VS2022/VS2023) 下C和汇编语言的混合编程方法；

(2) 熟悉掌握常用机器指令的使用方法；

(3) 掌握代码优化的基本方法，提高程序运行速度。

**二、实验内容**

**任务1 用C语言编写一个学生成绩管理程序，具有平均成绩计算、按平均成绩从高到低排序、显示学生成绩的功能。**

定义了 结构 student，设有 N 个学生

struct student {

char name[8];  
 char sid[11]; // 如U202315123

short scores[8]; // 8门课的分数

short average; // 平均分

};

**要求**：

1. 第0个人姓名(name)为自己的名字，sid为自己的学号；
2. 对于函数“计算平均成绩”、“按平均成绩从高到低排序”分别计时，显示两个函数的时间开销；
3. 显示排序前、排序后的学生信息（姓名、学号、8门课的分数、平均分），一个学生一行。
4. 对“计算平均成绩”、“按平均成绩排序”函数的Debug版、Release版的运行效率进行对比，比较两个版本下的反汇编代码的差异，分析程序执行速度提高的原因。

**任务2 用汇编语言编写函数“计算平均成绩”，代替原来用C语言编写的函数**

**任务3 对用汇编语言编写的“计算平均成绩”函数进行优化**

**任务4 对排序函数进行算法优化**

**三、实验记录及问题回答**

**（1）任务 1 的实验测试结果记录**

初始化函数void initStudents(student\* s,int num)思想大致如下：

对于结构体数组s，对于第0个人，运用strcpy函数将自己名字和学号写入，再对八门成绩依次赋值，对于剩余的num-1个学生，在for循环中运用sprintf函数依次写入每个学生的姓名和学号，再运用rand函数和for循环给每位学生的八门成绩赋值，范围在50分到100分之间，对于每位学生，其平均成绩此时均赋为0。

计算平均成绩函数void computeAverageScore(student\* s,int num)思想大致如下:

运用for的双循环，对于每个学生，均初始化一个局部变量int sum=0，用于保存每位学生的总成绩，再循环依次将该学生的八门成绩相加，最后直接运用整数除法，将sum/8赋给s[i].average，得到的结果为该学生的省去小数的平均成绩。

按平均成绩排序函数void sortByAverageScore(student\* s, int num)思想大致如下：

运用冒泡排序，在for双循环中，每次依次比较相邻两位学生的平均成绩，若后者平均成绩大于前者，则初始化一个结构体变量temp=s[j]，再进行交换，步骤为:

s[j]=s[j+1];s[j+1]=temp;

经过num^2次循环后，将所有学生排序完成。

运行结果如下：

对于展示函数display，由于考虑到排序函数在样本较少的情况下时间较短以至于用时显示为0，从而将样本数量设置为10000来延长排序函数的运行时间，因样本数量太多，所以其仅展示数组的前八个学生数据，然后分别在不同版下面进行运行，下面图1和图2分别是Debug版和Release版的运行结果。

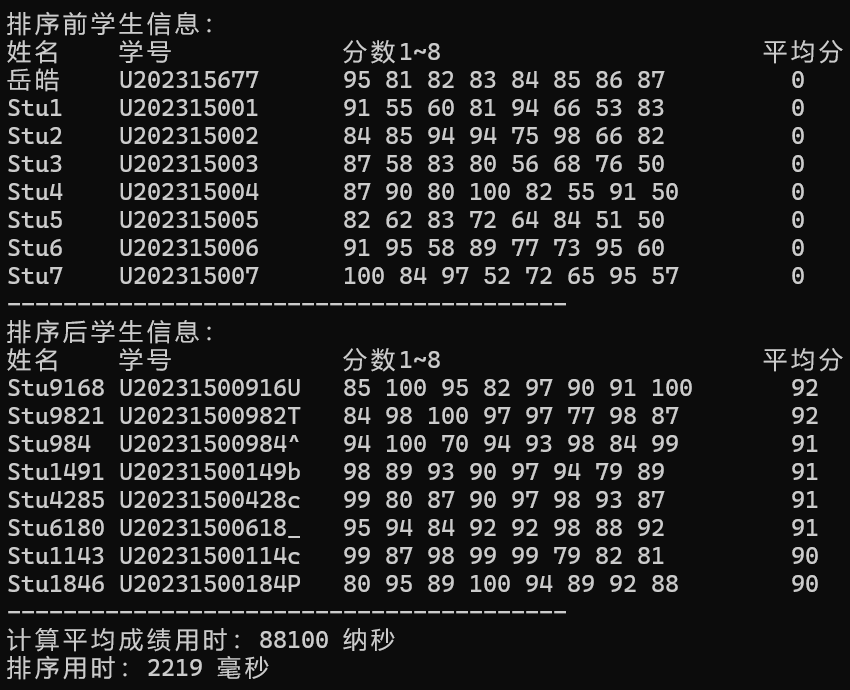


图 1 Debug版运行结果

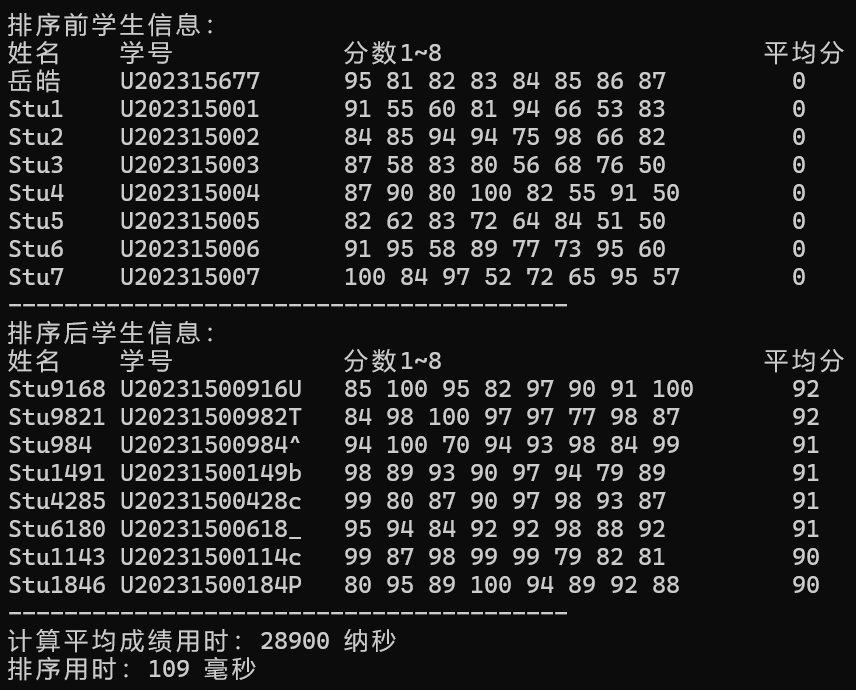


图 2 Release版运行结果

由于计算平均函数时间过于短小，将其计时部分换成更加精细的计时函数来输出精度更大的纳秒。

由图1和图2可知，两个函数在Release版的运行时间明显小于在Debug版运行时间，说明按平均成绩排序函数在Release版的运行效率明显优于在Debug版的运行效率。

对两个版本下的反汇编代码进行观察并节选相同的部分进行比较，如下图：

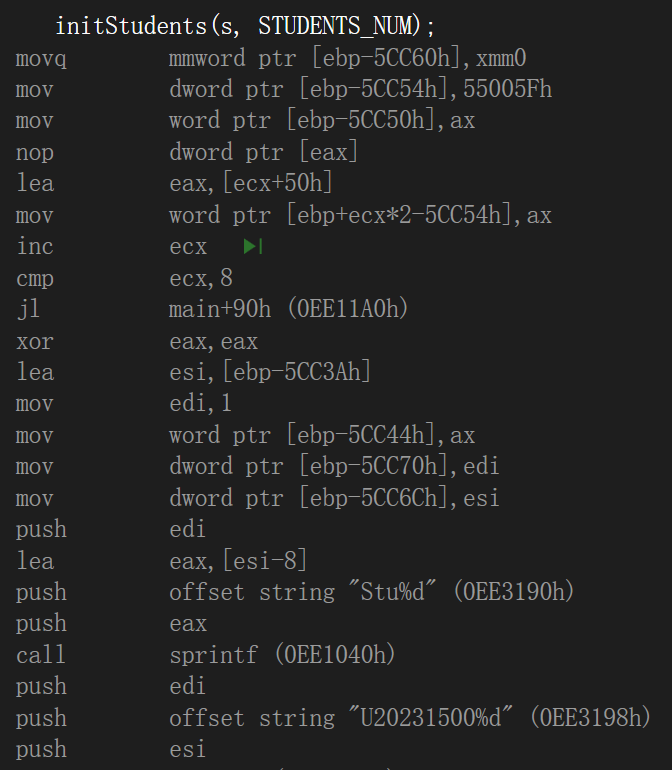
****

图 3 Release版反汇编代码节选

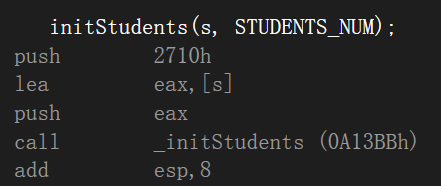
****

图 4 Debug版本反汇编代码节选

对比图3和图4，两者均为main函数相同部位在不同版本的运行部分反汇编代码的节选，观察可知，Debug版是直接使用call指令来调用函数，再运行该函数，而Release版的代码通过观察可知，该版本下，直接内联函数，将本来调用函数的部分直接去除，再嵌入本来调用的函数，从而消除函数调用的开销，进而提高运行效率，减少运行时间。

1. **任务 2 的实验测试结果记录**

根据前面c语言中的计算平均成绩函数代码，将其改写为汇编函数，并进行代替，可以在c文件中定义一个外部函数extern void computeAverageScore\_asm(student\* s, int num)，然后将原先的计算函数替换，替换后代码的运行结果如图5.

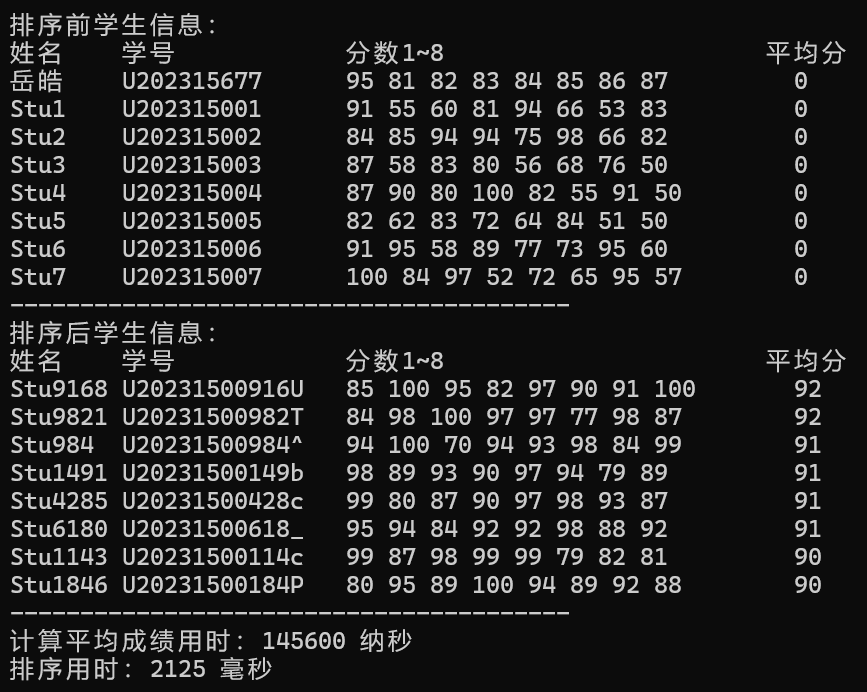
****

图 5 用汇编函数替换后运行结果

1. **任务 3 的实验测试结果记录、优化手段、优化原理分析**

优化原理：

1. 减少内存访问；

优化手段：对于原来的函数，每次运算student[i].[j]时，都要重新计算students

基地址并进行多次偏移计算，所以改为只计算依次student[i]的基址，并在外层循环中逐步递增ebx（即add ebx, 38）,而在内存循环中直接使用固定偏移。

1. 多采用寄存器：

优化手段：对于原来函数，其使用了多个局部变量（i,j,sum），而局部变量存储在栈上，访问速度比寄存器慢，所以用寄存器代替，从而加快访问速度。

1. 采用更高效的除法计算

优化手段：对于原来的函数，其先将8存储在ecx中，再用idiv对eax中的值进行除法运算，速度较慢，所以改为运用shr指令，将总成绩右移三位，由于二进制的存储特点，其等效于除以8的操作。

1. 优化循环：

优化手段：原函数中，多次使用jmp指令，导致影响分支预测效率，且代码看起来更加糅杂，所以改为使用更加紧凑的循环结构，减少了jmp的使用。

优化原理：

1. 减少内存访问；

优化手段：对于原来的函数，每次运算student[i].[j]时，都要重新计算students

基地址并进行多次偏移计算，所以改为只计算依次student[i]的基址，并在外层循环中逐步递增ebx（即add ebx, 38）,而在内存循环中直接使用固定偏移。

1. 多采用寄存器：

优化手段：对于原来函数，其使用了多个局部变量（i,j,sum），而局部变量存储在栈上，访问速度比寄存器慢，所以用寄存器代替，从而加快访问速度。

1. 采用更高效的除法计算

优化手段：对于原来的函数，其先将8存储在ecx中，再用idiv对eax中的值进行除法运算，速度较慢，所以改为运用shr指令，将总成绩右移三位，由于二进制的存储特点，其等效于除以8的操作。

1. 优化循环：

优化手段：原函数中，多次使用jmp指令，导致影响分支预测效率，且代码看起来更加糅杂，所以改为使用更加紧凑的循环结构，减少了jmp的使用。

对前面的汇编代码进行优化后运行结果如图6。

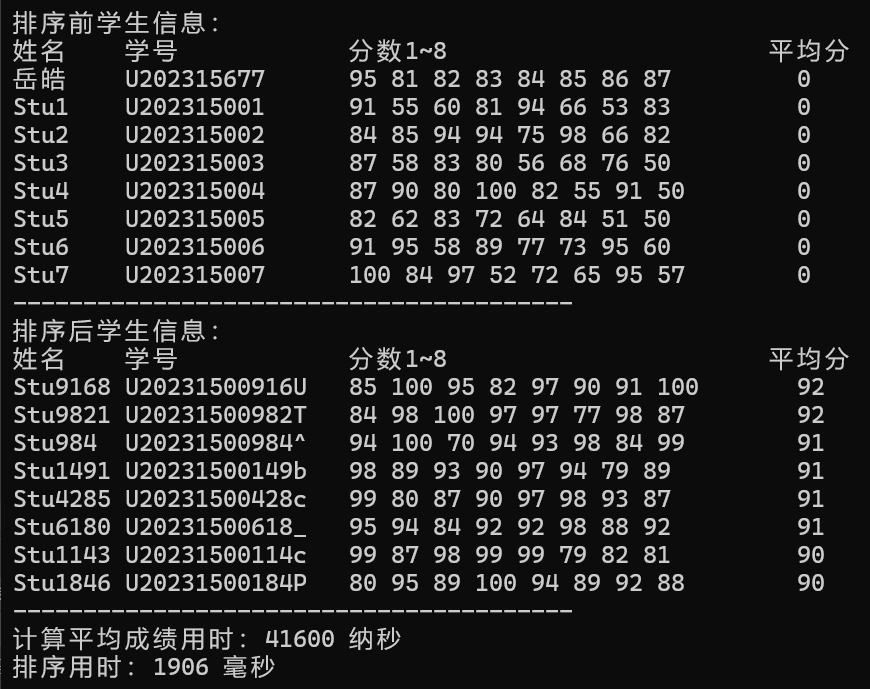
****

图 6 优化汇编函数后运行结果

**（4）任务 4 的实验测试结果记录、优化手段、优化原理分析**

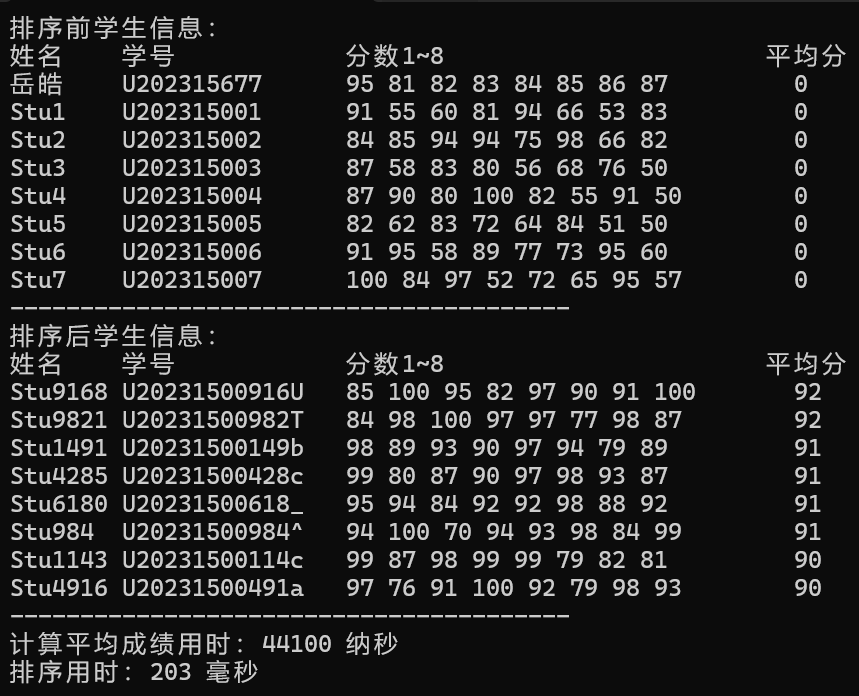
****

图 7 排序函数优化后运行结果

与任务一中的Debug版的运行结果比较，可以明显发现排序时间的缩短，从2219毫秒减少到203毫秒。

优化原理：将原排序函数所采用的冒泡排序改写为更加高效的快速排序。

优化手段：声明优化后的排序函数sortByAverageScore\_optimized(student\* s, int num)，其内容为quickSort(s,0,num-1)，而根据快速排序的原理，分别定义快排的分区函数partition，再将其运用到快排函数quickSort中，运用递归不断进行排序，直到条件(即low<high)不成立时排序完成。

**四、体会**

经过本次实验，我充分学习了如何使用汇编语言编写程序，并对代码进行优化。在这个过程中，我对于相关知识的运用更加熟练，对于vs的使用更加熟练，为今后的学习打下更加深厚的基础，并且在这个过程中体会到编写代码的乐趣，此外通过汇编语言，我更加深刻的体会到代码的逻辑性，理解了代码的运行逻辑，相信在今后的学习中对相加知识的掌握会更加牢固。

**五、源码**

实验任务 1、2、3、4 的源程序

任务一

main.c文件

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#include <stdint.h>

#define STUDENTS\_NUM 10000

typedef struct temp {

char sname[8];

char sid[11]; // 如U202315123

short scores[8]; // 8门课的分数

short average; // 平均分

}student;

extern void computeAverageScore\_asm(student\* s, int num);

extern void computeAverageScore\_greater(student\* s, int num);

void display(student\* s, int num);

void initStudents(student\* s,int num)

{

strcpy(s[0].sname, "岳皓");

strcpy(s[0].sid, "U202315677");

s[0].scores[0] = 95;

s[0].scores[1] = 85;

s[0].scores[2] = 90;

for (int i=1;i<8;i++)

s[0].scores[i] = 80+i;

s[0].average = 0;

for (int i = 1; i < num; i++) {

sprintf(s[i].sname, "Stu%d", i);

sprintf(s[i].sid, "U20231500%d", i);

for (int j = 0; j < 8; j++) {

s[i].scores[j] = 50 + rand() % 51;

}

s[i].average = 0;

}

}

void sortByAverageScore(student\* s, int num) {

for (int i = 0; i < num - 1; i++) {

for (int j = 0; j < num - i - 1; j++) {

if (s[j].average < s[j + 1].average) {

student temp = s[j];

s[j] = s[j + 1];

s[j + 1] = temp;

}

}

}

}

void computeAverageScore(student\* s,int num)

{

for (int i = 0; i < num; i++) {

int sum = 0;

for (int j = 0; j < 8; j++) {

sum += s[i].scores[j];

}

s[i].average = sum / 8; // 整数除法

}

}

int main()

{

student s[STUDENTS\_NUM];

int start, finish, duration;

initStudents(s, STUDENTS\_NUM);

printf("排序前学生信息：\n");

display(s, STUDENTS\_NUM);

struct timespec sstart, end;

uint64\_t start\_ns, end\_ns, elapsed\_ns;

// 计算平均分并计时

timespec\_get(&sstart, TIME\_UTC);

computeAverageScore\_greater(s, STUDENTS\_NUM);

timespec\_get(&end, TIME\_UTC);

start\_ns = sstart.tv\_sec \* 1000000000LL + sstart.tv\_nsec; // 可选，转换为纳秒以便计算差异（避免溢出）

end\_ns = end.tv\_sec \* 1000000000LL + end.tv\_nsec; // 可选，转换为纳秒以便计算差异（避免溢出）

elapsed\_ns = end\_ns - start\_ns; // 可选，计算时间差（纳秒）

// 排序并计时

start = GetTickCount();

sortByAverageScore\_optimized(s, STUDENTS\_NUM);

finish = GetTickCount();

duration = finish - start;

printf("排序后学生信息：\n");

display(s, STUDENTS\_NUM);

printf("计算平均成绩用时：%lld 纳秒\n", (unsigned long long)elapsed\_ns);

printf("排序用时：%d 毫秒\n", finish - start);

return 0;

}

computeAverageScore.asm文件

.686P

.model flat, c

printf proto c :ptr sbyte, :vararg

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

student struct

sname db 8 dup(0)

sid db 11 dup(0)

align 2 ; 指明对齐方式，汇编语言默认是紧凑存放

scores dw 8 dup(0)

average dw 0

student ends

.data

fmt\_header db "姓名",9,"学号",9,9,"分数1~8",9,9," 平均分",0dh,0ah,0

fmt\_name\_sid db "%s",9,"%s",9,0 ; 9为制表符ASCII码

fmt\_score db "%d ",0

fmt\_avg\_line db 9,"%d",0dh,0ah,0 ; 带换行的平均分格式

fmt\_separator db "----------------------------------------",0dh,0ah,0

.code

; 显示学生信息

; sptr 学生数组的首地址

; num 学生人数

display proc uses ebx esi edi,

sptr:dword,

num:dword

; 初始化寄存器

mov ebx, sptr ; 学生数组首地址

mov edi, 8 ; 改用EDI作为计数器

invoke printf, offset fmt\_header

student\_loop:

; 打印姓名学号

lea eax, [ebx].student.sname

lea edx, [ebx].student.sid

invoke printf, offset fmt\_name\_sid, eax, edx

; 打印分数

mov esi, 0

score\_loop:

movzx eax, word ptr [ebx].student.scores[esi\*2]

invoke printf, offset fmt\_score, eax

inc esi

cmp esi, 8

jl score\_loop

; 打印平均分

movzx eax, word ptr [ebx].student.average

invoke printf, offset fmt\_avg\_line, eax

; 移动指针

add ebx, sizeof student

; 循环控制

dec edi ; 使用EDI递减

jnz student\_loop

invoke printf, offset fmt\_separator

ret

display endp

end

任务二

computeAverageScore\_asm proc students: dword, num: dword

local i : dword

local j: dword ; 局部变量，用于内层循环计数器

local sum: dword ; 局部变量，用于存储分数总和

mov i, 0

outer\_loop:

mov eax, i

cmp eax, num

jge outer\_loop\_end ;如果i>=num，跳出循环

mov sum,0

mov j,0

inner\_loop:

cmp j,8

jge inner\_loop\_end ; 如果 j >= 8，跳出内层循环

; 计算当前分数的地址

mov ebx, students ; ebx 指向学生数组的首地址

mov eax, i ; eax = i

imul eax, 38

add ebx, eax ; ebx 指向第 i 个学生的结构体

mov eax, j ; eax = j

imul eax,2 ; eax = j \* 2（因为 scores 是 short 类型，每个元素占 2 字节）

add ebx, 20

add ebx, eax ; ebx 指向第 i 个学生的第 j 个分数

movsx eax, word ptr [ebx]

add sum, eax

inc j

jmp inner\_loop

inner\_loop\_end:

mov eax,sum

cdq ; 将 eax 符号扩展到 edx:eax

mov ecx, 8 ; ecx = 8

idiv ecx ; eax = sum / 8

mov ebx, students ; ebx 指向学生数组的首地址

mov ecx, i ; ecx = i

imul ecx, 38 ; ecx = i \* sizeof(student)

add ebx, ecx ; ebx 指向第 i 个学生的结构体

mov [ebx].student.average, ax ; 将平均分存入 students[i].average

; 外层循环计数器加 1

inc i

jmp outer\_loop ; 继续外层循环

outer\_loop\_end:

ret

computeAverageScore\_asm endp

任务三

computeAverageScore\_greater proc sptr: dword, num:dword

mov ebx,sptr

mov esi,0

loop\_student:

mov ecx,0

mov edi,0

loop\_score:

add di,[ebx+20+2\*ecx]

inc ecx

cmp ecx,8

jl loop\_score

shr di,3

mov [ebx+36],di ;只需要低十六位即可

add ebx,38

inc esi

cmp esi,num

jl loop\_student

ret

computeAverageScore\_greater endp

任务四

// 快速排序的分区函数

int partition(student\* s, int low, int high) {

short pivot = s[high].average;

int i = low - 1;

for (int j = low; j < high; j++) {

if (s[j].average >= pivot) { // 降序排序

i++;

// 交换 students[i] 和 students[j]

student temp = s[i];

s[i] = s[j];

s[j] = temp;

}

}

// 交换 students[i+1] 和 students[high]

student temp = s[i + 1];

s[i + 1] = s[high];

s[high] = temp;

return i + 1;

}

// 快速排序

void quickSort(student\* s, int low, int high) {

if (low < high) {

int pi = partition(s, low, high);

quickSort(s, low, pi - 1);

quickSort(s, pi + 1, high);

}

}

// 优化后的排序函数

void sortByAverageScore\_optimized(student\* s, int num) {

quickSort(s, 0, num - 1);

}