

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**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 ： CS2306**

**学 号 ： U202315696**

**姓 名 ： 彭冲**

**指导教师 ： 朱虹**

**2025 年 3 月 27 日**

**一、实验目的与要求**

⑴ 熟练掌握程序开发平台(VS2019/VS2022) 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； 可以用产生随机数的方法或者使用一定的规则，初始化N个学生的信息。N的个数自定；

（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  
 要求：只是将任务1中用C语言编写的“计算平均成绩”替换成用汇编语言编写的函数，其他程序保持不变；测试该函数的运行速度。

任务3 对用汇编语言编写的“计算平均成绩”函数进行优化  
 要求：使用寄存器与变量绑定、循环展开、使用更快的机器指令等手段，对程序进行优化；可以实现多个优化版本，比较它们的运行速度，并分析提高速度的原因。

任务4 对排序函数进行算法优化

要求：采用算法优化、程序优化等多种手段对函数进行优化；比较它们的运行速度，并分析提高速度的原因。

**三、实验记录及问题回答**

**（1）实验任务1 的实验测试结果记录**

**1.** **计算平均成绩calcAvg的源程序**

void calcAvg(student\* s, int sno) {

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

int sum = 0;

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

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

}

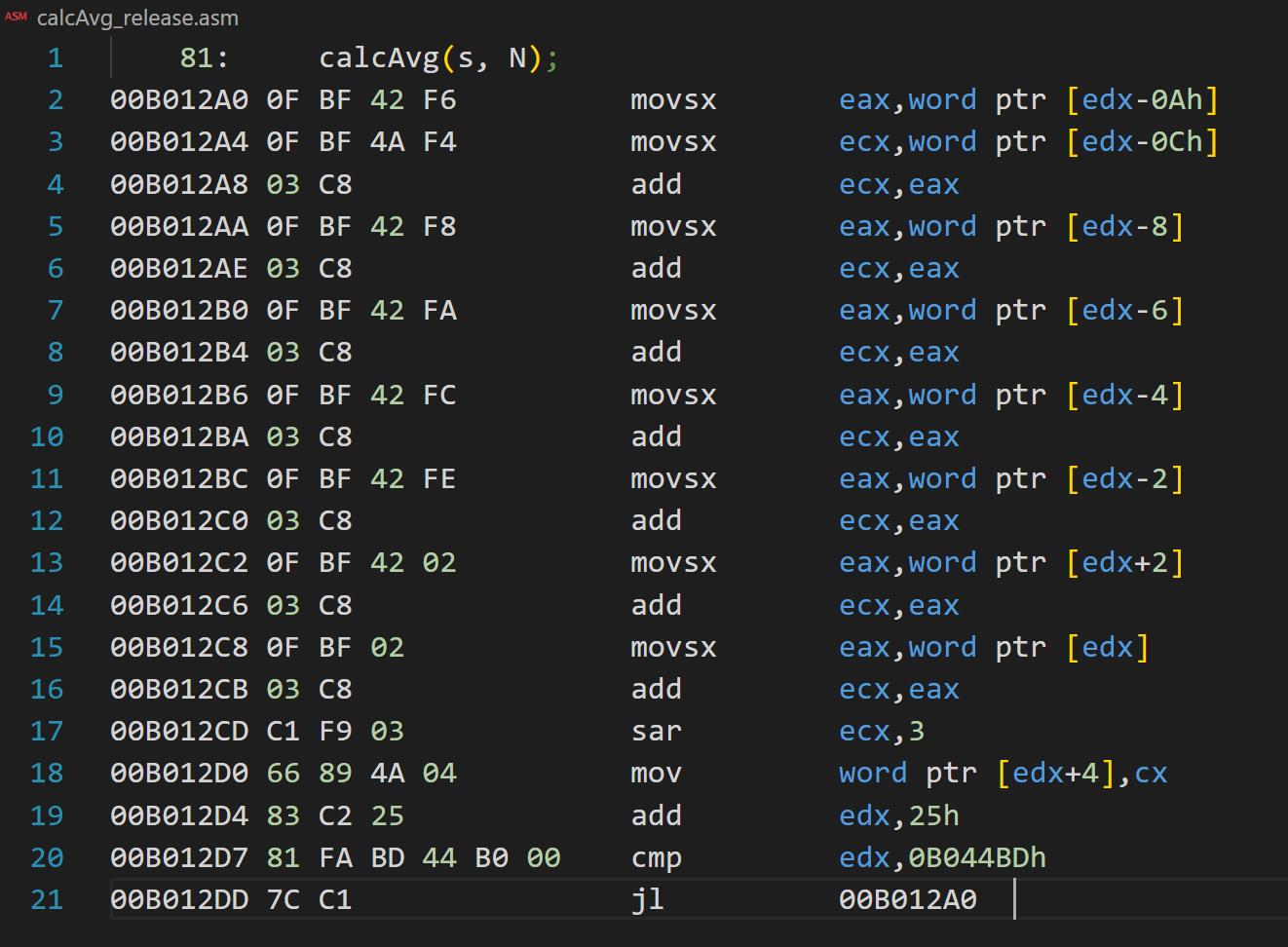
s[i].average = sum >> 3;

}

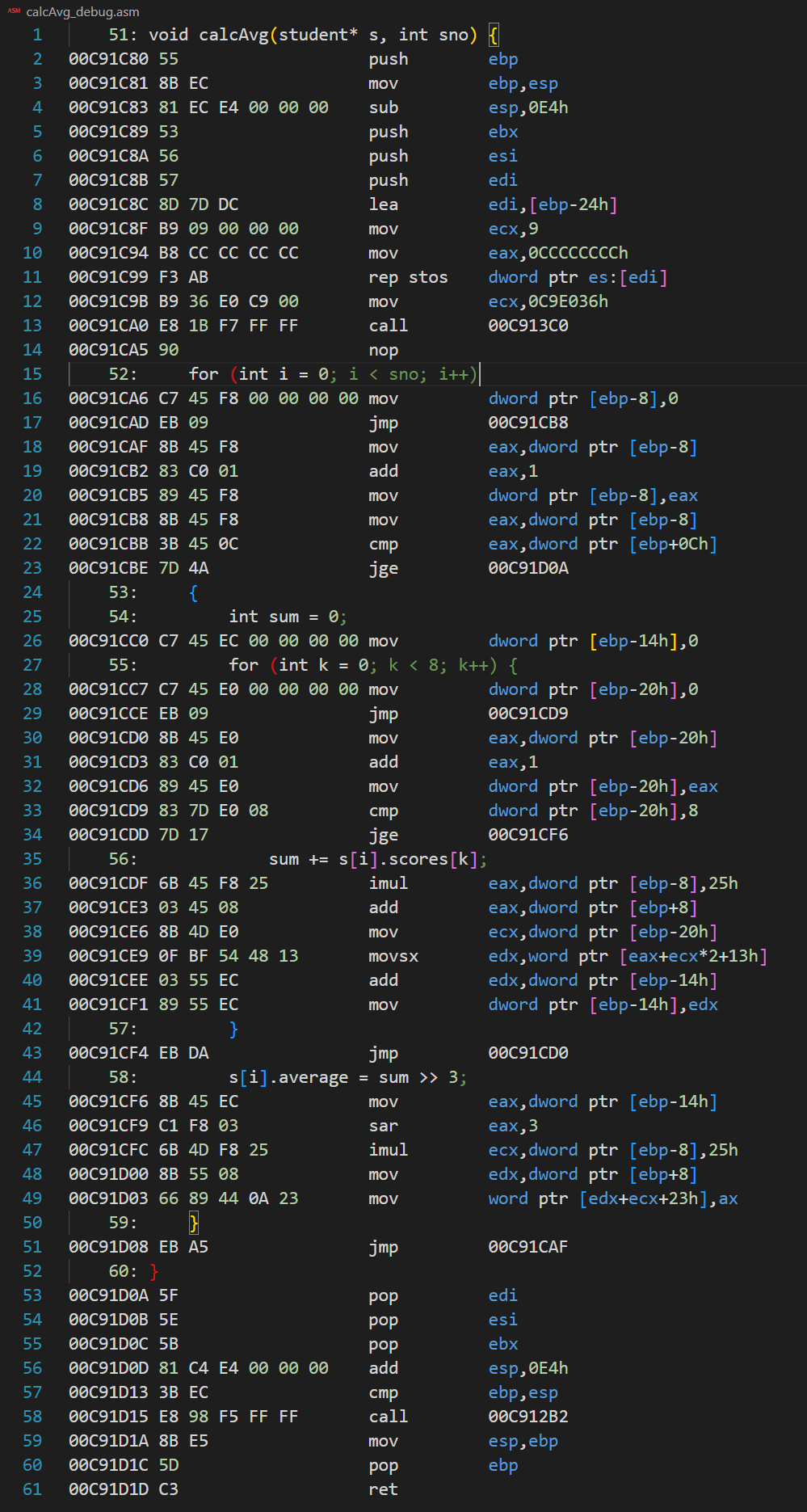
}

计算平均成绩calcAvg的Debug版与Release版对比，不难发现：

1. 直观上，Release版的指令远少于Debug版，较少的指令意味着更快的执行速度。
2. 寄存器方面，Debug版较多依赖源码语句，对局部变量的访问过多依赖堆栈， Release版则充分利用现有寄存器。
3. 内存访问方面，Debug版较多基址加变址寻址，而Release版直接计算出必要的偏移地址，省去了add、imul等指令。
4. 分支与循环方面，Debug版遵循源码的逻辑，而Release版对较少的循环进行展开，并减少了大量的跳转，提高指令流水线执行效率。



图表 1 计算平均成绩calcAvg的Release版

****

图表 2 计算平均成绩calcAvg的Debug版

**2. 按平均成绩排序sortByAvg 的源程序**

void sortByAvg(student\* s, int sno) { // 冒泡排序

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

for (int k = 0; k < sno - i; k++) {

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

student aux = s[k];

s[k] = s[k + 1];

s[k + 1] = aux;

}

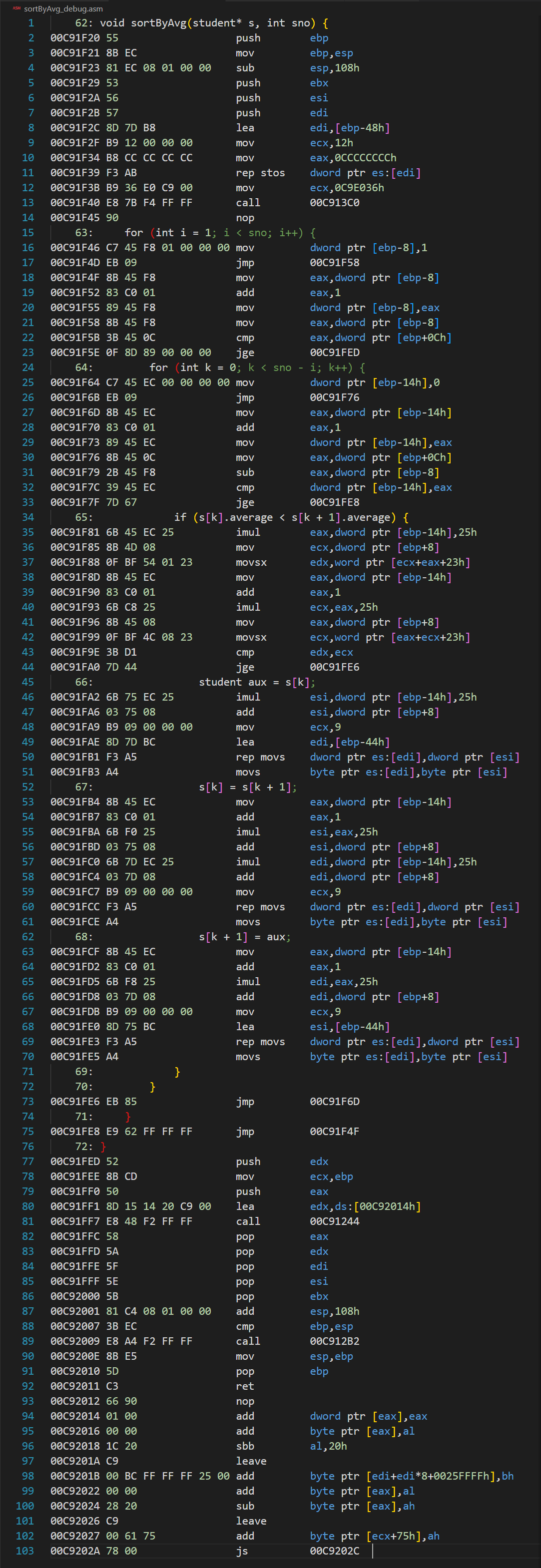
}

}

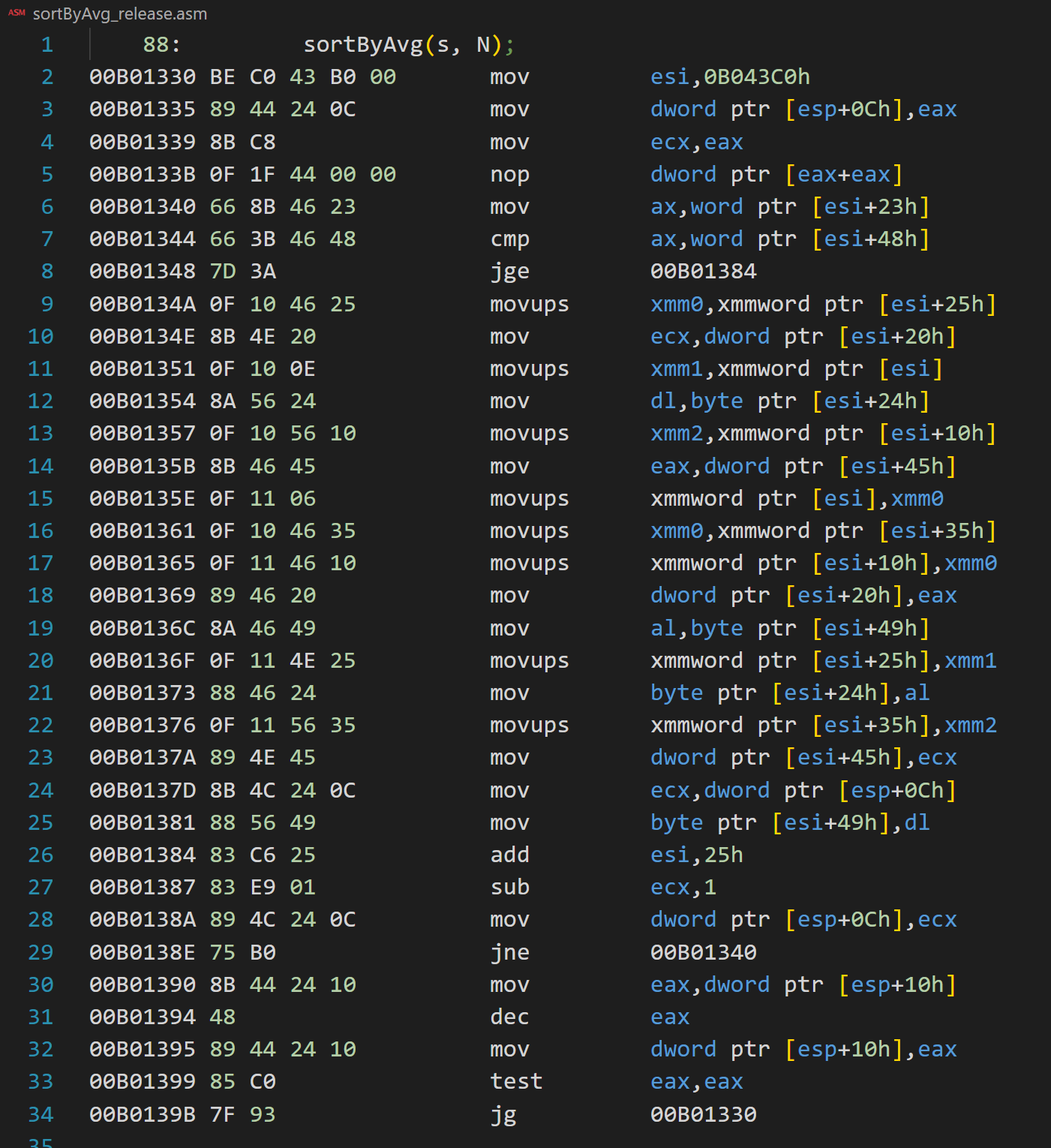
}

计算平均成绩calcAvg的Debug版与Release版对比，不难发现：

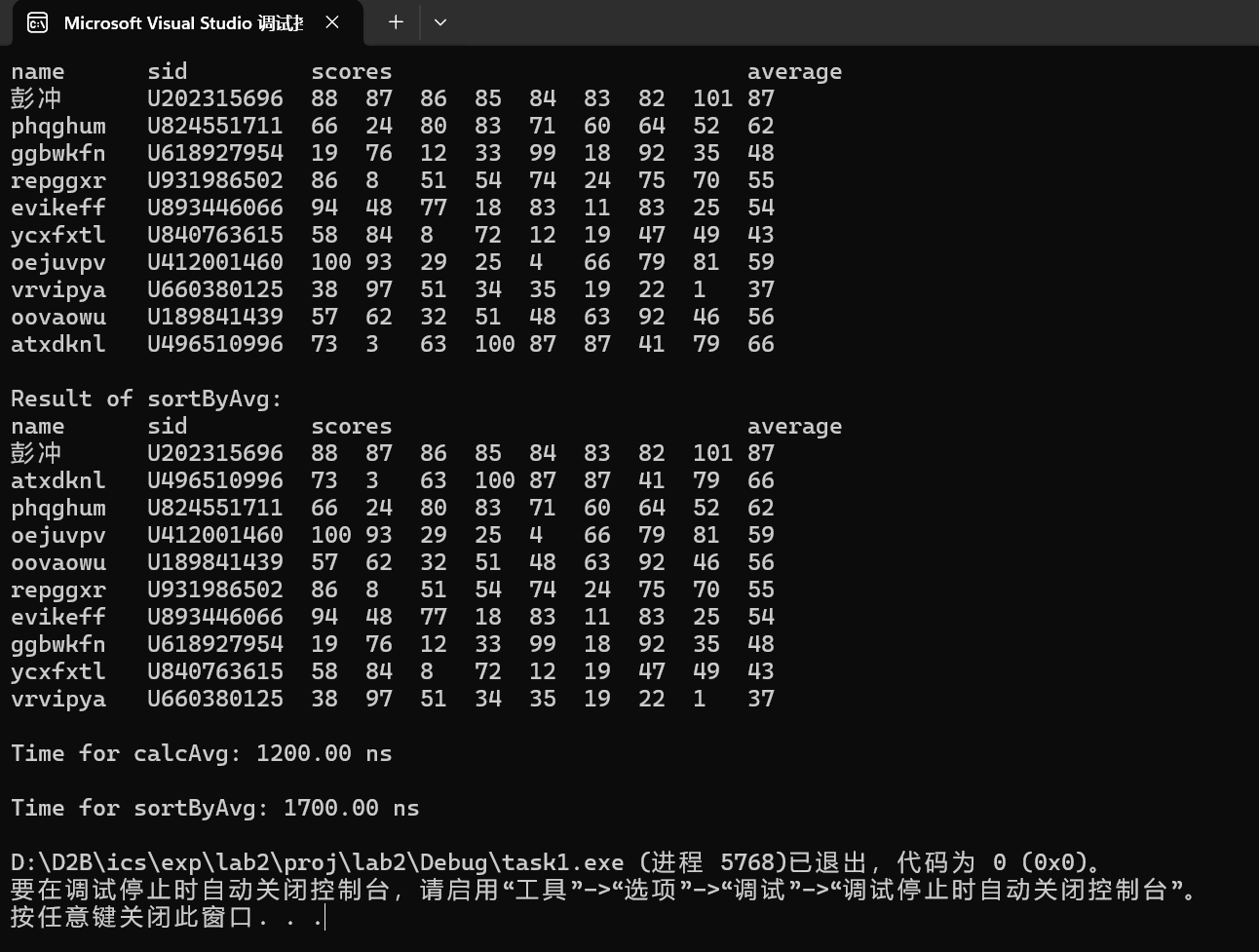
1. 直观上，同样，Release版的指令远少于Debug版，较少的指令意味着更快的执行速度。
2. 寄存器方面，Debug版仅使用了eax、ebx等32位寄存器，而Release版利用SSE指令，使用xxm寄存器加速内存交换。
3. 内存访问方面，Debug版较多基址加变址寻址，而Release版直接计算出必要的偏移地址，省去了add、imul等指令。
4. 分支与循环方面，Debug版遵循源码的逻辑，而Release版对较少的循环进行展开，并减少了大量的跳转，提高指令流水线执行效率。

****

图表 3 按平均成绩排序sortByAvg 的Debug版

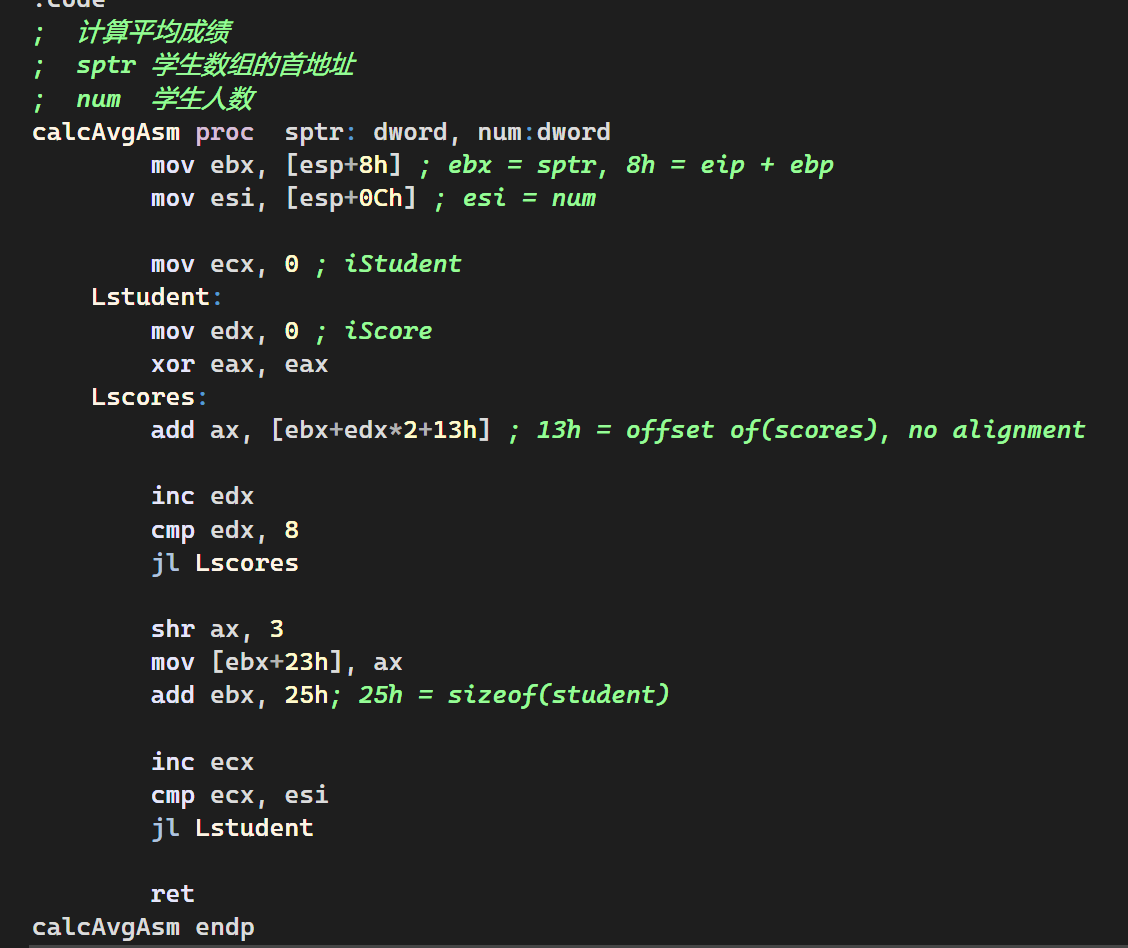
****

图表 4 按平均成绩排序sortByAvg 的Release版



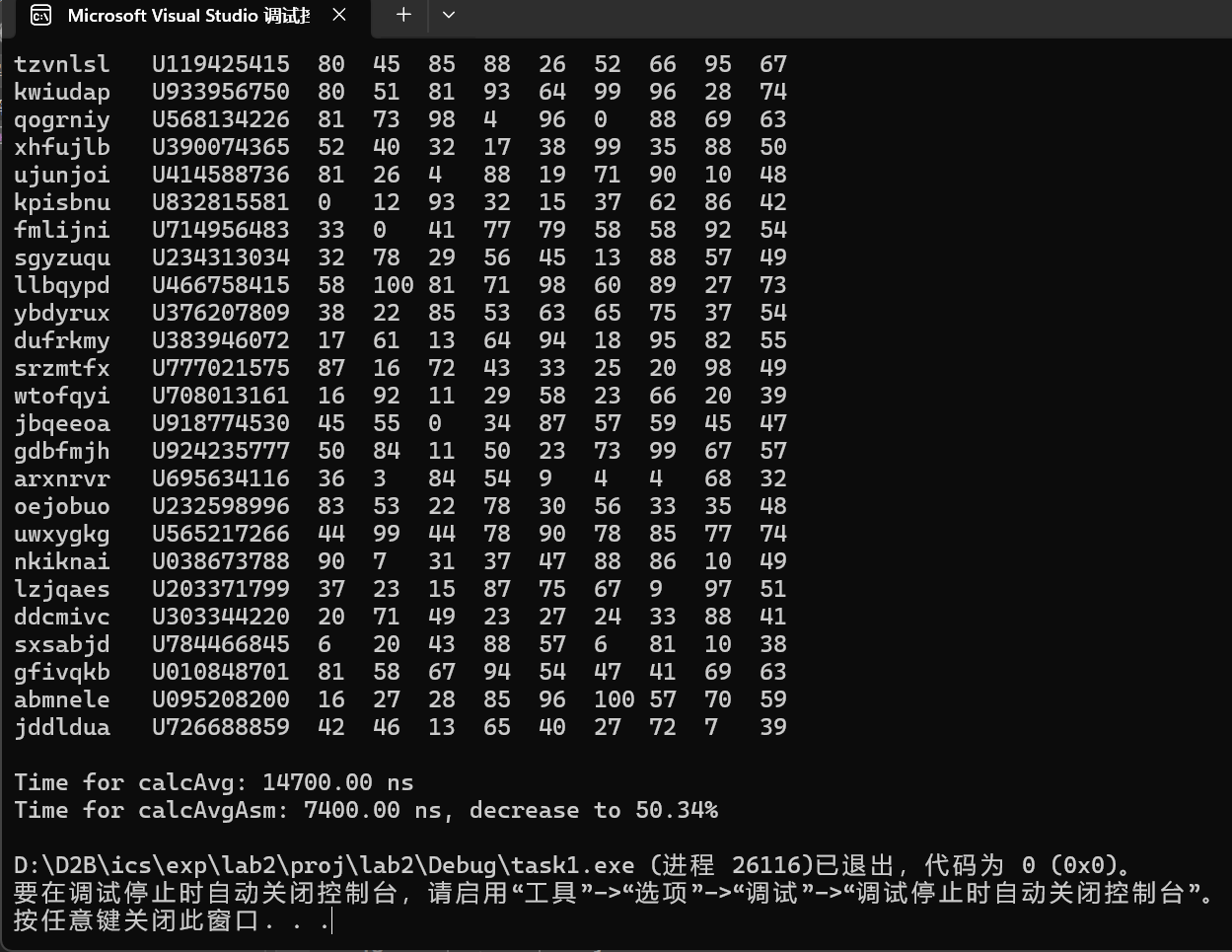
图表 5 任务1执行结果

**（2）实验任务2 的实验测试结果记录**



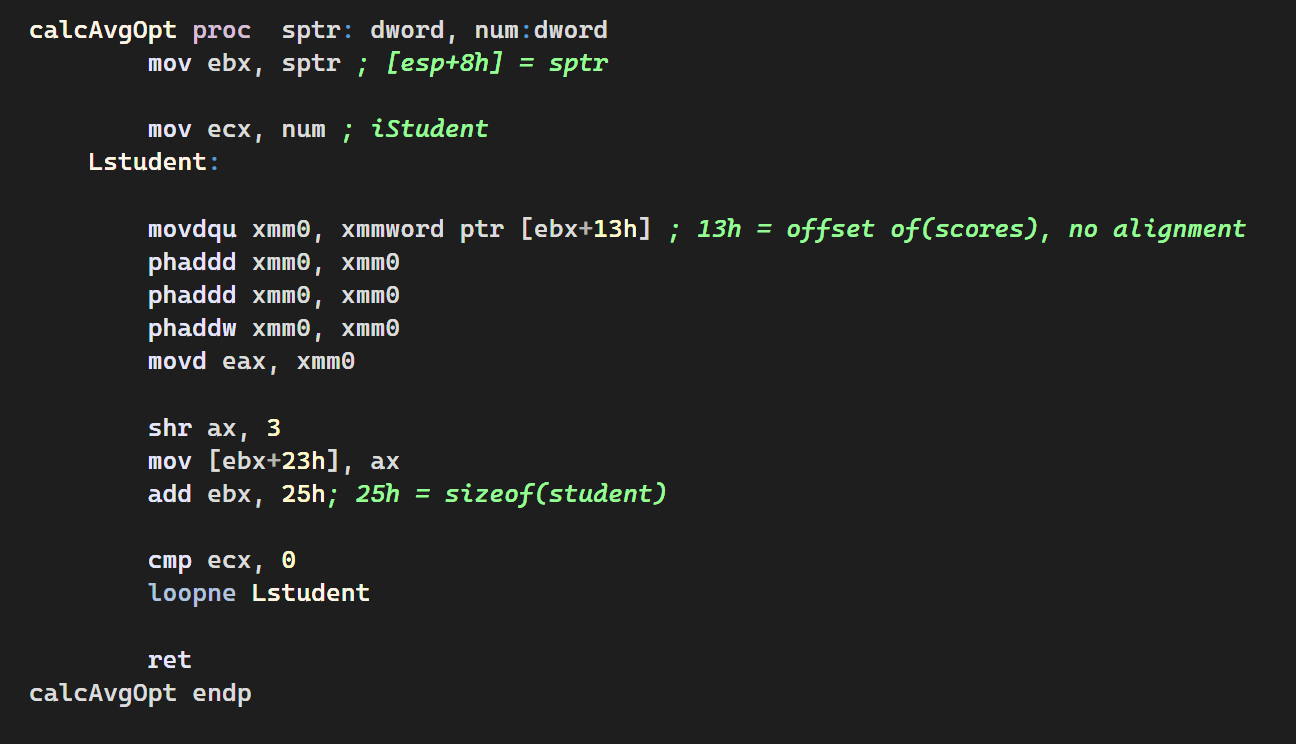
图表 6 用汇编语言编写的“计算平均成绩”函数

使用以上汇编程序对1000组数据进行计算，x86、debug环境下与高级语言运行结果比较，如下图运行时长为其50.34%，减少近一半：



图表 7 汇编程序对1000组数据进行计算结果

**（3）实验任务3 的实验测试结果记录、优化手段、优化原理分析**

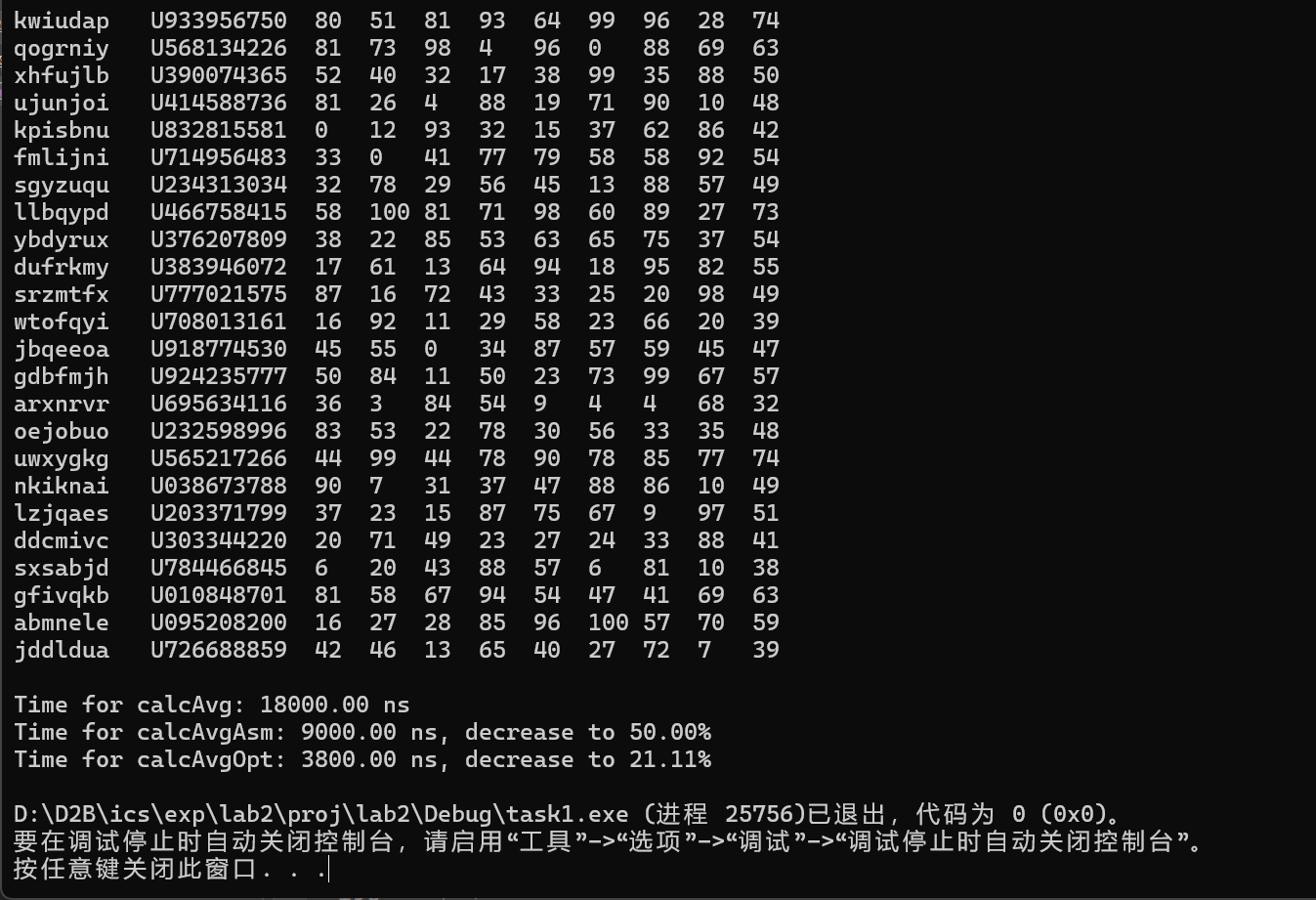


图表 8 汇编程序优化

主要优化点：

1. 尽量减少指令数量。由于指令执行流水线的特点，更少的指令意味着更少的执行时长。
2. 使用SSE指令phaddd和phaddw，将每个学生的8门成绩共16个字节（128位）加载至xmm寄存器中，进行水平方向的加法，既减少了访存次数，又避免了循环分支。SIMD指令计算加法更加高效。
3. 使用loop指令代替jmp，可以减少一条计数语句。
4. 优化了寄存器的使用效率。

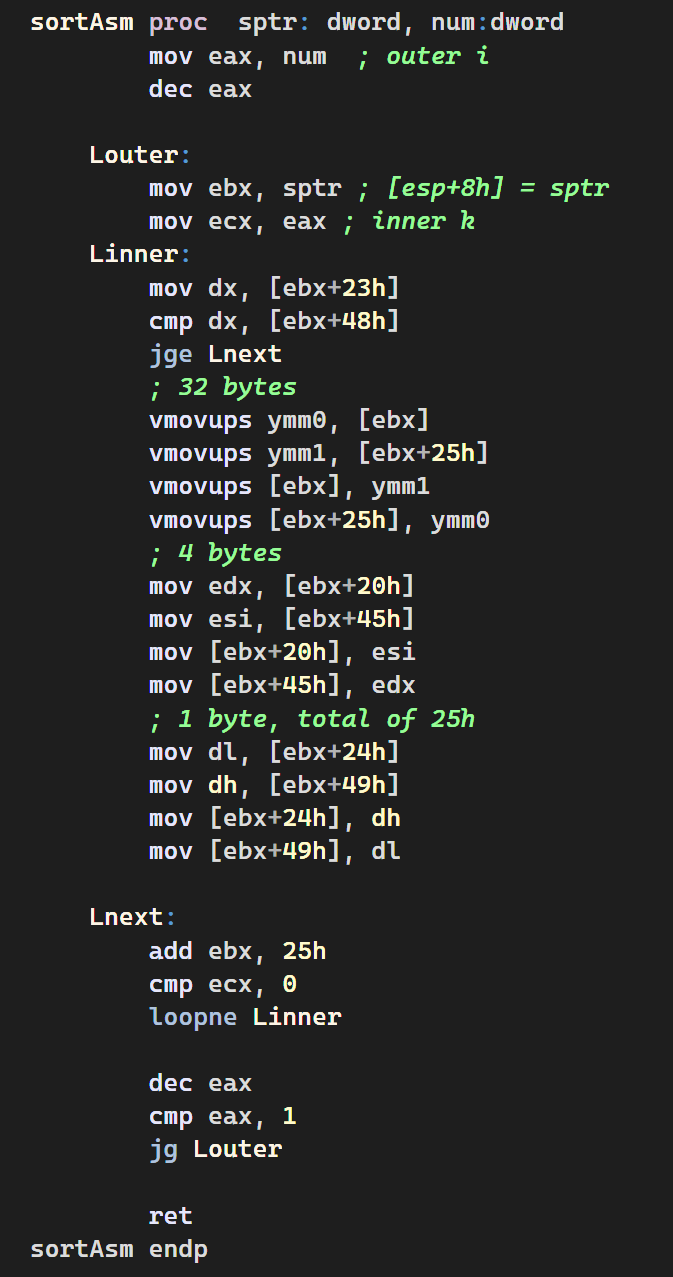
对1000组数据进行计算，运行结果显示，效果显著！优化后耗时是优化前的42%，是高级语言的21.11%。

****

图表 9 汇编程序优化后的运行结果

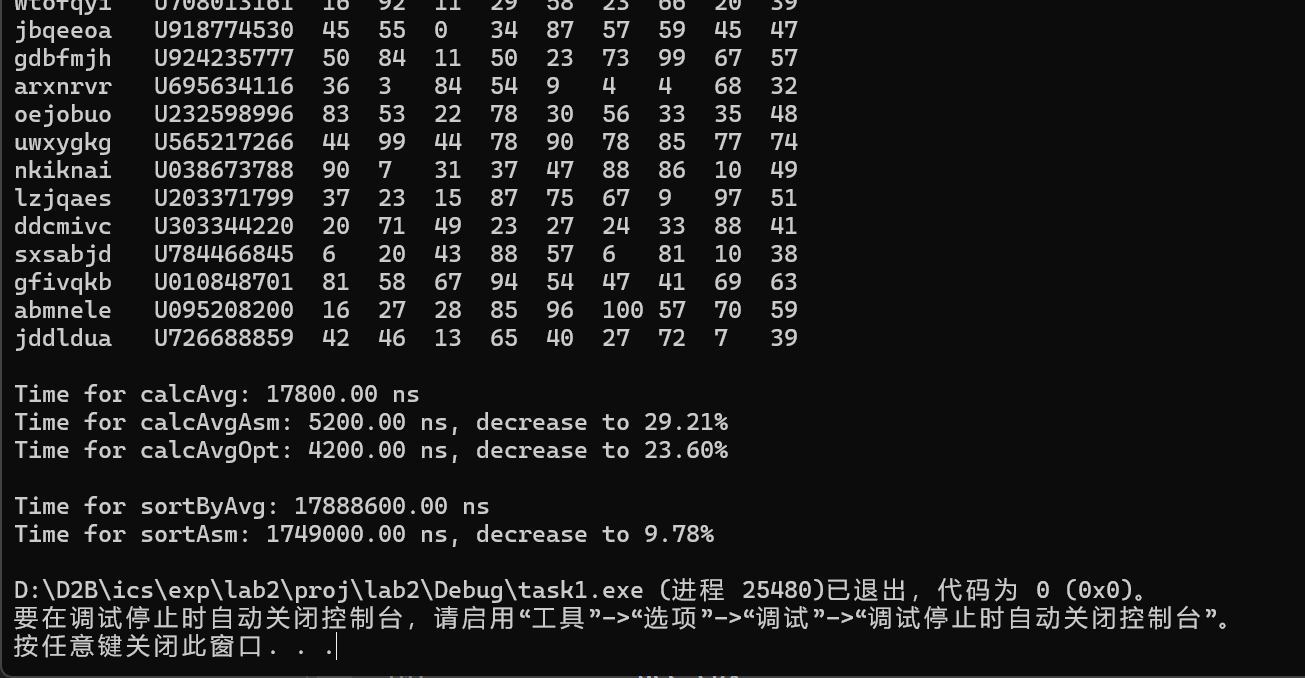
**（4）实验任务4 的实验测试结果记录、优化手段、优化原理分析**

优化思路：源程序耗时集中在冒泡排序时交换相邻的student存储单元上，因此要尽量减少这部分的汇编指令数量。因此，考虑使用256位（32字节）的ymm系列寄存器，结合32位寄存器对37字节的student存储单元进行交换操作，并尽量减少分支跳转。



图表 10 优化后的排序汇编程序

同样，将1000组数据赋值为2份，分别执行优化前后的两个算法，运行结果显示，优化后的耗时仅为优化前的9.78%，非常巨大的优化！



图表 11 排序算法优化后程序运行结果

**四、体会**

本次实验前，看到要编写汇编程序，我的内心是犯怵的，遑论对汇编程序进行优化。然而，当我真正开始接触汇编语言时，我发现它并非不可逾越的高山，而是一扇通往计算机底层世界的神秘大门。

在实验过程中，我逐渐熟悉了汇编语言的基本语法和指令集。虽然一开始对寄存器的使用、指令的格式以及程序的调试感到无从下手，但随着不断的实践和查阅资料，我开始能够理解程序的运行逻辑。

在这个过程中，我深刻体会到了汇编语言的高效性和对硬件的直接控制能力，我对计算机的工作原理的理解也更加深入，也感受到了老师们对课程的设计，非常感谢老师们！