华 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

实验名称：实验三 汇编语言与C语言混合编程

实验时间：2019.10.12

实验地点：南一楼116

指导教师：鲁宏伟

专业班级：信息安全1802班

学 号：U201814867

姓 名：王证儒

报告日期：2019.10.12

**成绩评定**

|  |  |  |
| --- | --- | --- |
| 实验完成质量（70分） | 报告撰写质量（30分） | 总成绩 |
| 实验步骤清晰、详细、深入，实验记录真实完整等 | 报告规范、完整、通顺、详实 |
|  |  |  |

**目 录**

[1.实验目的与要求 3](#_Toc22497991)

[2.实验内容 3](#_Toc22497992)

[3.实验过程 4](#_Toc22497993)

[3.1 开发环境 4](#_Toc22497994)

[3.2 实现方法 4](#_Toc22497995)

[3.3 任务1 5](#_Toc22497996)

[3.3.1 实现方法 5](#_Toc22497997)

[3.3.2 代码 7](#_Toc22497998)

[3.3.3 运行结果 10](#_Toc22497999)

[3.4 任务2 11](#_Toc22498000)

[3.4.1 实现方法 11](#_Toc22498001)

[3.4.2 代码 14](#_Toc22498002)

[3.4.3 运行结果 18](#_Toc22498003)

[4.总结与体会 19](#_Toc22498004)

# 1.实验目的与要求

1. 掌握汇编语言程序与C语言程序混合编程的方法；
2. 熟悉C编译器的基本优化方法;
3. 了解C语言编译器的命名方法，主、子程序之间参数传递的机制。

# 2.实验内容

任务1：在C语言序中嵌入汇编语言指令语句序列

对于实验二的程序进行改造，主控程序、以及输入输出等功能用C语言实现，学生姓名搜索和成绩计算用C程序中嵌入汇编指令语句序列的方式实现。

任务2：在C语言程序中调用汇编语言实现的函数

对于实验二的程序进行改造，主控程序、以及输入输出等功能用C语言实现，学生姓名搜索和成绩计算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：

（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。请大家选择白己熟悉的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，详细地描述采用的开发环境及其实现方法。

（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）尝试在C语言程序中不合理地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语言序、用好C编译器提供支持。

# 3.实验过程

## 3.1 开发环境

系统：Windows 10 专业版 64位

集成开发环境：Visual Studio 2019 Professional

编译器：MSVC 14.22.27905

语言：C++（兼容绝大部分C语法）

语言标准：C++17

解决方案平台：x86（32位，兼容部分16位寄存器指令）

内嵌汇编格式：Intel风格

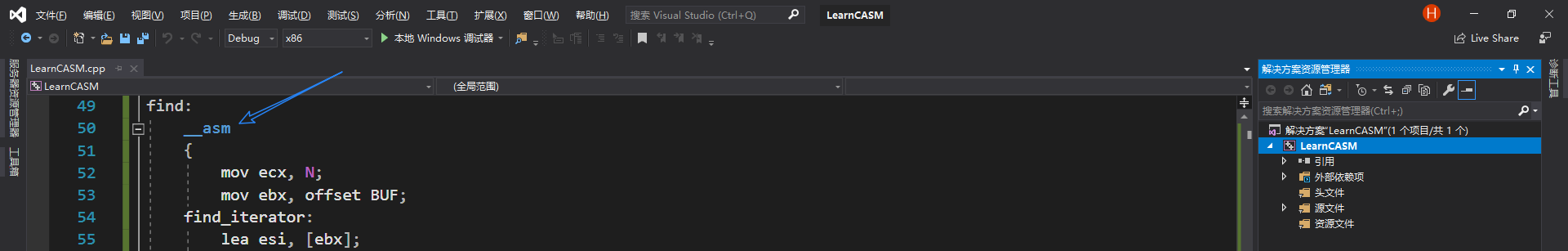


图3-1 开发环境

## 3.2 实现方法

考虑到调试的便捷性，并基于以下两点考虑选用了本次实验的平台：

①64位系统上兼容32位的SDK

②32位环境下兼容16位寄存器操作

使用\_\_asm{}来实现内嵌汇编，并可以直接引用外部C语言中的变量，也可以直接JMP到外部C语言的标签

对于原实验二中的汇编代码，大部分可以直接移植过来使用，但在涉及寻址的时候一般要做改变，因为80x86的地址总线是20位，而x86是32位的，所以此时要将寻址的16位寄存器改为32位寄存器（如把bx改为ebx，但与此同时要注意不要让ebx的高16位影响寻址）

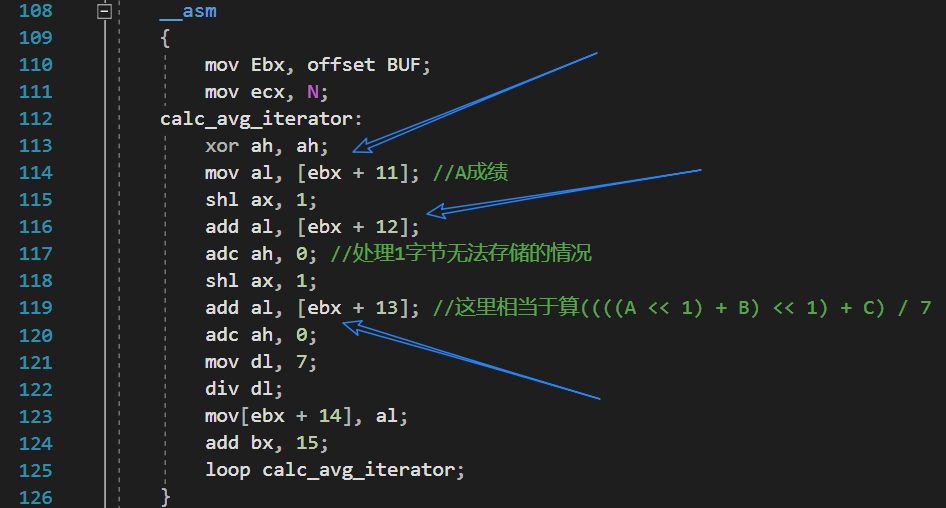


图3-2 16位汇编向32位汇编的移植

## 3.3 任务1

### 3.3.1 实现方法

1.数据部分均使用C语言定义，其中BUF为Data结构体数组，每个Data包含名字（char\*），四个成绩（BYTE），由于C语言中没有汇编中的dup伪指令，所以就在程序开头初始化了若干个‘TempValue’的数据。同时一些常量也使用了C中的#define进行宏定义

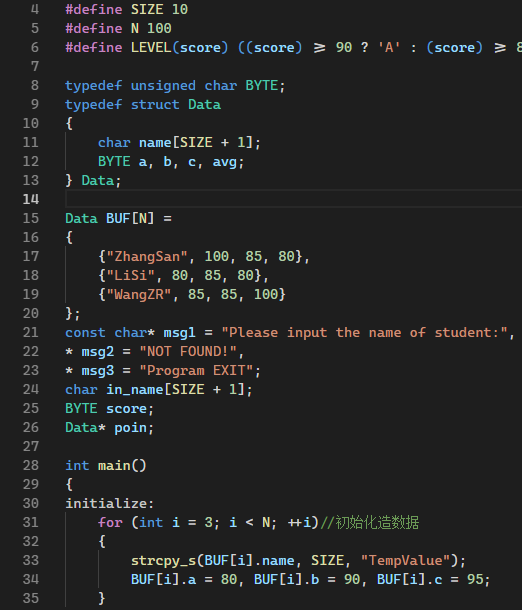


图3-3 数据定义及初始化

2.输入部分使用C的gets\_s函数输入，方便判断回车，同时避免缓冲区溢出攻击。同时判断输入是否合法和特殊操作也是使用C语言实现

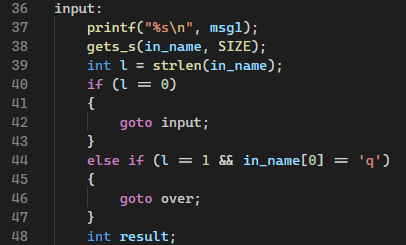


图3-4 输入及判断

3.查找部分使用内嵌汇编实现，大体上与实验二中的程序相同，除了部分地方（特别是寻址）的寄存器位宽要改，以及部分赋值时要进行类型强制转化

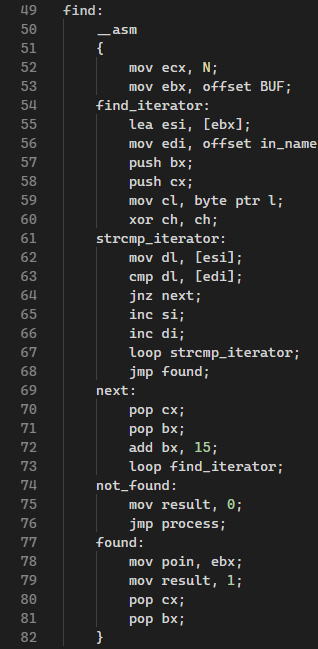


图3-5 查找部分

4.随后查找完成后将结果赋值给了外部C中的变量result，若查找到则result=1，否则result=0.之后程序再度交由C语言，若result=0，则输出提示信息，并goto到最开始的输入阶段，否则进入下一步处理

5.若查找到该学生，则计算其平均成绩和等级（汇编实现），并在C语言中输出分数和等级

6.调用汇编代码计算所有同学的平均成绩

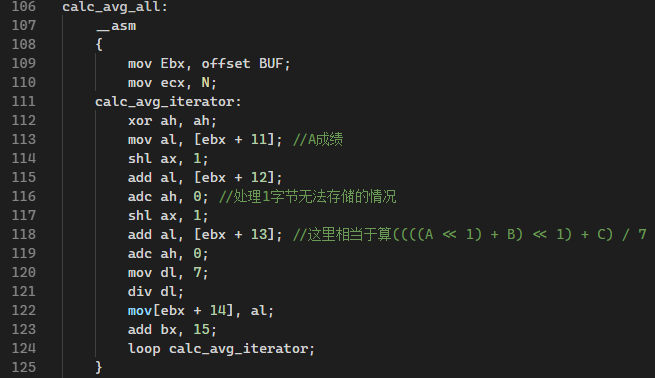


图3-6 计算所有同学的平均成绩

7.用C语言依次输出所有同学的平均分数并判断等级

### 3.3.2 代码

File: 3-1.c

001: #include <string.h>

002: #include <stdio.h>

003:

004: #define SIZE 10

005: #define N 100

006: #define LEVEL(score) ((score) >= 90 ? 'A' : (score) >= 80 ? 'B' : (score) >= 70 ? 'C' : (score) >= 60 ? 'D' : 'F')

007:

008: typedef unsigned char BYTE;

009: typedef struct Data

010: {

011: char name[SIZE + 1];

012: BYTE a, b, c, avg;

013: } Data;

014:

015: Data BUF[N] =

016: {

017: {"ZhangSan", 100, 85, 80},

018: {"LiSi", 80, 85, 80},

019: {"WangZR", 85, 85, 100}

020: };

021: const char\* msg1 = "Please input the name of student:",

022: \* msg2 = "NOT FOUND!",

023: \* msg3 = "Program EXIT";

024: char in\_name[SIZE + 1];

025: BYTE score;

026: Data\* poin;

027:

028: int main()

029: {

030: initialize:

031: for (int i = 3; i < N; ++i)//初始化造数据

032: {

033: strcpy\_s(BUF[i].name, SIZE, "TempValue");

034: BUF[i].a = 80, BUF[i].b = 90, BUF[i].c = 95;

035: }

036: input:

037: printf("%s\n", msg1);

038: gets\_s(in\_name, SIZE);

039: int l = strlen(in\_name);

040: if (l == 0)

041: {

042: goto input;

043: }

044: else if (l == 1 && in\_name[0] == 'q')

045: {

046: goto over;

047: }

048: int result;

049: find:

050: \_\_asm

051: {

052: mov ecx, N;

053: mov ebx, offset BUF;

054: find\_iterator:

055: lea esi, [ebx];

056: mov edi, offset in\_name;

057: push bx;

058: push cx;

059: mov cl, byte ptr l;

060: xor ch, ch;

061: strcmp\_iterator:

062: mov dl, [esi];

063: cmp dl, [edi];

064: jnz next;

065: inc si;

066: inc di;

067: loop strcmp\_iterator;

068: jmp found;

069: next:

070: pop cx;

071: pop bx;

072: add bx, 15;

073: loop find\_iterator;

074: not\_found:

075: mov result, 0;

076: jmp process;

077: found:

078: mov poin, ebx;

079: mov result, 1;

080: pop cx;

081: pop bx;

082: }

083: process:

084: if (result == 0)

085: {

086: printf\_s("%s\n", msg2);

087: goto input;

088: }

089: \_\_asm

090: {

091: mov ebx, dword ptr poin;

092: xor ah, ah;

093: mov al, [ebx + 11]; //A成绩

094: shl ax, 1;

095: add al, [ebx + 12];

096: adc ah, 0; //处理1字节无法存的情况

097: shl ax, 1;

098: add al, [ebx + 13]; //这里相当算((((A << 1) + B) << 1) + C) / 7

099: adc ah, 0;

100: mov dl, 7;

101: div dl;

102: mov score, al;

103: }

104: printf("%s's score is %d\n", in\_name, score);

105: printf("%s's level is %c\n", in\_name, LEVEL(score));

106: calc\_avg\_all:

107: \_\_asm

108: {

109: mov Ebx, offset BUF;

110: mov ecx, N;

111: calc\_avg\_iterator:

112: xor ah, ah;

113: mov al, [ebx + 11]; //A成绩

114: shl ax, 1;

115: add al, [ebx + 12];

116: adc ah, 0; //处理1字节无法存储的情况

117: shl ax, 1;

118: add al, [ebx + 13]; //这里相当于算((((A << 1) + B) << 1) + C) / 7

119: adc ah, 0;

120: mov dl, 7;

121: div dl;

122: mov[ebx + 14], al;

123: add bx, 15;

124: loop calc\_avg\_iterator;

125: }

126: for (int i = 0; i < N; ++i)

127: {

128: printf("%d:\n%s's average score is %d\n", i, BUF[i].name, BUF[i].avg);

129: printf("%s's level is %c\n\n", BUF[i].name, LEVEL(BUF[i].avg));

130: }

131: over:

132: printf\_s("%s\n", msg3);

133: return 0;

134: }

### 3.3.3 运行结果

运行截图：

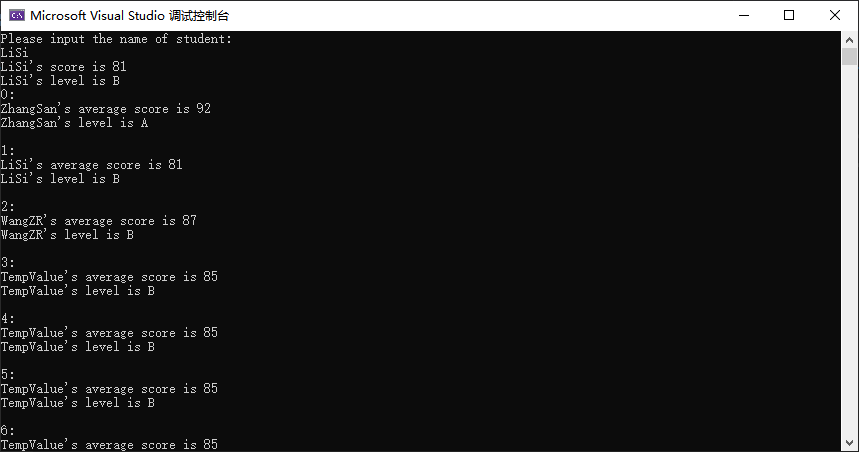


图3-7 运行截图

## 3.4 任务2

### 3.4.1 实现方法

这个任务我选择将汇编单独作为一个文件来调用，最后直接调用汇编器将asm文件翻译为obj文件（使用ml命令）

这些操作可以在VS2019中实现。这里我们新建一个VS2019控制台应用



图3-8 新建控制台应用

随后我们可以将初始的源文件的后缀改为.c，这样做是为了调用上方便，如果是.cpp的话后面在直接调用函数的时候略有不同。接着我们在源文件中再新建一个汇编文件（如3-2.asm）：

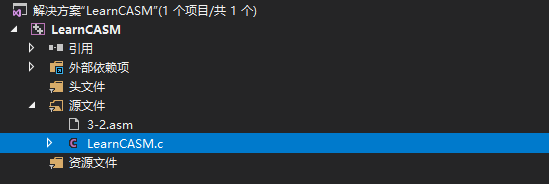


图3-9 源文件目录

由于VS默认不会去处理asm文件，所以我们需要进一步设置：对3-2.asm文件右键-属性，如下，可以看见该文件并不参与生成。

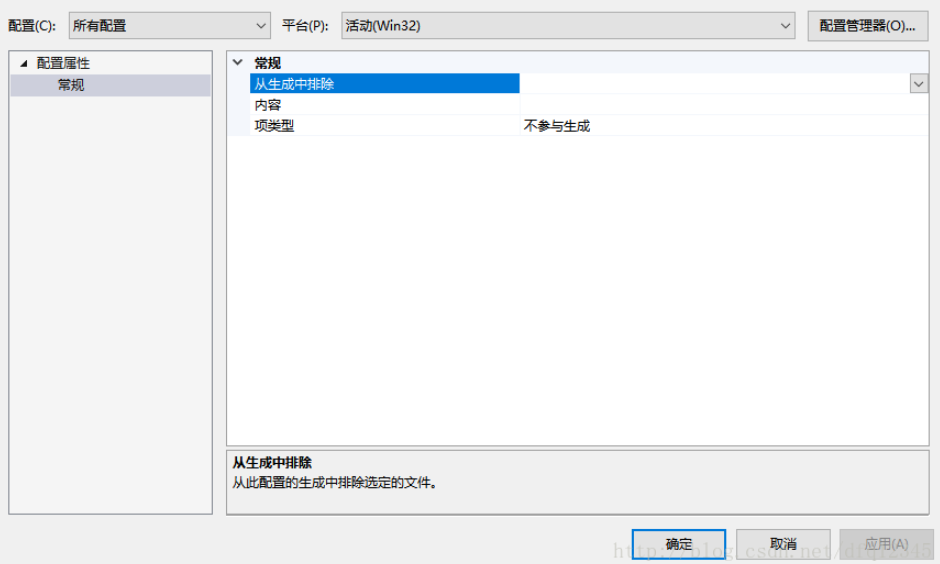


图3-10 汇编文件属性

修改属性如下：

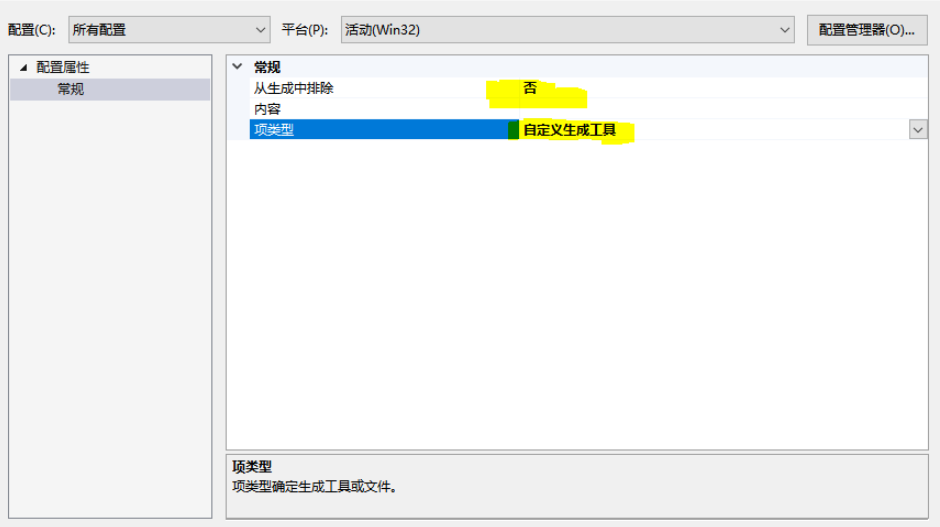


图3-11 修改汇编文件属性

随后点击应用，可以看到左侧属性页上增加了一栏配置“自定义生成工具”，点击修改为以下的内容：

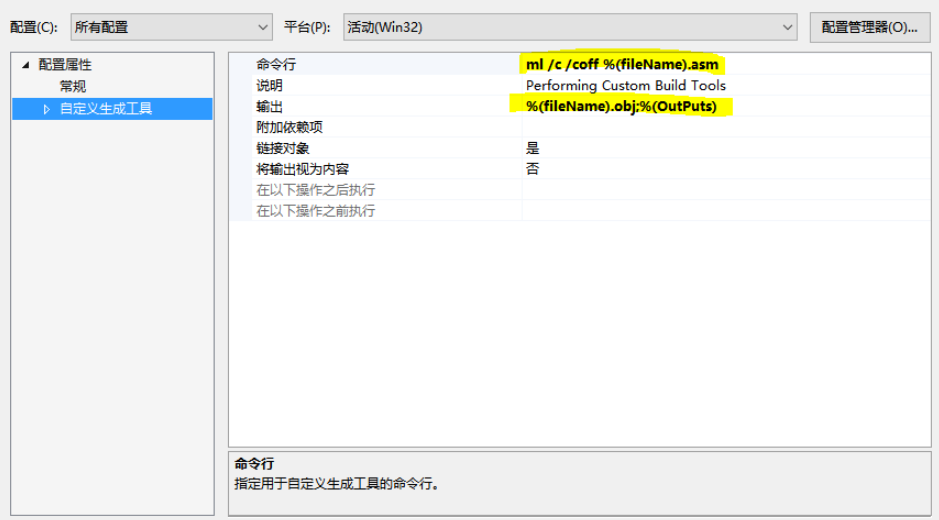


图3-12 修改自定义生成工具内容

这里的-ml是VS中的汇编和连接程序，用于将.asm转换为.obj文件并连接，/c 参数表示只汇编不连接，/coff表示使用的文件格式。

随后我们对asm文件正常编辑，可以进行如下测试：

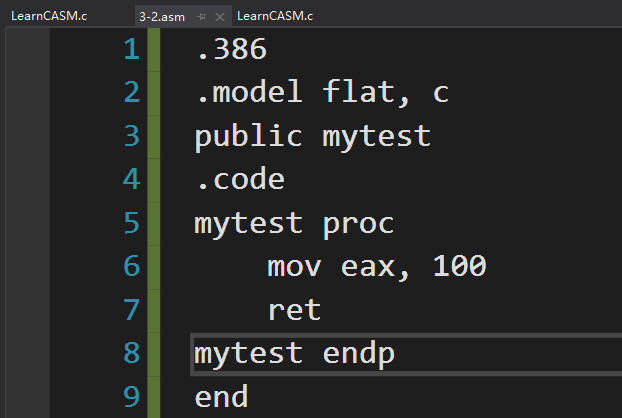


图3-13 测试汇编代码

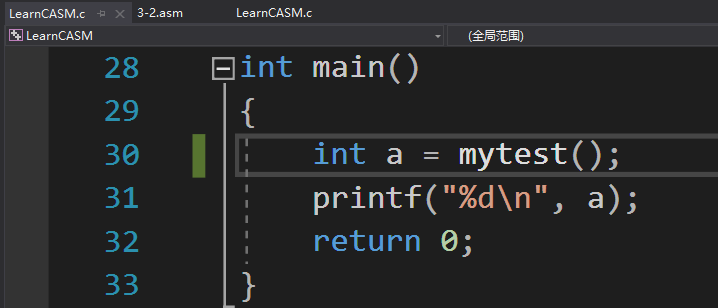


图3-14 测试主程序

需要注意的是汇编程序中需要显式声明public mytest，表明将mytest共享到外部，这样我们的C主程序才能调用到这个函数。

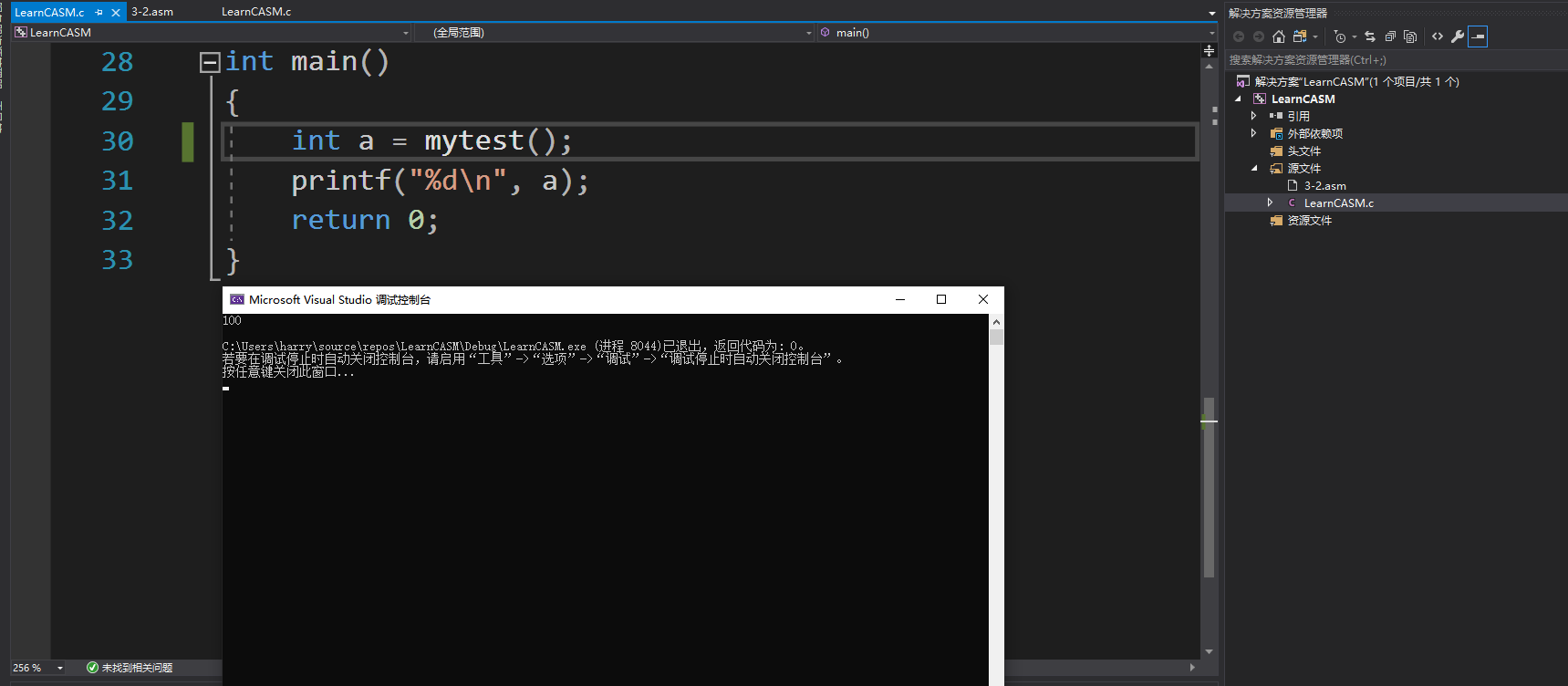


图3-15 测试运行截图

可以发现测试是正确的

我们先写出仿照任务1的代码，然后将内嵌汇编部分写到3-2.asm中的子过程部分即可。

1. 数据部分仍然定义在C语言的主程序中，通过汇编中的EXTERN关键字来实现共享，这样就可以直接访问到C语言主程序中的全局变量

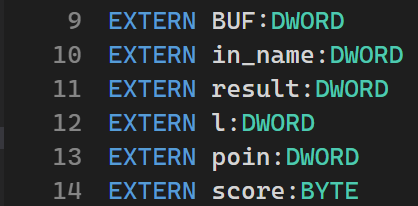


图3-16 汇编引用外部变量的方法

1. 子过程定义在汇编程序中，然后通过PUBLIC关键字来实现共享，这样可以直接在C主程序中调用

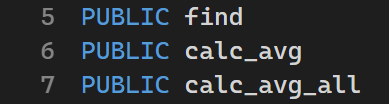


图3-17 汇编声明共享子过程的方法

### 3.4.2 代码

代码如下：

File: 3-2.c

01: #include <string.h>

02: #include <stdio.h>

03:

04: #define SIZE 10

05: #define N 100

06: #define LEVEL(score) ((score) >= 90 ? 'A' : (score) >= 80 ? 'B' : (score) >= 70 ? 'C' : (score) >= 60 ? 'D' : 'F')

07:

08: typedef unsigned char BYTE;

09: typedef struct Data

10: {

11: char name[SIZE + 1];

12: BYTE a, b, c, avg;

13: } Data;

14:

15: Data BUF[N] =

16: {

17: {"ZhangSan", 100, 85, 80},

18: {"LiSi", 80, 85, 80},

19: {"WangZR", 85, 85, 100}

20: };

21: const char\* msg1 = "Please input the name of student:",

22: \* msg2 = "NOT FOUND!",

23: \* msg3 = "Program EXIT";

24: char in\_name[SIZE + 1];

25: BYTE score;

26: Data\* poin;

27: int l, result;

28:

29: int main()

30: {

31: initialize:

32: for (int i = 3; i < N; ++i)//初始化造数据

33: {

34: strcpy\_s(BUF[i].name, SIZE, "TempValue");

35: BUF[i].a = 80, BUF[i].b = 90, BUF[i].c = 95;

36: }

37: input:

38: printf("%s\n", msg1);

39: gets\_s(in\_name, SIZE);

40: l = strlen(in\_name);

41: if (l == 0)

42: {

43: goto input;

44: }

45: else if (l == 1 && in\_name[0] == 'q')

46: {

47: goto over;

48: }

49: find:

50: find();

51: process:

52: if (result == 0)

53: {

54: printf\_s("%s\n", msg2);

55: goto input;

56: }

57: calc\_avg();

58: printf("%s's score is %d\n", in\_name, score);

59: printf("%s's level is %c\n", in\_name, LEVEL(score));

60: printf("\nPress any key to print ALL students' scores and levels...\n");

61: getch();

62: calc\_avg\_all:

63: calc\_avg\_all();

64: for (int i = 0; i < N; ++i)

65: {

66: printf("%d:\n%s's average score is %d\n", i, BUF[i].name, BUF[i].avg);

67: printf("%s's level is %c\n\n", BUF[i].name, LEVEL(BUF[i].avg));

68: }

69: over:

70: printf\_s("%s\n", msg3);

71: return 0;

72: }

File: 3-2.asm

01: .386

02: .model flat, c

03: N equ 100

04:

05: PUBLIC find

06: PUBLIC calc\_avg

07: PUBLIC calc\_avg\_all

08:

09: EXTERN BUF:DWORD

10: EXTERN in\_name:DWORD

11: EXTERN result:DWORD

12: EXTERN l:DWORD

13: EXTERN poin:DWORD

14: EXTERN score:BYTE

15:

16: .code

17: find proc

18: mov ecx, N;

19: mov ebx, offset BUF;

20: find\_iterator:

21: lea esi, [ebx];

22: mov edi, offset in\_name;

23: push bx;

24: push cx;

25: mov cl, byte ptr l;

26: xor ch, ch;

27: strcmp\_iterator:

28: mov dl, [esi];

29: cmp dl, [edi];

30: jnz next;

31: inc si;

32: inc di;

33: loop strcmp\_iterator;

34: jmp found;

35: next:

36: pop cx;

37: pop bx;

38: add bx, 15;

39: loop find\_iterator;

40: not\_found:

41: mov result, 0;

42: ret;

43: found:

44: mov poin, ebx;

45: mov result, 1;

46: pop cx;

47: pop bx;

48: ret;

49: find endp

50:

51: calc\_avg proc

52: mov ebx, dword ptr poin;

53: xor ah, ah;

54: mov al, [ebx + 11]; //A成绩

55: shl ax, 1;

56: add al, [ebx + 12];

57: adc ah, 0; //处理1字节无法存的情况

58: shl ax, 1;

59: add al, [ebx + 13]; //这里相当算((((A << 1) + B) << 1) + C) / 7

60: adc ah, 0;

61: mov dl, 7;

62: div dl;

63: mov score, al;

64: ret;

65: calc\_avg endp

66:

67: calc\_avg\_all proc

68: mov ebx, offset BUF;

69: mov ecx, N;

70: calc\_avg\_iterator:

71: xor ah, ah;

72: mov al, [ebx + 11]; //A成绩

73: shl ax, 1;

74: add al, [ebx + 12];

75: adc ah, 0; //处理1字节无法存储的情况

76: shl ax, 1;

77: add al, [ebx + 13]; //这里相当于算((((A << 1) + B) << 1) + C) / 7

78: adc ah, 0;

79: mov dl, 7;

80: div dl;

81: mov[ebx + 14], al;

82: add bx, 15;

83: loop calc\_avg\_iterator;

84: ret;

85: calc\_avg\_all endp

86: end

### 3.4.3 运行结果

运行截图：

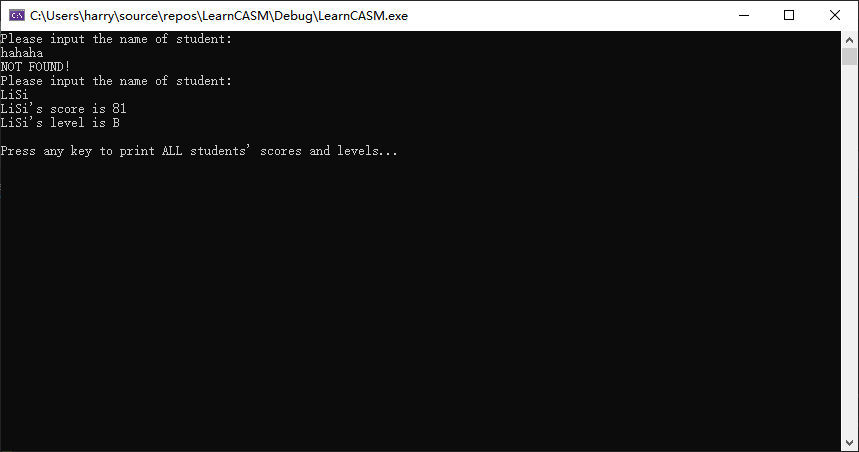


图3-18 运行截图1



图3-19 运行截图2

# 4.总结与体会

这次实验，我了解并掌握了在C/C++代码中内嵌汇编以及C/C++与汇编混合编程的技术方法。最初C语言的发明“拯救”了几乎所有还在使用汇编来编写操作系统和大型软件的开发者，也因此从C语言的设计哲学中我们也能看到很多汇编的影子，如内存管理、宏、子过程、跳转等，几乎都可以找到对应的部分。也因此，在C/C++中内嵌汇编是相对容易的——因为二者都比较底层（虽然C语言在发明之初仍然是“高级语言”），各种编译器都提供了不同的内嵌方法，如GCC、MSVC、Clang，并且可以直接互相引用变量，这无疑是十分方便的。在有些时候我们对性能需求非常高的时候（比如游戏渲染、高性能计算），我们既不希望全部都用高级语言实现——那样效率有待提高，也不会希望全部都用汇编语言实现——那样编码复杂度和调试难度都很大，因此可以采用折中的方法——对特定部分内嵌汇编来进行优化。这样可以兼顾执行效率和编码复杂度。而调用独立汇编程序中的子过程更加规范，这样可以让一个文件中只同时存在一种语言的代码，便于移植、调试。