

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实践**

**专业班级：计算机科学与技术202009班**

**学 号： U202015562**

**姓 名： 徐雨梦**

**指导教师： 张勇**

**实验时段： 2022年3月7日~4月29日**

**实验地点： 东九A315**

**原创性声明**

  本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：徐雨梦

报告日期：2022.6.6

实验报告成绩评定：

|  |  |  |  |
| --- | --- | --- | --- |
| 一（50分） | 二（35分） | 三（15分） | 合计（100分） |
|  |  |  |  |

指导教师签字：

                    日期：

**目录**

[一、程序设计的全程实践 2](#_Toc15367)

[1.1 目的与要求 2](#_Toc5528)

[1.2 实验内容 2](#_Toc21909)

[1.3 内容1.1的实验过程 2](#_Toc18500)

[1.3.1 设计思想 2](#_Toc20024)

[1.3.2 流程图 8](#_Toc133)

[1.3.3 源程序 12](#_Toc17038)

[1.3.4 实验记录与分析 20](#_Toc16724)

[1.4 内容1.2的实验过程 23](#_Toc18258)

[1.4.1 实验方法说明 23](#_Toc31635)

[1.4.2 实验记录与分析 24](#_Toc25743)

[1.5 小结 26](#_Toc20241)

[二、利用汇编语言特点的实验 27](#_Toc15128)

[2.1目的与要求 27](#_Toc24715)

[2.2实验内容 27](#_Toc12043)

[2.3实验过程 27](#_Toc21684)

[2.3.1实验方法说明 27](#_Toc1837)

[2.3.2实验记录与分析 30](#_Toc16531)

[2.4小结 34](#_Toc12382)

[三、工具环境的体验 35](#_Toc8629)

[3.1目的与要求 35](#_Toc3449)

[3.2实验过程 35](#_Toc17666)

[3.2.1 WINDOWS10下VS2019等工具包 35](#_Toc16214)

[3.2.2 DOSBOX下的工具包 36](#_Toc31038)

[3.2.3 QEMU下ARMv8的工具包 37](#_Toc28110)

[3.3小结 38](#_Toc4723)

[参考文献 39](#_Toc8690)

# 一、程序设计的全程实践

## 目的与要求

1.掌握汇编语言程序设计的全周期、全流程的基本方法与技术；

2.通过程序调试、数据记录和分析，了解影响设计目标和技术方案的多种因素。

## 实验内容

内容1.1：采用子程序、宏指令、多模块等编程技术设计实现一个较为完整的计算机系统运行状态的监测系统，给出完整的建模描述、方案设计、结果记录与分析。

内容1.2：初步探索影响设计目标和技术方案的多种因素，主要从指令优化对程序性能的影响，不同的约束条件对程序设计的影响，不同算法的选择对程序与程序结构的影响，不同程序结构对程序设计的影响，不同编程环境的影响等方面进行实践。

## 内容1.1的实验过程

### 设计思想

1. 算法思想

在该实验中，我们需要完成：①用户名与密码的验证，②计算数值并复制到相应区域，③展示数据，④按键提示下一步操作。

这些功能的实现：①功能由宏实现，该宏相当于一个字符串比较的子程序，在该宏中，我们采用了两个字符串的逐字比较的思想，具体实现看函数模块；②功能的实现采取了子程序来实现，主程序中对子程序进行调用得到结果；③功能采取子程序的方式进行实现，在主程序中采用调用来实现；④功能我们在主程序中实现。

主程序将整个程序和功能串起来：

①提示输入用户名和字符串，计数输入的次数

②调用宏比较用户名和字符串，如果正确就继续，如果不正确并且比较次数是否小于3，如果小于3，跳转回①，反之结束程序。

③计算number数组中的每个元素的SF的值，并且计数当前计算到了第几个

④调用compute子程序计算SF，比较计数的数值和10，如果小于10，跳转回③，反之进行下一步

⑤对每个元素的sf与100进行比较，同时计数当前进行到了第几个元素，如果sf小于100，跳转⑥，如果等于100，跳转⑦，如果大于100，跳转⑧

⑥调用copy\_num子程序对该区域和该元素的值进行复制，lowf的组数加一，当前的lowf区域的地址加3个sdwrod的长度。

⑦调用copy\_num子程序对该区域和该元素的值进行复制，midf的组数加一，当前的midf区域的地址加3个sdwrod的长度。

⑧调用copy\_num子程序对该区域和该元素的值进行复制，highf的组数加一，当前的highf区域的地址加3个sdwrod的长度。

⑨调用display子程序对该段数据进行输出

⑩提示输入字符，如果输入字符是R，跳转到③，反之退出。

整个过程的实现如上所示，主程序将所有的功能串联起来，子程序实现所要求的功能，其中需要注意的就是寄存器的取值！

1. 功能模块划分

3.1要求将汇编代码至少分解到两个不同的源文件中，也就是分解到不同的模块中去，同时还要求一定要使用子程序，因此我在本实验中分解为了main.asm和home.asm两个模块，一个用来保存所要求实现的功能的子程序，一个用来作为整体功能的实现，也就是功能的集装，使得程序能够调用模块的子程序并实现它应该实现的功能。

1. 模块说明

①home.asm

该模块保存了compute子程序、copy\_num子程序、display子程序和一些必要的段。

该实验的功能2和3要求可以计算每一组采集到的状态信息的sf并进行复制，复制完毕后将MIDF存储区的数据显示出来，要求2中表示这两个功能用子程序来实现，因此，我将这部分的实现放在了另一个模块中，与整体的代码相分离，在main模块中采取调用的方式调用该模块中的子程序和实现相应的功能。

②main.asm

该模块包含了main主程序，以及引用了所需要的库文件和函数，并设置了程序所必要的一些数据，是整个实验的执行开始文件。

所有功能都是在该模块中实现的，所要实现的功能1的宏定义在main这个模块中，功能1的实现的代码、功能2、3对另一个模块的子程序的调用、功能4的实现的代码，都在该模块中。

有关数据类型的定义和相关数据的说明，比如LOWF、MIDF、HIGHF等、结构体变量number的定义等，都在main这个模块中说明。

1. 模块之间公共符号说明

①compute

包含了四个参数：a,b.c,op，a、b、c都是有符号数，为sdword类型，op为dword类型，传递参数时，将一个结构体变量的SDA、SDB、SDC分别传递给a、b、c，将SF的地址传递给op。

②copy\_num

包含了两个参数：op,op\_num，都是dword类型，传递参数时，将对应的要存储的LOWF、MIDF、HIGHF的地址传递给op，将结构体变量的地址传递给op\_num。

③display

包含了两个参数：num、op，num是byte类型，op为dword类型，在调用时，将MIDF要输出的数字的组数传递给num，将MIDF的地址传递给op。

1. 段
2. Moder段声明了①所需要调用的函数和子程序的名称，②结构体的定义，③宏定义的字符串比较。
3. Data段定义了：①所需要的提示语句；②格式语句；③结构体变量及其初始化；④所需要的数据段，比如所要保存数据的LOWF/MIDF/HIGHF等数据段。
4. Code段完成了代码实现，在home模块中，实现了所需要的功能的子程序，在main模块中，实现了所需要的主程序。
5. 函数
6. 宏定义字符串比较

1）宏定义：string\_compare macro st1,st2,lop,zero\_compare,exit\_not,exit\_equal,accomplish

2）宏说明：该宏实现对两字符串st1和st2的比较，如果两个字符串相等，就标志全局变量flag为1，否则标志flag为0。lop,zero\_compare,exit\_not,exit\_equal,accomplish这几个参数是宏中的子段的定义。

3）宏实现：

传递过去的字符串为st1和st2。

采用ebp在这里进行计数，也就是下标，采用循环进行每一个字符的比较，首先先将以st1为偏移首址，ebp为组内偏移地址的位置的值赋值给al，将相同的相对位置的st2中的字符与al比较，如果不相等，直接跳转到exit\_not代码部分，该部分将标志flag置0并且跳转出该宏，也就是宏的结尾；如果相等，跳转到zero\_compare部分，该部分将检验两个字符串有没有到结尾，如果到了结尾，即两个字符串相等，将跳转到exit\_equal部分，将标志flag置为1并且调转到宏的结尾；如果不等于，那么将计数的ebp加一后跳转回比较的位置，也就是循环下一个字符的比较。

1. Compute计算函数

1）函数定义：compute proc,a:sdword,b:sdword,c\_num:sdword,op:dword

2）函数说明：该子程序实现所要求的计算，a对应SDA，b对应SDB，c对应SDC，op对应SF的地址，该子程序将计算的结果保存到op对应的地址的位置。

3）函数实现：

将a的值保存在eax中，对eax依次执行imul 5、add b、add 100、sub c、sar 7的操作，最后的结果保存在eax中，将op保存在ecx中，采用寄存器间接赋值将eax的结果保存在对应位置中。

1. Copy\_num复制数据函数

1）函数定义：copy\_num proc,op:dword,op\_num:dword

2）函数说明：将op\_num中的SDA、SDB、SDC复制到op中

3）函数实现：

Op\_num和op是地址。首先将op\_num复制给eax，接下来要访问eax的第2、3、4个数据，因此，我们需要先将eax所指的位置后移至该位置，在3.1中需要add 9，而在3.2中我们需要add 12，因为由于编译器的要求，采用c语言实现的时候需要对齐，而汇编中按照字节对齐。

再将op的值复制给ecx，采用寄存器间接访问的寻址方式，利用edx为中间变量，将eax位置的值赋值给ecx所保存的位置上，同时采用局部变量t进行计数，如果t小于3，就将eax和ecx都向后移动一个位置，也就是add 4，再重复上述步骤，如果等与3，复制完毕，ret。

1. Display展示数据函数

1）函数定义：display proc,num:byte,op:dword

2）函数说明：该函数实现对op地址对应的数组输出num组数据，每组数据有三个值

3）函数实现：

首先需要两层循环，外层用来循环几组数据，内层用来循环一个组的三个数据。

用counter\_wai和counter\_num来计数外层循环和内层循环，在外层循环每一次中，都需要将counter\_num清零，才能进入内层循环。

①将op地址保存在ecx中，将num的值保存在al中

②对counter\_wai进行初始化0

③lop循环：首先将counter\_wai和num比较，如果等于就已经输出完毕了，所以跳转到⑥的display\_exit，如果小于num，就对其加一并将counter\_num清零并开始内层循环

④首先将ecx的值保存给一个局部变量t，再将ecx所保存的位置上的值保存到edx，接下来调用printf函数将edx所保存的值输出，接下来再将t的值保存到ecx中，对ecx加4，对counter\_num加一

⑤比较counter\_num与3，如果小于就重复④，否则就继续③

⑥display\_exit直接ret

1. 寄存器
2. Eax
3. al用来保存字符串的值

在宏定义的字符串比较中，需要使用cmp对两个字符串的相应位置的字符进行比较，但是不能同时使用存储器变量，因此采用al保存其中的一个寄存器变量再进行比较。

1. 用来保存SF的地址

在调用compute子程序的时候需要将SF的地址传递给函数的参数，ebx用来保存number的各个变量的首地址，eax就是ebx后移21位。

1. 保存变量值

在compute函数中，我们需要计算所要求的公式，也就是传递进去的a，b，c进行相关数学操作，在这个函数中，我们采用a来作为主要的数值来进行计算，但由于不能同时使用存储器变量，因此我们需要将参数先保存在eax中，在该函数中，eax保存了a的值，接下来对eax进行imul 5、add b、add 100、sub c\_num、sar 7的操作，以此得到我们所要获取的SF的值。

1. 保存所要复制的位置的地址

在copy\_num子程序中，我们所要传递的第二个参数是所要复制的结构体元素的首址，在子程序中，用eax保存该值，在mov中，采用寄存器间接寻址的方式进行赋值。

1. Al保存所要输出的数据的组数

在display子程序中，我们需要传递所要输出的数据的组数给第一个参数，但是由于该参数无法与所定义的用来计数的局部变量进行比较，因此需要将该值保存在al中。

需要注意的是，在该子程序中，每一层内层循环都会调用一次printf，外层循环也需要，在调用过程中，会改变eax的值，也就会改变al的值，因此每次调用完printf之后就需要将num重新赋值给al。

1. Ebx
2. 保存结构体变量的地址

在main的主程序中，需要实现对其他子程序的调用和其他功能。其中一个功能为计算每一个变量的SF，该功能通过调用compute子程序来实现，在循环计算number数组中的每一个变量的SF的值的时候，我们需要将SF的地址在参数列表中传递给子程序，ebx用来保存每次所要计算的结构体变量的首址，在每次循环中，该首址就会移动一个结构体变量的地址长度。

同时还有一个作用：用作结构体变量的寄存器间接寻址访问中的寄存器保存变量首址，以此用来访问该变量中的SDA、SDB、SDC，将其作为参数传递给compute子程序中。

1. 调用copy\_num时传递参数

在copy\_num中需要将numbe数组中的每个元素中的SDA、SDB、SDC保存到相应的存储区，在该过程中，需要向copy\_num传递元素的地址，ebx用来保存该地址，初始化采用lea进行，循环过程中每次加25.

1. 调用display时传递参数

在调用display子程序的时候，我们需要传递所需要的显示的存储区的地址的参数，在本次实验中，也就是传递midf给该调用子程序的位置，因此我们将midf的地址保存在ebx中，再将ebx作为实参传递给该函数。

1. Ecx
2. 保存SF的地址用来进行寄存器间接访问

在compute子程序中，需要将计算好的存储在eax的地方的值保存在SF中去，我们传递了SF的地址进去，但由于mov不能直接对该地址形参传递数值，因此采用ecx对该地址进行保存，再采用寄存器间接访问的方式对该地址的值进行赋值。

1. 保存所要复制的区域的地址

在copy\_num子程序中，我们需要将所要复制的元素的地址和要复制的区域的地址都传递给参数，并采用寄存器间接寻址的方式对其赋值。

Ecx用来保存传递的第一个参数的值，也就是lowf/midf/highf当前所要复制的位置的地址。

1. 保存地址

在display子程序中，我们需要将所要输出的地址传递给参数，用ecx将传递的参数保存下来，接下来采用寄存器间接寻址的方式把该位置的值赋值给其他的变量，以此调用printf输出。

需要注意的是，在调用printf的过程中，ecx的值会发生改变，因此在调用printf之前需要将ecx 的值保存在一个局部变量中去，在调用完之后，再利用局部变量将ecx的值恢复，也可以采用压栈的方式。

1. Edx
2. 保存lowf、midf、highf的首址

在复制数据到相应的存储区的时候，我们需要将每次的要复制数据的去处的地址传递给copy\_num的参数中去，但是lowf、midf、highf所保存的地址不能改变，因此可以另外设置三个变量cur\_low、cur\_mid、cur\_high来保存当前复制的位置，因此首先需要对这三个变量赋初值，采用mov进行该操作，但是由于不能同时使用存储器变量，因此需要寄存器变量作为中间转接处，edx在该处起到此作用。

1. 保存中间值

在copy\_num子程序中，需要将一个位置的值复制到另一个位置，在该子程序中，我们采用寄存器间接寻址的方式进行该步骤，但是由于不能同时使用寄存器间接寻址，因此我们将所要复制的值先通过寄存器间接寻址保存到edx中去，然后将edx 的值复制给ecx所保存的位置。

1. 保存所要复制的值

在display子程序中，需要调用printf显示数值，但是寄存器间接寻址不能直接在printf中使用，因此需要先使用该寻址方式将数值保存在edx中，在调用printf中将edx作为参数传递。

1. Ebp
2. 字符串比较中用来计数

在字符串比较宏定义中，需要对字符逐个计数，在这里采用ebp进行计数，同时作为每次相对偏移地址用在变址寻址中。

### 流程图

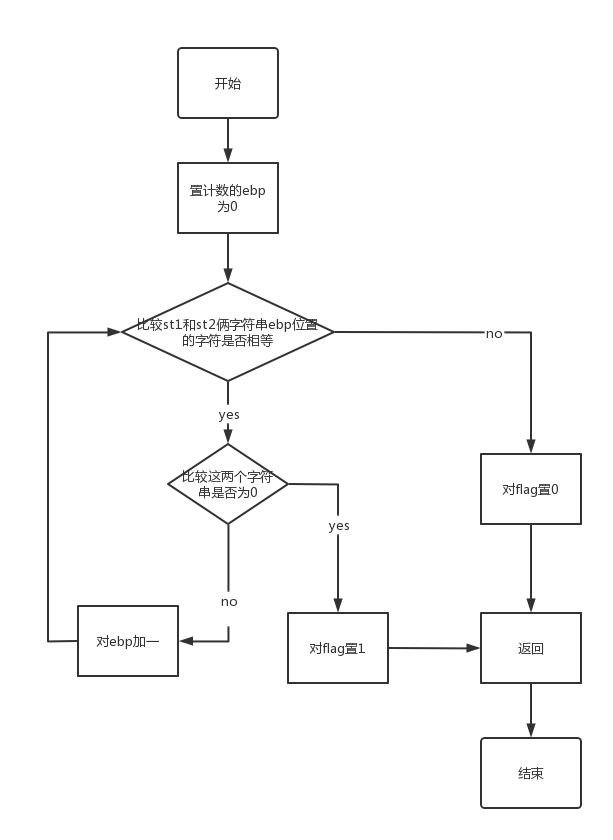


图1.1 宏定义的字符串比较

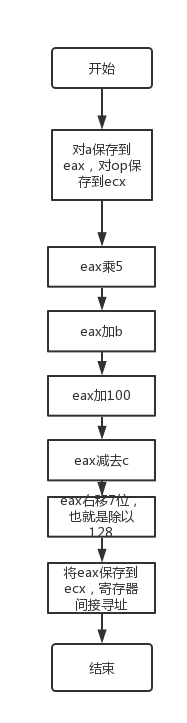


图1.2 compute子程序

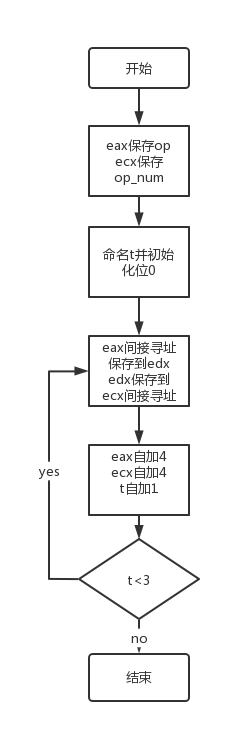


图1.3 copy\_num子程序

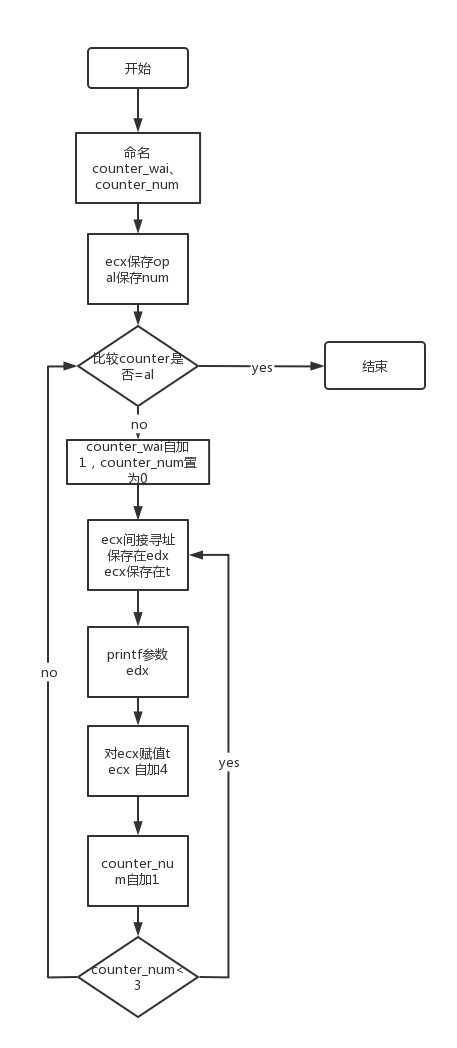


图1.4 display子程序

### 源程序

1. Main.asm

.686P

.model flat,c

INCLUDELIB KERNEL32.LIB

INCLUDELIB LIBCMT.LIB

INCLUDELIB LEGACY\_STDIO\_DEFINITIONS.LIB

scanf proto c:vararg

printf proto c:vararg

ExitProcess proto stdcall:dword

compute proto :sdword,:sdword,:sdword,:dword

copy\_num proto :dword,:dword

display proto :byte,:dword

SAMPLES STRUCT

SAMID DB 9 DUP(0) ;每组数据的流水号

SDA sdword 0 ;状态信息a

SDB sdword 0 ;状态信息b

SDC sdword 0 ;状态信息c

SF sdword 0 ;处理结果f

SAMPLES ENDS

string\_compare macro st1,st2,lop,zero\_compare,exit\_not,exit\_equal,accomplish

mov ebp,0

lop:

mov al,st1[ebp]

;mov ah,st2[ebp]

cmp st2[ebp],al

je zero\_compare

jne exit\_not

zero\_compare:

cmp al,0

je exit\_equal

inc ebp

jmp lop

exit\_not:

mov flag,0

jmp accomplish

exit\_equal:

mov flag,1

jmp accomplish

accomplish:

endm

.data

user\_name db "coding\_happily",0

user\_password db "xxxxxx20030308xxxx",0

input\_name db 20 dup(0),0

input\_password db 20 dup(0),0

error\_tip db "ERROR!please input again!",0DH,0AH,0

right\_tip db "ALL RIGHT! NEXT!",0DH,0AH,0

input\_tip1 db "please input your id: ",0

input\_tip2 db "please input your password: ",0

input\_tip3 db "please input R or Q :",0

exit\_tip db "three times !EXIT!",0DH,0AH,0

weibu db " ",0dh,0ah,0

disp\_tip db "numbers in midf are:",0ah,0dh,0

form1 db "%s",0

form2 db "%d ",0

form3 db "%s",0dh,0ah,0

number SAMPLES <,256809,-1023,1265,0>

SAMPLES <,200,356,34,0>

SAMPLES <,2500,300,100,0>

SAMPLES <,25641,2345,987651,0>

SAMPLES <,2568,-1023,126,0>

SAMPLES <,25809,-1023456,15,0>

SAMPLES <,12,34,56,0>

SAMPLES <,-1234,10236,1265,0>

SAMPLES <,1000,5000,-2700,0>

SAMPLES <,2003,308,0528,0>

flag sbyte 0

input\_num db 0

low\_num db 0

mid\_num db 0

high\_num db 0

cur\_mid dword 0

cur\_low dword 0

cur\_high dword 0

lowf sdword 30 dup(0),0

midf sdword 30 dup(0),0

highf sdword 30 dup(0),0

beichushu sdword 128

input\_continue byte 0

.stack 200

.code

main proc

input\_lop:

cmp input\_num,3 ;计数当前输了几次，三次就停止

je input\_error\_exit

inc input\_num

invoke printf,offset form1,offset input\_tip1

invoke scanf,offset form1,offset input\_name

invoke printf,offset form1,offset input\_tip2

invoke scanf,offset form1,offset input\_password

string\_compare user\_name,input\_name,lop1,zero\_compare1,exit\_not1,exit\_equal1,accomplish1 ;宏调用比较用户名

cmp flag,0

je input\_error

string\_compare user\_password,input\_password,lop2,zero\_compare2,exit\_not2,exit\_equal2,accomplish2 ;宏调用比较密码

cmp flag,0

je input\_error

invoke printf,offset form1,offset right\_tip

jmp lopk

input\_error: ;输入错误

invoke printf,offset form1,offset error\_tip

jmp input\_lop

input\_error\_exit: ;输入错误三次

invoke printf,offset form1,offset exit\_tip

jmp exit

lopk:

mov low\_num,0

mov mid\_num,0

mov high\_num,0

lea ebx,number

mov input\_num,0

lop10: ;算sf的值

cmp input\_num,10

je tongji ;算了10个数了，进入复制阶段

mov eax,ebx

add eax,21

invoke compute,[ebx].SAMPLES.SDA,[ebx].SAMPLES.SDB,[ebx].SAMPLES.SDC,eax

;invoke printf,offset form2,[ebx].SAMPLES.SDA

;invoke printf,offset form2,[EBX].SAMPLES.SDB

;invoke printf,offset form2,[EBX].SAMPLES.SDC

;invoke printf,offset form2,[EBX].SAMPLES.SF

;invoke printf,offset form1,offset weibu

inc input\_num

add ebx,25

jmp lop10

tongji:

lea ebx,number

mov input\_num,0

lea edx,midf

mov cur\_mid,edx

lea edx,lowf

mov cur\_low,edx

lea edx,highf

mov cur\_high,edx ;先保存要复制的位置

;sub edx,25

sub ebx,25

judge:

cmp input\_num,10

je shuchu

inc input\_num

add ebx,25

;add edx,25

cmp [ebx].SAMPLES.SF,100

jl low\_incr

je mid\_incr

jg high\_incr

low\_incr:

inc low\_num

invoke copy\_num, cur\_low,ebx

add cur\_low,12

jmp judge

mid\_incr:

inc mid\_num

invoke copy\_num, cur\_mid,ebx

add cur\_mid,12

jmp judge

high\_incr:

inc high\_num

invoke copy\_num ,cur\_high,ebx

add cur\_high,12

jmp judge

shuchu:

;invoke printf,offset form2,mid\_num

invoke printf,offset form3,offset disp\_tip

lea ebx,midf

invoke display,mid\_num,ebx

invoke printf,offset form1,offset weibu

invoke printf,offset form1,offset input\_tip3

invoke scanf,offset form1,offset input\_continue

cmp input\_continue[0],'R'

je lopk

jmp exit

exit:

invoke ExitProcess,0

main endp

END

1. Home .asm

.686P

.model flat,c

INCLUDELIB KERNEL32.LIB

INCLUDELIB LIBCMT.LIB

INCLUDELIB LEGACY\_STDIO\_DEFINITIONS.LIB

scanf proto c:vararg

printf proto c:vararg

ExitProcess proto stdcall:dword

SAMPLES STRUCT

SAMID DB 9 DUP(0) ;每组数据的流水号

SDA sdword 0 ;状态信息a

SDB sdword 0 ;状态信息b

SDC sdword 0 ;状态信息c

SF sdword 0 ;处理结果f

SAMPLES ENDS

string\_compare macro st1,st2,lop,zero\_compare,exit\_not,exit\_equal,accomplish

mov ebp,0

lop:

mov al,st1[ebp]

;mov ah,st2[ebp]

cmp st2[ebp],al

je zero\_compare

jne exit\_not

zero\_compare:

cmp al,0

je exit\_equal

inc ebp

jmp lop

exit\_not:

mov flag,0

jmp accomplish

exit\_equal:

mov flag,1

jmp accomplish

accomplish:

endm

.data

user\_name db "coding\_happily",0

user\_password db "xxxxxx20030308xxxx",0

input\_name db 20 dup(0),0

input\_password db 20 dup(0),0

error\_tip db "ERROR!please input again!",0DH,0AH,0

right\_tip db "ALL RIGHT! NEXT!",0DH,0AH,0

input\_tip1 db "please input your id: ",0

input\_tip2 db "please input your password: ",0

exit\_tip db "three times !EXIT!",0DH,0AH,0

weibu db " ",0dh,0ah,0

disp\_tip db "numbers in midf are:",0ah,0dh,0

form1 db "%s",0

form2 db "%d ",0

form3 db "%s",0dh,0ah,0

number SAMPLES <,256809,-1023,1265,0>

SAMPLES <,200,356,34,0>

SAMPLES <,2500,300,100,0>

SAMPLES <,25641,2345,987651,0>

SAMPLES <,2568,-1023,126,0>

SAMPLES <,25809,-1023456,15,0>

SAMPLES <,12,34,56,0>

SAMPLES <,-1234,10236,1265,0>

SAMPLES <,1000,5000,-2700,0>

SAMPLES <,2003,308,0528,0>

flag sbyte 0

input\_num db 0

low\_num db 0

mid\_num db 0

high\_num db 0

cur\_mid dword 0

cur\_low dword 0

cur\_high dword 0

lowf sdword 30 dup(0),0

midf sdword 30 dup(0),0

highf sdword 30 dup(0),0

beichushu sdword 128

.stack 200

.code

compute proc,a:sdword,b:sdword,c\_num:sdword,op:dword ;计算sf的子程序

mov eax,a

imul eax,5

add eax,b

add eax,100

sub eax,c\_num

sar eax,7

mov ecx,op

mov [ecx],eax

ret

compute endp

copy\_num proc,op:dword,op\_num:dword ;复制数据的子程序

local t:byte

mov t,0

mov ecx,op

mov eax,op\_num

add eax,9

lop:

mov edx,[eax]

mov [ecx],edx

add ecx,4

add eax,4

inc t

cmp t,3

jb lop

ret

copy\_num endp

display proc,num:byte,op:dword ;显示midf的数据

local counter\_num:byte,counter\_wai:byte,t:dword

mov ecx,op

mov al,num

mov counter\_wai,0

lop:

cmp counter\_wai,al

je display\_exit

inc counter\_wai

mov counter\_num,0

lop2:

mov edx,[ecx]

mov t,ecx

invoke printf,offset form2,edx

mov ecx,t

add ecx,4

inc counter\_num

cmp counter\_num,3

jb lop2

mov t,ecx

invoke printf,offset form1,offset weibu

mov ecx,t

mov al,num

jmp lop

display\_exit:

mov eax,0

ret

display endp

End

### 实验记录与分析

1. 实验环境条件

INTEL 处理器 2GHz，8G内存；WINDOWS10下VS2019社区版。

2. 汇编、链接中的情况

汇编过程中出现了3个问题，主要的一个问题是：寄存器的使用。

在整个实验过程中，我需要调用一些库函数，比如printf、scanf等函数，在这些函数中，eax、ecx等寄存器的值会使用并且发生变化，因此在这些函数调用完毕之后，就必须再对这些寄存器进行检查看是否还是我们所需要的值。

特别是当寄存器保存了一些变量，在调用了其他函数之后还需要使用的时候，需要在调用函数之前将寄存器的值保存到一个函数不会使用的变量中，一个最好的办法是使用自己定义的变量，因为很多函数使用的都是系统变量，自己定义的变量大概率是不会发生改变的，然后再在函数调用完成后把寄存器变量的值从自己定义的变量中重新赋值回去。这样才能使得我们使用的寄存器在逻辑上是正确的。

3. 程序基本功能的验证情况

（1）字符串比较功能的验证

①对输入错误的验证

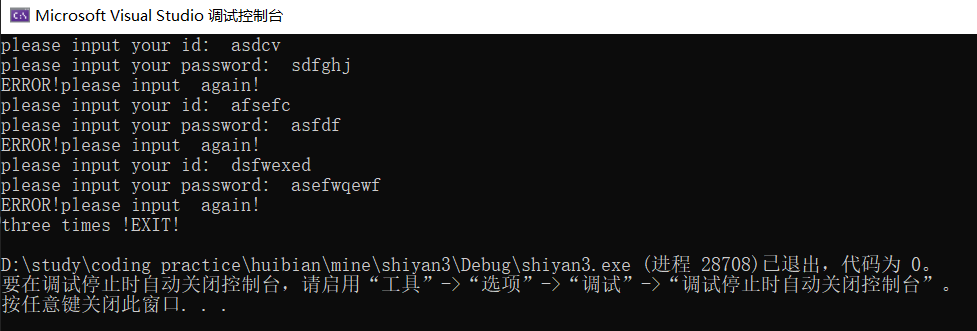


图1.5 输入错误三次的验证

②对输入次数的验证

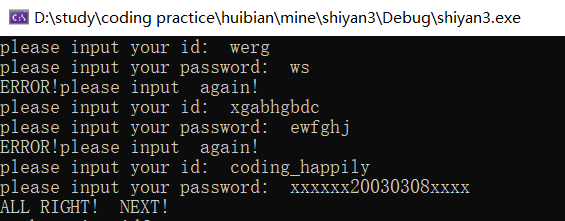


图1.6 输入正确的验证

（2）计算SF值并复制和展示的验证

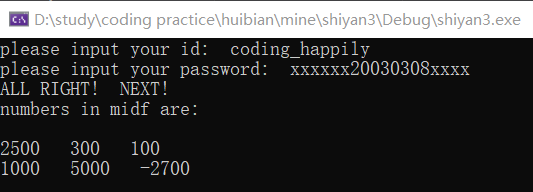


图1.7输出midf区域的值

（3）提示输入并且按照用户选择继续执行的验证

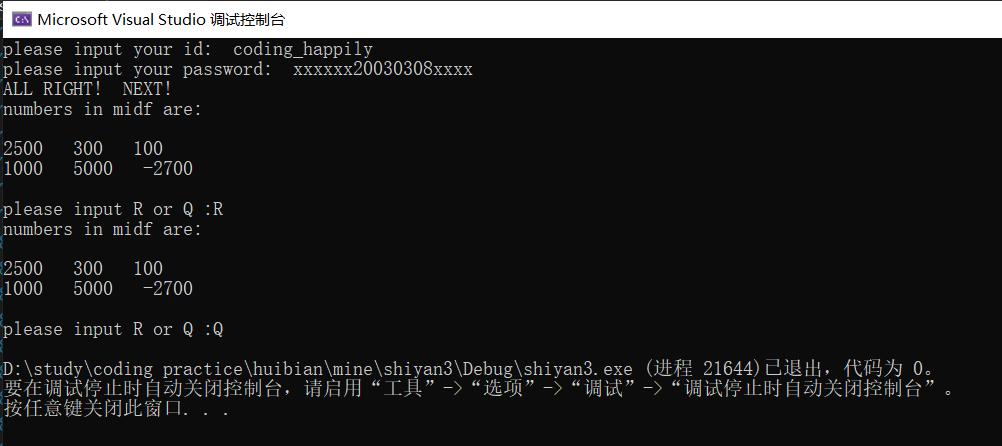


图1.8 功能四的验证

（4）3.2c语言与汇编结合的验证

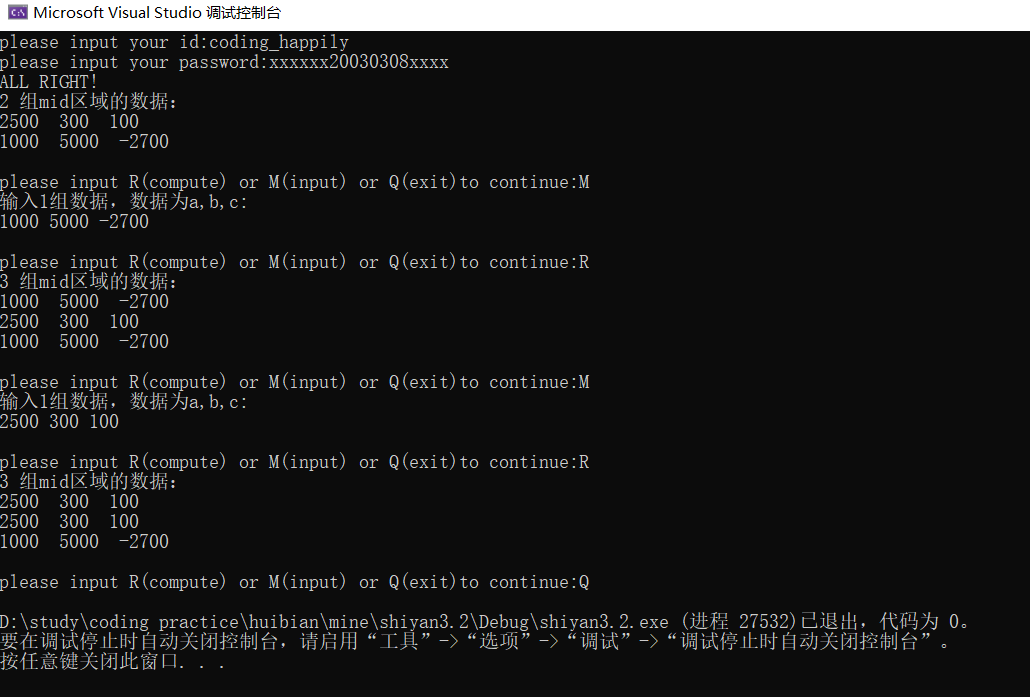


图1.9 主函数使用汇编的功能验证

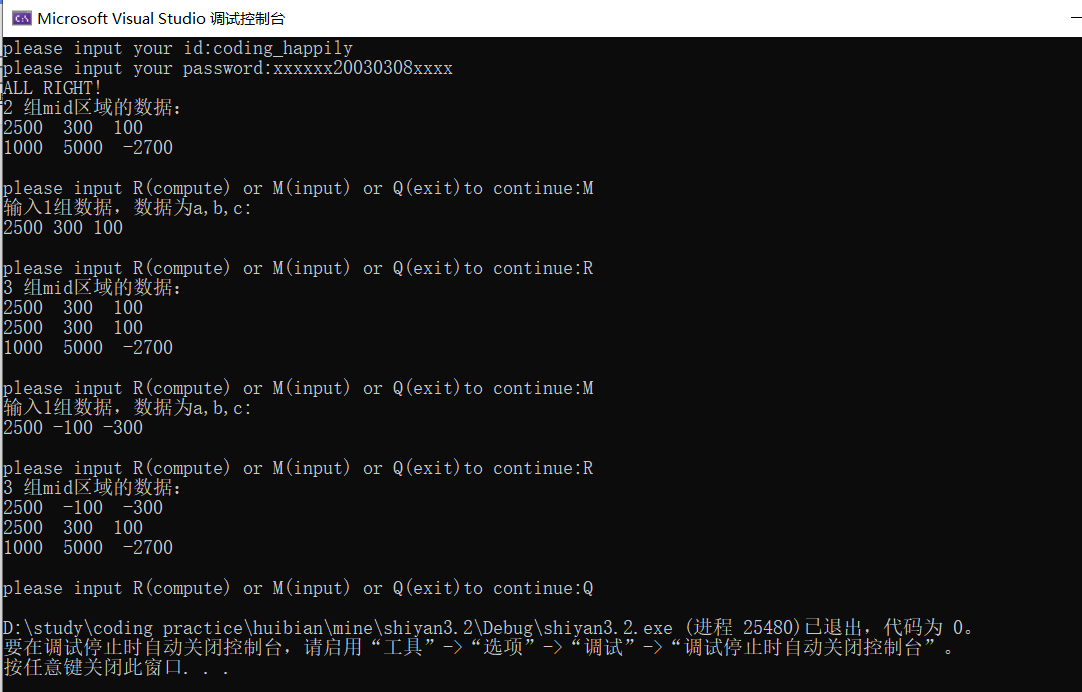


图1.10 主函数使用c语言实现的代码的功能验证

4. 使用调试工具观察、探究代码的情况

利用VS2019的单步执行的方法，观察调用子程序前后堆栈的变化情况等。

在整个过程中，重点观察了调用其他函数对寄存器变量的影响，认识到了需要先保存寄存器的值再调用函数然后再重新得到寄存器的值的方法。

5.其他

在该实验中，做了一些优化。第一个优化就是对计算的优化；第二个是实现了可以输入数据的功能，通过了c语言实现了该功能，具体的为assign\_again函数，该功能使用了两版代码来实现，第一个是主函数采用汇编编写的，第二个为主函数采用c语言编写的，在c语言编写的主函数中，由于assign\_again函数也是使用的c语言编写，所以主函数和该函数都自动同时解决了对齐的问题，在这个函数中，只需要传递结构体变量的地址就可以了。但是在汇编实现主函数的代码中，汇编中并不会严格要求对齐，而是对逐字节的进行保存，因此，在c语言实现的assign\_again函数中，重新读取一组SDA、SDB、SDC时，会发生冲突，因为汇编代码传递过来的地址在该函数中会自动对齐，这样的话，scanf得到三个数的时候，是从传递过来的地址后移12、16、20的地方进行赋值，而不是汇编中所对应的第10、14、18位，因此在汇编实现的主程序的这一版本的代码中，不能传递结构体变量的地址，而应该传递的是所要改变的位置的地址。

## 内容1.2的实验过程

### 实验方法说明

1. 指令优化对程序的影响

第一，优化内容：①乘法和赋值部分合并，②三数相加利用基址加变址寻址方式获取，③除法运算由idiv优化为sar，④counter计数改成ebx比较，减少语句，⑤获取地址采取lea。

第二，性能上更好了，需要的运行时间也更短了。

第三，优中不足的是偶尔需要牺牲轻微的准确性，由于其中一个优化的方法是通过寻址方式进行计算来做优化的，但是该方法具有一定的局限性，在结果为负数的时候会稍微有一点不适合。

1. 约束条件、算法与程序结构对程序的影响
2. 约束条件

该结构中的数据类型全部为DD，为无符号类型。在之后的运算中也是采用无符号数的运算。这个时候可以不用考虑溢出，而如果是有符号数的话，也就是SDWORD的话，那么这个计算就有可能发生溢出，因此在实验3中，将该运算改成有符号数以后，需要将运算更改，防止发生溢出。

1. 算法

对SF的运算可以采用直接的顺序的算术运算，也可以采用其他方式实现。

①可以采用移位代替乘法来增加程序的速度

SHL EAX,2可以替换MUL EAX,4大大提高程序运行的效率。

②可以利用寄存器间接寻址等寻址的方式来代替加减来加快速度

LEA edx,100[ecx][eax]这里采用了变址加基址的方式来替换了加的操作，这样可以使得程序编译后直接将该运算变成了结果，提高了运算效率。

③将一些参数优化保存可以节省空间和时间

比如在计数阶段，由于某些函数调用会影响寄存器的值，因此需要用一些局部或者全局变量来保存相关的值，或者在循环中使用定义的变量来控制循环。而这一部分需要用到的空间和时间都比较的多，如果熟悉调用的函数的内部程序，那么就可以将不需要的寄存器用来保存所需要的控制值。

④优化取地址部分

LEA对地址的获取会比MOV+OFFSET运行快。

1. 程序结构

程序结构对程序的影响主要体现在实验3中，实验三中采用了多种方法实现于是暖，并且采用了子程序、宏定义等方式对一个功能进行实现，可以发现，宏定义对短的程序有较好的影响，可以提高代码的阅读效率；对于长的代码，采用子程序调用进行功能实现可以更好的运行程序，但是需要注意的是参数的传递。

3. 编程环境对程序的影响

在所有的实验中，我们使用了VS2019、DOSBOX、OPENEULER三种实验编程环境进行了编程练习。

可以发现，VS2019对运行32位的汇编程序很成熟并且有很好的中断和异常处理程序；DOSBOX对16位的汇编程序很成熟并且可以很好的观察二进制代码；OPENEULER这个环境对gcc编译更加的灵活，可以将函数和主程序分开编写和编译，这样的话对调用函数非常的灵活和方便。

### 实验记录与分析

1. 优化实验的效果记录与分析

对优化前的代码和优化后的代码分别进行一亿次计算，最后输出时间，如下图所示。

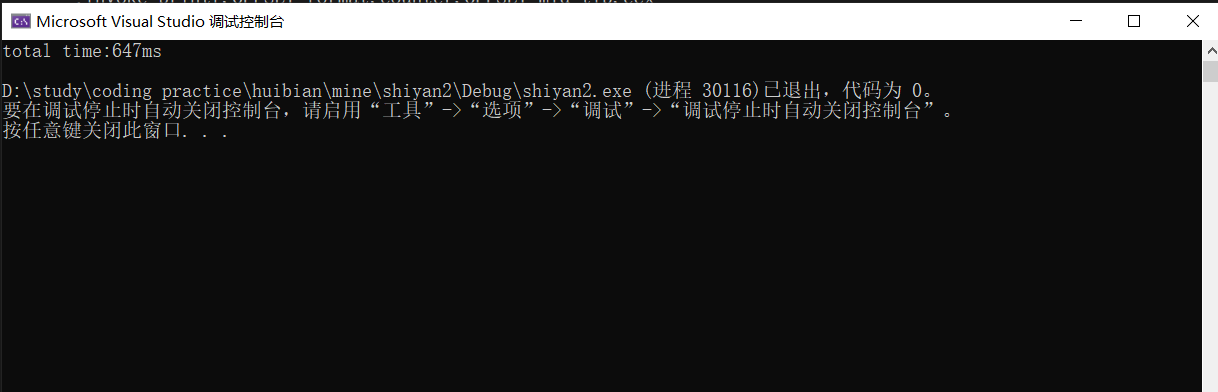


图1.11 优化前

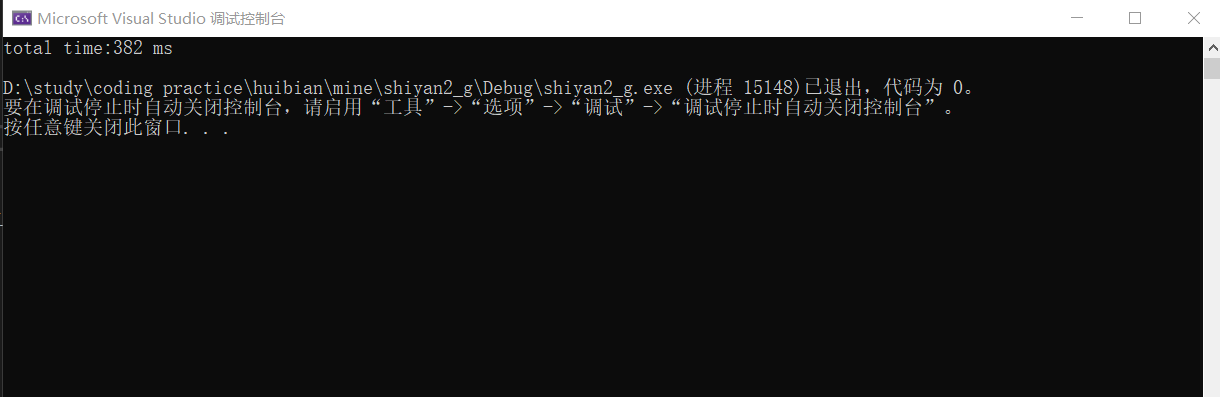


图1.12 优化后

1. 几种编程环境中程序的特点记录与分析

在OPENEULER环境中运行一个程序并计时得到如下的结果：

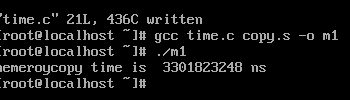


图1.13 优化前

在OPENEULER环境中运行双倍优化的程序所得到的结果：

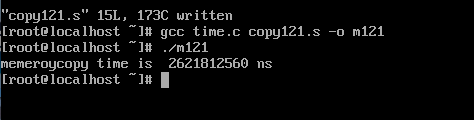


图1.14 双倍优化后

在OPENEULER环境中运行另一种方案的优化程序所得到的结果：

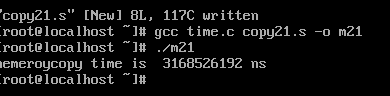


图1.15 另一种优化方案

## 小结

在本节实验中，主要是对观察不同的改变因素对程序执行的影响。其中，包括我们课上所学的子程序、宏指令等程序设计方法来优化程序结构，提高程序运行速度；而另一方面，优化程序还可以从算法设计优化、指令选择优化、程序结构优化、环境改变等各方面的经验性和技巧性更占一筹的方面进行。

而在这次的实验中，我也深刻体会到了上述的两点对程序优化的影响。在优化部分，我采用了五种优化方案对该程序进行优化，在上面的实验报告中也展示了试验记录，在一些情况下，优化率可以达到50%。

这部分在多个方面让我们探索多种因素对程序执行的影响，整体来说还是非常不错的。这一部分与课内的知识对应了起来，巩固了课内的知识，也更加认识到课内的知识如何实践出来。

# 二、利用汇编语言特点的实验

## 2.1目的与要求

掌握编写、调试汇编语言程序的基本方法与技术，能根据实验任务要求,设计出较充分利用了汇编语言优势的软件功能部件或软件系统。

## 2.2实验内容

在编写的程序中，通过加入内存操控，反跟踪，中断处理，指令优化，程序结构调整等实践内容，达到特殊的效果。

## 2.3实验过程

### 2.3.1实验方法说明

1. 中断处理程序的设计思想与实验方法

中断处理程序INT XXH是一个很好用的方案。

在4.1的中断处理程序中，我多次采用了INT 21H的功能调用，灵活的使用了INT 21H对应的AH的值对应的功能，使得程序更加高效的进行。

算法结构：

①执行代码引导输入

②输入s判断是否已经安装了中断程序，安装了的话就不再安装

③未安装开始中断显示时间

④输入q退出程序，时间驻留显示

结束后再执行其他程序的时候依旧驻留显示。

修改8号中断的代码如下：

MOV AH,35H

INT 21H

MOV OLD\_INT,BX

MOV OLD\_INT+2,ES

MOV DX,OFFSET NEW08H

CMP DX,OLD\_INT

JE ERROR

MOV AL,NUMBER

MOV AH,25H

INT 21H

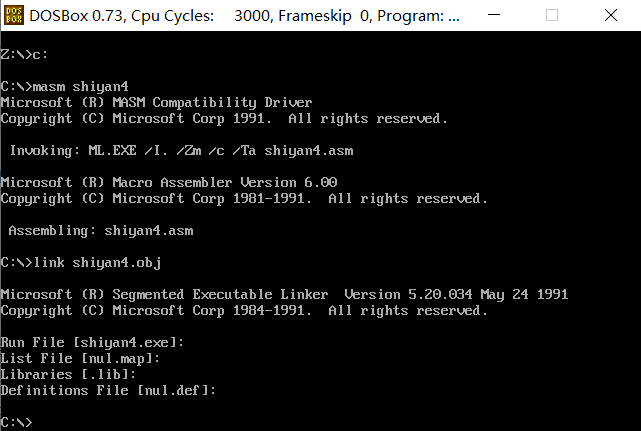


图2.1 汇编、链接、执行实验4

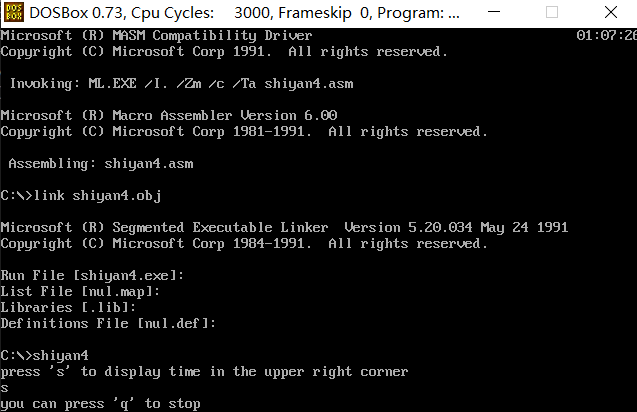


图2.2 执行实验4后显示时间

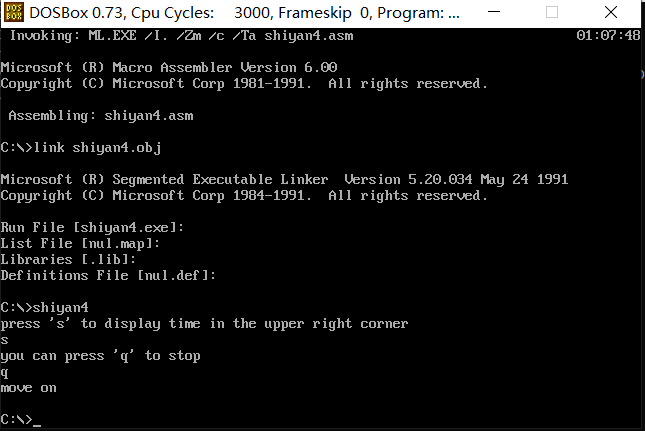


图2.3 退出程序后驻留

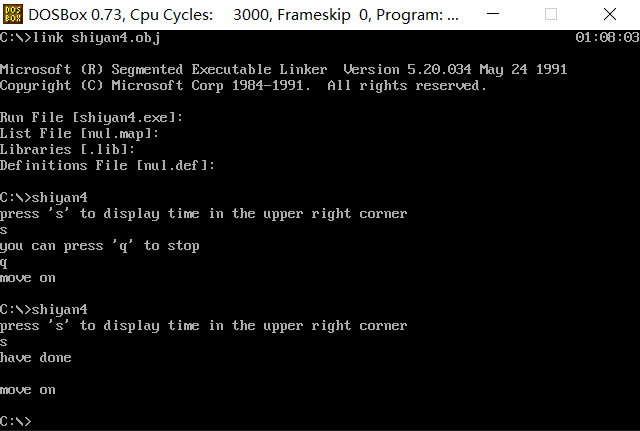


图2.4 再次安装中断程返回

1. 反跟踪程序的设计思想与实验方法
2. 加密

加密方案有对用户名和密码进行一些操作进行加密。

对用户名，采用了与一定的值进行异或的操作进行，对密码，采用了更改密码长度并截取其中的一部分作为密码的操作进行处理。这样的操作会使得原本的字符串比较的宏定义做出改变，并且需要我们进行相关的隐藏。

1. 反跟踪

①动态修改代码

对程序中的重要的需要机密的运算函数，我采用了动态修改代码的方案进行加密和反汇编，这样可以使得别人在静态反汇编的时候无法正确的获取到运算代码的值，并且可以混淆视听并且可以阻止动态反汇编，因为单步调试的时候运算部分的代码会发生改变。

②填充无用的数据或者代码

在JMP和CALL等比较关键的位置，容易成为反汇编的断点，这个地方我们可以多加一些无用的JMP或者在有用的JMP之前进行一些操作进行混淆视听；我们还可以做一些无用的计数程序用来混淆视听，因为计数往往会成为跟踪的关注点。

在一些地方我们可以加一些数据然后进行跳过，在跟踪后这一部分可以混淆视听。

③保存调用或者跳转的地点，然后采用寄存器间接寻址的方式进行调用

由于寄存器的内容只有在程序执行之后才能确定具体的值，当程序要转移到某个标号或子程序处执行时，可以先将地址表中保存的地址值送到某个寄存器中，再用JMP或CALL去转移，这样不易找到跳转的地址和目的操作。

④计时抵制动态跟踪

人工单步调试时的指令执行间隔很容易达到秒级，而连续执行的程序中的指令执行间隔一般不会达到毫秒级，因此，通过计算指定的几条指令执行所花费的时间，就可以发现该程序是否被调试。在某些关键可以获取信息容易被设置成断点的地方进行计时可以有效的察觉跟踪。

### 2.3.2实验记录与分析

1. 中断处理程序的特别之处

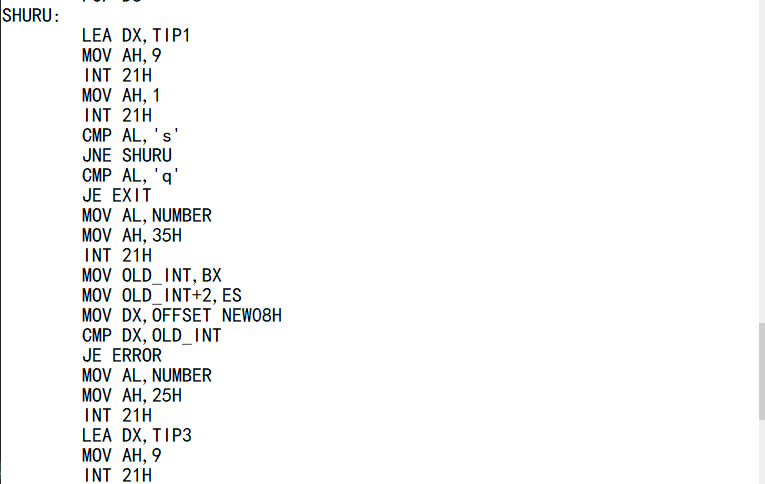


图2.5 更改新的中断程序

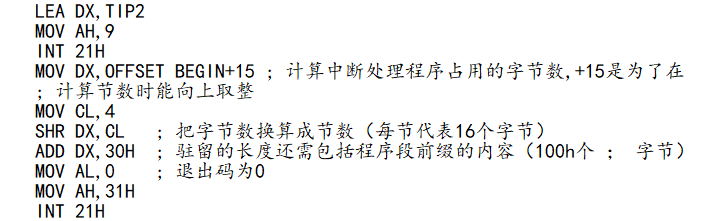


图2.6 处理驻留

1. 反跟踪效果的验证

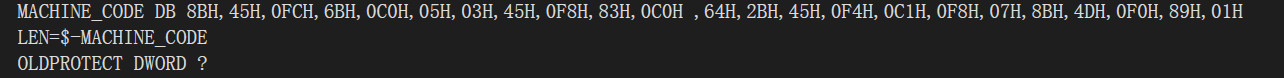


图2.7 计算的机器码

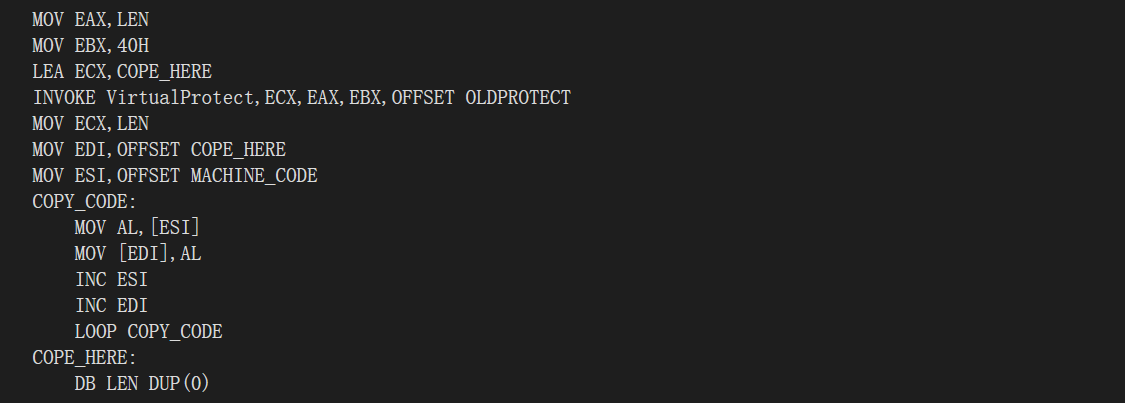


图2.8 动态修改执行代码

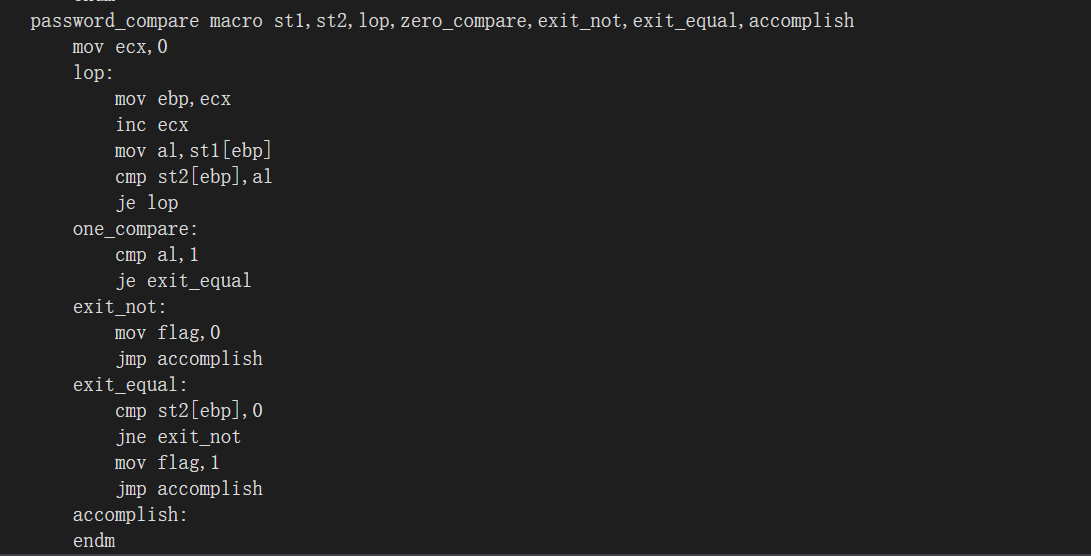


图2.9 密码加密

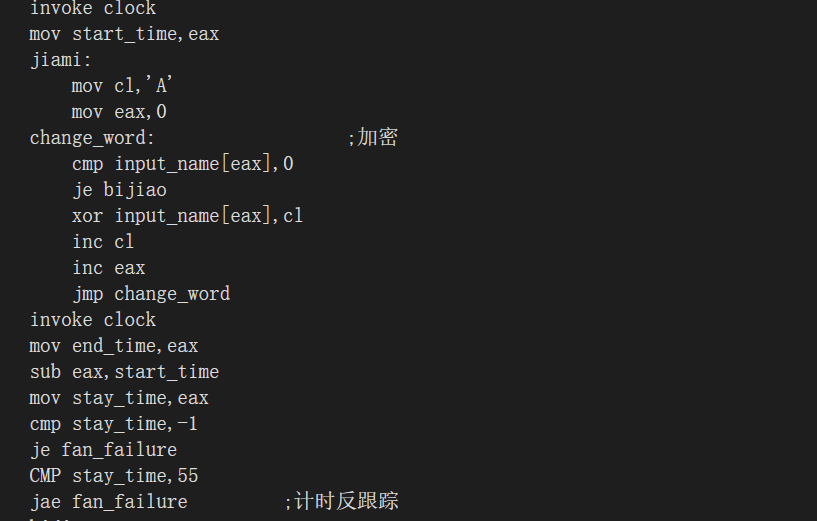


图2.10 计时反跟踪

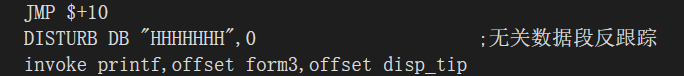


图2.11 无关数据干扰



图2.12 无关程序段干扰

1. 跟踪与破解程序

这部分实验是与同学组队破解她人的exe文件。采用了如下的跟踪破解的方式：

1. 利用静态反汇编工具将执行程序反汇编成源程序，观察源程序的特点。

利用IDA反汇编exe文件，得到汇编程序，在IDA中，可以观察模块与模块之间的联系获取到信息。

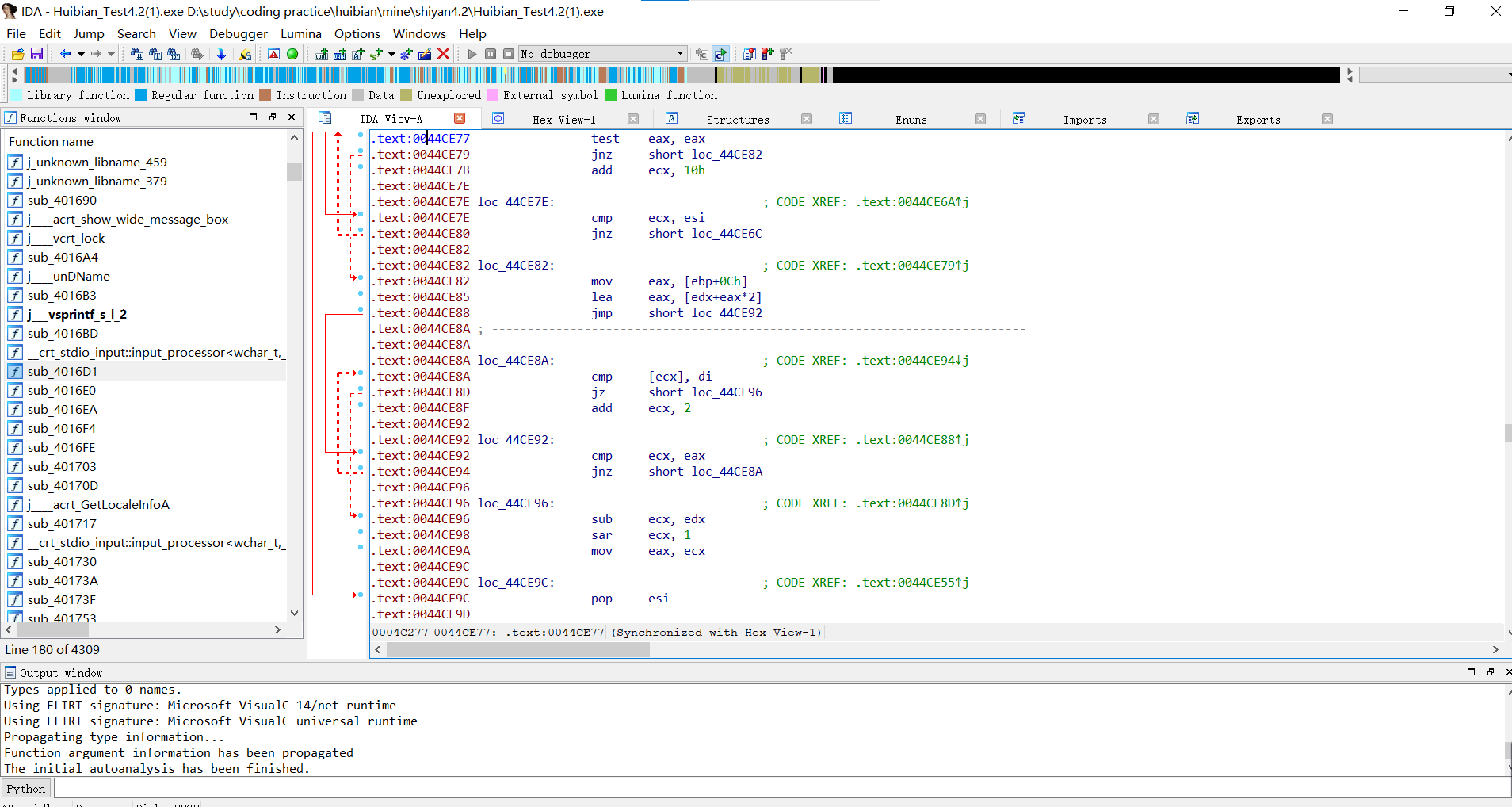


图2.13 IDA反汇编代码

1. 利用二进制文件编辑工具进行暴力破解，包括：直接观察和修改执行文件中的数据信息（比如，登录用的用户名的信息应先通过编辑工具观察执行文件来获取或修改成自己的名字）；或者直接修改执行程序的机器码，绕过输入密码后进行密码正确性检查的代码。

观察对应的存储信息那一部分的代码，多次尝试输入来破解。

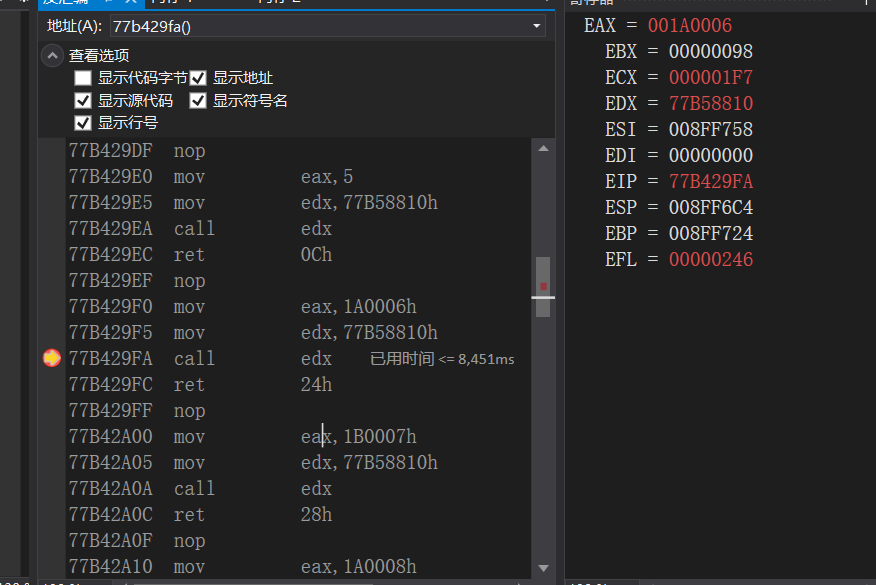


图2.14 找信息存储区域

3）动态跟踪调试，注意观察和跳过反跟踪的代码。请综合利用静态反汇编和动态反汇编的信息破解程序。

这一部分需要单步调试程序，并识别代码中反跟踪的部分以此跳过该部分防止自己被识别。

在这个过程中，采取了以上的反汇编和跟踪调试的方式后，在30分钟内并没有破解出密码，但是在与合作伙伴讨论了以后再次尝试破解成功了。

## 2.4小结

第二部分的主要实验内容为中断处理程序的编写的使用和密码加密与跟踪和反跟踪。

中断处理程序结合了时钟显示的程序进行了中断处理，这里需要我们编写新的中断处理程序，这个程序中要求我们执行原中断服务程序的功能，用IO指令从CMOS芯片中读取当前时钟的时、分钟和秒信息；用计数18次来判断是否需要执行显示时分秒的程序；调用显示程序将新的时分秒信息显示到指定位置；中断返回。这也是我们自己编写中断程序的大部分的编写方法。我更深的体会到了中断处理程序的使用。

密码加密程序主要是为了防止自己的程序被他人读取然后造成信息泄露，这一部分在前面的报告中也已经阐述了具体的实现方法，这里不赘述，在数据加密这一环节，所需要做的就是采取别人想不到的方法进行编写，无规则恰恰是更好的加密方式。

跟踪与反跟踪，这一部分很有意思，体现了我们自己保护自己程序的方法和体会别人保护程序的方法，在这一部分，大家都扮演了攻防两方的角色，更加深刻的体会到了保护程序和寻找密码的乐趣。很有意思。

# 三、工具环境的体验

## 3.1目的与要求

熟悉支持汇编语言开发、调试以及软件反汇编的主流工具的功能、特点与局限性及使用方法。

## 3.2实验过程

### 3.2.1 WINDOWS10下VS2019等工具包

VS2019中可以观察反汇编代码，帮助我们更清楚的理解汇编的底层实现，同时我们可以观察寄存器的变化。

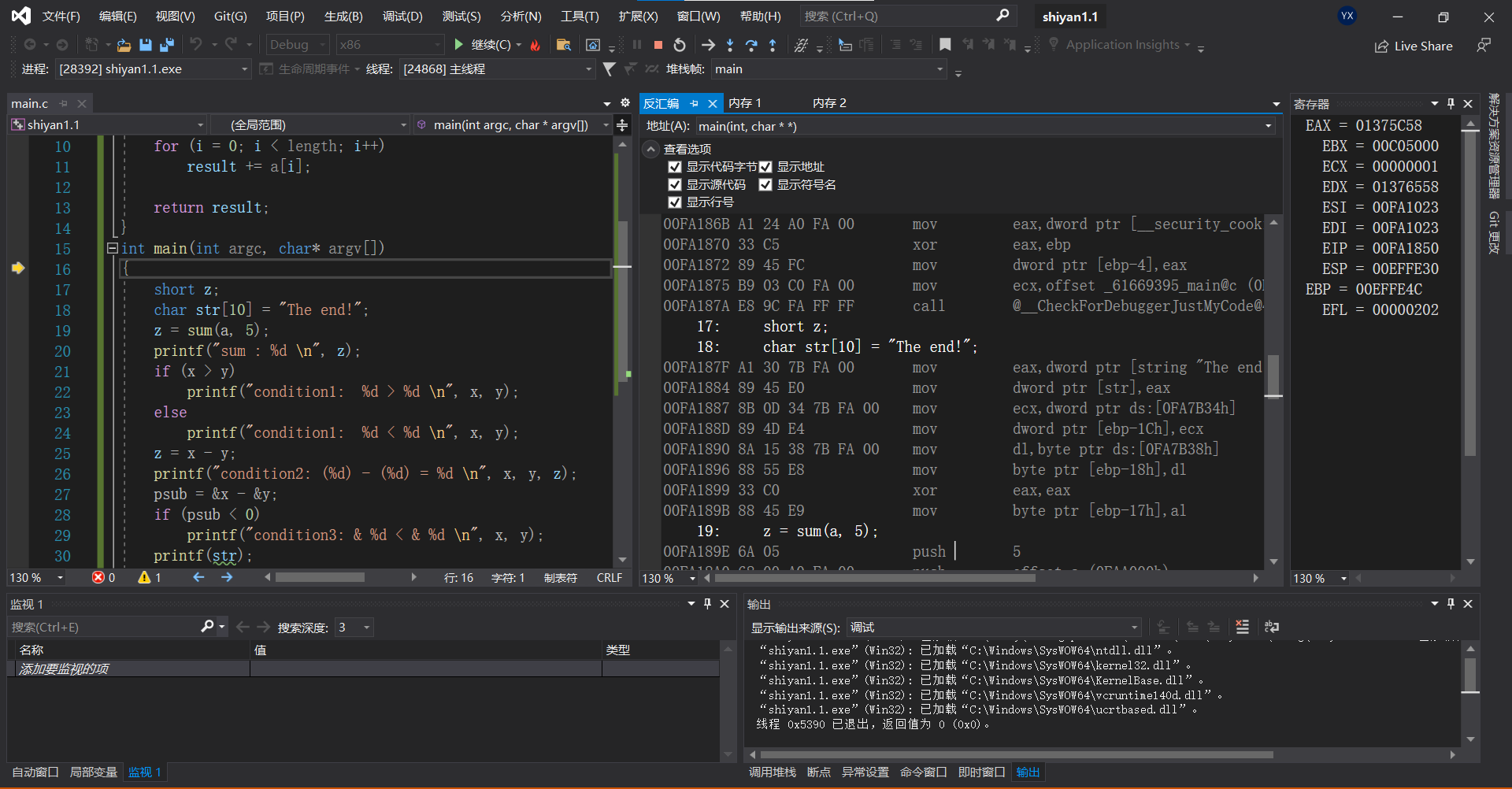


图3.1 观察代码的反汇编和寄存器变量的值

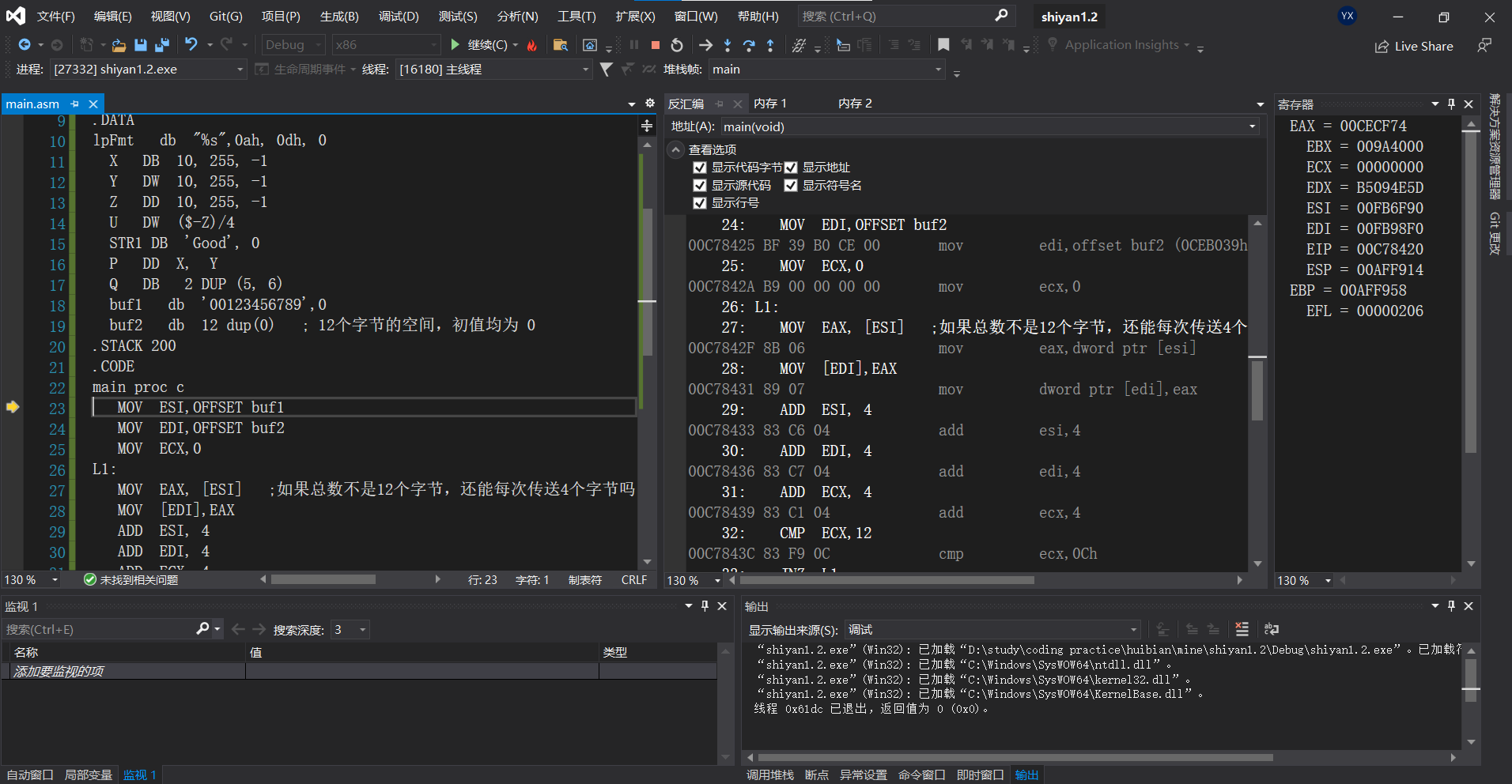


图3.2 观察汇编与反汇编的代码

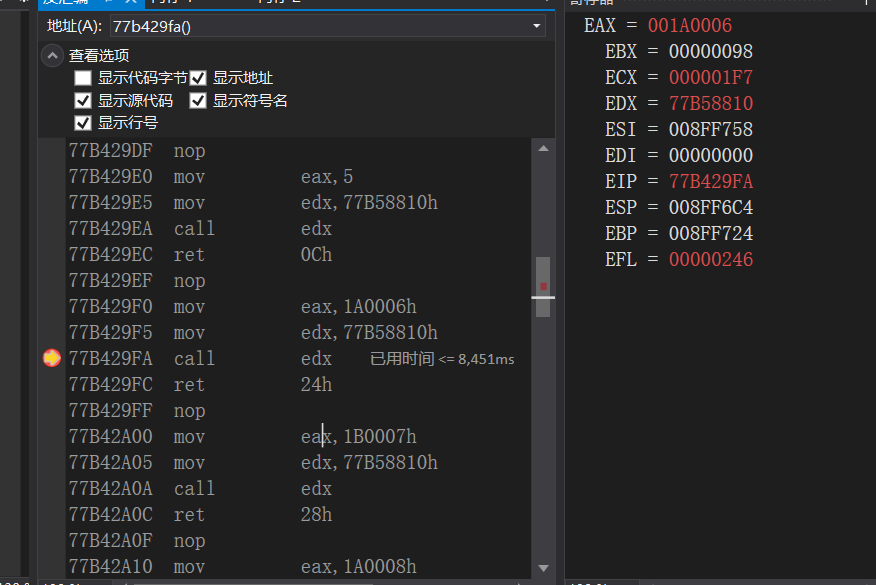


图3.3 采用VS反跟踪

### 3.2.2 DOSBOX下的工具包

在dosbox环境下，我们可以观察中断矢量表，只需要使用td命令可以将该程序对应的代码段、数据段、栈、寄存器都显示出来，分别在左上角、左下角、右下角、右上角。

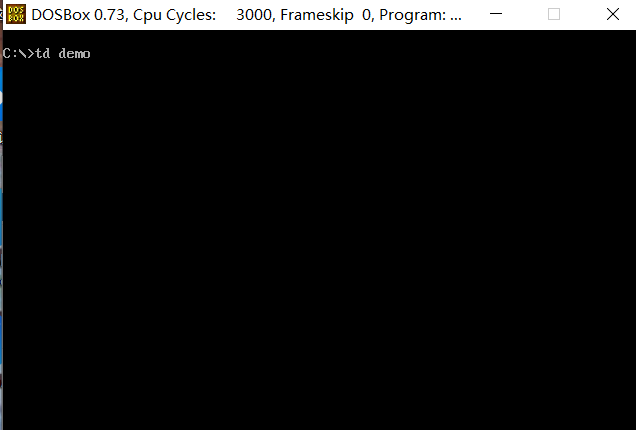


图3.4 进入二进制界面

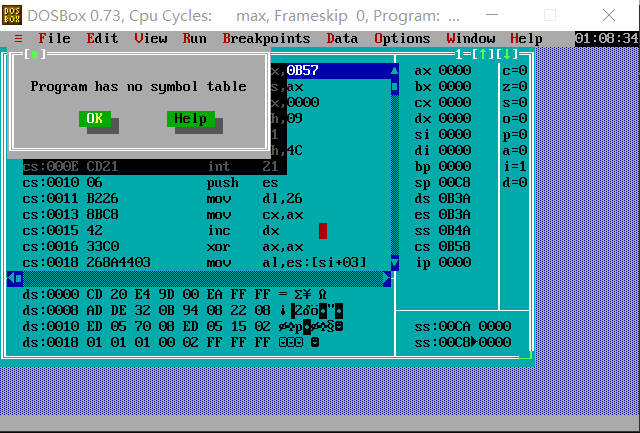


图3.5 显示中断矢量表

### 3.2.3 QEMU下ARMv8的工具包

1.QEMU下对一个显示Hello World!的汇编语言程序进行安装和编译运行：

①采用vi指令对源代码文件进行编辑

②as进行编译

③ld进行链接

④执行

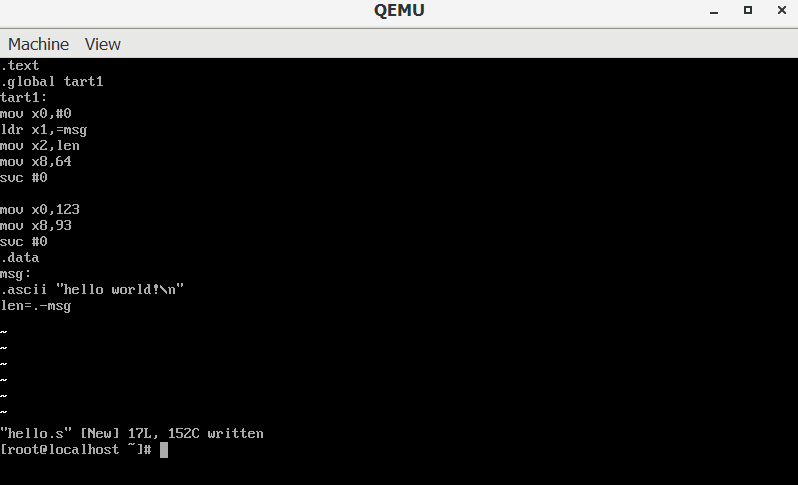


图3.6 编写文件

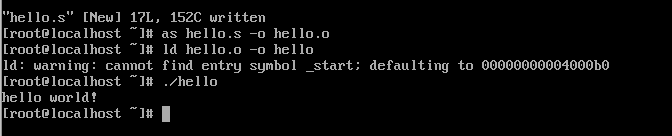


图3.7 编译链接执行

2.对一个显示时间的代码进行编译和优化

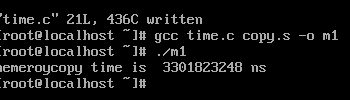


图3.8 gcc编译文件并执行

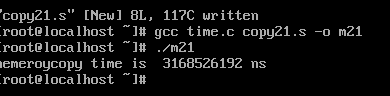


图3.9 优化后

## 3.3小结

这部分的实验主要是对汇编的编程环境的体验。

VS2019是我们最常使用的IDE，在C\C++等高级语言所体现出的功能比较强大，汇编中虽然没有高亮等提示，但还是在调试和异常处理等方面有很优秀的功能，并且可以很清楚的进行反汇编操作和代码展现，优点就是易操作和清楚。

DOSBOX是在16位的汇编程序中很好用的虚拟环境，在该环境中，可以对中断和编译链接的步骤更清楚。在该环境下实现中断处理程序的体会让我很深刻的体会到了中断处理程序的编写和使用方法。

QEMU是64位下的处理程序，在该环境中，运行一个文件可以有多种方法。虽然在编写源文件的代码的过程中相较于其他的IDE或者环境显得更加的复杂，因为它只能在中断对程序进行编写，并且没有自动配置和调试检错的功能；但是在多文件程序中，它的gcc编译功能比较强大好用，可以将几个文件进行不同的组合然后编译到一个新的文件中去，这样可以使得一个代码多用，增加了利用率和空间使用率。编译过程和结果也比较舒适。

# 参考文献

[1]许向阳. x86汇编语言程序设计. 武汉：华中科技大学出版社，2020

[2]许向阳. 80X86汇编语言程序设计上机指南. 武汉：华中科技大学出版社， 2007

[3]王元珍，曹忠升，韩宗芬. 80X86 汇编语言程序设计. 武汉：华中科技大学出版社，2005

[4]汇编语言课程组. 《汇编语言程序设计实践》任务书与指南，2022