

**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202003班**

**学 号： U202015375**

**姓 名： 汪宇飞**

**指导教师： 曹忠升**

**实验时段： 2022年3月7日~4月29日**

**实验地点： 东九A314**

**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：

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](#_Toc97397791)

[1.1 目的与要求 2](#_Toc97397792)

[1.2 实验内容 2](#_Toc97397793)

[1.3 内容1.1的实验过程 2](#_Toc97397794)

[1.3.1 设计思想 2](#_Toc97397795)

[1.3.2 流程图 3](#_Toc97397796)

[1.3.3 源程序 4](#_Toc97397797)

[1.3.4 实验记录与分析 7](#_Toc97397798)

[1.4 内容1.2的实验过程 12](#_Toc97397799)

[1.4.1 实验方法说明 12](#_Toc97397800)

[1.4.2 实验记录与分析 13](#_Toc97397801)

[1.5 小结 15](#_Toc97397802)

[二、利用汇编语言特点的实验 17](#_Toc97397803)

[2.1目的与要求 17](#_Toc97397804)

[2.2实验内容 17](#_Toc97397805)

[2.3实验过程 17](#_Toc97397806)

[2.3.1实验方法说明 17](#_Toc97397807)

[2.3.2实验记录与分析 17](#_Toc97397808)

[2.4小结 22](#_Toc97397809)

[三、工具环境的体验 24](#_Toc97397810)

[3.1目的与要求 24](#_Toc97397811)

[3.2实验过程 24](#_Toc97397812)

[3.2.1 WINDOWS10下VS2019等工具包 24](#_Toc97397813)

[3.2.2 DOSBOX下的工具包 30](#_Toc97397814)

[3.2.3 QEMU下ARMv8的工具包 32](#_Toc97397815)

[3.3小结 34](#_Toc97397816)

[参考文献 35](#_Toc97397817)

# 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

算法思想：

根据任务3.1要求，程序首先要求登录，有三次机会输入用户名和密码，若均正确则继续，然后依次计算给定的数据的值，根据所算出值的大小将其复制到HIGHF、MIDF和LOWF三个对应区域中，最后按给定格式输出MIDF区域的数据值。

模块划分与说明：

根据任务3.1要求，程序分为两个模块：ex1.asm和function.asm。其中ex1.asm包含了strcmp宏定义和main主程序，function.asm包含了computef、infoprocess和printfmidf子程序。模块之间公共符号有information:info,LOWF:info,lownum:dword,MIDF:info, midnum:dword,HIGHF:info和highnum:dword。

函数说明：

1. main函数：输入用户名和密码，比较字符串是否相等，若相等则进行后续操作。
2. computef函数：计算f=(5a+b-c+100)/128的值并返回
3. infoprocess函数：获取computef函数返回的值，判断返回值与100的大小，根据比较结构将对应的一组数据存放到对应的数据区域
4. printfmidf函数：按照给定格式打印MIDF区域存放的几组数据

模块之间的关联：

1. main函数作为主函数调用了strcmp宏定义、infoprocess函数和printfmidf函数。
2. infoprocess函数处理数据时调用了computf函数来计算数据，向computef函数传递了一个dword参数x来指示当前需要处理的数据位置，computef函数按公式计算的值f，无返回值。

寄存器分配：

1. EAX：根据程序当前的需要暂时存放变量的值。
2. EBX：根据程序当前的需要暂时存放变量的值。
3. ECX：作为计数器，存放当前正在处理的数据的首地址偏移量，由于一组数据为25个字节，故ECX的值从0开始，每处理一组数据后加25。
4. EDX：根据程序当前的需要暂时存放变量的值。

### 流程图

main主函数的流程图如图1.1所示。



**图1.1 main主函数流程图**

处理数据infoprocess函数的流程图如图1.2所示。



**图1.2 infoprocess函数流程图**

### 源程序

1. ex1.asm源程序如下：

;编写者：汪宇飞

;本模块中包含了strcmp宏定义和main主程序

.686

.model flat, stdcall

ExitProcess PROTO STDCALL :DWORD

includelib kernel32.lib

printf proto c:vararg

scanf proto c:vararg

infoprocess proto

printfmidf proto

GetTickCount proto STDCALL

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

public information,LOWF,lownum,MIDF,midnum,HIGHF,highnum

info struct

samid DB 9 DUP(0)

sda DD 0

sdb DD 0

sdc DD 0

sdf DD ?

info ends

.DATA

username DB 'WangYufei',0

password DB 'mima',0

inputstr DB 10 DUP(?)

choi DB 2 DUP(?)

lpfmt DB "%s",0

compe DB 0

sit DB 3

str1 DB '请输入用户名：',0

str2 DB '请输入密码：',0

str3 DB '输入错误，请重新输入',10,0

str4 DB '登录成功',10,10,0

str8 DB '按r键重新执行，按q键退出程序,请输入：',10,0

information info<'00000001',2540,0,0,>;=

info<'00000002',2540,128,0,>;>

info<'00000003',2540,0,128,>;>

info<'00000004',2540,0,-128,>;<

info<'00000005',2540,-128,0,>;<

info<'00000006',0,12700,0,>;=

info<'00000007',0,0,-12700,>;=

info<'00000008',0,12700,128,>;<

info<'00000009',0,12700,-128,>;>

LOWF info 10 DUP(<>)

lownum DD 0

MIDF info 10 DUP(<>)

midnum DD 0

HIGHF info 10 DUP(<>)

highnum DD 0

.STACK 200

.CODE

strcomp macro st1,st2

local comp1,comp2,comp3

mov eax,0

mov compe,0

comp1:

mov bl,st1[eax]

cmp bl,st2[eax]

jne comp3

cmp bl,0

je comp2

inc eax

jmp comp1

comp2:

mov compe,1

jmp comp3

comp3:

endm

main proc c

signin:

mov sit,3

sista:

dec sit

invoke printf,offset str1

invoke scanf,offset lpfmt,offset inputstr

strcomp inputstr,username

cmp compe,1

jne siwrong

invoke printf,offset str2

invoke scanf,offset lpfmt,offset inputstr

strcomp inputstr,password

cmp compe,1

jne siwrong

je work

siwrong:

cmp sit,0

je exit

invoke printf,offset str3

jmp sista

work:

invoke printf,offset str4

invoke infoprocess

invoke printfmidf

jmp choice

choice:

invoke printf,offset str8

invoke scanf,offset lpfmt,offset choi

cmp choi,114

je work

jmp exit

exit:

invoke ExitProcess, 0

main endp

END

1. function.asm源程序如下：

;编写者：汪宇飞

;本模块中包含了computef、infoprocess和printfmidf子程序

.686

.model flat, stdcall

ExitProcess PROTO STDCALL :DWORD

includelib kernel32.lib

printf proto c:vararg

scanf proto c:vararg

GetTickCount proto STDCALL

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

info struct

samid DB 9 DUP(0)

sda DD 0

sdb DD 0

sdc DD 0

sdf DD ?

info ends

extern information:info,LOWF:info,lownum:dword,MIDF:info,midnum:dword,HIGHF:info,highnum:dword

.data

str5 DB 'MIDF区域数据如下：',10,0

str6 DB '第%d组数据：',0

str7 DB '状态信息ID为%s，a为%d，b为%d，c为%d，f为%d',10,0

.code

computef proc x:dword

mov eax,0

mov ebx,x

mov eax,dword ptr information[ebx].sda

imul eax,5

add eax,dword ptr information[ebx].sdb

sub eax,dword ptr information[ebx].sdc

add eax,100

sar eax,7

mov information[ebx].sdf,eax

ret

computef endp

infoprocess proc

mov ecx,0

mov lownum,0

mov midnum,0

mov highnum,0

lop1:

invoke computef,ecx

cmp eax,100

je lop4

cmp eax,100

jg lop5

cmp eax,100

jl lop6

lop3:

add ecx,25

cmp ecx,200

jle lop1

jmp lop7

lop4:;=100

mov edx,midnum

mov eax,dword ptr information[ecx].samid

mov dword ptr MIDF[edx].samid,eax

mov eax,dword ptr information[ecx].samid+4

mov dword ptr MIDF[edx].samid+4,eax

mov eax,information[ecx].sda

mov MIDF[edx].sda,eax

mov eax,information[ecx].sdb

mov MIDF[edx].sdb,eax

mov eax,information[ecx].sdc

mov MIDF[edx].sdc,eax

mov eax,information[ecx].sdf

mov MIDF[edx].sdf,eax

ADD midnum,25

jmp lop3

lop5:;>100

mov edx,highnum

mov eax,dword ptr information[ecx].samid

mov dword ptr HIGHF[edx].samid,eax

mov eax,dword ptr information[ecx].samid+4

mov dword ptr HIGHF[edx].samid+4,eax

mov eax,information[ecx].sda

mov HIGHF[edx].sda,eax

mov eax,information[ecx].sdb

mov HIGHF[edx].sdb,eax

mov eax,information[ecx].sdc

mov HIGHF[edx].sdc,eax

mov eax,information[ecx].sdf

mov HIGHF[edx].sdf,eax

ADD highnum,25

jmp lop3

lop6:;<100

mov edx,lownum

mov eax,dword ptr information[ecx].samid

mov dword ptr LOWF[edx].samid,eax

mov eax,dword ptr information[ecx].samid+4

mov dword ptr LOWF[edx].samid+4,eax

mov eax,information[ecx].sda

mov LOWF[edx].sda,eax

mov eax,information[ecx].sdb

mov LOWF[edx].sdb,eax

mov eax,information[ecx].sdc

mov LOWF[edx].sdc,eax

mov eax,information[ecx].sdf

mov LOWF[edx].sdf,eax

ADD lownum,25

jmp lop3

lop7:

ret

infoprocess endp

printfmidf proc

local num:dword

mov num,0

invoke printf,offset str5

pm1:

cmp midnum,0

je pm3

jmp pm2

pm2:

sub midnum,25

mov ebx,midnum

inc num

invoke printf,offset str6,num

lea eax,MIDF[ebx]

invoke printf,offset str7,eax,MIDF[ebx].sda,MIDF[ebx].sdb,MIDF[ebx].sdc,MIDF[ebx].sdf

jmp pm1

pm3:

ret

printfmidf endp

END

### 实验记录与分析

1. 实验环境条件

AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz，内存16GB，软件为WINDOWS10下的Visual Studio 2019。

2. 汇编、链接中的情况

汇编过程中出现了2个问题：

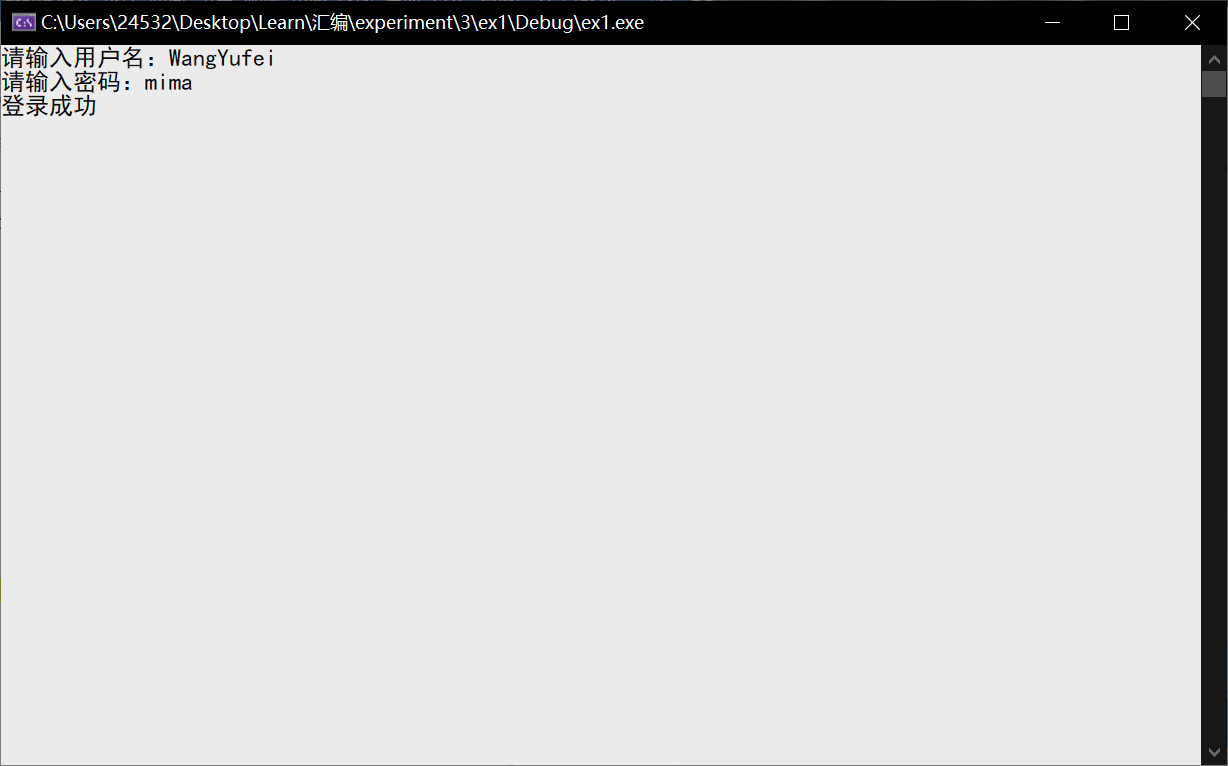
第一个问题是对于宏定义中的标签使用，由于在程序中多次使用宏定义时会让宏定义中的标签重复出现多次，因此编译出错。通过local语句解决了这一问题。

第二个问题是对于数据中的字符串输出问题，由于实验时并不太了解清楚在invoke printf函数时的输出规则，编译多次出错。通过将数据首地址放入寄存器中然后输出解决了这一问题。

3. 程序基本功能的验证情况

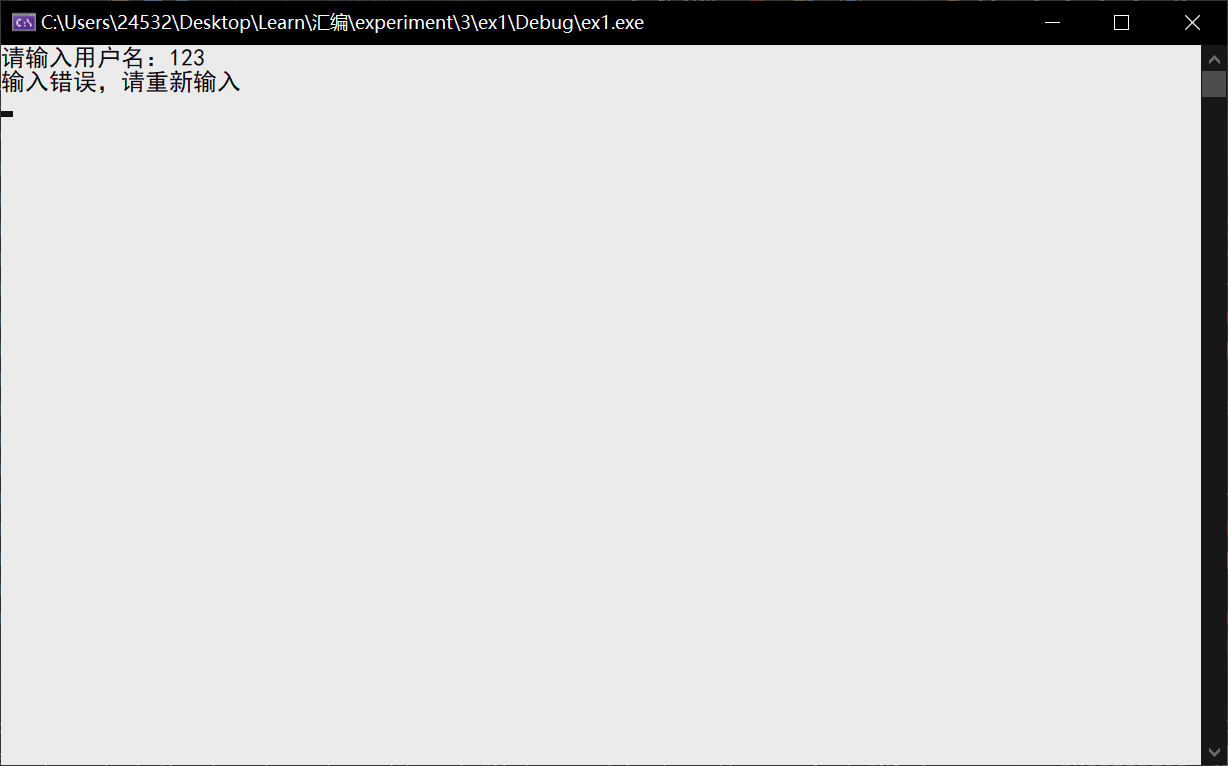
1. 登录功能验证

输入正确用户名和密码以验证登录功能，登录成功界面如图1.3所示



**图1.3 登陆成功界面**

输入错误用户名以验证登录失败功能，登录失败界面如图1.4所示

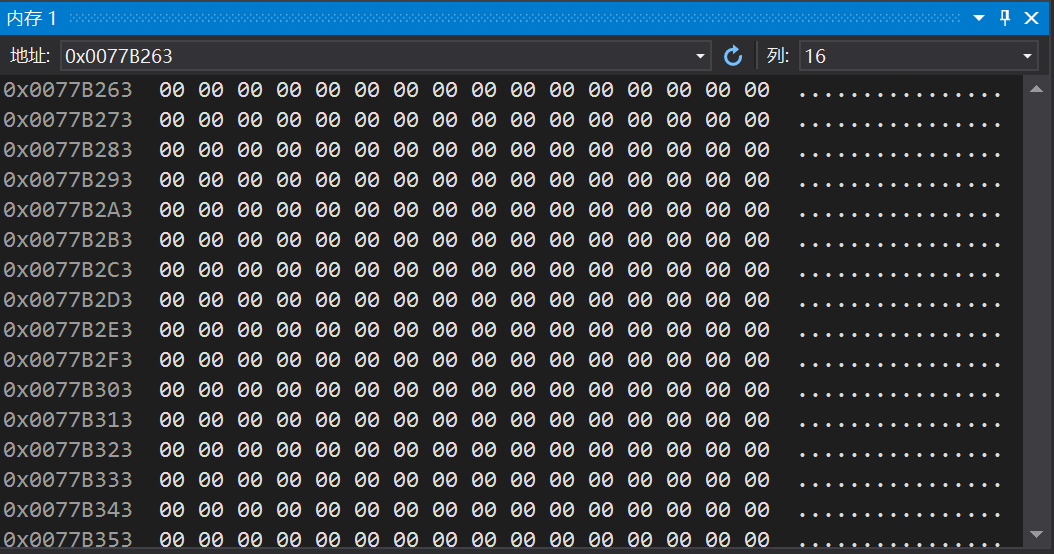


**图1.4 登录失败界面**

1. 数据处理功能验证

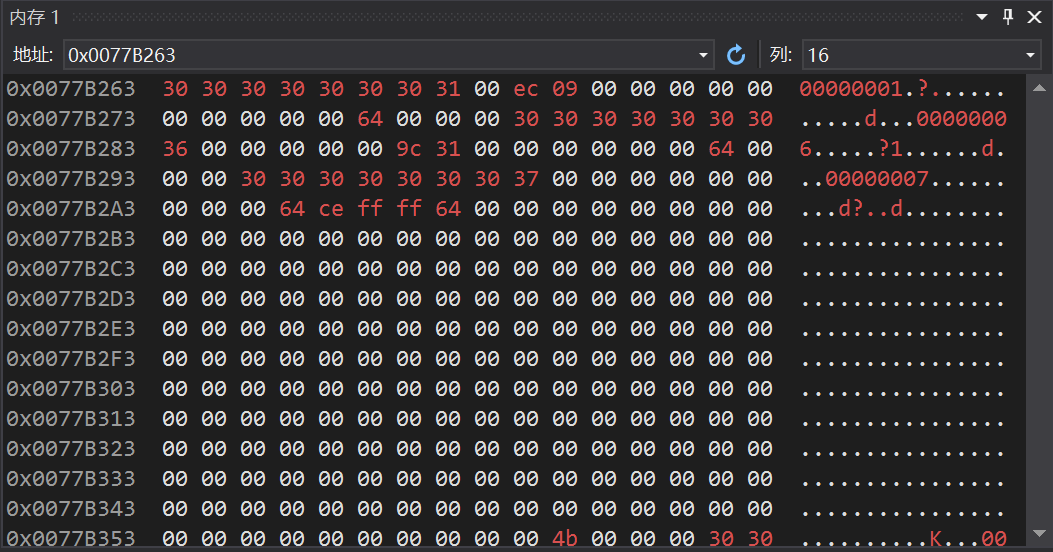
以MIDF区域为例

数据处理前MIDF区域内存如图1.5所示



**图1.5 数据处理前MIDF区域内存**

数据全部处理完后MIDF区域内存如图1.6所示

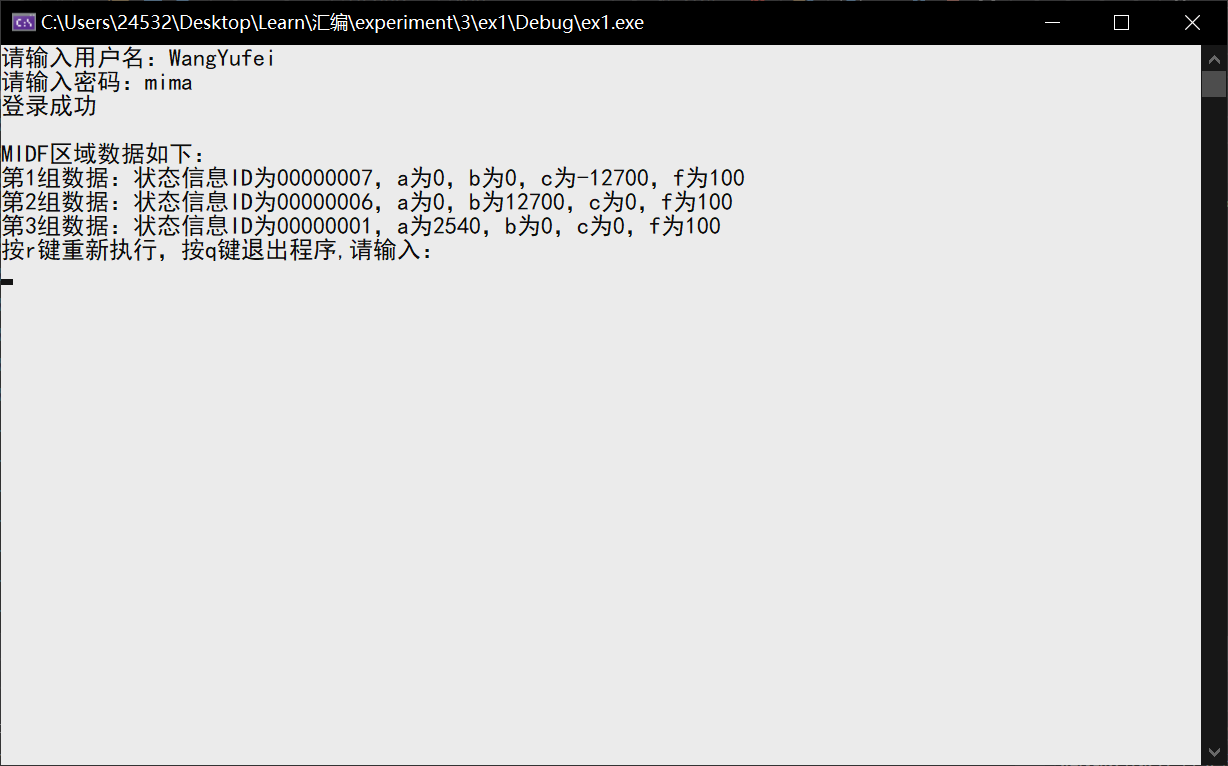


**图1.6 数据全部处理完后MIDF区域内存**

由图1.5和图1.6对比可知数据处理符合预期要求。

1. 数据输出功能验证

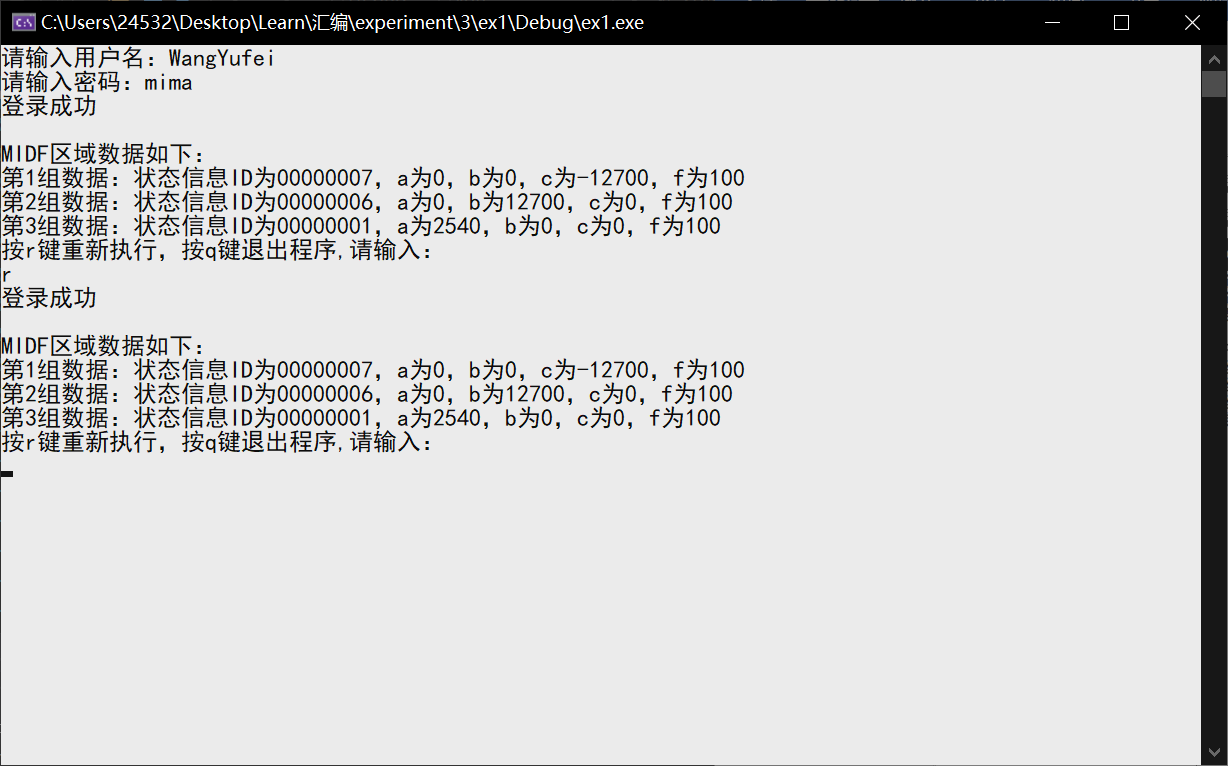
数据全部处理完成后输出MIDF区域数据界面如图1.7所示



**图1.7 数据输出界面**

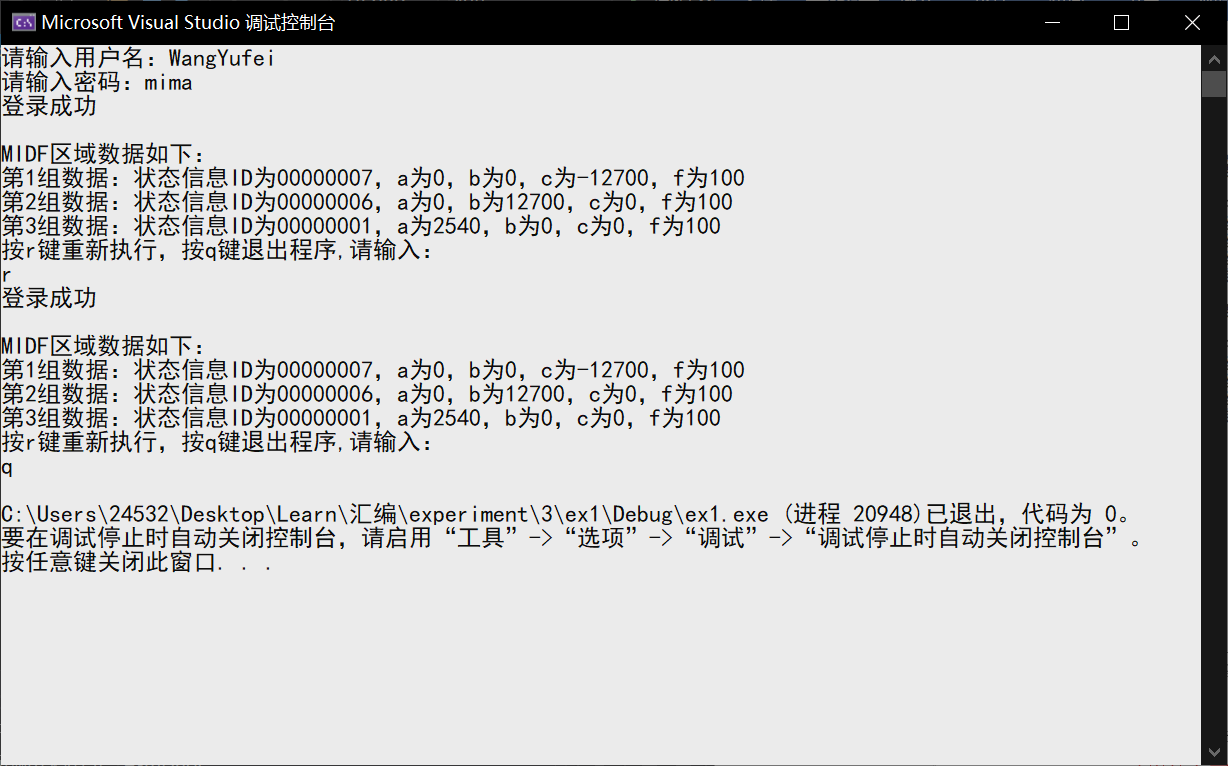
1. 操作选择功能验证

数据处理完后选择操作r时如图1.8所示



**图1.8 r操作**

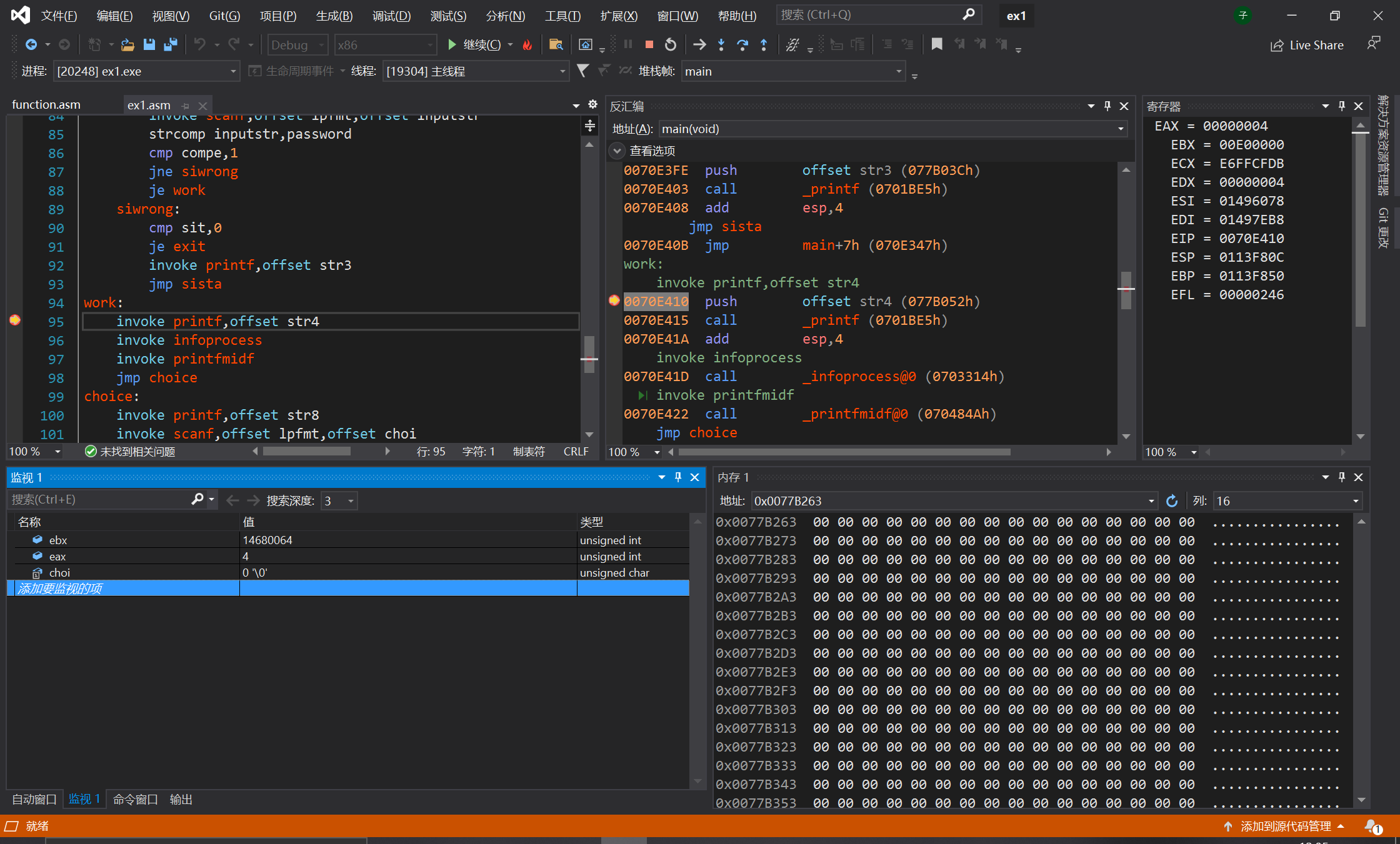
选择操作q如图1.9所示



**图1.9 选择q操作**

4. 使用调试工具观察、探究代码的情况

1. 通过使用VS2019设置断点功能，可以观察每一步的执行状态与执行结果。另外也可通过监视窗口观察变量的值、通过内存窗口观察内存的变化以及通过寄存器窗口观察寄存器的变化等。如图1.10所示。

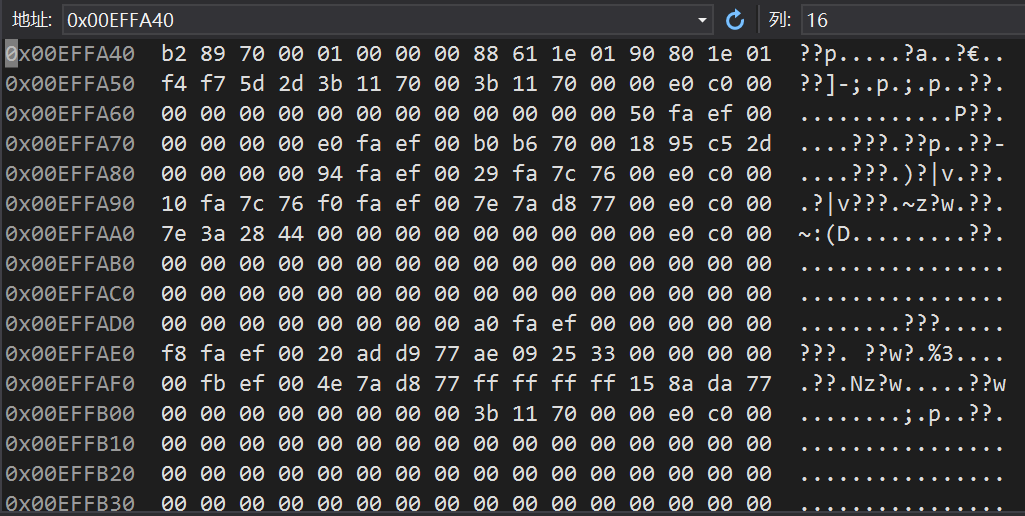


**图1.10 VS2019观察调试**

1. 子程序与主程序之间传递信息：

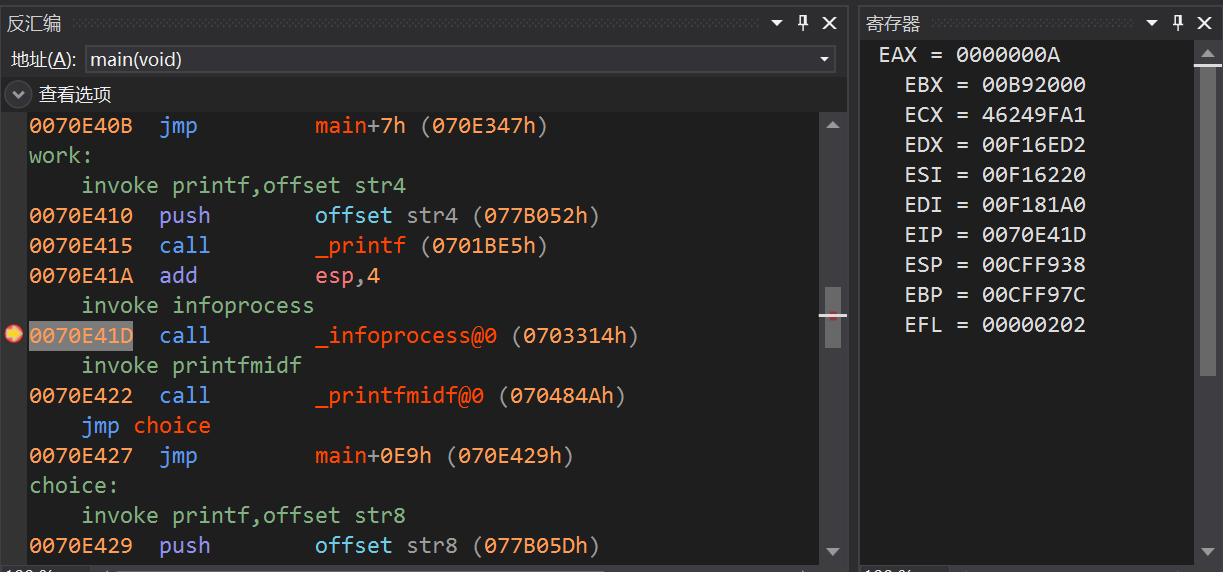
主程序向子程序传递信息的方法是通过寄存器、形参以及全局变量进行传递；子程序向主程序传递信息的方法是通过寄存器进行传递。

1. 刚进入堆栈时堆栈栈顶及以下存放的信息如图1.11所示，ESP=00EFFA40。

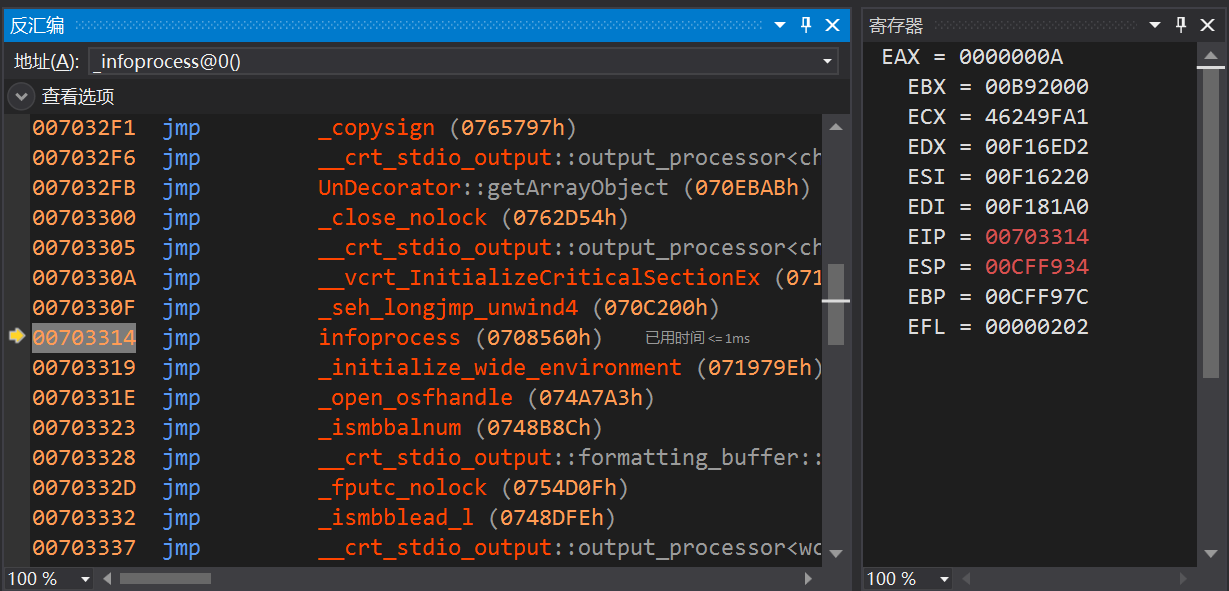


**图1.11 栈顶及以下存放的信息**

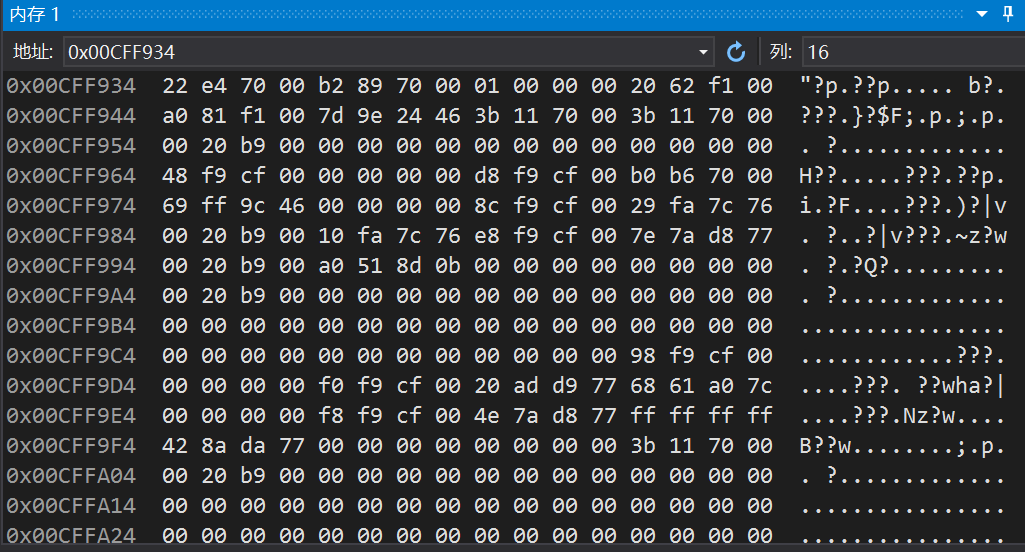
1. 执行CALL操作时，CPU完成的操作有将CALL的语句进栈并将ESP减四，如图1.12。图1.13和图1.14所示。



**图1.12 CALL操作反汇编**



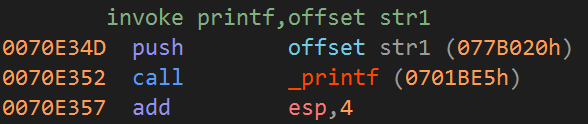
**图1.13 CALL操作**



**图1.14 CALL操作内存**

执行RET操作时，栈顶地址出栈，数据存入EIP并跳转至对应的语句。

1. 若执行RET前把栈顶数据改掉，则执行RET语句时程序依然跳转至栈顶元素地址处。
2. invoke伪指令对应的汇编语句有参数进栈和CALL语句，如图1.15所示。



**图1.15 invoke语句反汇编**

1. 子程序中的局部变量的存储空间在堆栈中。
2. 局部变量的地址可通过寻找反汇编语句中显示的地址获得。

## 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

根据任务书要求，对于任务1.4的代码进行优化并比较优化程度。分别对优化前后的程序进行计时，从而比较优化前后的性能。

对任务1.4的代码进行了两次指令优化：除法指令优化和加减指令和乘法指令。并分别对优化前、第一次优化后和第二次优化后的代码使用GetTickCount（）函数进行计时，从而比较优化效果。

2.约束条件、算法与程序结构的影响

根据任务书，对于约束条件、算法与程序结构对程序的影响进行观察。例如任务1.3中字符串比较时采用字节比较、字比较时的程序差异；任务1.4中的状态信息为无符号数时程序需要做出的调整；任务3.2中的C语言和汇编语言混合编程等。

3. 编程环境的影响

根据任务书，对于不同编程环境下的编程进行对比。对于16位、32位以及64位编程环境的异同处进行比较，对DOSBox、VS2019和QEMU观察比较。

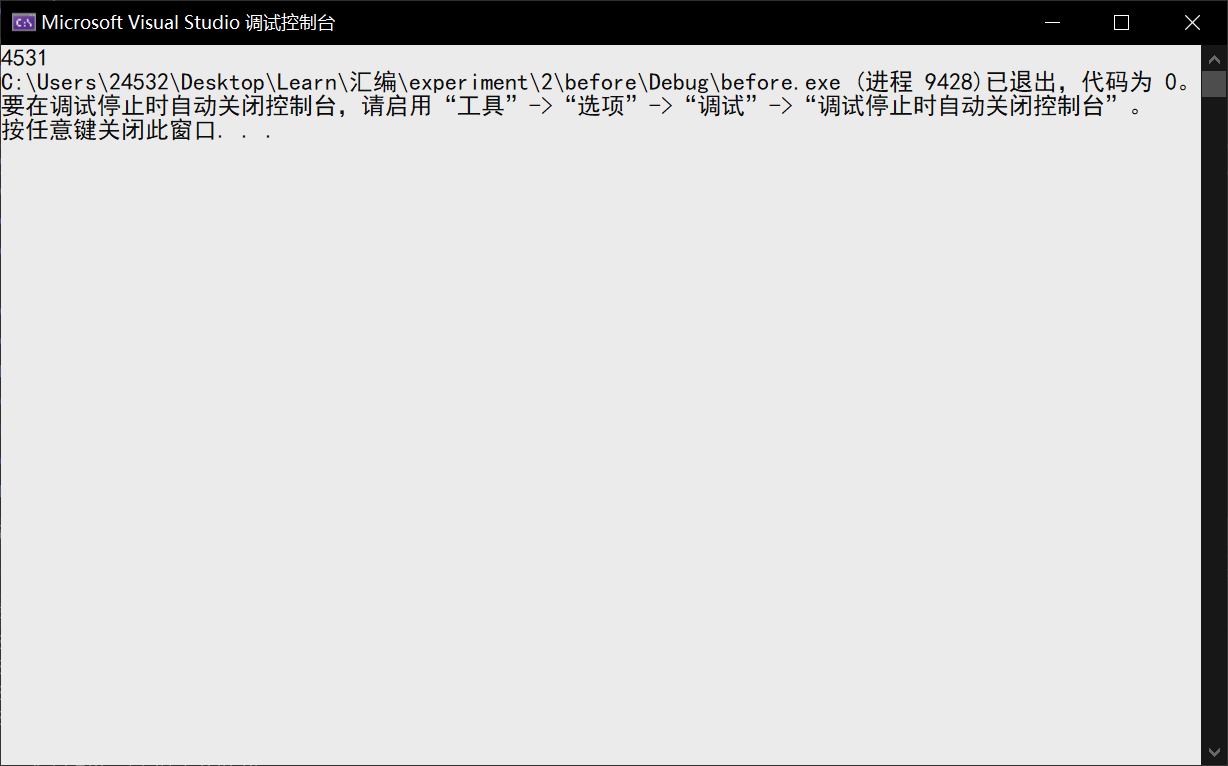
### 实验记录与分析

1.优化实验的效果记录与分析

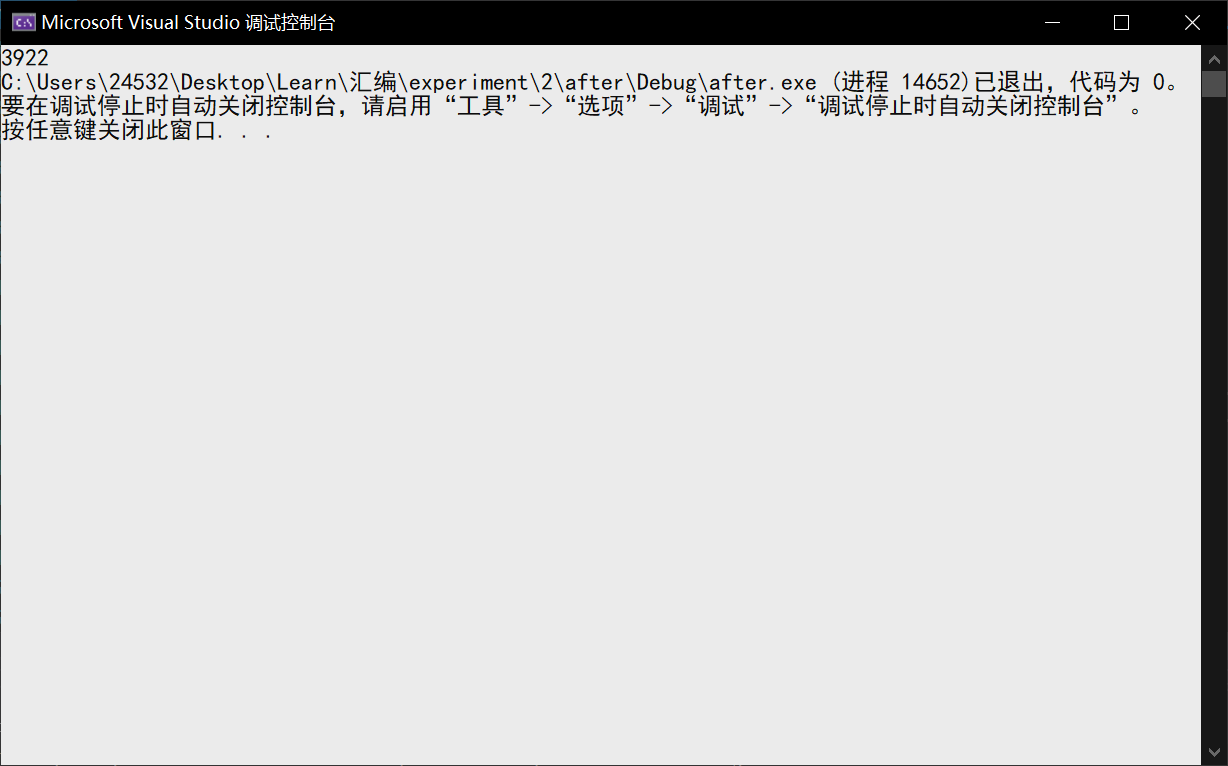
1. 优化除法指令

由于计算数据时需要除以128，而128为2的7次幂，故可以用算术右移指令代替除法指令，即使用“sar eax,7”指令，从而达到优化的效果。优化前和第一次优化后的时间分别如图1.16和图1.7所示。

可见优化程度为(4531-3922)/4531=13.44%，优化效果较明显。



**图1.16 优化前运行时间**



**图1.17 优化后运行时间**

1. 优化加减指令和乘法指令

由于计算数据时需要多次加、减、乘操作，故可以利用“LEA指令、基址+变址寻址方式”实现一些表达式的计算，从而达到优化的效果。在第一次优化后的基础上进行第二次优化的时间如图1.18所示。



**图1.18 优化后运行时间**

可见第二次优化为负优化，优化前后的代码如下：

优化前：

mov eax,0

mov eax,dword ptr information[ecx].sda

imul eax,5

add eax,dword ptr information[ecx].sdb

sub eax,dword ptr information[ecx].sdc

add eax,100

优化后：

mov eax,0

mov edx,dword ptr information[ecx].sda

lea eax,[edx\*4+edx+100]

add eax,dword ptr information[ecx].sdb

sub eax,dword ptr information[ecx].sdc

分析代码可知，优化后的操作步骤并未减少，且对于寄存器的访问次数反而增加，这可能是用lea指令后负优化的原因。

2. 不同约束条件、算法与程序结构带来的差异

1. 任务1.3的约束条件变化的影响

在字符串比较程序中，采用字节比较时可以直接一个字节一个字节地比较，而采用字比较时需要根据字符串中字符个数来进行选择，若为偶数个字符则可以逐个字比较，若为奇数个字符则需要对于多出一个字节进行额外处理。

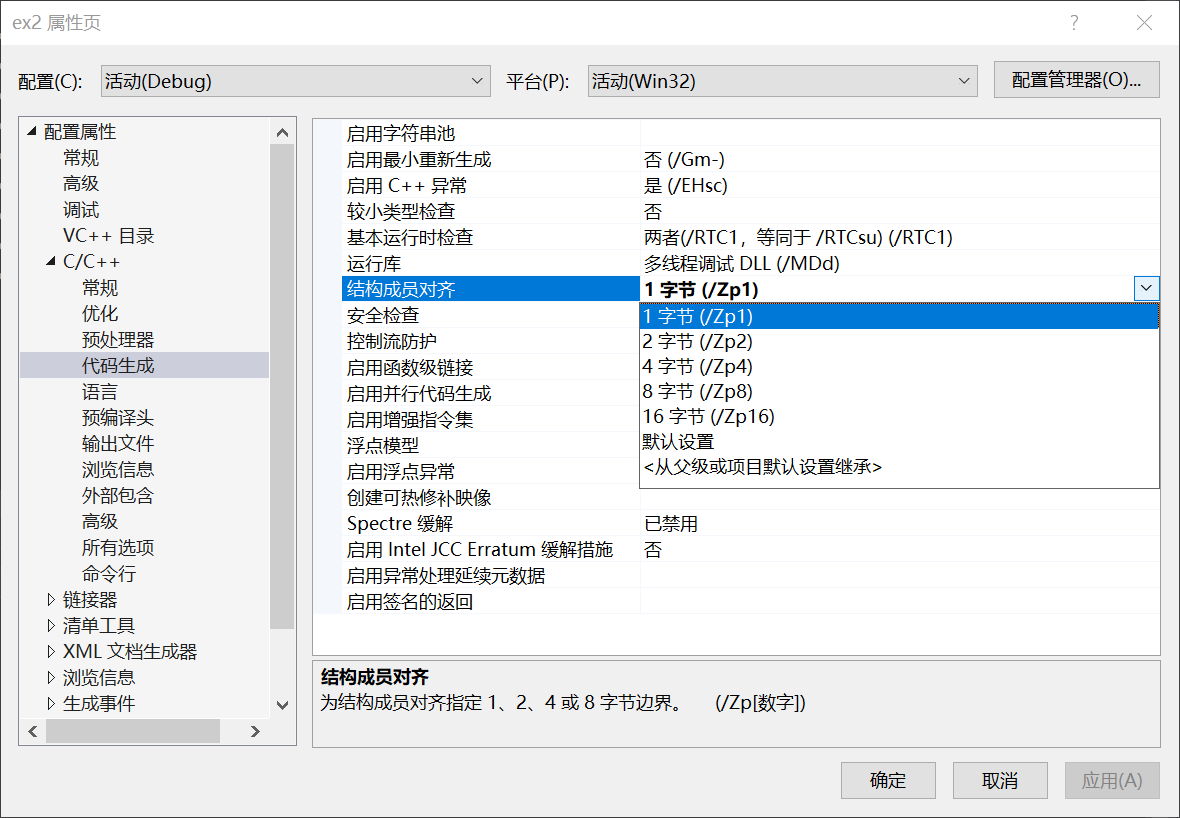
1. 任务1.4的约束条件变化的影响

在数据处理程序中，若将有符号数变为无符号数，则对应的计算语句也应改变为无符号数操作语句。

1. 任务3.2的程序结构变化的影响

使用C语言和汇编语言混合编程时，需要额外注意C语言和汇编语言之间的联系与区别。例如全局变量的声明与使用、结构体的使用等。

对于C语言结构体，其中的数据默认并未处在连续内存中，而汇编语言中的结构体数据是处在一块连续的内存中的，因此在混合编程时要进行相应的折中处理，如将C语言结构体结构成员对齐设置为1字节，即如图1.19所示。



**图1.19 结构成员对齐设置**

3.几种编程环境中程序的特点记录与分析

16位的DOSBox下：对于寄存器等的使用会更加慎重小心，16位的编程比32位会略微不便一些，但是其基本操作仍然是相似的。此外DOSBox可以突破限制，进行实方式下的编程调试，对于编程操作很有帮助。

32位的VS2019：VS2019是经典且基础的编程环境，功能齐全且强大，操作方便，32位也是我们课程学习的主体，因此用起来得心应手，能够较好地实现实验要求。

64位的VS2019和QEMU：64位的VS2019除了对于寄存器的使用的汇编语句的使用等上与32位区别较大以外，其他地方仍然十分相似。QEMU是对于ARMv8指令集的操作平台，ARMv8指令集有着与x86和x64指令不同的64位寄存器与操作语句等，差别较大，且需要使用gcc和gdb进行操作。

## 小结

这一部分的实验是汇编程序的基础编写和实现，以及不同优化措施对于程序运行和程序性能的影响。

在这一部分的实验，我熟悉并掌握了汇编程序编程的基本方法，熟悉了在VS2019中对于汇编程序的编写、调试和运行。在调试过程中我首次认识和了解了许多我从前未曾注意到的窗口及其功能。对于内存窗口，在每次运行相关语句之后内存窗口能很明显地显示该条语句所带来的改变，帮助我们检查是否达到我们的预期，给我们的编程带来很多便利。对于反汇编窗口，我们编写的每一条汇编语句都和反汇编一一对应，让我熟悉了汇编语句的具体操作与功能，对于汇编语言的认识与掌握更上一层。诸如此类的体验都让我拓宽了对于编程方法的眼界，让我受益良多。

在多次实验中，根据指导，我尝试了多种编程方法，诸如宏定义、分支循环结构、子程序、多模块和C语言汇编语言混合编程等一系列实用的编程方法。这一系列的实验让我对于相应的知识点的掌握与认识更加深刻。此外对于C语言和汇编语言之间的联系也有了更好的了解与认知。

在使用多种方式尝试优化程序的部分中，我体会到了从最基本的汇编语句对于程序优化的影响到程序约束条件、算法和结构对于程序的影响。比如一个最简单的除法指令经过优化之后，对于程序的优化效果也是十分显著的。通过这一系列的尝试与体验，我认识到在编程中的每一个细节都会影响整个程序，因此对于各种细节都需要好好斟酌，多次尝试多次比较以寻求最佳方案。除此之外也需要不断学习相应的知识，拓宽视野，对于多种编程方法进行了解尝试。

# 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

根据实验书要求，在DOSBox环境下，参考所给资料的例题，使用中断的思想并结合中断矢量表设计一个能在窗口指定位置按时分秒的格式显示当前时间的程序，并在该程序的基础上实现程序的驻留功能，并且要注意避免程序的重复安装。

2.反跟踪程序的设计思想与实验方法

在任务3.1的基础上，对程序中所设置的密码进行加密，采用多种方法改变与密码相关的程序部分，如添加计时功能以防止跟踪、采用间接寻址、在程序内加入无关的信息、密码加密等方法，实现程序加密。

设计好加密程序之后，与同组的同学交换运行程序，互相解密其程序得出对方的密码。

3.指令优化及程序结构调整的实验方法

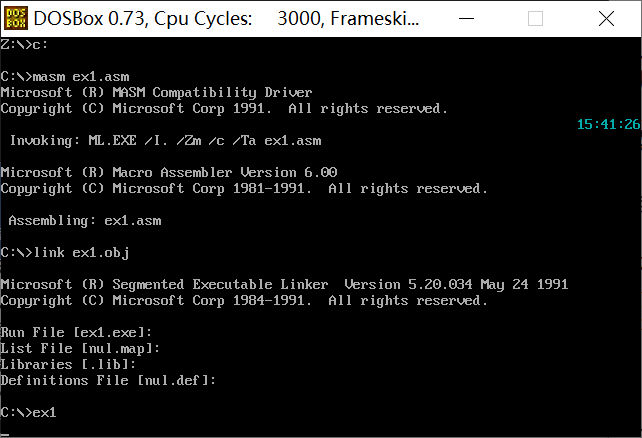
指令优化：对于一些操作改用执行速度更快的指令来提高性能、减少循环体内的指令、灵活利用机器指令的特点、寻址方式的特点等。如1.4节中对于加减乘除指令的优化。

程序结构调整：对于一些程序可以通过C语言与汇编语言混合编程的方式来调整其程序结构。如任务3.2。

### 2.3.2实验记录与分析

1.中断处理程序的特别之处

按要求所设计的显示当前时间的程序运行如图2.1所示，时间显示在右上角。



**图2.1 时间显示程序运行情况**

根据实现驻留和避免重复安装的要求，对于所给的源程序进行的修改如下：

…

NEXT: MOV AH,0

INT 16H

CMP AL,'q'

JNE NEXT

MOV DX, OFFSET BEGIN +15;计算中断处理程序占用的字节数, +15 是为了在

;计算节数时能向上取整

MOV CL, 4

SHR DX, CL;把字节数换算成节数(每节代表 16 个字节)

ADD DX, 10H;驻留的长度还需包括程序段前缀的内容(100H 个字节)

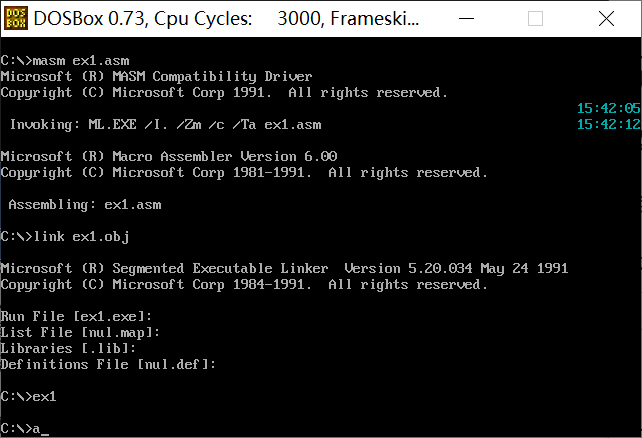
MOV AL, 0;退出码为 0

MOV AH, 31H;退出时,将(DX)节的主存单元驻留(不释放)

INT 21H

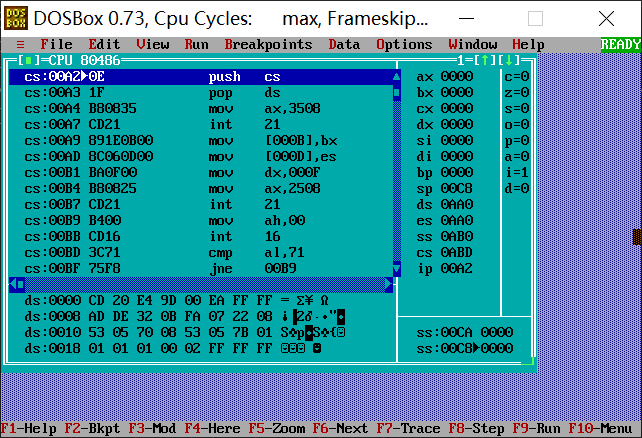
…

实现的驻留功能如图2.2所示



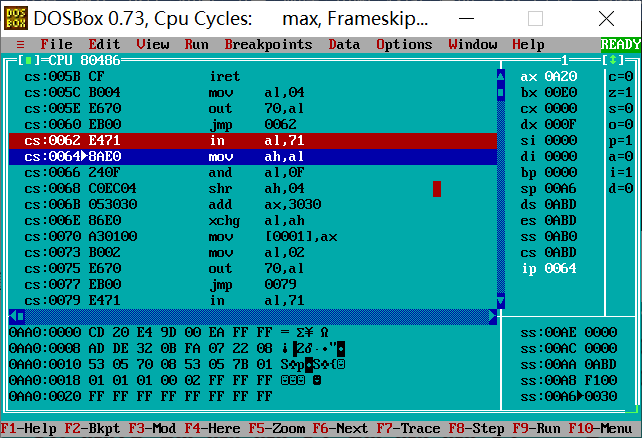
**图2.2 驻留功能的实现**

通过DOSBox进入调试界面如图2.3所示。



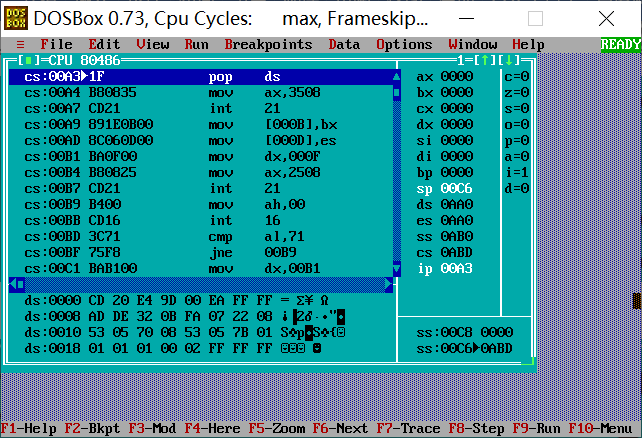
**图2.3 DOSBox调试界面**

在in al,71后设置断点并运行后如图2.4所示，al为当前时间的小时20。



**图2.4 获取小时运行情况**

INT语句处进入中断管理程序如图2.5所示

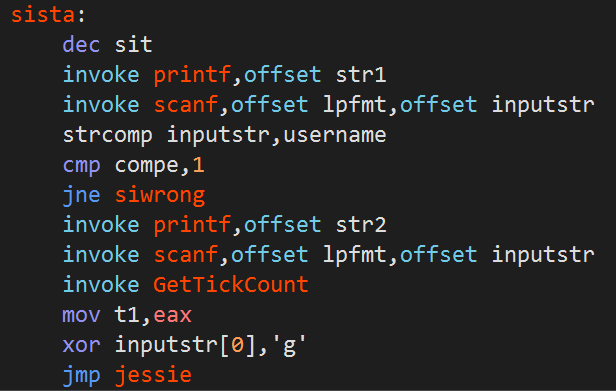


**图2.5 中断管理程序**

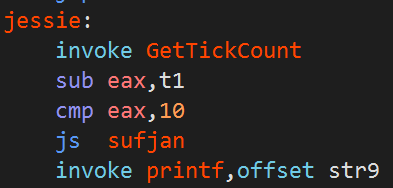
2.反跟踪效果的验证

在设计反跟踪程序中，使用了添加计时功能以防止跟踪、添加冗余干扰信息、添加多个标号打乱程序顺序以及对密码进行密码加密这一系列加密方法。

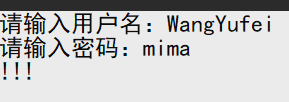
添加计时功能部分如图2.6和图2.7所示，在进行密码与输入字符串前计时，然后在比较完第一个字符后再次计时，如果时间超过10ns则输出警告信息并终止程序，如图2.8所示。



**图2.6 开始计时**

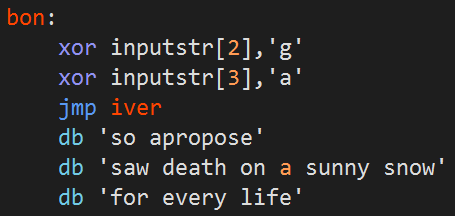


**图2.7 结束计时并判断**



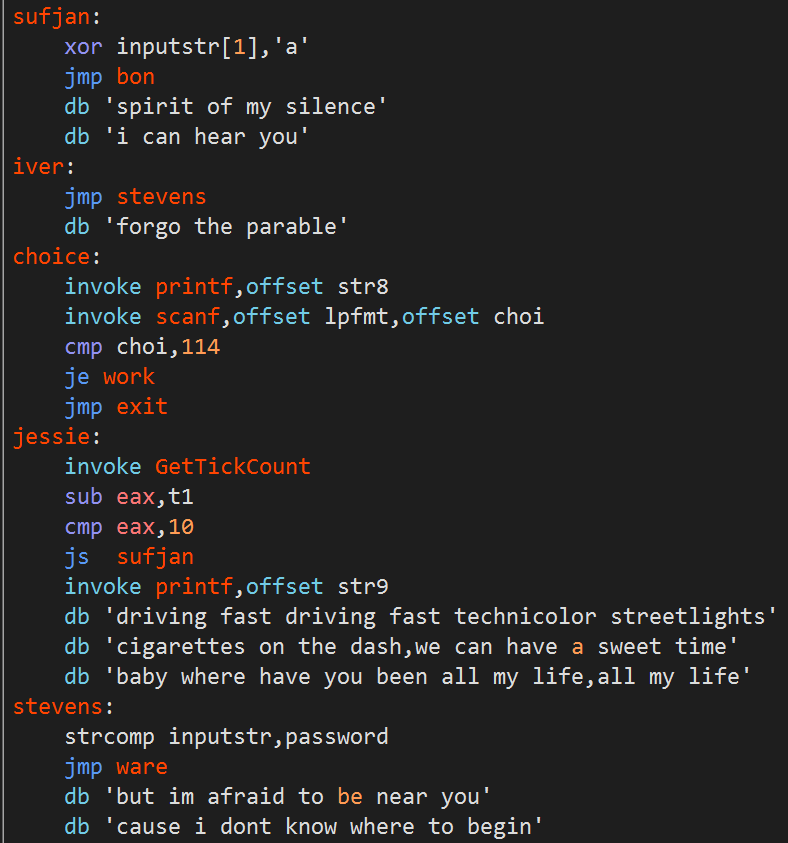
**图2.8 输出警告信息**

增加冗余干扰信息如图2.9所示，在多个地方加入无效的字符串信息，干扰跟踪过程。



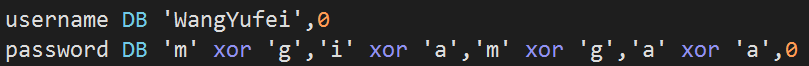
**图2.9 冗余信息**

添加多个标号如图2.10所示，将字符串比较部分通过添加多个标号进行打乱，干扰跟踪。



**图2.10 添加多个标号**

密码加密如图2.11所示，将密码逐个字母与特定字符异或。

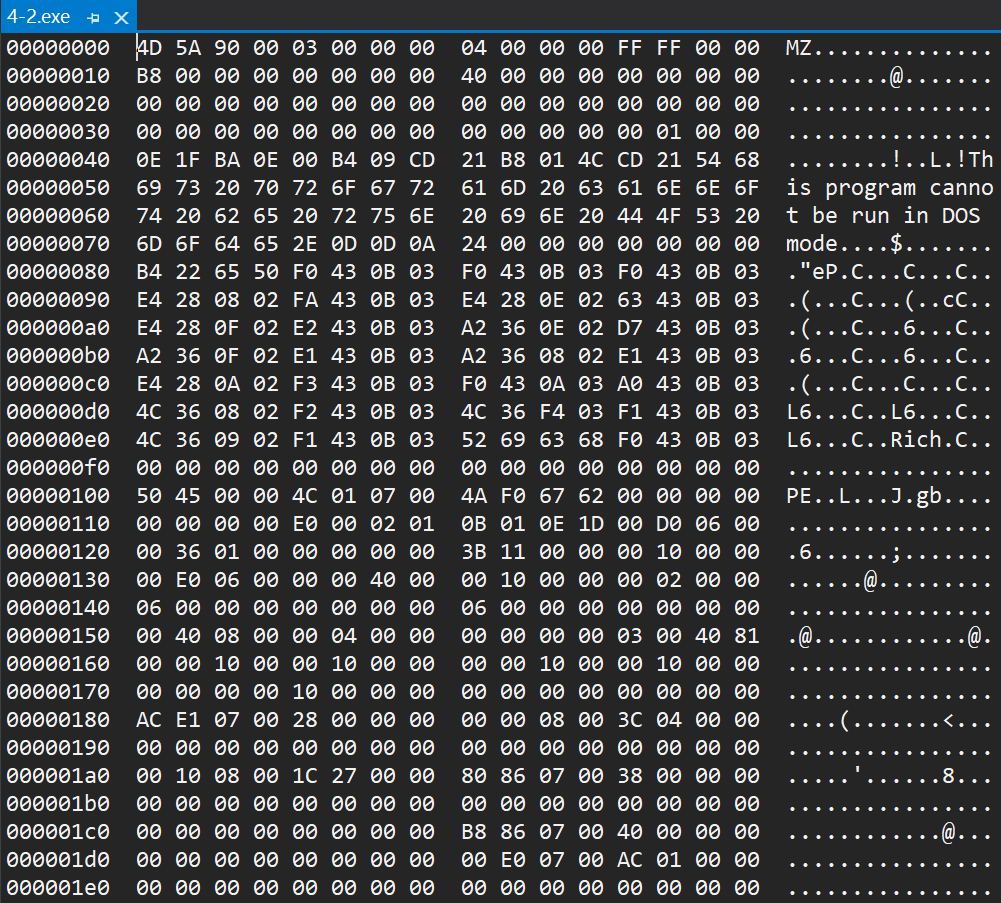


**图2.11 密码加密**

3.跟踪与破解程序

同组的同学为范辰睿。

在得到其exe文件后，首先使用VS2019打开该exe文件，使用二进制编辑器查看，如图2.12所示。



**图2.12 二进制编辑器查看exe文件**

通过VS2019打开调试界面，如图2.13所示。

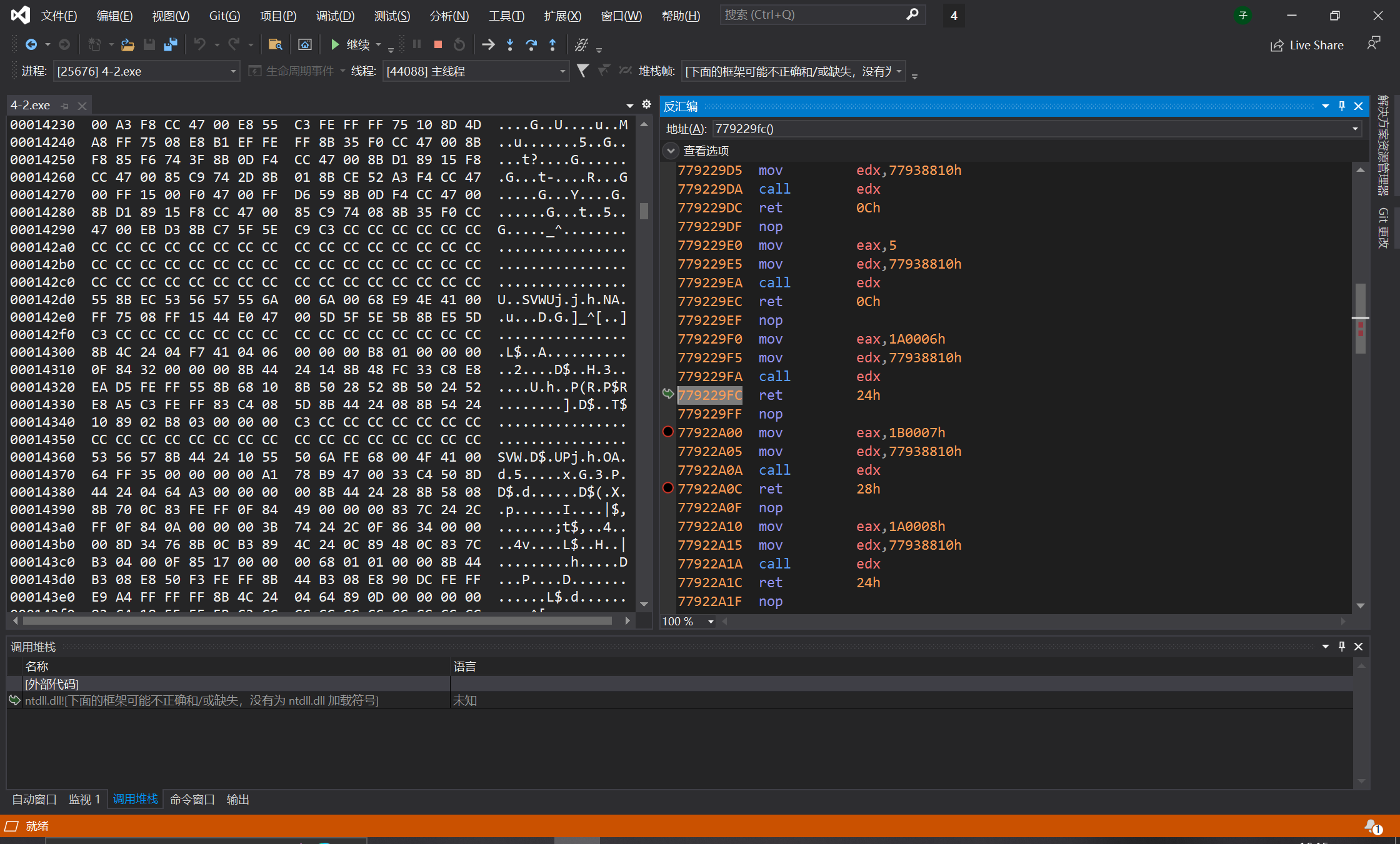


图2.13 exe的调试界面

此时的反汇编未处于最外层，因此通过设置断点的方法跳出到最外层，如图2.14所示。

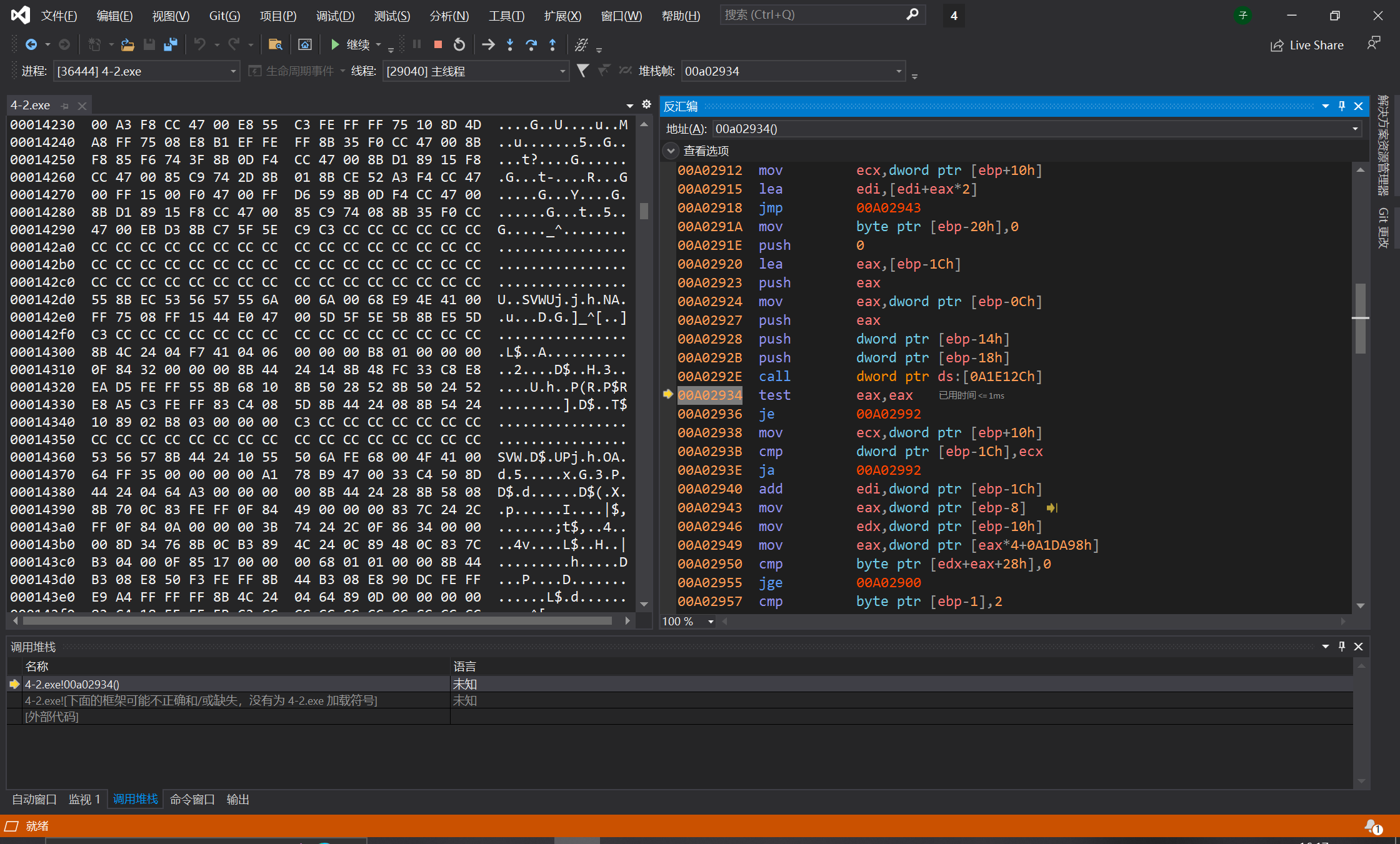


图2.14 最外层反汇编界面

由图2.15可知，当前对eax进行按位与操作，并当相等时跳转至00A02992处。

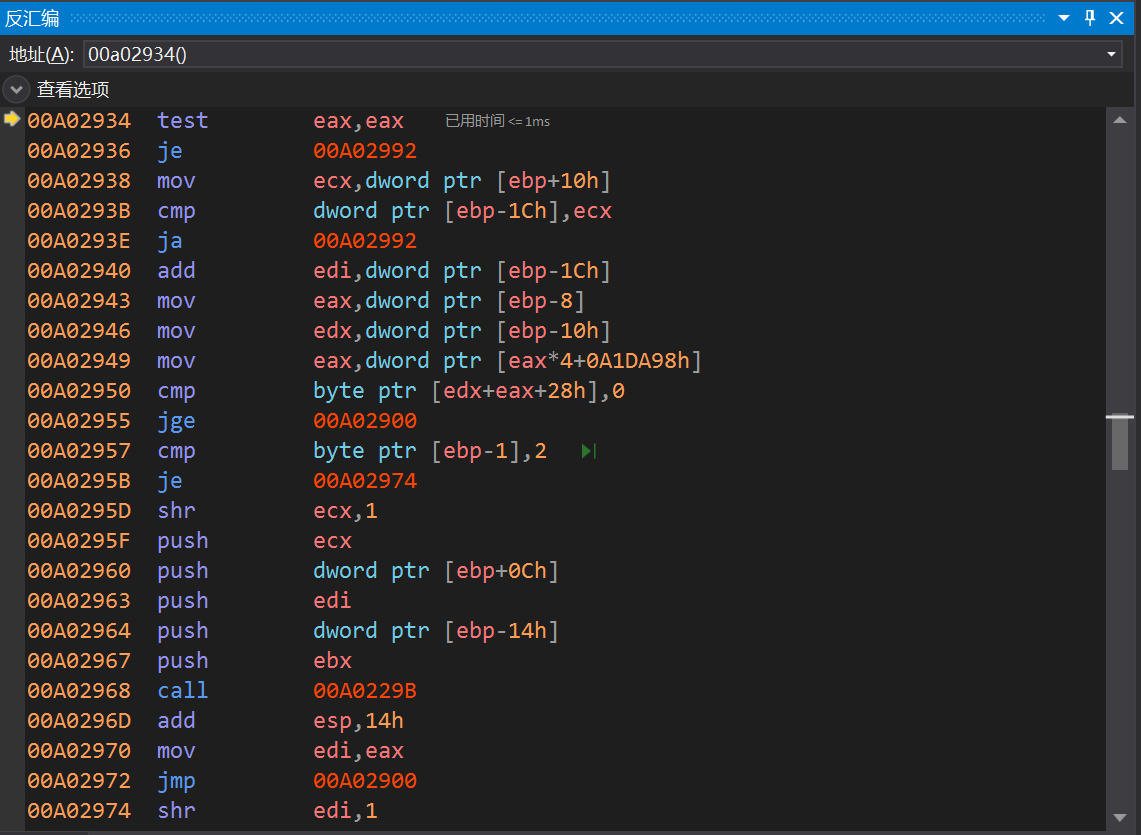


图2.15 反汇编窗口

4.特定指令及程序结构的效果

特定指令：对于一些计算指令可以进行优化，如内容1.2中对于除法中除数为128时采用右移7位的操作，极大地简化并加快了程序。

程序结构：对于一些较为复杂的程序，可以使用C语言和汇编语言混合编程，从而实现简化程序，且同时利用C语言和汇编语言各自的优点，实现取长补短的效果，如内容1.2中任务3.2的混合编程。

## 2.4小结

这一部分的实验是中断与反跟踪，是更加底层的尝试。

在中断实验我首次接触到了DOSBox的使用，首次尝试在实方式下编程和调试，对于实方式下的数据处理有了更好的了解与掌握。除此之外在对中断程序和中断矢量表的修改与处理之中，我对与中断这一重要知识点的掌握更加深入，也对中断这一高级指令的重要性有了更好的了解。

在设计反跟踪时，我了解到了对于程序加密的措施的诸多方法，诸如动态修改代码、密码进行密文处理、程序计时以及增加干扰信息等多种处理手段，极大地保护了程序的机密信息，提升了程序的安全性。

而在解密他人的程序时，我也体会到了这些方法的强大。解密他人的程序虽然有趣却又很有挑战性很有难度，通过使用多种工具来解密也让我体会到了解密工作的复杂与困难。在进行这部分实验后我对于程序加密和解密的了解更上一层，对于程序加密的重要性也有了更深的了解。

# 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。

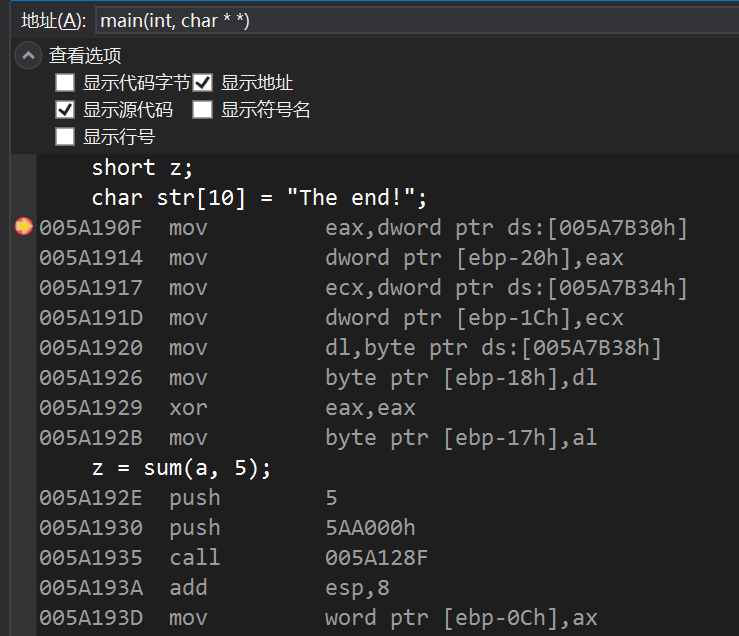
## 3.2实验过程

### 3.2.1 WINDOWS10下VS2019等工具包

3.2.1.1 从C语言到汇编语言

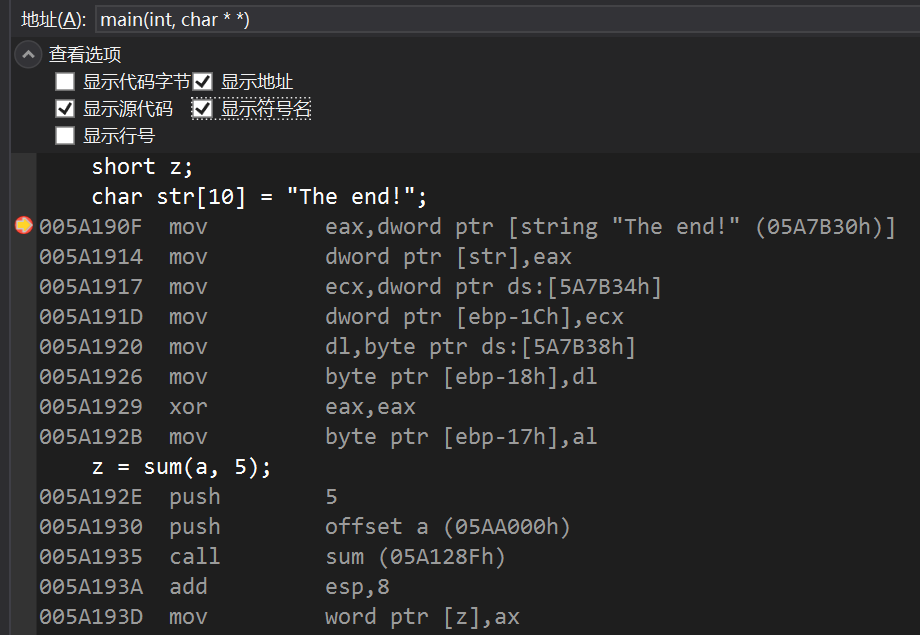
该部分实验环境：AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz，内存16GB，软件为WINDOWS10下的Visual Studio 2019。

1. 对于反汇编窗口中的“查看选项”，当未勾选时，调用变量时会直接使用其地址，如下图。



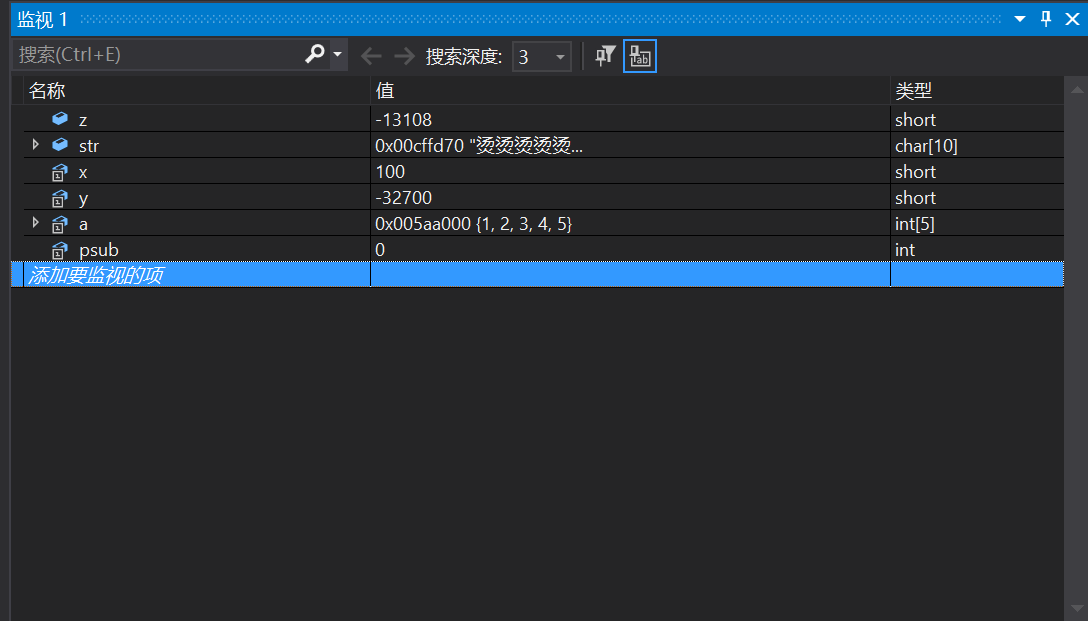
**图3.1 不勾选“显示符号名”**

而当勾选后，其调用变量时以[变量名]表示，如下图。



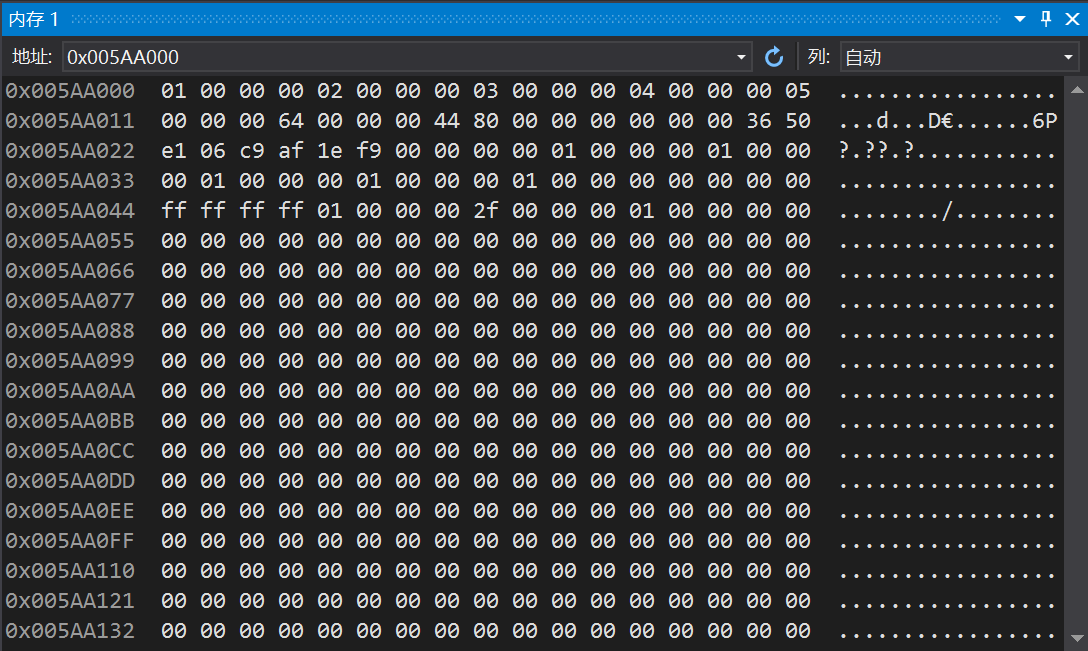
**图3.2 勾选“显示符号名”**

1. 查看监视窗口与内存窗口，监视窗口如下图所示，可以自行添加要监视的变量名称，从而可以看到其在程序运行期间的值。



**图3.3 监视窗口**

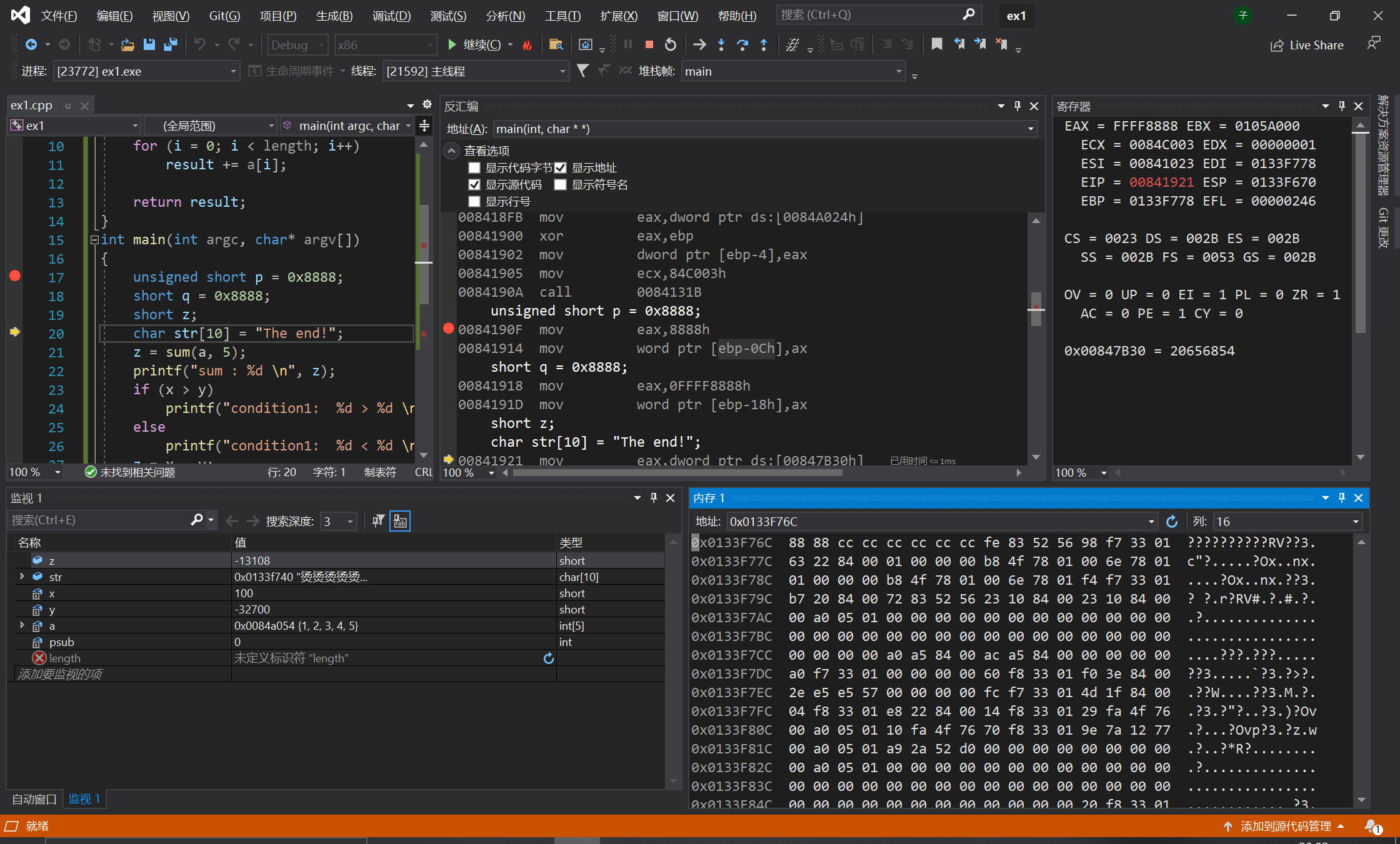
内存窗口如下图所示，图中0x005AA000为整型数组a的地址，可见每一个元素的存放方式和顺序。



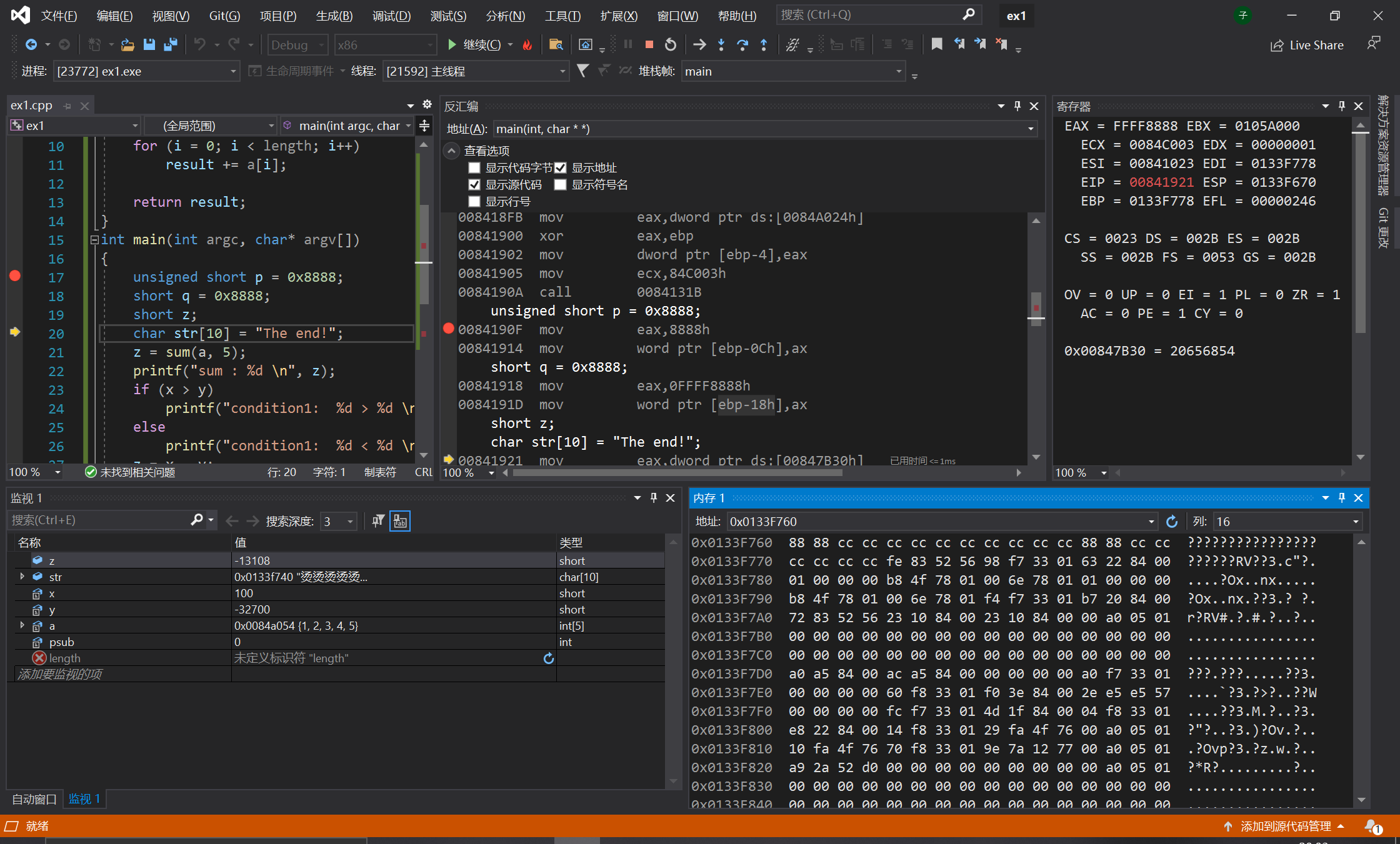
**图3.4 内存窗口**

1. 有符号和无符号整型数的存储方式，如下图所示。

其中p为unsigned short类型在内存中存储为8888，q为short类型在内存中存储也为8888。由此可见汇编器不会区分有符号和无符号整型数，而是将其均作为有符号数以补码形式存储。



**图3.5 p的存储**



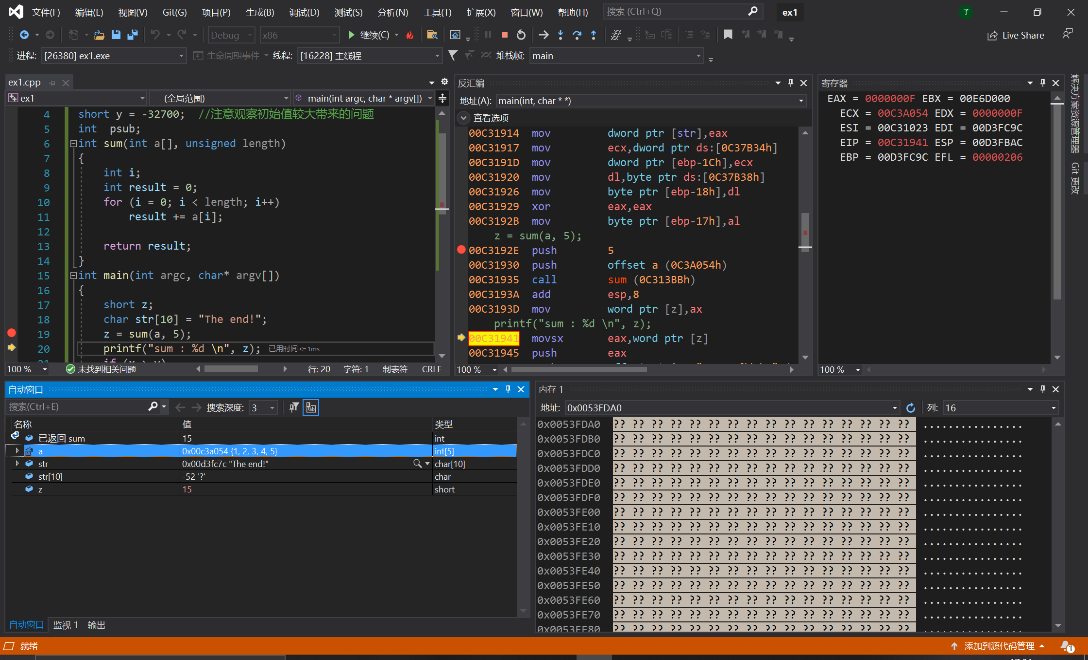
**图3.6 q的存储**

1. 有符号数和无符号数的加减没有区别，均是采用二进制的加减法。执行加法指令时，标志寄存器根据进位、溢出等情况进行更新对应的标志位的值。
2. 执行比较指令的时候，将被比较的两个数做减法，然后根据标志寄存器判断大小。并且根据比较数的有符号和无符号，比较依据也有所不同。
3. 观察思考：程序在编译时，在 sum 函数的for (i = 0; i < length ; i++) 处会给出警告信息： 有符号/无符号不匹配。请问，i<length 最终采用的是有符号数比较还是无符号数比较？ 若将语句“z = sum(a, 5);”换成“ z = sum(a,0x90000000)；”运行结果如何？为什么？如果将sum函数的参数 unsigned length改为 int length, 执行“z = sum(a,0x90000000)；”运行结果又如何？为什么？

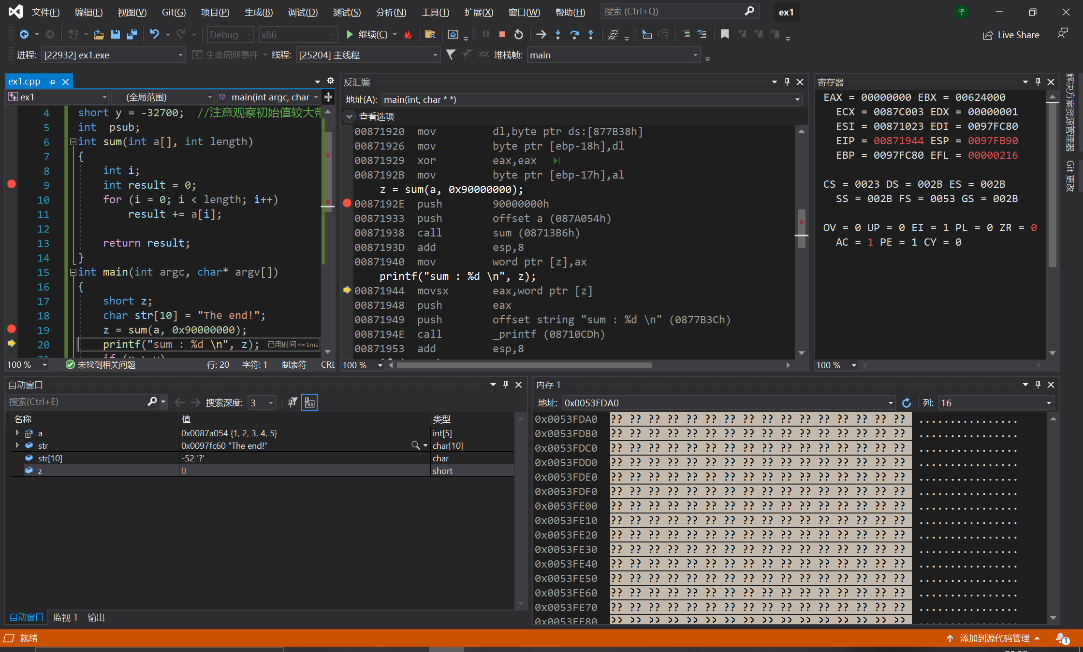
1.i<length 最终采用的是有符号数比较。

2.将语句“z = sum(a, 5);”换成“ z = sum(a,0x90000000)；”运行结果如图3.7所示，z=15。因为满足条件，将整形数组a中的元素相加，结果为15。

3.将sum函数的参数 unsigned length改为 int length, 执行“z = sum(a,0x90000000)；”运行结果如图3.8所示，z=0。因为采用有符号数比较，0x90000000为负数，for循环直接跳出，所以z最后为0。



**图3.7 换成“ z = sum(a,0x90000000)；”运行结果**



**图3.8 参数 unsigned length改为 int length运行结果**

3.2.1.2 观察汇编语言程序

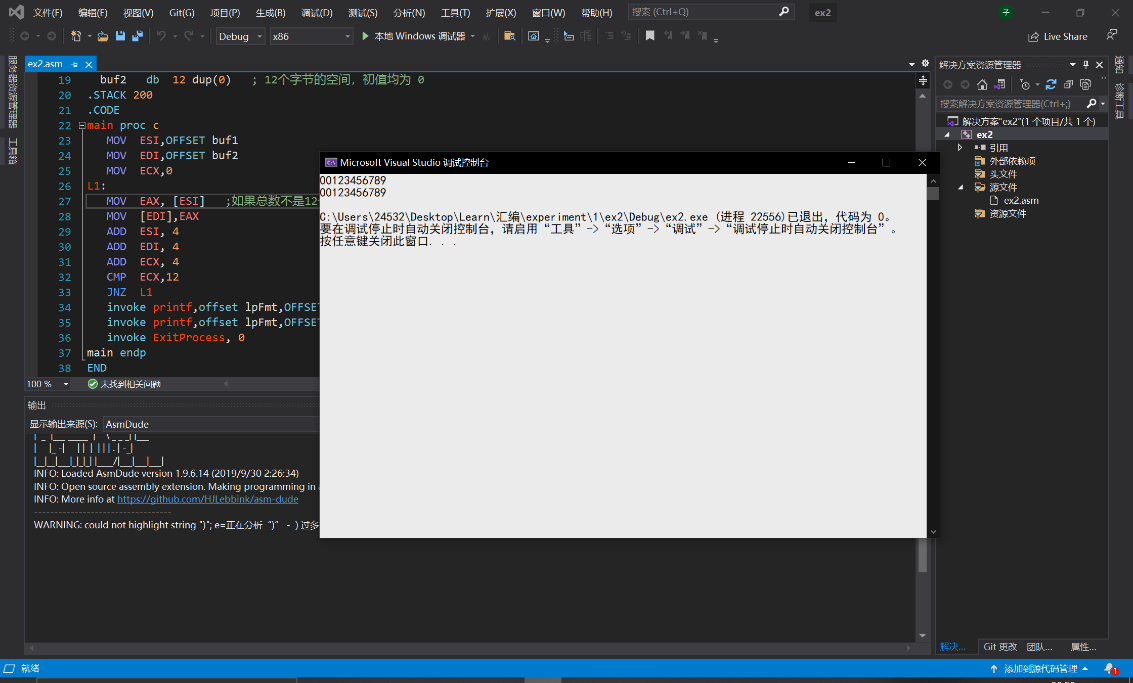
1. 实验任务：对给定的汇编语言源程序（其功能是：定义了一个数据段，并用指定的内存寻址方式，将buf1缓冲区中的12个字节内容拷贝到 buf2中,并显示两个缓冲区中的字符串），完成下列要求：

使用VS2019进行编译、链接和调试。完成反汇编窗口显示，了解汇编源程序中的语句与反汇编语句之间的关系。同时，完成同任务1.1的寄存器窗口、监视窗口、内存窗口的操作。观察数据段的存储结果，观察存储规律、各个变量的地址之间的关系；观察堆栈段内容。熟悉单步、断点等执行操作，尝试在调试状态下修改变量的值、指令的代码等。

分别将该程序不同部分的代码随意修改，观察编译器提示的错误信息的特点。尝试将访问buf1的寻址方式由寄存器间接寻址方式改成其他的寻址方式。

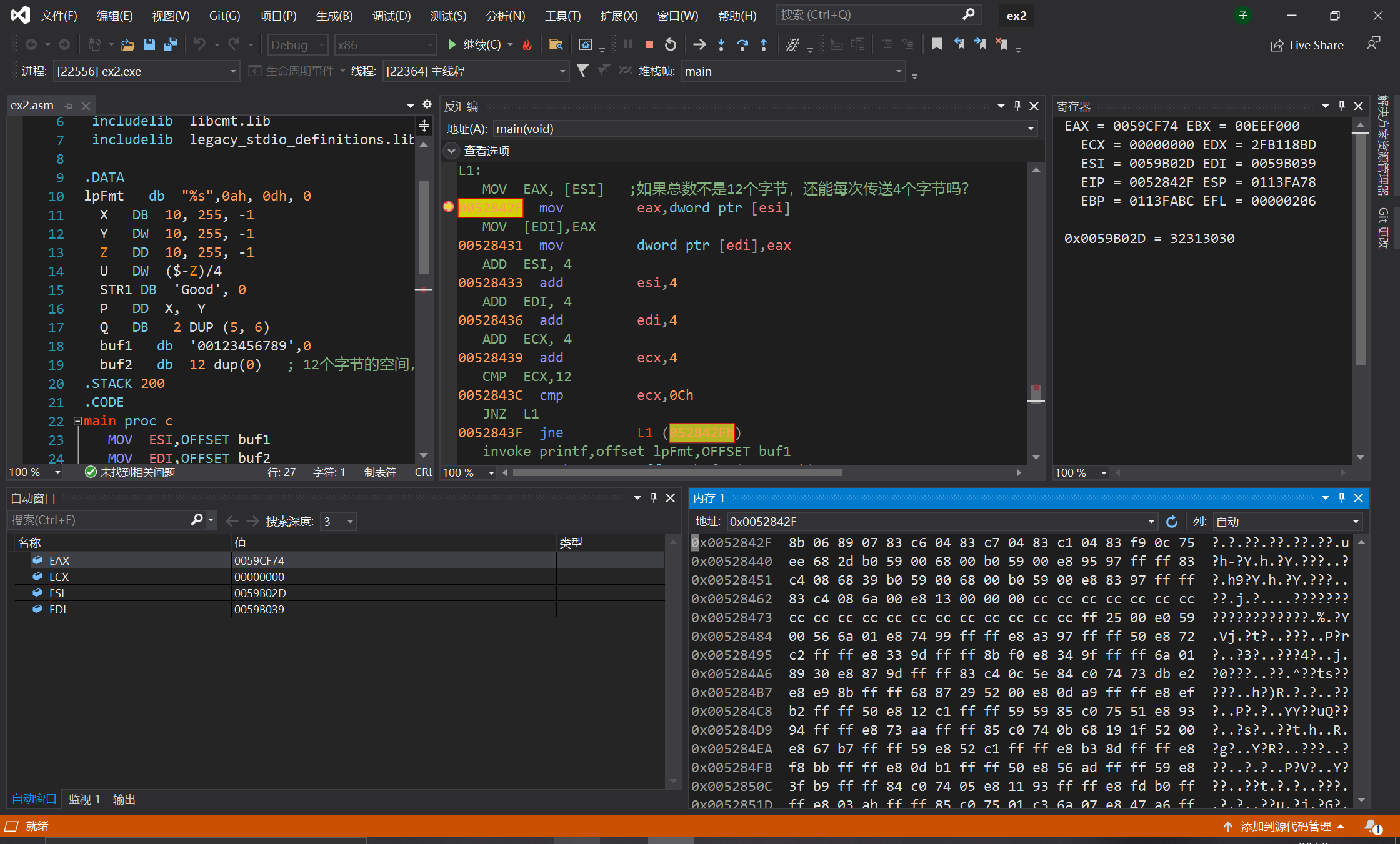
2. 实验记录与分析：

将给定的汇编语言源程序用VS2019运行，运行结果如图3.9所示。



**图3.9 程序运行结果**

寄存器窗口、监视窗口、内存窗口如图3.10所示



**图3.10 寄存器窗口、监视窗口、内存窗口**

将访问buf1的寻址方式由寄存器间接寻址方式改成其他的寻址方式如图3.11所示。将buf1的寄存器间接寻址改为了变址寻址，其输出结果相同。

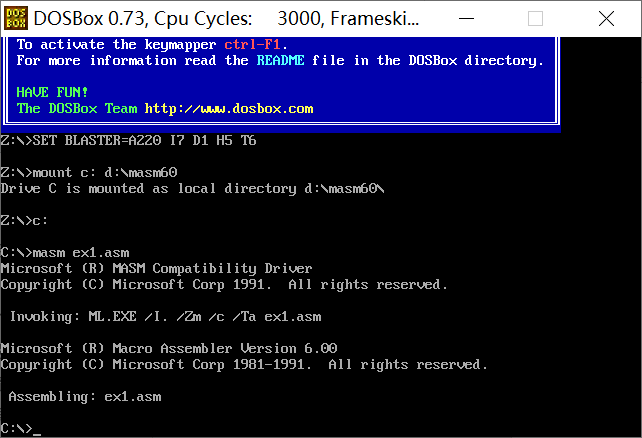


**图3.11 buf1寻址方式改为变址寻址**

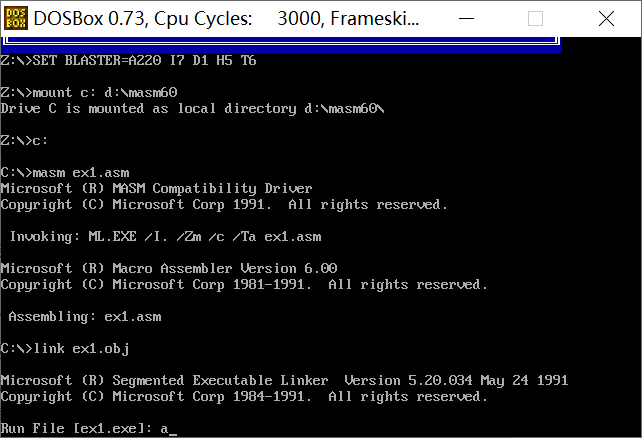
### 3.2.2 DOSBOX下的工具包

实验环境：AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz，内存16GB，软件为DOSBox。

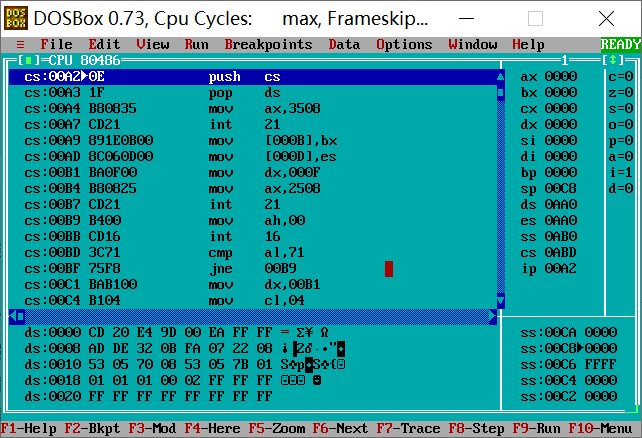
对于已编写好的asm汇编文件，在DOSBox中进行汇编和链接操作并进入调试分别如图3.12、图3.13和图3.14所示。



**图3.12 DOSBox汇编操作**

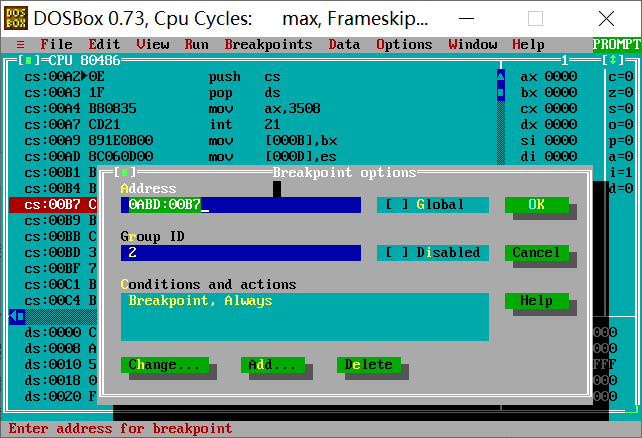


**图3.13 DOSBox链接操作**



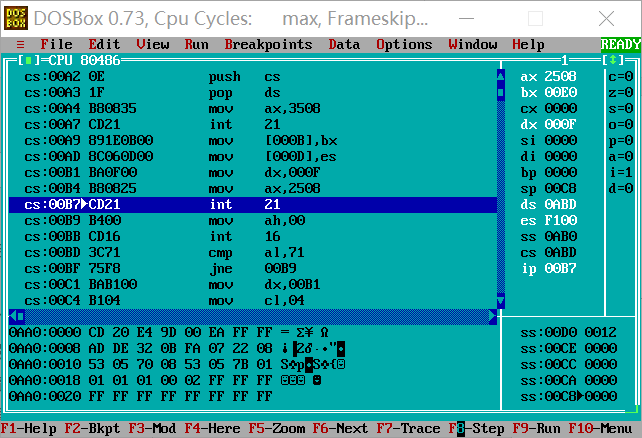
**图3.14 DOSBox调试界面**

在调试界面中可以进行通过F2设置断点、F6单步运行、F9直接运行等操作。设置断点如图3.15所示。



**图3.15 通过F2设置断点**

可见在00B7处设置了一处断点，设置完断点后运行至断点处如图3.16所示。

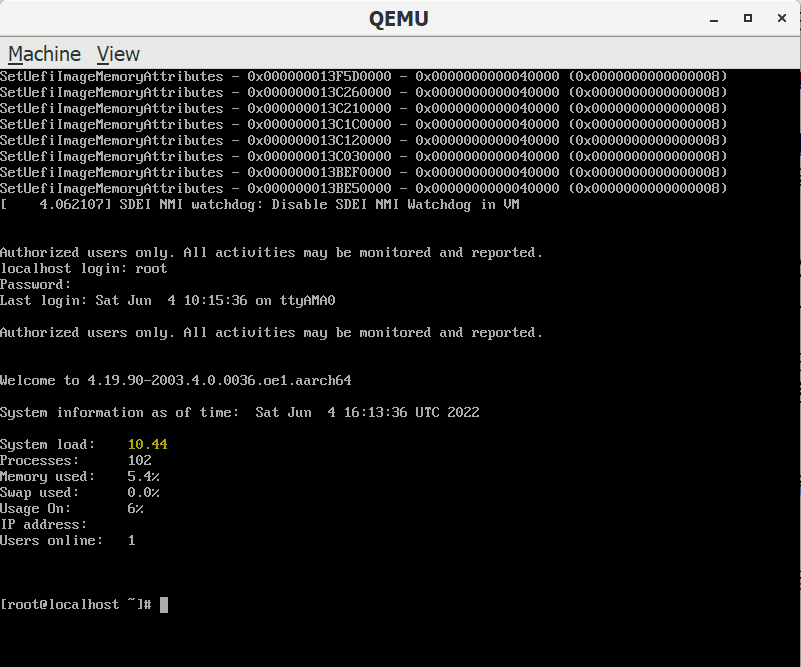


**图3.16 运行至断点处**

### 3.2.3 QEMU下ARMv8的工具包

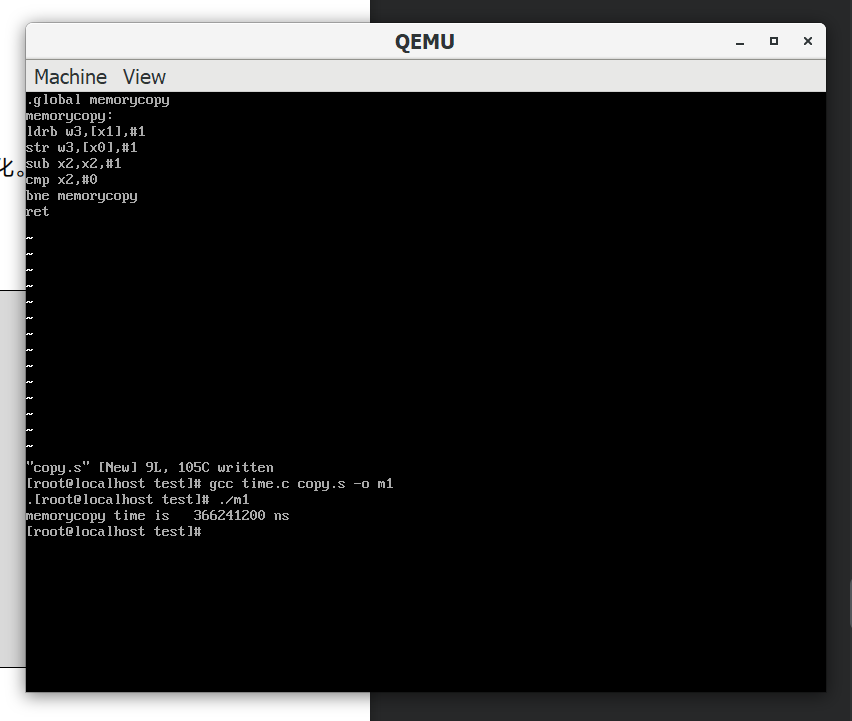
实验环境：AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz，内存16GB，软件为QEMU下ARMv8工具包。

通过CMD窗口打开QEMU并登陆成功如图3.17所示。



**图3.17 登录QEMU**

在QEMU中对编写的完成的源程序程序通过gcc进行编译链接并运行如图3.18和3.19所示。



**图3.18 通过gcc执行**



**图3.19 通过gcc执行**

## 3.3小结

Visual Studio2019使用感受：VS2019是一款功能齐全且强大的软件，界面美观操作方便，能实现很多基础的操作，但是鉴于其本身权限的局限性，无法进行实方式下的编程，所以仍然不能仅仅依靠这一个软件。

DOSBox使用感受：DOSBox给了用户在实方式下编写调试运行程序的工具，极大程度上方便了我们的开发与测试，突破了一定的限制，能够实际上亲身体验实方式下程序的编写时的各类事项。美中不足的时DOSBox本身虽然基本功能齐全，但是界面略微简陋且操作比较麻烦，这一点给实验带来了一些不便之处。

QEMU下ARMv8工具包使用感受：QEMU提供了使用ARMv8指令的平台，这一系列指令与x86和x64指令有着较大区别，通过QEMU对其进行操作能加深对ARMv8指令的认知，并更好地了解其与x86和x64指令之间的差别与共同点。但与DOSBox相似，QEMU界面较为简陋操作不太方便，需要熟悉gcc和gdb以及ARMv8指令本身的各种知识与操作，上手较慢。

# 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