汇 编 语 言 程 序 设 计 实 验 报 告

華中科技大學

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 202008 班

学　　号： U202015533

姓　　名： 徐瑞达

指导教师： 李专

实验时段： 2022 年 3 月 7 日~4 月 29 日

实验地点： 东九 A313

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：

汇 编 语 言 程 序 设 计 实 验 报 告

目录

一、程序设计的全程实践 3
1.1 目的与要求 3
1.2 实验内容 3
1.3 内容 1.1 的实验过程 3
1.3.1 设计思想 3
1.3.2 流程图 4
1.3.3 源程序 6
1.3.4 实验记录与分析 10
1.4 内容 1.2 的实验过程 11
1.4.1 实验方法说明 11
1.4.2 实验记录与分析 12
1.5 小结 13
二、利用汇编语言特点的实验 15
2.1 目的与要求 15
2.2 实验内容 15
2.3 实验过程 15
2.3.1 实验方法说明 15
2.3.2 实验记录与分析 16
2.4 小结 21
三、工具环境的体验 22
3.1 目的与要求 22
3.2 实验过程 22
3.2.1 WINDOWS10 下 VS2019 等工具包 22
3.2.2 DOSBOX 下的工具包 23
3.2.3 QEMU 下 ARMv8 的工具包 23
3.3 小结 24
参考文献 25

汇 编 语 言 程 序 设 计 实 验 报 告

一、程序设计的全程实践

1.1 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。

1.2 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。

1.3 内容 1.1 的实验过程

1.3.1 设计思想

任务 3.1 中，程序分为 3 部分：主程序、子程序和宏，具体功能说明与寄存器分配如下表：

表 1.1 程序功能说明表

程序划分		功能说明
主程序		**登录**:输入用户名和密码，直至用户名输入正确， 如果密码不正确则重新输入，错误超过三次则直接退出程序。
		执行:如果登录成功，则调用 copy 子程序和 display 子程序，然后根据用户输入决定下一步。
		用户操作:输入为 R 时，则初始化数据，重新执行 copy 子程序和 display 子程序；输入为 Q 时，则退出程序；输入其他内容时重新输入。
子程序	copy	**功能**:重复执行读取 N 组数据 numofTest 次 **参数**:无参数 **返回值**：无返回值 **传递参数方式**：约定寄存器 **寄存器分配**:使用 ecx 作为每次计算 N 组数据时的索引，使用 eax 存储计算结果和

		复制数据时的中间变量，使用 edi 作为每个数据区域的 index
	calculate	**功能**:计算某组数据的 f 值 **参数**:无参数 **返回值**:无返回值 **传递参数方式**：约定寄存器 **寄存器分配**:该子程序中，需要用到 ecx 作为读取的数据流中的 index，eax 用来存储计算结果
	display	**功能**:打印输出 MIDF 中的数据 **参数**:无参数 **返回值**:无返回值 **说明**:该子程序使用了全局变量 indexMID 用于打印 MIDF 中的有效数据 **寄存器分配**:该子程序使用了 ebx 作为当前打印 MIDF 区域数据的索引
宏	cmpstr	**功能**:比较字符串大小的宏 **参数**:存储比较结果的变量 flag，字符串 str1，str2
其他	存储结构说明	1. 使用结构体存储每组数据，结构体定义如下表 tuple struct SDA dd 0 SDB dd 0 SDC dd 0 tuple ends 2. 使用到的部分全局变量说明如下： LOWF,MIDF,HIGHF 均为结构体数组，用于存储各区域数据； indexLOW,indexMID,indexHIGH 分别为各区域数据数目； stream 为待计算数据数组；numofTest 为总测试次数；

1.3.2 流程图

任务 3.1 中主要程序流程图如下表：

表 1.2 主要程序流程图表

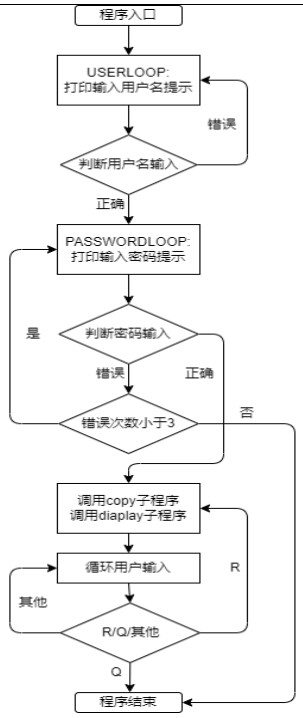


图 1.1 程序入口流程图

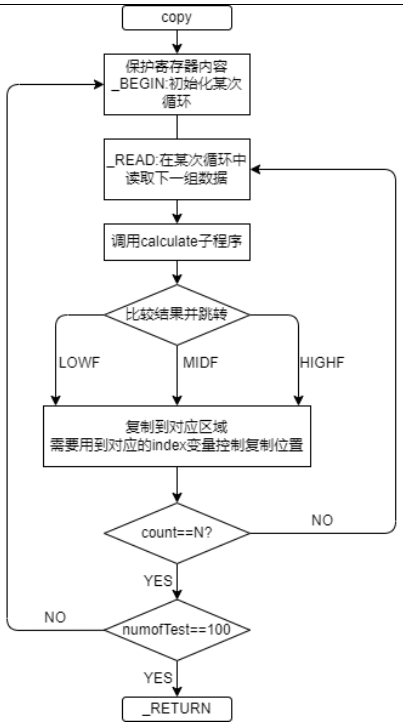


图 1.2 copy 子程序流程图

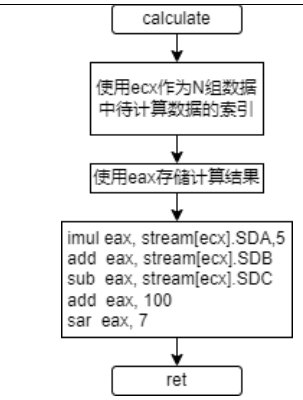


图 1.3 calculate 子程序流程图

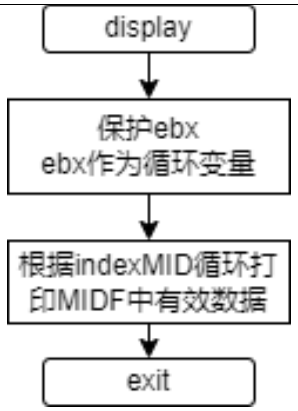


图 1.4 display 子程序流程图

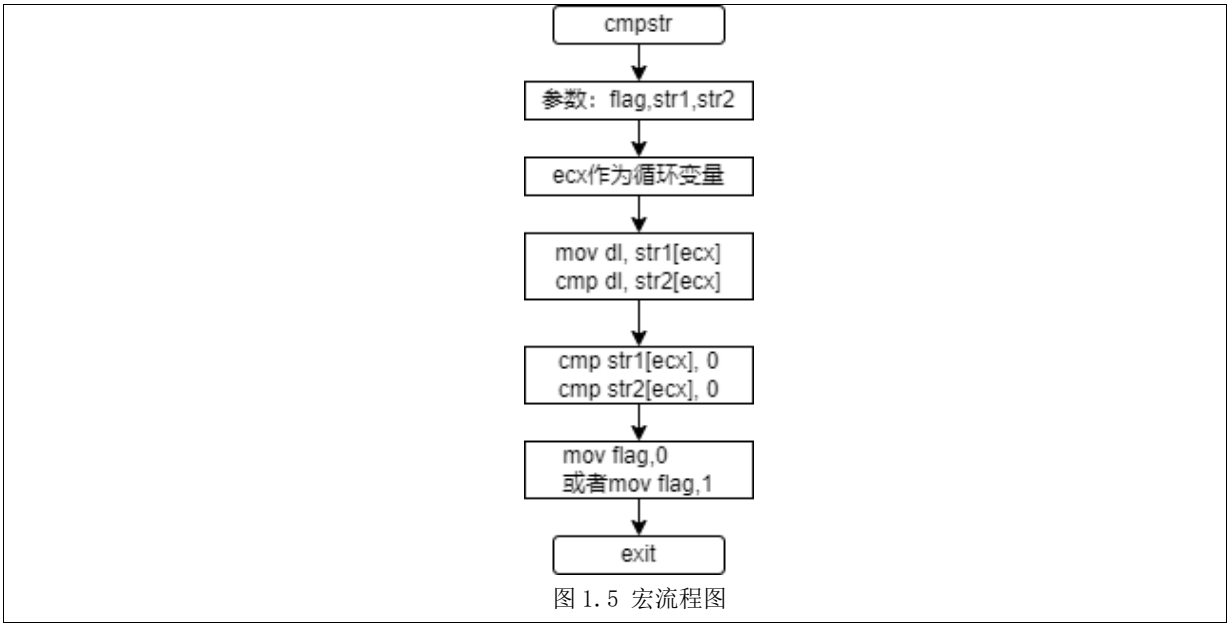


图 1.5 宏流程图

1.3.3 源程序

```
//*--------------------demo.asm----------------*//
.686P
.model flat, stdcall
 ExitProcess proto stdcall:dword
 includelib  kernel32.lib
 includelib  libcmt.lib
 includelib  legacy_stdio_definitions.lib
 printf PROTO C :VARARG
 scanf    PROTO C :VARARG
 getchar PROTO  C:VARARG
 calculate  proto
 copy       proto
 display    proto

 include struct.asm

cmpstr macro flag,str1,str2
 local L1, L2, L3, exit
 push ecx
 push edx
 xor ecx, ecx
L1:
 cmp str1[ecx], 0
 je L2
 mov dl, str1[ecx]
 cmp dl, str2[ecx]
 jnz L3
 inc ecx
 jmp L1
L2:
 cmp str2[ecx], 0
 jnz L3
 mov flag, 1
```

```
 jmp exit
L3:
 mov flag, 0
 jmp exit
exit:
 pop edx
 pop ecx
endm

 public
LOWF,MIDF,HIGHF,stream,indexLOW,indexMID,indexHIGH,numofTest,printTupleBuf,printBuf,calculateTip,di
splayTip

.DATA
  printBuf                      db  "%s",0ah,0dh,0
  printTupleBuf                 db  "%d,%d,%d",0ah,0dh,0
  scanBuf                    db  "%s",0
  inputOrderTip                 db  '>>Input your option(R|Q):',0
  usernameTip                db  '>>Input your username:',0
  usernameNotFoundTip        db  'User not found!',0
  passwordTip                db  '>>Input your password:',0
  passwordWrongTip           db  'Incorrect password!',0
  successTip                 db  'Logged in!',0
  calculateTip               db  'Calculating the tuples...',0
  displayTip                 db  'The tuples in the MIDF:',0
  defaultUsername            db  'xuruida',0
  defaultPassword            db  '123456',0
  inputPasswordTime          dd  0
  usernameFlag               dd  0
  passwordFlag               dd  0
  LOWF                          tuple 6 DUP(<0, 0, 0>)
  MIDF                          tuple 6 DUP(<0, 0, 0>)
  HIGHF                         tuple 6 DUP(<0, 0, 0>)
  stream                     tuple <250000,-1024,1300>
                             tuple <1,2,3>
                             tuple <2560,0,100>
                             tuple <25000, 30000, 7000>
                             tuple <-10110, -9333, 5678>
                             tuple <-12400, 29234, 9543>
  indexLOW                      dd  0
  indexMID                      dd  0
  indexHIGH                     dd  0
  numofTest                     dd  0
  inputUsername                 db  0
  inputPassword                 db  0
  inputOrder                 db  0
.STACK 200
.CODE
main proc c
USERLOOP:
          invoke printf,offset printBuf,offset usernameTip
          invoke scanf,offset scanBuf,offset inputUsername
          cmpstr usernameFlag,defaultUsername,inputUsername
          cmp usernameFlag,1
         je PASSWORDLOOP
          invoke printf,offset printBuf,offset usernameNotFoundTip
          jmp USERLOOP
```

```
    PASSWORDLOOP:
              invoke printf,offset printBuf,offset passwordTip
              invoke scanf,offset scanBuf,offset inputPassword
              cmpstr passwordFlag,defaultPassword,inputPassword
              cmp passwordFlag,1
              jz RIGHT
              inc inputPasswordTime
              .if inputPasswordTime < 3
                   invoke printf,offset printBuf,offset passwordWrongTip
                   jmp PASSWORDLOOP
              .else
                   jmp EXIT
              .endif
    RIGHT:
              invoke copy
              invoke display
    ORDERLOOP:
              invoke printf,offset printBuf,offset inputOrderTip
              invoke scanf,offset scanBuf,offset inputOrder
              cmp inputOrder,52H
              jz RIGHT
              cmp inputOrder,51H
              jz EXIT
              jmp ORDERLOOP
    EXIT:
              invoke Exitprocess,0
    main endp
    END
    //*-------------------function.asm----------------*//
    .686P
    .model flat, stdcall
     ExitProcess PROTO STDCALL :DWORD
     includelib  kernel32.lib
     includelib  libcmt.lib
     includelib  legacy_stdio_definitions.lib
     printf PROTO C :VARARG

      include struct.asm

    extern
LOWF:tuple,MIDF:tuple,HIGHF:tuple,stream:tuple,indexLOW:dword,indexMID:dword,indexHIGH:dword,numofT
est:dword,printTupleBuf:byte,printBuf:byte,calculateTip:byte,displayTip:byte;

    .STACK 200
    .CODE
    calculate proc
     imul eax, stream[ecx].SDA,5
     add  eax, stream[ecx].SDB
     sub  eax, stream[ecx].SDC
     add  eax, 100
     sar  eax, 7
     ret
    calculate endp

    copy proc
     push eax
     push ebx
```

```
push ecx
push esi
push edi
mov numofTest,0
invoke printf,offset printBuf,offset calculateTip
_BEGIN:
mov indexLOW,0
mov indexMID,0
mov indexHIGH,0
xor ecx,ecx
_READ:
invoke calculate
cmp  eax,100
je   _MID
jl   _LOW
jg   _HIGH
_MID:
    mov edi,indexMID
mov eax, stream[ecx].SDA
mov MIDF[edi].SDA,eax
mov eax, stream[ecx].SDB
mov MIDF[edi].SDB,eax
mov eax, stream[ecx].SDC
mov MIDF[edi].SDC,eax
add indexMID,12
jmp _CIRC
_LOW:
mov edi,indexLOW
mov eax, stream[ecx].SDA
mov LOWF[edi].SDA,eax
mov eax, stream[ecx].SDB
mov LOWF[edi].SDB,eax
mov eax, stream[ecx].SDC
mov LOWF[edi].SDC,eax
add indexLOW,12
jmp _CIRC
_HIGH:
mov edi,indexHIGH
mov eax, stream[ecx].SDA
mov HIGHF[edi].SDA,eax
mov eax, stream[ecx].SDB
mov HIGHF[edi].SDB,eax
mov eax, stream[ecx].SDC
mov HIGHF[edi].SDC,eax
add indexHIGH,12
jmp _CIRC
_CIRC:
add ecx,12
cmp ecx,72
jnz _READ
inc numofTest
cmp numofTest,100
jnz _BEGIN
_RETURN:
pop edi
pop esi
pop ecx
```

```
 pop ebx
 pop eax
 ret
copy endp

display proc
 push ebx
 invoke printf,offset printBuf,offset displayTip
 xor ebx,ebx
 .while(ebx < indexMID)
      invoke printf, offset printTupleBuf, MIDF[ebx].SDA, MIDF[ebx].SDB,MIDF[ebx].SDC
      add ebx,12
 .endw
 pop ebx
 ret
 display endp
 END
//*--------------------struct.asm----------------*//
tuple struct
 SDA       dd   0
 SDB       dd   0
 SDC       dd   0
tuple ends
```

1.3.4 实验记录与分析

1. 实验环境条件

Intel(R)Core(TM)i5-1035G1 CPU 1.00GHz(8 CPUs)；

CPU 主频为 1.2GHz，8G 内存；

WINDOWS11 下 VS2022 社区版。

2. 汇编、链接中的情况

汇编过程中出现了若干个问题，主要出现在变量的申明与引用，数据的复制两处:

（1） 在多模块间互相调用变量时，由于 **extern 和 public 的使用**存在问题，导致问题的产生，在阅读教材仔细了解其用法并试错后成功解决；

（2） 在进行数据的复制时，由于对**变址寻址方式**理解不透彻导致存储数据发生紊乱，在弄清楚相关概念后解决问题。

（3） 当尝试将结构体定义单独放在一个文件中以被其他模块引用时，发现编译总是无法通过，和同学讨论试错后，发现应该将 struct.asm 从项目中排除，直接在其他模块中 include 该文件。

（4） 宏定义中标号的展开——在进行编译时，编译器将为宏中使用 LOCAL 伪指令定义的局部标号表中的每一个局部标号建立唯一的符号(用??0000~??FFFF)，以代替在展开中存在的每一个局部标号，避免了多次调用宏时标号的重复。

3. 程序基本功能的验证情况

（1） 账号密码功能的验证

输入错误的账号会一直循环输入账号，输入正确的账号后方可输入密码；输入错误的密码后提示错误，错误超过三次后自动退出程序，正确输入后进入主程序。

（2） 数据的处理和复制功能验证

设定好能够落在各个区间范围内的数据，在调试时通过内存的观察、寄存器内容的观察、局部变

量的监视查看数据内容，与预期一致，验证正确。

（3） 用户循环输入功能验证

依次输入 R、Q 及其他值，与预期一致，验证正确。

表 1.3 程序基本功能验证表

输入密码正确的验证图	输入密码错误过多的验证图	用户选择输入的验证图
>>Input your username: xuruida123 User not found! >>Input your username: xuruida >>Input your password: 123456 Logged in!	>>Input your username: xuruida >>Input your password: 1234567 Incorrect password! >>Input your password: 12345 Incorrect password! >>Input your password: 1234567 D:\HUST-Courses\8_汇编语言实验\ 按任意键关闭此窗口. . .	>>Input your option(R|Q): R Calculating the tuples... The tuples in the MIDF: 2560,0,100 >>Input your option(R|Q): S >>Input your option(R|Q): Q

4. 使用调试工具观察、探究代码的情况

（1） 在使用反汇编功能时，发现调用函数的 invoke 指令被反汇编为 push 和 call 指令，这与上课时讲的内容相符合；如下图所示，invoke printf 函数时，两条 push 指令将参数压栈，之后 call 指令调用子程序，子程序返回后，esp 增加 8 以回收参数所占用的空间。

```
  invoke printf,offset printBuf,offset usernameTip
push        offset usernameTip (072D02Dh)
push        offset printBuf (072D000h)
call        _printf (06B1BE5h)
add         esp,8
```

图 1.6 invoke 指令反汇编图（以 printf 函数为例）

（2） 不同文件代码段的合并情况分析

在不同代码段处标记断点后，进入调试。在反汇编窗口中找到并记录对应代码的开始和结束位置，发现主程序(demo.asm)的代码段在其他文件(function.asm)的代码段之前。

1.4 内容 1.2 的实验过程

1.4.1 实验方法说明

1. 指令优化对程序的影响

指令优化方面，主要关注选择执行速度较快的指令，减少指令的条数或循环次数等策略。

（1） 置零时用 xor 指令替换 mov 指令；

（2） 除以 128 时用向右移位指令替换除法指令；

（3） 使用基址加变址寻址替换变址寻址，以减少对内存单元的访问；

（4） 复制结构体中的数据时，根据结构体内容的连续性，一次性复制某个结构体中的内容，提高效率。

以上四条优化策略都能提高程序运行效率。

2. 约束条件、算法与程序结构的影响

（1） 约束条件方面——主要考虑溢出问题、计算精度问题

在不考虑溢出时，可以优化逻辑结构，减少指令条数；如果考虑溢出，则需要分别对余数和商进行计算，这时进行运算的语句条数将有所增加；

计算结果 f 的精度问题取决于有效位的长度，当考虑较长的有效位时需要使用额外的空间存储。

（2） 算法方面——主要考虑公式的计算方面

在计算时，计算顺序不同，进行的运算次数也不同，这里采用先计算括号内公式，最后进行除法的计算顺序。

（3） 程序结构方面——将出现可能性较大的分支放在其他分支之前，可以减少跳转指令和比较指令的条数。

3. 编程环境的影响

在实验 5 中，通过观察 x64 与 x86 的 32 位环境下相同功能程序执行的效率，发现机器字长越长，对应汇编环境下执行效率要更高。

通过与其他同学的运行效率的对比，发现性能更好的 CPU 也能提高执行效率。

1.4.2 实验记录与分析

1.优化实验的效果记录与分析

表 1.4 优化前后程序执行时间对比

次数	优化前（CPU 时间）	优化前（ms）	优化后（CPU 时间）	优化后（ms）
1	3400855355	2834.046	1985066135	1654.222
2	3379666955	2816.389	2071900080	1726.583
3	3348557085	2790.464	1901517069	1584.598
4	3353198069	2794.332	2068433954	1723.695
5	3392750172	2827.292	2016943394	1680.786
平均时间	3375005527	2812.505	2008772126	1673.977

备注: 执行 100,000,000 次循环，根据 CPU 主频为 1.2GHz 计算所用的毫秒数

由上表计算得知，使用三条对指令的优化策略优化后用时仅为优化前用时的 59.5%，优化效果较理想。

2. 不同约束条件、算法与程序结构带来的差异

这部分差异记录时，采用在表 1.4 中对指令进行优化的基础上对代码进行更改来达到效果。

表 1.5 不同约束条件、算法与程序结构带来的差异对比

次数	算法方面优化前	算法方面优化后	约束条件优化前	约束条件优化后
1	2834.046	2978.010	1654.222	1585.623
2	2816.389	3125.784	1726.583	1760.392
3	2790.464	3108.245	1584.598	1542.824
4	2794.332	3027.446	1723.695	1842.642

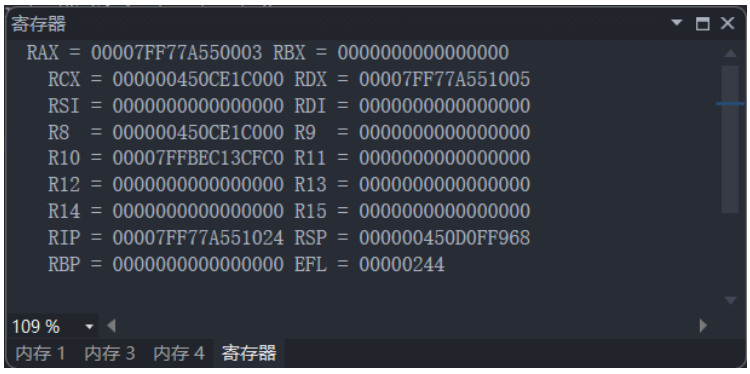
5	2827.292	2962.975	1680.786	1625.543
平均时间	2812.505	3040.492	1673.977	1671. 405

备注: 执行 100,000,000 次循环，表中展示的数据均为毫秒数

由上表计算得知，对算法方面的优化和对约束条件的优化效果并不明显，甚至有稍微的负优化。

3.几种编程环境中程序的特点记录与分析

表 1.6 x64 与 x86 的汇编程序风格特点比较对比

	x64 与 x86 的汇编程序风格特点比较
实验 5 中对 x64 和 x86 汇编程序的观察结果分析	（1）在汇编文件开始处，x64 环境下没有处理器选择伪指令、存储模型说明伪指令.model 等伪指令； （2)控制台默认的程序入口为 mainCRTStartup，需要用户自己设置程序运行入口； （3）x64 环境下不支持使用.stack 伪指令定义堆栈段大小； （4）invoke 伪指令只用于 x86 环境，在 x64 环境下，需要程序员自行传参并使用 call 伪指令调用函数； （5）x64 新增了 rip 相对寻址方式； （6）x64 与 x86 使用的寄存器位数和数目不同。  图 1.7 x64 下寄存器

1.5 小结

实验 2 中主要进行对汇编程序运行效率的优化。通过优化指令（使用移位指令替换除法指令等）和优化程序结构的方式（调整分支结构等），较为理想地提高了程序的运行效率。同时发现一些指令的优化（如使用 xor 指令置零）并不会给运行效率带来较大的优化，分析发现这些指令的位置并不是决定程序执行效率的关键位置，因此在分析如何提高运行效率时，需要先观察程序得出**决定程序执行效率的关键所在**，而不是盲目优化。

实验 3.1 中主要进行多模块程序设计。通过试错与讨论，得出了 extern 和 public 在多模块间跨模块引用变量与函数的使用方法，掌握了子程序的传参与调用方式，知道了宏的定义及与子程序的不同。

extern 与 public 的使用方法：跨模块使用变量时，需要在定义变量的模块中使用 public 声

明，而在引用变量的模块中使用 extern 声明；跨模块使用函数时，只需在引用函数的模块前使用 proto 声明即可。

子程序的传参与调用方式：使用 invoke 伪指令时，会将参数压栈后使用 call 调用子程序；使用 call 指令时，首先将断点地址压栈，然后送子程序地址给 EIP 以跳转至子程序所在位置；在子程序中，会将定义的局部变量压栈；使用 ret 指令时将栈顶的断点地址送 EIP 以回到调用处的下一条语句继续执行。

宏的相关理解：宏只是在调用宏的位置进行简单的替换和展开，在编译时即完成，而调用子程序时需要改变 EIP 以跳转，且多次调用子程序时，会多次访问子程序所在地址；宏定义中，也可以声明标号，可以选择使用寄存器或者变量传返回值。

实验 3.2 中主要进行 C 语言和汇编语言的混合编程。在此之中，观察得出了两种语言中调用子程序的不同之处——使用 call 调用 C 语言子程序时，会在调用处根据 invoke 传入堆栈的参数所占字节数增加 esp 的值回收空间，而使用 call 调用汇编子程序时，则根据 ret n 语句改变 esp 的值。同时，也更加深入理解了存储模型说明伪指令.model 声明的语言类型，函数声明时 proto 后的语言类型，函数定义时指定的语言类型之间的联系。

实验 5.1 中主要进行 x64 与 x86 汇编编程的对比。通过对实现了相同功能的 x64 和 x86 的汇编程序的观察，调试后得出了两种环境下汇编编程的异同点（在 1.4.2 中已经指出）。

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

（1） 设计思想：通过 INT 21H 读取八号中断矢量并存储，然后将新的中断程序地址写入八号中断矢量表。

（2） 主程序：判断是否安装中断处理程序（通过检查 BX 和 NEW08H 是否相同），如果已经安装则退出，否则安装中断处理程序，最后转到驻留退出程序（将中断处理程序占用的字节数送入 DX，将字节数转化为节数并设置驻留区大小后，通过 INT 21H 驻留退出）。

（3） 中断处理程序：执行原中断处理程序，读取时间信息并显示时间，中断返回。

（4） 实验方法：通过多次运行该程序并使用 TD 工具调试来观察中断矢量表及程序运行过程中的变化。

2.反跟踪程序的设计思想与实验方法

表 2.1 反跟踪与加密策略

主要方法	设计思想
加密密码	将用户名逐个字符与’A’异或，达到加密效果，防止通过用户名猜测密码位置；将在源程序数据段中定义的密码在汇编之后变成密文，如采用异或运算，这里采用的加密公式为(X-21H)*7，并使用随机字符填充密码。
静态修改代码	在代码中穿插数据定义或无关代码，起到扰乱作用； 在子程序返回处定义数据，在子程序内自我修改返回地址，起到扰乱作用。

动态修改代码	将无关代码对应的机器码定义到数据段中，在程序运行过程中将这些机器码复制到代码段中，即可达到动态修改代码的目的。

表 2.2 跟踪与解密策略

主要方法	设计思想
利用静态反汇编工具	将程序反汇编后，查看代码逻辑，尝试寻找计算公式；用到的工具有 objdump 和 Win32Dasm
利用二进制文件编辑工具暴力破解	利用 vs2019 以二进制格式打开程序，尝试破解用户名与密码。
动态跟踪调试	利用 vs2019 附加到进程调试，动态跟踪，尝试破解。

3.指令优化及程序结构的实验方法

实验 2、3 的优化方法见 1.4.1

2.3.2 实验记录与分析

1.中断处理程序的特别之处

（1） 与 P224 例 6.2 功能不同的代码部分如下：

```
;初始化(中断处理程序的安装)及主程序
BEGIN:    PUSH CS
      POP       DS
      MOV       AX,3508H  ;获取原08H的中断向量，调用参数为中断类型号为AL，返回参数为ES:BX=中断向量
      INT       21H
      ;;;;判断是否已安装中断处理程序
      CMP       BX,OLD_INT
      JNE       INSTALLED              ;若BX与OLD_INT对应BX不相等，不相等说明已安装
      mov      AX,ES
      cmp       AX,OLD_INT+2
      JNE       INSTALLED              ;若ES与OLD_INT对应ES不相等，说明已安装
      MOV       OLD_INT,BX
      MOV       OLD_INT+2,ES           ;将ES:BX保存至OLD_INT中(保存中断矢量)
      MOV       DX,OFFSET NEW08H
      MOV       AX,2508H          ;设置新的08H中断向量，调用参数为DS:DX=中断向量，AL=中断类型号
      INT       21H
NEXT: ;MOV      AH,0                   ;从键盘读入字符送AL寄存器
      ;INT      16H                         ;等待按键
      ;CMP      AL,'q'
      ;JNE      NEXT
      MOV  AX, DATA
      MOV  DS, AX                      ;DS指向DATA段以取出字符串
      LEA  DX, STR1                 ;DX表示待打印信息MEASSGAE的偏移地址
      MOV  AH, 9H                      ;显示字符串，调用参数为DS:DX=串地址，其中'$'结束字符串
      INT  21H
      JMP  INTERRUPT
INSTALLED:
      MOV  AX, DATA
      MOV  DS, AX                      ;DS指向DATA段以取出字符串
      LEA  DX, STR2                 ;DX表示待打印信息MEASSGAE的偏移地址
      MOV  AH, 9H                      ;显示字符串，调用参数为DS:DX=串地址，其中'$'结束字符串
```

```
    INT  21H
INTERRUPT:
    MOV  DX, OFFSET BEGIN +15    ;计算中断处理程序的字节数，+15以便于在计算节数时向上取整
    MOV  CL, 4
    SHR  DX, CL                  ;每节代表16个字节，将字节数转换为节数
    ADD  DX, 50H                 ;设置驻留区大小，需包括程序段前缀的内容(50H个字节)
    MOV  AL, 0                   ;设置返回码为0
    MOV  AH, 31H
   ;结束并驻留，调用参数为AL=返回码，DX=驻留区大小，退出时,将(DX)节的主存单元驻留(不释放)
    INT  21H
```

（2） 运行效果如下：

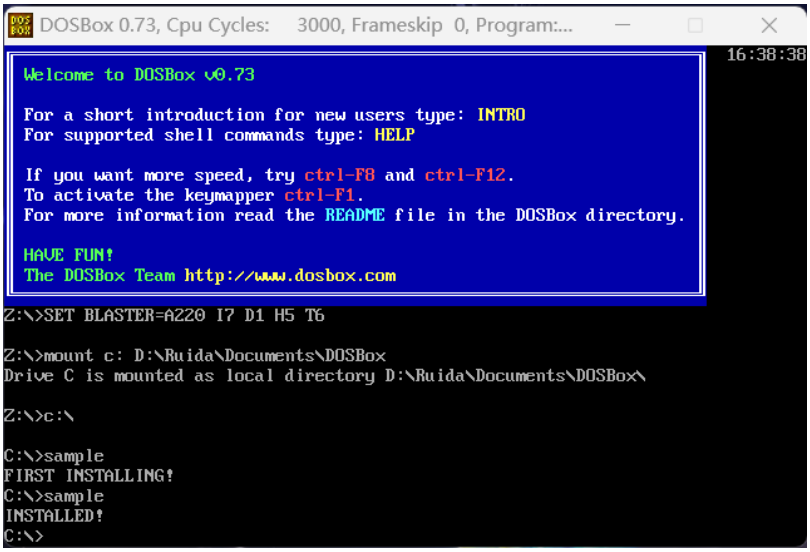


图 2.1 中断处理程序执行图

如图所示，可见，首次安装中断处理程序时会提示“TIRST INSTALLING”，而之后再次尝试安装中断处理程序时会提示“INSTALLED”，同时，实现了驻留退出，时钟显示的进程会驻留，即使退出程序，仍能够正常显示。

（3） 中断矢量表的观察与验证

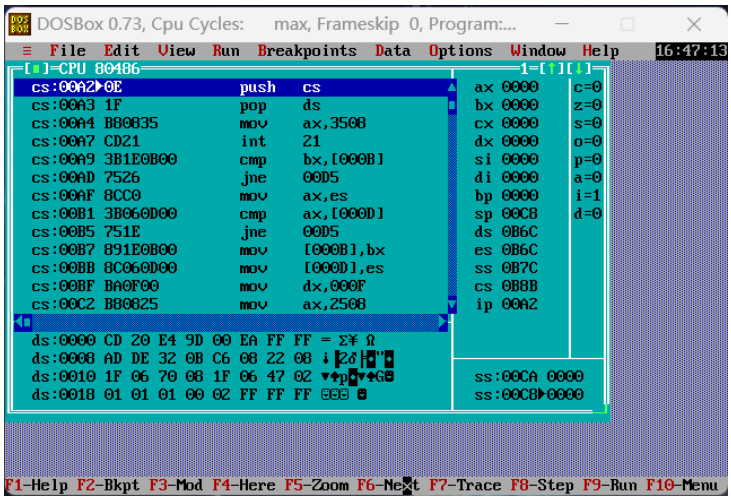


图 2.2 中断矢量表查看图

通过 TD 调试工具打开 sample.EXE 调试，在内存区域转到“0000:0000”即可查看中断矢量表；

通过 F7 逐步调试可以观察到，在执行过程中，首先会取出 08H 处的中断处理程序并比较判断是否已安装中断处理程序，如果已经安装便直接退出，否则安装后驻留退出。

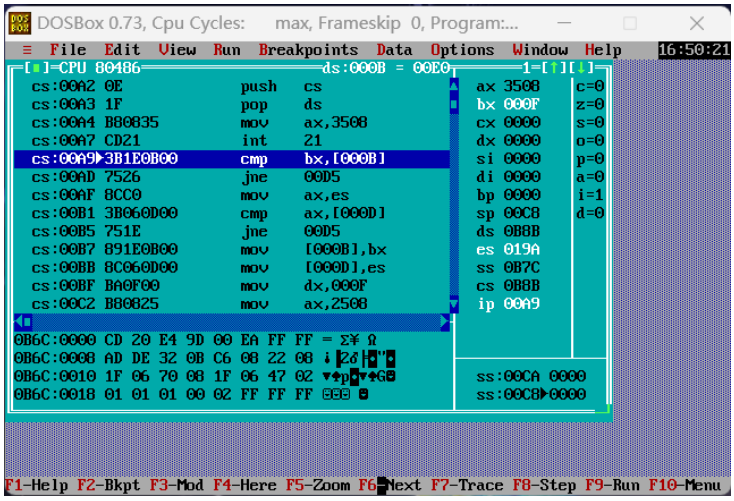


图 2.3 取出 08H 中断矢量图(可以看到右侧 bx 和 es 寄存器变化)

2.反跟踪效果的验证

（1） 加密用户名与密码

如图 2.4-2.5 所示，使用字符'A'与用户名逐字符异或形式加密，使用公式(X-21H)*7 的方法加密密码。

```
;采用与密码串依次异或的方法加密用户名，长度为7
username            db  'x' XOR 'A'
                    db  'u' XOR 'A'
                    db  'r' XOR 'A'
                    db  'u' XOR 'A'
                    db  'i' XOR 'A'
                    db  'd' XOR 'A'
                    db  'a' XOR 'A',3 DUP(0)
```

图 2.4 用户名加密

```
;采用函数(X-21H)*7的方法加密密码，同时使用随机字符填充密码，长度为10
password            db  ('d'-21H)*3
                    db  ('a'-21H)*3
                    db  ('r'-21H)*3
                    db  ('l'-21H)*3
                    db  ('i'-21H)*3
                    db  ('n'-21H)*3
                    db  ('g'-21H)*3,3 DUP(0)
```

图 2.5 密码加密

同时，在比较用户输入的用户名和密码时，也需要对明文使用同样的加密算法加密以与密文进行比较。

（2） 穿插数据定义与无关代码

通过穿插数据定义可以混淆反汇编时代码的解读，起到干扰作用；通过穿插无关代码也可以掩盖程序的算法逻辑。同时，在 call 语句的下一行进行数据定义，并在子程序内自我修改返回地址，有助于进一步干扰破解工作。

以比较用户名时的自我修改返回地址操作为例，如图 2.6 所示。

在图中，首先将比较模式（mode 为 0 时比较用户名，mode 为 1 时比较密码）和用户输入的字符串压栈，然后调用 cmpstr 进行比较，同时，在 call 语句下面定义了变量 username，该数据既可

以作为子程序中比较时的参数（因为其地址即为断点地址，将在 call 语句中被压栈），也可以作为干扰代码，被反汇编为无关代码，如图 2.7 所示。定义的数据 username 将被返回变为 call 语句下的三条无关代码，混淆视线。而在子程序中，将通过“pop edi”，“add edi,10”，“push edi”，“ret”等语句修改返回地址，以在子程序返回后回到正确的位置，如在图 2.6 中将返回到“cmp eax,1”语句处。

```
mov mode,0
push mode
push offset inputUsername
call cmpstr
;采用与密码串依次异或的方法加密用户名，长度为7
username            db  'x' XOR 'A'
                    db  'u' XOR 'A'
                    db  'r' XOR 'A'
                    db  'u' XOR 'A'
                    db  'i' XOR 'A'
                    db  'd' XOR 'A'
                    db  'a' XOR 'A',3 DUP(0)
cmp  eax,1
je PASSWORDLOOP
```

图 2.6 比较用户名时的自我修改返回地址操作

```
            call cmpstr
00F68489 E8 DC 00 00 00       call        cmpstr (0F6856Ah)
00F6848E 39 34 33             cmp         dword ptr [ebx+esi],esi
00F68491 34 28                xor         al,28h
00F68493 25 20 00 00 00       and         eax,20h
            ;采用与密码串依次异或的方法加密用户名，长度为7
            username            db  'x' XOR 'A'
                                db  'u' XOR 'A'
                                db  'r' XOR 'A'
                                db  'u' XOR 'A'
                                db  'i' XOR 'A'
                                db  'd' XOR 'A'
                                db  'a' XOR 'A',3 DUP(0)
```

图 2.7 数据定义反汇编示意

此外，在程序中还穿插了多处数据定义与无关代码，举例如图 2.8 所示

```
inc  edi
inc  esi
jmp  usernameJudge
trueUsername db 'daruixu',0
```

图 2.8 穿插数据定义与无关代码示意

（3） 动态修改代码

动态修改执行代码需要在程序执行过程中修改代码段，因此需要使用 VirtualProtect 函数来暂时允许向相关代码段中写入代码。这里采用该手段处理计算结果 f 的函数 calculate，如图 2.9 所示。

首先，将计算过程反汇编后得到的机器码定义到数据段中，定义如下：

```
1.    abstractCode  db 06BH,004H,00EH,005H,003H,044H,00EH,004H,02BH,044H,00EH
,008H,083H,0C0H,064H,0C1H,0F8H,007H,0C3H
2.    abstractCodeLen=$-abstractCode
```

```
    imul eax, [ESI+ecx].tuple.SDA,5
00DE8630 6B 04 0E 05          imul        eax,dword ptr [esi+ecx],5
    add  eax, [ESI+ecx].tuple.SDB
00DE8634 03 44 0E 04          add         eax,dword ptr [esi+ecx+4]
    sub  eax, [ESI+ecx].tuple.SDC
00DE8638 2B 44 0E 08          sub         eax,dword ptr [esi+ecx+8]
    add  eax, 100
00DE863C 83 C0 64             add         eax,64h
    sar  eax, 7
00DE863F C1 F8 07             sar         eax,7
    ret
00DE8642 C3                   ret
```

图 2.9 计算过程反汇编

然后，在 calculate 子程序中定义如下代码：

```
calculate proc
    cmp copyFinished,0
    jne abstractCodeDes
    mov eax,abstractCodeLen
    mov ebx,40h
    lea ecx,abstractCodeDes
    invoke VirtualProtect,ecx,eax,ebx,offset oldprotect
    mov esi,offset abstractcode
    mov edi,offset abstractCodeDes
    mov ecx,abstractCodeLen
    mov copyFinished,1
copycode:
    mov al,[esi]
    mov [edi],al
    inc esi
    inc edi
    loop copyCode
    mov esi, offset stream
abstractCodeDes:
    db abstractCodeLen dup(0)
```

图 2.10 calculate 代码图

在该子程序中，首先判断是否已经复制完成，如果已完成，则直接执行 abstractCodeDes 区域代码即可；否则调用 VirtualProtect 后将数据段的机器码复制到 abstractCodeDes 区域，并标记已经复制完成。

3.跟踪与破解程序

表 2.3 跟踪与破解的策略和说明

同组同学：刘鉴之	
跟踪与破解策略	说明
通过二进制文件编辑器打开程序，尝试查找数据段中的用户名和密码。	发现用户名和密码均被加密，只找到乱码，未发现明文的用户名和密码，破译失败
通过 Win32Dasm 得到反汇编代码,观察代码分析加密方案。	发现在比较字符串的代码区域，将输入的字符串与‘A’异或后再比较，最终破解得到用户名为 ljz，密码为 abcdef
通过 VS2019 动态调试，尝试查看运算过程。	由于存在计时反跟踪代码，调试过程中子程序总是直接返回，破解失败。

4.特定指令及程序结构的效果

汇编语言可以实现一条比较语句和多条跳转语句实现分支结构，可以利用跳转语句更好地实现循环和判断语句，可以使用 invoke 和 call 实现模块化结构，更便于理解函数调用的内部处理机制。详细内容可见 1.4.1

2.4 小结

实验 2 中主要进行对汇编程序运行效率的优化。通过优化指令（使用移位指令替换除法指令等）和优化程序结构的方式（调整分支结构等），较为理想地提高了程序的运行效率。同时发现一些指令的优化（如使用 xor 指令置零）并不会给运行效率带来较大的优化，分析发现这些指令的位置并不是决定程序执行效率的关键位置，因此在分析如何提高运行效率时，需要先观察程序得出**决定程序执行效率的关键所在**，而不是盲目优化。

实验 3.1 中主要进行多模块程序设计。通过试错与讨论，得出了 extern 和 public 在多模块间跨模块引用变量与函数的使用方法，掌握了子程序的传参与调用方式，知道了宏的定义及与子程序的不同。相关内容可见 1.5 小结.

实验 4.1 中主要进行中断矢量表的修改与驻留退出。本实验主要通过 DOSBOX 环境下的 TD 调试工具完成，在实验中，我深入理解了中断矢量表，知道了如何使用 INT 21H 来获取和设置中断矢量，掌握了如何进行驻留退出。其中，驻留退出的作用便是在程序关闭之后，将程序的代码保存在内存中，并可以被系统调用。通过 TD 调试工具，我对于寄存器、内存、中断的理解更加明了清晰。

实验 4.2 中主要进行反跟踪与破解。在这个实验中，我掌握了如何利用自我修改返回地址的技术扰乱视线，防止破译，同时，也知道了如何使用 VirtualProtect 函数实现动态修改代码，提高破译难度。在跟踪与破解中，学会如何根据反汇编代码破译加密方案，却对于对方计时反跟踪的技术束手无策，没有找到好的解决方法。

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 对于反汇编等功能的支持性很好，在调试时选择打开的窗口有助于对汇编语句和变量的存储的观察和理解。我观察到的该工具的特点与局限性如下：

1. 反汇编窗口显示符号名与否的差异为：显示符号名时会注明变量名称（直接寻址），不显示符号名时为则均以地址或存储器寻址显示（变址寻址）；

表 3.1 反汇编窗口显示符号名与否的差异

显示符号名	不显示符号名
for (i = 0; i < length; i++) 01001AEC mov dword ptr [i],0 01001AF3 jmp __$EncStackInitStart+32h 01001AF5 mov eax,dword ptr [i] 01001AF8 add eax,1 01001AFB mov dword ptr [i],eax 01001AFE mov eax,dword ptr [i] 01001B01 cmp eax,dword ptr [length] 01001B04 jae __$EncStackInitStart+4Bh	for (i = 0; i < length; i++) 01001AEC mov dword ptr [ebp-8],0 01001AF3 jmp 01001AFE 01001AF5 mov eax,dword ptr [ebp-8] 01001AF8 add eax,1 01001AFB mov dword ptr [ebp-8],eax 01001AFE mov eax,dword ptr [ebp-8] 01001B01 cmp eax,dword ptr [ebp+0Ch] 01001B04 jae 01001B17

2. 监视窗口中，可以通过在代码框中选择变量进行监视，或者直接输入变量名称进行监视；
3. 尝试后发现，内存窗口可以通过**[&变量名]**操作直接定位至某变量的内存单元，若当前所在的代码块内未定义该变量时则无法计算该表达式；并且如下图所示，内存窗口能够记录用户输入的内容（如地址或者取地址表达式）；同时该窗口也有一定不足，未对输入方式进行提示，容易出现无法计算该表达式的情况；

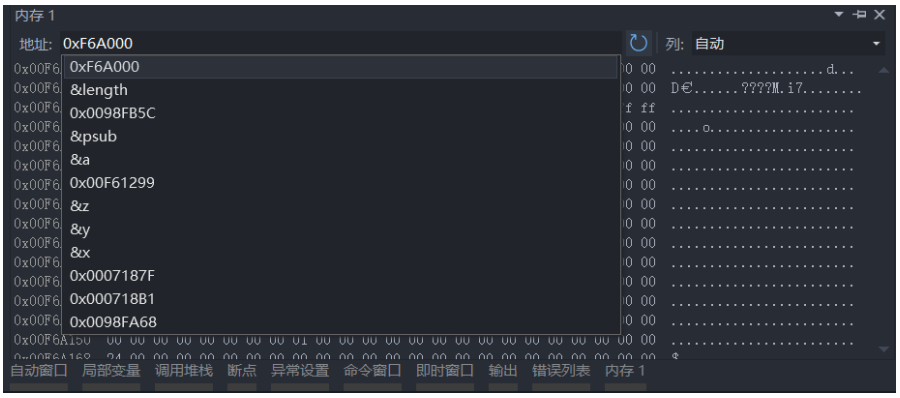


图 3.1 内存窗口示意

4. 有符号和无符号整数的比较

表 3.2 有符号和无符号整数的比较

结论	论证
有符号和无符号整数在内存中**均以补码存储**	定义 int a = 14234678; unsigned int b = 14234678;如下 36 34 d9 00 cc cc cc cc cc cc cc cc 36 34 d9 00 cc cc cc cc
有符号数和无符号数的加减运算与比较均以补码进行	调试后发现两者均以补码进行运算，由反汇编发现，区分方式为有符号和无符号运算时的指令不同

3.2.2 DOSBOX 下的工具包

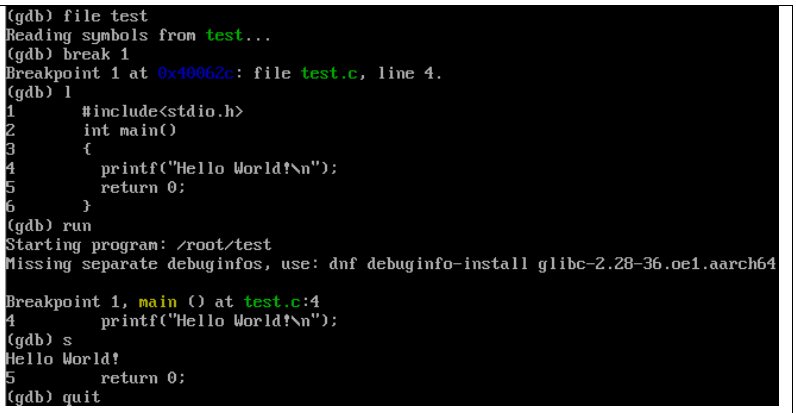
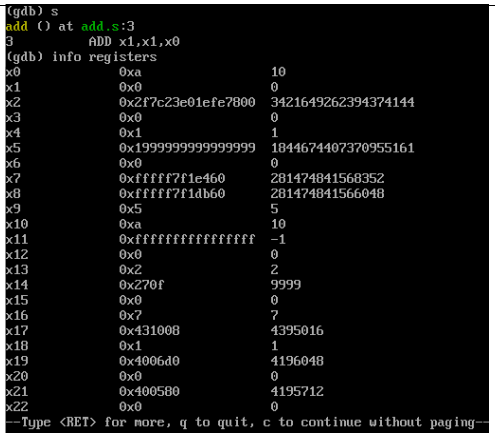
（1） DOSBOX 环境下，使用简单的命令 masm、link 等即可完成对汇编程序的编译；

（2） DOSBOX 环境下的 TD 调试工具也十分便利。在 TD 中，可以使用 Tab 键快速切换区域，使用热键切换菜单，使用 Alt+F10 可以打开区域内菜单，配合鼠标移动可以快捷完成调试、内存查看、寄存器查看与修改、代码修改、查看反汇编代码等功能。

3.2.3 QEMU 下 ARMv8 的工具包

表 3.3 ARMv8 操作示意图

功能	示意图
使用 gcc 编译链接文件，生成可执行文件并运行	[root@localhost ~]# gcc test.c -o test1 [root@localhost ~]# gcc -g test.c -o test [root@localhost ~]# ./test Hello World!

使用 gdb 载入待调试的文件，在第一行设置断点，查看当前待调试的代码，执行当前代码，单步执行，退出	
使用 gdb 在调试中查看寄存器内容	

3.3 小结

通过使用不同的环境下的不同工具，发现它们各有优缺点

表 3.4 不同环境/工具优缺点总结

环境/工具	优点	缺点
VS2022	调试功能强大。在调试时可以很方便地查看寄存器、内存、反汇编代码（包括机器码）。	不支持汇编代码的自动补全与高亮显示。
DOSBOX（含 TD）	操作界面简洁明了，并且使用简单的命令 masm、link 等即可完成对汇编程序的编译。TD 调试工具也十分便利。在 TD 中，可以使用 Tab 键快速切换区域，使用热键切换菜单，使用 Alt+F10 可以打开区域内菜单，配合鼠标移动可以快捷完成调试、内存查看、寄存器查看与修改、代码修改、查看反汇编代码等功能。	操作不够便利，直接修改内存和寄存器可能会对程序正常执行造成影响。
QEMU（含 ARMv8）	ARMv8 基于 Linux 命令行操作，编译、链接和调试的指令较为简单。	无可视化界面，无法直观地查看调试过程与代码。每次查看寄存器或内存时都需要输入指令，操作繁琐。

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
[5]赖晓晨. ARM 虚拟环境相关安装及示例程序. 大连：大连理工大学，2022