

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践

专业班级： 计算机科学与技术 202005 班

学 号： U202090063

姓 名： 董玲晶

指导教师： 班鹏新

实验时段： 2022 年 3 月 7 日~4 月 29 日

实验地点： 东九楼 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：

汇编语言程序设计实验报告

目录

一、程序设计的全程实践	1
1.1 目的与要求	1
1.2 实验内容	1
1.3 内容 1.1 的实验过程	1
1.3.1 设计思想	1
1.3.2 流程图	2
1.3.3 源程序	10
1.3.4 实验记录与分析	15
1.4 内容 1.2 的实验过程	17
1.4.1 实验方法说明	17
1.4.2 实验记录与分析	17
1.5 小结	20
二、利用汇编语言特点的实验	22
2.1 目的与要求	22
2.2 实验内容	22
2.3 实验过程	22
2.3.1 实验方法说明	22
2.3.2 实验记录与分析	23
2.4 小结	27
三、工具环境的体验	28
3.1 目的与要求	28
3.2 实验过程	28
3.2.1 WINDOWS10 下 VS2019 等工具包	28
3.2.2 DOSBOX 下的工具包	29
3.2.3 QEMU 下 ARMv8 的工具包	31
3.3 小结	31
参考文献	32

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

1. 本系统包含两个文件，分别是 main.asm 和 proc.asm。在 proc 中定义了 1 个宏和五个子程序，最后一个子程序中嵌套着前面定义的宏和前四个子程序；而在 main.asm 中定义了一个宏，在 main 中调用宏并可人为控制循环调用 proc.asm 中的最后一个子程序。

总结来说，是一个三层嵌套结构，如图 1-1。而具体每个算法思想在流程图章节里写的很详细，从流程图也能很好地看出算法，详细可看流程图。

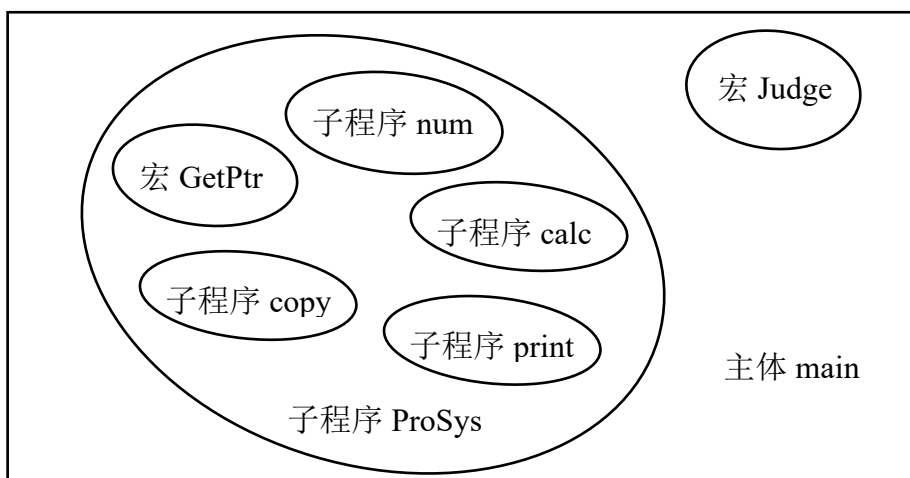


图 1.1 系统结构图

汇编语言程序设计实验报告

2. 功能模块划分和模块说明在每个流程图的上方逐一说明清楚了，这里做一个简单版汇总以便查阅。

(1) `proc.asm` => 先对 N 组采集到的状态信息计算 f，并进行分组复制，然后将 MIDF 存储区的各组数据在屏幕上显示出来

- ① 宏 `GetPtr` => 根据 f 计算出要复制储存的目标区域地址并储存在 EDI 寄存器中
- ② 子程序 `num` => 计算每组数据的流水号（因为 `dup` 无法在结构体中初始化）
- ③ 子程序 `calc` => 根据 a、b、c 的值计算出 f 的值
- ④ 子程序 `copy` => 复制结构体中的流水信息到对应区域
- ⑤ 子程序 `print` => 打印输出流水信息
- ⑥ 子程序 `ProcSys` => 调用以上所有实现功能以供 `main.asm` 文件调用。

(2) `main.asm`

- ① 宏 `Judge` => 判断输入得字符串和某字符串是否相等
- ② `main` 主体 => 实现所有功能模块的调用

3. 因为在流程图里写寄存器的分配和使用总体感觉比较杂乱，在这里逐个对每个文件按寄存器分类进行再次说明以便看流程图时有疑问进行查阅，请在对应的文件名下寻找寄存器查找。

(1) `main.asm`

- ① EAX 用作保存宏 `Judge` 的返回值（1 则说明两个字符串相等而 0 则说明不相等）
- ② EBX 在 `main` 中用作输入用户名密码次数的计数器；
- ③ ECX 在宏 `Judge` 中用作比较字符个数的计数器，以判断字符串相等判断循环是否结束。

(2) `proc.asm`

- ① EDI 在宏 `GetPtr` 中用于保存返回值，即所计算出的分类待储存区域；
- ② ESI 用作寄存源操作数地址；
- ③ EAX 在子程序 `calc` 中用作临时寄存器，储存 f 分步计算的中间值；
在子程序 `copy` 中作数据复制的临时寄存器；
- ④ ECX 作为各个模块的计数器；
在子程序 `num` 中作临时寄存器（因为 `mov` 的两个操作数不能同时在内存里）。

4. 无合并段；存在公共模块 `ProcSys`（调用所有功能子程序实现要求功能以供 `main.asm` 文件调用）和 `fmtStr`（字符串调用 `%s`）。

1.3.2 流程图

1. `proc.asm` => 对实验 1.4 的功能进行分装（先对 N 组采集到的状态信息计算 f，并进行分组复制，然后将 MIDF 存储区的各组数据在屏幕上显示出来）。

(1) 宏模块 `GetPtr` => 根据 f 计算出要复制储存的目标区域地址并储存在 EDI 寄存器中，如图

汇编语言程序设计实验报告

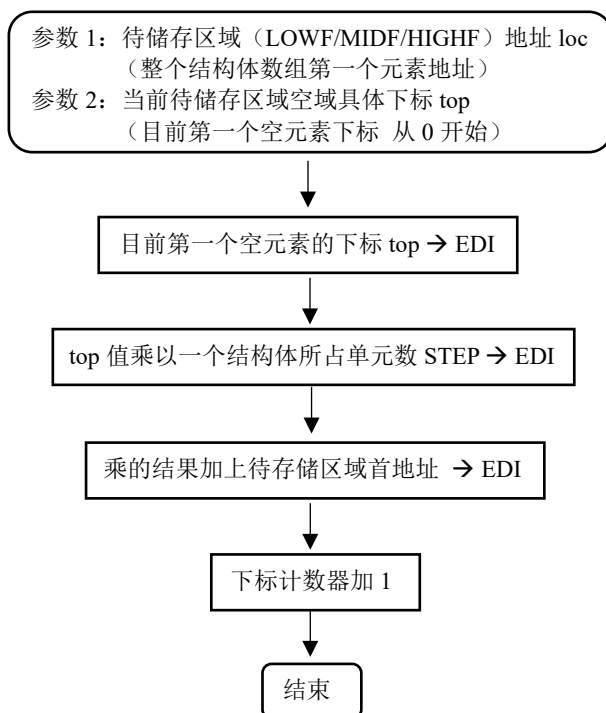


图 1.2 宏模块 GetPtr 流程图

(2) 子程序 num => 计算当前流水号，如图 1.3 所示。

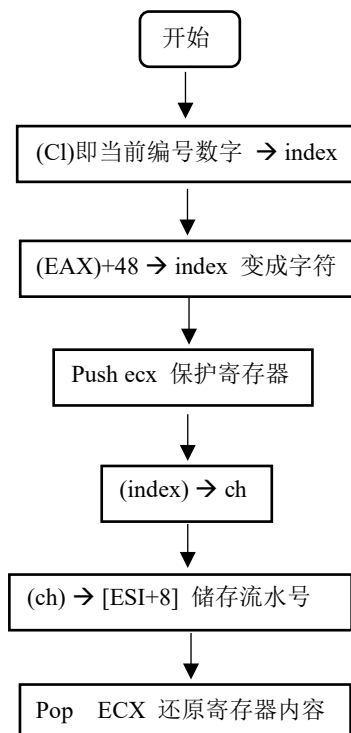


图 1.3 子程序 num 流程图

(3) 子程序 calc => 根据 a、b、c 的值计算出 f 的值，如图 1.4 所示。

汇编语言程序设计实验报告

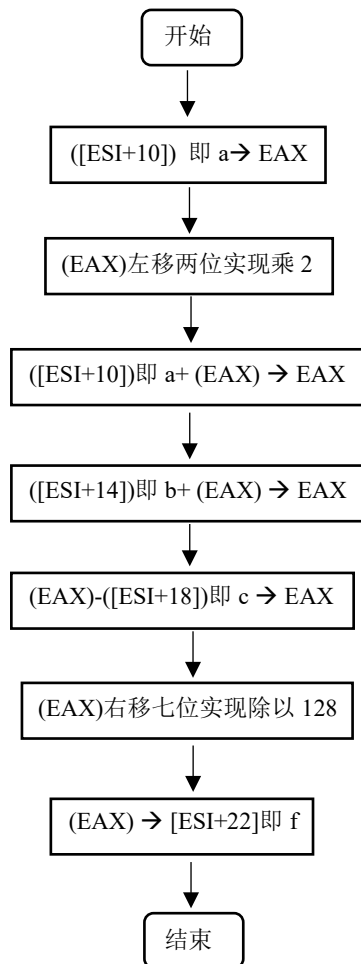


图 1.4 子程序 calc 流程图

(4) 子程序 copy => 复制结构体中的流水信息到对应区域，如图 1.5 所示。

汇编语言程序设计实验报告

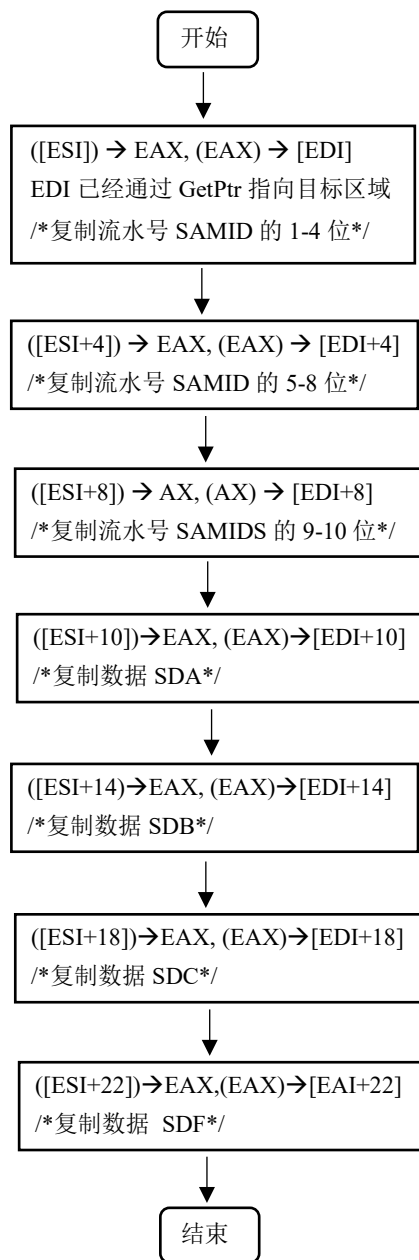


图 1.5 子程序 copy 流程图

(5) 子程序 print => 打印输出流水信息，有三个输入参数：①loc 已储存待打印区域的地址 loc；②len 已储存待打印结构体数组第一个空元素的下标；③hint 打印时的提示标识字符串（lStr 为 LOWF，mStr 为 MIDF，hStr 为 HIGHF）。如图 1.6 所示。

汇编语言程序设计实验报告

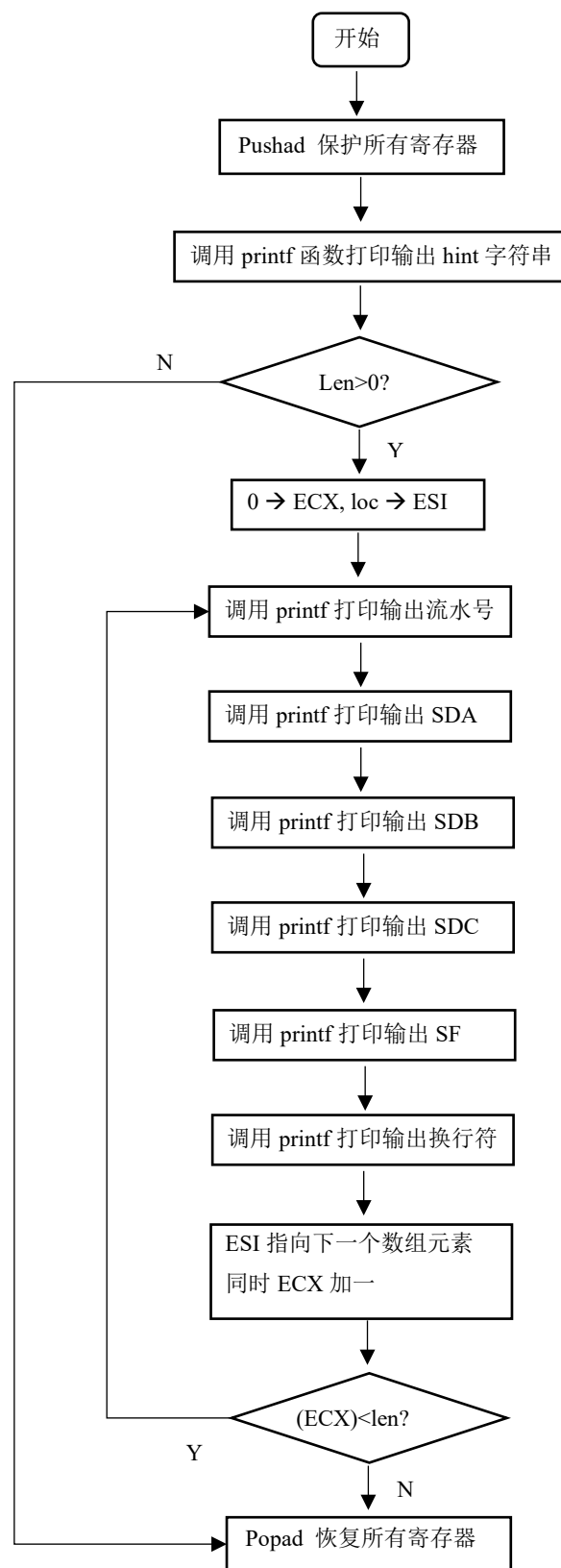


图 1.6 子程序 copy 流程图

汇编语言程序设计实验报告

(6) 子程序 ProcSys => 调用以上所有子程序实现功能以供 main.asm 文件调用, 且为公用, 如图 1.7 所示。

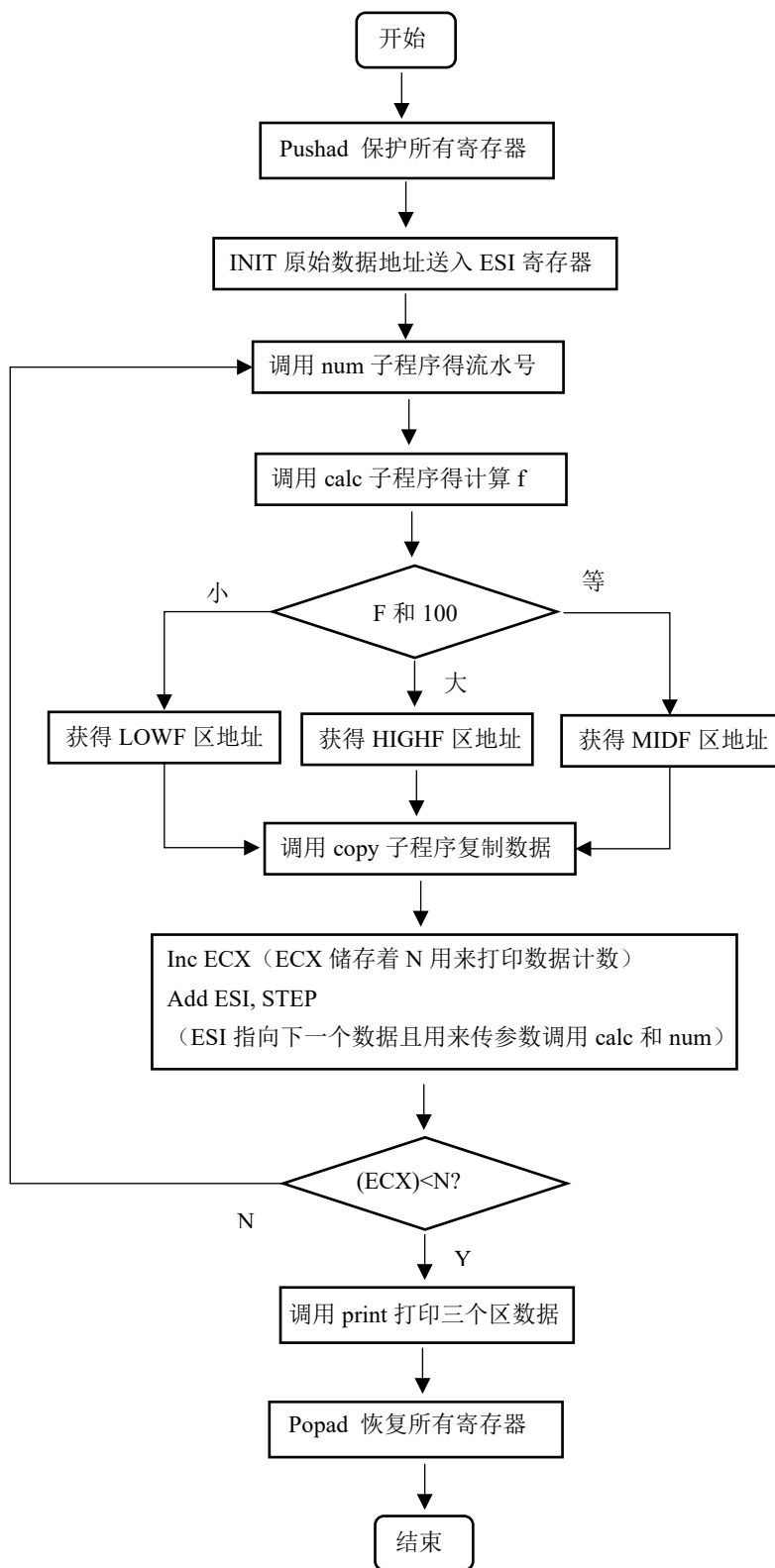


图 1.7 子程序 ProcSys 流程图

汇编语言程序设计实验报告

1. main.asm => 登录功能和调用 ProcSys 子程序实现监测功能

(1)宏 Judge => 判断输入得字符串和某字符串是否相等。三个参数：①src 模板字符串地址；②des 待比较字符串地址；③len 待比较字符串长度，如图 1.8 所示。

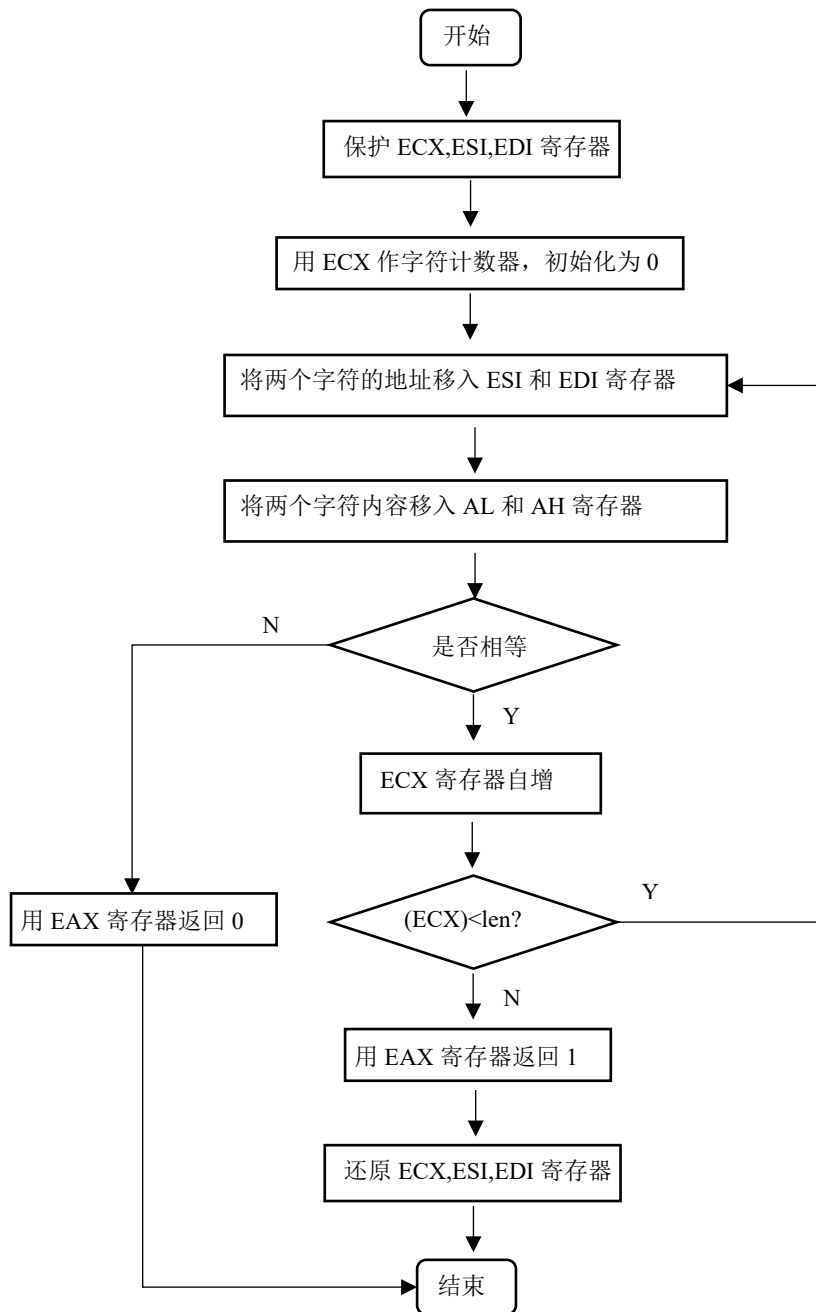


图 1.8 宏模块 Judge 流程图

(2)Main 主体 => 实现所有功能模块的调用，如图 1.9 所示。

汇编语言程序设计实验报告

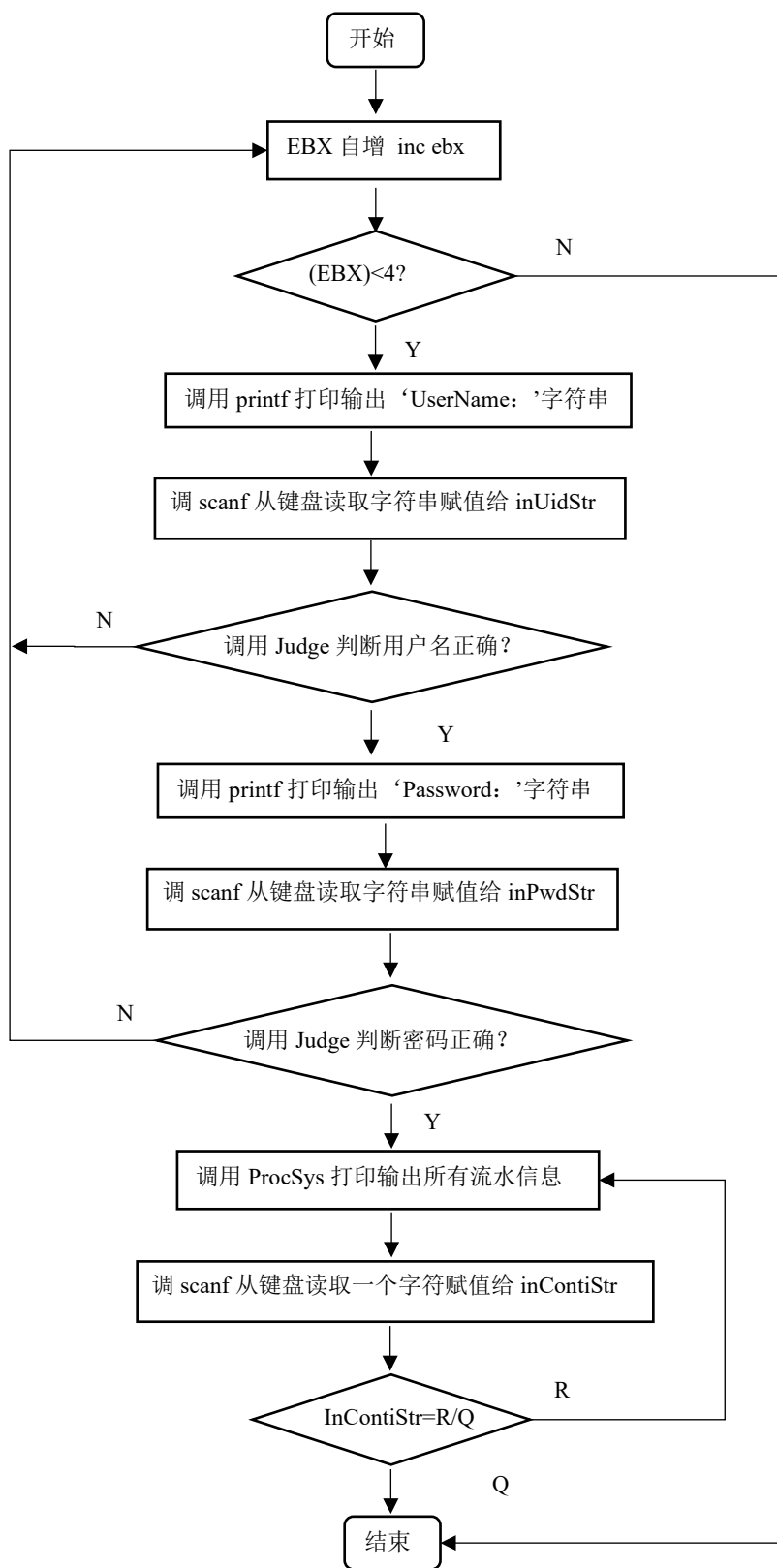


图 1.9 main 主体程序流程图

汇编语言程序设计实验报告

1.3.3 源程序

1. proc.asm

```
.686
.model flat, stdcall
includelib kernel32.lib ; ExitProcess 在 kernel32.lib 中实现
includelib libcmnt.lib
includelib legacy_stdio_definitions.lib
printf PROTO C :VARARG
scanf PROTO C :ptr sbyte, :vararg
ExitProcess PROTO STDCALL :DWORD
GetTickCount PROTO stdcall
public ProcSys
status struct
    SAMID db 9 dup('0'), 0 ;流水号
    SDA dd 0
    SDB dd 0
    SDC dd 0
    SF dd 0 ;处理结果
status ends
.data
    fmtStr db '%s', 0
    fmtInt db '%d', 0
    lStr db 'LOWF:', 0dh, 0ah, 0
    mStr db 'MIDF:', 0dh, 0ah, 0
    hStr db 'HIGHF:', 0dh, 0ah, 0
    line db 0dh, 0ah, 0
    LIMIT equ 100 ;处理结果阈值
    ARRLEN equ 5 ;数组规模
    INIT status <, 2000, 3000, 300,>
        status <, 2000, 3000, 300,>
        status <, 2500, 3000, 300,>
        status <, 2000, 3000, 300,>
        status <, 1000, 3000, 300,>
    LOWF status ARRLEN dup(<>)
    MIDF status ARRLEN dup(<>)
    HIGHF status ARRLEN dup(<>)
    STEP dd type status
    ;三个存储区各自已存个数
    LOWT dd 0
    MIDT dd 0
    HIGHT dd 0
.stack 1024
.code
    GetPtr macro loc, top ;计算出目标地址并存储在 edi 中
        mov edi, top
        imul edi, STEP
        add edi, loc
        inc top
    endm
```

汇编语言程序设计实验报告

num proc C

```
local index: byte
mov     index, cl
add     index, 49
push    ecx
mov     ch, index
mov     [esi+8], ch
pop     ecx
ret
```

num endp

calc proc C ;需要利用 eax

```
mov     eax, [esi+10]
sal     eax, 2 ;优化乘法为移位和加法（其实没有区别）
add     eax, [esi+10]
add     eax, [esi+14]
sub     eax, [esi+18]
add     eax, 100
sar     eax, 7 ;优化除法为移位
mov     [esi+22], eax ;存储 f
ret
```

calc endp

copy proc C

```
;字符串复制
mov     eax, [esi] ;0~4
mov     [edi], eax
mov     eax, [esi+4] ;4~8
mov     [edi+4], eax
mov     ax, [esi+8] ;8~10
mov     [edi+8], ax
;数据复制
mov     eax, [esi+10]
mov     [edi+10], eax
mov     eax, [esi+14]
mov     [edi+14], eax
mov     eax, [esi+18]
mov     [edi+18], eax
mov     eax, [esi+22]
mov     [edi+22], eax
ret
```

copy endp

print proc C loc:dword, len:dword, hint:dword

```
pushad
invoke  printf, offset fmtStr, hint
cmp     len, 0
jle     PRINTEND
mov     ecx, 0
mov     esi, loc
```

汇编语言程序设计实验报告

```
PRINTLOOP:
push ecx
push esi
invoke    printf, offset fmtStr, esi
invoke    printf, offset fmtInt, dword ptr[esi+10]
invoke    printf, offset fmtInt, dword ptr[esi+14]
invoke    printf, offset fmtInt, dword ptr[esi+18]
invoke    printf, offset fmtInt, dword ptr[esi+22]
invoke    printf, offset fmtStr, offset line
pop       esi
pop       ecx
inc       ecx
add       esi, STEP
cmp       ecx, len
jl        PRINTLOOP
PRINTEND:
popad
ret
print endp
```

```
ProcSys proc C
pushad
mov       ecx, 0
lea       esi, INIT
LOOPSTART:
invoke    num
invoke    calc
cmp       eax, LIMIT
jl        LESS
je        EQUAL
jg        GREAT
;根据 f 计算目标地址 edi
LESS:
GetPtr    offset LOWF, LOWT
jmp       COPYSTART
EQUAL:
GetPtr    offset MIDF, MIDT
jmp       COPYSTART
GREAT:
GetPtr    offset HIGHF, HIGHT
COPYSTART:
invoke    copy
inc       ecx
add       esi, STEP
cmp       ecx, ARRLen
jl        LOOPSTART
invoke    print, offset LOWF, LOWT, offset lStr
invoke    print, offset MIDF, MIDT, offset mStr
invoke    print, offset HIGHF, HIGHT, offset hStr
popad
ret
```

汇编语言程序设计实验报告

```
ProcSys endp  
END
```

2. main.asm

```
.686  
.model flat, stdcall  
includelib kernel32.lib ; ExitProcess 在 kernel32.lib 中实现  
includelib libcmnt.lib  
includelib legacy_stdio_definitions.lib  
printf PROTO C :VARARG  
scanf PROTO C :ptr sbype, :vararg  
ExitProcess PROTO STDCALL :DWORD  
GetTickCount PROTO STDCALL  
ProcSys PROTO C  
public fmtStr  
.data  
;输入上限为 20  
num equ 1  
fmtStr db '%s', 0  
uid db 'admin', 0  
uidLen equ $-uid-1  
pwd db '123456', 0  
pwdLen equ $-pwd-1  
rStr db 'R', 0  
qStr db 'Q', 0  
scsStr db 'Successfully logged in!', 0dh, 0ah, 0  
flStr db 'Wrong username or password', 0dh, 0ah, 0  
htUidStr db 'UserName: ', 0  
htPwdStr db 'Password: ', 0  
inUidStr db 21 dup(0)  
inPwdStr db 21 dup(0)  
inContiStr db 2 dup(0)  
.stack 1024  
.code  
;需要调用 eax 作为返回值, 请在调用前保护 eax  
Judge macro src, des, len  
    local JSTART  
    local JFAIL  
    local JEND  
    push ecx  
    push esi  
    push edi  
    mov     esi, src  
    mov     edi, des  
    mov     ecx, 0  
    JSTART:  
    mov     ah, [esi+ecx]
```

汇编语言程序设计实验报告

```
mov     al, [edi+ecx]
cmp     ah, al
jne     JFAIL
inc     ecx
cmp     ecx, len
jle     JSTART
mov     eax, 1
jmp     JEND
JFAIL:
mov     eax, 0
JEND:
pop     edi
pop     esi
pop     ecx
endm
```

```
main proc C
    mov ebx, 0
    RESTART:
    inc ebx
    cmp ebx, 4
    je    LOGINFAIL
    invoke printf, offset fmtStr, offset htUidStr
    invoke scanf, offset fmtStr, offset inUidStr
    Judgeoffset uid, offset inUidStr, uidLen
    cmp     eax, 0
    je      RESTART
    invoke  printf, offset fmtStr, offset htPwdStr
    invoke  scanf, offset fmtStr, offset inPwdStr
    Judgeoffset pwd, offset inPwdStr, pwdLen
    cmp     eax, 0
    je      RESTART
    jmp LOGINSUC
    LOGINFAIL:
    invoke  printf, offset fmtStr, offset flStr
    jmp     EXIT
    LOGINSUC:
    invoke ProcSys
    JCONT:
    invoke scanf, offset fmtStr, offset inContiStr
    Judge offset qStr, offset inContiStr, num
    cmp eax, 1
    je EXIT
    Judge offset rStr, offset inContiStr, num
    cmp eax, 0
    je EXIT
    jmp LOGINSUC
    EXIT:
```


汇编语言程序设计实验报告

```
invoke ExitProcess, 0
main endp
END
```

1.3.4 实验记录与分析

1. 实验环境条件

INTEL 处理器 2GHz，8G 内存；WINDOWS11 下 VS2019 社区版。

2. 汇编、链接中的情况

(1) 因为调用在另一个文件里的子程序没有进行声明，如图 1.10 所示；在一个文件里使用的另一个文件里定义的子程序需要在此文件里先使用 `public` 进行声明，再进行调用。

代码	说明	项目	文件
A2006	undefined symbol : ProcSys	3-1	main.asm

图 1.10 调用子程序没有进行声明而导致的错误

(2) 因为先使用后定义了宏出错，如图 1.11 所示；宏需要先进行声明再进行使用，而子程序声明和调用先定义后调用和先调用后定义皆可。

代码	说明	项目	文件
A2008	syntax error : offset	3-1	proc.asm
A2008	syntax error : offset	3-1	proc.asm
A2008	syntax error : offset	3-1	proc.asm

图 1.11 使用宏前没有先进行定义导致的错误

(3) 宏中的标号和变量没有使用 `LOCAL` 语句而出现了重复定义的错误，如图 1.12 所示；

代码	说明	项目	文件
A2005	symbol redefinition : JSTART	3-1	main.asm
A2005	symbol redefinition : JFAIL	3-1	main.asm
A2005	symbol redefinition : JEND	3-1	main.asm
A2005	symbol redefinition : JSTART	3-1	main.asm
A2005	symbol redefinition : JFAIL	3-1	main.asm

图 1.12 重复使用的代码标号没有进行 `LOCAL` 声明产生的错误

3. 程序基本功能的验证情况

(1) 三次登录机会，如图 1.13 所示。输入用户名和密码一共有三次输入机会，在输入三次之后提示“用户名或者密码输入错误”。

```
UserName: U202090063
Password: 1234
UserName: U202090063
Password: 12345
UserName: U202090063
Password: 123
Wrong username or password
```

图 1.13 用户名或密码输入三次错误后结果

汇编语言程序设计实验报告

(2) 若在三次机会之内输入用户名和密码成功，登录成功后打印流水信息，已对流水信息进行分类并分为 LOWF、MIDF、HIGHF 三个区域，如图 1.14 所示

```
UserName: U202090063
Password: 123456
LOWF:
000000005/1000/3000/300/60
MIDF:
000000001/2000/3000/300/100
000000002/2000/3000/300/100
000000004/2000/3000/300/100
HIGHF:
000000003/2500/3000/300/119
```

图 1.14 所有数据计算、存储、打印结果

(3) 输入 R 继续打印输出，如图 1.15 所示。

```
UserName: U202090063
Password: 123456
LOWF:
000000005/1000/3000/300/60
MIDF:
000000001/2000/3000/300/100
000000002/2000/3000/300/100
000000004/2000/3000/300/100
HIGHF:
000000003/2500/3000/300/119
R
LOWF:
000000005/1000/3000/300/60
000000005/1000/3000/300/60
MIDF:
000000001/2000/3000/300/100
000000002/2000/3000/300/100
000000004/2000/3000/300/100
000000001/2000/3000/300/100
000000002/2000/3000/300/100
000000004/2000/3000/300/100
HIGHF:
000000004/2000/3000/300/100
000000003/2500/3000/300/119
R
LOWF:
000000005/1000/3000/300/60
000000005/1000/3000/300/60
000000005/1000/3000/300/60
```

图 1.15 输入 R 的运行结果

(4) 输入 Q 退出本程序，如图 1.16 所示。

汇编语言程序设计实验报告

```
HIGHF:
000000004/2000/3000/300/100
000000003/2500/3000/300/119
R
LOWF:
000000005/1000/3000/300/60
000000005/1000/3000/300/60
000000005/1000/3000/300/60
MIDF:
000000001/2000/3000/300/100
000000002/2000/3000/300/100
000000004/2000/3000/300/100
000000001/2000/3000/300/100
000000002/2000/3000/300/100
000000004/2000/3000/300/100
000000001/2000/3000/300/100
000000003/2500/3000/300/119
000000004/2000/3000/300/100
HIGHF:
000000004/2000/3000/300/100
000000001/2000/3000/300/100
000000003/2500/3000/300/119
Q
```

图 1.16 输入 Q 的运行结果

4. 使用调试工具观察、探究代码的情况

(1) 在写按 R 重复执行的功能的时候，一直觉得自己代码没问题但是每次按 R 就直接退出程序。后来打了断点，设置了单步调试，打开了寄存器窗口看了一下 EAX 的值，发现此时调用宏寄存器返回的值有问题，和预期不符。最后发现再将断点设置在前几个语句，发现是上面对 scanf 的使用出问题了。解决了之后觉得虽然是个很愚蠢的小 bug，但是真的困扰了我很久，通过调试和监视寄存器或内存窗口，可以很好地和操作系统内部进行交流，及时发现错误。

(2) 一开始写子程序的时候，因为没有保护 ESI 寄存器让寄存器内容入栈和出栈，导致调用完回到原程序后公用的寄存器里的数据丢失，虽然汇编没问题，但是最后输出的结果不对，而且循环了几次程序就自己退出。这个问题困扰了我将近一周，后来是请教了实践课的老师，老师让我认真一点调试，多看看寄存器的值，再到监视窗口添加观察变量观察其值的变化，最后找到了问题的所在。

1.4 内容 1.2 的实验过程

1.4.1 实验方法说明、记录与分析

1. 指令优化对程序的影响

① 将 imul 和 idiv 指令换成移位和加法。乘法换成右移和加法组合，除法换成左移指令，可以大大加速如图 1.17 所示；

汇编语言程序设计实验报告

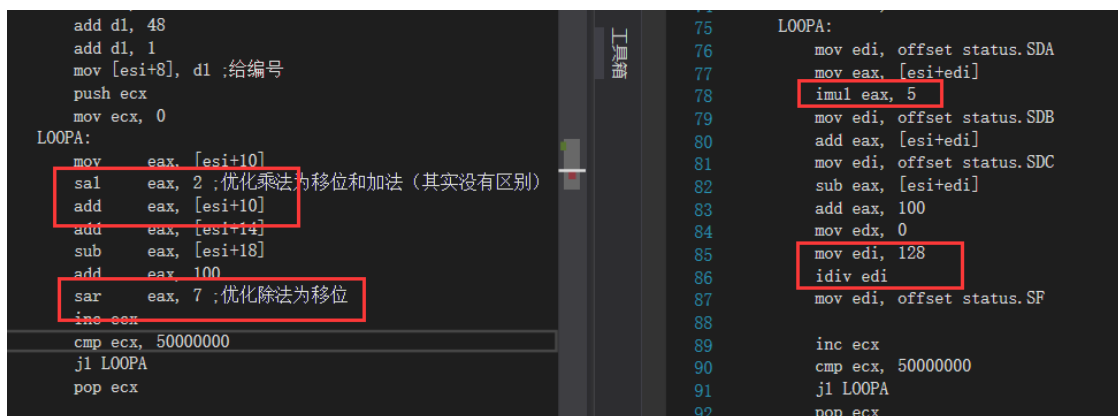


图 1.17 将乘除法指令转换成移位指令

② 减少不必要的 mov 操作，把所有 mov 中 offset 相关的取地址指令换成 lea 指令，如图 1.18 所示：

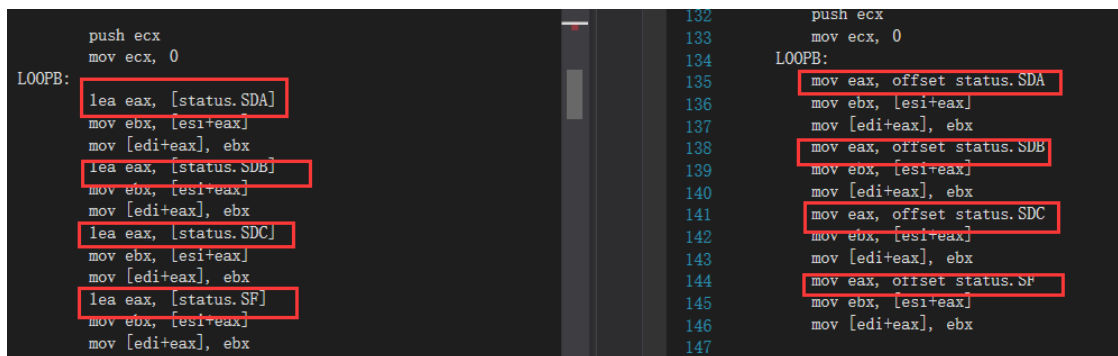


图 1.18 将 mov 指令换成 lea 指令

③ 在计数时使用 inc 指令而不使用“add 寄存器，1”指令，可以加快运行速度，如图 1.19 所示。

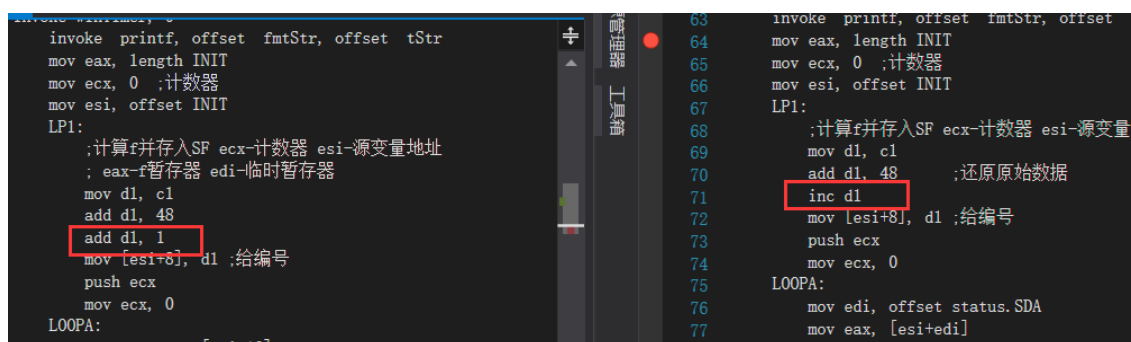


图 1.19 将 add 指令替换成 inc 指令

④ 减少指令执行的次数：将拷贝中的一个字节一个字节复制改为 4 个字节复制。

⑤ 去掉比较、跳转指令即将循环结构变成不循环结构：将拷贝中的 cmp 和 jl 指令去掉，由循环改为不循环拷贝。

说明：总体进行优化后时间是优化前的 50%左右，优化率达到了 52.3%，如图 1.20 所示。

汇编语言程序设计实验报告



图 1.20 改进前后运行时间对比

2. 约束条件、算法与程序结构的影响

(1) 约束条件

① 小小的约束条件可能会对程序带来很大的改变。比如实验一中无符号和有符号数，对于 `sum(a, 0x90000000)` 语句在 `unsigned` 和 `int` 两个不同条件下的结构也不相同。当 `length` 为无符号数时，`0x90000000` 这个数太大，超出了数组 `a` 的范围，运行会引发读取访问权限冲突异常，如图 1.21 所示。而当将 `unsigned length` 改为 `int length`，`length` 是有符号数，执行上述语句，`0x90000000` 是负数所以没有执行循环体，最终的结果是 0。因此综上所述，有符号数和无符号数的约束条件是会对程序结果造成不同的。

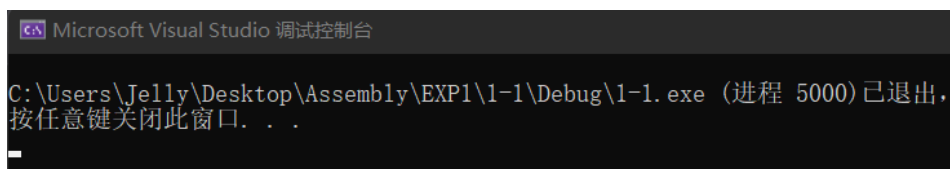


图 1.21 条件为无符号情况下实验一程序运行结果



图 1.22 条件为有符号情况下实验一程序运行结果

(2) 程序结构

① 经常使用、功能明确且比较复杂的代码段写成子程序的形式可以简化主体和方便反复调用，如图 1.23 所示；写成子程序后代码量骤降，而且原来的程序里代码全是重复的片段。

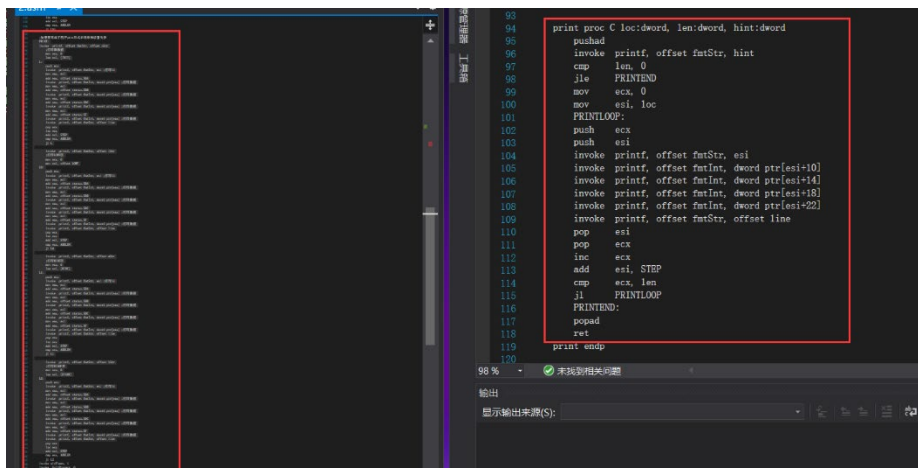


图 1.23 改进程序结构前后代码量对比

汇编语言程序设计实验报告

② 经常使用、具有独立功能、而且语句序列比较简单的语句写成宏指令速度比子程序更快(因为避免了使用堆栈保护恢复现场等指令)且节约空间,如图 1.24 所示;进行了宏定义后代码量减少了而且分装成了一个指令可以直接像系统自带的指令一样使用,十分的方便

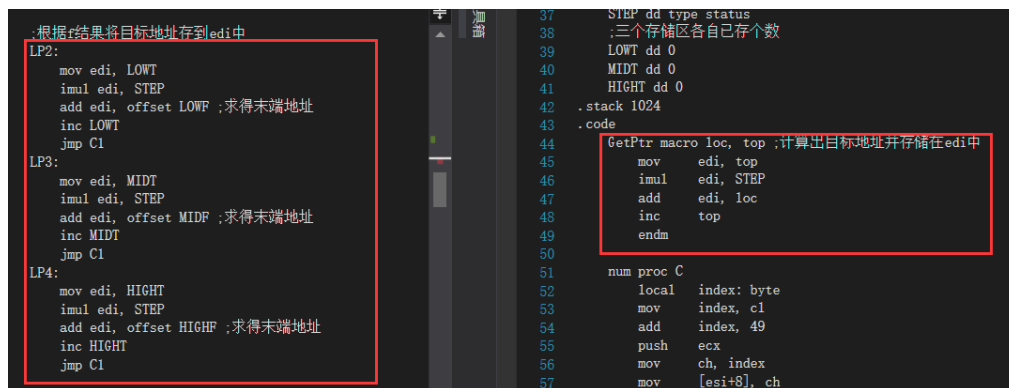


图 1.24 改进程序结构前后代码量对比

3. 编程环境的影响

(1) x64 程序与 32 位程序的不同之处

- ① x64 中用的是 64 位寄存器而且一共有 16 个 64 位的通用寄存器。
- ② x64 中内存地址是 64 位的。
- ③ x64 不支持许多在 32 位中支持的伪指令用法。如 x64 不支持 invoke 伪指令调用函数和传递函数的参数、不支持“.stack”的方法来定义堆栈段,也不支持处理器选择、储存模型说明等伪指令。
- ④ 在 C 语言函数、Windows API 函数调用方面,参数传递时要遵循约定,将参数放入指定的寄存器中或精确控制放入中的位置,32 位程序中则是将参数从右至左压入堆栈。

1.5 小结

1. 本次实验我感觉最大的收获在于充分掌握了子程序和宏的用法,还有就是结构体的使用。一开始写的时候编译老出问题,一步步解决了之后又是结果不正确,让人心态很崩,好在最后都完美解决了。接下来浅写一下我的教训吧。

2. 一定要在写代码之前就把寄存器的功能大致划分好,寄存器的个数实在有限,代码多起来之后会感觉很不够用,后续写或者加功能的时候要列一张寄存器功能表,方便记录添加的寄存器用处,在后面写的时候不会乱。

3. 调用子程序的时候一定一定一定要记得保护寄存器,以防止运算的结果或者中间值遭到破坏。用寄存器传参数的时候,一定要先自己捋清楚寄存器内容的变化才能写得对,对于地址的变化要自己推算清楚,尤其要注意双字、字、字节操作的区别。

4. 在模块化编程的时候,各模块同性质的段名最好要相同,不然连接时会报错,同时要在确定每个模块功能无误后再连接验证,可以节约时间。

汇 编 语 言 程 序 设 计 实 验 报 告

5. 在实验一中我通过单步调试观察程序运行情况，以及通过监视窗口观察变量，查看内存观察数据在内存中的存储，查看寄存器等，我掌握了数据在计算机内的存储、表现形式。

6. 通过任务 3.2 实现了 C 和汇编的混合编程，学习了不同的模块之间如何连接，能够调用不同模块的函数与变量，知道了原来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，在进行实验时也学会了 VS2019 的使用，同时也发现相同的程序，汇编的执行效率明显要更优秀一些。

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) 中断处理程序的流程图,如图 2.1 所示。

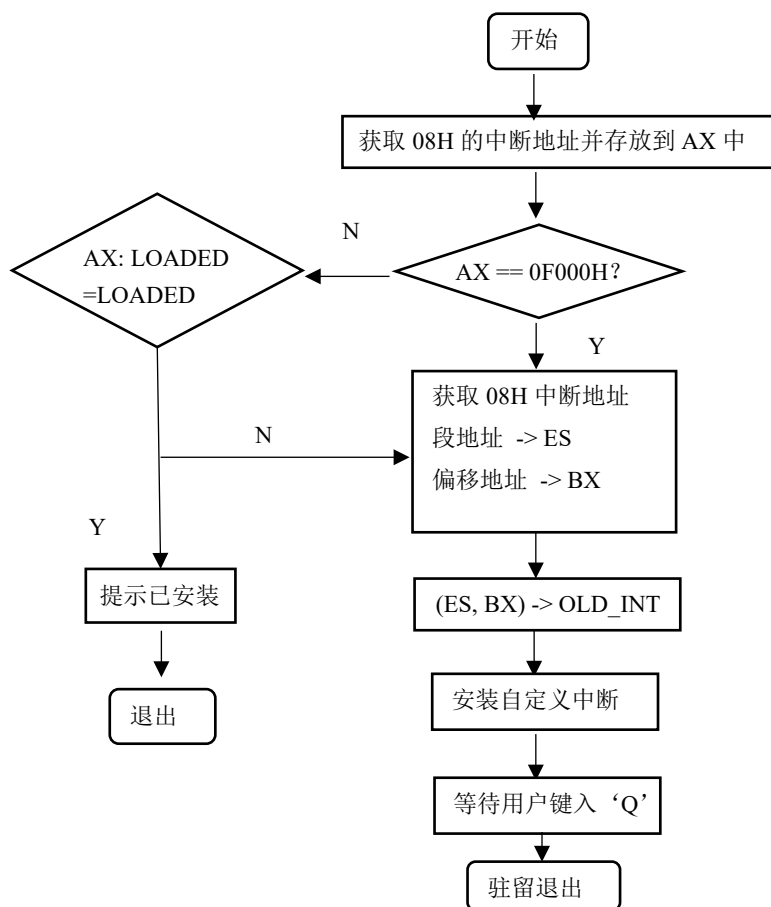


图 2.1 中断处理程序流程图

汇编语言程序设计实验报告

(2) 中断程序设计思想

扩充 8 号中断处理程序的功能。每次中断时，时钟显示程序首先调用原 8 号中断处理程序的功能，然后对中断次数计数：当计数值没有从 18 减到 0 时，直接中断返回；当计数值从 18 减到 0 时，则重新给计数器赋初值。获取到当前的时间之后，转换成对应的 ASCII 码串，利用“INT 10H”指令将时间显示在屏幕上。进入该程序时首先检查当前是否按安装该程序且驻留内存，若没有则安装并驻留内存，否则提示已经安装该程序，避免重复安装。

2.反跟踪程序的设计思想与实验方法

(1) 对用户密码进行加密，采用密文的方式存放在数据段，加密方式采用 $X \oplus 1$ 或者 $X+1h$ 或者 $X-29h$ 来加密，在最后交付出去的执行程序中看不到密码的明文，但用户名仍然是明文。

(2) 采用计时反跟踪方式来进行反跟踪

(3) 代码中穿插数据定义或无关代码

(4) 在二进制文件编译工具里看到的是加密了的密码，并不知道存在哪里，所以避免了被发现加密算法而破解。

3.指令优化及程序结构的实验方法

(1) 经常使用的具有独立功能的程序段而且语句序列比较简单的时候可以写成宏指令；

(2) 宏定义的宏体中的一些变量或者标号最好加上 local 写在宏体的第一句，这样可以避免多次调用和宏扩展后引起重复定义的错误；

(3) 如果程序中连续地重复完全相同的语句，可以使用重复汇编的方式来简化；

(4) 程序中分支太多的情况下可以使用条件汇编伪指令来节约储存空间，同时也可以去除许多无用的、庞杂的代码；

(5) 对于经常使用的、功能明确且比较复杂的代码段，可以写成子程序的形式来简化主体结构 and 方便反复调用；

(6) 在子程序中要注意保护一些用到的寄存器，防止破坏原有数据，保存的方法一般是在子程序的开始将寄存器的内容入栈，返回指令前出栈逐个恢复；

(7) 宏是先定义后调用而子程序皆可；

(8) 将乘除法换成移位和加减法；

(9) 能用 lea 就不用 mov 和 offset；

(10) 使用 inc 指令而不用 add 来加一计数；

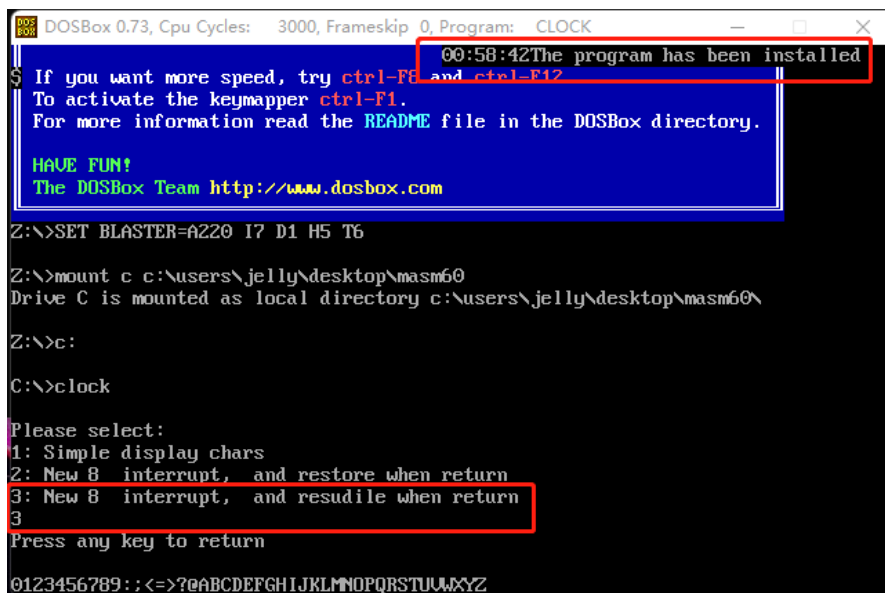
2.3.2 实验记录与分析

1.中断处理程序的特别之处

汇编语言程序设计实验报告

```
RESIDULE_INTR8 PROC
;将新的中断处理程序驻留内存
MOV DX, OFFSET DELAY+15
MOV CL, 4
SHR DX, CL
ADD DX, 10H
ADD DX, 70H
MOV AL, 0
MOV AH, 31H
INT 21H
RESIDULE_INTR8 ENDP
```

图 2.2 将新的中断处理程序驻留内存的代码



```
DOSBox 0.73, Cpu Cycles: 3000, Frameskip 0, Program: CLOCK
00:58:42The program has been installed
$ If you want more speed, try ctrl-F1 and ctrl-F12
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

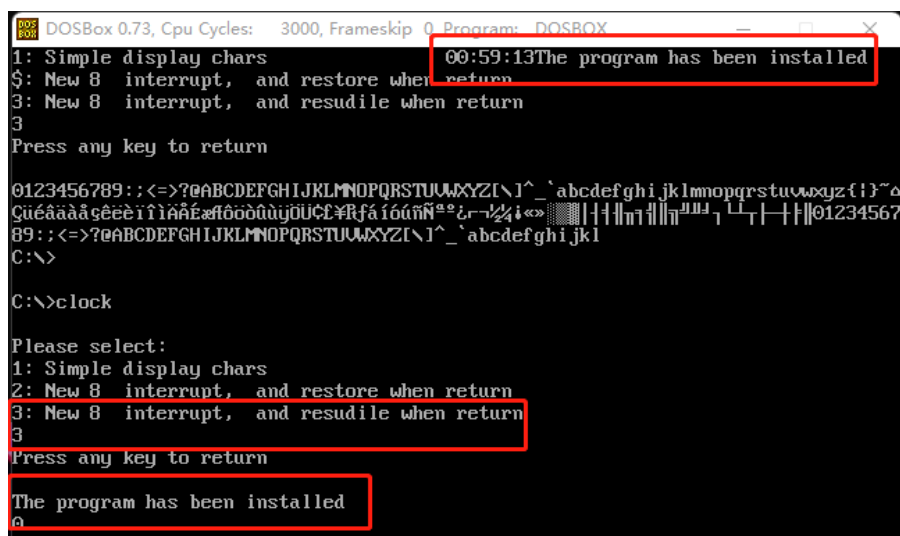
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c c:\users\jelly\desktop\masm60
Drive C is mounted as local directory c:\users\jelly\desktop\masm60\

Z:\>c:
C:\>clock

Please select:
1: Simple display chars
2: New 8 interrupt, and restore when return
3: New 8 interrupt, and resudile when return
3
Press any key to return
0123456789:;<=>?@ABCDEFGHIJKLMNQRSTUUVWXYZ
```

图 2.3 DOSBOX 下第一次安装中断处理程序显示结果



```
DOSBox 0.73, Cpu Cycles: 3000, Frameskip 0, Program: DOSBOX
00:59:13The program has been installed
$ New 8 interrupt, and restore when return
3: New 8 interrupt, and resudile when return
3
Press any key to return
0123456789:;<=>?@ABCDEFGHIJKLMNQRSTUUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
C:\>

C:\>clock

Please select:
1: Simple display chars
2: New 8 interrupt, and restore when return
3: New 8 interrupt, and resudile when return
3
Press any key to return
The program has been installed
0
```

图 2.4 DOSBOX 下重复安装中断处理程序显示结果

汇编语言程序设计实验报告

2.反跟踪效果的验证

(1) 采用计时反跟踪方式来进行反跟踪，在跳转到 f 计算函数前获取一下当前时间，执行 call 指令，之后再次获取当前时间，计算两个时间的差值，大于限制时间说明被跟踪调试了，直接跳转到 EXIT 结束程序。

```
CLI                                ;计时反跟踪开始
MOV AH, 2CH
INT 21H
PUSH DX
MOV BL, in_name+2                 ;看输入是不是回车
CMP BL, 0dh
JE BEGIN

NEWLINE

MOV AH, 2CH                       ;获取第二次秒与百分秒
INT 21H
STI
CMP DX, [ESP]
POP DX
JNZ JIBSHU                       ;如果计时相同，通过本次计时反跟踪
```

图 2.5 计时反跟踪代码



```
Microsoft Visual Studio 调试控制台
UserName: U202090063
Password: 123456
C:\Users\Jelly\Desktop\Assembly\EXP3\3-1\Debug\3-1.exe (
9296)已退出，代码为 -1073741819。
按任意键关闭此窗口...
```

图 2.6 单步调试下计时反跟踪成功，退出程序

(2) 利用二进制编译工具 UltraEdit，由于密码存储区域未知，所以起到了保护作用，如图 2.7 所示。

```
00000660h: 24 00 00 53 48 4F 50 00 00 00 00 00 00 00 00 00 ; $..SHOP.....
00000670h: 00 00 00 00 00 00 00 00 00 00 00 00 03 00 70 65 ; .....pe
00000680h: 6E 00 00 00 00 00 00 00 0A 1A 00 38 00 42 27 19 ; n.....8.B'.
00000690h: 00 00 00 62 6F 6F 6B 00 00 00 00 00 00 09 35 00 ; ...book.....5.
000006a0h: 1E 00 19 00 05 00 00 00 74 6D 70 00 00 00 00 00 ; .....tmp.....
000006b0h: 00 00 08 35 00 1E 00 19 00 05 00 00 00 00 00 00 ; ...5.....
000006c0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
```

图 2.7 利用制编译工具反跟踪

(3) 在原来的代码中添加冗余的代码，如图 2.8 所示。以及在 JMP 或 CALL 指令之后定义一些初始值随意的变量存储区，由于在程序正常控制下是不会执行到这段存储区的，所以反汇编程序把这段存储区的数据反汇编成指令语句是没有任何意义的，达到了扰乱视线的目的。

汇编语言程序设计实验报告

```
redundancy proc;添加多余代码
    pop ebx
    mov eax, [ebx]
    mov cx, 4
    mul cx
    add ebx, eax
    push ebx
    ret
redundancy endp
```

图 2.8 在代码中加入冗余代码

3.跟踪与破解程序

(1) 用二进制文件编辑工具 UltraEdit 直接观察执行文件中的信息，如图 2.9 所示。

```
00000390h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
000003a0h: 2C C8 00 00 00 00 4E 49 4C 59 00 00 00 00 ; ,?...NILY.....
000003b0h: 15 12 12 15 00 00 00 50 45 4E 00 00 00 00 ; .....PEN.....
000003c0h: 00 0A 61 00 38 00 46 00 19 00 00 00 42 4F 4B ; ..a.8.F....BOOK
000003d0h: 00 00 00 00 00 00 09 4E 00 1E 00 20 4E 00 00 ; .....N... N...
000003e0h: 00 52 55 4C 45 52 00 00 00 00 08 4D 00 14 00 ; .RULER.....M...
000003f0h: 1E 00 02 00 00 00 54 65 6D 70 56 61 6C 75 65 ; .....TempValue.
00000400h: 08 0E 00 14 00 1E 00 02 00 00 00 54 65 6D 70 56 ; .....TempV
```

图 2.9 利用 UltraEdit 观察执行文件中的信息

(2) 利用静态反汇编工具 wdasm8.93 反汇编工具将执行程序反汇编成源程序，观察代码，查找 add、shl 等语句，最终计算相关的代码，并且通过分析得知 a=5, b=1, c=-1, d=100。

```

:004084F6 55          push ebp
:004084F7 8BEC       mov ebp, esp
:004084F9 83C4FC     add esp, FFFFFFFC
:004084FC C745FC64000000 mov [ebp-04], 00000064
:00408503 E848FFFFFF call 00408450
:00408508 0500000000 add eax, 58000000
:0040850D 00000000   BYTE 3 DUP(0)

:00408510 7931       jns 00408543
:00408512 0000       add byte ptr [eax], al
:00408514 320F       xor cl, byte ptr [edi]
:00408516 0000       add byte ptr [eax], al
:00408518 83000000   add dword ptr [eax], 00000000
:0040851B 008B7508B46 add byte ptr [ebx+468B0875], cl
:00408521 09998BC8D1E2 or dword ptr [ecx+E2D1C88B], ebx
:00408527 D1E0       shl eax, 1
:00408529 7301       jnb 0040852C
:0040852B 42         inc edx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408529(C)
|
:0040852C D1E2       shl edx, 1
:0040852E D1E0       shl eax, 1
:00408530 7301       jnb 00408533
:00408532 42         inc edx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408530(C)
|
:00408533 83F900     cmp ecx, 00000000
:00408536 7F03       jg 00408538
:00408538 83C2FF     add edx, FFFFFFFF

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408536(C)
|
:0040853B 03C1       add eax, ecx
:0040853D 7301       jnb 00408543
:0040853F 42         inc edx
```

图 2.10 利用 wdasm 8.93 反汇编工具找到加密信息 (1)

汇编语言程序设计实验报告

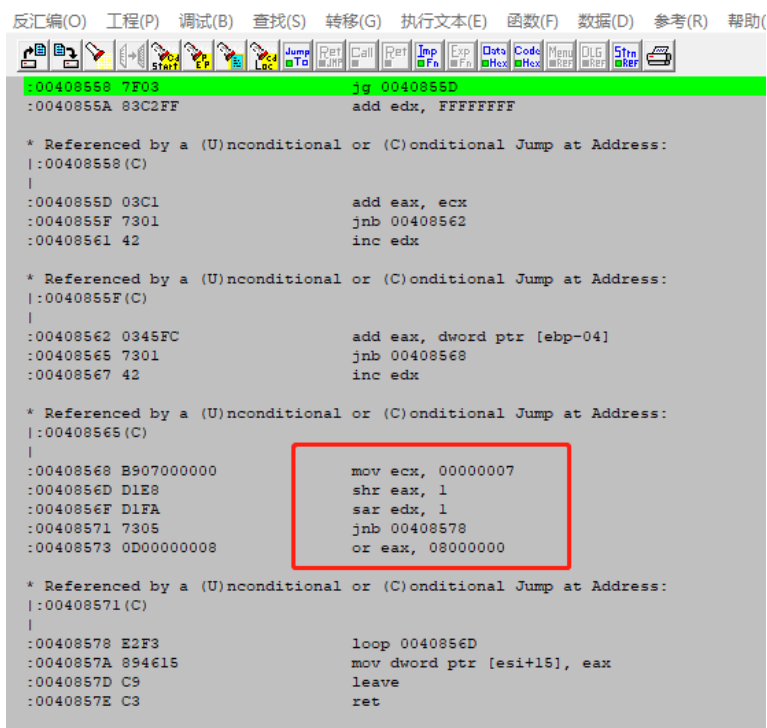


图 2.11 利用 wdasm 8.93 反汇编工具找到加密信息（2）

4.特定指令及程序结构的效果

可以直接看前面一章的图，在前面也已经讲的很清晰且完整，不反复赘述。

2.4 小结

通过实验 4-1，我加强了对中断相关知识的掌握，深入理解中断矢量表的概念，熟悉了 I/O 访问和 BIOS 功能的调用方法，掌握了实方式下中断处理程序的编制和调式方法，也进一步熟悉了内存的一些基本操作技术。

通过对原来的程序进行反跟踪加密，我学了一些反跟踪技术，如倒计时、动态修改代码等有趣的保护措施。在对同学的 exe 文件破解中，我学会了 W32Dasm 工具的运用，并发现原来还能通过这样的工具和手段对一个 exe 文件进行破解和复原代码操作，拓宽了我的认知。

同时，在实验五中我第一次认识和使用 GDB 工具包，尝试了每一个指令，这让我感到很新鲜，也感到很便捷。而且值得一提的是在后面开设的另一门课（计算机系统基础）课后，也用到了 GDB 工具包来反汇编和跟踪、调试汇编程序，从而破解汇编程序。所以我非常感激这门课以及实验，让我更靠近计算机底层，更加深入了解计算机的运行机制，学会了许多面向底层的代码编写方法和工具的使用方法。

三、工具环境的体验

3.1 目的与要求

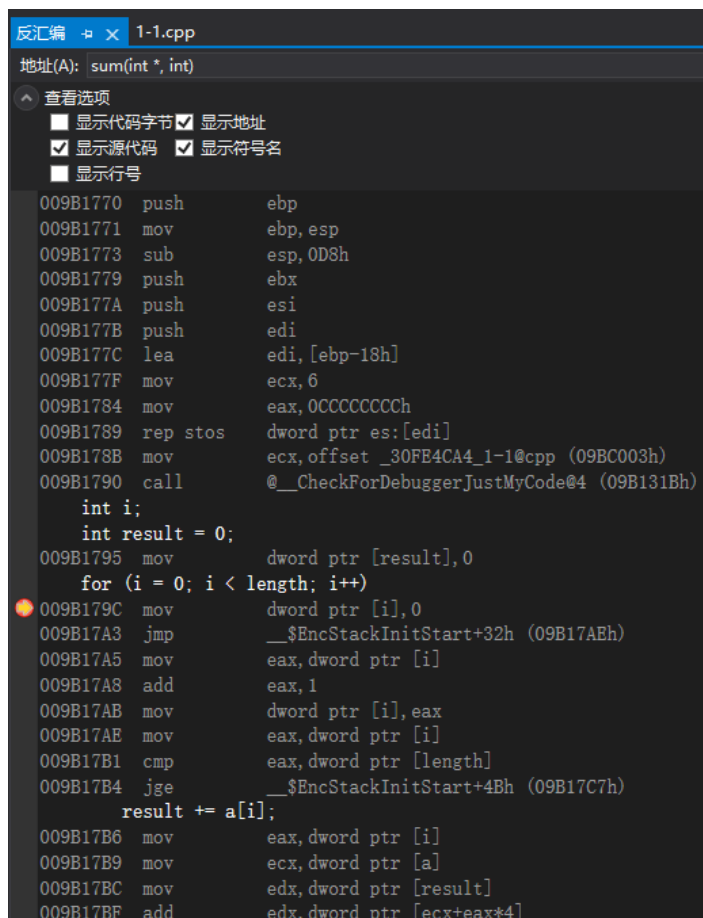
熟悉支持汇编语言开发、调试以及软件反汇编的主流工具的功能、特点与局限性及使用方法。

3.2 实验过程

3.2.1 WINDOWS10 下 VS2019 等工具包

VS2019 可以编译、链接、调试程序，可以显示反汇编窗口、寄存器窗口、监视窗口、内存窗口，可以进行单步执行和设置断点的操作。

(1) 反汇编窗口，如图所示。可以选择勾选代码字节、显示地址、显示源代码、显示符号名、行号等。这个功能很方便，可以很让我们将汇编代码和 C 代码一一对应并显示出指令机器码、地址等，一目了然，十分适合调试和观察。



```
反汇编 - 1-1.cpp
地址(A): sum(int *, int)
查看选项
  [ ] 显示代码字节 [x] 显示地址
  [x] 显示源代码  [x] 显示符号名
  [ ] 显示行号
009B1770 push     ebp
009B1771 mov      ebp, esp
009B1773 sub      esp, 0D8h
009B1779 push     ebx
009B177A push     esi
009B177B push     edi
009B177C lea      edi, [ebp-18h]
009B177F mov      ecx, 6
009B1784 mov      eax, 0CCCCCCCCh
009B1789 rep stos  dword ptr es:[edi]
009B178B mov      ecx, offset _30FB4CA4_1-1@cpp (09BC003h)
009B1790 call     @__CheckForDebuggerJustMyCode@4 (09B131Bh)
      int i;
      int result = 0;
009B1795 mov      dword ptr [result], 0
      for (i = 0; i < length; i++)
009B179C mov      dword ptr [i], 0
009B17A3 jmp      __$EncStackInitStart+32h (09B17AEh)
009B17A5 mov      eax, dword ptr [i]
009B17A8 add      eax, 1
009B17AB mov      dword ptr [i], eax
009B17AE mov      eax, dword ptr [i]
009B17B1 cmp      eax, dword ptr [length]
009B17B4 jge      __$EncStackInitStart+4Bh (09B17C7h)
      result += a[i];
009B17B6 mov      eax, dword ptr [i]
009B17B9 mov      ecx, dword ptr [a]
009B17BC mov      edx, dword ptr [result]
009B17BF add      edx, dword ptr [ecx+eax*4]
```

图 3.1 反汇编窗口

(2) 内存窗口，如图 3.2 所示。在没有 c 代码只有汇编代码时，有时只显示一个地址值以表示某个变量，通过内存窗口我们就可以看到该地址单元里存放的数据内容，从而倒推该地址值表示的

汇编语言程序设计实验报告

是某个变量，有助于反汇编的实现。

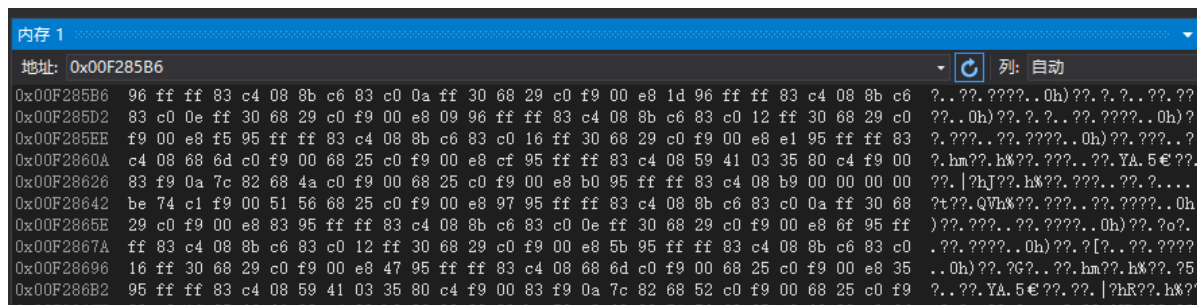


图 3.2 内存窗口

(3) 寄存器窗口，如图 3.3 所示。在寄存器窗口中可以很直白的看到当前步骤中寄存器的值，在程序发生错误时，调试的过程中便可以通过观察寄存器值的变化来揪出一些逻辑错误。寄存器窗口比较有趣的是可以自己选择显示哪些寄存器。

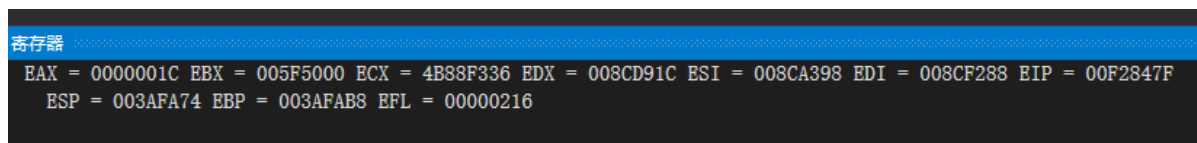


图 3.3 寄存器窗口

(4) 监视窗口，如图 3.4 所示。监视窗口可以观察变量、寄存器、地址等值。该窗口相当于把多个功能集于一体，同时可以做到只关心某个特定的变量。但是在结构体数组变量的观察和监视中他不能看到所有元素，只能看到第一个元素，比如图中的 INIT，只能看到 INIT[0]的各成员值。

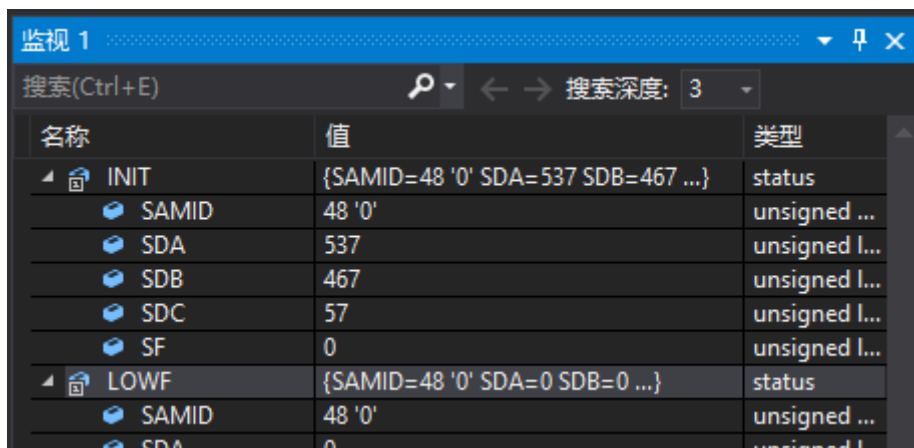


图 3.4 监视窗口

3.2.2 DOSBOX 下的工具包

DOSBox 是一个 DOS 模拟程序，可以很方便的移植到其他的平台。DOXBox 界面不是图形化界面，需要手动输入指令来执行操作，界面如图 3.5 所示。

汇编语言程序设计实验报告

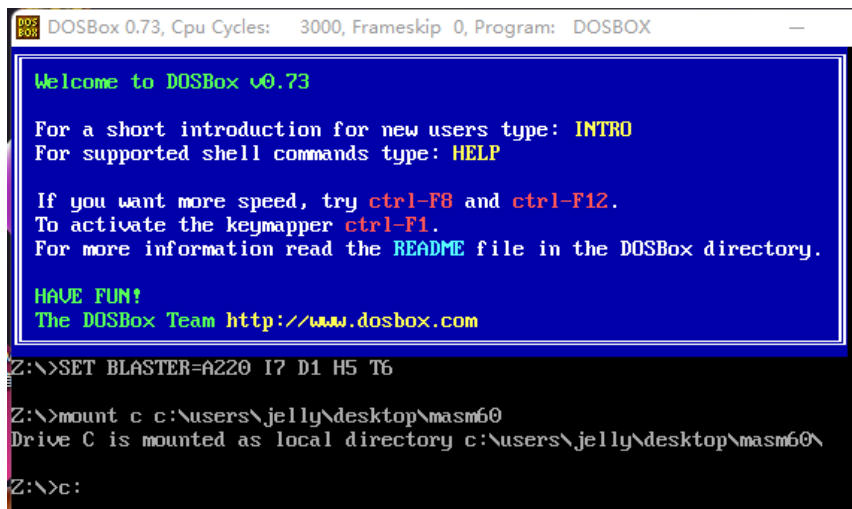


图 3.5 DOSBOX 初始界面图

操作 mount c c:\users\jelly\desktop\masm60 :把“c”作为虚拟 C 盘," c:\users\jelly\desktop\masm60"是要虚拟的文件夹位置

进入虚拟 C 盘的指令: c: ;

编译.asm 文件的指令: MASM DEMO.ASM, 执行后会生成.obj 文件;

链接.obj 文件的指令: LINK DEMO.OBJ, 执行后会生成.exe 文件;

运行.exe 文件的指令: DEMO;

调试程序的指令: TD DEMO.EXE

DOSBOX 调试界面如图 3.6 所示, 我个人觉得比较有趣的特色是界面很古早, 鼠标用起来不太方便用键盘操作这个特点比较有意思和复古, 而且真个界面比较方正和齐全, 左上方是代码, 右上方是寄存器, 还可以直接看到堆栈段和标志位和数据段, 全部都显示在一个页面里, 感觉很方便。还有就是最下面很贴心地显示了快捷键, 整个设计还是较为人性化的。

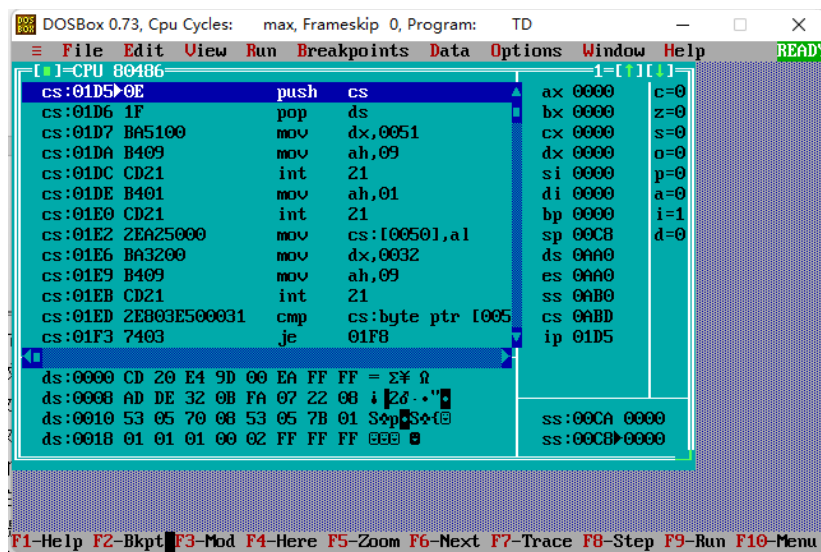


图 3.6 DOSBOX 调试界面图

汇编语言程序设计实验报告

3.2.3 QEMU 下 ARMv8 的工具包

GDB 是我最喜欢的一个工具了在本此实验里。个人感觉 GDB 的功能很强大，只是通过命令行的操作就能完成打断点（如图 3.7 所示）、单步调试运行、运行等各种功能，我觉得最有趣的在于它还能通过 `disas` 操作只打印出某个部分的汇编语句代码，如图 3.8 所示。

```
(gdb) b 108
Breakpoint 1 at 0x8048b14: file bomb.c, line 108.
```

图 3.7 GDB 打断点操作

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
0x08048dee <+0>:      push    %ebx
0x08048def <+1>:      sub     $0x24,%esp
0x08048df2 <+4>:      mov     0x2c(%esp),%ebx
0x08048df6 <+8>:      mov     %gs:0x14,%eax
0x08048dfc <+14>:     mov     %eax,0x18(%esp)
0x08048e00 <+18>:     xor     %eax,%eax
0x08048e02 <+20>:     push    %ebx
0x08048e03 <+21>:     call   0x8049094 <string_length>
0x08048e08 <+26>:     add     $0x10,%esp
0x08048e0b <+29>:     cmp     $0x6,%eax
```

图 3.9 GDB 的 `disas` 操作

3.3 小结

通过实验一、实验四和实验五，我尝试和使用了各种不同的编程工具。个人感觉从集成和完备的方便 VS2019 是最成熟的一款工具，界面操作起来是最便捷的。DOSBOX 界面虽然小巧和好看但是在调试的过程中由于操作的单一性比较繁琐，而且几乎只能用键盘操作还是比较麻烦，调试功能也没有像 GDB 那样完善。GDB 虽然是由命令行控制，在一开始不熟悉指令的情况下可能会觉得用起来比较困难，但是在熟悉各种指令的用法之后会非常的方便，而且不像 VS2019 那样集成度高、系统庞大。

汇 编 语 言 程 序 设 计 实 验 报 告

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].....