

# 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 202001 班

学 号： U202011641

姓 名： 刘景宇

指导教师： 朱虹

实验时段： 2022 年 3 月 7 日~4 月 29 日

实验地点： 东九 A311

### 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. 5. 13

### 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 .....	2
1.1 目的与要求 .....	2
1.2 实验内容 .....	2
1.3 内容 1.1 的实验过程 .....	2
1.3.1 设计思想 .....	2
1.3.2 流程图 .....	3
1.3.3 源程序 .....	7
1.3.4 实验记录与分析 .....	14
1.4 内容 1.2 的实验过程 .....	18
1.4.1 实验方法说明 .....	18
1.4.2 实验记录与分析 .....	19
1.5 小结 .....	22
二、利用汇编语言特点的实验 .....	23
2.1 目的与要求 .....	23
2.2 实验内容 .....	23
2.3 实验过程 .....	23
2.3.1 实验方法说明 .....	23
2.3.2 实验记录与分析 .....	24
2.4 小结 .....	28
三、工具环境的体验 .....	30
3.1 目的与要求 .....	30
3.2 实验过程 .....	30
3.2.1 WINDOWS10 下 VS2019 等工具包 .....	30
3.2.2 DOSBOX 下的工具包 .....	32
3.2.3 QEMU 下 ARMv8 的工具包 .....	33
3.3 小结 .....	34
参考文献 .....	35

## 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

内容 1.1 要求实现一个较为完整的计算机系统运行状态的监测系统，该监测系统的主要功能为（1）程序开始执行后，能够提示用户输入用户名和密码，如果不正确，则要给出出错信息后再次重新输入，最多输入三次，三次都出错时程序退出；（2）先对 N 组采集到的状态信息计算 f，并进行分组复制；（3）然后将 MIDF 存储区的各组数据在屏幕上显示出来；（4）最后，按 R 键重新从“（2）”处执行，按 Q 键退出程序。

对于功能（1），为正确用户名和密码以及用户输入的用户名和密码分配空间，判断用户输入是否正确，即要判断用户输入的字符串是否和正确的字符串一致。对于该功能，我们采用宏来实现用户名和密码的比较，该宏接受五个参数，分别为真实字符串及其的长度，用户输入字符串及其长度，以及存放比较结果的参数。处理时，优先比较字符串长度，然后将正确字符串和用户输入字符串逐字节比较。为了避免宏调用中修改寄存器的值引起程序异常，在宏定义中使用 PUSHAD、POPAD 指令来保护寄存器中的值。

为了求得字符串长度，另外设计一个子程序，接受一个字符串首地址参数，返回该字符串的长度，求取字符串长度的子程序采用堆栈来传递参数，求字符串长度时将每个字节和 0 比较以得到长度。为了保护寄存器，在子程序内部将所有寄存器的值进栈，最后再将寄存器的值出栈。

对于功能（2），我们要对每组采集到的数据进行计算并且按照计算结果复制到对应区域，为了更好的实现分组复制，需要定义一些变量用来存储地址信息，包括 SAMPLES、LOWF、MIDF、HIGHF 存储区域的首地址以及尾地址，尾地址初始等于首地址，随着一组组数据的复制，尾地址逐渐更新，指向已存放数据的末尾，同时定义 LEN 存放一个结构体的大小，NUMOFMIDF 存放 MIDF 中已经复制的数据的个数，NUM 存放当前要计算的 SAMPLES 数据在结构体数组中的位置。

# 汇编语言程序设计实验报告

为了计算 F，我们另外设计一个子程序 COMPUTE\_F，接收结构体数组的地址信息，以及当前计算的结构体在数组中的位置，读取相关信息进行考虑溢出的计算，如果发生溢出，并将结构存储到结构体相应位置。为了保护寄存器，在子程序内部将所有寄存器的值进栈，最后再将寄存器的值出栈。

为了实现复制，我们定义一个子程序 COPY\_SAMPLES，接收结构体数组的地址信息，以及 LOWF、MIDF、HIGHF 相关的地址信息，结构体的大小 LEN，每次读取一个结构体的 SF，按照 SF 的值进行数据拷贝，SF>100，拷贝到 HIGHF，SF=100，则拷贝到 MIDF，SF<100 拷贝到 LOWF，拷贝过程中定位数据位置以及要拷贝到的内存空间位置，逐字节拷贝 LEN 个字节。为了保护寄存器，在子程序内部将所有寄存器的值进栈，最后再将寄存器的值出栈。

对于功能（3），我们定义一个子程序 DISPLAY，输出 MIDF 中的数据，接受 MIDF 区域的首地址以及 MIDF 中存放的结构体的个数 NUMOFMIDF，调用 C 语言库 printf 函数，对数据数据进行输出。为了提升对于多模块的认识，掌握汇编语言的多模块设计方法，在这里，我们采用多模块，在该工程下另外定义一个 display.asm 文件，用以实现该子程序。并且为了保护寄存器，在子程序内部将所有寄存器的值进栈，最后再将寄存器的值出栈。

为了在多模块中共享数据，在主模块 main.asm 中对 DISPLAY 子程序要用到的参数声明为 public，同时在子模块中采用 extern 关键字声明外部变量，实现模块间的数据通信，除此之外，在主模块中声明 DISPLAY PROTO C MIDFLOW:DWORD, LENG:DWORD，用来在主模块中调用子模块中的子程序。

## 1.3.2 流程图

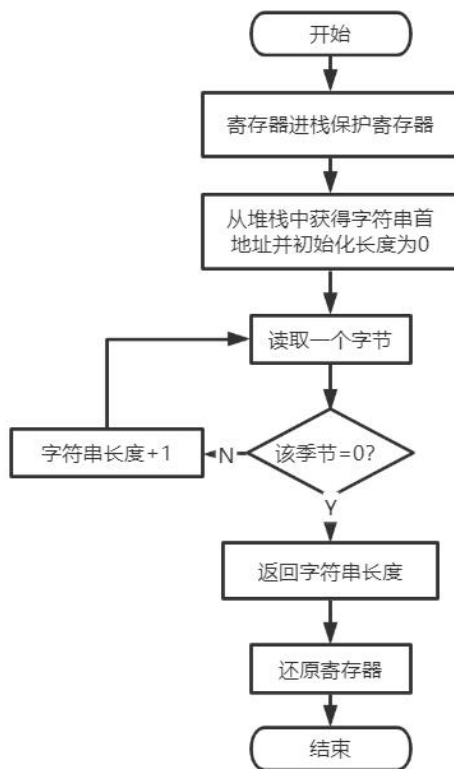


图 1.1 STRLEN 子程序流程图

# 汇编语言程序设计实验报告

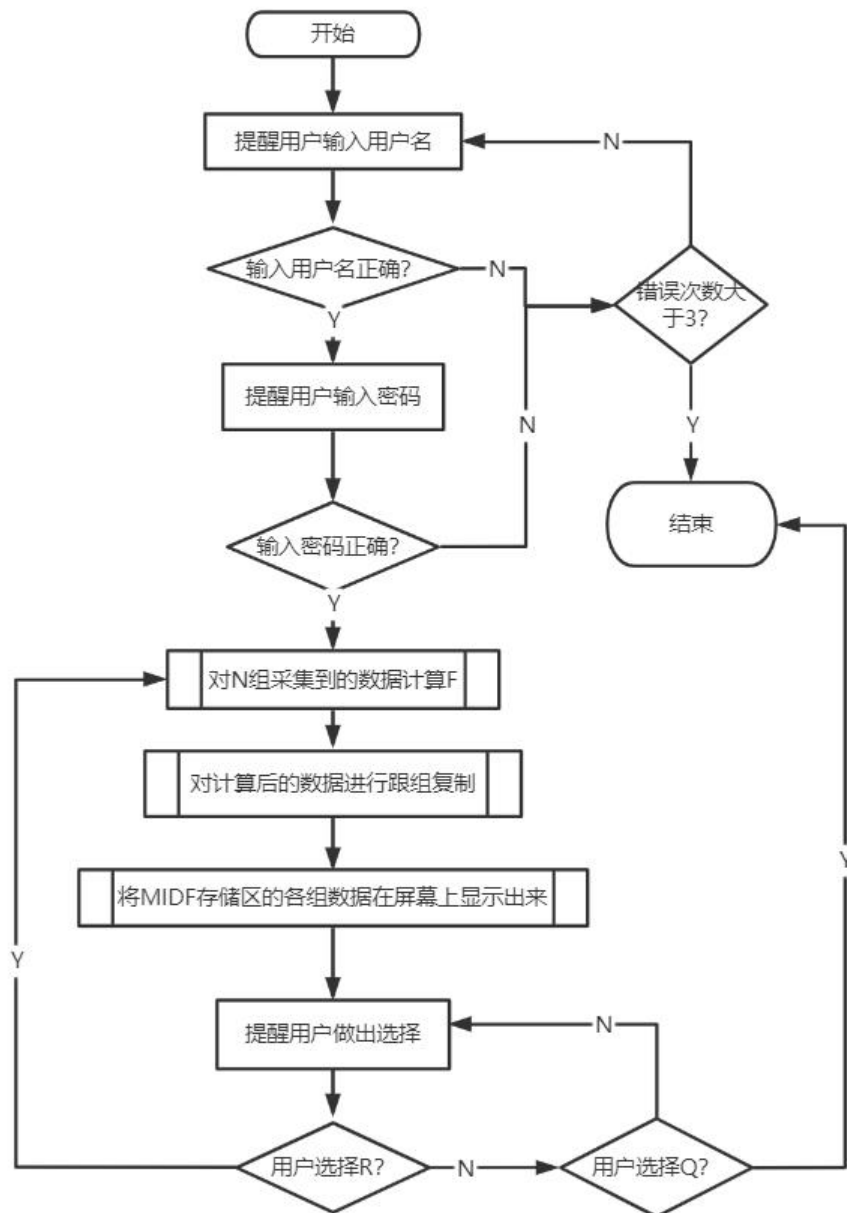


图 1.2 主程序流程图

# 汇编语言程序设计实验报告

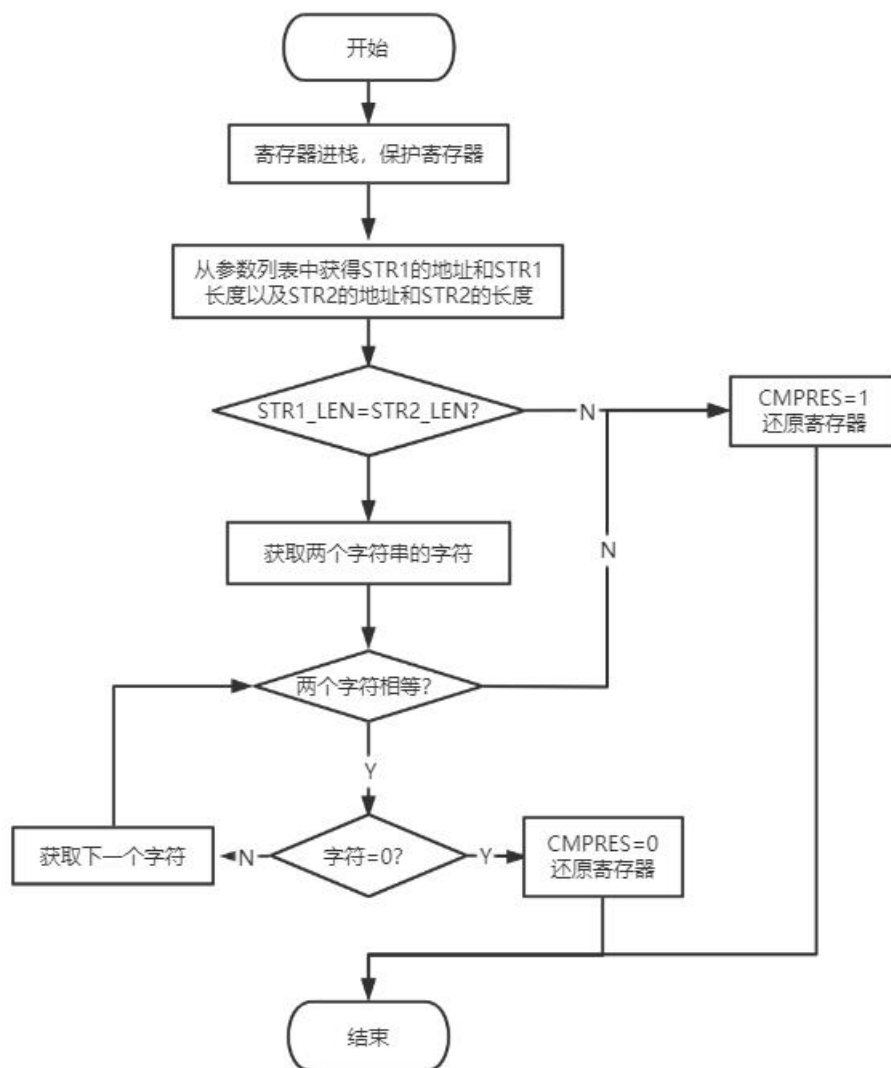


图 1.3 USER\_CMP、PASSWD\_CMP 子程序流程图

# 汇编语言程序设计实验报告

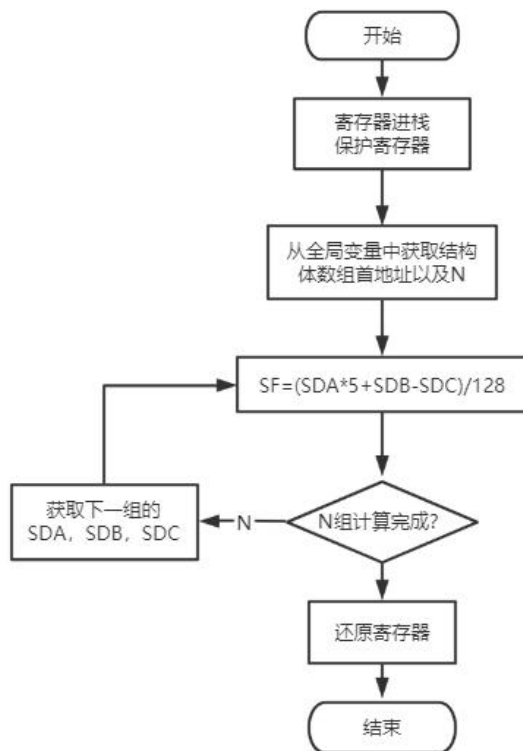


图 1.4 COMPUTE\_F 子程序流程图

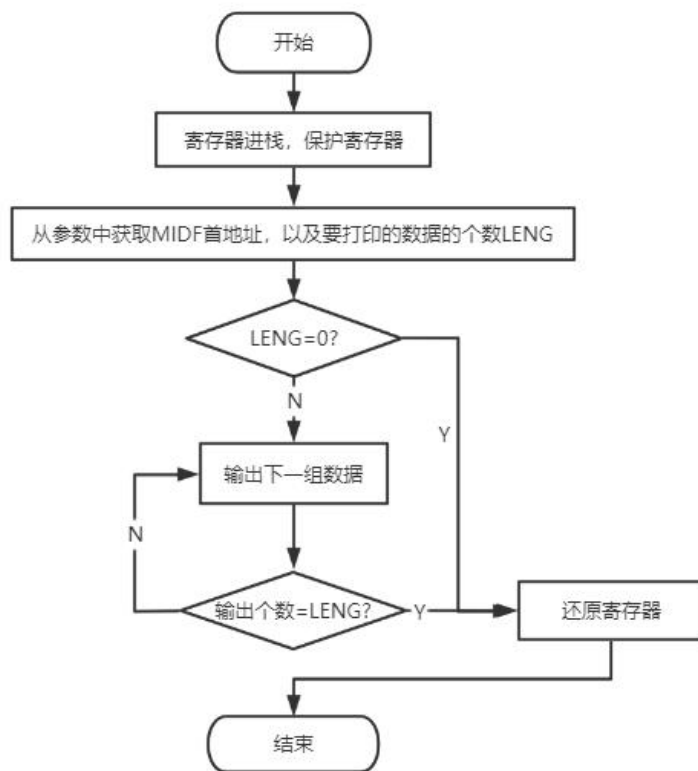


图 1.5 DISPLAY 子程序流程图

# 汇编语言程序设计实验报告

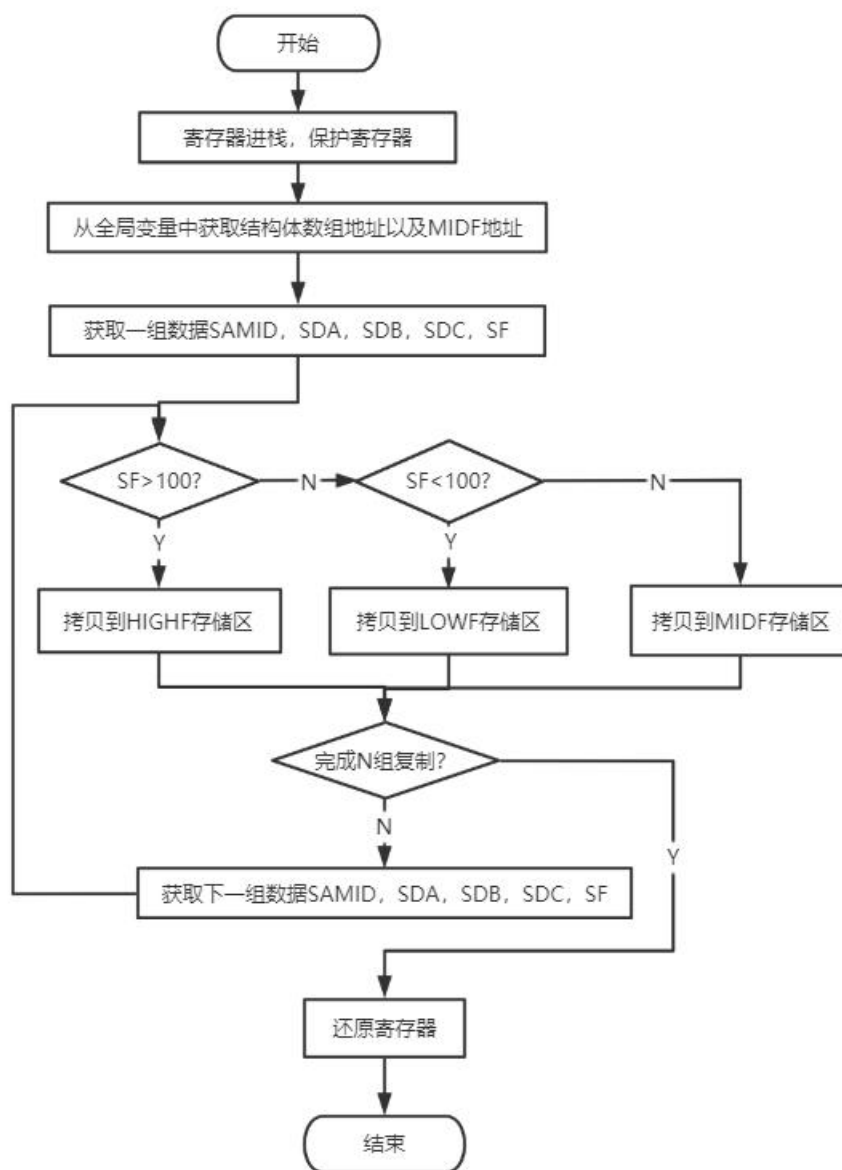


图 1.6 COPY\_SAMPLES 流程图

## 1.3.3 源程序

主模块 main.asm

```
;author: Liujingyu      CS2001
;
;main.asm
;
;STRLEN                :求解字符串长度
;USER_CMP              :比较用户名
;PASSWD_CMP            :比较密码
;COMPUTE_F             :计算 SF 并保存
```



# 汇编语言程序设计实验报告

;COPY\_SAMPLES :内存拷贝

```
.686
.model flat, c
    ExitProcess PROTO STDCALL :DWORD
    includelib kernel32.lib
    printf PROTO C :VARARG
    scanf PROTO C :VARARG
    DISPLAY PROTO C MIDFLOW:DWORD, LENG:DWORD;子程序中的函数
    includelib libcmnt.lib
    includelib legacy_stdio_definitions.lib
```

;宏实现定义字符串比较（定义形参），用户名、密码的比较都用该宏

```
USER_CMP MACRO STR1, STR1_LEN, STR2, STR2_LEN, CMPRES
;CMPRESULT 用来记录比较结果，1 表示不等，0 表示等
    PUSHAD ;保护寄存器
    MOV EAX, STR1_LEN
    XOR EAX, STR2_LEN ;用异或判断两个数相同
    JNZ NOT_EQUAL1 ;两个串长度不等
    MOV ECX, STR1_LEN ;比较次数
    MOV ESI, 0
CMP_LP1:
    MOV AL, STR1[ESI]
    XOR AL, STR2[ESI] ;
    JNZ NOT_EQUAL1
    INC ESI
    DEC ECX
    JG CMP_LP1
EQUAL1: MOV CMPRES, 0 ;相等
    JMP END_CMP1
NOT_EQUAL1:MOV CMPRES, 1 ;不等
END_CMP1: POPAD
    ENDM
```

```
PASSWD_CMP MACRO STR1, STR1_LEN, STR2, STR2_LEN, CMPRES
;CMPRES 用来记录比较结果，1 表示不等，0 表示等
    PUSHAD ;保护寄存器
    MOV EAX, STR1_LEN
    XOR EAX, STR2_LEN ;用异或判断两个数相同
    JNZ NOT_EQUAL2 ;两个串长度不等
    MOV ECX, STR1_LEN ;比较次数
    MOV ESI, 0
CMP_LP2:
    MOV AL, STR1[ESI]
    XOR AL, STR2[ESI] ;
    JNZ NOT_EQUAL2
    INC ESI
    DEC ECX
    JG CMP_LP2
EQUAL2: MOV CMPRES, 0 ;相等
    JMP END_CMP2
NOT_EQUAL2:MOV CMPRES, 1 ;不等
END_CMP2: POPAD
    ENDM
```

SAMPLES STRUCT

# 汇 编 语 言 程 序 设 计 实 验 报 告

```

        SAMID DB "123456789"                ;每组数据的流水号
        SDA DD 12800/5                      ;状态信息 a
        SDB DD 100                          ;状态信息 b
        SDC DD 100                          ;状态信息 c
        SF DD 0                             ;处理结果 f
SAMPLES ENDS

        public PRINTFRONT                    ;public 数据
        public MSGFRONT
        public MSGNOMIDF
        public PRINTCHAR
        public PRINTINT

.DATA
print db "%s",0ah,0                        ;输出字符串格式
scan db "%s",0                             ;输入字符串格式
MSGNAME DB "Please enter user name (No more than 15 characters)",0;提示
MSGPASSWD DB "Please enter your password (No more than 10 characters)",0
MSGOK DB "Successfully Log In!",0          ;全部输入正确
MSGNAMEER DB "Incorrect Username",0 ;用户名错误
MSGPASSWDER DB "Incorrect Password!",0 ;错误
MSGTIPEND DB "All input times have been exhausted",0
USER DB "LiuJingyu"
LENNAME = $-USER                          ;自动计算用户名的长度
PASSWD DB "liujingyu"
LENPASSWD = $-PASSWD                      ;自动计算密码的长度
ERCOUNT DB 3                             ;最多输入错误次数
INPUTNAME DB 16 DUP(0)                   ;存放用户输入
INPUTPASSWD DB 11 DUP(0)

;以下为计算 F 相关的数据
SDATA SAMPLES 10000 DUP(<>)
LEN=($-SDATA)/10000                      ;一个结构体的大小，正好是 25
N DD 10000                               ;SAMPLES 总个数
LOOPN DD 10000                           ;外循环次数
LOWF DB 260000 DUP('L')                  ;LOWF 数组
MIDF DB 260000 DUP('M')                  ;MIDF 数组
HIGHF DB 260000 DUP('H')                 ;HIGHF 数组
NUMOFMIDF DD 0                           ;记录 MIDF 中数据的个数
NOW DD SDATA                             ;指向 SAMPLES 中的位置
NUM DD 0                                  ;当前指向的 SAMPLES 在 SAMPLES 数组中的位置
LOWS DD offset LOWF                      ;存放三个空间的首地址以及结尾地址
LOWE DD offset LOWF                      ;S-start 表示开始地址，E-end 表示当前末尾位置
MIDS DD offset MIDF
MIDE DD offset MIDF
HIGHS DD offset HIGHF
HIGHE DD offset HIGHF
MSGCOMPUTE DB "Compute completely!",0
MSGCOPY DB "Copy completely!",0
MSGFRONT DB "SAMID:",0
PRINTFRONT DB "%s",0
PRINTCHAR DB "%c",0
PRINTINT DB ", SDA:%d, SDB:%d, SDC:%d, SDF:%d",0ah,0
MSGNOMIDF DB "There is nop data in MIDF",0
MSGTIP DB "Plearse input R or Q,R for run,Q for exit!",0

.STACK 1024
.CODE
main proc c

```

# 汇编语言程序设计实验报告

```
LOCAL LENINPUT:DWORD ;记录输入的用户名\密码的长度
LOCAL CMPRES:DWORD
LOCAL LENUSERNAME:DWORD
;输出提示信息
INPUT_LP:
    INVOKE printf,offset print,OFFSET MSGNAME
    INVOKE scanf,offset scan,OFFSET INPUTNAME
    MOV LENINPUT,0
    MOV CMPRES,1 ;默认不相等
    PUSH OFFSET INPUTNAME
    PUSH LENINPUT
    CALL STRLEN
    POP LENINPUT ;弹出最后
    MOV EAX,LENAME
    MOV LENUSERNAME,EAX
    USER_CMP USER,LENUSERNAME,INPUTNAME,LEINPUT,CMPRES
    CMP CMPRES,0
    JE NEXTINPUT ;如果用户名相同
    invoke printf,offset print,offset MSGNAMEER
    MOV CL,ERCOUNT ;用户名不同的情况
    DEC CL
    MOV ERCOUNT,CL
    JNE NEXTTIP1
    invoke printf,offset print,offset MSGTIPEND
    JMP EXIT
NEXTTIP1:
    MOV ERCOUNT,CL
    AND CL,-1
    JG INPUT_LP
    JMP EXIT
NEXTINPUT:
    invoke printf,offset print,offset MSGPASSWD;输入密码
    INVOKE scanf,offset scan,OFFSET INPUTPASSWD
    MOV LENINPUT,0
    MOV CMPRES,1 ;默认不相等
    PUSH OFFSET INPUTPASSWD
    PUSH LENINPUT
    CALL STRLEN
    POP LENINPUT ;弹出最后
    PASSWD_CMP PASSWD,LENPASSWD,INPUTPASSWD,LEINPUT,CMPRES
    OR CMPRES,0
    JE NEXTSTEP ;如果密码相同
    invoke printf,offset print,offset MSGPASSWDER
    MOV CL,ERCOUNT ;用户名不同的情况
    DEC CL
    MOV ERCOUNT,CL
    JNE NEXTTIP2
    invoke printf,offset print,offset MSGTIPEND
    JMP EXIT
NEXTTIP2:
    MOV ERCOUNT,CL
    AND CL,-1
    JG INPUT_LP
    JMP EXIT
NEXTSTEP:
    ;登录成功，这里做其他数据处理
    invoke printf,offset print,offset MSGOK
```

# 汇编语言程序设计实验报告

```
        ;计算所有 SAMPLES 的 F, 并存入
OUTLOOP:
    MOV EAX,offset MIDF
    MOV MIDE,EAX
    MOV EAX,offset LOWF
    MOV LOWE,EAX
    MOV EAX,offset HIGHF
    MOV HIGHE,EAX
    MOV NUMOFMIDF,0
    MOV NUM,0
    MOV EAX,offset SDATA
    MOV NOW,EAX
    CALL COMPUTE_F
    invoke printf,offset print,offset MSGCOMPUTE
    ;完成相应的复制工作
    CALL COPY_SAMPLES
    invoke printf,offset print,offset MSGCOPY
    ;完成 MIDF 的输出工作
    invoke DISPLAY,OFFSET MIDF,NUMOFMIDF

INPUTCHOICE:
    invoke printf,offset PRINTFRONT, offset MSGTIP
    invoke scanf,offset PRINTCHAR, offset CHOICE
    cmp CHOICE,'R'
    JE OUTLOOP
    cmp CHOICE,'Q'
    JE EXIT
    JNE INPUTCHOICEEXIT:
EXIT:
    invoke ExitProcess, 0
    RET
main endp

STRLEN PROC                                ;计算字符串长度
    PUSH EBP
    MOV EBP,ESP
    PUSH EBX
    PUSH ECX
    MOV ECX,[EBP+8]                        ;存放长度结果
    MOV EBX,[EBP+12]                       ;字符串地址
STRLEN_LP:
    CMP BYTE PTR [EBX],0
    JNE LENNEXT
    JMP EXIT2
LENNEXT:
    INC EBX
    INC ECX
    JMP STRLEN_LP
EXIT2:
    MOV [EBP+8],ECX
    POP ECX
    POP EBX
    POP EBP
    RET
STRLEN ENDP

COMPUTE_F PROC ;计算 F, 并将 F 存入相应的位置
```

# 汇编语言程序设计实验报告

```
PUSHAD
MOV EDI, N
MOV ESI, 0
LOOPTOTAL:
MOV EAX, SDATA[ESI]. SDA
MOV EBX, 5
MOV EDX, 0
IMUL EBX                                ;结算结果存在 (EDX, EAX) 中
ADD EAX, SDATA[ESI]. SDB
JNC NEXT1                               ;如果没有溢出 (有符号)
ADC EDX, 0                               ;完成进位
NEXT1:
SUB EAX, SDATA[ESI]. SDC
JNC NEXT2
SBB EDX, 0
NEXT2:
ADD EAX, 100
ADC EDX, 0
SHRD EAX, EDX, 7
SAR EDX, 7
MOV SDATA[ESI]. SF, EAX                 ;这里完成计算并存入 SF 中
ADD ESI, LEN
DEC EDI
JG LOOPTOTAL                            ;这里循环计算 N 组数据
POPAD
RET
COMPUTE_F ENDP

COPY_SAMPLES PROC
LOCAL NUMS:DWORD
PUSHAD
MOV EDX, N                               ;总拷贝次数
MOV ESI, 0                               ;SAMPLES 中的位置
MOV NUMS, EDX
COPY_LOOP:
MOV EAX, SDATA[ESI]. SF
CMP EAX, 100
JG CP_LP1                                ;SF>100
JL CP_LP2                                ;SF<100
JZ CP_LP2                                ;SF=100

MOV ECX, LEN
MOV EDX, NOW
INC NUMOFMIDF
cp_loop0:
MOV EDI, MIDE
MOV BL, [EDX]
MOV [EDI], BL                            ;按字节拷贝
INC MIDE
INC EDX                                  ;指向下一个字节
DEC ECX
JG cp_loop0                              ;执行 25 次
MOV NOW, EDX                             ;更新 NOW 指针
JMP COPY_EXIT
CP_LP1:
MOV ECX, LEN
MOV EDX, NOW
cp_loop1:
```

# 汇编语言程序设计实验报告

```
    MOV EDI, HIGHE
    MOV BL, [EDX]
    MOV [EDI], BL           ;按字节拷贝
    INC HIGHE
    INC EDX                 ;指向下一个字节
    DEC ECX
    JG cp_loop1             ;执行 25 次
    MOV NOW, EDX            ;更新 NOW 指针
    JMP COPY_EXIT          ;拷贝完成
CP_LP2:
    MOV ECX, LEN
    MOV EDX, NOW
cp_loop2:
    MOV EDI, LOWE
    MOV BL, [EDX]
    MOV [EDI], BL           ;按字节拷贝
    INC LOWE
    INC EDX                 ;指向下一个字节
    DEC ECX
    JG cp_loop2             ;执行 25 次
    MOV NOW, EDX            ;更新 NOW 指针
COPY_EXIT:
    ADD ESI, LEN
    DEC NUMS
    JG COPY_LOOP
    POPAD
    RET
COPY_SAMPLES ENDP

END
```

## 子模块 display.asm

```
;author: Liujingyu      CS2001
;
;display.asm
;
;DISPLAY                :展示 MIDF 中存储的数据

.686
.model flat, c
    printf                PROTO C :VARARG
    includelib    libcmtd.lib
    includelib    legacy_stdio_definitions.lib
    extern PRINTFRONT :sbyte
    extern MSGFRONT :sbyte
    extern MSGNOMIDF :sbyte
    extern PRINTCHAR :sbyte
    extern PRINTINT :sbyte
.code
; display : 实现展示结果
; MIDFLOW : 要打印的首地址
; LENG    : 要打印的长度
```

```
DISPLAY PROC MIDFLOW:DWORD, LENG:DWORD ;将 MIDF 中的各组数据输出出来, 包括一个 9 位字符串, 剩余的 SDA, SDB, SDC, SF 均为 4 字节
    LOCAL SAMIDLEN:DWORD
```

# 汇编语言程序设计实验报告

```
LOCAL DIGITNUM:DWORD
LOCAL LOOPTIMES:DWORD
LOCAL TMPCHAR:BYTE
PUSHAD
CMP LENG, 0
JNZ DISPLAY_NEXT
invoke printf, offset PRINTFRONT, offset MSGNOMIDF
JMP DIS_EXIT
DISPLAY_NEXT:
JE DIS_EXIT
MOV EDI, MIDFLOW ;MIDF 的开头
MOV ECX, LENG
MOV LOOPTIMES, ECX
DISPLAY_LP:
MOV SAMIDLEN, 9 ;开头的 9 个字符
MOV DIGITNUM, 4
invoke printf, offset PRINTFRONT, offset MSGFRONT
DIS_LP1:
MOV AL, [EDI]
MOV TMPCHAR, AL
invoke printf, offset PRINTCHAR, TMPCHAR
INC EDI
DEC SAMIDLEN
JG DIS_LP1
invoke printf, offset PRINTINT, dword ptr[EDI], dword ptr[EDI+4], dword ptr[EDI+8], dword
ptr[EDI+12]
ADD EDI, 16
DEC LOOPTIMES
JG DISPLAY_LP
DIS_EXIT:POPAD
RET
DISPLAY 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; WINDOWS10 下 VS2019 社区版。

### 2. 汇编、链接中的情况

汇编过程中出现了 6 个问题, 主要是编写汇编程序时汇编语法的问题。

使用宏定义用户名和密码检测时, 出现重复定义的问题, 通过观察 VS2019 给出的提示, 发现由于宏是在编译时将宏定义程序代码插入在宏调用的地方, 当宏中定义有循环或分支语句时, 会出现问题。对于该问题, 可以将宏定义中的循环、分支语句编写为另一个新的子程序, 在宏定义中调用, 或者将用户名和密码的判断分割成两个宏定义, 都可以解决重复定义的问题。

调用子程序以及 C 语言库函数可能会修改部分寄存器的值, 在调用 printf 时出现观察到寄存器中发生了变化。为了避免寄存器的变化带来不可预知的问题, 我们要尽量避免该风险, 可以采用保护寄存器的方式或者定义全局或局部变量的存储数据的方式。

访问冲突, 通过设置断点, 调试状态下观察内存情况, 发现由于数组越界等使得程序要访问非数据段的内容, 使得运行过程中出现访问冲突, 程序退出。要尽量避免数组越界等可能产生非法访问的情况, 对代码逻辑足够清晰。

# 汇编语言程序设计实验报告

invalid instruction operands, 通过 VS2019 指向的错误点, 分析, 发现是由于寻址方式不正确出现的问题, 例如两个操作数不能都是存储器寻址等。修改为正确的汇编语言语法格式, 再次编译运行, 能够解决该问题。

instruction operands must be the same size, 通过 VS2019 指向的发生错误的地方, 发现是由于操作数两边数据类型不一致, 即两个操作数大小不一致, 经常发生在想用地址却少写 offset 的语句中。通过补全 offset 或者其他扩展指令使得两个操作数类型一致。

局部变量的声明必须紧跟在子程序定义后面, 中间不能插入其他内容。

## 3. 程序基本功能的验证情况

主要通过观察终端输出内容以及内存中存储情况等来验证程序基本功能。

### (1) 用户登录功能的验证

运行该程序, 测试输入正确用户名和密码以及错误的用户名和密码, 观察终端输出情况。

当用户输入错误的用户名时, 提示用户重新输入, 当输入正确的用户名时, 则提示输入密码, 输入错误的密码时, 重新输入, 输入密码正确时, 提示用户 “Successfully Log In!”, 表示成功登录。测试结果如图 1.7 所示。

当用户输入次数超过限定次数 3 次时, 提示用户 “All input times have been exhausted”, 表示尝试次数用尽。测试结果如图 1.8 所示。

同时, 该测试也验证了 STRLEN、USER\_CMP 和 PASSWD\_CMP 子程序的正确性。

```
Please enter user name (No more than 15 characters)
liujingyu
Incorrect Username
Please enter user name (No more than 15 characters)
LiuJingyu
Please enter your password (No more than 10 characters)
liujingyu
Incorrect Password!
Please enter user name (No more than 15 characters)
LiuJingyu
Please enter your password (No more than 10 characters)
liujingyu1
Successfully Log In!
```

图 1.7 正确用户登录功能验证

```
Please enter user name (No more than 15 characters)
liujingyu
Incorrect Username
Please enter user name (No more than 15 characters)
liujingyu
Incorrect Username
Please enter user name (No more than 15 characters)
liujingyu
Incorrect Username
All input times have been exhausted
```

图 1.8 超过登录最大尝试次数

### (2) 计算 F 功能、内存拷贝功能以及展示 MIDF 功能的验证

N 组数据采用初始值, 如图 1.9 所示。设置断点进入调试状态, 观察内存中的情况, 并且观察终端输出。

设置 SAMPLES 初始值, 流水号为 123456789, SDA=23113, SDB=1023, SDC=-1011, SF=0。



# 汇编语言程序设计实验报告

```

SAMPLES STRUCT
    SAMID DB "123456789" ;每组数据的流水号
    SDA DD 23113 ;状态信息a
    SDB DD 1023 ;状态信息b
    SDC DD -1011 ;状态信息c
    SF DD 0 ;处理结果f
SAMPLES ENDS

```

图 1.9 SAMPLES 初始值

观察内存中的数据，可以观察到 SF 已经计算完成并且已经存入到内存中的相应位置，通过计算器验算，SF 计算正确，并且<100，存入到 LOWF 存储区，通过内存中找到 LOWF 存储区，可以发现相应的内容已经存入到对应的位置。SDATA 中的内容如图 1.10 所示，LOWF 中的内容如图 1.11 所示。

内存 1	
地址:	0x0078C102
0x0078C102	31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37
0x0078C122	38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff
0x0078C142	03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff
0x0078C162	97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33
0x0078C182	34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49
0x0078C1A2	5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00
0x0078C1C2	0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00
0x0078C1E2	fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36

图 1.10 SDATA 内容

内存 1	
地址:	0x007C919A
0x007C919A	31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37
0x007C91BA	38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff
0x007C91DA	03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff
0x007C91FA	97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33
0x007C921A	34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49
0x007C923A	5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00
0x007C925A	0d fc ff ff 97 03 00 fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00
0x007C927A	fe 31 32 33 34 35 36 37 38 39 49 5a 00 00 ff 03 00 00 0d fc ff ff 97 03 00 fe 31 32 33 34 35 36

图 1.11 LOWF 内容

当终端中提示，计算完成，内存拷贝完成，并且没有数据在 MIDF 中，如图 1.12 所示。

```

Please enter user name (No more than 15 characters)
LiuJingyu
Please enter your password (No more than 10 characters)
liujingyu!
Successfully Log In!
Compute completely!
Copy completely!
There is nop data in MIDF.

```

图 1.12 终端输出内容 1

通过观察上述内容，可以验证 COMPUTE\_F、COPY\_SAMPLES 子程序功能实现，DISPLAY 子程序部分完成，为了完整验证 DISPLAY，修改 SAMPLES 数据的初始值，重新运行程序，结果

# 汇编语言程序设计实验报告

如图 1.13 所示。

```
Please enter user name (No more than 15 characters)
LiuJingyu
Please enter your password (No more than 10 characters)
liujingyu1
Successfully Log In!
Compute completely!
Copy completely!
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
SAMID:123456789, SDA:2560, SDB:100, SDC:100, SDF:100
```

图 1.13 终端输出内容 2

## 4. 使用调试工具观察、探究代码的情况

主程序与子程序之间通过寄存器、全局变量或者传递形参等来互相传递信息。

利用 VS2019 设置断点，进入调试状态，通过单步执行逐步执行，观察调用子程序对应的反汇编代码，调用子程序代码如下：

invoke DISPLAY, OFFSET MIDF, NUMOFMIDF

观察子程序开头的反汇编代码，如图 1.14 所示，可以观察到进入在有局部变量的子程序中，编译器移动 esp，预留出了 16 字节，用来存放局部变量，可以使用 LEA 指令获得局部变量的地址。对于没有定义局部变量以及参数的子程序，编译器不会补充反汇编代码。

```
22: DISPLAY PROC MIDFLOW:DWORD, LENG:DWORD ;将MIDF中的各组数据输出出来
00938450 55                push     ebp
00938451 8B EC            mov      ebp, esp
00938453 83 C4 F0         add      esp, 0FFFFFFF0h
```

图 1.14 编译器自动补充的反汇编代码

在执行 call 语句时，CPU 将 call 语句后面的指令进栈，即将 EIP 进栈，同时改变 ESP、EIP，如图 1.15 所示，随后跳转到 call 语句后面的位置，继续执行。在执行 RET 语句时，将栈顶地址出栈，存入 EIP，接下来进入执行 EIP 对应的字句，即实现了子程序回到主程序。

```
寄存器
EAX = 00000011 EBX = 00E3D000 ECX = 6C65B76F EDX = 01398FF9 ESI = 013966F8 EDI = 01398A58 EIP = 00BB873B ESP = 010FFD9C EBP = 010FFDB3 EFL = 00000212
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 0 AC = 1 PE = 0 CY = 0

寄存器
EAX = 00000011 EBX = 00E3D000 ECX = 6C65B76F EDX = 01398FF9 ESI = 013966F8 EDI = 01398A58 EIP = 00BB1159 ESP = 010FFD98 EBP = 010FFDB3 EFL = 00000212
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 0 AC = 1 PE = 0 CY = 0
```

图 1.15 call 指令寄存器变化

```
寄存器
EAX = 00000011 EBX = 0057F000 ECX = 719F7F81 EDX = 00848FF9 ESI = 008466F8 EDI = 00848A58 EIP = 00BB84E2 ESP = 006FFBC8 EBP = 006FFBES EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0

寄存器
EAX = 00000011 EBX = 0057F000 ECX = 719F7F81 EDX = 00848FF9 ESI = 008466F8 EDI = 00848A58 EIP = 00BB8740 ESP = 006FFBCC EBP = 006FFBES EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
```

图 1.16 ret 指令寄存器变化

# 汇编语言程序设计实验报告

宏调用时，编译器将在调用位置插入宏定义的内容，如图 1.15，进行了验证。可以观察到宏和子程序之间的主要差别是：宏指令在执行过程中采用替换和扩展，而子程序通过修改 EIP 的值，改变执行指令。

```
136:      USER_CMP USER, LENUSERNAME, INPUTNAME, LENINPUT, CMPRES
00BB8560 60          pushad
00BB8561 8B 45 F4      mov     eax, dword ptr [LENUSERNAME]
00BB8564 33 45 FC      xor     eax, dword ptr [LENINPUT]
00BB8567 75 23         jne     INPUT_LP+76h (0BB858Ch)
00BB8569 8B 4D F4      mov     ecx, dword ptr [LENUSERNAME]
00BB856C BE 00 00 00 00 mov     esi, 0
00BB8571 8A 86 D3 C0 C2 00 mov     al, byte ptr USER (0C2C0D3h)[esi]
00BB8577 32 86 E7 C0 C2 00 xor     al, byte ptr INPUTNAME (0C2C0E7h)[esi]
00BB857D 75 0D         jne     INPUT_LP+76h (0BB858Ch)
00BB857F 46           inc     esi
00BB8580 49           dec     ecx
00BB8581 7F EE         jg      INPUT_LP+5Bh (0BB8571h)
00BB8583 C7 45 F8 00 00 00 00 mov     dword ptr [CMPRES], 0
00BB858A EB 07         jmp     INPUT_LP+7Dh (0BB8593h)
00BB858C C7 45 F8 01 00 00 00 mov     dword ptr [CMPRES], 1
00BB8593 61          popad
```

图 1.15 宏调用反汇编

invoke 伪指令包含将参数进栈以及 call 语句，并且在使用 invoke 调用有参数的 C 语言库函数或者其他自己编写的子程序时，会将参数从右到左逐个进栈，说明了 C 语言中，是利用堆栈来传递参数，如图 1.16 所示。

```
44:      invoke printf, offset PRINTCHAR, TMPCHAR
009384A1 66 6A 00      push    0
009384A4 8A 45 F3      mov     al, byte ptr [TMPCHAR]
009384A7 66 0F B6 C0   movzx   ax, al
009384AB 66 50         push    ax
009384AD 68 CE 78 AA 00 push    offset PRINTCHAR (0AA78CEh)
009384B2 E8 33 97 FF FF call     _printf (0931BEAh)
009384B7 83 C4 08      add     esp, 8
```

图 1.16 调用有参数的子程序

模块之间通过 public 和 extern 关键字来传递参数，如果符号名不一样，VS2019 会报出无法“解析的外部符号，函数中引用了改符号”的错误，程序编译错误，不能运行。

## 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

使用 LEA 指令代替 OFFSET 获取地址，使用 LEA 替换 ADD 完成加法运算，减少实际运行的指令数目，减少不必要的指令，采用移位运算代替乘除法运算，尽可能减少循环中指令条数，从而提高程序执行效率。

#### 2. 约束条件、算法与程序结构的影响

约束条件不同，设计出来的程序也不同，要根据条件选择合适的数据类型来定义变量，以及指

# 汇编语言程序设计实验报告

令的使用，根据数据的大致范围确定数据的类型，根据数据有无符号选择合适的操作指令，根据需要选择 ADD、ADC 等等。

算法不同，程序结构也不同，比较字符串时可以先判断两个字符串的长度，也可以直接进行比较，这两种不同的算法逻辑不同，结构也不同，功能一致。逻辑清晰的算法会比较好的程序结构。

实验 3 采用多模块，相较其他只有一个主模块的实验来说，实验 3 执行时寻址会略微慢，但是适合多人合作，并且可以控制各模块共享的数据，多模块更有利于调试和修改程序功能，使得整个工程的结构更加清晰。

## 3. 编程环境的影响

在不同的机器上运行相同的程序观察不同程序上运行的差异，在不同的架构上编写、运行程序，体验不同的变成环境。

## 1.4.2 实验记录与分析

### 1. 优化实验的效果记录与分析

采用 GetTickCount 函数来进行计时，并且通过外加循环的方式来增强优化前后的对比效果，程序运行结果如图 1.17 所示。

运行结果即为所花费时间，例如：图 1.17 运行结果表示花费 6469ms。

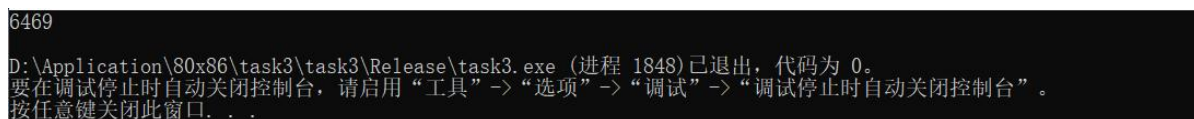


图 1.17 运行结果

为了能够跟准确的确定优化的效果，采用多次结果取平均值，利用平均值来代表整体运行时间。得到的结果如表 1.1 所示。

通过观察表格，我们可以发现，VS2019 在 release 模式下要比在 debug 模式下的运行效率高一些，release 模式会自动采用一些指令优化提升执行效率。优化 1 将原本的 OFFSET 语句换成 LEA 语句，优化 2 将原本的 ADD 指令改写成 LEA 指令，可以观察到指令的选择对于程序执行效率有一定的影响，选择合适的指令能够提高效率。

表 1.1 优化前后运行时间对比

\	1	2	3	4	5	平均值/ms
Debug 模式	6578	6765	6578	6703	6453	6615.4
Release 模式	6359	6534	6500	6469	6515	6475.4
优化 1	6547	6422	6319	6515	6421	6444.8
优化 2	6422	6437	6344	6563	6360	6425.2

### 2. 不同约束条件、算法与程序结构带来的差异

本实验中，采用了不同的约束条件、程序结构，利用了宏定义以及子程序的设计方法。其中宏定义实际上采用的是替换的方式，在编译时就替换成宏定义了，而子程序则是通过设置地址的方式。



# 汇编语言程序设计实验报告

对于不经常使用的代码使用子程序更高效,对于频繁使用的并且代码量比较小的部分采用宏定义则更加高效。

```
USER_CMP MACRO STR1, STR1_LEN, STR2, STR2_LEN, CMPRES ;CMPRESULT用来记录比较结果, 1表示不等, 0表示等
    PUSHAD ;保护寄存器
    MOV EAX, STR1_LEN
    XOR EAX, STR2_LEN ;用异或判断两个数相同
    JNZ NOT_EQUAL1 ;两个串长度不等
    MOV ECX, STR1_LEN ;比较次数
    MOV ESI, 0
CMP_LP1:
    MOV AL, STR1[ESI]
    XOR AL, STR2[ESI]
    JNZ NOT_EQUAL1
    INC ESI
    DEC ECX
    JG CMP_LP1
EQUAL1: MOV CMPRES, 0 ;相等
    JMP END_CMP1
NOT_EQUAL1: MOV CMPRES, 1 ;不等
END_CMP1: POPAD
ENDM
```

图 1.18 宏定义部分代码

```
invoke printf, offset print, offset MSGCOPY ;完成MIDF的输出工作
invoke DISPLAY, OFFSET MIDF, NUMOFMIDF
invoke scanf, offset PRINTCHAR, offset CHOICE ;这里为了读去上面多读到的回车
INPUTCHOICE:
    invoke printf, offset PRINTFRONT, offset MSGTIP
    invoke scanf, offset PRINTCHAR, offset CHOICE
    cmp CHOICE, 'R'
```

图 1.19 子程序部分代码

## 2. 几种编程环境中程序的特点记录与分析

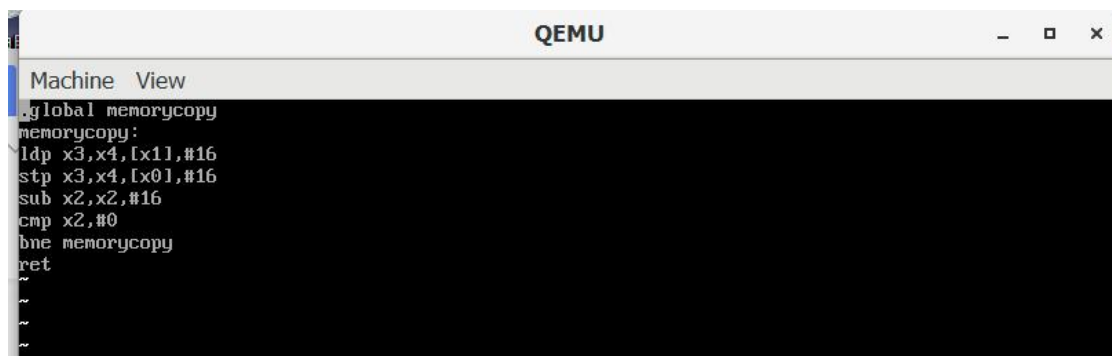
表 1.2 不同机器运行时间对比

\	1	2	3	4	5	平均值/ms
Magicbook	6578	6765	6578	6703	6453	6615.4
联想小新	6535	6241	6522	6891	6313	6500.4

在不同的机器上运行相同的代码,通过观察上表中记录的信息,可以看出,在性能差异不大的电脑上执行相同的程序,运行时间差异没有明显的差异,表明在硬件差异不大的情况下,可以在软件层面下功夫,获得较大的性能提升。

DOSBOX 和 VS2019 均采用 intel 的架构,都是复杂指令集,其中 DOSBOX 采用的 16 位,VS2019 中采用的是 32 位,除了虚拟内存大小、指令长度等不同,其他相似,而 Armv8 架构下采用精简指令集,其中很多寄存器命名不同,并且指令名称也略有差异。

# 汇编语言程序设计实验报告

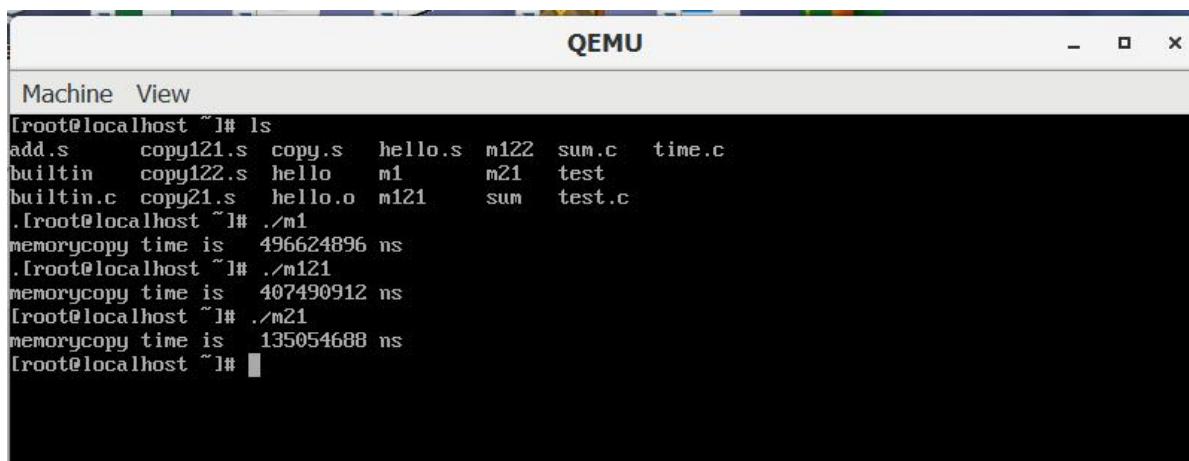


```
Machine View
.global memorycopy
memorycopy:
    ldp x3,x4,[x1],#16
    stp x3,x4,[x0],#16
    sub x2,x2,#16
    cmp x2,#0
    bne memorycopy
    ret
~
~
~
```

图 1.20 Arm 实例程序

Machine View	Machine View	Machine View
<pre>.global memorycopy memorycopy: sub x1,x1,#1 sub x0,x0,#1 lp: ldrb w3,[x1],#1! ldrb w4,[x1],#1! str w3,[x0],#1! str w4,[x0],#1! sub x2,x2,#2 cmp x2,#0 bne lp ret ~</pre>	<pre>.global memorycopy memorycopy: sub x1,x1,#1 sub x0,x0,#1 lp: ldrb w3,[x1],#1! ldrb w4,[x1],#1! ldrb w5,[x1],#1! ldrb w6,[x1],#1! str w3,[x0],#1! str w4,[x0],#1! str w5,[x0],#1! str w6,[x0],#1! sub x2,x2,#4 cmp x2,#0 bne lp ret ~</pre>	<pre>.global memorycopy memorycopy: sub x1,x1,#1 sub x0,x0,#1 lp: ldrb w3,[x1],#1! ldrb w4,[x1],#1! str w3,[x0],#1! str w4,[x0],#1! sub x2,x2,#2 cmp x2,#0 bne lp ret ~</pre>

图 1.21 三种复制代码



```
QEMU
Machine View
[root@localhost ~]# ls
add.s      copy121.s  copy.s     hello.s    m122       sum.c      time.c
builtin    copy122.s  hello      m1         m21        test
builtin.c  copy21.s   hello.o    m121       sum        test.c
[root@localhost ~]# ./m1
memorycopy time is 496624896 ns
[root@localhost ~]# ./m121
memorycopy time is 407490912 ns
[root@localhost ~]# ./m21
memorycopy time is 135054688 ns
[root@localhost ~]#
```

图 1.22 三种复制的效率差异

通过观察输出的时间，可以看出指令不同时，程序执行效率的差异也比较大，有时甚至能够达到几倍的性能差异。

# 汇编语言程序设计实验报告

---

## 1.5 小结

通过本次实验，体验多模块编程，学会了 `public`、`extern` 关键字在 C 语言以及汇编程序之间的使用。经过实验 3，编写子程序，以及观察反汇编代码，对于参数传递以及局部变量在底层的实现有了更加深刻的认识，并且认识后堆栈传递参数广泛应用于高级语言。在实验 3.1 中编写了宏定义以及子程序，通过观察反汇编，认识到了宏和子程序之间的差异。

寄存器的个数很有限，在多模块中等需要很多变量的程序中，要合理使用寄存器，或者采用全局变量和局部变量等来传递保存数据，采用这种方式还可以保护寄存器，避免调用某些函数时修改寄存器引起不可预知的错误。

由于 x86 采用的是复杂指令集，指令比较高效，比较消耗资源，而 Armv8 采用的是精简指令集架构，指令执行起来比较慢，节约资源。

通过本次实验，学会了调试工具的使用，通过断点、寄存器、内存、反汇编、监视等窗口观察程序运行过程中的变化。

## 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

实方式下，当发生中断时，CPU 会根据中断矢量表中记录的地址执行相应的中断处理程序，在实方式下，中断矢量表大小为 1KB，起始位置固定在从物理地址 0 开始。

为了能够检查中断处理程序是否已经安装，从而避免重复安装，需要我们实现驻留。代码一旦确定下来，程序内的偏移就确定下来，进行安装时将此偏移记录，通过对比记录的偏移与实际偏移，判断中断处理程序是否已经安装过。

接管 8 号时钟中断，需要我们重写 8 号中断处理程序，并将修改中断矢量表中对应的位置。为了保证原功能仍能实现，使用 35H 号 DOS 功能调用取中断号是 8 的中断处理程序入口地址，在新中断处理程序中调用原 8 号中断处理程序，通过 25H 号 DOS 功能调用设置中断向量，将新的中断处理程序的 CS、IP 存入到中断矢量表。

为了实现驻留，计算需要驻留程序的节长度，并调用 31H 号 DOS 功能调用实现内存驻留。

利用 DOS 系统下的 TD 进行调试，单步调试中断处理程序，观察中断矢量表中的内容以及寄存器值的变化。

##### 2. 反跟踪程序的设计思想与实验方法

设计反跟踪的目的是为了放置他人轻易解读程序的功能，可以采用多种方式来增加根据反汇编解读程序的复杂度。在本次实验中，采用动态修改执行代码、穿插无关代码等反跟踪方法，对于定义的 f 计算公式，在汇编实现中增加和计算 f 毫无关联的代码或无用代码。将部分代码的机器码存储到数段段，在代码段中通过加载数据段和更改 IP 实现动态修改程序。编写完成后利用相关工具对生成的可执行文件进行反汇编，生成汇编代码，根据汇编代码尝试解读原程序。

为了另外的实现保护，需要将在源程序数据段中定义的密码在汇编之后变成密文，在本实验中



# 汇编语言程序设计实验报告

采用异或逻辑对原文进行加密，在反汇编代码中难以简单地根据数据段中存储的内容得到密码。

## 3.指令优化及程序结构的实验方法

为了突显指令优化的效果，将整个程序执行多次记录时间，并求时间的平均值作为性能的代表，为了实现指令优化，需要对汇编代码进行分析，观察是否可以采用高效的指令，例如乘法可以分析是否可以用移位运算代替，观察在循环中是否可以简化指令条数从而提高效率，是否某些变量的寻址方式可以进行优化。在本次实验中，采用 LEA 指令代替寻址加 ADD 指令，采用移位运算代替部分乘法，并且将循环中的部分代码简化，记录每次执行的时间。

将整个程序拆分成实现不同功能的模块，每个模块完成不同的功能，利用 public、extern 关键字实现模块间数据的共享，将字符串比较功能改写成宏定义的形式。对于子程序与主程序之间参数的传递采用堆栈法，宏定义由参数表传递，并通过反汇编代码等观察宏定义和子程序之间差别。

将整个程序采用 C 语言和汇编语言混合编程，观察 C 语言和汇编混合编程与纯汇编语言之间的区别，通过反汇编、寄存器窗口、内存窗口等观察 C 语言和汇编语言如何声明外部变量和函数，如何翻译，如何传递参数等。

## 2.3.2 实验记录与分析

### 1.中断处理程序的特别之处

中断处理程序可以使用 INT 指令显式进行调用，在 DOSBOX 中使用 35H 号功能调用实现获取中断矢量、25H 号功能调用实现设置中断矢量，31H 号功能调用实现内存驻留，4CH 号功能调用实现结束中断程序。中断处理程序特别之处主要在于要将 CS、IP 存入到中断矢量中，而一般的程序则只需要使用函数名进行调用。

```
BEGIN: PUSH    CS
      POP     DS
      MOV     AX, 3508H      ;获取原08H的中断矢量
      INT     21H           ;系统功能调用
      LEA     CX, NEWOSH
      ;MOV     CX, 0FH
      CMP     BX, CX        ;判断是否相等
      JE      EXIT
      MOV     OLD_INT, BX    ;保存中断矢量
      MOV     OLD_INT + 2, ES
      MOV     DX, OFFSET NEWOSH ;设置新的08中断矢量
      MOV     AX, 2508H
      INT     21H           ;获取原08H的中断矢量
      INT     21H           ;系统功能调用
NEXT:  MOV     AH, 0        ;等待按键
      INT     16H
      CMP     AL, 'q'
      JNE     NEXT
      MOV     DX, OFFSET BEGIN + 15 + 2000 ;实现内存驻留
      SHR     DX, 4
      ADD     DX, 10H
      MOV     AL, 0
      MOV     AH, 31H       ;驻留
      INT     21H
EXIT:  MOV     AH, 4CH
      INT     21H
```

图 2.1 中断程序截图

在调试中断处理程序调试过程中需要观察 0:4\*n 对应的内存中存放的数据，即对应的中断处理

# 汇编语言程序设计实验报告

程序的入口地址，时钟中断处理在内存中的位置为  $0:4*8$ 。35H 号中断处理程序，将中断处理程序偏移地址存入 BX，将中断处理程序段地址存入 ES 中。

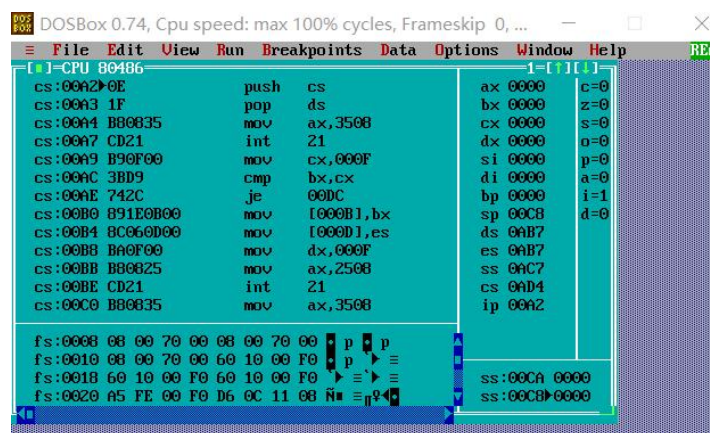


图 2.2 中断处理程序调试

运行中断处理程序的结果如下图所示，当第一次运行 timer 时，在右上侧显示出了当前时间，当按下 q 键后退出程序，并且退出程序后，该时间仍能显示在命令行窗口右上角，并且自动更新，可以看出新 08H 号中断处理程序实现了内存驻留，当再次执行 timer 时，自动退出，即不需要手动按下 q 键程序就自动退出了。

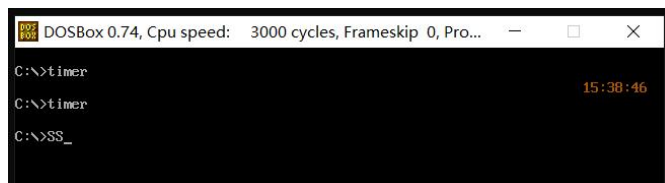


图 2.3 中断处理程序运行结果

## 2.反跟踪效果的验证

‘l’ 对应的 ASCII 码为 6CH，将其和 ‘y’ 字符异或，得到 15H，通过观察数据段 PASSWD 内容以及实际内存中的数据可以观察到实现了对于密码的加密。



图 2.4 数据段变量声明以及内存中真实存储的数据

动态修改执行代码通过将部分机器码写入到内存，通过调用 VirtualProtect 系统函数对于修改代码段数据开放权限，使得可以在运行时动态修改代码，增加了通过反汇编跟踪程序的难度，通过下面的图片可以看出，在代码段开辟了一段空间用来存放要动态写入的机器码，在运行到代码段开辟的空间前已经将动态代码写入，随着 IP 寄存器的变化，开始执行动态写入的机器码。

# 汇编语言程序设计实验报告



图 2.5 动态修改代码机器码存入到内存

```
MOV EAX, LEN_CODE
MOV EBX, 40H
LEA ECX, CopyHere
invoke VirtualProtect, ECX, EAX, EBX, offset OLDPROTECT ;会修改EDX的值
MOV ECX, LEN_CODE
LEA EDI, CopyHere
MOV ESI, offset MACHINE_CODE

CopyCode:
MOV AL, [ESI]
MOV [EDI], AL
INC ESI
INC EDI
LOOP CopyCode

CopyHere:
DB LEN_CODE DUP(0) ;保护1: 实现了动态修改子程序
```

图 2.6 动态修改程序代码

```
270: MOV EAX, LEN_CODE
00F91378 B8 05 00 00 00 mov     eax, 5
271: MOV EBX, 40H
00F9137D BB 40 00 00 00 mov     ebx, 40h
272: LEA ECX, CopyHere
00F91382 8D 0D AD 13 F9 00 lea     ecx, ds:[0F913ADh]
273: invoke VirtualProtect, ECX, EAX, EBX, offset OLDPROTECT ;??? ?EDX??
00F91388 68 43 99 0A 01 push    offset OLDPROTECT (010A9943h)
00F9138D 53 push    ebx
00F9138E 50 push    eax
00F9138F 51 push    ecx
00F91390 E8 11 01 00 00 call     _VirtualProtect@16 (0F914A6h)
274: MOV ECX, LEN_CODE
00F91395 B9 05 00 00 00 mov     ecx, 5
275: LEA EDI, CopyHere
00F9139A 8D 3D AD 13 F9 00 lea     edi, ds:[0F913ADh]
276: MOV ESI, offset MACHINE_CODE
00F913A0 BE 3E 99 0A 01 mov     esi, offset MACHINE_CODE (010A993Eh)
277:
278: CopyCode:
279: MOV AL, [ESI]
00F913A5 8A 06 mov     al, byte ptr [esi]
280: MOV [EDI], AL
00F913A7 8B 07 mov     byte ptr [edi], al
281: INC ESI
00F913A9 46 inc     esi
282: INC EDI
00F913AA 47 inc     edi
283: LOOP CopyCode
00F913AB E2 F8 loop     LOOPTOTAL+2Dh (0F913A5h)
00F913AD 00 00 add     byte ptr [eax], al
00F913AF 00 00 add     byte ptr [eax], al
00F913B1 00 51 8B add     byte ptr [ecx-75h], dl
00F913B4 C8 S3 C3 0A enter   0C383h, 0Ah
```

图 2.7 动态修改程序反汇编代码

## 3.跟踪与破解程序

利用静态反汇编工具 W32Dasm 对可执行文件进行反汇编，观察到源程序中具有很多中断处理指令，并且只包括了汇编指令以及地址，消去了变量名，最左边为代码段地址，右边为对应的汇编指令，如图 2.8 所示。

利用动态反汇编工具 OllyDbg 工具对可执行文件进行单步调试（如图 2.10 所示），通过观察内存数据段可以查看数据段存储的内容，结合代码段中比较部分的代码，找到数据段中用户、密码

# 汇编语言程序设计实验报告

存放的正确位置。通过比对 ASCII 码，初步判断可能的密码，找到代码段中对该变量地址内容的处理，分析可能对密码采用的加密，依次来获取可能的密码。

对于 F 的计算表达式破解，则要复杂得多，表达式采用了动态执行修改的过程，根据代码段表达式的可能的处理，进行分析，尽可能分析 F 的正确表达式。

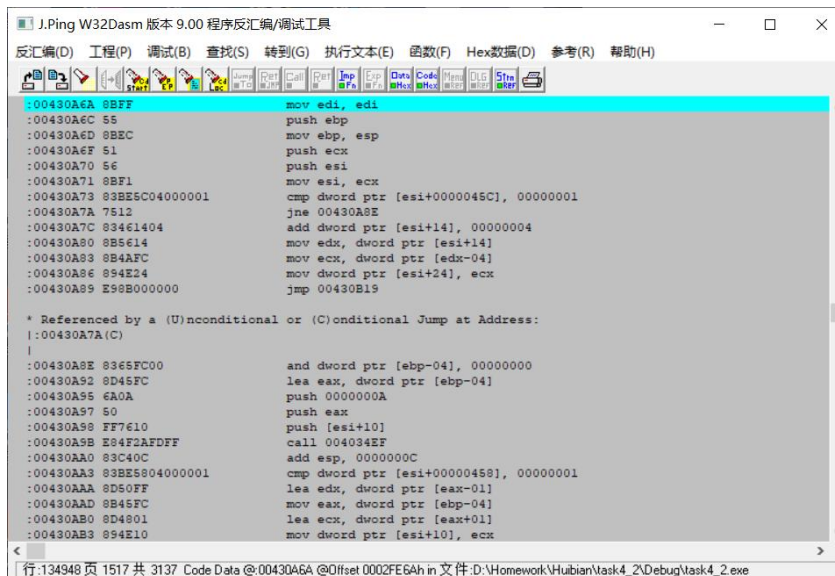


图 2.8 反汇编代码

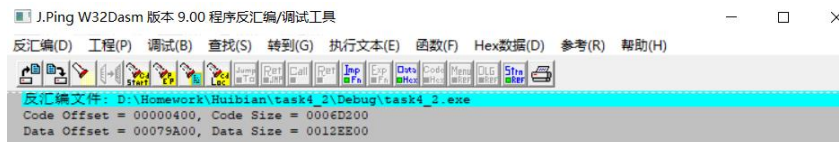


图 2.9 反汇编代码头部信息

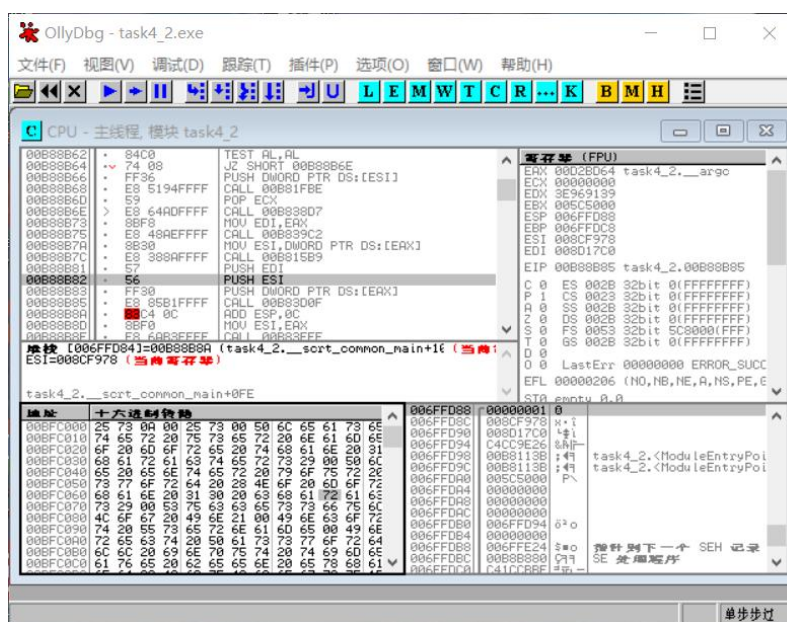


图 2.10 动态反汇编信息



# 汇编语言程序设计实验报告

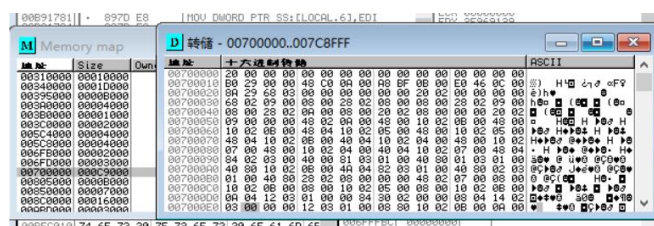


图 2.11 动态反汇编内存

## 4.特定指令及程序结构的效果

PUSHAD、POPAD，用来保护寄存器，在汇编中由于可以直接操作寄存器，所以要时刻注意寄存器的变化。ADC、SUB，带进位的加法和带借位的减法，在汇编中结合标志寄存器中的进位标志，完成带进位的加法、带借位的减法。JG、JL 等实现有条件的跳转，在高级语言中通过 if 条件语句实现跳转。LOCAL 声明局部变量，声明只能在子程序中使用的变量。MACRO、ENDM 用来进行宏定义。

```
USER_CMP MACRO STR1, STR1_LEN, STR2, STR2_LEN, CMPRES ;CMPRESULT用来记录比较结果, 1表示不等, 0表示等
    PUSHAD ;保护寄存器
    MOV EAX, STR1_LEN
    XOR EAX, STR2_LEN ;用异或判断两个数相同
    JNZ NOT_EQUAL1 ;两个串长度不等
    MOV ECX, STR1_LEN ;比较次数
    MOV ESI, 0
CMP_LP1:
    MOV AL, STR1[ESI]
    XOR AL, STR2[ESI]
    JNZ NOT_EQUAL1
    INC ESI
    DEC ECX
    JG CMP_LP1
EQUAL1: MOV CMPRES, 0 ;相等
    JMP END_CMP1
NOT_EQUAL1: MOV CMPRES, 1 ;不等
END_CMP1: POPAD
ENDM
```

图 2.12 汇编语言宏定义

```
invoke printf, offset print, offset MSGCOMPUTE
;完成相应的复制工作
MOV EAX, ADR1+4
CALL EAX
invoke printf, offset print, offset MSGCOPY
;完成MIDF的输出工作
invoke DISPLAY, OFFSET MIDF, NUMOFMIDF
invoke scanf, offset PRINTCHAR, offset CHOICE ;这里为了读去上面多读到的回车
INPUTCHOICE:
    invoke printf, offset PRINTFRONT, offset MSGTIP
    invoke scanf, offset PRINTCHAR, offset CHOICE
```

图 2.13 汇编语言子程序调用

## 2.4 小结

通过本次实验熟悉了指令的使用，并且通过指令优化，明白了汇编指令在底层执行效率上的差异，理解了中断矢量表的概念，掌握实方式下中断处理程序的编制与调试方法，进一步熟悉内存的一些基本操纵技术，学会了一些反跟踪方法，以及一些程序跟踪的方法，并且明白了宏定义、子程

## 汇 编 语 言 程 序 设 计 实 验 报 告

---

序等程序结构的实现方法，并能够根据需要设计相应的比较高效的汇编代码，接触了 DOSBOX 虚拟机以及 W32asm、OllyDbg 等反汇编工具。

在中断处理程序中，由于虚拟机的原因，当内存中的数据更改后，可能在图形化页面中查找到的数据仍未原来的数据，并且设置的节长度要比原计算所得的节长度略微长一点，在运行程序的过程中，可能因为这几个问题不能得到想要的结果。

## 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 可以通过选择 x86 或者 x64 来选择是 32 位还是 64 位，x64 比 x86 寄存器多了一倍，可使用的内存更多了，除此之外没有特别大的不同，以下仅说明使用 x86 的 VS2019 中的一些特色。

##### 1. 寄存器

设置断点，进入调试状态下，点击调试->窗口->寄存器。（如下图所示）

寄存器这里可以右键选择显示的内容，这里可以选择 CPU 得到八个通用寄存器以及 EIP 和 EFL，选择标志可以得到八个标志寄存器。

通过单步调试可以观察各种寄存器的值得变化，监督寄存器使用是否正常，标志寄存器中的值会自动进行变化，单步调试时，如果寄存器窗口中某个值显示为红色，表示与上一步值不同。

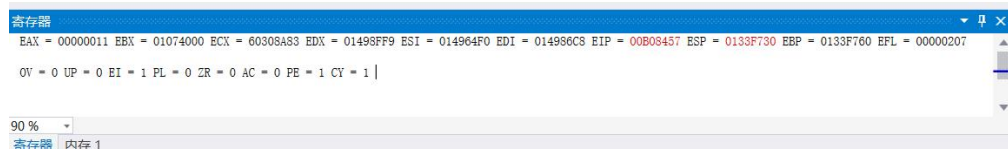


图 3.1 寄存器窗口



图 3.2 设置寄存器窗口显示内容

##### 2. 内存

设置断点，进入调试状态，点击调试->窗口->内存。（如下图所示）

内存窗口上方的地址栏可以输入地址，下方显示以地址栏中的地址开始的若干字节，并且窗

# 汇编语言程序设计实验报告

口右方简略按照显示各字节表示的内容。

如果在地址栏中输入十进制或者八进制数，会自动转换成十六进制，寻找到内存中的位置进行显示。如果在地址栏中输入某个寄存器，会将寄存器中存储的内容放到地址栏中，寻找内容中的位置进行显示。如果在地址栏中输入某一个变量，并且该变量此时存在，那么就会将变量表示的值变成十六进制数，寻找内容中的位置并显示。地址栏中输入方式没有提示，容易出错，且不能使用 offset 等。

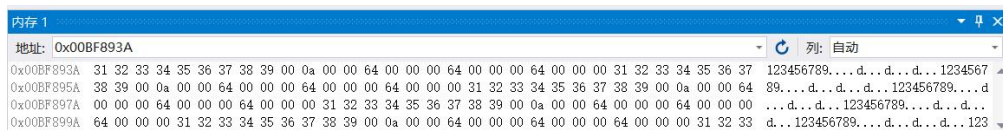


图 3.3 内存窗口

## 3.反汇编

设置断点，进入调试状态，点击调试->窗口->反汇编。（如下图所示）

上方地址栏显示现在当前所在函数的函数声明，例如 `main(void)` 表示当前在主函数中。可以设置显示行号、地址、符号名等。勾选显示符号名会将变量地址换成程序中对应的变量名。

通过反汇编可以观察程序在底层的具体实现的逻辑，更容易理解程序逻辑。

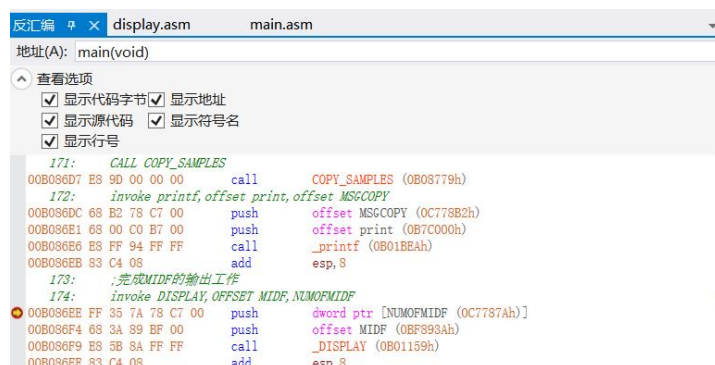


图 3.4 反汇编窗口

## 4.监视

设置断点，进入调试状态，点击调试->窗口->监视。（如下图所示）

监视可以对第一个变量的值进行监视，也可以对一个变量的地址进行查找，和内存搭配使用，可以更方便的确定内容中存放的内容。监视窗口添加要监视的项中可以输入寄存器，即可以监视寄存器，当输入一个常量时，会显示值为该常量，但此时无意义。

所监视的变量存在时，监视窗口会显示出对应的值，不存在时，会显示一个 X 号。

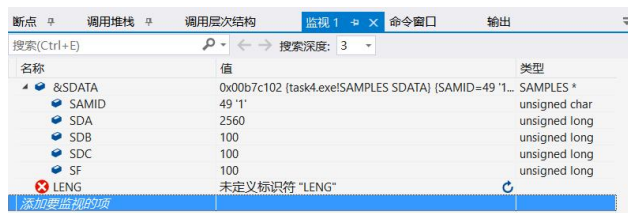


图 3.5 监视窗口



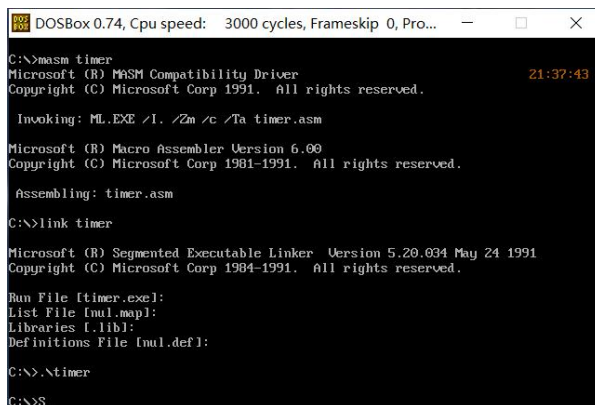
# 汇编语言程序设计实验报告

## 3.2.2 DOSBOX 下的工具包

DOSBOX 是 16 位虚拟操作系统，其中的寄存器都是 16 位，

### 1. 命令行窗口

DOSBOX 采用命令行的方式，不同于 VS2019 图形化的操作，在 DOSBOX 下，利用 `masm` 命令进行编译、`link` 命令进行链接、`.timer` 来运行名为 `timer` 的可执行文件。



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
C:\>masm timer
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1991. All rights reserved.

Invoking: ML.EXE /I. /Zm /c /Ta timer.asm

Microsoft (R) Macro Assembler Version 6.00
Copyright (C) Microsoft Corp 1981-1991. All rights reserved.

Assembling: timer.asm

C:\>link timer

Microsoft (R) Segmented Executable Linker Version 5.20.034 May 24 1991
Copyright (C) Microsoft Corp 1984-1991. All rights reserved.

Run File [timer.exe]:
List File [nul.map]:
Libraries [libl]:
Definitions File [nul.def]:

C:\>.timer
C:\>S_
```

图 3.6 DOSBOX 命令行窗口

### 2. TD 窗口

在 DOSBOX 下，通过 TD 进入调试窗口，得到如下页面，包含了代码段、数据段、寄存器、标志寄存器、段寄存器等。在该窗口下，实现了简单的图形化，尽管简洁，但是基本包含了调试过程中常用的内容。

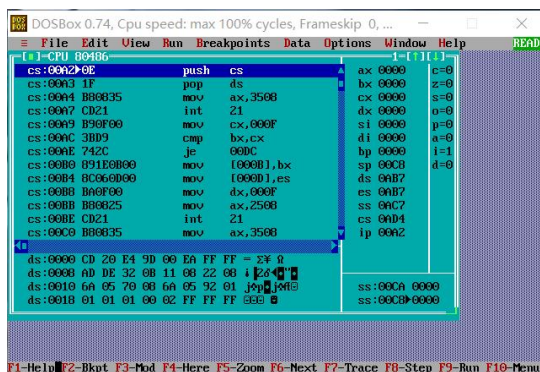


图 3.7 DOSBOX 下 TD 调试窗口

查找某个地址存放的数据，利用快捷键 `ctrl+G` 或者鼠标右击选择 `Goto`，输入要查找的地址，可以显示对应地址的内存中的内容，通过查找 `0:4*n`，可以快速定位中断矢量表中某个中断处理程序的 CS、IP。

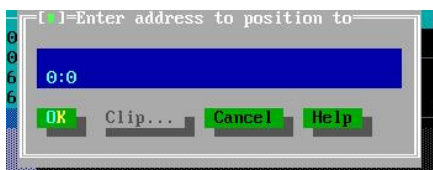


图 3.8 DOSBOX 下查找

# 汇编语言程序设计实验报告

## 3. 寄存器

左侧 ax、bx 等为通用寄存器，ds、ss 等为段寄存器，右侧 c、z 等为标志寄存器。在寄存器右侧都标注了当前寄存器的值，可以在调试过程中观察各个寄存器的值的变化。

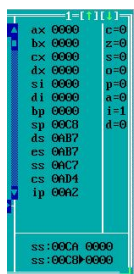


图 3.8 DOSBOX 的寄存器显示

## 4. 内存

cs:开头的部分为代码段，左侧为逻辑地址以及机器码，右侧为机器码对应的汇编代码，ip 寄存器中存放着 cs 段中要执行的下一条指令。ds:开头的部分为数据段，左侧为地址，中间为内存中的数据，右侧为数据对应的 ascii 码，通过单步运行程序，可以观察汇编代码做了哪些事情。

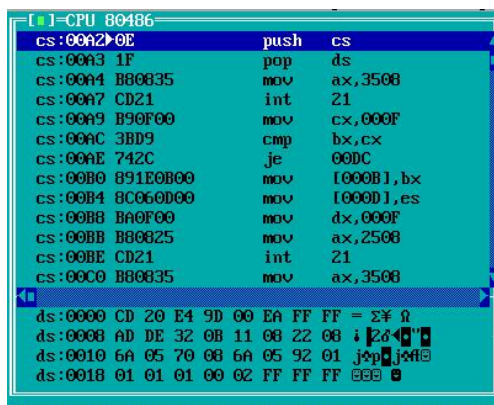


图 3.9 DOSBOX 内存显示

## 3.2.3 QEMU 下 ARMv8 的工具包

QEMU 下，全部展示为命令行窗口，所有操作均采用命令，使用 cat 命令显示最后一屏内容，gcc 命令进行编译，vi 或 vim 命令进行文本编辑，as 命令将汇编文件生成二进制目标文件，ld 进行链接，gdb 命令进行调试等，在 gdb 命令调试下，通过 b 设置断点，r 运行程序，n 单步执行程序等命令可以用来对程序执行调试，在 gdb 下，查找内存中某个数据的值或者寄存器比较麻烦。（如图 3.10 所示）

# 汇编语言程序设计实验报告

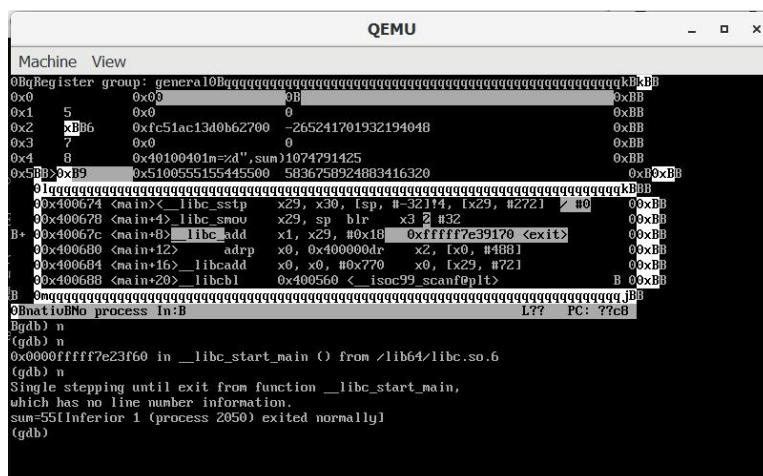


图 3.10 命令行下显示寄存器

## 3.3 小结

VS2019 为调试汇编程序提供了很多帮助性的工具，例如反汇编、内存、监视窗口、寄存器等，通过设置断点的方式，可以进入调试使用这些工具，对于观察底层逻辑非常有帮助。而且 VS2019 的界面非常丰富，可以看源码的同时观察寄存器、内存等等，既可以按照反汇编单步执行也可以按照原汇编程序单步执行，同时使用 VS2019 进行调试，调试方法可以迁移到其他更高级的语言上。总体来看，VS2019 功能丰富、界面美观、操作简便。

DOSBOX 在命令行下进行编译、链接、执行，TD 工具下可以观察内存、寄存器等数据，还可以进行单步调试程序。DOSBOX 基本能够满足一般的调试程序，但是 DOSBOX 作为虚拟机仍存在一些小的问题，例如：尽管实际内存中的数据已经改变，但是当查找时，数据可能仍未发生变化。DOSBOX 对于观察实方式下的中断处理程序提供了很大的方便。总体来说，DOSBOX 功能完善、界面简洁、操作容易。

QEMU 下的 Armv8，在 openEuler 虚拟操作系统下，只能使用命令行进行操作，包括编译、执行、调试等，尤其调试的过程相较图形化的调试过程复杂许多，初次使用者难以很快上手。GDB 作为很多编译器底层调试的实现工具，具有调试的所有功能，可以进行单步调试，查看寄存器等操作。总的来说，QEMU 功能齐全，界面单一，操作复杂。

# 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]王爽. 汇编语言. 北京: 清华大学出版社, 2003