

汇编语言与逆向技术实验报告

Lab3-整数数组的冒泡排序

学号：2112060 姓名：孙蓓 专业：信息安全

一、实验目的

- 1、熟悉汇编语言的整数数组；
- 2、熟悉基址变址操作数、相对基址变址操作数；
- 3、掌握排序算法的底层实现细节

二、实验环境

Windows 记事本的汇编语言编写环境

MASM32 编译环境

Windows 命令行窗口

三、实验原理

同学们已经使用 C++ 高级语言实现了搜索和排序算法。汇编语言能够清晰地看到底层的实现细节，使得汇编语言为研究算法提供了另一个视角。20 世纪最著名的算法学家之一 Donald Knuth 在其出版的《The Art of Computer Programming》中也是使用汇编语言来编写例子的。

排序算法中，汇编语言的基址变址寻址方式和相对基址变址寻址方式起到了重要的作用。

基址变址（base-index）操作数把两个寄存器的值相加，得到一个偏移地址。两个寄存器分别称为基址寄存器（base）和变址寄存器（index）。格式为[base + index]，例如 `mov eax, [ebx + esi]`。在例子中，`ebx` 是基址寄存器，`esi` 是变址寄存器。基址寄存器和变址寄存器可以使用任意的 32 位通用寄存器。

相对基址变址（based-indexed with displacement）操作数把偏移、基址、变址以及可选的比例因子组合起来，产生一个偏移地址。常见的两种格式为：[base + index + displacement]和 `displacement[base + index]`，例子如下：

```
table dword 10h, 20h, 30h, 40h
```

```
row_size = ($ - table)
```

```
dword 50h, 60h, 70h, 80h
```

```

        dword 90h, 0a0h, 0b0h, 0c0h

mov ebx, row_size

mov esi, 2

mov eax, table[ebx + esi * 4]

```

`table` 是一个二维数组，共 3 行 4 列。`ebx` 是基址寄存器，相当于二维数组的行索引，`esi` 是变址寄存器，相当于二维数组的列索引。

冒泡排序算法 (Bubble Sort) 的过程是从位置 0 和 1 开始比较每对数据的值，如果两个数据的顺序不对，就进行交换。如果一遍处理完之后，数组没有排好序，就开始下一次循环。在最多完成 $n-1$ 次循环后，数组排序完成。

四、 实验内容

本次实验要求编写汇编程序 `bubble_sort.asm`，功能是将 Windows 命令行输入的 10 个 1 万以内的十进制无符号整数，进行排序，然后输出在 Windows 命令行中。10 个无符号整数之间用逗号","或者空格" "分割。

使用 `StdIn` 函数获得用户输入的十进制整数序列。`StdIn` 函数的定义在 `\masm32\include\masm32.inc`，库文件是 `\masm32\lib\masm32.lib`。`StdIn` 函数的定义 “`StdIn PROTO :DWORD, :DWORD`”，有两个参数，第一个是内存存储空间的起始地址，第二个是内存存储空间的大小。

使用 `StdOut` 函数在 Windows 命令函中输出排好序的十进制整数序列。`StdOut` 函数的定义在 `\masm32\include\masm32.inc`，库文件是 `\masm32\lib\masm32.lib`。`StdOut` 函数的定义 “`StdOut PROTO :DWORD`”，只有一个参数，是内存存储空间的起始地址。

使用 `ml` 和 `link` 程序将源代码编译、链接成可执行文件 `bubble_sort.exe`。

五、 源代码和注释

```

.386

.model flat, stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc

```

```

include\masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
includelib\masm32\lib\user32.lib
includelib \masm32\lib\msvcrt.lib
scanf PROTO C:DWORD,:vararg
.data
ArrayA dword 10 dup(?);数组
infmt byte '%d,%d,%d,%d,%d,%d,%d,%d,%d,%d',0;格式
szText db 100 dup(0)
szCharsFormat db 'After Bubble Sort :
ArrayA :%d,%d,%d,%d,%d,%d,%d,%d,%d,%d',0
.code
BubbleSort proc uses eax ecx esi,
    pArray:ptr dword,
    Count:dword
local @exchange:dword
    mov ecx,Count
    dec ecx
    mov @exchange,0
L1:
    push ecx
    mov esi,pArray ;赋地址
    mov @exchange,0
L2:
    mov eax,[esi]
    cmp [esi+4],eax ;比较
    jge L3 ;如果大于或等于, 执行 L3
    xchg eax,[esi+4] ;小于就交换
    mov [esi],eax
    mov @exchange,1

```

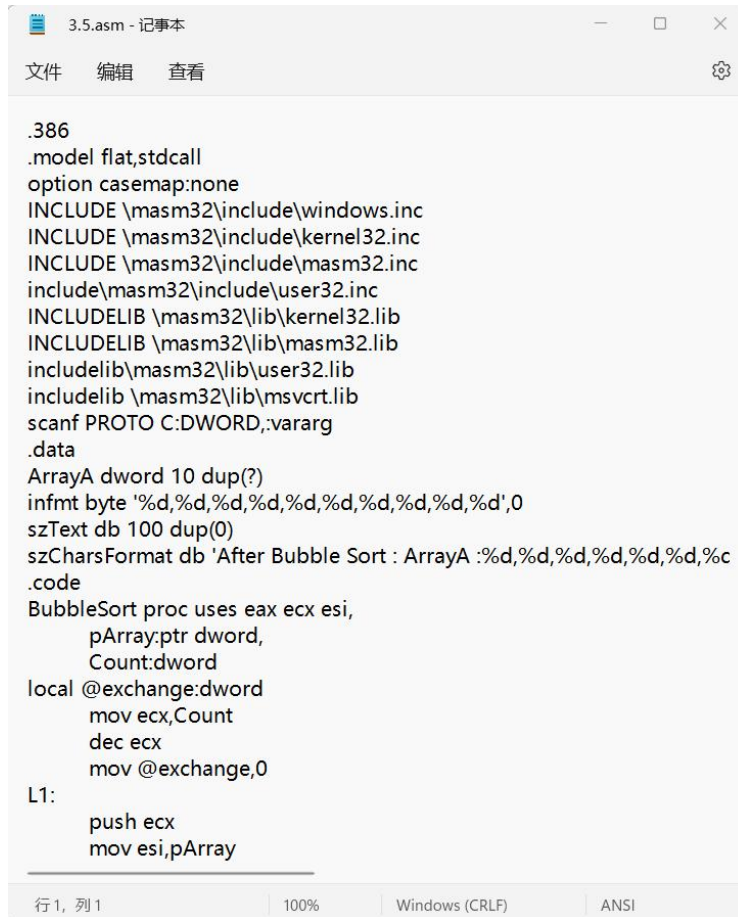
```

L3:
    add esi,4    ;找下一个数
    loop L2
    cmp @exchange,0
    je L4        ;大于执行到 L4
    pop ecx
    loop L1
L4:ret
BubbleSort endp
start:
    invoke scanf,ADDR infmt,ADDR ArrayA        ;输入
    invoke scanf,ADDR infmt,ADDR ArrayA+4
    invoke scanf,ADDR infmt,ADDR ArrayA+8
    invoke scanf,ADDR infmt,ADDR ArrayA+12
    invoke scanf,ADDR infmt,ADDR ArrayA+16
    invoke scanf,ADDR infmt,ADDR ArrayA+20
    invoke scanf,ADDR infmt,ADDR ArrayA+24
    invoke scanf,ADDR infmt,ADDR ArrayA+28
    invoke scanf,ADDR infmt,ADDR ArrayA+32
    invoke scanf,ADDR infmt,ADDR ArrayA+36
    invoke BubbleSort,addr ArrayA,10
    invoke wsprintf,addr szText,addr
szCharsFormat,ArrayA,ArrayA+4,ArrayA+8,ArrayA+12,ArrayA+16,ArrayA+20,
ArrayA+24,ArrayA+28,ArrayA+32,ArrayA+36,ArrayA+40
    invoke StdOut,offset szText ;输出
    invoke ExitProcess,0
end start

```

六、 实验步骤

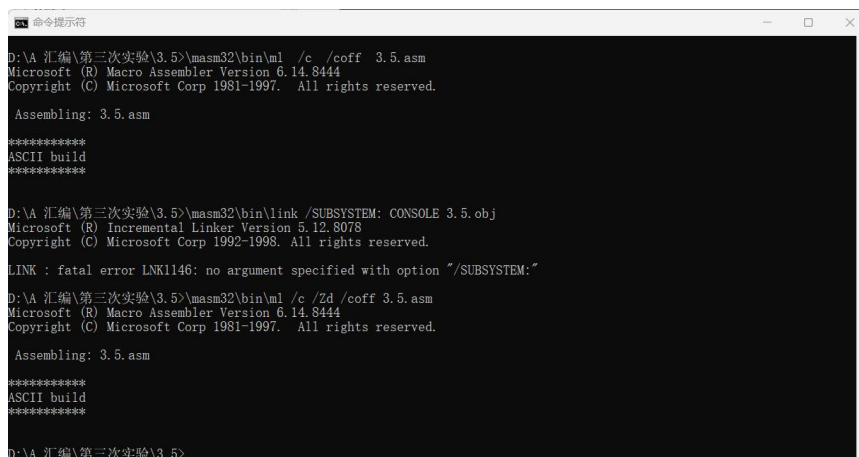
1. 源文件：用文本编辑器编写的 asm 文本文件



```
.386
.model flat,stdcall
option casemap:none
INCLUDE \masm32\include\windows.inc
INCLUDE \masm32\include\kernel32.inc
INCLUDE \masm32\include\masm32.inc
include\masm32\include\user32.inc
INCLUDELIB \masm32\lib\kernel32.lib
INCLUDELIB \masm32\lib\masm32.lib
includelib\masm32\lib\user32.lib
includelib \masm32\lib\msvcrt.lib
scanf PROTO C:DWORD, :vararg
.data
ArrayA dword 10 dup(?)
infmt byte '%d,%d,%d,%d,%d,%d,%d,%d,%d,%d',0
szText db 100 dup(0)
szCharsFormat db 'After Bubble Sort : ArrayA :%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%c'
.code
BubbleSort proc uses eax ecx esi,
    pArray:ptr dword,
    Count:dword
local @exchange:dword
    mov ecx,Count
    dec ecx
    mov @exchange,0
L1:
    push ecx
    mov esi,pArray
```

2. 使用 ml 将 3.5.asm 文件汇编到 3.5.obj 目标文件，编译命令：

“\masm32\bin\ml /c /Zd /coff 3.5.asm”



```
D:\A 汇编\第三次实验\3.5>\masm32\bin\ml /c /coff 3.5.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: 3.5.asm

*****
ASCII build
*****

D:\A 汇编\第三次实验\3.5>\masm32\bin\link /SUBSYSTEM: CONSOLE 3.5.obj
Microsoft (R) Incremental Linker Version 5.12.6078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

LINK : fatal error LNK1146: no argument specified with option "/SUBSYSTEM:"

D:\A 汇编\第三次实验\3.5>\masm32\bin\ml /c /Zd /coff 3.5.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: 3.5.asm

*****
ASCII build
*****

D:\A 汇编\第三次实验\3.5>
```

3. 使用 link 将目标文件 dec2hex.obj 链接成 dec2hex.exe 可执行文件，链接

命令：“\masm32\bin\Link /SUBSYSTEM:CONSOLE 3.5.obj”

```

D:\A 汇编\第三次实验\3.5>\masm32\bin\ml /c /Zd /coff 3.5.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: 3.5.asm

*****
ASCII build
*****

D:\A 汇编\第三次实验\3.5>\masm32\bin\Link /SUBSYSTEM:CONSOLE 3.5.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

D:\A 汇编\第三次实验\3.5>

```

 3.5.asm	2022/10/28 18:32	ASM 文件
 3.5.exe	2022/10/28 18:32	应用程序
 3.5.obj	2022/10/28 18:32	3D Object

4. 测试和截图

```

D:\A 汇编\第三次实验\3.5>3.5
3 45 87 32 65 23 12 76 90 34
After Bubble Sort : ArrayA :3, 12, 23, 32, 34, 45, 65, 76, 87, 90
D:\A 汇编\第三次实验\3.5>

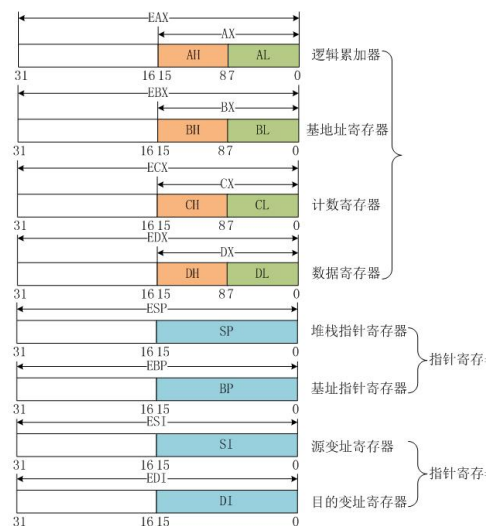
```

七、 汇编语言数组操作知识点的总结

1. 计算数组大小

dw_array DWORD 0, 1, 2, 3, 4
array_size = (\$ - dw_array)/4

2. 通用寄存器



EIP: 下一条指令地址

ESP: 栈顶指针


EBP: 栈底指针

3. 控制流转移指令

jmp : 无限循环


无符号数的条件跳转指令:

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JA/JNBE	Jump Above or Jump Not Below/Equal	CF, ZF
JAЕ/JNB	Jump Above/Equal or Jump Not Below	CF
JB/JNAE	Jump Below or Jump Not Above/Equal	CF
JBE/JNA	Jump Below/Equal or Jump Not Above	AF, CF



有符号数的条件跳转指令:

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JA/JNBE	Jump Above or Jump Not Below/Equal	CF, ZF
JAЕ/JNB	Jump Above/Equal or Jump Not Below	CF
JB/JNAE	Jump Below or Jump Not Above/Equal	CF
JBE/JNA	Jump Below/Equal or Jump Not Above	AF, CF



loop: 循环

4.

add esi, 4 偏移量, 找下一个元素

offset 获取数据标号的内存偏移地址

inc eax eax 寄存器的值加 1

dec (decrement) 指令从操作数中减 1

mov count, ebx (第一个操作数是目的操作数, 第二个操作数是源操作数) 源操作数的值赋给目的操作数

数据类型:

byte, db, 8 位

word, dw, 16 位

dword, dd, 32 位

qword , dq, 64 位

my_var DWORD 0, 1, 2, 3 数组

dup 伪指令, 为字符串或数组分配内存空间

BYTE 20 DUP (0) ; 20 个字节的内存空间

mov destination, source 两个操作数不能同时为内存操作数

mov eax, [var1] 使用方括号表示内存寻址操作

movzx r32, r/m8 复制尺寸较小的操作数到尺寸较大的操作数

movzx r32, r/m16

movzx r16, r/m8

movsx r32, r/m8 MOVSX (move with sign-extend) 符号扩展传送指令, 最高位循环填充所有扩展位

movsx r32, r/m16

movsx r16, r/m8

XCHG (exchange data) 指令交换两个操作数的内容

add 指令将同尺寸的源操作数和目的操作数相加, 相加的结果存储在目的操作数中

sub 指令将源操作数从目的操作数中减掉

neg (negate) 指令通过将数字转换为对应的补码而求得其相反数

align 指令将变量的位置按 BYTE、WORD、DWORD 边界对齐

PTR 操作符可以重载操作数声明的默认尺寸

TYPE 操作符返回变量的字节数

LENGTHOF 操作符计算数组中元素的数目, 元素由出现在同一行的值定义

sizeof 操作符的返回值等于 LENGTHOF 和 TYPE 返回值的乘积