**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：孙蕗 专业：信息安全**

1. **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
2. **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

1. **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。

1. **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。

1. **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

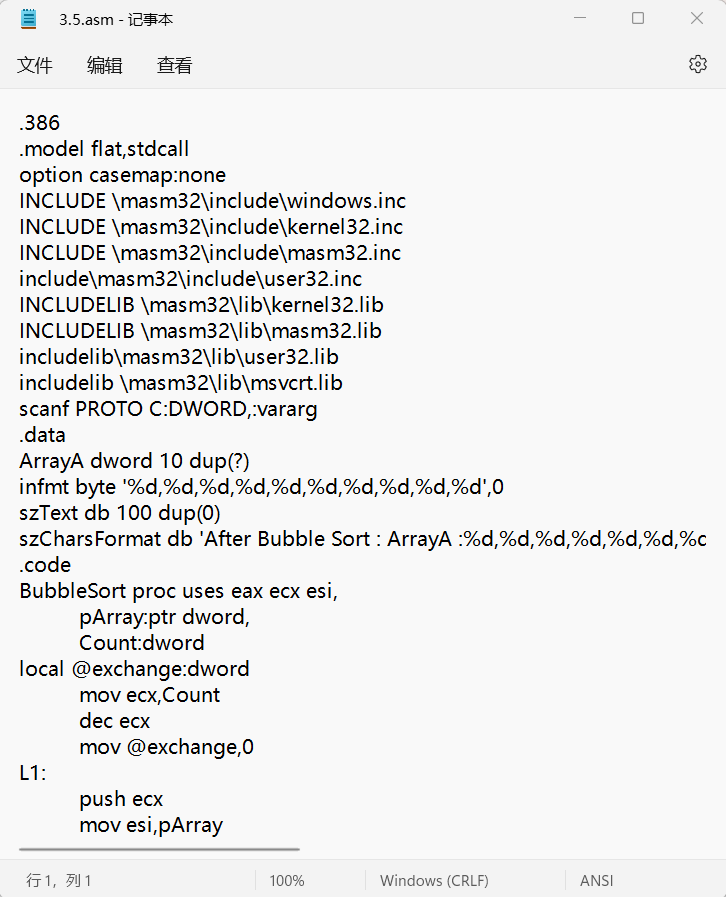
invoke wsprintf,addr szText,addr szCharsFormat,ArrayA,ArrayA+4,ArrayA+8,ArrayA+12,ArrayA+16,ArrayA+20,ArrayA+24,ArrayA+28,ArrayA+32,ArrayA+36,ArrayA+40

invoke StdOut,offset szText ;输出

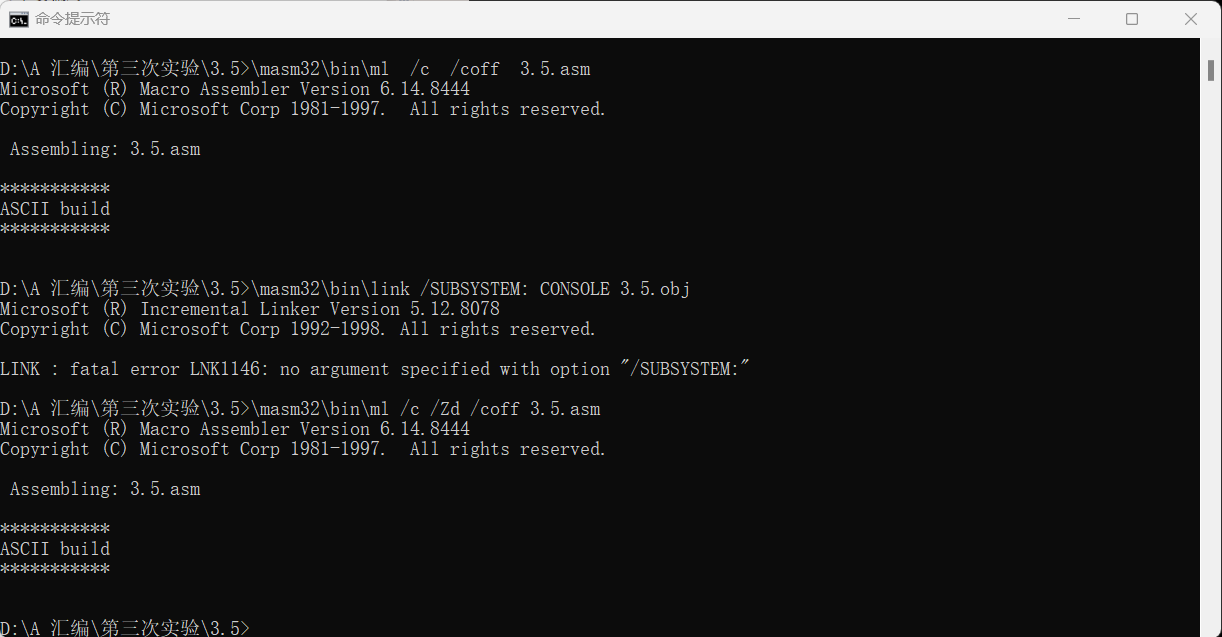
invoke ExitProcess,0

end start

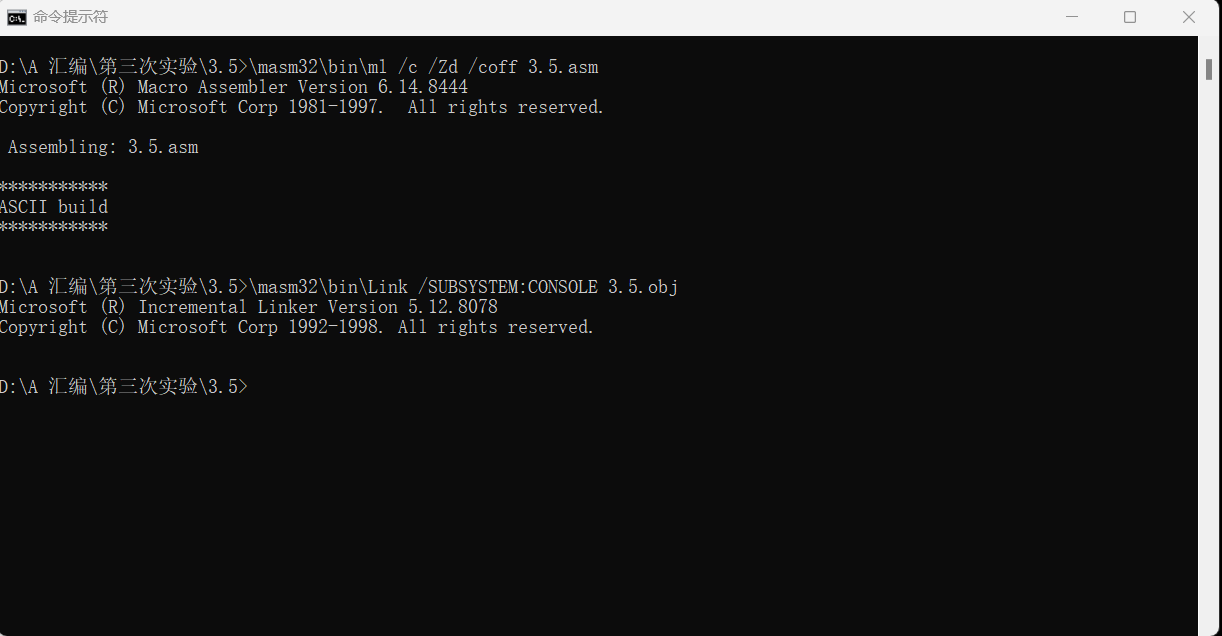
1. **实验步骤**
2. 源文件：用文本编辑器编写的asm文本文件

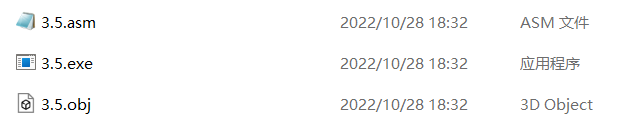
****

1. 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”

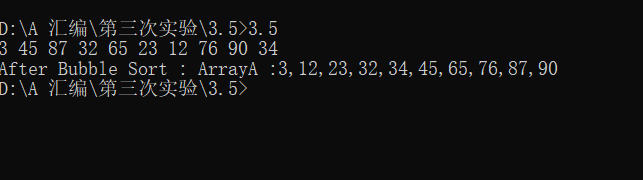


1. 使用link将目标文件dec2hex.obj链接成dec2hex.exe可执行文件，链接命令：“\masm32\bin\Link /SUBSYSTEM:CONSOLE 3.5.obj”



****

1. 测试和截图

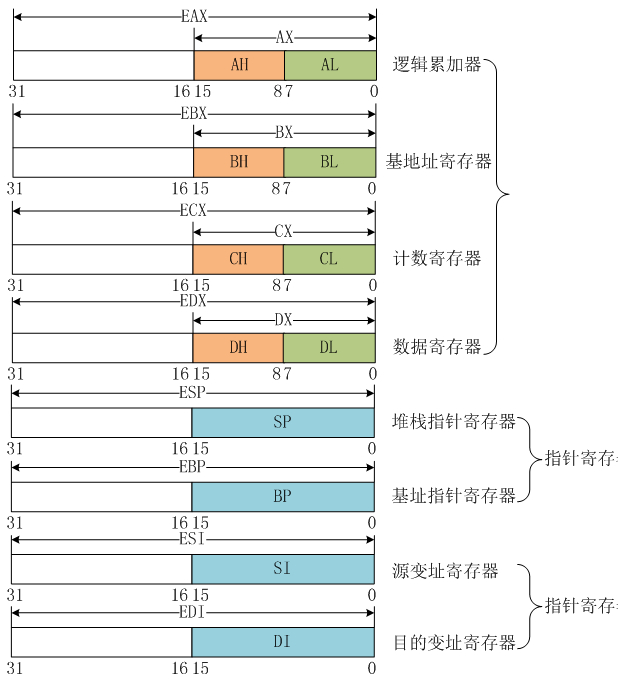
****

1. **汇编语言数组操作知识点的总结**
2. 计算数组大小

dw\_array DWORD 0, 1, 2, 3, 4

array\_size = ($ - dw\_array)/4

1. 通用寄存器



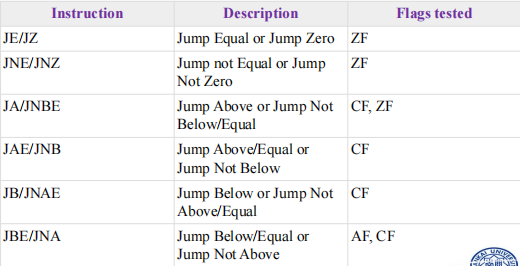
EIP:下一条指令地址

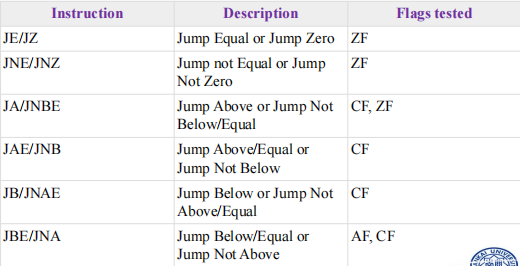
ESP:栈顶指针

EBP:栈底指针

1. 控制流转移指令

jmp ：无限循环

无符号数的条件跳转指令：

有符号数的条件跳转指令：

loop: 循环

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