

**汇编语言与逆向技术实验报告**

**实验三：bubble**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2212046

姓 名 王昱

班 级 信息安全班

**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

1. **实验环境**

Windows记事本

MASM32编译环境

Windows命令行窗口

**三、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。

**四、实验过程**

1. **实验思路**

本次实验的要求是将十个一万以内的数字用冒泡排序的方法进行排序并输出，整个程序可以分为三块，分别是输入，排序和输出。

在输入部分，由于输入的是十个一万以内的数字，也就是说每个数字的位数不会超过四位，我的想法是将十个数字输入放到一个字符串中，字符串的大小只要申请十个四位数的长度再加上空格的长度便足够。除此之外，由于之后还要进行排序，这就需要将输入的字符转换成整数，由于输入的数字是以空格为间隔的，于是便可以用输入本身自带的空格来对字符串进行切分，切分的每一块再单独进行ASCII码向十进制整数的转换。

在排序部分，冒泡排序的基本思想就是两重循环，于是需要用到两个寄存器来记录循环进行的次数，而在循环内部就是将相邻的元素进行比较，将较小的元素放在前面，较大的放在相对较后面的位置。

在输出部分，因为之前用到的StdOut函数不可以直接将整数进行输出，于是就需要将排序好的整数再转换成可以输出的字符串，这个过程要用到除十取余的方法，但由于这个方法首先得到的是低位的数字，与存储的顺序不同，于是便需要先将得到的数字进行缓存，最后将完整的一个数字的字符从缓存的区域再转存到输出的字符串中进行输出。

1. **源代码**

**.386**

**.model flat, stdcall**

**option casemap :none**

**include \masm32\include\windows.inc**

**include \masm32\include\kernel32.inc**

**include \masm32\include\masm32.inc**

**includelib \masm32\lib\kernel32.lib**

**includelib \masm32\lib\masm32.lib**

**.data**

**input db 50 DUP(0) ;用来存储输入的字符串，申请的内存足够十个四位整数的储存**

**array dd 10 DUP(0) ;用来存储整数的数组**

**temp db "0000",0 ;在整数转换成为字符串的时候作为缓存变量，整数位数最多为四位， 于是申请长度为4的字符串**

**output db 50 DUP(0);用来存储输出的字符串，与输入的字符串大小相同**

**const10 dd 10 ;表示常数10，用来进行字符和整数的转换**

**.code**

**str2int PROC;用来将输入的字符串转换成整数的函数**

**mov esi,0;esi寄存器用来遍历输入字符串**

**mov ebx,0;之后会用到ebx寄存器的低八位部分，所以要先清零，以免其他位置 上的值产生影响**

**mov ecx,0;ecx寄存器用来遍历整数数组**

**L1:**

**mov edx,0**

**mov bl,input[esi];按位取出输入字符串中的字符**

**sub ebx,'0';将字符从ASCII码转换成十进制数字**

**mov eax,array[ecx\*TYPE array];取出数组中对应的数字，进行十进制转换**

**mul const10**

**mov array[ecx\*TYPE array],eax**

**add array[ecx\*TYPE array],ebx ;将取出的数字乘十之后加上从输入字符串中得 到的数字**

**inc esi ;继续处理输入字符串的下一位字符**

**mov bl,input[esi];将下一位字符取出**

**cmp bl,' ' ;与空格进行比较，判断是否完成一个十进制整数的转换**

**jne none;不是空格的话跳转到none处**

**inc esi;偏移到字符串的下一位**

**inc ecx;存储整数的数组也偏移到下一位**

**none:**

**cmp bl,0;判断是否到达整个输入字符串的末端，不是的话跳回L1继续处理**

**jnz L1**

**ret**

**str2int ENDP**

**bubble PROC;进行整数数组的排序功能实现**

**mov ecx,0;用于外层循环的计数**

**mov edx,0;用于内层循环的计数**

**L2:**

**inc edx;每次外部循坏edx寄存器递增1**

**mov esi,9;遍历数组，将esi置为起始位置**

**L3:**

**mov eax,array[esi\*TYPE array]**

**cmp eax,array[esi\*TYPE array-TYPE array];比较数组当前位元素与前一位元素**

**jae above;如果大于等于则跳过，直接跳转到above**

**mov eax,array[esi\*TYPE array-TYPE array]**

**mov ebx,array[esi\*TYPE array]**

**mov array[esi\*TYPE array-TYPE array],ebx**

**mov array[esi\*TYPE array],eax;完成当前元素与前一元素的交换**

**above:**

**dec esi;将数组当前位置向前移动一位**

**cmp esi,edx;判断当前内部循环是否结束**

**jae L3;如果没有结束的话就返回L3继续遍历**

**inc ecx**

**cmp ecx,9;判断外部循环是否结束**

**jbe L2;如果没有结束的话就返回L2继续下一次循环**

**ret ;函数结束**

**bubble ENDP**

**trans PROC;将数组中的整数转换成字符串**

**mov ecx,0;ecx寄存器用来遍历整数数组**

**mov edi,0;edi寄存器用来遍历输出字符串**

**L4:**

**mov esi,3;因为整数转换成字符串要用到除十取余法，先得到的数字在字符的低位， 所以从最后一位开始存**

**mov eax,array[ecx\*TYPE array];取出整数数组的第一个元素放到eax中作为被除 数**

**L5:**

**mov edx,0;edx中存储除法的余数，因此先将其置零**

**div const10;除十**

**add edx,'0' ;由于是字符，要转换成ASCII的十进制字符**

**mov temp[esi],dl将转换成十进制字符的值存入缓存区**

**dec esi ;缓存区向前移位**

**cmp eax,0 ;判断转换是否结束**

**jnz L5;如果没有结束的话跳回L5继续转换**

**mov esi,0;已经结束的话需要将esi置为零，便于之后将缓存区的字符串存储到输 出字符串中**

**L6:**

**mov eax,0;将eax寄存器清零**

**mov al,temp[esi];将缓存区的第一个字符取出**

**cmp al,'0';判断字符是否为'0'，进而判断是否需要存储到输出字符串中**

**jbe needless;不需要的话直接跳过need过程**

**need:;将需要存储的字符存储到输出字符串中**

**mov al,temp[esi]**

**mov output[edi],al;将之前取出的字符存入输出字符串中**

**inc edi**

**inc esi;输出字符串和缓存区的当前位置都向后移一位**

**mov al,temp[esi];再取出缓存区的下一个字符**

**cmp al,0;判断缓存区是否取完**

**jne need;未取完的话返回need处继续取出放入输出字符串**

**je L7;取完的话就跳转到L7部分**

**needless:**

**inc esi;将缓存区的位置向后移位**

**jmp L6;跳回到L6继续处理缓存区的下一个字符**

**L7:**

**mov output[edi],' ';在这个位置代表一个整数的字符存储完成，向输出字符串中添 加空格以便于最后输出**

**inc edi**

**inc ecx;将整数数组中的位置和输出字符串的位置都向后移一位**

**cmp ecx,9;判断是否全部转换完成**

**jbe L4;未完成的话就跳回L4继续转换下一个字符**

**ret**

**trans ENDP**

**.code**

**start:**

**invoke StdIn, addr input, 50**

**call str2int**

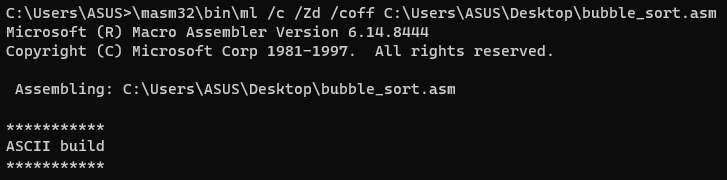
**call bubble**

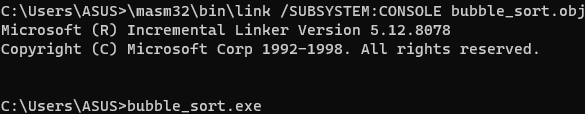
**call trans;调用编写的三个函数来实现转换和排序**

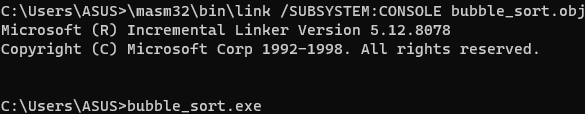
**invoke StdOut, addr output**

**END start**

1. **测试结果**

****

****



1. **心得感悟**

汇编语言中数组部分的知识点：

首先是数组的定义，在data段对数组进行定义时可以直接给出整数数组的值，比如array DWORD 100, 101, 102, 103，也可以先用DUP伪指令为数组赋初值，比如array BYTE 20 DUP（0），字符串也可以当作字符的数组，在给字符串进行定义的时候要注意如果不是用DUP伪指令则需要在结尾加0。

在数组的操作上，大部分是对数组的元素进行存取或者遍历数组等操作，对于存取的话，要注意需要将数组的元素取出到一个寄存器中，因为mov指令不允许内存之间进行操作，除此之外，要注意数组元素类型的大小和要放入的寄存器要匹配，如果不匹配可以用寄存器的一部分，如低位部分等等，但如果这样操作要注意将用到的寄存器整体先置为零，以免有其他数据造成混乱。而对数组的遍历操作需要注意格式，比如output[edi],也可以表示成[output+edi]，而且用寄存器来遍历数组的时候要注意数组中的元素类型，因为在这里寄存器充当的是地址的偏移量，这与数组中存储的每一个元素所占用的内存相关，比较保险的方法是用TYPE直接得到数组存储元素的类型，比如array[ecx\*TYPE array]，这样子可以比较迅速的得到数组元素的大小，进而对数组进行遍历。

除了对于数组的操作的熟悉，在本次实验中我对汇编语言中一些其他操作也加深了了解，比如循环的使用，在本次实验中我是用寄存器来进行计数进而控制循环的，在我对循环的控制中，明白了在条件判断和条件跳转的过程中控制权是怎样进行转移的，于是在我编写代码的过程中开始尝试用跳转使代码变得更加简洁明了。不仅如此，在本次实验的过程中我也有一次熟悉了字符和数字之间的转换过程，二者之间的双向转换过程也更加熟练。

当然，在我的代码中也有许多部分可以改进，比如可以用loop来实现循环而不是一味的使用寄存器计数来控制循环，这些地方我在之后编写程序的过程中会进行改进。